Smarter. Greener. Together.

Industrial Automation Headquarters

Delta Electronics, Inc.
Taoyuan Technology Center
No.18, Xinglong Rd., Taoyuan City,
Taoyuan County 33068, Taiwan
TEL: 886-3-362-6301 / FAX: 886-3-371-6301

Asia

Delta Electronics (Jiangsu) Ltd.
Wujiang Plant 3
1688 Jiangxing East Road,
Wujiang Economic Development Zone
Wujiang City, Jiang Su Province, P.R.C. 215200
TEL: 86-512-6340-3008 / FAX: 86-769-6340-7290

Delta Greentech (China) Co., Ltd.
238 Min-Xia Road, Pudong District,
ShangHai, P.R.C. 201209
TEL: 86-21-58635678 / FAX: 86-21-58630003

Delta Electronics (Japan), Inc.
Tokyo Office
2-1-14 Minato-ku Shibadaimon,
Tokyo 105-0012, Japan
TEL: 81-3-5733-1111 / FAX: 81-3-5733-1211

Delta Electronics (Korea), Inc.
1511, Byucksan Digital Valley 6-cha, Gasan-dong,
Geumcheon-gu, Seoul, Korea, 153-704
TEL: 82-2-515-5303 / FAX: 82-2-515-5302

Delta Electronics Int'l (S) Pte Ltd.
4 Kaki Bukit Ave 1, #05-05, Singapore 417939
TEL: 65-6747-5155 / FAX: 65-6744-9228

Delta Electronics (India) Pvt. Ltd.
Plot No 43 Sector 35, HSIIDC
Gurgaon, PIN 122001, Haryana, India
TEL : 91-124-4874900 / FAX : 91-124-4874945

Americas

Delta Products Corporation (USA)
Raleigh Office
P.O. Box 12173,5101 Davis Drive,
Research Triangle Park, NC 27709, U.S.A.
TEL: 1-919-767-3800 / FAX: 1-919-767-8080

Delta Greentech (Brasil) S.A.
Sao Paulo Office
Rua Itapeva, 26 - 3° andar Edificio Itapeva One-Bela Vista
01332-000-São Paulo-SP-Brazil
TEL: 55 11 3568-3855 / FAX: 55 11 3568-3865

Europe

Delta Electronics (Netherlands) B.V.
Eindhoven Office
De Witbogt 20, 5652 AG Eindhoven, The Netherlands
TEL : +31 (0)40-8003800 / FAX : +31 (0)40-8003898

DVP-0191920-00

2016-06-30

DVP15MC Operation Manual

DVP15MC Operation Manual

www.deltaww.com

Smarter. Greener. Together.

DVP15MC11T Operation Manual

Table of Contents

Chapter 1 Preface 1-1

1.1 Explanation of Symbols in This Manual 1-2

1.2 Revision History 1-2

Chapter 2 Overview of DVP15MC11T 2-1

2.1 Product Description 2-2

2.2 Functions 2-2

2.3 Profile and Components 2-3

Chapter 3 Specifications 3-1

3.1 Function Specifications 3-2

3.1.1 Specifications.. 3-2
3.1.2 Devices and Data Types 3-3
3.1.2.1 Devices.. 3-3
3.1.2.2 Valid Ranges of Devices 3-4
3.1.2.3 Latched Devices 3-5
3.1.2.4 Data Types and Valid Ranges Supported 3-6

3.2 Electrical Specifications 3-7

Chapter 4 System Architecture 4-1

4.1 System Constitution 4-2

4.2 Power Supply 4-2

4.3 Left-side Extension 4-2

4.3.1 Connectable Left-side Extension Module.. 4-2
4.3.2 Allocation of Left-side Network Module Addresses..................................... 4-3
4.3.3 Method of Reading/Writing of Left-side Modules 4-3

4.4 Right-side Extension 4-3

4.4.1 Connectable Right-side Extension Modules 4-3
4.4.2 Allocation of Right-side Extension Module Addresses................................. 4-4

4.5 Connectable Servo Drives.. 4-4

4.6 SD Memory Card... 4-6

4.6.1 Model and Specification 4-6
4.6.2 Function.. 4-8

Chapter 5 Installation.. 5-1

5.1 Dimensions 5-2
5.1.1 Profile and Dimensions of DVP15MC11T 5-2
5.1.2 Dimensions of Left-side and Right-side Extension Modules 5-2
5.1.3 Connecting to the Left-side Extension Module 5-3
5.1.4 Connecting to the Right-side Extension Module 5-4
5.1.5 SD Card Installing and Removing 5-5

5.2 Installing the Module in the Control Cabinet 5-7
5.2.1 Installing the Module to DIN rail 5-7
5.2.2 Illustration of Installation Inside the Control Cabinet 5-7
5.2.3 Environmental Temperature in the Control Cabinet 5-7
5.2.4 Actions for Anti-interference 5-8
5.2.5 Dimension Requirement in the Control Cabinet 5-8

Chapter 6 Wiring, Communication Setting and Network Construction 6-1

6.1 Wiring 6-3
6.1.1 Power Supply 6-3
6.1.2 Safety Circuit Wiring 6-3

6.2 Input Point and Output Point Wiring 6-4
6.2.1 Function that Input Points Support 6-4
6.2.2 Input Point Wiring 6-5
6.2.3 Output Point Wiring 6-6

6.3 RS-485 Communication Port 6-8
6.3.1 Function that RS-485 Port Supports 6-8
6.3.2 Definitions of RS-485 Port Pins 6-8
6.3.3 RS-485 Hardware Connection 6-8
6.3.4 Supported Function Codes and Exception Codes 6-10

6.4 RS-232 Communication Port 6-11
6.4.1 Function that RS-232 Port Supports 6-11
6.4.2 Definitions of RS-232 Port Pins 6-11
6.4.3 RS-232 Hardware Connection 6-11
6.4.4 Supported Function Codes and Exception Codes 6-12

6.5 SSI Absolute Encoder Port 6-13
6.5.1 Function of SSI Absolute Encoder 6-13
6.5.2 Definitions of SSI Port Pins 6-13
6.5.3 SSI Absolute Encoder Hardware Connection 6-13

6.6 Incremental Encoders 6-15
6.6.1 Function of Incremental Encoder 6-15
6.6.2 Definition of Incremental Encoder Port Pins 6-15
6.6.3 Incremental Encoder Hardware Connection 6-16

6.7 Ethernet Communication Port 6-17
6.7.1 Function that Ethernet Communication Port Supports 6-17
6.7.2 Pins of Ethernet Communication Port 6-17
6.7.3 Network Connection of Ethernet Communication Port 6-17
6.7.4 Function Codes that Ethernet Communication Port Supports 6-18

6.8 Motion Communication Port 6-18
6.8.1 Function that Motion Communication Port Supports 6-18

6.8.2 Pins of Motion Communication Port 6-18
6.8.3 Motion Network Connection 6-19
6.8.4 Communication Speed and Communication Distance 6-19

6.9 CANopen Communication Port 6-20
6.9.1 Functions that CANopen Communication Port Supports 6-20
6.9.2 Pins of CANopen Communication Port 6-21
6.9.3 PDO Mapping at CANopen Communication Port 6-21
6.9.4 Network Connection at CANopen Communication Port 6-21
6.9.5 CANopen Communication Rate and Communication Distance 6-22

Chapter 7 Execution Principle of DVP15MC11T Controller 7-1

7.1 Tasks 7-2
7.1.1 Task Types 7-2
7.1.2 Priority levels of Tasks 7-4
7.1.3 Watchdog for a Task 7-6
7.1.4 Motion Instructions for Each Task Type 7-7

7.2 The Impact of PLC RUN or STOP on Variables and Devices 7-8

7.3 Relationship between Motion Program and Motion Bus 7-8

7.4 Synchronization Cycle Period Setting 7-9

Chapter 8 Logic Instructions 8-1

8.1 Table of Logic Instructions 8-4

8.2 Explanation of Logic Instructions 8-7
8.2.1 EN and ENO 8-7

8.3 Sequence Input /Output Instructions 8-7
8.3.1 R_TRIG 8-7
8.3.2 F_TRIG 8-9
8.3.3 RS 8-11
8.3.4 SR 8-13
8.3.5 SEMA 8-15

8.4 Data Movement Instructions 8-17
8.4.1 MOVE 8-17
8.4.2 MoveBit 8-18
8.4.3 TransBit 8-20
8.4.4 MoveDigit 8-22
8.4.5 Exchange 8-24
8.4.6 Swap 8-26

8.5 Comparison Instructions 8-28
8.5.1 LT 8-28
8.5.2 LE 8-30
8.5.3 GT 8-32
8.5.4 GE 8-34
8.5.5 EQ 8-36
8.5.6 NE 8-38

8.6 Timer Instructions 8-40

8.6.1 TON 8-40
8.6.2 TOF 8-42
8.6.3 TP 8-44

8.7 Counter Instructions 8-46
8.7.1 CTU 8-46
8.7.2 CTD 8-48
8.7.3 CTUD 8-50

8.8 Math Instructions 8-53
8.8.1 ADD 8-53
8.8.2 SUB 8-56
8.8.3 MUL 8-59
8.8.4 DIV 8-62
8.8.5 MOD 8-65
8.8.6 MODREAL 8-67
8.8.7 MODTURNS 8-69
8.8.8 MODABS 8-71
8.8.9 ABS 8-73
8.8.10 DegToRad 8-75
8.8.11 RadToDeg 8-77
8.8.12 SIN 8-79
8.8.13 COS 8-81
8.8.14 TAN 8-83
8.8.15 ASIN 8-85
8.8.16 ACOS 8-88
8.8.17 ATAN 8-90
8.8.18 LN 8-92
8.8.19 LOG 8-94
8.8.20 SQRT 8-96
8.8.21 EXP 8-98
8.8.22 EXPT 8-100
8.8.23 RAND 8-102
8.8.24 TRUNC 8-104
8.8.25 FLOOR 8-106
8.8.26 FRACTION 8-108

8.9 Bit String Instructions 8-110
8.9.1 AND 8-110
8.9.2 OR 8-113
8.9.3 NOT 8-116
8.9.4 XOR 8-118
8.9.5 XORN 8-121

8.10 Shift Instructions 8-124
8.10.1 SHL 8-124
8.10.2 SHR 8-126
8.10.3 ROL 8-128
8.10.4 ROR 8-130

8.11 Selection Instructions 8-132
8.11.1 MAX 8-132
8.11.2 MIN 8-134

8.11.3 SEL 8-136
8.11.4 MUX 8-138
8.11.5 LIMIT 8-140
8.11.6 BAND 8-143
8.11.7 ZONE 8-146

8.12 Data Type Conversion Instructions 8-149
8.12.1 BOOL_TO_*** 8-149
8.12.2 Bit strings_TO_*** 8-152
8.12.3 Integers_TO_*** 8-159
8.12.4 Real numbers_TO_*** 8-168
8.12.5 Times,dates_TO_*** 8-171
8.12.6 Strings_TO_*** 8-173

8.13 CANopen Communication Instructions 8-176
8.13.1 DMC_ReadParameter_CANopen 8-176
8.13.2 DMC_WriteParameter_CANopen 8-182

8.14 String Processing Instructions 8-187
8.14.1 CONCAT 8-187
8.14.2 DELETE 8-189
8.14.3 INSERT 8-191
8.14.4 LEFT / RIGHT 8-193
8.14.5 MID 8-195
8.14.6 REPLACE 8-197
8.14.7 LEN 8-199
8.14.8 FIND 8-200

8.15 Immediate Refresh Instructions 8-202
8.15.1 FROM 8-202
8.15.2 TO 8-206
8.15.3 ImmediateInput 8-210
8.15.4 ImmediateOutput 8-212

Chapter 9 Introductions of Axis Parameters 9-1

9.1 Description of Axis Parameters 9-2

Chapter 10 Motion Control Function 10-1

10.1 EN and ENO 10-2

10.2 Relation among Velocity, Acceleration and Jerk 10-3

10.3 Introduction of BufferMode 10-6

10.4 The State Machine 10-32

Chapter 11 Motion Control Instructions 11-1

11.1 Table of Motion Control Instructions 11-3

11.2 About Motion Control Instructions 11-4
11.2.1 Composition of A Motion Control Instruction 11-4
11.2.2 Program Languages that Motion Control Instructions Support 11-4
11.2.3 Configuration of Motion Control Instructions 11-4

11.3 Single-axis Instructions 11-5
11.3.1 MC_Power 11-5
11.3.2 MC_Home 11-14
11.3.3 MC_MoveVelocity 11-19
11.3.4 MC_Halt 11-26
11.3.5 MC_Stop 11-31
11.3.6 MC_MoveRelative 11-36
11.3.7 MC_MoveAdditive 11-44
11.3.8 MC_MoveAbsolute 11-52
11.3.9 MC_MoveSuperimposed 11-61
11.3.10 MC_HaltSuperimposed 11-68
11.3.11 MC_SetPosition 11-73
11.3.12 MC_SetOverride 11-83
11.3.13 MC_Reset 11-87
11.3.14 DMC_SetTorque 11-90
11.3.15 MC_ReadAxisError 11-94
11.3.16 MC_ReadActualPosition 11-96
11.3.17 MC_ReadStatus 11-101
11.3.18 MC_ReadMotionState 11-106
11.3.19 DMC_ReadParameter_Motion 11-111
11.3.20 DMC_WriteParameter_Motion 11-115
11.3.21 DMC_TouchProbe 11-119

11.4 Multi-axis Instructions 11-128
11.4.1 MC_GearIn 11-128
11.4.2 MC_GearOut 11-134
11.4.3 MC_CombineAxes 11-139
11.4.4 Introduction of Electronic Cam 11-147
11.4.5 MC_CamIn 11-148
11.4.6 MC_CamOut 11-169

11.5 Application Instructions 11-175
11.5.1 Rotary Cut Technology 11-175
11.5.2 Rotary Cut Parameters 11-176
11.5.3 Control Feature of Rotary Cut Function 11-177
11.5.4 Introduction to Cam Curve with Rotary Cut Function 11-178
11.5.5 Rotary-cut Instructions 11-182
11.5.5.1 APF_RotaryCut_Init 11-182
11.5.5.2 APF_RotaryCut_In 11-185
11.5.5.3 APF_RotaryCut_Out 11-187
11.5.6 Application Example of Rotary Cut Instructions 11-189

Chapter 12 Troubleshooting 12-1

12.1 Explanation of LED Indicators 12-2

12.2 Table of Error IDs in Motion Instructions 12-8

12.3 System Trouble Diagnosis through System Error Codes 12-12

Appendix A Modbus Communication A-1

A.1 Message Format in ASCII Mode A-2

A.2 Message Format in RTU Mode........A-5
A.3 Modbus Function Codes Supported........A-7
A.4 Modbus Exception Response Code Supported A-7
A.5 Introduction to Modbus Function Codes........A-8
A.6 Table of Registers and Corresponding Modbus addresses A-15
Appendix B Modbus TCP Communication........B-1
B.1 Modbus TCP Message Structure........B-2
B.2 Modbus Function Codes Supported in Modbus TCP........B-2
B.3 Exception Response Code in Modbus TCP........B-3
B.4 Modbus Function Codes in Modbus TCP B-3
B.5 Registers in DVP15MC11T and Corresponding Modbus Addresses........B-12
Appendix C CANopen Protocol........C-1
C.1 Node States........C-4
C.2 Network Management (NMT) C-7
C.3 PDO (Process Data Object) C-7
C.4 SDO (Service Data Object) C-9
Appendix D Explanation of Homing Modes D-1
D.1 Explanation of Homing Modes D-2
Appendix E List of Accessories........E-1
E.1 Accessories for CANopen Communication E-2
E.2 Accessories for PROFIBUS DP Communication E-4
E.3 Accessories for DeviceNet Communication........E-4

Memo

1

Chapter 1 Preface

Table of Contents

1.1 Explanation of Symbols in This Manual 1-2

1.2 Revision History 1-2

Thank you for purchasing DVP15MC11T motion controller which is created on the basis of motion control and we are providing you with a high-end motion control system.

This manual describes the product specifications, functions, system architecture, installation, wiring, execution principle, logic instructions and motion control instructions, trouble-shooting, communication protocols, homing modes and other relevant information.

Make sure that you have well known about the motion control system configuration and product operation before using DVP15MC11T.

1.1 Explanation of Symbols in This Manual

- **Precautions before operation**

Before operation, please read relevant safety instructions carefully so as to prevent an injury to personnel and damage to products.

⚠ Danger	It indicates the highly potential hazards. It is possible to cause a severe injury or even fatal harm to personnel if you do not follow the instructions.
⚠ Warning	It indicates the potential hazards. It is possible to cause a minor injury or even fatal harm to personnel if you do not follow the instructions.
⚠ Caution	It indicates much attention should be paid. An unexpected result may occur if you do not follow the instructions.

1.2 Revision History

Version	Revision	Release Date
1st	The first version was published.	May 30, 2018

2

Chapter 2 Overview of DVP15MC11T

Table of Contents

2.1 Product Description 2-2

2.2 Functions 2-2

2.3 Profile and Components 2-3

2.1 Product Description

DVP15MC11T is a type of multi-axis motion controller researched and produced by Delta autonomously on the basis of CANopen field bus. It complies with CANopen DS301 basic communication protocol and DSP402 motion control protocol. In addition, it also supports standard instruction libraries defined by international organizations for motion control. It brings great convenience to user to learn and develop projects quickly. Maximum 24 axes can be controlled by means of Motion port. The single-axis motion instructions including velocity, position, torque and homing instructions as well as multi-axis instructions such as electronic gear, electronic cam, rotary cut and G code are supported.

Multiple communication ports are built in DVP15MC11T. And thus various communication functions can be realized without adding modules. DVP15MC11T has left-side and right-side extension ports for adding DVP-S series modules to its left and right sides. (The left-side port is a high-speed parallel extension port.)

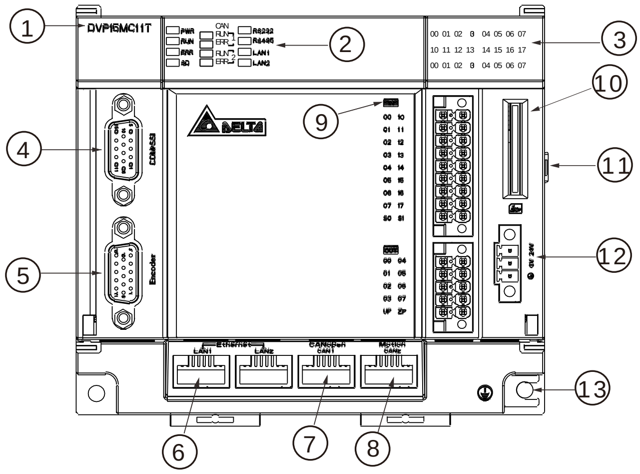
The communication system adopts highly reliable CAN bus as the main line and hence users just need simple cables for wiring.

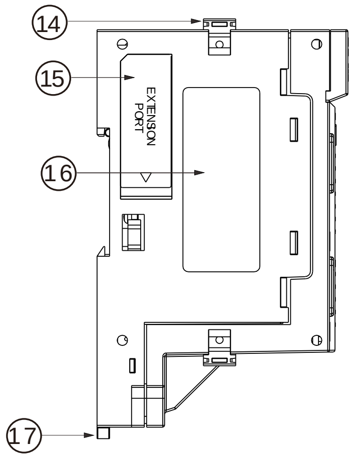
Thanks to the high-speed reliable motion control system, DVP15MC11T can be widely applied to a variety of automation control industries such as packaging, printing, encapsulating, wire cutting, drug manufacturing and so on.

2.2 Functions

- Able to control up to 24 real axes (with axis No. ranging from 1 to 32).
- The virtual axis and encoder axis can be built inside DVP15MC11T (with the axis No. ranging from 1 to 32, which can not be the same as that of real axes).
- Equipped with 1GHz high-speed floating-point operation processor; supporting 64-bit floating point (Lreal) and capable to meet various complicated motion control.
- With two built-in incremental encoder ports and one SSI absolute encoder port.
- With one RS232 port, one RS485 port and two Ethernet ports.
- With one built-in CAN port serving as CANopen master or slave.
- Supports powerful field network (as Ethernet master or slave, CANopen master or slave and Profibus-DP slave) for construction of a function-complicated control system.
- With a variety of I/O extensions (Left-side high-speed AIAO; right-side low-speed AIAO and DIDO and temperature modules).
- Using the easy-to-use software interface with the features of complete function and convenient application.
- Providing standard bus cables, terminal resistors, distributor boxes and other accessories as well as easy and convenient plug-and-play wiring.

2.3 Profile and Components





1	Model name	10	SD card slot
2	State indicators	11	Right-side extension module port
3	IO indicators	12	24V power port
4	COM/SSI communication port	13	Screw fixing clip
5	Incremental encoder port	14	Extension module fixing clip
6	Ethernet communication port	15	Left-side extension module port
7	CANopen communication port	16	Nameplate
8	CANmotion communication port	17	DIN rail fixing clip
9	Input and output pins and symbols		

MEMO

3

Chapter 3 Specifications

Table of Contents

3.1 Function Specifications 3-2
3.1.1 Specifications.. 3-2
3.1.2 Devices and Data Types 3-3
3.1.2.1 Devices.. 3-3
3.1.2.2 Valid Ranges of Devices 3-4
3.1.2.3 Latched Devices.. 3-5
3.1.2.4 Data Types and Valid Ranges Supported.. 3-6

3.2 Electrical Specifications 3-7

3.1 Function Specifications

3.1.1 Specifications

Item				Specification
Programming	Program capacity	Size		20M
		Quantity	Number of POU definitions	1024
	Memory capacity for variables	Retained	Size	128K
		Non-retained	Size	20M
	G code	One single G code program	Size	256K
		G code programs	Quantity	64
Motion control	Number of controlled axes	Max. number of axes for single-axis control		Real axis: 1~24, Virtual axis: 1~32. The virtual axis number is different from the real axis number.
		Max. number of axes for linear interpolation		8
		Max. number of axes for circular interpolation		3
	Number of cams	Size	Quantity	64
	Cam key points	Key points of one single cam	Quantity	2048
Built-in ports of DVP15MC11T		CAN	2	One CAN port supports the standard CANopen protocol and the other CAN port is used in Motion.
		Ethernet	2	Two independent Ethernet ports
		RS-232	1	Used as a master or slave
		RS485	1	Used as a master or slave
		Incremental encoder	2	Builds an encoder axis. Z signal can trigger an interrupt program.
		SSI absolute encoder	1	Builds an encoder axis
		Input points	Quantity	16 points (External interrupt trigger is supported.)
		Output points	Quantity	8 points
		Left-side extension port	1	Slim-series left-side extension module
		Right-side extension port	1	Slim series special module
Left-side and right-side extension	Left-side extension	Left-side extension modules	Quantity	8 pieces of Slim series left-side extension modules
	Right-side extension	Special modules	Quantity	8 pieces of Slim series special modules
		Digital modules	Number of points	240 input points and 240 output points

3.1.2 Devices and Data Types

3.1.2.1 Devices

- **Device Name Explanation**

```
% I X 0.0
│ │ │  └──── Digital point number
│ │ └─────── Prefix 2 symbol
│ └───────── Prefix 1 symbol
└─────────── A fixed character
```

- **Relevant Devices of DVP15MC11T Used in the Software**

No.	Item	Content				
1	Prefix 1 symbol	I	Q	M		
2	Prefix 1 name	Input device	Output device	Intermediate device		
3	Prefix 2 symbol	X	B	W	D	L
4	Data type of prefix 2	BIT	BYTE	WORD	DWORD	QWORD
5	Device example	%IX0.0	%IB0	%IW0	%ID0	%IL0
6		%QX0.0	%QB0	%QW0	%QD0	%QL0
7		%MX0.0	%MB0	%MW0	%MD0	%ML0

- **The Corresponding Relationships of Devices**

%ML0 includes %MB0~%MB7, %MD0 includes %MB0~%MB3 and %MW0 includes %MB0~%MB1 as shown in the following table.

<table>
<tr><th rowspan="3">Device name</th><th colspan="24">Corresponding relationships</th></tr>
<tr><th colspan="6">The 1st WORD</th><th colspan="6">The 2nd WORD</th><th colspan="6">The 3rd WORD</th><th colspan="6">The 4th WORD</th></tr>
<tr><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th></tr>
<tr><td>%MX</td><td colspan="3">%MX0.0~0.7</td><td colspan="3">%MX1.0~1.7</td><td colspan="3">%MX2.0~2.7</td><td colspan="3">%MX3.0~3.7</td><td colspan="3">%MX4.0~4.7</td><td colspan="3">%MX5.0~5.7</td><td colspan="3">%MX6.0~6.7</td><td colspan="3">%MX7.0~7.7</td></tr>
<tr><td>%MB</td><td colspan="3">%MB0</td><td colspan="3">%MB1</td><td colspan="3">%MB2</td><td colspan="3">%MB3</td><td colspan="3">%MB4</td><td colspan="3">%MB5</td><td colspan="3">%MB6</td><td colspan="3">%MB7</td></tr>
<tr><td>%MW</td><td colspan="6">%MW0</td><td colspan="6">%MW1</td><td colspan="6">%MW2</td><td colspan="6">%MW3</td></tr>
<tr><td>%MD</td><td colspan="12">%MD0</td><td colspan="12">%MD1</td></tr>
<tr><td>%ML</td><td colspan="24">%ML0</td></tr>
</table>

%ML1 includes %MB8~%MB15, %MD2 includes %MB8~%MB11, %MW4 includes %MB8~%MB9 and %MB8 includes %MX8.0~8.7 as shown in the following table.

<table>
<tr><th rowspan="3">Device name</th><th colspan="24">Corresponding relationships</th></tr>
<tr><th colspan="6">The 5th WORD</th><th colspan="6">The 6th WORD</th><th colspan="6">The 7th WORD</th><th colspan="6">The 8th WORD</th></tr>
<tr><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th><th>Bit 0</th><th>…</th><th>Bit 7</th><th>Bit 8</th><th>…</th><th>Bit 15</th></tr>
<tr><td>%MX</td><td colspan="3">%MX8.0~8.7</td><td colspan="3">%MX9.0~9.7</td><td colspan="3">%MX10.0~10.7</td><td colspan="3">%MX11.0~11.7</td><td colspan="3">%MX12.0~12.7</td><td colspan="3">%MX13.0~13.7</td><td colspan="3">%MX14.0~14.7</td><td colspan="3">%MX15.0~15.7</td></tr>
<tr><td>%MB</td><td colspan="3">%MB8</td><td colspan="3">%MB9</td><td colspan="3">%MB10</td><td colspan="3">%MB11</td><td colspan="3">%MB12</td><td colspan="3">%MB13</td><td colspan="3">%MB14</td><td colspan="3">%MB15</td></tr>
<tr><td>%MW</td><td colspan="6">%MW4</td><td colspan="6">%MW5</td><td colspan="6">%MW6</td><td colspan="6">%MW7</td></tr>
<tr><td>%MD</td><td colspan="12">%MD2</td><td colspan="12">%MD3</td></tr>
</table>

Device name	Corresponding relationships																							
	The 5th WORD						The 6th WORD						The 7th WORD						The 8th WORD					
	Bit 0	…	Bit 7	Bit 8	…	Bit 15	Bit 0	…	Bit 7	Bit 8	…	Bit 15	Bit 0	…	Bit 7	Bit 8	…	Bit 15	Bit 0	…	Bit 7	Bit 8	…	Bit 15
%ML	%ML1																							

3.1.2.2 Valid Ranges of Devices

- **The table of valid ranges of the devices in DVP15MC11T**

Device name	Expression	Range
%IX	%IX0.0~%IX0.7	%IX0.0~%IX127.7
%QX	%QX0.0~%QX0.7	%QX0.0~%QX127.7
%MX	%MX0.0	%MX0.0~%MX131071.7
%IB	%IB0	%IB0~%IB127
%QB	%QB0	%QB0~%QB127
%MB	%MB0	%MB0~%MB131071
%IW	%IW0	%IW0~%IW63
%QW	%QW0	%QW0~%QW63
%MW	%MW0	%MW0~%MW65535
%ID	%ID0	%ID0~%ID31
%QD	%QD0	%QD0~%QD31
%MD	%MD0	%MD0~%MD32767
%IL	%IL0	%IL0~%IL15
%QL	%QL0	%QL0~%QL15
%ML	%ML0	%ML0~%ML16383

- **The table of Modbus device addresses**

Device area	Device type	Range	Modbus address	Modbus address type
I (Input)	Bit	%IX0.0~%IX0.7	0x6000~0x6007	Standard Modbus address
		%IX1.0~%IX1.7	0x6008~0x600F	
		……	……	
		%IX127.0~%IX127.7	0x63F8~0x63FF	
	Word	%IW0~%IW63	0x8000~0x803F	
Q (Output)	Bit	%QX0.0~%QX0.7	0xA000~0xA007	
		%QX1.0~%QX1.7	0xA008~0xA00F	
		……	……	
		%QX127.0~%QX127.7	0xA3F8~0xA3FF	
	Word	%QW0~%QW63	0xA000~0xA03F	
M (Register)	Bit	%MX0.0~%MX0.7	0x10000000~0x10000007	Delta-extended Modbus addresses
		%MX1.0~%MX1.7	0x10000008~0x1000000F	
		……	……	
		%MX131071.0~%MX131071.7	0x100FFFF8~0x100FFFFF	
	Word	%MW0~%MW32767	0x0000~0x7FFF	Standard Modbus address
	Word	%MW32768~%MW65535	0x20008000~0x2000FFFF	Delta-extended Modbus addresses

3.1.2.3 Latched Devices

The %MW0~%MW999 devices are latched devices in which data are retained when power off. Besides, the variables defined in the software can select Retain as its property. The capacity of latched devices is 128K bytes.

3.1.2.4 Data Types and Valid Ranges Supported

The data types and valid ranges of the variables in the software that DVP15MC11T uses are shown in the following table.

No.	Data type	Valid range	Initial value
1	BOOL	TRUE or FALSE	FALSE
2	BYTE	16#00 ~ FF	16#00
3	WORD	16#0000 ~ FFFF	16#0000
4	DWORD	16#00000000 ~ FFFFFFFF	16#00000000
5	LWORD	16#0000000000000000 ~ FFFFFFFFFFFFFFFF	16#0000000000000000
6	USINT	0 ~ +255	0
7	UINT	0 ~ +65535	0
8	UDINT	0 ~ +4294967295	0
9	ULINT	0 ~ +18446744073709551615	0
10	SINT	−128 ~ +127	0
11	INT	−32768 ~ +32767	0
12	DINT	−2147483648 ~ +2147483647	0
13	LINT	−9223372036854775808 ~ +9223372036854775807	0
14	REAL	−3.402823e+38 ~ −1.175495e-38, 0, +1.175495e-38 ~ +3.402823e+38	0.0
15	LREAL	−1.79769313486231e+308 ~ −2.22507385850721e-308, 0, +2.22507385850721e−308 ~ +1.79769313486231e+308,	0.0
16	TIME	T#XXXXXXdXXhXXmXXsXXXms，Unit: ns. Range:T#0ns~213503d23h34m33s709.551ms	T#0ms
17	DATE	D#Y-M-D. Range: D#1970-01-01~D#2106-02-07. Unit: s.	D#1970-01-01
18	TOD	TOD#H:M:S:MS, Range:TOD#00:00:00~23:59:59.999. Unit: ms. If 0 is written, TOD#00:00:00 is displayed. If 1 is written, TOD#00:00:00.001 is displayed. If 86399999 is written, TOD#23:59:59.999 is displayed. If 86400000 is written, TOD#00:00:00 is displayed. If 4294967295 is written, TOD#17:2:47.295 is displayed.	TOD#00:00:00
19	DT	DT#Y-M-D-H-M-S. Range: DT#1970-01-01-0:0:0~2106-02-07-6:28:15. Unit: s.	DT#1970-01-01-0:0:0
20	STRING	0~32000 characters	‘’

3.2 Electrical Specifications

- Electrical specification

Item	Content
Power voltage	24 VDC（-15% ~ +20%）
Fuse capacity	3 A/30 VDC, Polyswitch
Isolation voltage	500 VDC（Secondary-PE）
Consumption power	8W Max
Vibration/shock immunity	Standard: IEC61131-2,IEC 68-2-6 （TEST Fc）/IEC61131-2 & IEC 68-2-27 （TEST Ea）
Interference immunity	Static electricity: 8KV Air Discharge, 4KV Contact Discharge EFT: Power Line: ±2KV, Digital Input: ±1KV, Communication I/O: ±1KV RS: 80MHz ~ 1000MHz, 10V/m. Conducted Susceptibility Test: 150kHz ~ 80MHz, 3V/m Surge Test: Power line 0.5KV DM/CM
Environment	Work: 0°C ~ 55°C (Temperature), 5 ~ 95% (Humidity), pollution level 2 Storage: -25°C ~ 70°C (Temperature), 5 ~ 95% (Humidity).
Weight	About 425g

- Electrical specification for input points

Item	Content
Number of input channels	16 channels
Channel type	High-speed digital input type for the 16 channels
Input terminals	Terminal I0~I7．I10~I17
Common terminal for input points	Terminal S0/S1
Input type	Sink or Source mode
Input delay	2.5µS （OFF ->ON）, 5 µS （ON -> OFF）
Input current	24 VDC, 5mA
Max. cable length	The shielded cable: 500m； The unshielded cable: 300m

- Electrical specification for output points

Item	Content
Number of output channels	8 transistors for output (N-MOS)
Channel type	High-speed digital output type for 8 channels
Output terminals	Terminal Q0~Q7
Common terminal for output points	Terminal UP/ZP (Used for connection of anode or cathode of supply power)
Power voltage for output points	24 VDC（-15% ~ +20%）[#1]
Output delay	2µS （ OFF -> ON）, 3µS （ ON -> OFF）

Item	Content
Max. switch frequency	1KHZ
Max. loading	Resistance: 0.5A/1point （2A/ZP）
	Inductance: 13W（24VDC）
	Bulb: 2.5W（24VDC）
Max. cable length	The shielded cable: 500m
	The unshielded cable: 300m

#1: UP and ZP must connect the auxiliary power 24VDC (-15%~20%).

4

Chapter 4 System Architecture

Table of Contents

4.1 System Constitution 4-2

4.2 Power Supply 4-2

4.3 Left-side Extension 4-2
4.3.1 Connectable Left-side Extension Module 4-2
4.3.2 Allocation of Left-side Network Module Addresses 4-3
4.3.3 Method of Reading/Writing of Left-side Modules 4-3

4.4 Right-side Extension 4-3
4.4.1 Connectable Right-side Extension Modules 4-3
4.4.2 Allocation of Right-side Extension Module Addresses 4-4

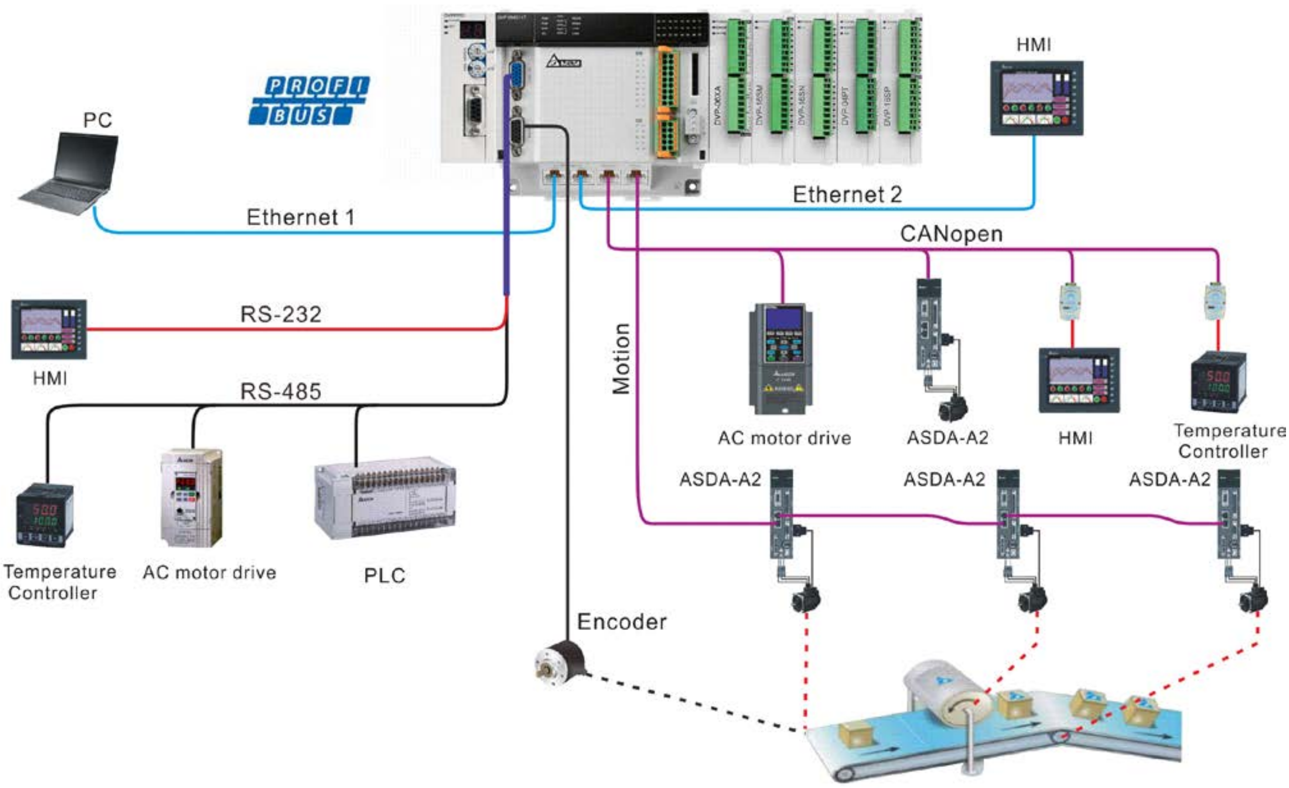
4.5 Connectable Servo Drives........ 4-4

4.6 SD Memory Card........ 4-6
4.6.1 Model and Specification 4-6
4.6.2 Function 4-8

4

4.1 System Constitution

A multi-layer industrial network can be built by means of DVP15MC11T. By using DVP15MC11T, the network can consist of top-layer Ethernet, middle-layer CANopen and Profibus bus as well as bottom-layer RS-485 bus which supports Modbus as follows.



The figure above illustrates the peripheral devices which are connected to various ports of DVP15MC11T in the entire system. Refer to chapter 6 for details on the functions of communication ports.

4.2 Power Supply

Delta power modules are recommended as the power supply for DVP15MC11T. The information of Delta power modules is shown in the following table.

No.	Module name	Phase	Input voltage	Output voltage	Power	Output current	International Standard
1	DVPPS02	Single phase	85~264VAC	24VDC	48W	2A	CE cULus
2	DVPPS05				120W	5A	

4.3 Left-side Extension

4.3.1 Connectable Left-side Extension Module

Max. 8 high-speed extension modules can be connected to the left side of DVP15MC11T and the connectable modules are listed in the following table.

No.	Module name	Module type	Description
1	DVP04AD-SL	Analog module	Analog input
2	DVP04DA-SL	Analog module	Analog output
3	DVPPF02-SL	Network module	Profibus communication

4.3.2 Allocation of Left-side Network Module Addresses

- **About Input and Output Mapping Areas of Left-side Network Modules**

The input and output mapping areas of different positions of the left side of PLC CPU are listed as follows when the network modules connected to the left side of DVP15MC11T serve as a slave. The position 1 is for the first module connected to the left side of PLC CPU; the position 2 is for the second one connected to the left side of PLC CPU and so on.

Mapping area / Position	Output mapping area	Input mapping area
1	%MW6250~%MW6377	%MW6000~%MW6127
2	%MW6750~%MW6877	%MW6500~%MW6627
3	%MW7250~%MW7377	%MW7000~%MW7127
4	%MW7750~%MW7877	%MW7500~%MW7627
5	%MW8250~%MW8377	%MW8000~%MW8127
6	%MW8750~%MW8877	%MW8500~%MW8627
7	%MW9250~%MW9377	%MW9000~%MW9127
8	%MW9750~%MW9877	%MW9500~%MW9627

4

Refer to the operation manuals of modules for details on allocation of left-side extension module mapping areas. Pay attention to how the mapping address expression format is changed in the operation manual. For example, the output mapping area for DVPPF02-SL is D6250~D6349. But the area address is expressed as %MW6250~%MW6349 when the module is connected to the left of DVP15MC11T.

4.3.3 Method of Reading / Writing of Left-side Modules

The controller can read and write the data in CR registers of the left-side extension modules via FROM/TO instruction. For instance, the modules such as DVP04AD-SL and DVP04DA-SL may use FROM/TO to read and write data in CR.

4.4 Right-side Extension

4.4.1 Connectable Right-side Extension Modules

Slim-series extension modules including digital modules, analog modules and temperature modules can be connected to the right side of DVP15MC11T. Digital modules can connect maximum 240 input points and 240 output points. Maximum 8 analog modules can be connected. The connectable right-side extension modules are listed in the following table.

No.	Module name	Input data length	Output data length	Extension type
1	DVP08SM11N	8 bits	-	Input point extension
2	DVP16SM11N	16 bits	-	
3	DVP06SN11R	-	6 bits	Output point extension
4	DVP08SN11R/T	-	8 bits	
5	DVP16SN11T	-	16 bits	
6	DVP08SP11R/T	4 bits	4 bits	Input extension and output extension
7	DVP16SP11R/T	8 bits	8 bits	
8	DVP16SP11TS（PNP）	8 bits	8 bits	
9	DVP32SM11N	32 bits	-	Pin-connector input

No.	Module name	Input data length	Output data length	Extension type
10	DVP32SN11TN	-	32 bits	Pin-connector output
11	DVP08ST11N	8 bits	-	Digital switch
12	DVP04AD-S	4 words	-	Analog input
13	DVP06AD-S	6 words	-	
14	DVP04DA-S	-	4 words	Analog output
15	DVP02DA-S	-	2 words	
16	DVP06XA-S	4 words	2 words	Analog input and analog output
17	DVP04PT-S	4 words	-	Sensor (Model: PT100)
18	DVP06PT-S	6 words	-	
19	DVP04TC-S	4 words	-	Sensor (Model: J, K, R, S, T thermocouples)

4.4.2 Allocation of Right-side Extension Module Addresses

4

DVP15MC11T can connect Slim-series extension modules to its right side and max. 240 digital input points and 240 digital output points are connectable. Max. 8 special modules are connectable such as analog modules, temperature modules and pulse modules. Up to 14 digital modules and special modules at most are connectable to the right side of DVP15MC11T.

- **Input point number and output point number of right-side digital extension modules**

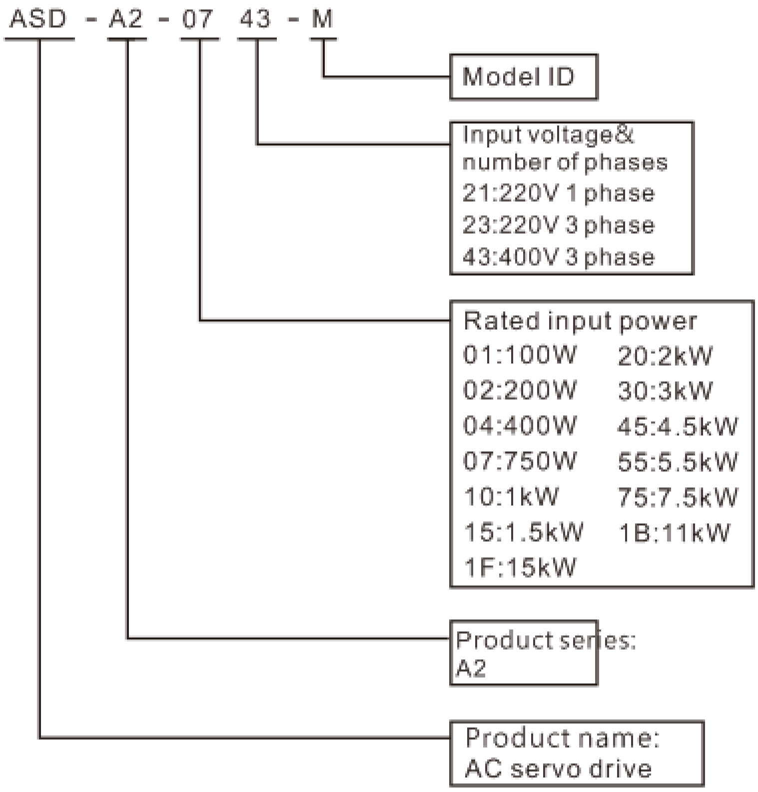
The input point number and output point number of the digital extension modules connected to the right of DVP15MC11T start from 2.0. For example, the input point for the first digital module starts from %IX2.0 and the output point starts from %QX2.0. It is counted as 8 points if the number is less than 8.
Digital input points and output points are numbered as below: (Octal)
%IX2.0 ~%IX2.7,......, %IX16.0 ~%IX16.7,......, %IX31.0 ~ %IX31.7
%QX2.0 ~ %QX2.7,......, %QX16.0 ~ %QX16.7,......, %QX31.0 ~ %QX31.7

- **About the right-side special module and serial number**
  - The right-side extension modules such as analog modules, temperature modules and pulse modules are regarded as special modules.
  - The serial number of the first special module to the right side of DVP15MC11T is 0; the serial number of the second one is 1, and so on. Maximum 8 special modules can be connected. The start address for input of the right-side special module is %MW10000 and the start address for output of the right-side special module is %MW10500.
  - DVP15MC11T can directly read and write the right-side module parameters through the hardware configuration interface of the software. Also, it can grant a value to an address or grant a value to a variable with which an address is combined in a program to read and write right-side module parameters.

4.5 Connectable Servo Drives

There are many models for ASDA-A2-series servo drives. ASDA-A2-XXXX-M model supports CANopen communication. Only ASDA-A2-XXXX-M servo drives can be used to build CANopen motion control network through connecting the motion port of DVP15MC11T. The connection between DVP15MC11T and the servo drive can be made with UC-CMC003-01A or UC-CMC005-01A cable through CN6 port.

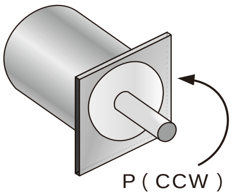
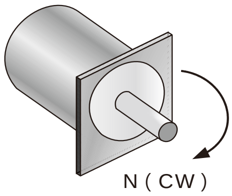
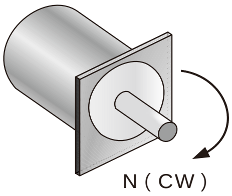
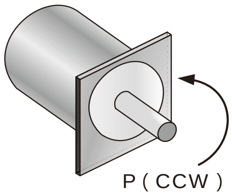
- **Illustration of the servo drive model**



- **Relevant servo parameter settings are shown in the following table when DVP15MC11T and the servo drive are connected.**

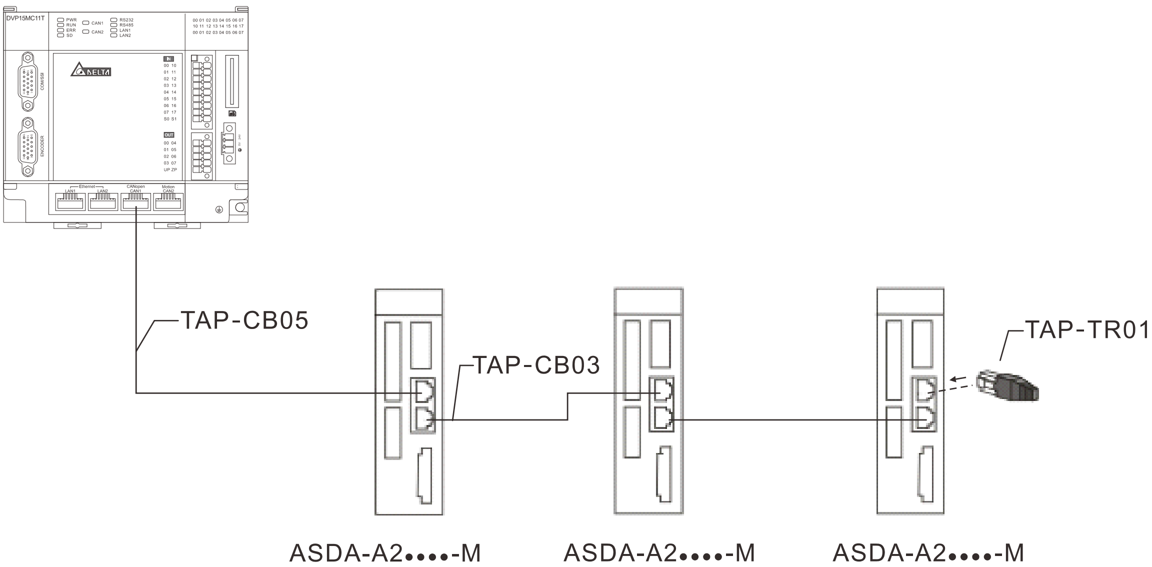
Parameter	Explanation	Setting value	Explanation
P1-01	Setting the control mode of the servo	X0B*[1]	Set as CANopen mode
P3-00	Setting a node ID	Setting range: 1~24	The setting of this parameter corresponds to the node address of the servo in the CANopen network
P3-01	Baud rate	0403	The baud rate that the parameter value corresponds to must be consistent with that of DVP15MC11T. 0403: CANopen baud rate is 1Mbps 0203: CANopen baud rate is 500Kbps

*[1]: The output directions of the torque are illustrated as below when the value of X is 0 and 1 respectively.

	0	**1**
Positive direction	 P (CCW)	 N (CW)
Negative direction	 N (CW)	 P (CCW)

- **The wiring figure of DVP15MC11T and ASDA-A2-XXXX-M-series servo drives**

DVP15MC11T



Notes:

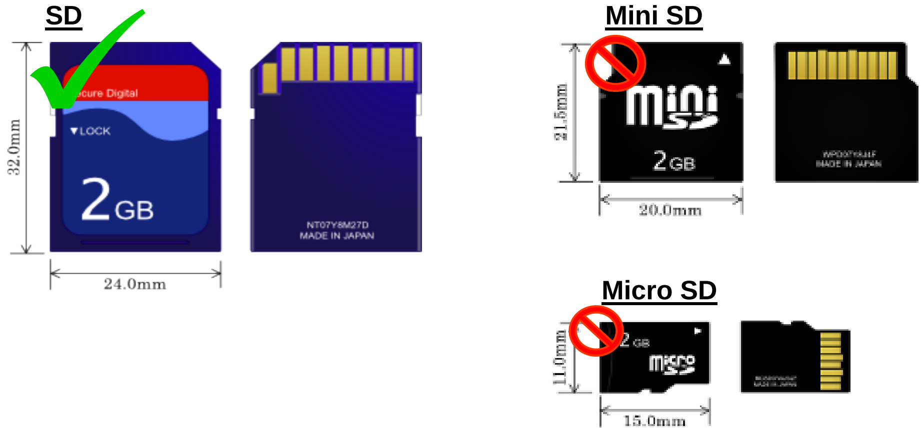
1. Please refer to the servo user manual for the wiring of ASDA-A2-XXXX-M-series servo drives, servo motors and encoders.
2. Choose UC-CMC003-01A or UC-CMC005-01A or UC-CMC010-01A communication cable according to the field status.
3. There is one 120Ω terminal resistor embedded at Motion port. In the CANopen network consisting of Motion port and servos, the other end of the network must be connected with a terminal resistor TAP-TR01 which could be found in the packing box of DVP15MC11T.

4.6 SD Memory Card

4.6.1 Model and Specification

- **Model and Appearance**

SD memory cards can be classified into SD, Mini SD and Micro SD according to its size. DVP15MC11T only supports the standard-dimension SD.



- **Specification**

There are various SD card specifications on current market. Except that SD cards are different in size, they can be classified into SD, SDHC and SDXC according to its capacity. However, DVP15MC11T only supports basic SD specification currently. The following table includes the information of SD card family members. DVP15MC11T only supports SD and SDHC. Please make sure to purchase the SD card of the right specification that DVP15MC11T supports.

- **SD card classification**

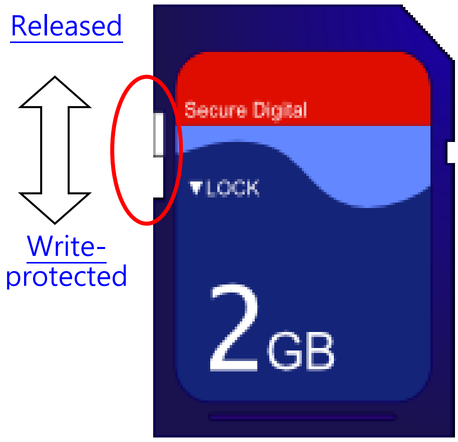
Class	SD	SDHC			SDXC	
Capacity	32MB~2GB	4GB~32GB			32GB~2TB	
File system	FAT16/FAT32	FAT32			exFAT（FAT64）	
Size	SD	SDHC	Mini SDHC	Micro SDHC	SDXC	Micro SDXC
SD speed level	N/A	CLASS 2 (Min. 2MB/Sec.) CLASS 4 (Min. 4MB/Sec.) CLASS 6 (Min. 6MB/Sec.) CLASS 10 (Min. 10MB/Sec.)			CLASS 2 (Min. 2MB/Sec.) CLASS 4 (Min. 4MB/Sec.) CLASS 6 (Min. 6MB/Sec.) CLASS 10 (Min. 10MB/Sec.)	

* Please notice that there is a kind of MMC card which is very similar to SD card in appearance and thus please differentiate them carefully during purchase.

- **Before use of SD card**

  - **Write-protection function of the memory card**

  There is a write-protection switch for general SD cards. The data can not be written into SD card if the switch is moved to the Lock position. Hence, please ensure that the write-protection switch of SD card has been released correctly before SD card is used and then the write-into function can be executed in DVP15MC11T.



4.6.2 Function

The main purpose of SD card is to upgrade the firmware of DVP15MC11T.

4

5

Chapter 5 Installation

Table of Contents

5.1 Dimensions 5-2

5.1.1 Profile and Dimensions of DVP15MC11T 5-2

5.1.2 Dimensions of Left-side and Right-side Extension Modules 5-2

5.1.3 Connecting to the Left-side Extension Module 5-3

5.1.4 Connecting to the Right-side Extension Module 5-4

5.1.5 SD Card Installing and Removing 5-5

5.2 Installing the Module in the Control Cabinet 5-7

5.2.1 Installing the Module to DIN rail 5-7

5.2.2 Illustration of Installation Inside the Control Cabinet 5-7

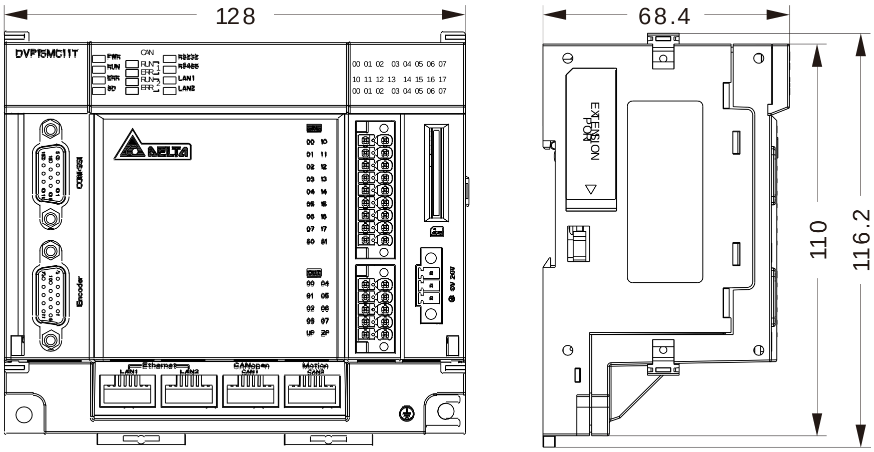
5.2.3 Environmental Temperature in the Control Cabinet 5-7

5.2.4 Actions for Anti-interference 5-8

5.2.5 Dimension Requirement in the Control Cabinet 5-8

5.1 Dimensions

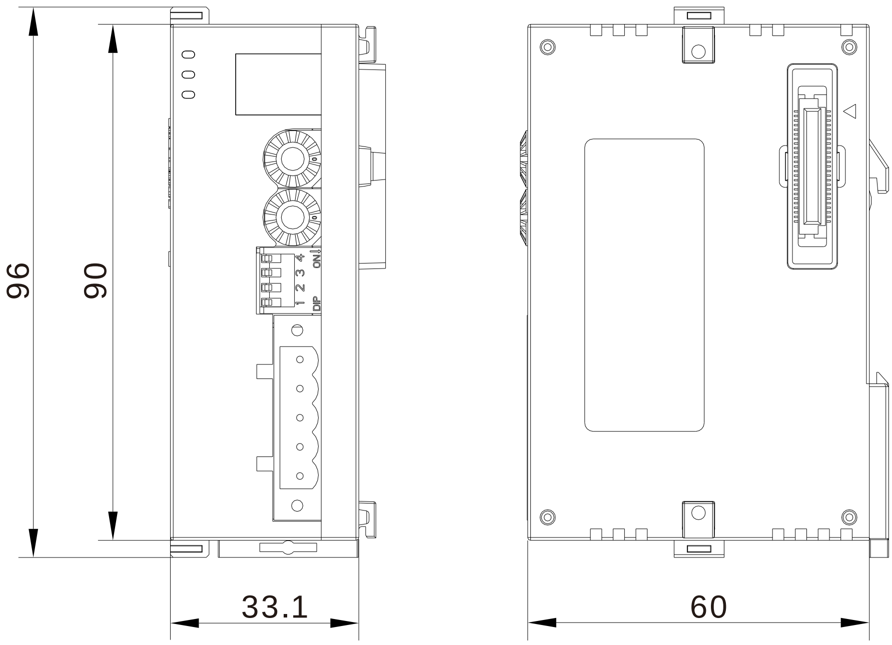
5.1.1 Profile and Dimensions of DVP15MC11T



Unit: mm

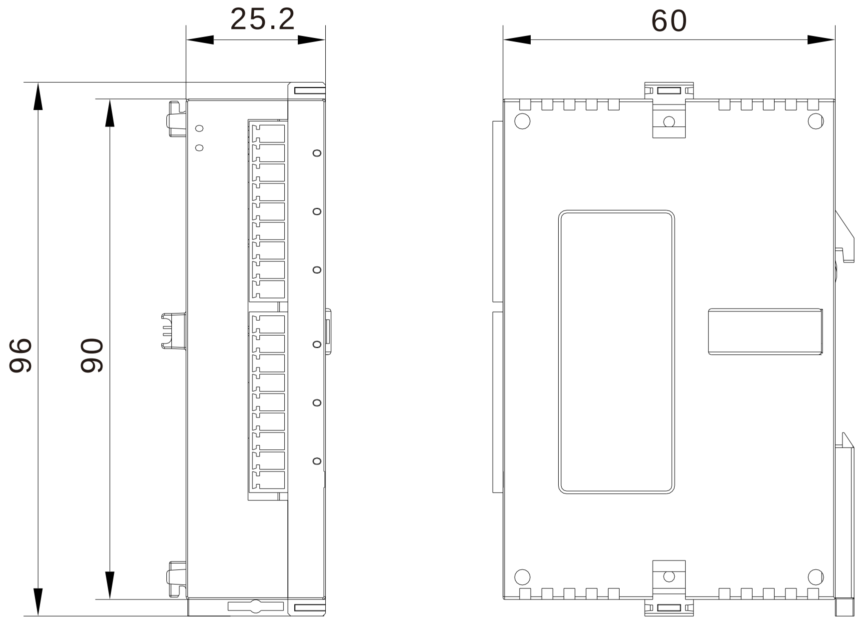
5.1.2 Dimensions of Left-side and Right-side Extension Modules

- See the following dimension figure of a left-side extension module by taking DVPCOPM-SL for example. The length, width and height of all left-side modules are the same as that of DVPCOPM-SL.



Unit: mm

- See the following dimension figure of a right-side extension module, which takes DVP04AD-S for example. The length, width and height of all left-side modules are the same as that of DVP04AD-S.



Unit: mm

5.1.3 Connecting to the Left-side Extension Module

- Connection of DVP15MC11T and DVPDNET-SL
  - Pull open the extension module clips on the top left and bottom left of DVP15MC11T and install DVPDNET-SL along four mounting holes in the four angles of DVP15MC11T as step 1 in figure 5.1.3.1.
  - Press the clips respectively on the top left and bottom left of DVP15MC11T to fix the module tightly and ensure that their contact is normal as step 2 in figure 5.1.3.1.

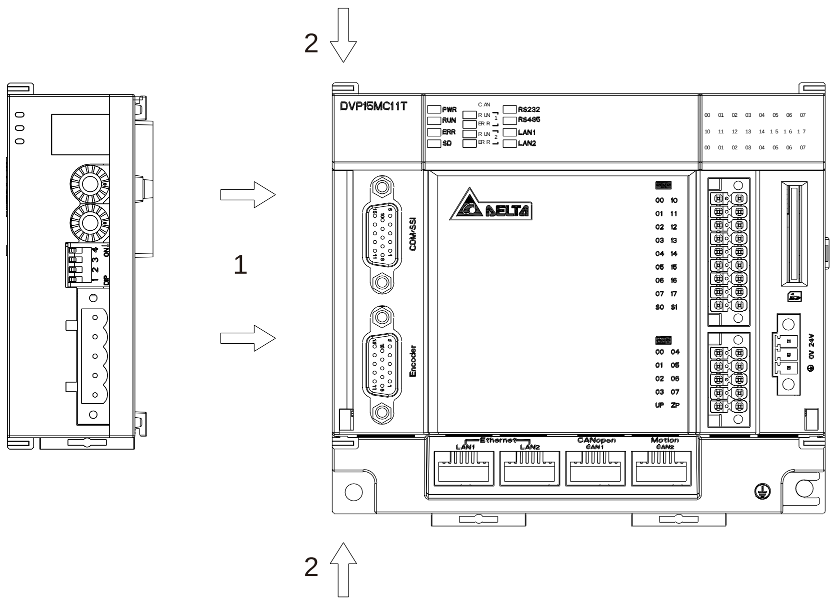


Figure 5.1.3.1

- Installing DVP15MC11T and DVPDNET-SL into DIN rail
  - Use standard 35mm DIN rail.
  - Pull open DIN rail clips of DVP15MC11T and DVPDNET-SL and then insert the two modules into DIN rail.

- Press the DIN rail clips into DVP15MC11T and DVPDNET-SL to fix the two modules in DIN rail as figure 5.1.3.2.

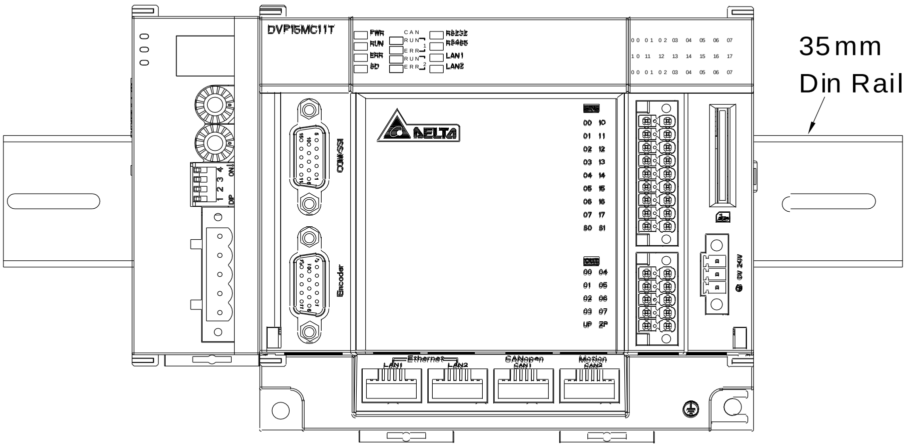


Figure 5.1.3.2

5.1.4 Connecting to the Right-side Extension Module

- Connection of DVP15MC11T and DVP16SP11T
  - Pull open the extension module clips on the top right and bottom right of DVP15MC11T and install DVP16SP11T along four mounting holes in the four angles of DVP15MC11T as step 1 in figure 5.1.4.1.
  - Press the clips on the upper right and bottom right of DVP15MC11T to fix the module tightly and ensure that their contact is normal as step 2 in figure 5.1.4.1

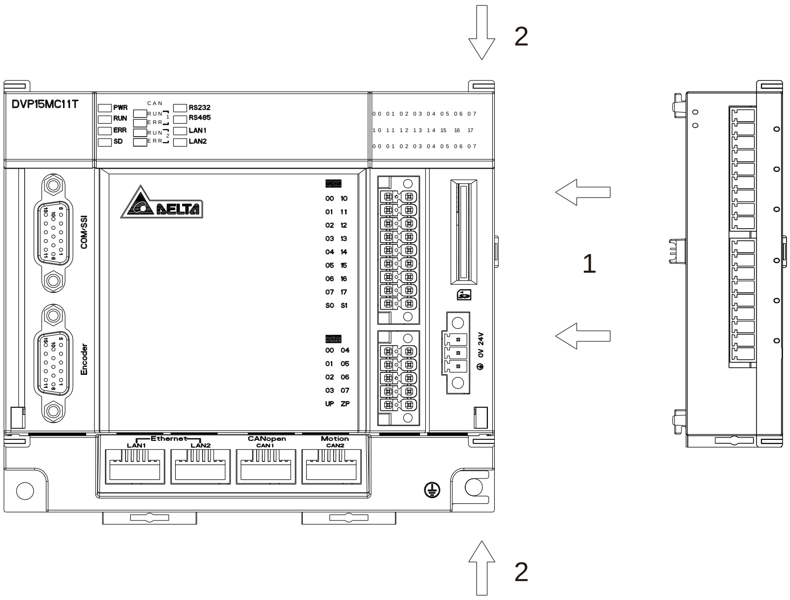


Figure 5.1.4.1

- Installing DVP15MC11T and DVP16SP11T in DIN Rail
  - Use standard 35mm DIN rail.
  - Pull open DIN rail clips of DVP15MC11T and DVP16SP11T and then insert the two modules into DIN rail.
  - Press the DIN rail clips into DVP15MC11T and DVP16SP11T to fix the two modules in DIN rail as figure 5.1.4.2.

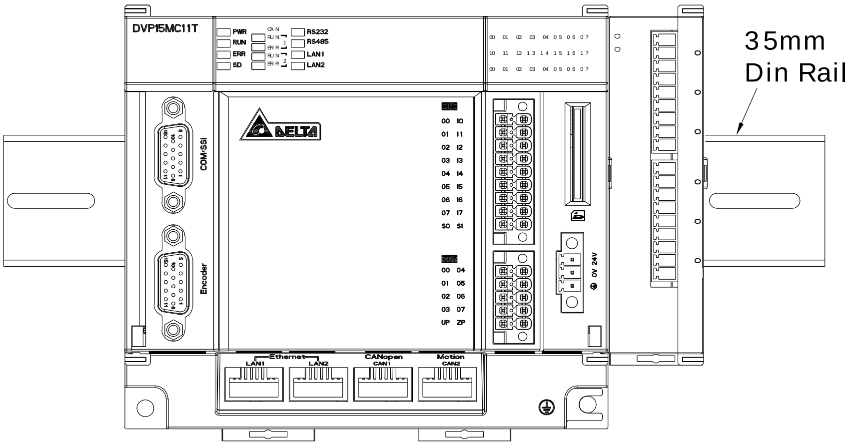
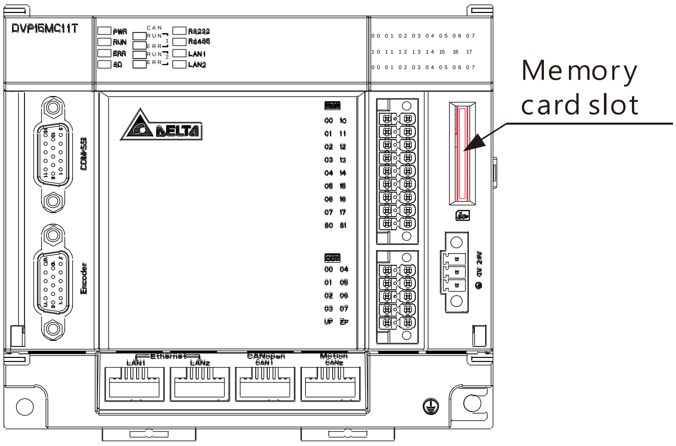


Figure 5.1.4.2

5.1.5 SD Card Installing and Removing

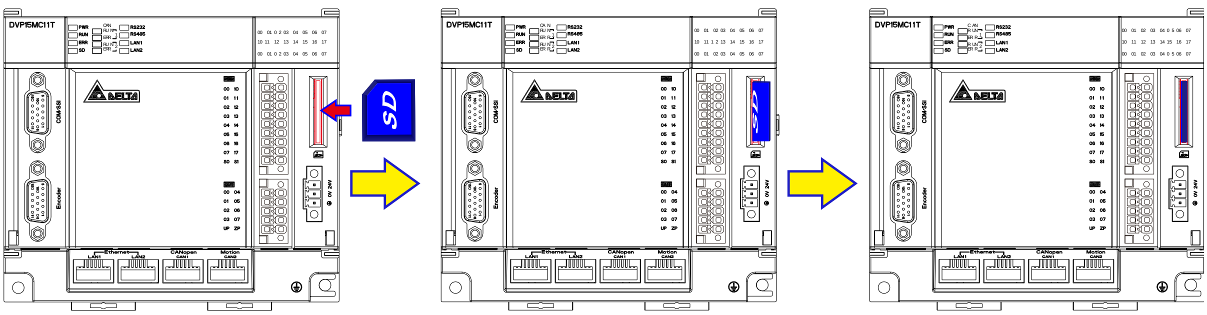
- The memory card slot of DVP15MC11T

The memory card slot is seated in the right side of the front of DVP15MC11T as illustrated below.



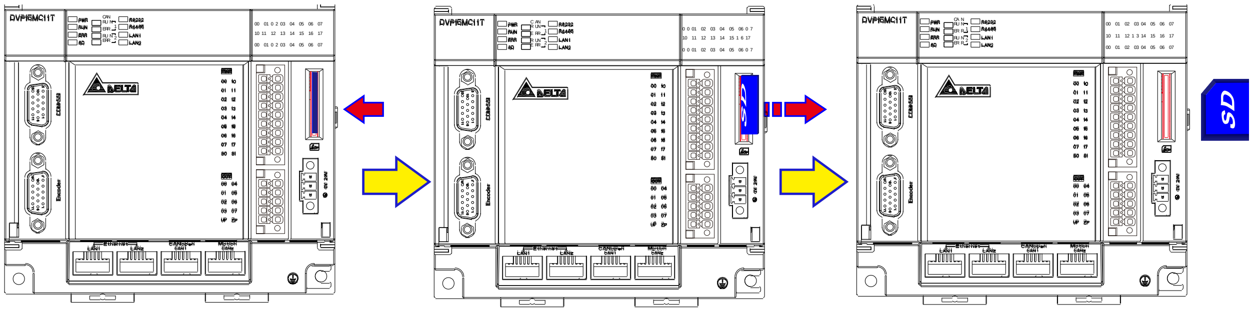
- Installing SD card

Insert an SD card to the memory card slot directly and push it to the end of the slot until hearing a click. After the installation is finished, the SD card should be fixed tightly. If the SD card inserted to the slot is loose, the installation is unsuccessful. In addition, the SD card has a fool-proofing design. If the direction in which SD card is inserted is wrong, the card will fail to reach the end of the slot. In this case, do not force to push the SD card toward the end of the slot in order to avoid the damage to the module and SD card.
Follow the instructions in the figures below to insert the SD card in the right direction.



- Removing SD card

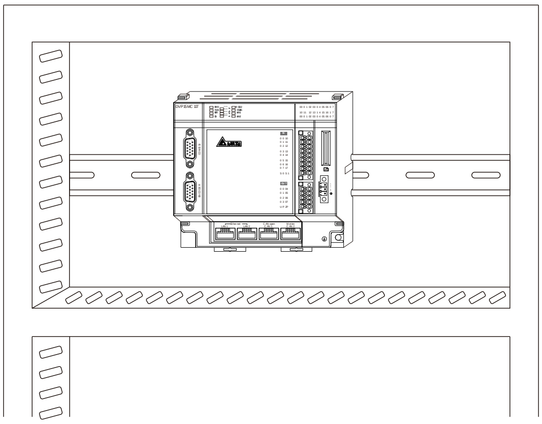
Just push the SD card to the end of the slot so that the SD card will loosen and rebound from inside the slot. And then remove the SD card out of the slot easily.



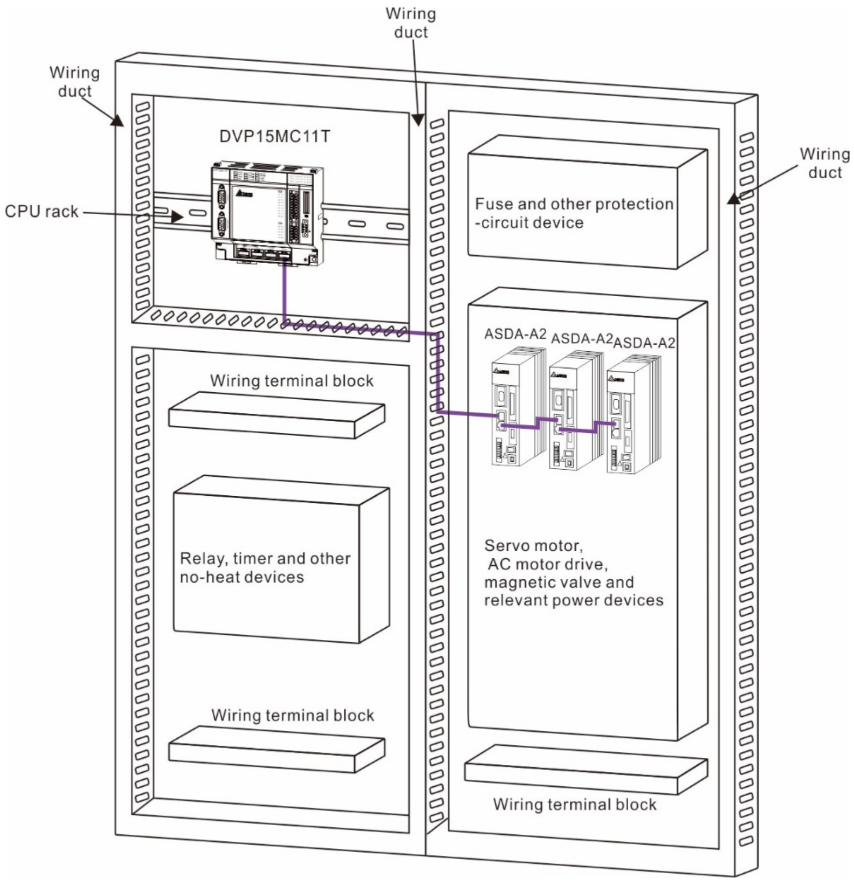
5.2 Installing the Module in the Control Cabinet

5.2.1 Installing the Module to DIN rail

Pull down the clips at the bottom of DVP15MC11T. Then stick the horizontal slots at the rear of the module on the DIN rail. Finally, push up the clips to fix the module inside the control cabinet.



5.2.2 Illustration of Installation Inside the Control Cabinet



5.2.3 Environmental Temperature in the Control Cabinet

- **Requirements**
  1. The environment inside the control cabinet for DVP15MC11T is 0°C ~ 55°C in temperature and 5 ~ 95% in humidity.
  2. Please do not make the installation near the equipment of high temperature.

3. Keep enough space for air ventilation.
4. The fan or air conditioner must be installed if the environment temperature is higher than 55°C.

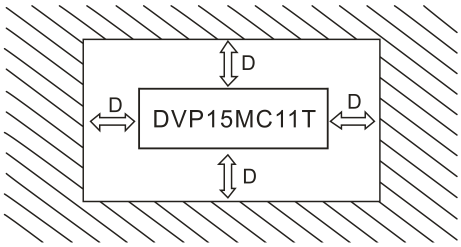
- **Notes:**
  1. The control cabinet of the height 1.0m~2.0m is easy for installation and operation.
  2. Make the installation away from the high-voltage equipment and power equipment.
  3. The power supply in the control cabinet must be cut before installation.

5.2.4 Actions for Anti-interference

- Do not install the controller in the control cabinet where there is high-voltage equipment.
- Please keep at least 200mm far away from the power wire for the installation.
- There should be a grounding wire for the control cabinet.

5.2.5 Dimension Requirement in the Control Cabinet

- **Installation Figure**

DVP15MC11T has to be installed in an enclosure. In order to ensure that the controller radiates heat normally, the space between the controller and the enclosure has to be larger than 50 millimeters. D > 50mm	

5

6

Chapter 6 Wiring, Communication Setting and Network Construction

Table of Contents

6.1 Wiring 6-3
6.1.1 Power Supply 6-3
6.1.2 Safety Circuit Wiring 6-3

6.2 Input Point and Output Point Wiring 6-4
6.2.1 Function that Input Points Support 6-4
6.2.2 Input Point Wiring 6-5
6.2.3 Output Point Wiring 6-6

6.3 RS-485 Communication Port 6-8
6.3.1 Function that RS-485 Port Supports 6-8
6.3.2 Definitions of RS-485 Port Pins 6-8
6.3.3 RS-485 Hardware Connection 6-8
6.3.4 Supported Function Codes and Exception Codes 6-10

6.4 RS-232 Communication Port 6-11
6.4.1 Function that RS-232 Port Supports 6-11
6.4.2 Definitions of RS-232 Port Pins 6-11
6.4.3 RS-232 Hardware Connection 6-11
6.4.4 Supported Function Codes and Exception Codes 6-12

6.5 SSI Absolute Encoder Port 6-13
6.5.1 Function of SSI Absolute Encoder 6-13
6.5.2 Definitions of SSI Port Pins 6-13
6.5.3 SSI Absolute Encoder Hardware Connection 6-13

6.6 Incremental Encoders 6-15
6.6.1 Function of Incremental Encoder 6-15
6.6.2 Definition of Incremental Encoder Port Pins 6-15
6.6.3 Incremental Encoder Hardware Connection 6-16

6.7 Ethernet Communication Port 6-17
6.7.1 Function that Ethernet Communication Port Supports 6-17
6.7.2 Pins of Ethernet Communication Port 6-17
6.7.3 Network Connection of Ethernet Communication Port 6-17
6.7.4 Function Codes that Ethernet Communication Port Supports 6-18

6.8 Motion Communication Port 6-18

6.8.1 Function that Motion Communication Port Supports 6-18
6.8.2 Pins of Motion Communication Port 6-18
6.8.3 Motion Network Connection 6-19
6.8.4 Communication Speed and Communication Distance 6-19

6.9 CANopen Communication Port 6-20
6.9.1 Functions that CANopen Communication Port Supports 6-20
6.9.2 Pins of CANopen Communication Port 6-21
6.9.3 PDO Mapping at CANopen Communication Port 6-21
6.9.4 Network Connection at CANopen Communication Port............................... 6-21
6.9.5 CANopen Communication Rate and Communication Distance 6-22

6.1 Wiring

6.1.1 Power Supply

The power input of DVP15MC11T CPU is 24V DC input. Please notice the following points when operating DVP15MC11T.

1. The range of the power is 20.4VDC~ 28.8VDC. The power is connected to two terminals, 24V and 0V and the grounding terminal should be in the ground connection. Please note that DVP15MC11T will probably be damaged if the positive and negative polarities of the power are connected wrongly.
2. The cable of 1.6mm or above is used for connecting the ground terminal of DVP15MC11T.
3. Too long power shutdown time or power voltage drop will stop DVP15MC11T running and communicating with the servo drive and all output will turn off. DVP15MC11T will resume the connection with the servo drive when the power returns to normal.

6.1.2 Safety Circuit Wiring

The action of any device inside DVP15MC11T may affect the behavior of the external equipment under DVP15MC11T's control over the servo drive. Therefore, any device trouble may cause the whole automatic control system to lose control and even result in injuries and death of personnel. For these reasons, we suggest the following safety device should be added to the power input circuit.

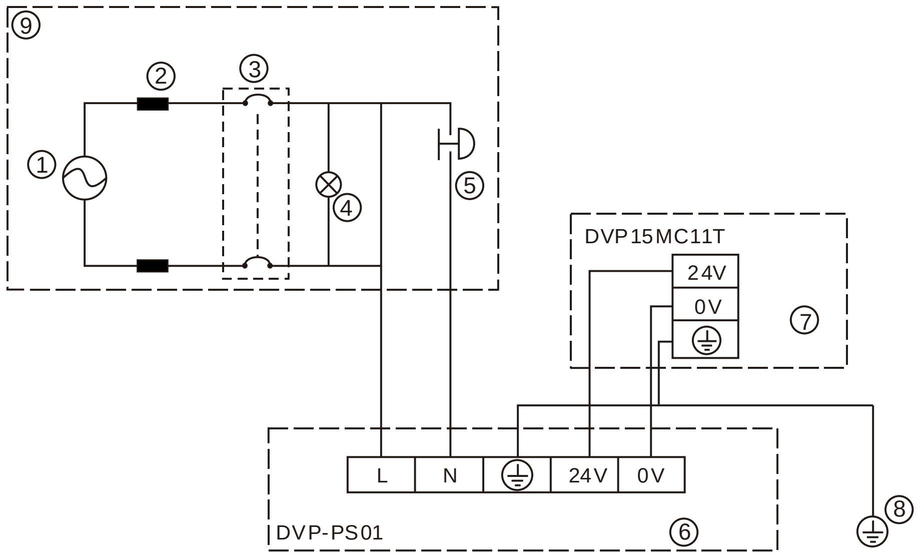


Figure 3.2.1

①	AC power supply: 100～240VAC；50/60Hz。
②	Power supply circuit protection fuse
③	System circuit isolation device: The electromagnetic contactor, relay and other switch can be used as the isolation device to prevent the system from becoming unstable when the power supply is discontinuous.
④	Power indicator
⑤	Emergency stop button: The button cuts off the system power supply when an accidental emergency takes place.
⑥	Delta power module DVPPS02/24VDC (DVPPS02 is recommended for DVP15MC11T)

⑦	DVP15MC11T
⑧	Ground
⑨	Safety circuit

6.2 Input Point and Output Point Wiring

6.2.1 Function that Input Points Support

There are 16 input points which support external interrupt and filter functions in DVP15MC11T. In addition, the input points can be used to capture the encoder position.
Refer to the explanation of the DMC_TouchProbe instruction for details on position capture.

- **The work principle of the input filter**

  The input filter filters short pulse signals via the 16 I points I0~I7 and I10~I17 to reduce the influence of the input interference signals. Increasing the filter value can decrease the vibration of input signals or the influence from external interference.

  Input filter time: t=31us *（0~255）. So the filter time is a multiple of 31us and 0 is the default value. The input filter time can be set through the software.

  - **When there is the set filter:**

    When the filter time is set to t (us), the signal is valid if the ON or OFF time of the input signal is greater than t (us). If the ON or OFF time of input signal is less than t (us), the signal will be eliminated. The input signal left after being filtered will be input after being delayed by t (us).

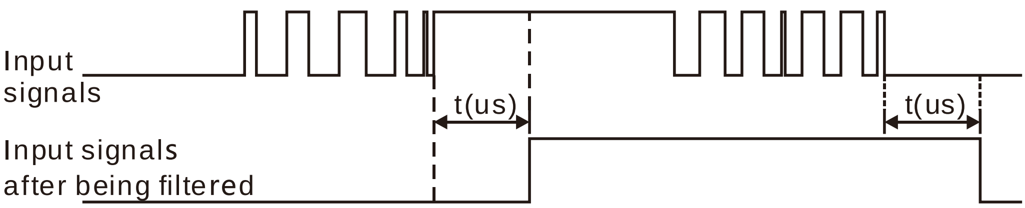


Figure 6.2.1.1

  - **When there is no filter set:**

    The input signals have no change when no filter time is set.

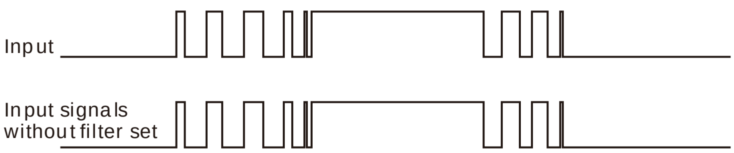


Figure 6.2.1.2

6.2.2 Input Point Wiring

There are two types of DC inputs, SINK and SOURCE. See the details for the wiring in the following two modes.

- **Sink Mode**

  Under Sink mode, the simplified model is shown below and the current flows into the common ports S0 and S1.

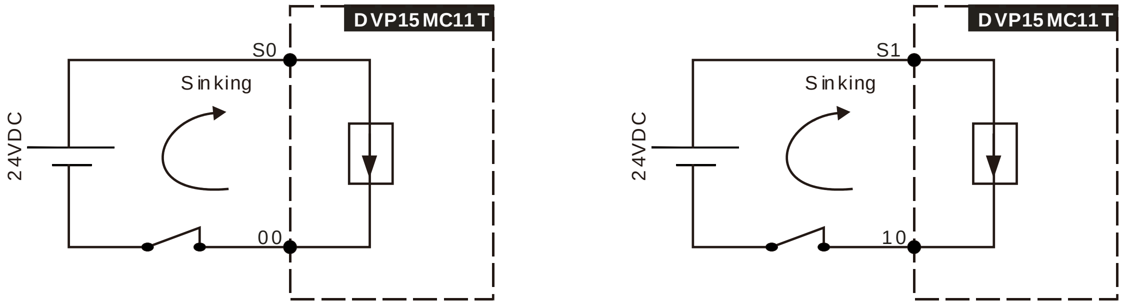


Figure 6.2.2.1

See the relevant wiring circuit in the following figures.

1. The input points of DVP15MC11T, 00~07 correspond to S0 as shown below.

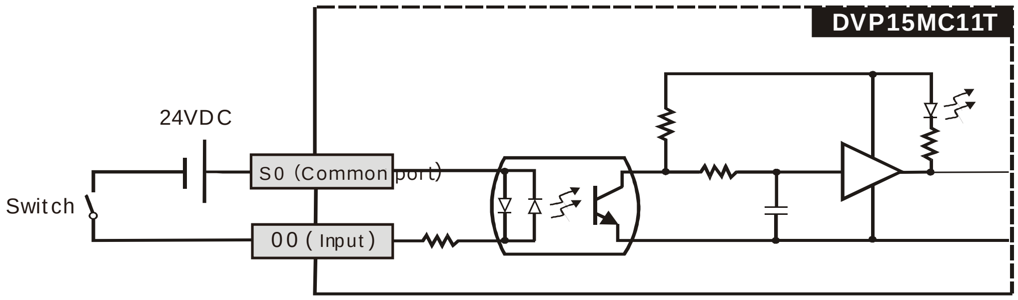


Figure 6.2.2.2

2. The input points of DVP15MC11T, 10~17 correspond to S1 as shown below.

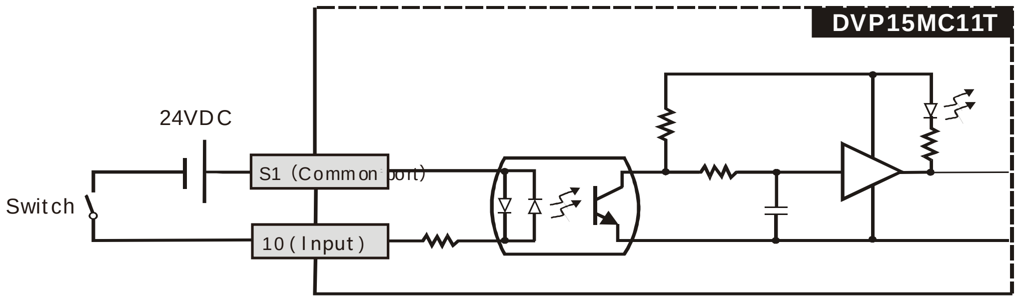


Figure 6.2.2.3

- **Source Mode**

  Under Source mode, the simplified model is illustrated below and the current flows into the common ports S0 and S1.

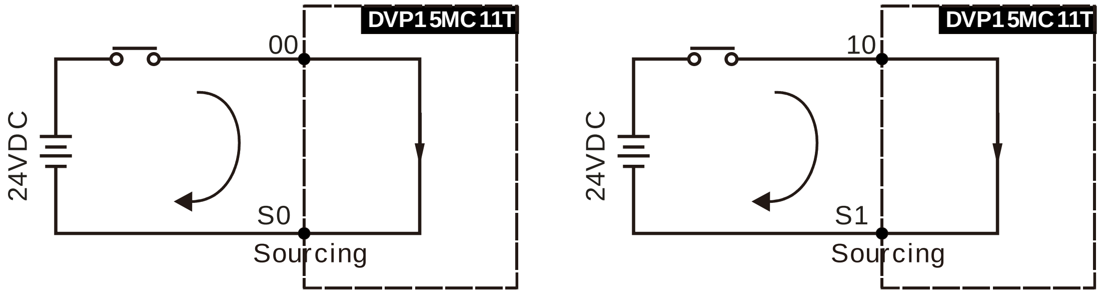


Figure 6.2.2.4

See the wiring circuit below

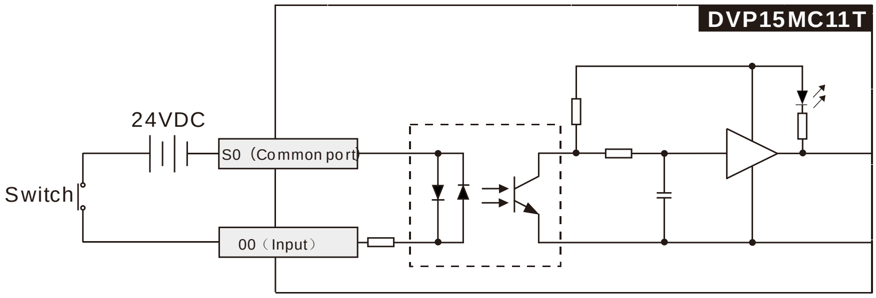


Figure 6.2.2.5

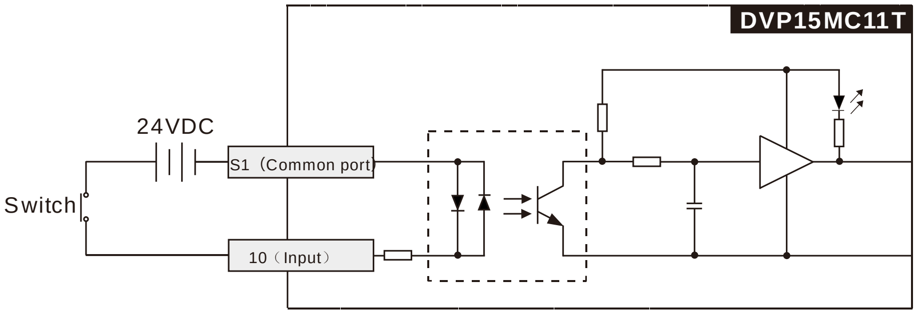


Figure 6.2.2.6

6.2.3 Output Point Wiring

All transistor outputs in DVP15MC11T contain diodes for suppression which are sufficient for use in the inductive load of smaller power and infrequent On/Off. However, in the event of larger power and frequent On/Off, the following suppression circuit is necessary for reducing interferences and preventing the transistor output circuit from being damaged due to overvoltage or overheat.

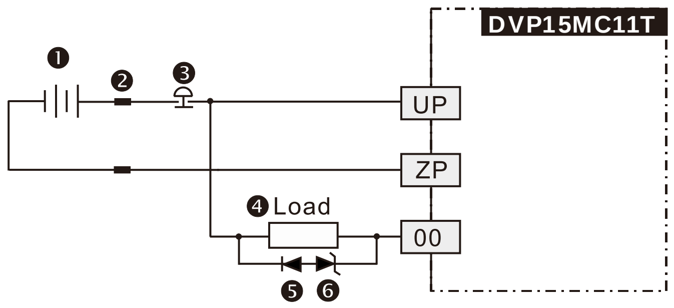


圖 3.2.6

❶	DC power supply of 24 V
❷	Circuit protection fuse
❸	Emergency stop button
❹	Switch, inductive load
❺	component for suppression (❻ is not used but ❺ when in smaller power).
❻	9V Zener diode, 5W (❺ and ❻ are both used when in bigger power and frequent On/Off.

6.3 RS-485 Communication Port

6.3.1 Function that RS-485 Port Supports

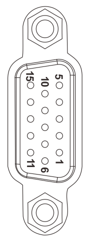
The RS-485 communication port of DVP15MC11T can function as Modbus master or slave. HMI, PLC or other Modbus master device can read and write data in the devices inside DVP15MC11T. The interval time when the Modbus master accesses DVP15MC11T should exceed 5ms.

The progrom can not be downloaded via RS-485 port. RS-485 supports Modbus protocol, ASCII as well as RTU mode. The function codes which RS-485 port supports include 0x01, 0x02, 0x03, 0x05, 0x06, 0x0F and 0x10. The station addresses that RS-485 port supports are 1~255. The broadcast function is not supported.

Refer to appendix A for details on Modbus communication and Modbus device addresses.

6.3.2 Definitions of RS-485 Port Pins

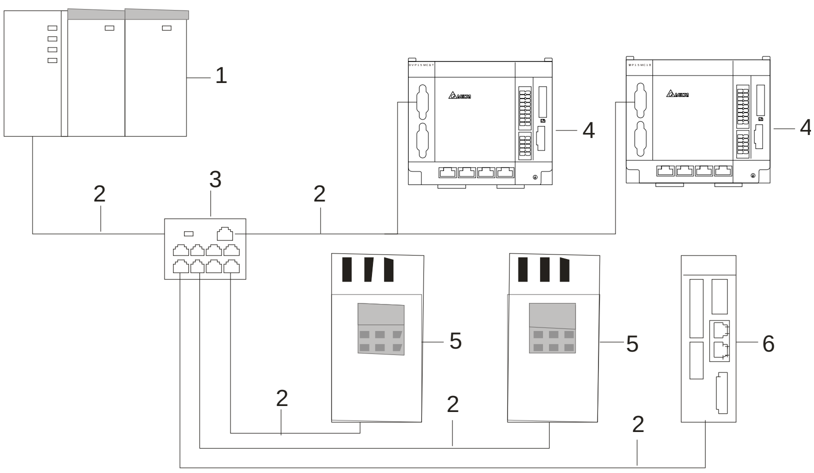
DVP15MC11T's COM/SSI port consists of 15 pins. The external port is commonly used for RS-485 communication and SSI absolute encoder. See the table below for definitions of respective RS-485 communication port pins.

Pin No.	Signal	Definition	 COM/SSI
11	D+	Positive pole	
12	D-	Negtive pole	
5	SG	Signal ground	

6.3.3 RS-485 Hardware Connection

- **Example on Connection of DVP15MC11T into Modbus Network**

DVP15MC11T is connected to Modbus network via RS-485.



Device No.	1	2	3	4	5	6
Device name	Modbus master	Communication cable	VFD-CM08	DVP15MC11T	AC motor drive	Servo drive

- **RS-485 Wiring:**

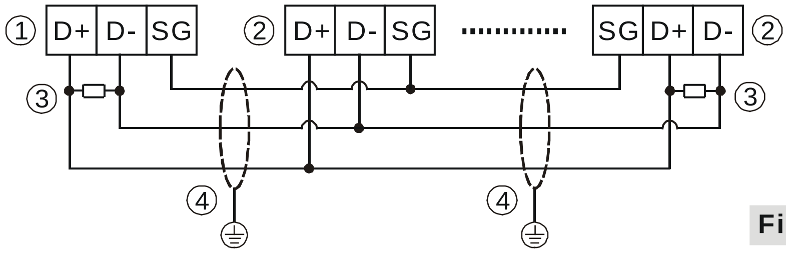


Figure 19

Explanation of numbers

①	②	③	④
Master	Slave	Terminal resistor	Shielded cable

Notes:

1. Terminal resistors with the value of 120Ω are recommended to connect to both ends of the bus.
2. To ensure high communication quality, please use the shielded twisted pair cable (20AWG).
3. When the internal voltages of two devices are different, make SG (Signal Ground) of the two devices connected with each other to balance their SG voltages and make the communication more stable.

- **Communication Format that RS-485 Supports**

RS-485 communication port supports ASCII or RTU communication formats and the supported baud rate can be up to 115200bps.

Baud rate	9600, 19200, 38400, 57600, 115200					
Mode	ASCII				RTU	
Communication format	7,E,1	7,E,2	7,N,1	7,N,2	8,E,1	8,E,2
	7,O,1	7,O,2	8,E,1	8,E,2	8,N,1	8,N,2
	8,N,1	8,N,2	8,O,1	8,O,2	8,O,1	8,O,2

6

6.3.4 Supported Function Codes and Exception Codes

- **Modbus Function Codes:**

1. The function codes that RS-485 port of DVP15MC11T supports are listed in the following table.

Function code	Indication	Whether to broadcast (Y/N)	Max. number of writable/readable registers	Available register
0x01	Read output bit register values.	N	256	Bit register
0x02	Read bit register values.	N	256	Bit register
0x03	Read one single or multiple word register values.	N	100	Word register
0x05	Write one single bit register value.	Y	1	Bit register
0x06	Write one single word register value.	Y	1	Word register
0x0F	Write multiple bit register values.	Y	256	Bit register
0x10	Write multiple word register values.	Y	100	Word register

2. The exception codes that RS-485 port of DVP15MC11T supports are listed in the following table.

Exception response code	Indication
0x01	Unsupportive function code
0x02	Unsupportive Modbus address
0x03	The data length is out of the valid range.

6.4 RS-232 Communication Port

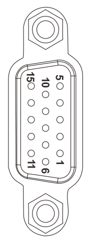
6.4.1 Function that RS-232 Port Supports

The RS-232 communication port of DVP15MC11T can function as Modbus master or slave. HMI, PLC or other Modbus device can read and write data in the devices inside DVP15MC11T. The progrom can not be downloaded through RS-232 port. RS-232 supports Modbus protocol, ASCII mode as well as RTU mode. The function codes which RS-232 port supports include 0x01, 0x02, 0x03, 0x05, 0x06, 0x0F and 0x10. The station addresses that RS-232 port supports are 1~255. The broadcast function is not supported.

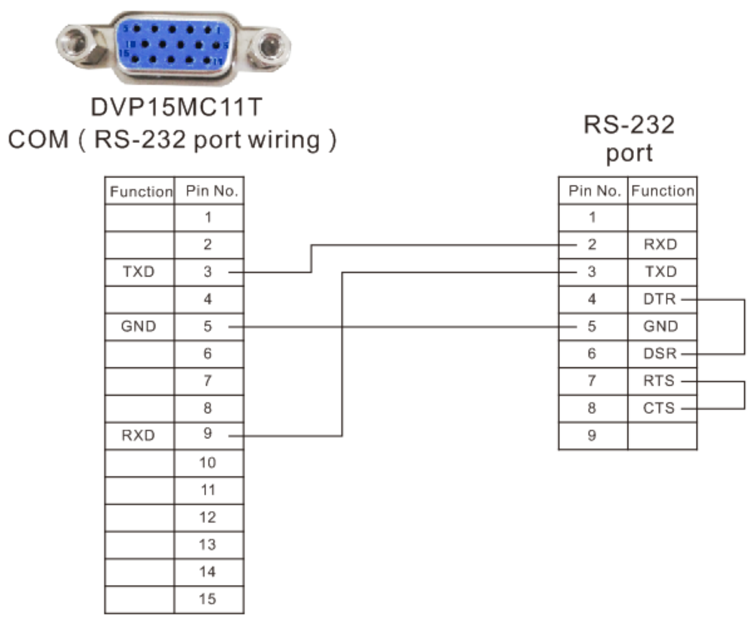
Refer to appendix A for details on Modbus communication and Modbus device addresses.

6.4.2 Definitions of RS-232 Port Pins

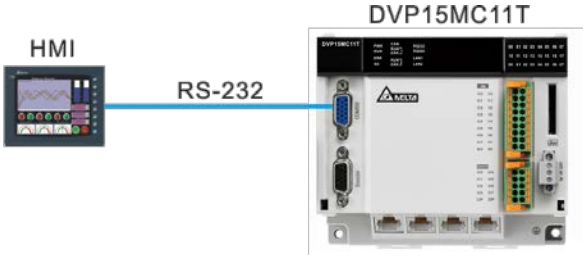
DVP15MC11T's COM/SSI port consists of 15 pins. See the table below for definitions of respective RS-232 communication port pins.

Pin No.	Signal	Definition	
3	Tx	Transmitting data	 COM/SSI
9	Rx	Receiving data	
5	GND	Signal ground	

6.4.3 RS-232 Hardware Connection



- RS-232 port is connected to HMI when DVP15MC11T functions as a slave.



- **The communication format that RS-232 supports**

Baud rate	9600, 19200, 38400, 57600, 115200					
Mode	ASCII				RTU	
Communication format	7,E,1	7,E,2	7,N,1	7,N,2	8,E,1	8,E,2
	7,O,1	7,O,2	8,E,1	8,E,2	8,N,1	8,N,2
	8,N,1	8,N,2	8,O,1	8,O,2	8,O,1	8,O,2

6.4.4 Supported Function Codes and Exception Codes

- **Modbus Function Codes:**

1. The function codes that RS-232 port of DVP15MC11T supports are listed in the following table.

Function code	Indication	Max. number of writable/readable registers	Available register
0x01	Read output bit register values.	256	Bit register
0x02	Read bit register values.	256	Bit register
0x03	Read one single or multiple word register values.	100	Word register
0x05	Write one single bit register value.	1	Bit register
0x06	Write one single word register value.	1	Word register
0x0F	Write multiple bit register values.	256	Bit register
0x10	Write multiple word register values.	100	Word register

2. The exception codes that RS-232 port of DVP15MC11T supports are listed in the following table.

Exception code	Indication
0x01	Unsupportive function code
0x02	Unsupportive Modbus address
0x03	The data length is out of the valid range.

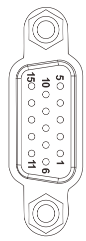
6.5 SSI Absolute Encoder Port

6.5.1 Function of SSI Absolute Encoder

DVP15MC11T's COM/SSI port is a 15-pin D-SUB interface which can be used to connect SSI encoder. In addition, the port also includes the 5V (400mA) power output which provides the power supply to the encoder. Users can create an SSI encoder axis to control the motion of slave axes according to the number of pulses received via the encoder port.

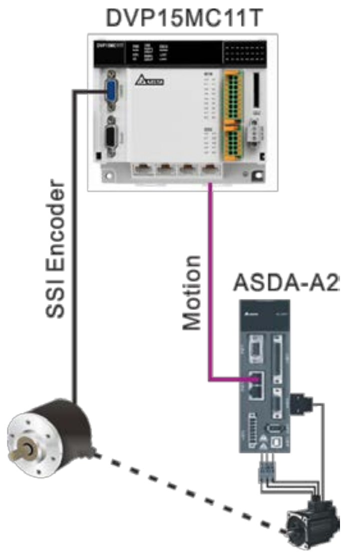
6.5.2 Definitions of SSI Port Pins

DVP15MC11T's COM/SSI port is a 15-pin D-SUB interface. See the table below for definitions of respective SSI communication port pins.

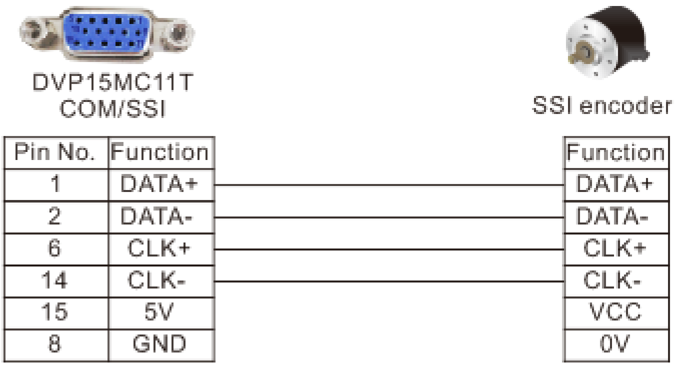
Pin No.	Signal	Definition	
1	DATA+	Positive pole of absolute encoder data	
2	DATA-	Negative pole of absolute encoder data	
6	CLK+	Positive pole of absolute encoder clock	
14	CLK-	Negative pole of absolute encoder clock	
8	GND	Power ground of the absolute encoder	
15	5V	Absolute encoder power	COM/SSI

6.5.3 SSI Absolute Encoder Hardware Connection

- Illustration of SSI Absolute Encoder Wiring



- Specification for SSI Absolute Encoder Interface Wiring
  SSI encoder interface of DVP15MC11T and the wiring method are shown below.



Note: The power supply for COM/SSI port of DVP15MC11T is 5V power.

When VCC = 5V, connect the power voltage VCC of SSI encoder to pin 15 of COM/SSI interface and 0V of SSI encoder to pin 8 of COM/SSI interface.

When VCC ≠ 5V, the power is supplied to SSI encoder alone according to the actual power voltage of the SSI encoder which is connected.

- Specification for SSI Absolute Encoder Communication Cable

  Please use the shielded pair-twisted cable for CLK+, CLK-, DATA+ and DATA- signal transmission.

6.6 Incremental Encoders

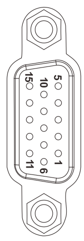
6.6.1 Function of Incremental Encoder

DVP15MC11T's incremental encoder port is a 15-pin D-SUB interface which can connect two independent incremental encoders. Both of the two encoder ports support differential signal input with maximum work frequency of 1MHz (250Kx 4 = 1MHz) per one. Additionally, the port integrates two 5V (400mA) power outputs to supply power to the two encoders. Users can create an incremental encoder axis for either of the two encoders to control the motion of slave axes according to the number of pulses received at the encoder port.

6.6.2 Definition of Incremental Encoder Port Pins

DVP15MC11T's incremental encoder port is a 15-pin interface. See the table below for definitions of respective encoder communication port pins.

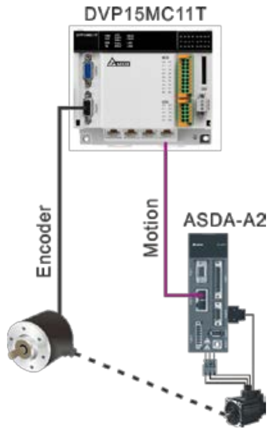
Pin No.	Signal	Definition
1	A1+	Differential signals of the first incremental encoder
2	A1-	
10	B1+	
11	B1-	
4	Z1+	
5	Z1-	
15	+5V	Power supply for the first encoder
3	A2+	Differential signals of the second incremental encoder
9	A2-	
6	B2+	
12	B2-	
13	Z2+	
14	Z2-	
7	+5V	Power supply for the second encoder
8	0V	0V shared by the two encoders
Outer metal shell		Shielding layer



Encoder

6.6.3 Incremental Encoder Hardware Connection

- Illustration of Incremental Encoder Wiring



- Specification for Incremental Encoder Port Wiring
  The incremental encoder interface of DVP15MC11T and the wiring method are shown below.

Pin No.	Function	Encoder Function
1	A1+	A
2	A1-	$\bar{A}$
10	B1+	B
11	B1-	$\bar{B}$
4	Z1+	Z
5	Z1-	$\bar{Z}$
15	+5V	Vcc
8	GND	0V

Note: The power supply for Encoder port of DVP15MC11T is 5V power.

When VCC = 5V, connect the power voltage VCC of an encoder to pin 15 of DVP15MC11T's Encoder interface and 0V of the encoder to pin 8 of Encoder interface.

When VCC ≠ 5V, the power is supplied to the encoder alone according to the actual power voltage of the encoder which is connected.

6.7 Ethernet Communication Port

6.7.1 Function that Ethernet Communication Port Supports

There are two independent Ethernet communication ports in DVP15MC11T, which both support Modbus TCP protocol. Of the two Ethernet ports, LAN1 port can only work as a slave and LAN2 port can work as a master or a slave in the Ethernet network. Either of them can accept a maximum of 4 master access at a time and their IP addresses need be set separately. HMI, PLC or other Modbus TCP master device can read and write data in the devices inside DVP15MC11T via the two Ethernet ports. For details on Modbus TCP communication, refer to appendix A.

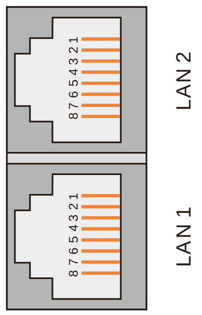
Both of the two Ethernet ports can be used to download configuration files, execution files and CAM files. They also support automatic jumper function and users do not need to additionally select wire jumper when the Ethernet port is connected to the computer or switchboard. Besides, they can automatically detect the transmission speed of 10Mbps and 100 Mbps as well.

The Ethernet communication port supports Ethernet/IP protocol, Ethernet/IP slaves only as well as maximum 200 bytes of input and maximum 200 bytes of output

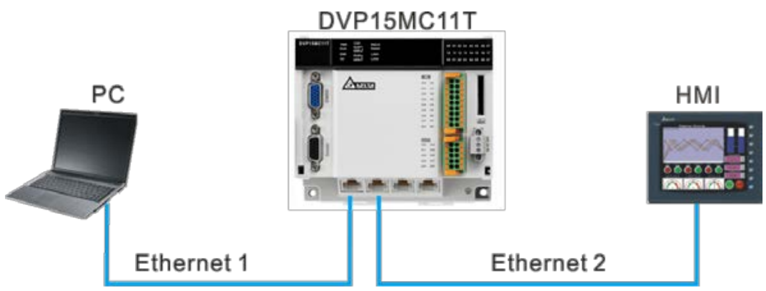
6.7.2 Pins of Ethernet Communication Port

DVP15MC11T has two independant Ethernet ports supporting Modbus TCP protocol with the pins shown in the following table. The IP addresses of the two Ethernet ports need be set respectively. The default IP address for LAN1 is 192.168.0.1 and the default IP address for LAN2 is 192.168.1.1.

Pin No.	Signal	Definition
1	Tx+	Positive pole for transmiting data
2	Tx-	Negative pole for transmitting data
3	Rx+	Positive pole for receiving data
4	Reserved	Reserved
5	Reserved	Reserved
6	Rx-	Negative pole for receiving data
7	Reserved	Reserved
8	Reserved	Reserved



6.7.3 Network Connection of Ethernet Communication Port



6.7.4 Function Codes that Ethernet Communication Port Supports

Below is the list of the function codes and exception response codes which are supported when DVP15MC11T's Ethernet communication ports LAN1 and LAN2 use Modbus TCP protocol.

Function code	Indication	Max. number of writable/readable registers	Available register
0x02	Read bit register values.	256	Bit register
0x03	Read one single or multiple word register values.	100	Word register
0x05	Write one single bit register value.	1	Bit register
0x06	Write one single word register value.	1	Word register
0x0F	Write multiple bit register values.	256	Bit register
0x10	Write multiple word register values.	100	Word register

Exception response code	Indication
0x01	Unsupportive function code
0x02	Unsupportive Modbus address
0x03	The data length is out of the valid range.

6.8 Motion Communication Port

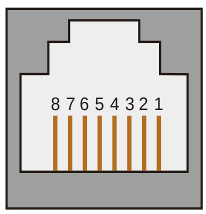
6.8.1 Function that Motion Communication Port Supports

Motion communication port is used for motion control. Motion instructions control a servo via the communication port. SDO command can be sent out through the communication port. But users can not carry out the PDO configuration through the communication port.

6.8.2 Pins of Motion Communication Port

The following table lists the pins of Motion communication port which is used for the motion control.

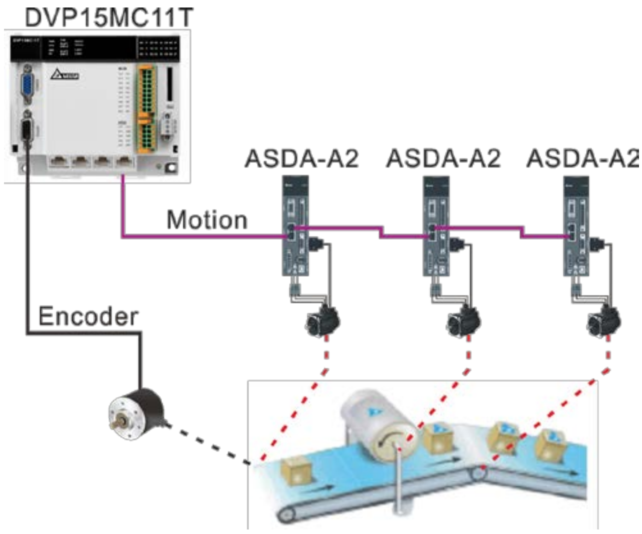
Pin No.	Signal	Definition
1	CAN_H	Signal+
2	CAN_L	Signal-
3	CAN_GND	0 VDC
4	Reserved	Reserved
5	Reserved	Reserved
6	Reserved	Reserved
7	CAN_GND	0 VDC
8	Reserved	Reserved



Motion

CAN2

6.8.3 Motion Network Connection



Note: DVP15MC11T is embedded with one 120 Ohm terminal resistor in its Motion interface.

6.8.4 Communication Speed and Communication Distance

The transmission distance of the bus network depends on the transmission speed of Motion bus. Below is the table where the maximum communication distances correspond to different transmission speeds.

Transmission speed (Bit/second)	500K	1M
Max. communication distance (Meter)	100	25

6.9 CANopen Communication Port

6.9.1 Functions that CANopen Communication Port Supports

CANopen communication port can be used as CANopen network master or as a slave of other master.

- As a master, CAN1 communication port supports following functions.
  - Standard CANopen protocol DS301V4.02;
  - NMT (Network Management Object) Master service;
  - NMT Error control;
    NMT error control is used to watch if some slave is offline. NMT error control includes Heartbeat and Node Guarding. The module supports Heartbeat function.
  - Connects max. 32 slaves.
  - PDO (Process Data Object) service.

    The number of RxPDOs: max. 200, data length: max. 1000 bytes

    The number of TxPDOs: max. 200, data length: max. 1000 bytes
    Maximum 8 TxPDOs and 8 RxPDOs are configured for each slave.
    PDO transmission type: supporting event trigger, time trigger, synchronous and cyclic, synchronous and acyclic
    PDO mapping: every PDO can map 32 parameters at most.
    The data type that CAN communication port supports

Storage capacity	Data type
1bit	BOOL
8bit	SINT, USINT,BYTE
16bit	INT, UINT, WORD
32bit	DINT, UDINT, REAL, DWORD
64bit	LINT, ULINT, LREAL, LWORD

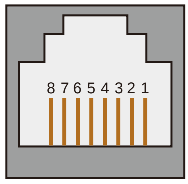
  - Supports SDO service
    Supports standard expedited SDO transmission mode;
    Supports Auto SDO function; capable of sending a maximum of 30 Auto SDOs to each slave;
    Supports reading and writing of slave data by using SDO service in PLC ladder diagram program.
  - SYNC producer, range 0-65535ms
    Multiple devices perform an action synchronously through SYNC message.
  - As the connection interface between Delta CANopen Builder configuration software and CANopen network, the configuration software can be directly used to configure the network through DVPCOPM-SL module
  - Supports the CANopen communication speeds: 20K, 50K, 125K, 250K, 500K, 1Mbps

- As a slave, CAN1 communication port supports following functions.
  - Standard CANopen protocol DS301V4.02
  - NMT slave service
  - NMT Error control
    Supporting Heartbeat Protocol error control instead of Node Guarding error control
  - PDO service
    The number of RxPDOs: max. 8, data length: max. 64 bytes
    The number of TxPDOs: max .8, data length: max. 64 bytes
  - PDO transmission type: event trigger, time trigger, synchronous and cyclic, synchronous and acyclic
  - SDO service
    Supporting standard expedited SDO transmission mode.

6.9.2 Pins of CANopen Communication Port

DVP15MC11T's CANopen communication port is used in the standard CANopen communication and its pin descriptions are listed in the following table.

Pin No.	Signal	Definition
1	CAN_H	Signal+
2	CAN_L	Signal-
3	CAN_GND	0 VDC
4	Reserved	Reserved
5	Reserved	Reserved
6	Reserved	Reserved
7	CAN_GND	0 VDC
8	Reserved	Reserved



CANopen
CAN1

6.9.3 PDO Mapping at CANopen Communication Port

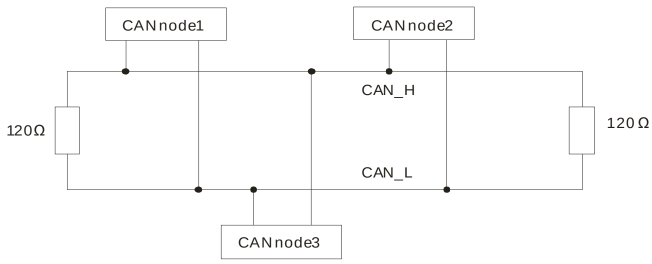
The input mapping area is %MW5000~%MW5499 and output mapping area is %MW5500~%MW5999 when DVP15MC11T works as CANopen master.

The input mapping area is %MW5000~%MW5031 and output mapping area is %MW5500~%MW5531 when DVP15MC11T works as CANopen slave.

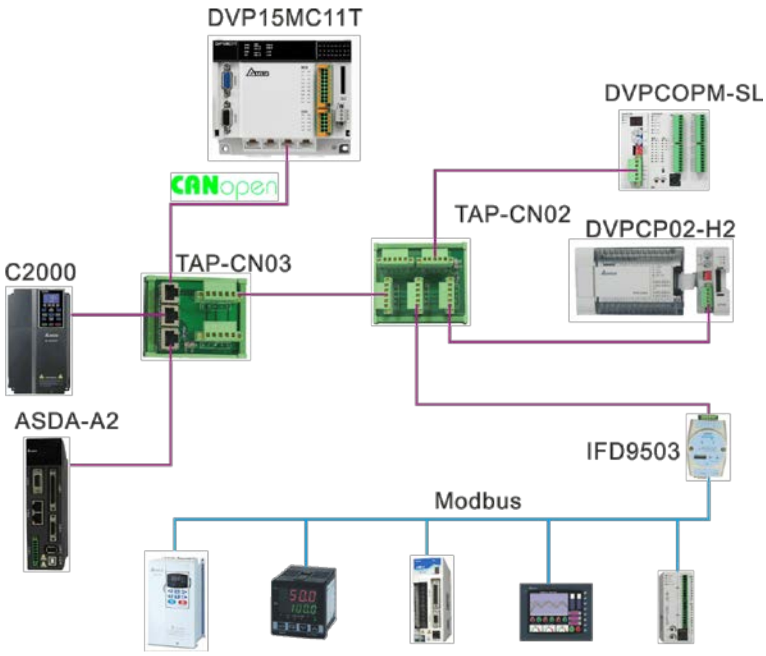
6.9.4 Network Connection at CANopen Communication Port

- **CANopen Bus Terminals and Network Topology**

Both of the two ends of a CANopen network need be connected with the terminal resistors of 120Ω to enhance the stability of CANopen communication. See the illustration of a basic CANopen network topology below.



- **CANopen Bus Network Topology**



1> Delta's standard cables such as UC-DN01Z-01A thick cable, UC-DN01Z-02A thin cable and UC-CMC010-01A thin cable are recommended to use in construction of a CANopen network. The communication cable must keep away from the power cable.

2> The terminal resistor of 120Ω should be connected between CAN_H and CAN_L of two respective ends of the network. Users can purchase Delta terminal resistor, TAP-TR01.

6

6.9.5 CANopen Communication Rate and Communication Distance

The transmission distance of CANopen bus network depends on the transmission speed of CANopen bus. Below is the table where the maximum communication distances correspond to different transmission speeds.

Transmission speed (Bit/second)	20K	50K	125K	250K	500K	1M
Max. communication distance (Meter)	2500	1000	500	250	100	25

7

Chapter 7 Execution Principle of DVP15MC11T Controller

Table of Contents

7.1 Tasks.. 7-2
7.1.1 Task Types 7-2
7.1.2 Priority levels of Tasks.. 7-4
7.1.3 Watchdog for a Task 7-6
7.1.4 Motion Instructions for Each Task Type 7-7

7.2 The Impact of PLC RUN or STOP on Variables and Devices.............. 7-8

7.3 Relationship between Motion Program and Motion Bus.................. 7-8

7.4 Synchronization Cycle Period Setting 7-9

7.1 Tasks

- Tasks are a series of functions of processing specified execution conditions and execution sequences for I/O refresh and user program execution.
- A task is defined with a name, priority level and type. Tasks can be classified into three types, the cyclic task, freewheeling task and event-triggered task.
- For every task, a group of POUs which are triggered by the task can be specified. If the task is executed in current period, the POUs will be processed within a period of time.
- The priority level and task type determine the execution sequence of the task.
- A watchdog can be assigned for every task.

7.1.1 Task Types

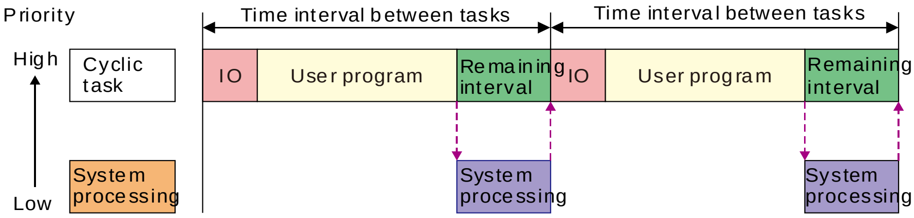
- **Three task types that DVP15MC11T supports**
  1. Cyclic
  2. Freewheeling
  3. Triggered by event

- **Maximum 24 tasks that DVP15MC11T supports are respectively described below.**

  - **Cyclic task**

    The cyclic task will be executed cyclically according to the set time interval.

    - **The way the cyclic task is executed**



IO: **IO** means I/O refresh. I/O includes local I/O points and left-side and right-side extension module data and CANopen data. The data can be specified to refresh before the set task is executed. If not specified, the data will be refreshed during the system processing.

User Program: User Program stands for user program execution which is based on the execution sequences of programs assigned in a task.

Remaining interval:

When the controller is to perform system processing, the low-priority task is executed first if any and then the system processing is performed.

System processing:

The controller will perform the system processing which includes Ethernet, RS232 and RS485 communication processing after all task requests are completed.

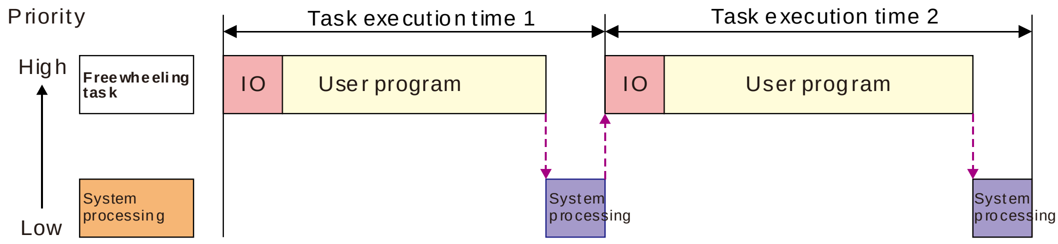
The four terms mentioned above have the same meanings as those in the following sections.

Note: If the cycle set for a cyclic task is too short, after the user program execution is finished, the task execution will be repeated immediately and no low-priority task or no system processing will be executed. In this case, the execution of all tasks will be affected. If the watchdog is set for the task, the watchdog timeout will occur, the controller will enter Error status and user program execution will stop. If the watchdog is not set for the task, the controller will not be able to perform system processing and the problems such as communication timeout will take place.

■ **Freewheeling task**

Freewheeling task: The task will be handled as soon as the program running starts. The task will be restarted automatically in the next cycle after one execution cycle ends.

➢ **The way a freewheeling task is executed**

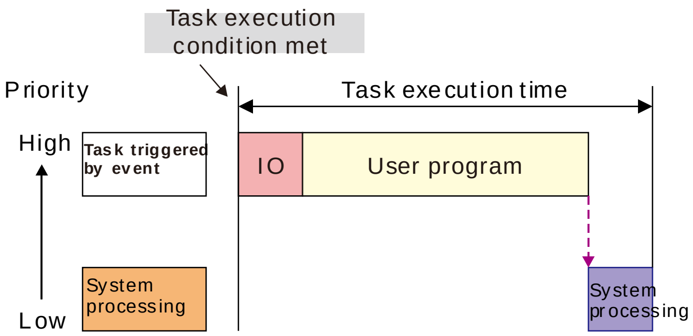


Note: There is no fixed execution time for the freewheeling task. So the values of task execution time 1 and task execution time 2 may not be equal in the above figure.

■ **Task triggered by event**

Event task: An event task is executed once just when the specified event happens. The timing for execution of an event task depends on the timing for occurring of the event and the priority level of the event task.

➢ **The way an event task is executed**



➢ **The event tasks for option contain following few types.**

- Motion event (Motion control task)
- Rising edge or falling edge of local input points (I0~I7 and I10~I17)
- CANopen SYNC signal
- Z pulse rising edge of incremental encoder 1 or encoder 2

The condition for the second-time execution is ignored when the condition required for execution of the event task is met again before the event task is completed. The period before an event task is completed is the course while the event task is being executed or is waiting to be executed.

● **Motion Event**

Motion port of the controller sends out SYNC signal and the task is triggered.

Note: The motion task is set to priority 1 by default. The priority level can be modified. However, make sure that there is enough time for execution of the motion task within CANopen SYNC period.

■ **SYNC cycle setting should meet following conditions.**

➢ There must be enough time for execution of the program defined in a motion task.

➢ There must be sufficient time for PDO and SDO data exchange between the controller and servo drive.

Insufficient SYNC period time will result in the controlled device to fail to receive SYNC signal and unpredictable operations. Refer to section 7.3 for SYNC period setting.

- **Rising edge or falling edge of local input points (I0~I7、I10~I17)**

  The task is triggered when rising edge or falling edge of input point signal is detected. The response time of input points can be set through the filter function.

- **CANopen bus SYNC message**

  The task is triggered when SYNC signal is produced at CANopen port of the controller.

- **Z pulse rising edge for incremental encoder 1**

  The task is triggered when the rising edge of Z signal of the first encoder is detected at Encoder port of the controller.

- **Z pulse rising edge for incremental encoder 2**

  The task is triggered when the rising edge of Z signal of the second encoder is detected at Encoder port of the controller.

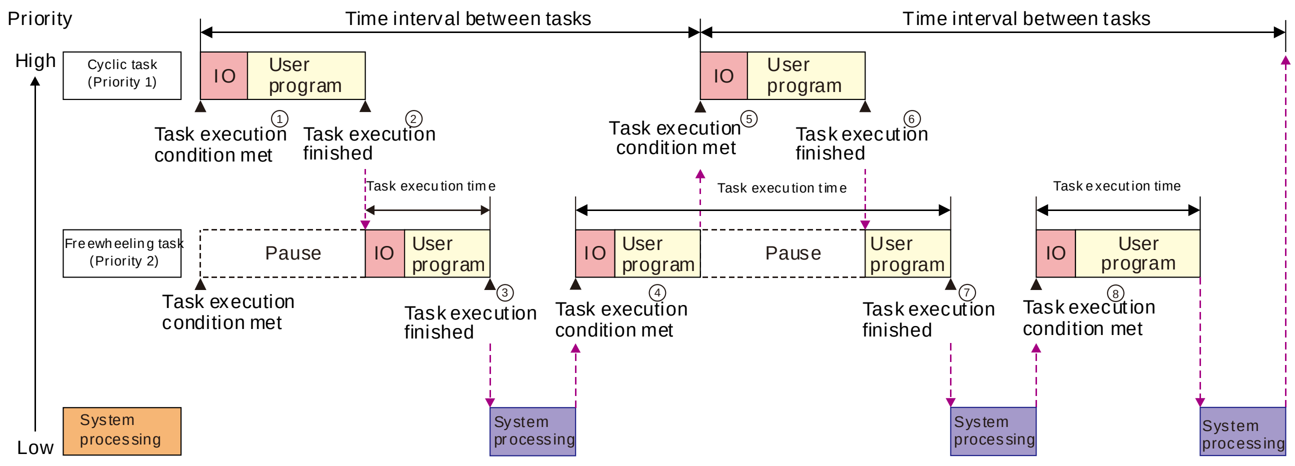
7.1.2 Priority levels of Tasks

The controller can not perform multiple tasks simultaneously. Every task must be given a priority level and they are executed according to preset priorities. Priority level can be set within the range of 1 to 24. (1 is the highest priority and 24 is the lowest priority.) The priority level of each task must be unique. The task with higher priority takes priority to perform. The high-priority task can interrupt the low-priority task.

We recommend that the task which has a high requirement of real time should be given a high priority and the task which has a low requirement of real time should be given a low priority. The priority of the default motion control task built in the CANopen Builder software is 1 by default.

- **The principle for multi-task execution**

■ **When the execution conditions of two tasks are met simultaneously (Cyclic task and freewheeling task)**



① The execution conditions for the cyclic task and freewheeling task are met at the same time. The cyclic task is executed first because of its higher priority.

② When the cyclic task execution is finished, the freewheeling task execution starts.

③ The controller will execute the system processing if there is no other task after the execution of the freewheeling task is completed.

④ The execution of the freewheeling task continues since the high-priority cyclic task request has not arrived.

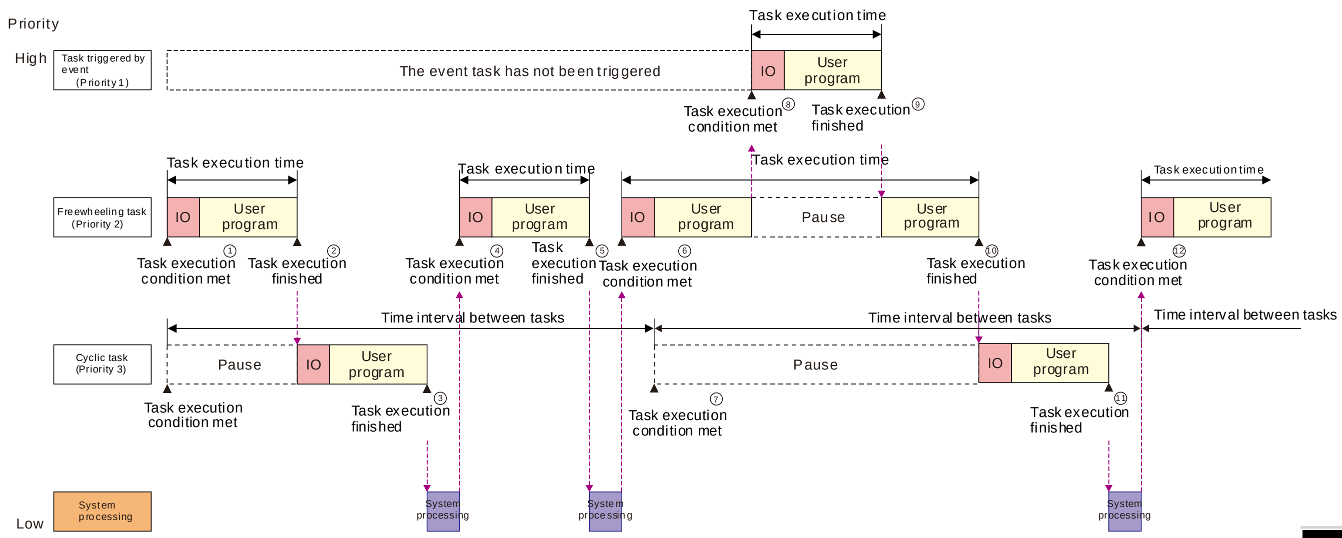
⑤ The cyclic task interrupts the freewheeling task execution and the controller executes the cyclic task because of the arrival of the high-priority cyclic task request during the execution of the freewheeling task.

⑥ The controller continues to execute the part of the low-priority freewheeling task, which has not been executed yet when the execution of the cyclic task is completed.

⑦ When the execution of the freewheeling task is completed, the controller executes the system processing due to no other task request.

⑧ When the system processing is completed, the execution of the freewheeling task continues due to no high-priority cyclic task request.

■ **When three tasks are executed in mixture (Event task, Cyclic task and Freewheeling task)**



① When the conditions for execution of the freewheeling task and cyclic task are both met, the freewheeling task is executed first because the priority of the freewheeling task is higher.

② The cyclic task execution starts when the freewheeling task execution is completed.

③ When the cyclic task execution is completed, the controller executes the system processing due to no other task request.

④ The freewheeling task is executed when the system processing is completed.

⑤ When the freewheeling task execution is completed, the controller executes the system processing due to no other task request.

⑥ The freewheeling task is executed when the system processing is completed.

⑦ The freewheeling task execution continues because the freewheeling task has a higher priority than the cyclic task although the execution condition for the cyclic task is met. And the cyclic task waits to execute.

⑧ The event task interrupts the freewheeling task execution because the event task has the highest priority and the execution condition for the event task is met.

⑨ The controller continues to execute the part of the low-priority freewheeling task, which has not been executed yet when the event task execution is completed.

⑩ The freewheeling task execution is completed. The controller executes the cyclic task since the cyclic task request in ⑦ is not responded yet.

⑪ The cyclic task execution is completed. The controller executes the system processing due to no other task request.

7.1.3 Watchdog for a Task

Every task can be given a watchdog. When the task execution time exceeds the set watchdog time, the controller will enter Error state and the user program execution will stop.

***Watchdog time*:** The longest time allowed for the execution of a task

7.1.4 Motion Instructions for Each Task Type

Here is the table of motion instructions for different task types. “V” means the motion instruction can be executed for the task type and “–” means the motion instruction can not be executed for the task type.

Classification	Instruction name	Task type			
		Cyclic task	Freewheeling task	Event-triggered task	
				Motion task	Non-motion task
Single-axis instructions	MC_Power			V	
	MC_MoveAbsolute			V	
	MC_MoveRelative			V	
	MC_MoveAdditive			V	
	MC_MoveSuperimposed			V	
	MC_Haltsuperimposed			V	
	MC_MoveVelocity			V	
	MC_Stop			V	
	MC_Halt			V	
	MC_Home			V	
	MC_Reset			V	
	MC_ReadStatus	V	V	V	V
	MC_ReadActualPosition	V	V	V	V
	MC_SetOverride			V	
	MC_SetPosition			V	
	MC_ReadAxisError	V	V	V	V
	MC_ReadMotionState	V	V	V	V
	DMC_SetTorque			V	
	DMC_ReadParameter_Motion	V	V	V	V
	DMC_WriteParameter_Motion			V	
	DMC_TouchProbe			V	
Multi-axis instructions	MC_CombineAxes			V	
	MC_GearIn			V	
	MC_GearOut			V	
	MC_CamIn			V	
	MC_CamOut			V	
CANopen	DMC_WriteParameter_CANopen	V	V	V	V
	DMC_ReadParameter_CANopen	V	V	V	V

7

7.2 The Impact of PLC RUN or STOP on Variables and Devices

When DVP15MC11T is switched from RUN to STOP, variables and devices keep current values. When DVP15MC11T is switched from STOP to RUN, users can select one option that the values of variables and non-latched devices are cleared or retained as below.

- **The values of variables and non-latched devices are cleared.**

  When DVP15MC11T is switched from STOP to RUN, the values of variables and non-latched devices are cleared and restored to the initial values. If variables and non-latched devices have no initial values, the values of variables and non-latched areas will be restored to the default value 0.

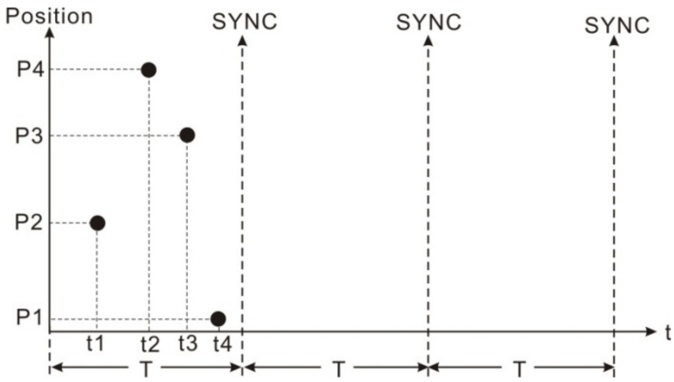
- **The values of variables and devices are retained.**

  When DVP15MC11T is switched from STOP to RUN, variables and devices keep current values.

7.3 Relationship between Motion Program and Motion Bus

DVP15MC11T makes the synchronization achieved through issuing SYNC signal in the method of broadcasting while more than one servo is connected with DVP15MC11T. The servo drives receive the control data sent by DVP15MC11T. But the control data received will not be effective right away until the SYNC signal comes to the servos so as to realize the synchronization of multiple servos.

In the following figure, DVP15MC11T is connected with 4 servo drives and T is the synchronization period. The four servo drives receive control data at different time (t1, t2, t3 and t4) but the control data received are not effective at once. As the servo drives receive SYNC signal, the control data will go effective immediately.



7

7.4 Synchronization Cycle Period Setting

The synchronization cycle is a very important parameter for the bus motion control. If the synchronization period is not set properly, the servo may display AL303/AL302/AL301 fault alarm in communication or the servo could not run normally.

Let's introduce the constitution of the synchronization period first.

The motion control program is scanned at the very beginning of the synchronization period, and then the control messages got through calculation are sent to all axes. So we can regard the synchronization period as the time for execution of motion control program plus the time for communication between DVP15MC and all servos.

The time for execution of motion control program is the maximum execution time of motion event tasks with the unit: μs (microsecond) which can be viewed by double clicks on **Task** on the CANopen Builder software interface. 1000μs (microseconds) are 1ms (millisecond).

The value is rounded up to an integer in the actual application. For example, the maximum time for program execution is 2567μs=2.5ms, in this case, we can regard 3ms as the time for program execution.
It is about 0.5ms for the communication between DVP15MC and a servo.

We recommend that the value is rounded up to an integer in application. For example, 5 servos are configured in an application. And the communication time is 5*0.5ms=2.5ms. In this case, we can regard 3ms as the time for communication.

Therefore, we can get the formula: a synchronization time (ms) = an integer obtained by rounding up the value of maximum program execution time (ms) + time for the communication between DVP15MC11T and all servos (ms) +1 (time reserved for a program change) (ms).

If the running time of the program is increased too much after the program changes, the preset synchronization time will not fit any more. So the reserved time should be set to 1~2ms.

For example, the maximum program execution time is 1634μs and there are totally 5 servos in the application. The reserved time for a program change is 1ms.

A synchronization cycle period= 2ms (obtained by rounding up the maximum program execution time, 1634μs) + 3ms (obtained by rounding up 5*0.5) +1ms (reserved for a program change)=6ms

Note:

The above method is used for getting an estimated time, which is suitable for most applications. If you need a more precise synchronization cycle period, the actual time can be recalculated by omitting the reserved time after the application development is completed.

Memo

8

Chapter 8 Logic Instructions

Table of Contents

8.1 Table of Logic Instructions 8-4

8.2 Explanation of Logic Instructions 8-7
8.2.1 EN and ENO 8-7

8.3 Sequence Input / Output Instructions 8-7
8.3.1 R_TRIG 8-7
8.3.2 F_TRIG 8-9
8.3.3 RS 8-11
8.3.4 SR 8-13
8.3.5 SEMA 8-15

8.4 Data Movement Instructions 8-17
8.4.1 MOVE 8-17
8.4.2 MoveBit 8-18
8.4.3 TransBit 8-20
8.4.4 MoveDigit 8-22
8.4.5 Exchange 8-24
8.4.6 Swap 8-26

8.5 Comparison Instructions 8-28
8.5.1 LT 8-28
8.5.2 LE 8-30
8.5.3 GT 8-32
8.5.4 GE 8-34
8.5.5 EQ 8-36
8.5.6 NE 8-38

8.6 Timer Instructions 8-40
8.6.1 TON 8-40
8.6.2 TOF 8-42
8.6.3 TP 8-44

8.7 Counter Instructions 8-46
8.7.1 CTU 8-46
8.7.2 CTD 8-48
8.7.3 CTUD 8-50

8.8 Math Instructions 8-53
8.8.1 ADD 8-53
8.8.2 SUB 8-56
8.8.3 MUL 8-59
8.8.4 DIV 8-62

8.8.5 MOD 8-65
8.8.6 MODREAL 8-67
8.8.7 MODTURNS 8-69
8.8.8 MODABS 8-71
8.8.9 ABS 8-73
8.8.10 DegToRad 8-75
8.8.11 RadToDeg 8-77
8.8.12 SIN 8-79
8.8.13 COS 8-81
8.8.14 TAN 8-83
8.8.15 ASIN 8-85
8.8.16 ACOS 8-88
8.8.17 ATAN 8-90
8.8.18 LN 8-92
8.8.19 LOG 8-94
8.8.20 SQRT 8-96
8.8.21 EXP 8-98
8.8.22 EXPT 8-100
8.8.23 RAND 8-102
8.8.24 TRUNC 8-104
8.8.25 FLOOR 8-106
8.8.26 FRACTION 8-108

8.9 Bit String Instructions 8-110
8.9.1 AND 8-110
8.9.2 OR 8-113
8.9.3 NOT 8-116
8.9.4 XOR 8-118
8.9.5 XORN 8-121

8.10 Shift Instructions 8-124
8.10.1 SHL 8-124
8.10.2 SHR 8-126
8.10.3 ROL 8-128
8.10.4 ROR 8-130

8.11 Selection Instructions 8-132
8.11.1 MAX 8-132
8.11.2 MIN 8-134
8.11.3 SEL 8-136
8.11.4 MUX 8-138
8.11.5 LIMIT 8-140
8.11.6 BAND 8-143
8.11.7 ZONE 8-146

8.12 Data Type Conversion Instructions 8-149
8.12.1 BOOL_TO_*** 8-149
8.12.2 Bit strings_TO_*** 8-152
8.12.3 Integers_TO_*** 8-159
8.12.4 Real numbers_TO_*** 8-168
8.12.5 Times,dates_TO_*** 8-171
8.12.6 Strings_TO_*** 8-173

8.13 CANopen Communication Instructions 8-176
8.13.1 DMC_ReadParameter_CANopen 8-176
8.13.2 DMC_WriteParameter_CANopen 8-182

8.14 String Processing Instructions 8-187
8.14.1 CONCAT 8-187
8.14.2 DELETE 8-189
8.14.3 INSERT 8-191
8.14.4 LEFT / RIGHT 8-193
8.14.5 MID 8-195
8.14.6 REPLACE 8-197
8.14.7 LEN 8-199
8.14.8 FIND 8-200

8.15 Immediate Refresh Instructions 8-202
8.15.1 FROM 8-202
8.15.2 TO 8-206
8.15.3 ImmediateInput 8-210
8.15.4 ImmediateOutput 8-212

8.1 Table of Logic Instructions

Instruction set	Instruction code	Name
Sequence Input/Output Instructions	R_TRIG	Rising Edge Trigger
	F_TRIG	Falling Edge Trigger
	RS	Reset–Priority Instruction
	SR	SET–Priority Instruction
	SEMA	Claim-Priority Instruction
Data Movement Instructions	MOVE	Move
	MoveBit	Move One Bit
	TransBit	Move Bits
	MoveDigit	Move Digits
	Exchange	Data Exchange
	Swap	Swap Bytes
Comparison Instructions	LT	Less Than
	LE	Less Than or Equal
	GT	Greater Than
	GE	Greater Than or Equal
	EQ	Equal
	NE	Not Equal
Timer Instructions	TON	On-Delay Timer
	TOF	Off-Delay Timer
	TP	Pulse-type Timer
Counter Instructions	CTU	Up-Counter
	CTD	Down-Counter
	CTUD	Up-Down Counter
Math Instructions	ADD	Addition
	SUB	Subtraction
	MUL	Multiplication
	DIV	Division
	MOD	Integer Modulo Division to Get the Remainder
	MODREAL	Real-Number Modulo Division to Get the Remainder
	MODTURNS	Real-Number Modulo Division to Get Signed Integral Part

Instruction set	Instruction code	Name
	MODABS	Real-Number Modulo Division to Get the Unsigned Modulo Value
	ABS	Absolute value
	DegToRad	Degrees to Radians
	RadToDeg	Radians to Degrees
	SIN	Sine
	COS	Cosine
	TAN	Tangent
	ASIN	Arc sine
	ACOS	Arc cosine
	ATAN	Arc tangent
	LN	Natural Logarithm
	LOG	Base-10 Logarithm
	SQRT	Square Root
	EXP	Natural Exponential Operation
	EXPT	Exponentiation
	RAND	Random Number
	TRUNC	Truncate
	FLOOR	Real-Number Floor
	FRACTION	Real-Number Fraction
Bit String Instructions	AND	Logical AND
	OR	Logical OR
	NOT	Bit Reversal
	XOR	Logical Exclusive OR
	XORN	Logical Exclusive NOR
Shift Instructions	SHL	Shift Bits Left
	SHR	Shift Bits Right
	ROL	Rotate Bits Left
	ROR	Rotate Bits Right
Selection Instructions	MAX	Maximum
	MIN	Minimum
	SEL	Selection
	MUX	Multiplexer

Instruction set	Instruction code	Name
	LIMIT	Limiter
	BAND	Deadband Control
	ZONE	Dead Zone Control
Data Type Conversion Instructions	BOOL_TO_***	Bool Conversion Group
	Bit strings_TO_***	Bit String Conversion Group
	Integers_TO_***	Integer Conversion Group
	Real numbers_TO_***	Real Number Conversion Group
	Times,dates_TO_***	Time and Data Conversion Group
	Text strings_TO_***	String Conversion Group
CANopen Communication Instructions	DMC_ReadParameter_CANopen	Read a slave parameter value
	DMC_WriteParameter_CANopen	Write a slave parameter value
String Processing Instructions	CONCAT	Concatenate String
	DELETE	Delete String
	INSERT	Insert String
	LEFT / RIGHT	Get String Left/Right
	MID	Get String
	REPLACE	Replace String
	LEN	String Length
	FIND	Find String
Immediate Refresh Instructions	FROM	Read CR value
	TO	Write Value to CR
	ImmediateInput	Immediate Refresh of Input Points
	ImmediateOutput	Immediate Refresh of Output Points

8.2 Explanation of Logic Instructions

8.2.1 EN and ENO

If the used instruction has EN and ENO input parameters and the value of EN is FALSE (0), the function of the instruction will not be performed and the output of the instruction will not be updated. However, if the value of EN of the instruction is TRUE (1), the function of the instruction will be performed and the output will be updated.

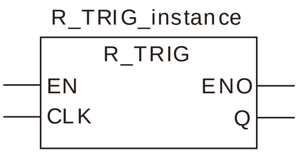
The output state of ENO is consistent with that of EN. When EN is TRUE, ENO changes to TRUE. When EN is FALSE, ENO changes to FALSE.

When the instruction is a function block (FB) and its EN changes from TRUE to FALSE after the FB instruction is executed, the execution of the FB instruction will continue, but the output values of the FB instruction will not be updated.

8.3 Sequence Input / Output Instructions

8.3.1 R_TRIG

FB/FC	Explanation	Applicable model
FB	R_TRIG is used for the rising edge trigger.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
CLK	Input signal	Input	Rising edge trigger signal	TRUE or FALSE
Q	Output signal	Output	Output for a period	TRUE or FALSE

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CLK	●																			
Q	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  When CLK of R_TRIG changes from FALSE to TRUE, Q output is TRUE for only one period. In other circumstances, Q is FALSE.

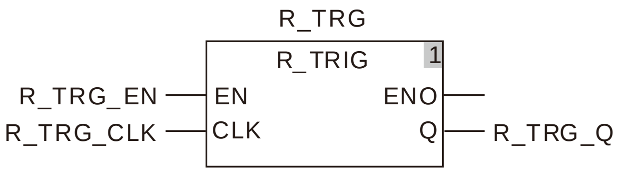
- **Precautions for Correct Use**

  Q will have no output until the rising edge signal at CLK is detected.

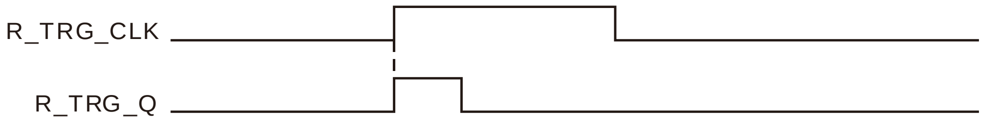
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
R_TRG	R_TRIG	
R_TRG_EN	BOOL	FALSE
R_TRG_CLK	BOOL	FALSE
R_TRG_Q	BOOL	

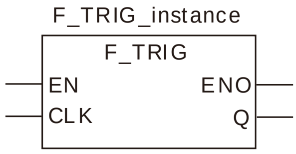


- **Timing Chart:**



8.3.2 F_TRIG

FB/FC	Explanation	Applicable model
FB	F_TRIG is used for the falling edge trigger.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
CLK	Input signal	Input	Falling edge trigger signal	TRUE or FALSE
Q	Output signal	Output	Output for a period	TRUE or FALSE

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CLK	●																			
Q	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  When CLK of F_TRIG changes from TRUE to FALSE, Q output is TRUE for only one period. In other circumstances, Q is FALSE.

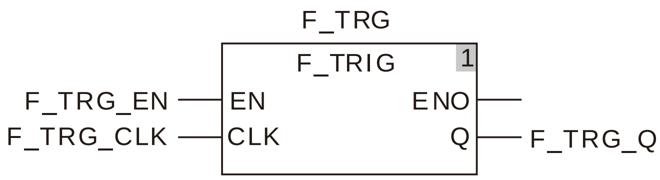
- **Precautions for Correct Use**

  Q will have no output until the falling edge signal at CLK is detected.

Programming Example

■ **The variable table and program**

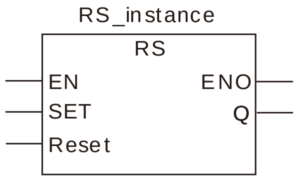
Variable name	Data type	Initial value
F_TRG	F_TRIG	
F_TRG_EN	BOOL	FALSE
F_TRG_CLK	BOOL	FALSE
F_TRG_Q	BOOL	



- **Timing Chart:**

8.3.3 RS

FB/FC	Explanation	Applicable model
FB	RS is used for giving priority to the *Reset* input.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
SET	Input signal	Input	SET signal	TRUE or FALSE
Reset	Input signal	Input	Reset signal	TRUE or FALSE
Q	Output signal	Output	Output signal	TRUE or FALSE

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SET	●																			
Reset	●																			
Q	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

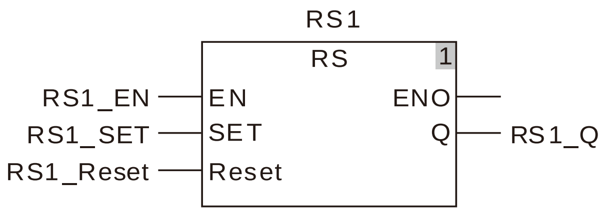
- **Function Explanation**

When the *SET* and *Reset* inputs of RS are both TRUE, *Reset* is given the priority.

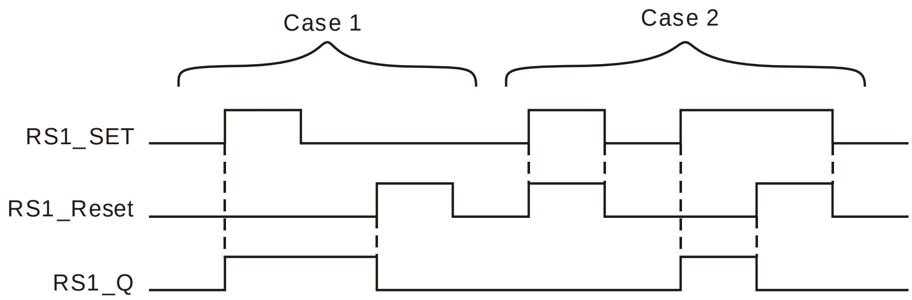
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
RS1	RS	
RS1_EN	BOOL	FALSE
RS1_SET	BOOL	FALSE
RS1_Reset	BOOL	FALSE
RS1_Q	BOOL	



- **Timing Chart:**



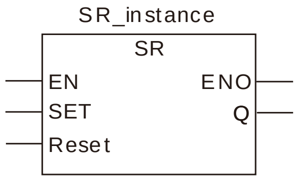
Case 1： When RS1_SET is TRUE, the output RS1_Q is TRUE. If RS1_Reset is TRUE, RS1_Q is FALSE.

Case 2： When RS1_Reset is TRUE, RS1_Q is always FALSE.

8

8.3.4 SR

FB/FC	Explanation	Applicable model
FB	SR is used for giving priority to the *Set* input.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
SET	Input signal	Input	SET signal	TRUE or FALSE
Reset	Input signal	Input	Reset signal	TRUE or FALSE
Q	Output signal	Output	Output signal	TRUE or FALSE

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
SET	●																			
Reset	●																			
Q	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

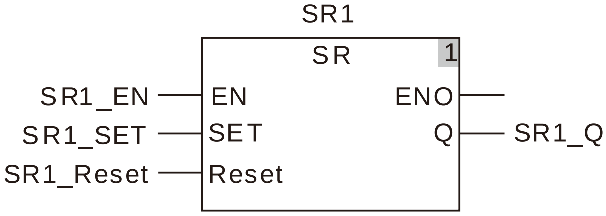
  When the *SET* and *Reset* inputs of RS are both TRUE, *SET* is given the priority.

8

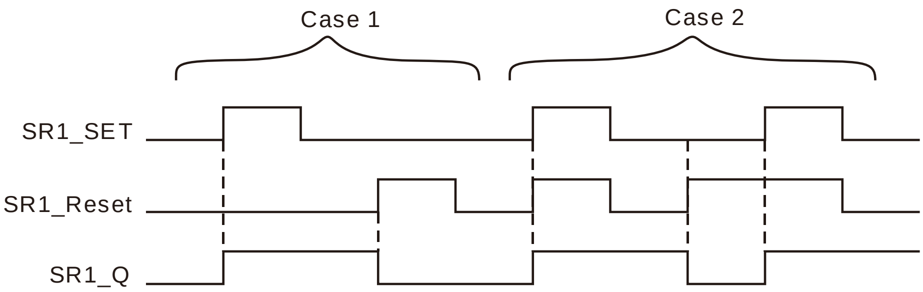
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
SR1	SR	
SR1_EN	BOOL	FALSE
SR1_SET	BOOL	FALSE
SR1_Reset	BOOL	FALSE
SR1_Q	BOOL	



- **Timing Chart:**

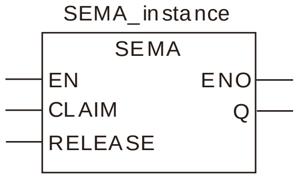


Case 1：When SR1_SET is TRUE, SR1_Q is TRUE. When SR1_Reset is TRUE, SR1_Q is FALSE.

Case 2：SR1_SET is given the priority when SR1_SET and SR1_Reset are both TRUE.

8.3.5 SEMA

FB/FC	Explanation	Applicable model
FB	SEMA is used for giving priority to CLAIM. (The output will be valid in the second period.)	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
CLAIM	Input signal	Input	Set signal	TRUE or FALSE
RELEASE	Input signal	Input	Reset signal	TRUE or FALSE
Q	Output signal	Output	Output signal	TRUE or FALSE

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CLAIM	●																			
RELEASE	●																			
Q	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  When *CLAIM* of SEMA is TRUE, *Q* is TRUE. When *RELEASE* is TRUE, *Q* is FALSE. When *CLAIM* and *RELEASE* are both TRUE, *Q* is TRUE.

- **Precautions for Correct Use**

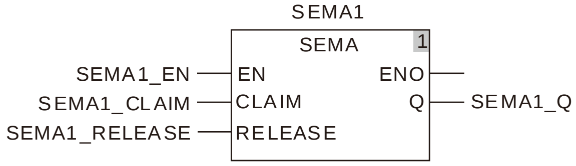
  When *CLAIM* is TRUE, Q will be TRUE in the second period.

Programming Example

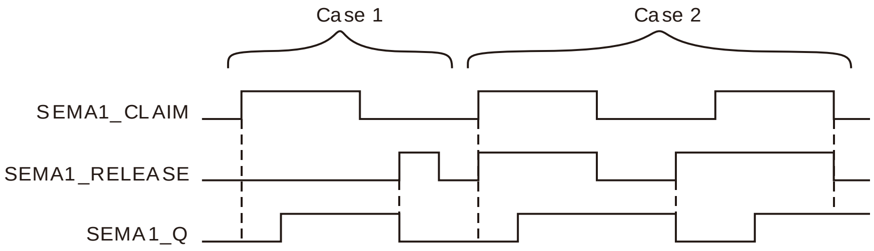
- **The variable table and program**

Variable name	Data type	Initial value
SEMA1	SEMA	
SEMA1_EN	BOOL	FALSE
SEMA1_CLAIM	BOOL	FALSE
SEMA1_RELEASE	BOOL	FALSE

Variable name	Data type	Initial value
SEMA1_Q	BOOL	



■ **Timing Chart:**



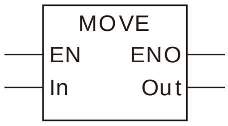
Case 1： When SEMA1_CLAIM is TRUE, SEMA1_Q is TRUE in the second period. When SEMA1_RELEASE is TRUE, SEMA1_Q changes to FALSE immediately.

Case 2： When SEMA1_CLAIM is TRUE, SEMA1_Q is TRUE in the second period no matter whether SEMA1_RELEASE is TRUE or FALSE.

8.4 Data Movement Instructions

8.4.1 MOVE

FB/FC	Explanation	Applicable model
FC	Move is used for moving data.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Input signal	Input	Move Source	Depends on the data type of the variable that the input parameter is connected to.
Out	Output signal	Output	Move destination	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The Move instruction moves the value of move source *In* to move destination *Out*.
  - The instruction supports the transmission of the values of array elements.

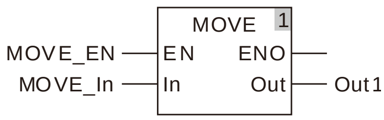
- **Precautions for Correct Use**

  The data type of *Out* must be the same as that of *In*. Otherwise, an error will occur in the compiling of the software.

Programming Example

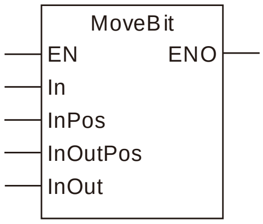
- **The variable table and program**

Variable name	Data type	Current value
MOVE_EN	BOOL	TRUE
MOVE_In	INT	200
Out1	INT	200



8.4.2 MoveBit

FB/FC	Explanation	Applicable model
FC	MoveBit is used for sending one bit in a string.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Input signal	Input	Move source	Depends on the data type of the variable that the input parameter is connected to.
InPos	Input signal	Input	Move source bit	Depends on the data type of the variable that the input parameter is connected to.
InOutPos	Input signal	Input	Move destination bit	Depends on the data type of the variable that the input parameter is connected to.
InOut	Input signal	Input	Move destination	Depends on the data type of the variable that the input parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
InPos							●													
InOutPos							●													
InOut		●	●	●	●	●	●	●	●											

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

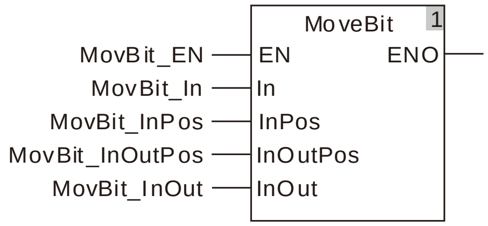
  MoveBit moves one bit value from the bit position *InPos* in move source *In* to the bit position *InOutPos* in move destination *InOut*.

- **Precautions for Correct Use**
  - The instruction has no ouput but input.
  - If the value of *InPos* exceeds the range of the data type of *In*, the movement of one bit is not performed.
  - If the value of *InOutPos* exceeds the range of the data type of *InOut*, the movement of one bit is not performed.

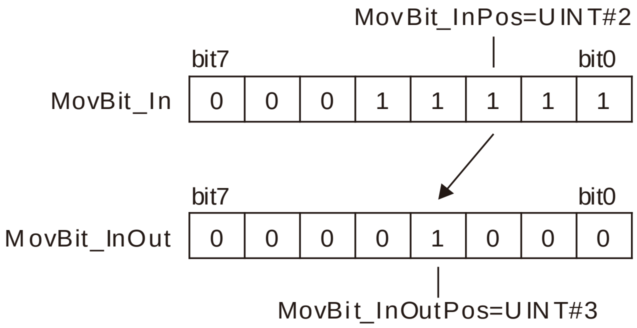
Programming Example

■ **The variable table and program**

Variable name	Data type	Current value
MovBit_EN	BOOL	TRUE
MovBit_In	USINT	31
MovBit_Inpos	UINT	2
MovBit_InOutPos	UINT	3
MovBit_Inout	USINT	8

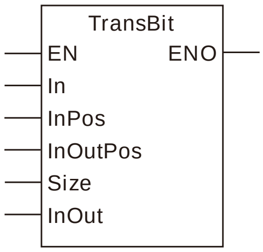


■ **Move Figure**



8.4.3 TransBit

FB/FC	Explanation	Applicable model
FC	TransBit is used for sending one or more bits in a bit string.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Input signal	Input	Move source	Depends on the data type of the variable that the input parameter is connected to.
InPos	Input signal	Input	Move source bit	Depends on the data type of the variable that the input parameter is connected to.
InOutPos	Input signal	Input	Move destination bit	Depends on the data type of the variable that the input parameter is connected to.
Size	Input signal	Input	Number of bits to move	Depends on the data type of the variable that the input parameter is connected to.
InOut	Input signal	Input	Move destination	Depends on the data type of the variable that the input parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
InPos							●													
InOutPos							●													
Size							●													
InOut		●	●	●	●	●	●	●	●											

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  TransBit moves data of *Size* bits from the bit *InPos* in move source *In* to the bit *InOutPos* in move destination *InOut*.

- **Precautions for Correct Use**
  - The instruction has no output but input.
  - The movement can not be performed if the value of *Size* is 0.

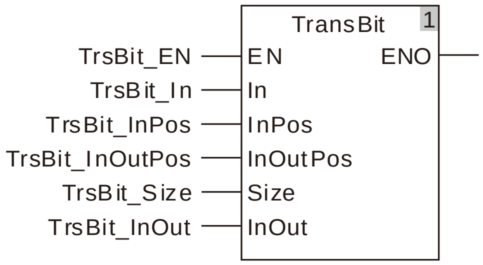
8

- If the value of *InPos* exceeds the range of the data type of *In*, the movement is not performed.
- If the value of *InOutPos* exceeds the range of the data type of *InOut*, the movement is not performed.
- If the value of *Size* exceeds the range, the movement is not performed.

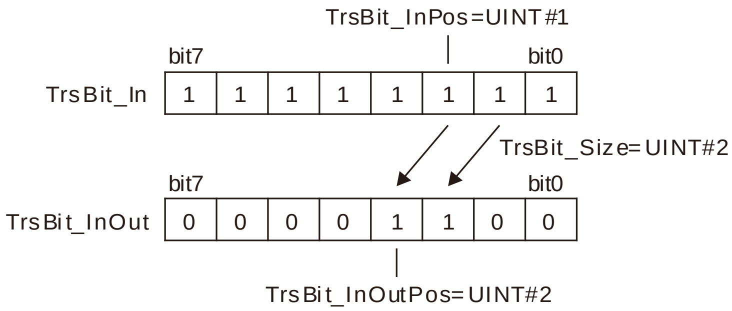
Programming Example

- **The variable table and program**

Variable name	Data type	Current value
TrsBit_EN	BOOL	TRUE
TrsBit_In	USINT	63
TrsBit_InPos	UINT	1
TrsBit_InOutPos	UINT	2
TrsBit_Size	UINT	2
TrsBit_Inout	USINT	12

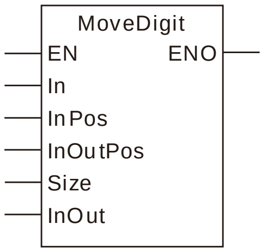


- **Move Figure**



8.4.4 MoveDigit

FB/FC	Explanation	Applicable model
FC	MoveDigit is used for moving digits.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Input signal	Input	Move source	Depends on the data type of the variable that the input parameter is connected to.
InPos	Input signal	Input	Position of digit in *In* to move	Depends on the data type of the variable that the input parameter is connected to.
InOutPos	Input signal	Input	Position of digit in *Out* to receive the digit	Depends on the data type of the variable that the input parameter is connected to.
Size	Input signal	Input	Number of digits to move	Depends on the data type of the variable that the input parameter is connected to.
InOut	Input signal	Input	Move destination	Depends on the data type of the variable that the input parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
InPos							●													
InOutPos							●													
Size							●													
InOut		●	●	●	●	●	●	●	●											

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  MoveDigit moves *Size* digits from *InPos* of move source *In* to *InOutPos* of move destination *InOut*.

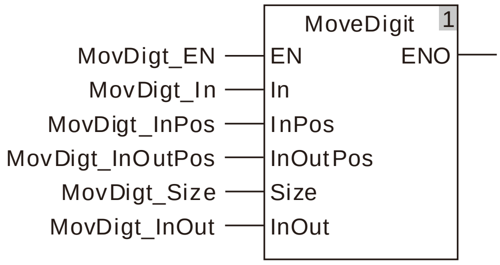
- **Precautions for Correct Use**
  - The instruction has no output but input parameter.
  - The move can not be performed if the value of *Size* is 0.
  - If the value of *InPos* exceeds the range of the data type of *In*, the move will not be performed.

- If the value of *InOutPos* exceeds the range of the data type of *InOut*, the movement is not performed.
- If the value of *Size* exceeds the range, the movement is not performed.

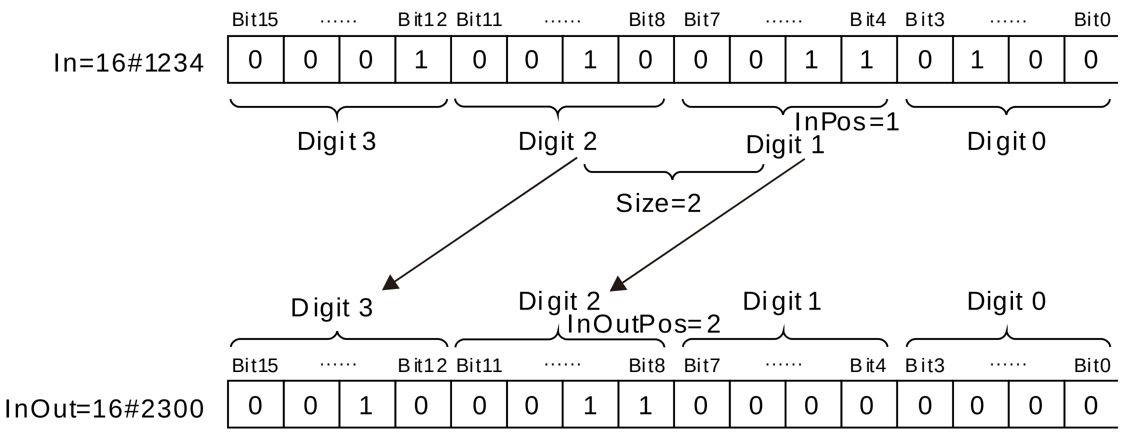
Programming Example

- **The variable table and program**

Variable name	Data type	Current value
MovDigt_EN	BOOL	TRUE
MovDigt_In	UDINT	16#1234
MovDigt_InPos	UINT	1
MovDigt_InOutPos	UINT	2
MovDigt_Size	UINT	2
MovDigt_Inout	UDINT	16#2300

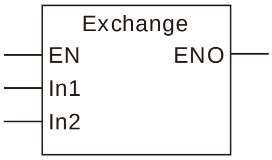


- **Move Figure**



8.4.5 Exchange

FB/FC	Explanation	Applicable model
FC	Exchange is used for the data exchange.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Input signal	Input	Data to exchange	Depends on the data type of the variable that the input parameter is connected to.
In2	Input signal	Input	Data to exchange	Depends on the data type of the variable that the input parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
In2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

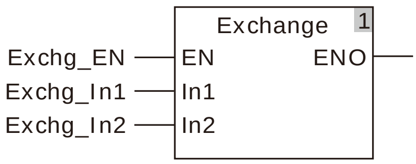
  The Exchange instruction exchanges the values of *In1* and *In2*.

- **Precautions for Correct Use**
  - The data types of *In1* and *In2* must be same.
  - The instruction has no output but two input parameters.

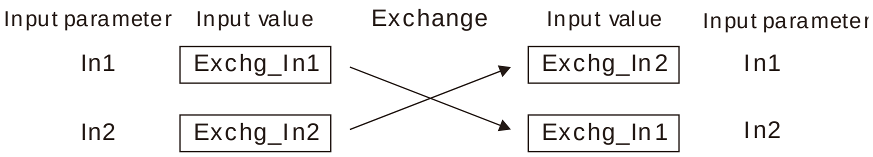
Programming Example

- **The variable table and program**

Variable name	Data type	Current value
Exchg_EN	BOOL	TRUE
Exchg_In1	INT	30
Exchg_In2	INT	10



■ Exchange Figure

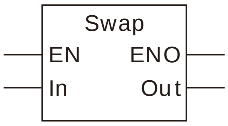


The values of In1 and In2 are exchanged.

While the Exchange instruction is executed, the values of Exchg_In1 and Exchg_In2 are always exchanged.

8.4.6 Swap

FB/FC	Explanation	Applicable model
FC	Swap is used for swapping the high byte and low byte of a 16-bit value.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Input signal	Input	Data to swap	0~65535 for word data type
Out	Output signal	Output	Result	0~65535 for word data type

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			●				●													
Out			●				●													

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

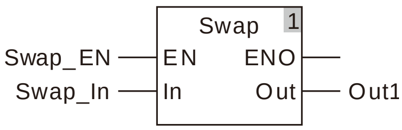
- **Function Explanation**

  The Swap instruction exchanges the high byte and low byte of the value of *In* and the result is output to *Out*.

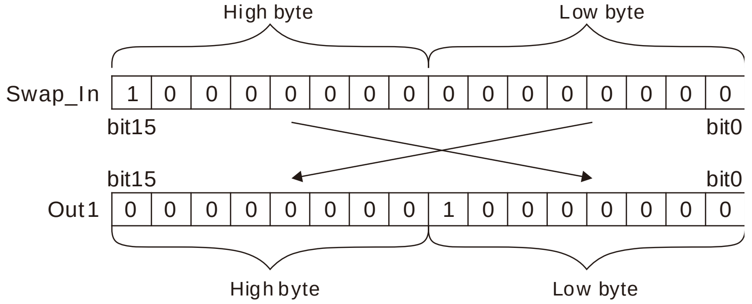
Programming Example

- **The variable table and program**

Variable name	Data type	Current value
Swap_EN	BOOL	TRUE
Swap_In	UINT	32768
Out1	UINT	128



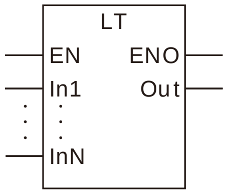
■ **Swap Figure**


High byte
Low byte
Swap_In
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
bit15
bit0
bit15
bit0
Out1
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
High byte
Low byte

8.5 Comparison Instructions

8.5.1 LT

FB/FC	Explanation	Applicable model
FC	LT is used for a less-than comparison of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The number of comparison data can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2~8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	Comparison result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - LT is used for a less-than comparison of two or more variables or constants. if *In1<In2<...<InN, Out* is TRUE. Otherwise, *Out* is FALSE.

  - The input parameters *In1~InN* are allowed to be the variables of different data types in this instruction when the data types of input variables are not BOOL, TIME, DATE, TOD and STRING. When the data type of one input variable is one of BOOL, TIME, DATE, TOD and STRING, input parameters In1~InN are all required to be of the data type. For example, if the data type of In1 is TIME, the data type of In2~InN must be TIME. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The data type of output variables must be BOOL. Otherwise, an error will occur during the compiling of the software.

Programming Example

- **The data types of LT_In1, LT_In2 and LT_In3 are INT, UINT and DINT respectively and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of LT_In1, LT_In2 and LT_In3 are -10, 50 and 100 respectively and LT_EN changes to TRUE as shown in Variable 1.
  Out1 changes to FALSE when the values of LT_In1, LT_In2 and LT_In3 are 20, 10 and 100 respectively and LT_EN changes to TRUE as shown in Variable 2.

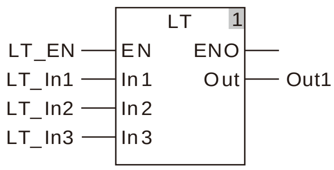
  - **Variable 1**

Variable name	Data type	Current value
LT_EN	BOOL	TRUE
LT_In1	INT	-10
LT_In2	UINT	50
LT_In3	DINT	100
Out1	BOOL	TRUE

  - **Variable 2**

Variable name	Data type	Current value
LT_EN	BOOL	TRUE
LT_In1	INT	20
LT_In2	UINT	10
LT_In3	DINT	100
Out1	BOOL	FALSE

  - **The Program**

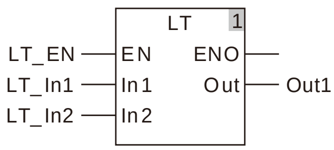


- **The data types of LT_In1 and LT_In2 are both TIME and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of LT_In1 and LT_In2 are T#1ms and T#50ms respectively and LT_EN is TRUE.

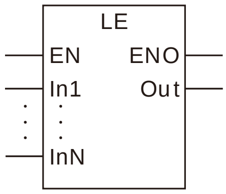
  - **The variable table and program**

Variable name	Data type	Current value
LT_EN	BOOL	TRUE
LT_In1	TIME	T#1ms
LT_In2	TIME	T#50ms
Out1	BOOL	TRUE



8.5.2 LE

FB/FC	Explanation	Applicable model
FC	LE is used for a less- than or equal comparison of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The number of comparison data can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	Comparison result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - LE is used for a less than or equal comparison of two or more variables or constants. if *In1* ≤ *In2*≤ ... ≤ *InN, Out* is TRUE. Otherwise, *Out* is FALSE.
  - The input parameters *In1~InN* are allowed to be the variables of different data types in this instruction when the data types of input variables are not BOOL, TIME, DATE, TOD and STRING. When the data type of one input variable is one of BOOL, TIME, DATE, TOD and STRING, input parameters In1~InN are all required to be of the data type. For example, if the data type of In1 is TIME, the data type of In2~InN must be TIME. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The data type of output variables must be BOOL. Otherwise, an error will occur during the compiling of the software.

Programming Example

- **The data types of LE_In1, LE_In2 and LE_In3 are INT, UINT and DINT respectively and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of LE_In1, LE_In2 and LE_In3 are -10, 50 and 50 respectively and LE_EN changes to TRUE as shown in Variable 1.
  Out1 changes to FALSE when the values of LE_In1, LE_In2 and LE_In3 are 20, 10 and 100 respectively and LE_EN changes to TRUE as shown in Variable 2.

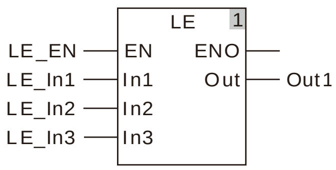
➢ **Variable 1**

Variable name	Data type	Current value
LE_EN	BOOL	TRUE
LE_In1	INT	-10
LE_In2	UINT	50
LE_In3	DINT	50
Out1	BOOL	TRUE

➢ **Variable 2**

Variable name	Data type	Current value
LE_EN	BOOL	TRUE
LE_In1	INT	20
LE_In2	UINT	10
LE_In3	DINT	100
Out1	BOOL	FALSE

➢ **The Program**

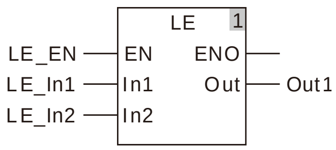


- **The data types of LE_In1 and LE_In2 are both TIME and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of LE_In1 and LE_In2 are T#1ms and T#50ms respectively and LE_EN is TRUE.

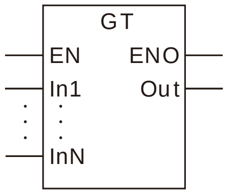
➢ **The variable table and program**

Variable name	Data type	Current value
LE_EN	BOOL	TRUE
LE_In1	TIME	T#1ms
LE_In2	TIME	T#50ms
Out1	BOOL	TRUE



8.5.3 GT

FB/FC	Explanation	Applicable model
FC	GT is used for a greater-than comparison of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The number of comparison data can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	Comparison result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - LE is used for a greater than comparison of two or more variables or constants. if *In1*>*In2*>...>*InN*, *Out* is TRUE. Otherwise, *Out* is FALSE.
  - The input parameters *In1~InN* are allowed to be the variables of different data types in this instruction when the data types of input variables are not BOOL, TIME, DATE, TOD and STRING. When the data type of one input variable is one of BOOL, TIME, DATE, TOD and STRING, input parameters In1~InN are all required to be of the data type. For example, if the data type of In1 is TIME, the data type of In2~InN must be TIME. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The data type of output variables must be BOOL. Otherwise, an error will occur during the compiling of the software.

Programming Example

- **The data types of GT_In1, GT_In2 and GT_In3 are INT, UINT and DINT respectively and the data type of Out1 is BOOL.**

Out1 changes to TRUE when the values of GT_In1, GT_In2 and GT_In3 are 100, 50 and 10 respectively and GT_EN changes to TRUE as shown in Variable 1.

Out1 changes to FALSE when the values of GT_In1, GT_In2 and GT_In3 are 20, 10 and 100 respectively and GT_EN changes to TRUE as shown in Variable 2.

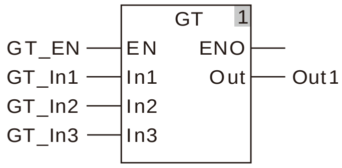
➢ **Variable 1**

Variable name	Data type	Current value
GT_EN	BOOL	TRUE
GT _In1	INT	100
GT _In2	UINT	50
GT _In3	DINT	10
Out1	BOOL	TRUE

➢ **Variable 2**

Variable name	Data type	Current value
GT_EN	BOOL	TRUE
GT _In1	INT	20
GT _In2	UINT	10
GT _In3	DINT	100
Out1	BOOL	FALSE

➢ **The Program**

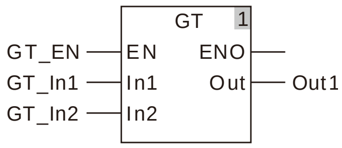


- **The data types of GT_In1 and GT_In2 are both TIME and the data type of Out1 is BOOL.**

Out1 changes to TRUE when the values of GT_In1 and GT_In2 are T#100ms and T#50ms respectively and GT_EN changes to TRUE.

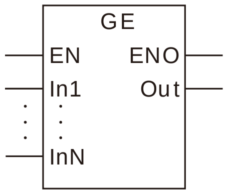
➢ **The variable table and program**

Variable name	Data type	Current value
GT_EN	BOOL	TRUE
GT _In1	TIME	T#100ms
GT _In2	TIME	T#50ms
Out1	BOOL	TRUE



8.5.4 GE

FB/FC	Explanation	Applicable model
FC	GE is used for a greater- than or equal comparison of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The number of comparison data can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	Comparison result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - GE is used for a greater than or equal comparison of two or more variables or constants. if *In1* ≥ *In2* ≥ ... ≥ *InN, Out* is TRUE. Otherwise, *Out* is FALSE.
  - The input parameters *In1~InN* are allowed to be the variables of different data types in this instruction when the data types of input variables are not BOOL, TIME, DATE, TOD and STRING. When the data type of one input variable is one of BOOL, TIME, DATE, TOD and STRING, input parameters In1~InN are all required to be of the data type. For example, if the data type of In1 is TIME, the data type of In2~InN must be TIME. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The data type of output variables must be BOOL. Otherwise, an error will occur during the compiling of the software.

Programming Example

- **The data types of GE_In1, GE_In2 and GE_In3 are INT, UINT and DINT respectively and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of GE_In1, GE_In2 and GE_In3 are 100, 50 and 50 respectively and GE_EN changes to TRUE as shown in Variable 1.
  Out1 changes to FALSE when the values of GE_In1, GE_In2 and GE_In3 are 10, 10 and 100 respectively and GE_EN changes to TRUE as shown in Variable 2.

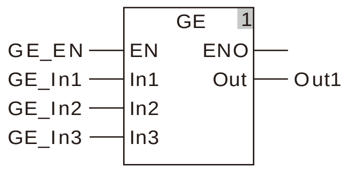
➢ **Variable 1**

Variable name	Data type	Current value
GE_EN	BOOL	TRUE
GE_In1	INT	100
GE_In2	UINT	50
GE_In3	DINT	50
Out1	BOOL	TRUE

➢ **Variable 2**

Variable name	Data type	Current value
GE_EN	BOOL	TRUE
GE_In1	INT	10
GE_In2	UINT	10
GE_In3	DINT	100
Out1	BOOL	FALSE

➢ **The program**

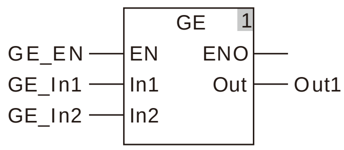


- **The data types of GE_In1 and GE_In2 are both TIME and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of GE_In1 and GE_In2 are T#100ms and T#50ms respectively and GE_EN changes to TRUE.

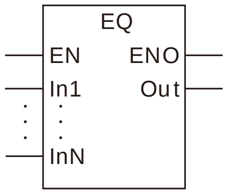
➢ **The variable table and program**

Variable name	Data type	Current value
GE_EN	BOOL	TRUE
GE_In1	TIME	T#100ms
GE_In2	TIME	T#50ms
Out1	BOOL	TRUE



8.5.5 EQ

FB/FC	Explanation	Applicable model
FC	EQ is used for an equal comparison of two or more variables and constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The number of comparison data can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	Comparison result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

1. EQ is used for an equal comparison of two or more variables and constants. If *In1* = *In2* = ... = *InN, Out* is TRUE*.* Otherwise, *Out* is FALSE.
2. The input parameters *In1~InN* are allowed to be the variables of different data types in this instruction when the data types of input variables are not BOOL, TIME, DATE, TOD and STRING. When the data type of one input variable is one of BOOL, TIME, DATE, TOD and STRING, input parameters In1~InN are all required to be of the data type. For example, if the data type of In1 is TIME, the data type of In2~InN must be TIME. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The data type of output variables must be BOOL. Otherwise, an error will occur during the compiling of the software.

Programming Example

- **The data types of EQ_In1, EQ_In2 and EQ_In3 are INT, UINT and DINT respectively and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of EQ_In1, EQ_In2 and EQ_In3 are 50, 50 and 50 respectively and EQ_EN changes to TRUE as shown in Variable 1.
  Out1 changes to FALSE when the values of EQ_In1, EQ_In2 and EQ_In3 are 10, 50 and 100 respectively and EQ_EN changes to TRUE as shown in Variable 2.

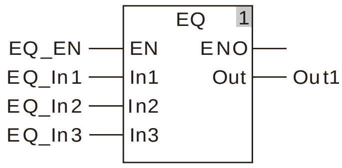
Variable 1

Variable name	Data type	Current value
EQ_EN	BOOL	TRUE
EQ _In1	INT	50
EQ _In2	UINT	50
EQ _In3	DINT	50
Out1	BOOL	TRUE

Variable 2

Variable name	Data type	Current value
EQ_EN	BOOL	TRUE
EQ _In1	INT	10
EQ _In2	UINT	50
EQ _In3	DINT	100
Out1	BOOL	FALSE

The Program

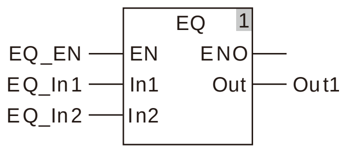


- **The data types of EQ_In1 and EQ_In2 are both TIME and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of EQ_In1 and EQ_In2 are T#50ms and T#50ms respectively and EQ_EN changes to TRUE.

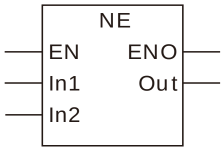
The variable table and program

Variable name	Data type	Current value
EQ_EN	BOOL	TRUE
EQ _In1	TIME	T#50ms
EQ _In2	TIME	T#50ms
Out1	BOOL	TRUE



8.5.6 NE

FB/FC	Explanation	Applicable model
FC	NE is used for a not-equal comparison of two variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Comparison data	Input	A value to compare	Depends on the data type of the variable that the input parameter is connected to.
In2	Comparison data	Input	A value to compare	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	Comparison result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 and In2	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - NE is used for a not-equal comparison of two variables and constants. *Out* is TRUE if *In1≠In2.* Otherwise, *Out* is FALSE.
  - The input parameters *In1* and *In2* are allowed to be the variables of different data types in this instruction when the data types of input variables are not BOOL, TIME, DATE, TOD and STRING. When the data type of one input variable is one of BOOL, TIME, DATE, TOD and STRING, input parameters *In1* and *In2* are both required to be of the data type. For example, if the data type of *In1* is TIME, the data type of *In2* must be TIME. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The data type of output variables must be BOOL. Otherwise, an error will occur during the compiling of the software.

Programming Example

- **The data types of NE_In1 and NE_In2 are INT and DINT respectively and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of NE_In1 and NE_In2 are 100 and 50 respectively and NE _EN changes to TRUE as shown in Variable 1.
  Out1 changes to FALSE when the values of NE_In1 and NE _In2 are 100 and 100 respectively and NE_EN changes to TRUE as shown in Variable 2.

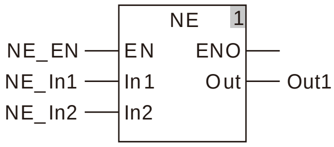
  - **Variable 1**

Variable name	Data type	Current value
NE_EN	BOOL	TRUE
NE _In1	INT	100
NE _In2	UINT	50
Out1	BOOL	TRUE

  - **Variable 2**

Variable name	Data type	Current value
NE_EN	BOOL	TRUE
NE _In1	INT	100
NE _In2	UINT	100
Out1	BOOL	FALSE

  - **The Program**

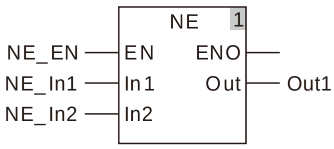


- **The data types of NE_In1 and NE_In2 are both TIME and the data type of Out1 is BOOL.**

  Out1 changes to TRUE when the values of NE_In1 and NE_In2 are T#10ms and T#50ms respectively and NE_EN changes to TRUE.

  - **The variable table and program**

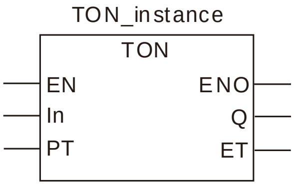
Variable name	Data type	Current value
NE_EN	BOOL	TRUE
NE _In1	TIME	T#10ms
NE _In2	TIME	T#50ms
Out1	BOOL	TRUE



8.6 Timer Instructions

8.6.1 TON

FB/FC	Explanation	Applicable model
FB	TON is used for the ON delay.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Timer input	Input	Controls the timer to start or reset	TRUE or FALSE
PT	Set time	Input	Time from when the timer starts until Q changes to TRUE.	
Q	Timer output	Output	Q is TRUE when the set time *PT* is reached.	TRUE or FALSE
ET	Elapsed time	Output	Elapsed time from the time when the timer starts to current time.	

◆ T#0ns ~ 213503d23h34m33s709ms551us615ns

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	●																			
PT																●				
Q	●																			
ET																●				

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The TON instruction is defined as the function of a timer for the ON delay.
  - When *In* is TRUE, the timer starts to measure the time and the value of *ET* increases accordingly. When *ET* equals PT, Q is TRUE. When *In* is set to FALSE, the measuring of the time stops and *Q* and *ET* are both reset.

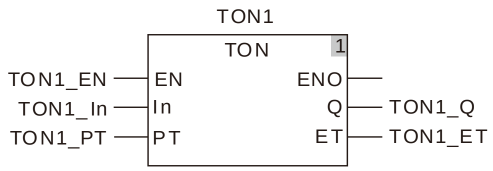
- **Precautions for Correct Use**

  When the output value of *ET* reaches the set value of *PT*, the timer stops measuring time. *ET* is reset to 0 (0ns) when *In* changes from TRUE to FALSE.

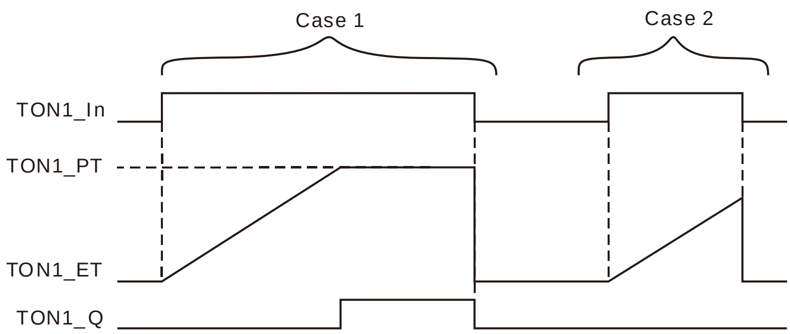
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
TON1	TON	
TON1_EN	BOOL	FALSE
TON1_In	BOOL	FALSE
TON1_PT	TIME	
TON1_Q	BOOL	
TON1_ET	TIME	



- **Timing Chart:**



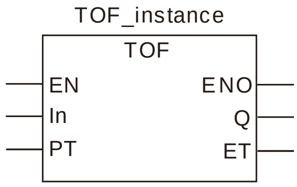
Case 1 : TON1_PT is the set time. When TON1_In is TRUE, the timer starts to measure the time. When the value of TON1_ET equals the setting value of TON1_PT, TON1_Q is TRUE. When the timer stops measuring time, TON1_In is reset to FALSE and TON1_ET and TON1_Q are both reset.

Case 2 : When the currently measured time of the timer TON1_ET is less than the set time TON1_PT and TON1_In is reset to FALSE, TON1_ET is reset and the state of TON1_Q does not change.

8

8.6.2 TOF

FB/FC	Explanation	Applicable model
FB	TOF is used for the off delay.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Timer input	Input	Controls the timer to start or reset	TRUE or FALSE
PT	Set time	Input	Set the time from when the timer starts until Q changes to TRUE	
Q	Timer output	Output	Q is FALSE when the set time *PT* is reached.	TRUE or FALSE
ET	Elapsed time	Output	Elapsed time from the time when the timer starts to current time.	

◆ T#0ns ~ 213503d23h34m33s709ms551us615ns

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	●																			
PT																●				
Q	●																			
ET																●				

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The TOF instruction is defined as the function of a timer for the OFF delay.
  - When *In* is TRUE, *Q* is TRUE. When *In* changes from TRUE to FALSE, the timer starts to measure the time and the value of *ET* increases accordingly. At the moment, Q remains TRUE. When *ET* equals PT, Q is FALSE and the timer stops measuring time. When *In* is set to TRUE, ET is reset and *Q* changes to TRUE again.

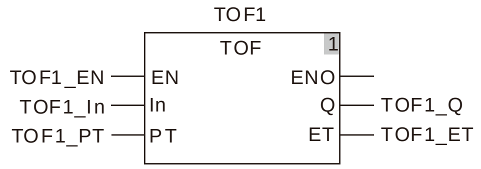
- **Precautions for Correct Use**

  When the output value of *ET* reaches the set value of *PT*, the timer stops measuring time. *ET* is reset to 0 (0ns) when *In* changes from FALSE to TRUE.

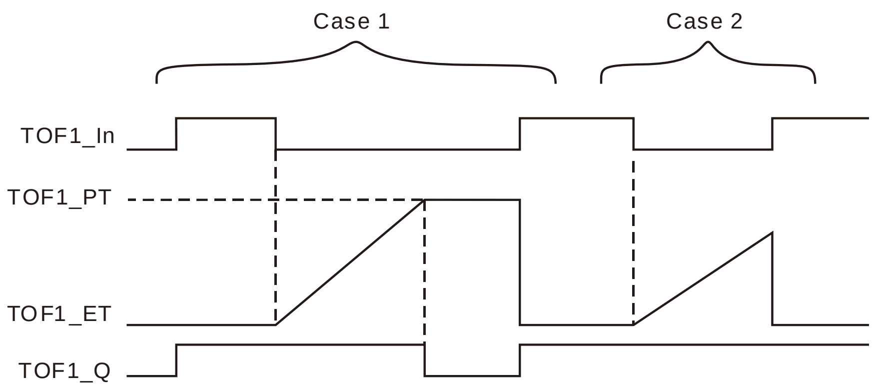
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
TOF1	TOF	
TOF1_EN	BOOL	FALSE
TOF1_In	BOOL	FALSE
TOF1_PT	TIME	
TOF1_Q	BOOL	
TOF1_ET	TIME	



- **Timing Chart:**

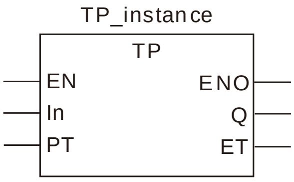


Case 1 : TOF1_PT is the set time for off delay. When TOF1_In is TRUE, TOF1_Q is TRUE. When TOF1_In is FALSE, the timer starts to measure the time. When the value of TOF1_ET equals the setting value of TOF1_PT, TOF1_Q is FALSE and the timer stops timing.

Case 3 : When TOF1_In changes from TRUE to FALSE, the timer starts timing. When current time (TOF1_ET) is less than the set time (TOF1_PT) and TOF1_In is set to TRUE, TOF1_ET is reset and the state of TOF1_Q does not change.

8.6.3 TP

FB/FC	**Explanation**	**Applicable model**
FB	TP is used for the off delay after the input **In** is TRUE.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Timer input	Input	Controls the timer to start or reset	TRUE or FALSE
PT	Set time	Input	Set the time from when the timer starts until Q changes to TRUE	◆
Q	Timer output	Output	Q is FALSE when the set time *PT* is reached.	TRUE or FALSE
ET	Elapsed time	Output	Elapsed time from the time when the timer starts to current time.	◆

◆ T#0ns ~ 213503d23h34m33s709ms551us615ns

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	●																			
PT																●				
Q	●																			
ET																●				

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  When *In* is TRUE, *Q* is TRUE and the timer starts measuring time and the value of *ET* increases accordingly. At the moment, Q remains TRUE. When *ET* equals PT, Q is FALSE and the timer stops measuring time. When *In* changes from TRUE to FALSE, ET is reset.

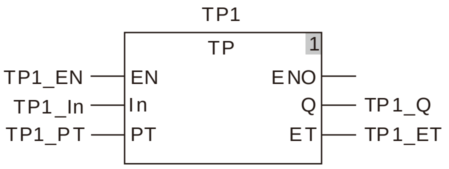
- **Precautions for Correct Use**

  When the output value of *ET* reaches the set value of *PT*, the timer stops measuring time. *ET* is reset to 0 (0ns) when *In* changes from TRUE to FALSE.

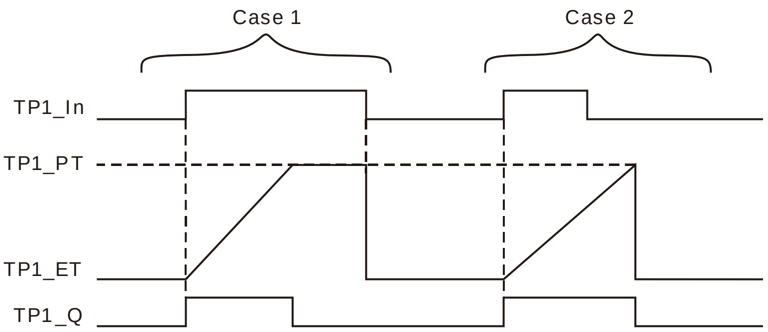
Programming Example

■ **The variable table and program**

Variable name	Data type	Initial value
TP1	TP	
TP1_EN	BOOL	FALSE
TP1_In	BOOL	FALSE
TP1_PT	TIME	
TP1_Q	BOOL	
TP1_ET	TIME	



■ **Timing Chart:**



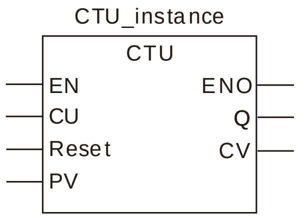
Case 1： TP1_PT sets the time for off delay. When TP1_In is TRUE, the timer starts to measure time and TP1_Q is TRUE. When the value of TP1_ET equals the setting value of TP1_PT, TP1_Q is FALSE. When TP1_In is FALSE, TP1_ET is reset.

Case 2： TP1_PT sets the time for off delay. When TP1_In is TRUE and the timer starts to measure time, TP1_Q is TRUE. When TP1_In is FALSE and the value of TP1_ET is less than the setting value of TP1_PT, TP1_ET keeps timing and TP1_Q keeps TRUE state. When the value of TP1_ET equals the setting value of TP1_PT, TP1_ET and TP1_Q are both reset.

8.7 Counter Instructions

8.7.1 CTU

FB/FC	Explanation	Applicable model
FB	CTU is used as an up counter.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
CU	Up-counter input signal	Input	Control the up-counter to start counting up	TRUE or FALSE
Reset	Reset signal	Input	Reset the counter present value	TRUE or FALSE
PV	Preset value	Input	Counter setting value	0 ~ 4294967295
Q	Output signal	Output	Q is TRUE when CV equals PV.	TRUE or FALSE
CV	Counter value	Output	Counter present value	0 ~ 4294967295

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	●																			
Reset	●																			
PV								●												
Q	●																			
CV								●												

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

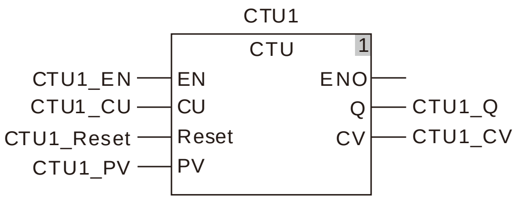
1. CTU functions as an up counter.
2. When *CU* changes from FALSE to TRUE, the counter perfoms the up-counting once and the value of *CV* is increased by 1. When *CV* equals *PV*, *Q* is TRUE. When *Reset* is set to TRUE, *CV* is cleared to 0 and *Q* is reset to FALSE.

- **Precautions for Correct Use**
  - While *Reset* is TRUE, the counter will not count up.
  - When CV equals PV, the counter stops counting.

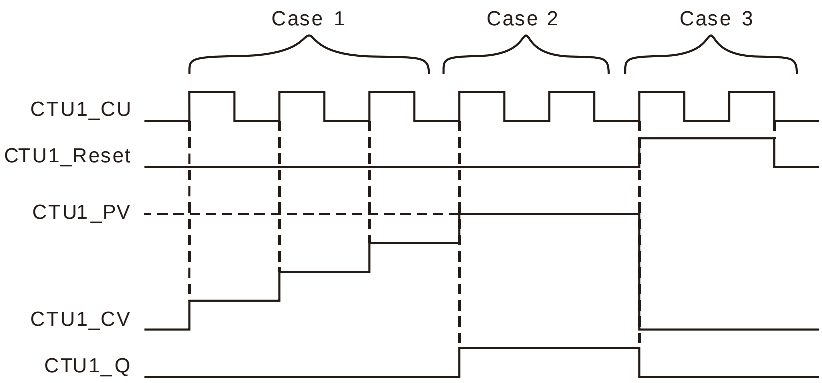
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
CTU1	CTU	
CTU1_EN	BOOL	FALSE
CTU1_CU	BOOL	FALSE
CTU1_Reset	BOOL	FALSE
CTU1_PV	UDINT	4
CTU1_Q	BOOL	
CTU1_CV	UDINT	



- **Timing Chart:**



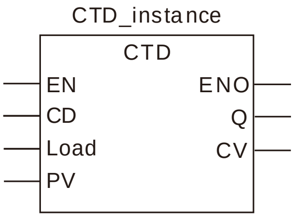
Case 1 : If CTU counts up normally, the value of CTU1_CV is increased by 1 whenever CTU1_CU is triggered once.

Case 2 : When CTU1_CV equals CTU1_PV, CTU1_Q is TRUE and CTU stops counting.

Case 3 : When CTU1_Reset is TRUE, CTU1_CV is cleared to 0, CTU1_Q is FALSE. And the counter will not count when CTU1_CU is triggered.

8.7.2 CTD

FB/FC	Explanation	Applicable model
FB	CTD is used as a down counter.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
CD	Down-counter input signal	Input	Control the counter to start counting down	TRUE or FALSE
Load	Load signal	Input	For writing the down-counter value	TRUE or FALSE
PV	Preset value	Input	Counter setting value	0 ~ 4294967295
Q	Output signal	Output	Q is TRUE when the counter counts down to 0.	TRUE or FALSE
CV	Counter value	Output	Counter present value	0 ~ 4294967295

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	●																			
Load	●																			
PV								●												
Q	●																			
CV								●												

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - CTU functions as a down counter.
  - When *Load* is reset to FALSE after being set to TRUE, the value of *PV* is written to *CV*. When *CD* changes from FALSE to TRUE, the counter makes the counter value decreased once and the value of *CV* is decreased by 1. When the value of *CV* reaches 0, *Q* is TRUE.

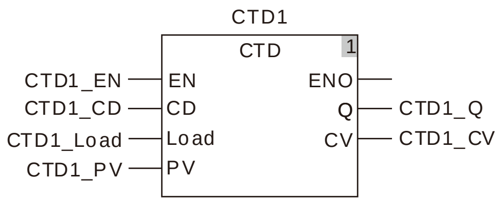
- **Precautions for Correct Use**

  While Load is TRUE, the counter will not count down.

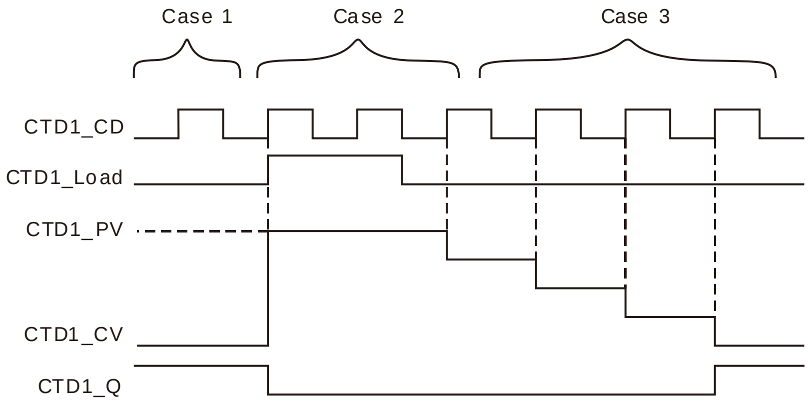
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
CTD1	CTD	
CTD1_EN	BOOL	FALSE
CTD1_CD	BOOL	FALSE
CTD1_Load	BOOL	FALSE
CTD1_PV	UDINT	4
CTD1_Q	BOOL	
CTD1_CV	UDINT	



- **Timing Chart:**



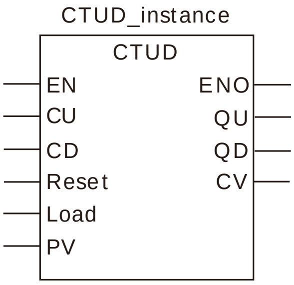
Case 1 : There is no impact on the ouput by triggering CTD1_CD when the value of CTD1_CV is 0.

Case 2 : When CTD1_Load is TRUE and CTD1_CV equals the set value of CTD1_PV, CTD1_Q changes from TRUE to FALSE. At the moment, CTD1_CV does not count down when CTD1_CD is triggered.

Case 3 : If CTD counts down normally and CTD1_Load is FALSE, the value of CTD1_CV is decreased by 1 whenever CTD1_CD is triggered once. CTD1_Q is TRUE when the value of CTD1_CV is decreased to 0.

8.7.3 CTUD

FB/FC	Explanation	Applicable model
FB	CTUD is used as an up-down counter.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
CU	Up-counter input signal	Input	Control the counter to count up	TRUE or FALSE
CD	Down-counter input signal	Input	Control the counter to count down	TRUE or FALSE
Reset	Reset signal	Input	Reset counter present value	TRUE or FALSE
Load	Load signal	Input	For writing the down-counter value	TRUE or FALSE
PV	Preset value	Input	Counter setting value	0 ~ 4294967295
QU	Output signal	Output	Q is TRUE when CV equals PV.	TRUE or FALSE
QD	Output signal	Output	Q is TRUE when the counter counts down to 0.	TRUE or FALSE
CV	Counter value	Output	Counter present value	0 ~ 4294967295

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	●																			
CD	●																			
Reset	●																			
Load	●																			
PV								●												
QU	●																			
QD	●																			
CV								●												

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

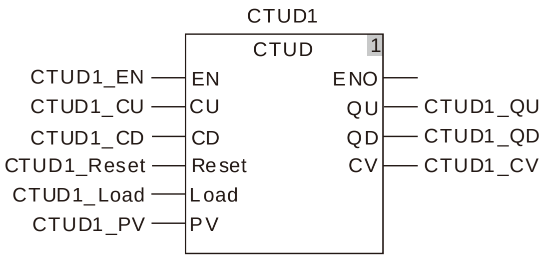
  CTUD is used as an up counter for counting up and a down counter for counting down.

- **Precautions for Correct Use**
  - The counter will not count down while *Load* is TRUE.
  - The counter will not count up while *Reset* is TRUE.

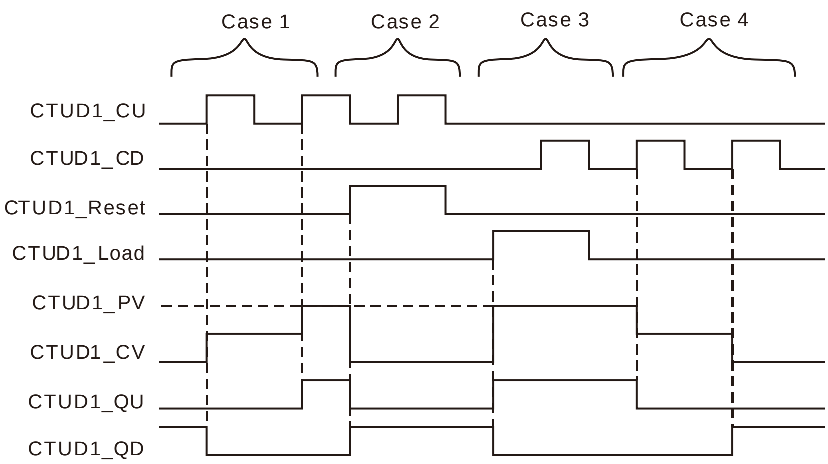
Programming Example

- **The variable table and program**

Variable name	Data type	Initial value
CTUD1	CTUD	
CTUD1_EN	BOOL	FALSE
CTUD1_CU	BOOL	FALSE
CTUD1_CD	BOOL	FALSE
CTUD1_Reset	BOOL	FALSE
CTUD1_Load	BOOL	FALSE
CTUD1_PV	UDINT	4
CTUD1_QU	BOOL	
CTUD1_QD	BOOL	
CTUD1_CV	UDINT	



- **Timing Chart:**



Case 1： If CTUD counts up normally, the value of CTUD1_CV is increased by 1 whenever CTUD1_CU is triggered once.

Case 2： When CTUD1_Reset is TRUE, CTUD1_CV is cleared to 0, CTUD1_QU changes to FALSE and CTUD1_QD changes to TRUE.

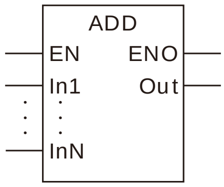
Case 3：When CTUD1_Load is TRUE and CTUD1_CV equals CTUD1_PV, CTUD1_QU changes to TRUE and CTUD1_QD changes to FALSE. At the moment, if CTUD1_CD is triggered, the instruction can not count down.

Case 4：If the instruction counts down normally, CTUD1_QU is FALSE when CTUD1_CD is TRUE. The value of CTUD1_CV is decreased by 1 whenever CTUD1_CD is triggered once. CTUD1_QD is TRUE when the value of CTUD1_CV is decreased to 0.

8.8 Math Instructions

8.8.1 ADD

FB/FC	Explanation	Applicable model
FC	ADD is used for the addition operation of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Augend	Input	Augend	Depends on the data type of the variable that the input parameter is connected to.
In2 to InN	Addend	Input	The maximum number of addends is 7, which means that N can be 2~8 and the number can be increased or reduced via the programming software in creating a program.	Depends on the data type of the variable that the input parameter is connected to.
Out	Sum	Output	The addition result of In1 to InN	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	
In2 to InN		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●				
Out		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	

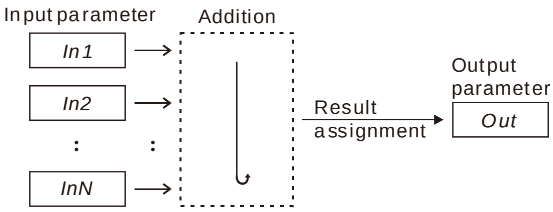
Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The instruction is used for the addition of two or more variables or constants. The result is output to *Out*, that is, *Out= In1 + In2 +…+ InN*.

- The input parameters *In1~InN* in this instruction are allowed to be the variables of different types among bits, integers and real numbers. When *In1~InN* are the variables of different types, the addition operation will be performed based on the data type which can contain all valid ranges of *In1~InN* values. For example, the data type of *Out* is DINT if the data type of *In1* is INT and *In2* is DINT.



- The input and output variables are allowed to be of different data types among bits, integers and real numbers. When the data types of input and output variables are different, the data type of the output variable must include the valid ranges of data types of all input variables. Otherwise, there will be an error during the compiling of the software. For example, if the data types of *In1* and *In2* are INT and DINT respectively, the data type of *Out* is DINT. There will be an error during compiling of the software if the data type of the variable that *Out* is connected to is INT. No error will occur during the compiling of the software if the data type of the variable that *Out* is connected to is LINT.
- For the data type about time and date, following combinations are supported only.
  1. In1 is TIME, In2 is TIME and Out is TIME;
  2. In1 is TOD (TIME_OF_DAY), In2 is TIME and Out is TOD;
  3. In1 is DT (DAY_AND_TIME), In2 is TIME and Out is DT.

- **Precautions for Correct Use**

- The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
- The sum of *In1~InN* may be out of the valid range of the data type of *Out.*
- The difference between *In1* and *In2* may be out of the valid range of the data type of *Out.*
  For example, the data types of “ADD_In1” and “ADD_In2” are both INT with their respective values, 32767 and 1. If the data type of the output variable is INT, the output variable value will be -32768 as shown in the following table, variable 1. If the data type of the output variable is set to DINT, the output variable value will be 32768 as shown in the following table, variable 2.

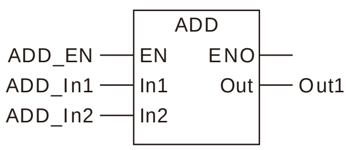
➢ Variable 1

Variable name	Data type	Current value
ADD_EN	BOOL	TRUE
ADD_In1	INT	32767
ADD_In2	INT	1
Out1	INT	-32768

➢ Variable 2

Variable name	Data type	Current value
ADD_EN	BOOL	TRUE
ADD_In1	INT	32767
ADD_In2	INT	1
Out1	DINT	32768

➢ The program



- **Programming Example1**

  - The data types of variables *ADD_In1*, *ADD_In2* and *Out1* are all INT. The values of *ADD_In1* and *ADD_In2* are 10 and 50 respectively. The value of *Out1* is 60 when *ADD_EN* changes to TRUE as shown in Variable 1.
  - The data types of variables *ADD_In1*, *ADD_In2* and *Out1* are all TIME. The values of *ADD_In1* and *ADD_In2* are TIME #1s and TIME #2s respectively. The value of *Out1* is TIME #3s when *ADD_EN* changes to TRUE as shown in Variable 2.
  - The data types of variables *ADD_In1*, *ADD_In2* and *Out1* are DT, TIME and DT respectively. The values of *ADD_In1* and *ADD_In2* are DT#2016-9-1-8:00:00 and TIME#1H53M34S respectively. The value of *Out1* is DT#2016-09-01-09:53:34 when *ADD_EN* changes to TRUE as shown in Variable 3.

➢ **Variable 1**

Variable name	Data type	Current value
ADD_EN	BOOL	TRUE
ADD_In1	INT	10
ADD_In2	INT	50
Out1	INT	60

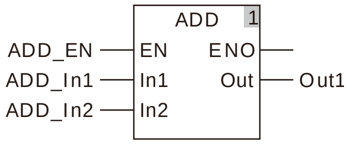
➢ **Variable 2**

Variable name	Data type	Current value
ADD_EN	BOOL	TRUE
ADD_In1	TIME	TIME #1s
ADD_In2	TIME	TIME #2s
Out1	TIME	TIME #3s

➢ **Variable 3**

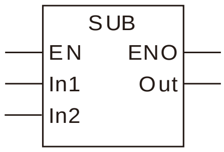
Variable name	Data type	Current value
ADD_EN	BOOL	TRUE
ADD_In1	DT	DT#2016-9-1-8:00:00
ADD_In2	TIME	TIME#1H53M34S
Out1	DT	DT#2016-09-01-09:53:34

➢ **The program**



8.8.2 SUB

FB/FC	Explanation	Applicable model
FC	SUB is used for the subtraction operation of two variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Minuend	Input	Minuend	Depends on the data type of the variable that the input parameter is connected to.
In2	Subtrahend	Input	Subtrahend	Depends on the data type of the variable that the input parameter is connected to.
Out	Difference	Output	The subtraction result of In1 and In2	Depends on the data type of the variable that the output parameter is connected to.

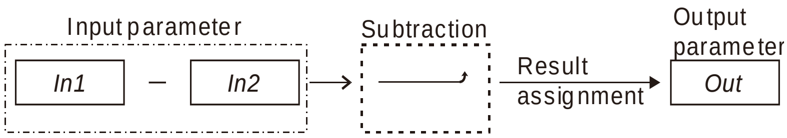
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
In2		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	
Out		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●	●	

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

8

- **Function Explanation**
  - The instruction is used for the subtraction of two or more variables or constants. The result is output to *Out,* that is, *Out= In1 - In2*.
  - The input parameters *In1* and *In2* in this instruction are allowed to be the variables of different data types among bits, integers and real numbers. When *In1* and *In2* are the variables of different types, the subtraction operation will be performed based on the data type which can contain valid ranges of *In1* and *In2* values. For example, the data type of *Out* is DINT if the data type of *In1* is INT and *In2* is DINT.



- The input and output variables are allowed to be of different data types among bits, integers and real numbers. When the data types of input and output variables are different, the data type of the output variable must include the valid ranges of data types of all input variables. Otherwise, there will be an error during the compiling of the software. For example, if the data types of *In1* and *In2* are INT and DINT respectively, the data type of *Out* is DINT. There will be an error during the compiling of the software if the data type of the variable that *Out* is connected to is INT. No error will occur during the compiling of the software if the data type of the variable that *Out* is connected to is LINT.
- For the data type of time and date, only following combinations are supported.

1. In1 is TIME, In2 is TIME and Out is TIME;
2. In1 is TOD, In2 is TIME and Out is TOD;
3. In1 is TOD, In2 is TOD and Out is TIME;
4. In1 is DATE, In2 is DATE and Out is TIME;
5. In1 is DT, In2 is DT and Out is TIME;
6. In1 is DT, In2 is TIME and Out is DT.

- **Precautions for Correct Use**

- The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
- The difference between *In1* and *In2* may be out of the valid range of the data type of *Out.*
  For example, the data types of “SUB _In1” and “SUB _In2” are both INT with their respective values, -32768 and 1. If the data type of the output variable is INT, the output variable value will be 32767 as shown in the following table, variable 1. If the data type of the output variable is set to DINT, the output variable value will be -32769 as shown in the following table, variable 2.

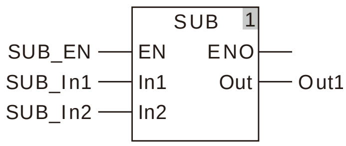
➢ **Variable 1**

Variable name	Data type	Current value
SUB_EN	BOOL	TRUE
SUB _In1	INT	-32768
SUB _In2	INT	1
Out1	INT	32767

➢ **Variable 2**

Variable name	Data type	Current value
SUB_EN	BOOL	TRUE
SUB _In1	INT	-32768
SUB _In2	INT	1
Out1	DINT	-32769

➢ **The Program**



- **Programming Example**

- The data types of variables *SUB_In1, SUB _In2* and *Out1* are all INT and the values of *SUB _In1* and *SUB _In2* are 100 and 40 respectively. The value of *Out1* is 60 when *SUB_EN* changes to TRUE as shown in Variable 1.

- The data types of variables *SUB_In1, SUB _In2* and *Out1* are all TIME and the values of *SUB _In1* and *SUB _In2* are TIME#4s and TIME#1s respectively. The value of *Out1* is TIME#3s when *SUB_EN* changes to TRUE as shown in Variable 2.
- The data types of variables *SUB_In1, SUB _In2* and *Out1* are DATE, DATE and TIME and the values of *SUB _In1* and *SUB _In2* are DATE#2016-10-1 and DATE#2016-9-1 respectively. The value of *Out1* is TIME#30D when *SUB_EN* changes to TRUE as shown in Variable 3.

➢ **Variable 1**

Variable name	Data type	Current value
SUB_EN	BOOL	TRUE
SUB _In1	INT	100
SUB _In2	INT	40
Out1	INT	60

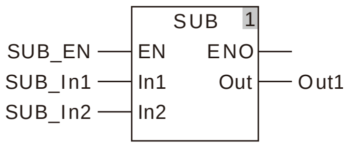
➢ **Variable 2**

Variable name	Data type	Current value
SUB_EN	BOOL	TRUE
SUB_In1	TIME	TIME#4s
SUB_In2	TIME	TIME#1s
Out1	TIME	TIME#3s

➢ **Variable 3**

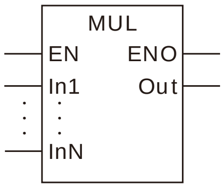
Variable name	Data type	Current value
SUB_EN	BOOL	TRUE
SUB_In1	DATE	DATE#2016-10-1
SUB_In2	DATE	DATE#2016-9-1
Out1	TIME	TIME#30D

➢ **The program**



8.8.3 MUL

FB/FC	Explanation	Applicable model
FC	MUL is used for the multiplication of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Multiplicand	Input	Multiplicand	Depends on the data type of the variable that the input parameter is connected to.
In2 to InN	Multiplier	Input	The maximum number of multipliers is 7, which means that N can be 2~8 and the number can be increased or reduced via the programming software in creating a program.	Depends on the data type of the variable that the input parameter is connected to.
Out	Product	Output	The multiplication result of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

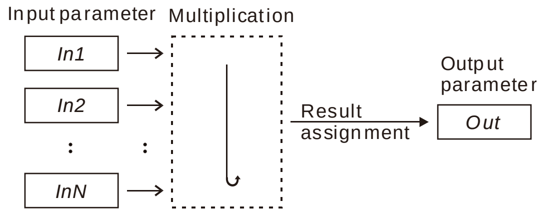
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out		●	●	●	●	●	●	●	●	●	●	●	●	●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The instruction is used for the multiplication of two or more variables or constants. The result is output to *Out,* that is, *Out*= In1 * In2 * … * InN.
  - The input parameters *In1 ~ InN* are allowed to be the variables of different data types in this instruction. When *In1 ~ InN* are the variables of different data types, the multiplication will be performed based on the data type which can contain valid ranges of *In1 ~ InN* values. For example, the data type of *Out* is DINT if the data type of *In1* is INT and *In2* is DINT.



■ The input and output variables are allowed to be of different data types in this instruction. When the data types of input and output variables are different, the range of the data type of the output variable must include the valid ranges of data types of all input variables. Otherwise, there will be an error during the compiling of the software. For example, if the data types of *In1* and *In2* are INT and DINT respectively, the data type of *Out* is DINT. There will be an error during the compiling of the software if the data type of the variable that *Out* is connected to is INT. No error will occur during the compiling of the software if the data type of the variable that *Out* is connected to is LINT.

- **Precautions for Correct Use**

■ The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.

■ The multiplication result of *In1 ~ In2* may be out of the valid range of the data type of *Out.*
For example, the data types of “MUL _In1” and “MUL _In2” are both INT with their respective values, 20000 and 2. If the data type of the output variable is INT, the output variable value will be -25536 as shown in the following table, Variable 1. If the data type of the output variable is set to DINT, the output variable value will be 40000 as shown in the following table, Variable 2.

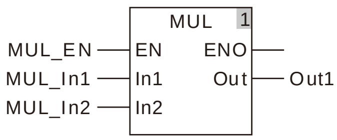
➢ **Variable 1**

Variable name	Data type	Current value
MUL_EN	BOOL	TRUE
MUL _In1	INT	20000
MUL _In2	INT	2
Out1	INT	-25536

➢ **Variable 2**

Variable name	Data type	Current value
MUL_EN	BOOL	TRUE
MUL _In1	INT	20000
MUL _In2	INT	2
Out1	DINT	40000

➢ **The Program**

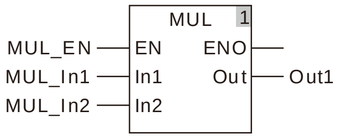


- **Programming Example**

■ The data types of variables *MUL _In1, MUL _In2* and *Out1* are all INT. The values of *MUL _In1* and *MUL _In2* are 10 and 50 respectively. The value of *Out1* is 500 when *MUL _EN* changes to TRUE.

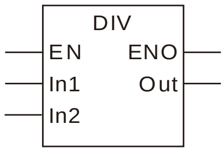
The variable table and program

Variable name	Data type	Initial value
MUL_EN	BOOL	TRUE
MUL _In1	INT	10
MUL _In2	INT	50
Out1	INT	500



8.8.4 DIV

FB/FC	Explanation	Applicable model
FC	DIV is used for the division operation of two variables or constants.	DVP15MC11T



- **Parameters**

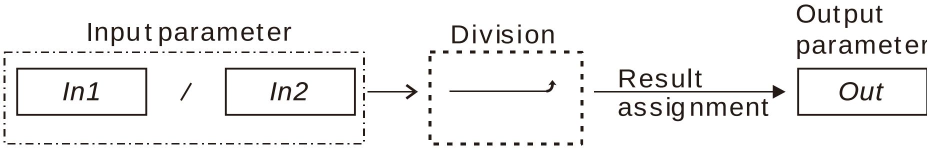
Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Dividend	Input	Dividend	Depends on the data type of the variable that the input parameter is connected to.
In2	Divisor	Input	Divisor	Depends on the data type of the variable that the input parameter is connected to. 0 is excluded.
Out	Quotient	Output	The division result of In1 andIn2	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
In2		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out		●	●	●	●	●	●	●	●	●	●	●	●	●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The instruction is used for the division of two variables or constants. The result is output to *Out,* that is, *Out*= In1 / In2.
  - The input parameters *In1* and *In2* are allowed to be the variables of different data types in this instruction. When *In1* and *In2* are the variables of different data types, the division will be performed based on the data type which can contain valid ranges of *In1* and *In2*. For example, the data type of *Out* is DINT if the data type of *In1* is INT and *In2* is DINT.



- The input and output variables are allowed to be of different data types in this instruction. When the data types of input and output variables are different, the range of the data type of the output variable must include the valid ranges of data types of all input variables. Otherwise, there will be an error during the compiling of the software. For example, if the data types of *In1* and *In2* are INT and DINT respectively, the data type of *Out* is DINT. There will be an error during the compiling of the software if the data type of *Out* is INT. No error will occur during the compiling of the software if the data type of *Out* is LINT.

- **Precautions for Correct Use**

- The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
- The input value of *In2* can not be 0. In other words, the divisor in the division operation can not be 0. The value of *Out* will be 0 if the value of *In2* is 0.
- The division result of *In1* and *In2* may be out of the valid range of the data type of *Out.*
  For example, the data types of “DIV _In1” and “DIV _In2” are both INT with their respective values, -32768 and -1. If the data type of the output variable is INT, the output variable value will be -32768 as shown in the following table, variable 1. If the data type of the output variable is set to DINT, the output variable value will be 32768 as shown in the following table, variable 2.

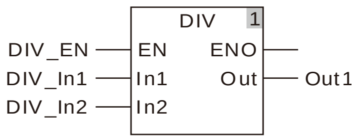
➢ **Variable 1**

Variable name	Data type	Current value
DIV_EN	BOOL	TRUE
DIV_In1	INT	-32768
DIV_In2	INT	-1
Out1	INT	-32768

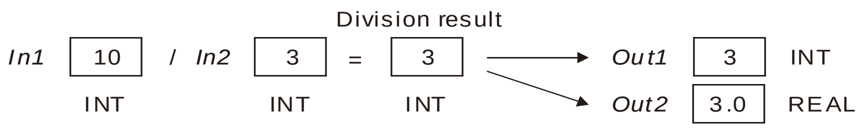
➢ **Variable 2**

Variable name	Data type	Current value
DIV_EN	BOOL	TRUE
DIV_In1	INT	-32768
DIV_In2	INT	-1
Out1	DINT	32768

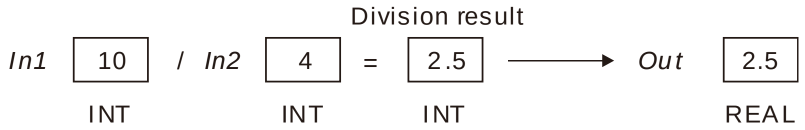
➢ **The Program**



- The result is always an integer for the division of two integers. Even if there is a remainder for the division of two integers, the remainder is cut.
  For example, the data types of *In1* and *In2* are both INT with their respective values, 10 and 3. And the data type of *Out* is INT and Real and thus its value is 3 and 3.0 respectively as illustrated in the following figure.



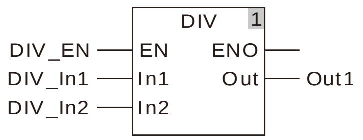
The data type of *Out* is a real number for the division of an integer and a real number or the division of two real numbers. The value of *Out* is shown as below including its fractional part when there is a remainder for this type of division.



- **Programming Example**
  - The data types of variables *DIV _In1, DIV _In2* and *Out1* are all INT. The values of *DIV _In1* and *DIV _In2* are 100 and 20 respectively. The value of *Out1* is 5 when *DIV _EN* changes to TRUE.

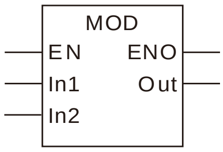
The variable table and program

Variable name	Data type	Initial value
DIV_EN	BOOL	TRUE
DIV_In1	INT	100
DIV_In2	INT	20
Out1	INT	5



8.8.5 MOD

FB/FC	Explanation	Applicable model
FC	MOD finds the remainder for division of two integer variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Dividend	Input	Dividend	Depends on the data type of the variable that the input parameter is connected to.
In2	Divisor	Input	Divisor	Depends on the data type of the variable that the input parameter is connected to. 0 is excluded.
Out	Remainder	Output	The remainder got by dividing In1 by In2	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1		●	●	●	●	●	●	●	●	●	●	●	●							
In2		●	●	●	●	●	●	●	●	●	●	●	●							
Out		●	●	●	●	●	●	●	●	●	●	●	●							

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

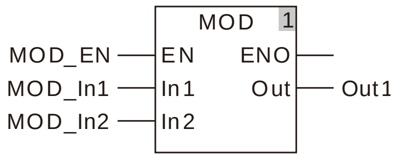
  - The instruction is used to get the remainder of the division of two integer variables or constants. The result is output to *Out,* that is, *Out*= In1 – (In1/ In2)*In2.
  - The input variable and input variable or the input variable and output variable are allowed to be of different data types in this instruction. When the data types of input and output variables are different, the data type of the output variable must include the valid ranges of data types of all input variables. Otherwise, there will be an error during the compiling of the software. For example, if the data types of *In1* and *In2* are INT and DINT respectively, the data type of *Out* is DINT. There will be an error during the compiling of the software if the data type of *Out* is INT. No error will occur during the compiling of the software if the data type of *Out* is LINT.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The input value of *In2* can not be 0. In other words, the divisor in the division operation can not be 0. The value of *Out* will be 0 if the value of *In2* is 0.

- **Programming Example**
  - The data types of variables *MOD _In1, MOD _In2* and *Out1* are all INT. The values of *MOD _In1* and *MOD _In2* are 10 and 4 respectively. The value of *Out1* is 2 when *MOD _EN* changes to TRUE.

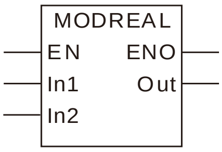
The Variable and program

Variable name	Data type	Current value
MOD_EN	BOOL	TRUE
MOD _In1	INT	10
MOD _In2	INT	4
Out1	INT	2



8.8.6 MODREAL

FB/FC	Explanation	Applicable model
FC	MODREAL finds the remainder for division of two floating- point variables or constants.	**DVP15MC11T**



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Dividend	Input	Dividend	Depends on the data type of the variable that the input parameter is connected to.
In2	Divisor	Input	Divisor	Depends on the data type of the variable that the input parameter is connected to. 0 is excluded.
Out	Remainder	Output	The remainder got by dividing In1 by In2	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1														●	●					
In2														●	●					
Out														●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The instruction is used to find the remainder of the division of two floating- point variables or constants and the result is output to *Out*.
  - The input variable and input variable or the input variable and output variable are allowed to be of different data types in this instruction.

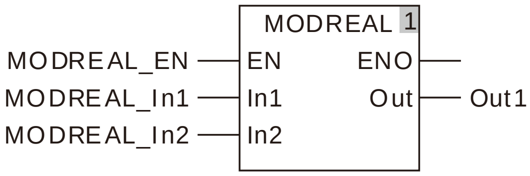
- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The input value of *In2* can not be 0. In other words, the divisor in the division operation can not be 0. The value of *Out* will be 0 if the value of *In2* is 0.

8

- **Programming Example**
  - The data types of variables *MODREAL _In1, MODREAL _In2* and *Out1* are REAL, REAL and LREAL respectively. The values of *MODREAL _In1* and *MOD _In2* are 10.5 and 2.5 respectively. The value of *Out1* is 0.5 when *MODREAL _EN* changes to TRUE.

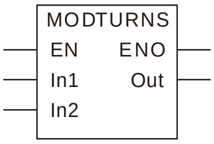
The variable table and program

Variable name	Data type	Current value
MODREAL_EN	BOOL	TRUE
MODREAL _In1	REAL	10.5
MODREAL _In2	REAL	2.5
Out1	LREAL	0.5



8.8.7 MODTURNS

FB/FC	Explanation	Applicable model
FC	MODTURN finds the signed integral part for modulo division of two floating-point variables or constants.	**DVP15MC11T**



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Input value	Input	Input value	Depends on the data type of the variable that the input parameter is connected to.
In2	Modulo range	Input	Modulo range	Depends on the data type of the variable that the input parameter is connected to. 0 is excluded.
Out	Number of modulo rotations	Output	Number of modulo rotations	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1														●	●					
In2														●	●					
Out												●								

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

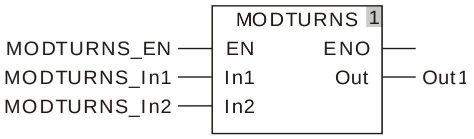
  - MODTURN is used to carry out modulo division of two floating-point variables or constants and get the signed integral component. The result is output to *Out*. The number of modulo rotations of an axis can be calculated according to its set absolute position.
  - The input variable and input variable or the input variable and output variable are allowed to be of different data types in this instruction.

8

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The input value of *In2* can not be 0. In other words, the divisor in the division operation can not be 0. The value of *Out* will be 0 if the value of *In2* is 0.
- **Programming Example**
  - The data types of variables *MODTURNS _In1, MODTURNS _In2 are* both REAL and *Out1* is DINT. The values of *MODTURNS _In1* and *MODTURNS _In2* are 800.23 and 360.0 respectively. The value of *Out1* is 2 when *MODTURNS _EN* changes to TRUE.

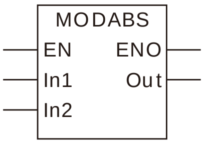
The variable table and program

Variable name	Data type	Current value
MODTURNS_EN	BOOL	TRUE
MODTURNS _In1	REAL	800.23
MODTURNS _In2	REAL	360.0
Out1	DINT	2



8.8.8 MODABS

FB/FC	Explanation	Applicable model
FC	MODABS finds the unsigned modulo value for modulo division of two floating-point variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Input value	Input	Input value	Depends on the data type of the variable that the input parameter is connected to.
In2	Modulo range	Input	Modulo range	Depends on the data type of the variable that the input parameter is connected to.由0 is excluded.
Out	Modulo value	Output	Modulo value	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1														●	●					
In2														●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - MODABS is used to perform modulo division of two floating-point variables or constants and get the unsigned modulo value. The result is output to *Out*. The modulo position can be calculated according to the absolute position of the axis.
  - The input variable and input variable or the input variable and output variable are allowed to be of different data types in this instruction.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The input value of *In2* can not be 0. In other words, the divisor in the division operation can not be 0. The value of *Out* will be 0 if the value of *In2* is 0.

- **Programming Example**
  - The data types of variables *MODABS _In1* and *MODABS _In2* are both REAL and the data type of *Out1* is LREAL. The values of *MODABS _In1* and *MODABS _In2* are 400.23 and 360.0 respectively. The value of *Out1* is 40.2300109863281 when *MODABS _EN* changes to TRUE. The values of *MODABS _In1* and *MODABS _In2* are -400.23 and 360.0 respectively. The value of *Out1* is 319.769989013672 when *MODABS _EN* changes to TRUE.

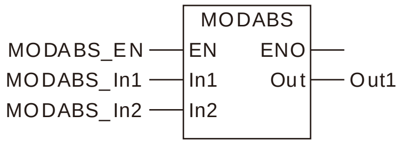
➢ **Variable 1**

Variable name	Data type	Current value
MODABS_EN	BOOL	TRUE
MODABS _In1	REAL	400.23
MODABS _In2	REAL	360.0
Out1	LREAL	40.2300109863281

➢ **Variable 2**

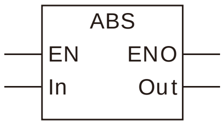
Variable name	Data type	Current value
MODABS_EN	BOOL	TRUE
MODABS _In1	REAL	-400.23
MODABS _In2	REAL	360.0
Out1	LREAL	319.769989013672

➢ **The program**



8.8.9 ABS

FB/FC	Explanation	Applicable model
FC	ABS finds the absolute value of an integer or a real number.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Absolute value	Output	Absolute value of *In*	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out		●	●	●	●	●	●	●	●	●	●	●	●	●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - ABS finds the absolute value of the input parameter *In*. The result is output to *Out*. That is, Out = | In1 |.
  - The input variable and output variable are allowed to be of different data types in this instruction. When the data types of input and output variables are different, the range of the data type of the output variable must include the valid ranges of data types of all input variables. Otherwise, there will be an error during the compiling of the software.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**
  - The data types of variables *ABS _In* and *Out1* are both INT and the value of *ABS _In* is -10. The value of *Out1* is 10 when *ABS _EN* changes to TRUE. The value of *Out1* is 20 as *ABS_In* is 20.

8

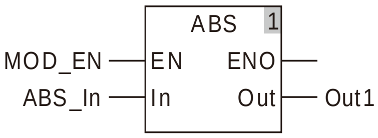
- **Variable 1**

Variable name	Data type	Current value
ABS_EN	BOOL	TRUE
ABS _In	INT	-10
Out1	INT	10

- **Variable 2**

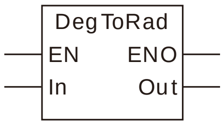
Variable name	Data type	Current value
ABS_EN	BOOL	TRUE
ABS _In	INT	20
Out1	INT	20

- **The program**



8.8.10 DegToRad

FB/FC	Explanation	Applicable model
FC	DegToRad is used to convert degrees to radians.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Degrees	Input	Degrees to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Radians	Output	Radians converted from degrees	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - DegToRad is used to convert the input parameter *In* to a radian and the result is output to *Out*. That is, Out =(In/180)* π.
  - The units of *In* and *Out* are degree (°) and radian respectively.
  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variables are allowed to omit.

- **Programming Example**
  - The data types of *DegToRad _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 0.174532925199433 if the value of *DegToRad _In* is 10 when *DegToRad _EN* changes to TRUE. The value of *Out1* is -0.174532925199433 as *DegToRad _In* is -10.

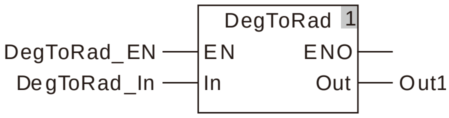
➢ **Variable 1**

Variable name	Data type	Current value
DegToRad_EN	BOOL	TRUE
DegToRad _In	INT	10
Out1	LREAL	0.174532925199433

➢ **Variable 2**

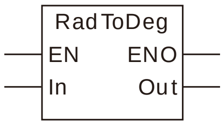
Variable name	Data type	Current value
DegToRad_EN	BOOL	TRUE
DegToRad _In	INT	-10
Out1	LREAL	-0.174532925199433

➢ **The program**



8.8.11 RadToDeg

FB/FC	Explanation	Applicable model
FC	DegToRad is used to convert radians to degrees.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Radians	Input	Radians to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Degrees	Output	Degrees converted from radians	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - RadToDeg is used to convert the input parameter *In* to degrees and the result is output to *Out*. That is, Out =(In/π)* 180.
  - The units of *In* and *Out* are radian and degree (°) respectively.
  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**
  - The data types of variables *RadToDeg _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 572. 957795130824 if the value of *RadToDeg _In* is 10 when *RadToDeg _EN* changes to TRUE. The value of *Out1* is -572. 957795130824 as *RadToDeg _In* is -10.

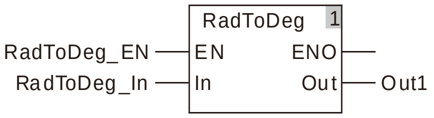
➢ **Variable 1**

Variable name	Data type	Current value
RadToDeg _EN	BOOL	TRUE
RadToDeg _In	INT	10
Out1	LREAL	572. 957795130824

➢ **Variable 2**

Variable name	Data type	Current value
RadToDeg_EN	BOOL	TRUE
RadToDeg_In	INT	-10
Out1	LREAL	-572. 957795130824

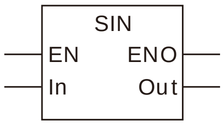
➢ **The program**



8

8.8.12 SIN

FB/FC	Explanation	Applicable model
FC	SIN is used to find the sine of a number and the result is output to *Out*. The unit of *In* is radian.	DVP15MC11T



- **Parameters**

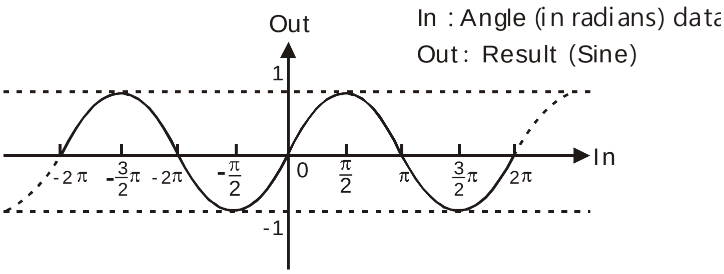
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Radians to process	Input	Radians to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result	-1.00000000000000 ~ 1.00000000000000

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - SIN is used to calculate the sine of the input parameter *In* and the result is output to *Out*.



  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

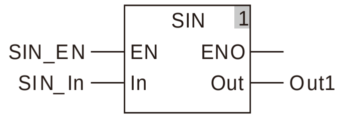
- **Precautions for Correct Use**
  - The input variable setting is not allowed to omit. An error will occur during the compiling of the software if any input variable setting is omitted. But the output variable setting is allowed to omit.
- **Programming Example**
  - The data types of variables *SIN _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is -0.54402111088937 if the value of *SIN _In* is 10 when *SIN _EN* changes to TRUE. The value of *Out1* is 0.54402111088937 as *SIN _In* is -10.
    - Variable 1

Variable name	Data type	Current value
SIN_EN	BOOL	TRUE
SIN_In	INT	10
Out1	LREAL	-0.54402111088937

    - Variable 2

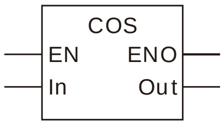
Variable name	Data type	Current value
SIN_EN	BOOL	TRUE
SIN_In	INT	-10
Out1	LREAL	0.54402111088937

    - The program



8.8.13 COS

FB/FC	Explanation	Applicable model
FC	COS is used to get the cosine of a number and the result is output to *Out*. The unit of *In* is radian.	DVP15MC11T



- **Parameters**

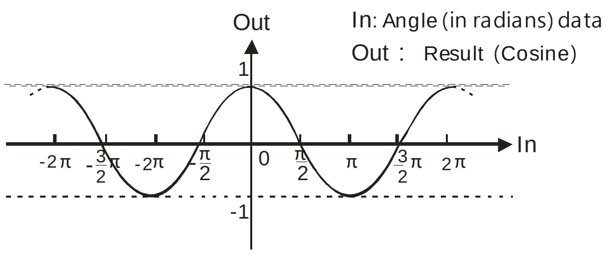
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Radians to process	Input	Radians to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result	-1.00000000000000~ 1.00000000000000

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - COS is used to calculate the cosine of the input parameter *In* and the result is output to *Out*.



  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
- **Programming Example**
  - The data types of variables *COS _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is -0.839071529076452 if the value of *COS _In* is 10 when *COS _EN* changes to TRUE. The value of *Out1* is -0.839071529076452 as *COS _In* is -10.

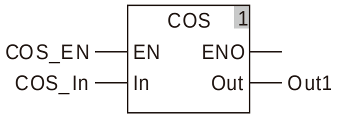
➢ Variable 1

Variable name	Data type	Current value
COS_EN	BOOL	TRUE
COS_In	INT	10
Out1	LREAL	-0.839071529076452

➢ Variable 2

Variable name	Data type	Current value
COS_EN	BOOL	TRUE
COS_In	INT	-10
Out1	LREAL	-0.839071529076452

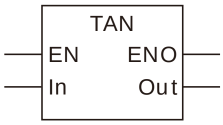
➢ The program



8

8.8.14 TAN

FB/FC	Explanation	Applicable model
FC	TAN is used to get the tangent of a number and the result is output to *Out*. The unit of *In* is radian.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Radians to process	Input	Radians to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result	Depends on the data type of the variable that the output parameter is connected to.

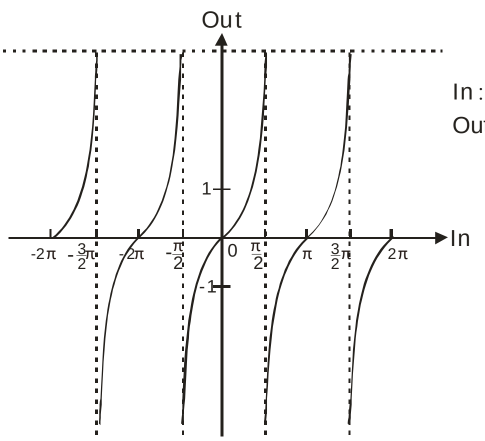
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ TAN is used to calculate the tangent of the input parameter *In* and the result is output to *Out*.



8

- Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**

  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**

  - The data types of variables *TAN _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 0.648360827459087 if the value of *TAN _In* is 10 when *TAN _EN* changes to TRUE. The value of *Out1* is -0.648360827459087 as *TAN _In* is -10.

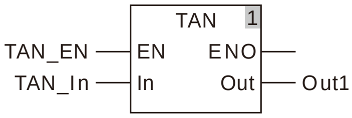
    - Variable 1

Variable name	Data type	Current value
TAN_EN	BOOL	TRUE
TAN_In	INT	10
Out1	LREAL	0.648360827459087

    - Variable 2

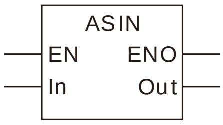
Variable name	Data type	Current value
TAN_EN	BOOL	TRUE
TAN_In	INT	-10
Out1	LREAL	-0.648360827459087

    - The program



8.8.15 ASIN

FB/FC	Explanation	Applicable model
FC	ASIN is used to get the arc sine of a number and the result is output to *Out*. The unit of *Out* is radian.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result	-π/2 ~ π/2

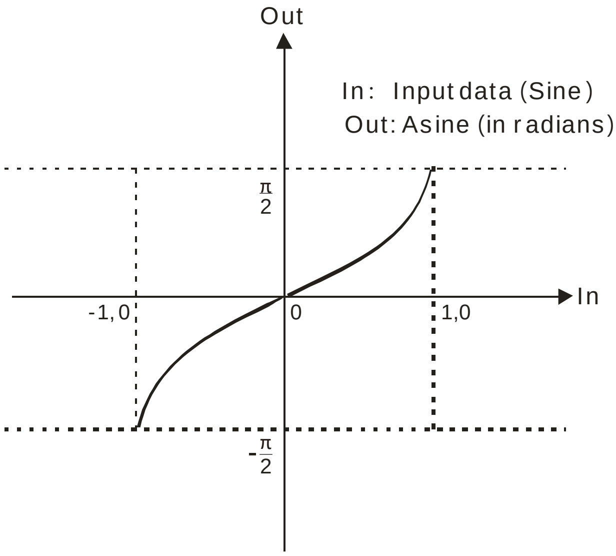
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - ASIN is used to calculate the arc sine of the input parameter *In* and the result is output to *Out*.

8



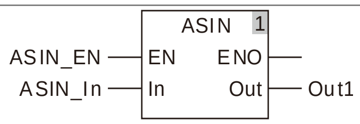
- Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**

  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
  - The value of *Out* varies between -π/2 and π/2 when the value of *In* changes between -1.0 and 1.0. The instruction will not go to the error state if the value of *In* is out of -1.0 ~1.0 and the value of *Out* is nonnumeric as shown in the following table and program.

The variable table and program

Variable name	Data type	Current value
ASIN_EN	BOOL	TRUE
ASIN_In	REAL	2.0
Out1	LREAL	1.#QNAN



- **Programming Example**

  - The data types of variables *ASIN_In* and *Out1* are REAL and LREAL respectively. The value of *Out1* is 1.5707963267949 if the value of *ASIN_In* is 1.0 when *ASIN_EN* changes to TRUE. The value of *Out1* is -1.5707963267949 as *ASIN_In* is -1.0.

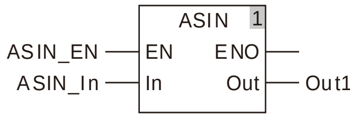
➢ Variable 1

Variable name	Data type	Current value
ASIN_EN	BOOL	TRUE
ASIN_In	REAL	1.0
Out1	LREAL	1.5707963267949

➢ Variable 2

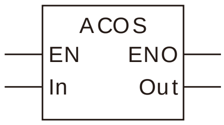
Variable name	Data type	Current value
ASIN_EN	BOOL	TRUE
ASIN_In	REAL	-1.0
Out1	LREAL	-1.5707963267949

➢ The program



8.8.16 ACOS

FB/FC	Explanation	Applicable model
FC	ACOS is used to get the arc cosine of a number and the result is output to *Out*. The unit of *Out* is radian.	DVP15MC11T



- **Parameters**

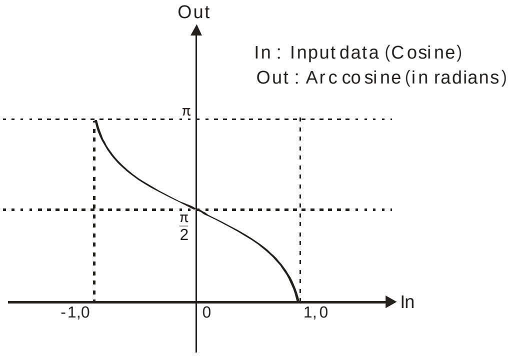
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result	0 ~ π

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - ACOS is used to calculate the arc cosine of the input parameter *In* and the result is output to *Out*.



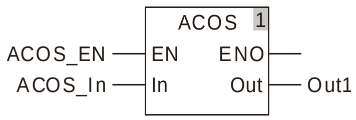
  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

8

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
  - The value of *Out* varies between 0 and π when the value of *In* changes between -1.0 and 1.0. The instruction will not go to the error state if the value of *In* is out of -1.0 ~1.0 and the value of *Out* is nonnumeric.

The variable table and program

Variable name	Data type	Current value
ACOS_EN	BOOL	TRUE
ACOS_In	REAL	2.0
Out1	LREAL	1.#QNAN



- **Programming Example**
  - The data types of variables *ACOS_In* and *Out1* are REAL and LREAL respectively. The value of *Out1* is 0 if the value of *ACOS_In* is 1.0 when *ACOS_EN* changes to TRUE. The value of *Out1* is 3.14159265358979 as *ACOS_In* is -1.0.

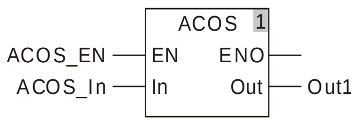
➢ Variable

Variable name	Data type	Current value
ACOS_EN	BOOL	TRUE
ACOS_In	REAL	1.0
Out1	LREAL	0

➢ Variable

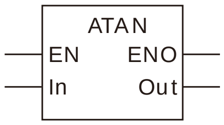
Variable name	Data type	Current value
ACOS_EN	BOOL	TRUE
ACOS_In	REAL	-1.0
Out1	LREAL	3.14159265358979

➢ The program



8.8.17 ATAN

FB/FC	Explanation	Applicable model
FC	ATAN is used to find the arc tangent of a number and the result is output to *Out*. The unit of *Out* is radian.	DVP15MC11T



- **Parameters**

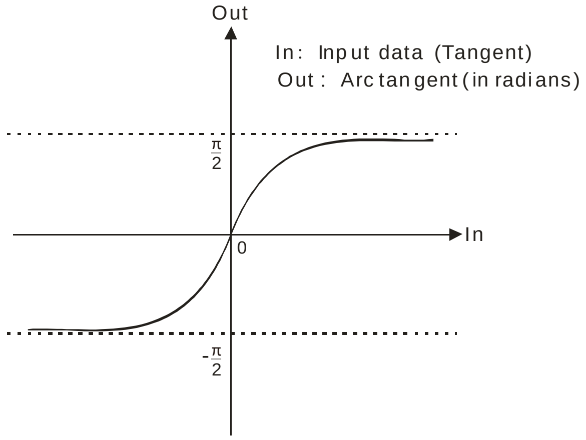
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result	-π/2 ~ π/2

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - ATAN is used to calculate the arc tangent of the input parameter *In* and the result is output to *Out*.



  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
  - The output value of *Out* is -π/2 if the input value of *In* is -∞. The output value of *Out* is π/2 if the input value of *In* is +∞.

- **Programming Example**
  - The data types of variables *ATAN_In* and *Out1* are REAL and LREAL respectively. The value of *Out1* is 0.785398163397448 if the value of *ATAN_In* is 1.0 when *ATAN_EN* changes to TRUE. The value of *Out1* is -0.785398163397448 as *ATAN_In* is -1.0.

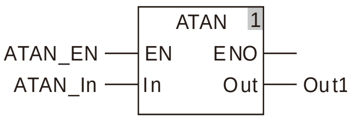
    ➢ Variable 1

Variable name	Data type	Current value
ATAN_EN	BOOL	TRUE
ATAN_In	REAL	1.0
Out1	LREAL	0.785398163397448

    ➢ Variable 2

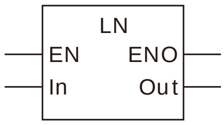
Variable name	Data type	Current value
ATAN_EN	BOOL	TRUE
ATAN_In	REAL	-1.0
Out1	LREAL	-0.785398163397448

    ➢ The program



8.8.18 LN

FB/FC	Explanation	Applicable model
FC	LN is used to find the natural logarithm of a number and the result is output to *Out*.	DVP15MC11T



- **Parameters**

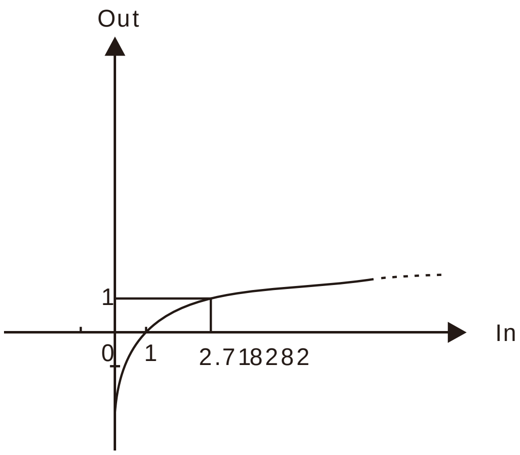
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Logarithm	Output	The natural logarithm of *In*	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - LN is used to calculate the natural logarithm of the input parameter *In,* that is the logarithm with e （e=2.718282） as the base, and the result is output to *Out*.



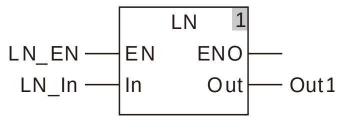
- Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**

  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
  - The output value of *Out* is nonnumeric when the input value of *In* is a non-positive number as shown in the following table.

    The variable table and program

Variable name	Data type	Current value
LN_EN	BOOL	TRUE
LN_In	REAL	-2.0
Out1	LREAL	1.#QNAN



- **Programming Example**

  - The data types of variables *LN_In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 0.0 if the value of *LN_In* is 1 when *LN_EN* changes to TRUE. The value of *Out1* is 1.00000005734143 as *LN_In* is 2.718282.

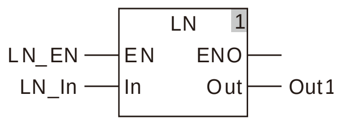
➢ Variable 1

Variable name	Data type	Current value
LN_EN	BOOL	TRUE
LN_In	INT	1
Out1	LREAL	0.0

➢ Variable 2

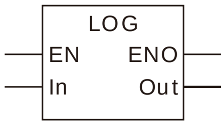
Variable name	Data type	Current value
LN_EN	BOOL	TRUE
LN_In	REAL	2.718282
Out1	LREAL	1.00000005734143

➢ The program



8.8.19 LOG

FB/FC	Explanation	Applicable model
FC	LOG is used to find the base-10 logarithm of a number and the result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to.
Out	Logarithm	Output	The base-10 logarithm	Depends on the data type of the variable that the output parameter is connected to.

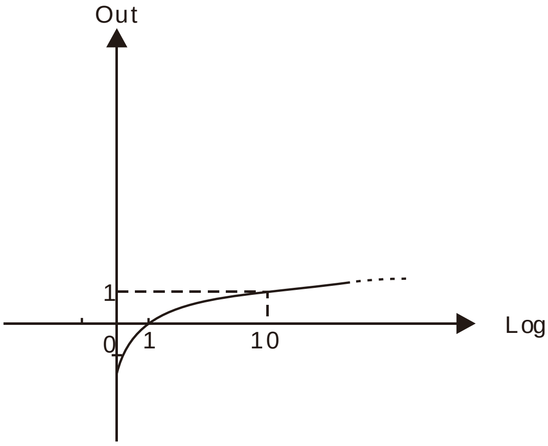
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ LOG is used to calculate the base-10 logarithm of the input parameter *In* and the result is output to *Out*.



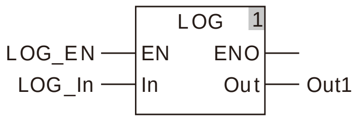
- Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

Precautions for Correct Use

- The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
- The output value of *Out* is a nonnumeric value when the input value of *In* is a non-positive number as shown in the following table.

The variable table and program

Variable name	Data type	Current value
LOG_EN	BOOL	TRUE
LOG_In	REAL	-2.0
Out1	LREAL	1.#QNAN



Programming Example

- The data types of variables *LOG_In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 0.0 if the value of *LOG_In* is 1 when LOG_EN changes to TRUE. The value of *Out1* is 1.0 as *LOG_In* is 10.

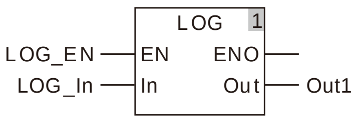
  - Variable 1

Variable name	Data type	Current value
LOG_EN	BOOL	TRUE
LOG_In	INT	1
Out1	LREAL	0.0

  - Variable 2

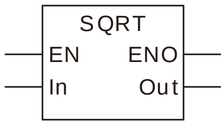
Variable name	Data type	Current value
LOG_EN	BOOL	TRUE
LOG_In	INT	10
Out1	LREAL	1.0

  - The program



8.8.20 SQRT

FB/FC	Explanation	Applicable model
FC	SQRT is used to calculate the square root of a number and the result is output to *Out*.	DVP15MC11T



- **Parameters**

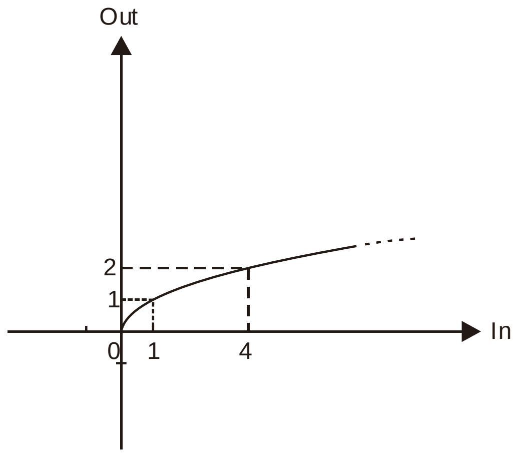
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Number to process	Input	Number to process	Depends on the data type of the variable that the input parameter is connected to. And it is a non-negative number.
Out	Square root	Output	Square root	Depends on the data type of the variable that the output parameter is connected to. And it is a non-negative number.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - SQRT is used to calculate the square root of *In* and the result is output to *Out*.



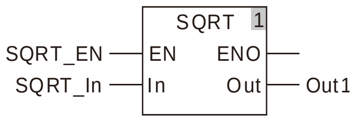
- Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**

  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
  - The output value of *Out* is a nonnumeric value when the input value of *In* is a negative number.

The variable table and program

Variable name	Data type	Current value
SQRT_EN	BOOL	TRUE
SQRT_In	REAL	-2.0
Out1	LREAL	1.#QNAN



- **Programming Example**

  - The data types of variables *SQRT _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 4.0 if the value of *SQRT _In* is 16 when *SQRT _EN* changes to TRUE. The value of *Out1* is 10.0 as *SQRT_In* is 100.

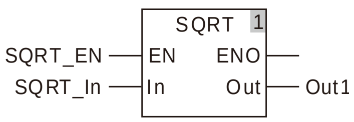
    - Variable 1

Variable name	Data type	Current value
SQRT_EN	BOOL	TRUE
SQRT_In	INT	16
Out1	LREAL	4.0

- Variable 2

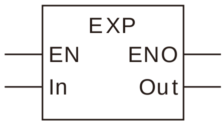
Variable name	Data type	Current value
SQRT_EN	BOOL	TRUE
SQRT_In	INT	100
Out1	LREAL	10.0

- The program



8.8.21 EXP

FB/FC	Explanation	Applicable model
FC	EXP is used to perform the operation with e as the base number and *In* as the exponent. The result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Exponent	Input	Exponent	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Operation result with the base number e and exponent *In*	Depends on the data type of the variable that the output parameter is connected to. And it is a non-negative number.

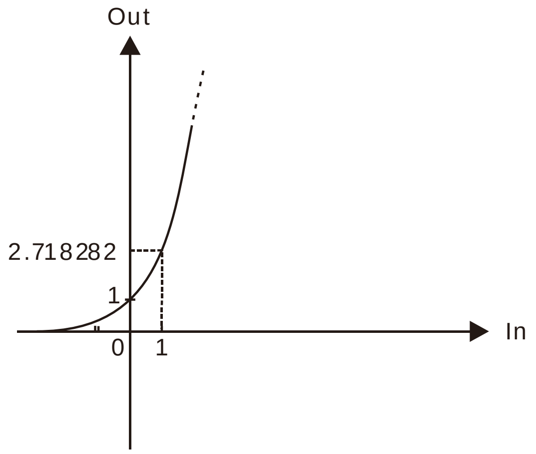
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - EXP is used to perform the operation with e （e=2.718282） as the base number and *In* as the exponent. The result is output to *Out*.



- Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.
  - When the value of *In* is 0, +∞, -∞ and a nonnumeric value, the corresponding output values of *Out* is listed in the following table.

In	Out
0	1.0
+∞	+∞
-∞	0.0
nonnumeric	nonnumeric

- **Programming Example**
  - The data types of *EXP _In* and *Out1* are INT and LREAL respectively. The value of *Out1* is 1.0 if the value of *EXP _In* is 0 when *EXP _EN* changes to TRUE. And the value of *Out1* is 2.71828182845905 as *EXP _In* is 1.

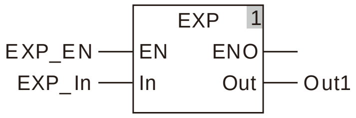
➢ Variable 1

Variable name	Data type	Current value
EXP_EN	BOOL	TRUE
EXP_In	INT	0
Out1	LREAL	1.0

➢ Variable 2

Variable name	Data type	Current value
EXP_EN	BOOL	TRUE
EXP_In	INT	1
Out1	LREAL	2.71828182845905

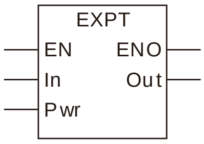
➢ The program



8

8.8.22 EXPT

FB/FC	Explanation	Applicable model
FC	EXPT is used to perform the exponentiation operation with *In* as the base number and *Pwr* as the exponent. The result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Base number	Input	Base number	Depends on the data type of the variable that the input parameter is connected to.
Pwr	Exponent	Input	Exponent	Depends on the data type of the variable that the input parameter is connected to.
Out	Calculation result	Output	Operation result with *In* as the base number and *Pwr as* the exponent	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Pwr		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - EXPT is used to perform the exponentiation operation with *In* as the base number and *Pwr* as the exponent. And the result is output to *Out*.
  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**

  - The data types of variables *EXPT _In* and *EXPT_Pwr* are both INT with their respective values 10 and 2. The data type of *Out1* is LREAL. Then the value of *Out1* is 100.0 when *EXPT _EN* changes to TRUE. The value of *Out1* is 100.0 as the values of *EXPT _In* and *EXPT_Pwr* are -10 and 2 respectively.

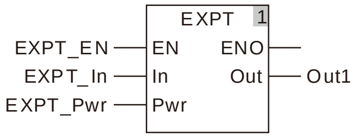
    - Variable 1

Variable name	Data type	Current value
EXPT_EN	BOOL	TRUE
EXPT_In	INT	10
EXPT_Pwr	INT	2
Out1	LREAL	100.0

    - Variable 2

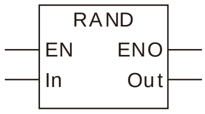
Variable name	Data type	Current value
EXPT_EN	BOOL	TRUE
EXPT_In	INT	-10
EXPT_Pwr	INT	2
Out1	LREAL	100.0

    - The program



8.8.23 RAND

FB/FC	Explanation	Applicable model
FC	RAND is used to generate a random number.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Reserved	Input	Reserved	Depends on the data type of the variable that the input parameter is connected to.
Out	Random number	Output	Random number	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1								●												
Out												●								

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

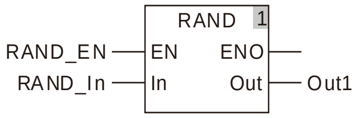
- **Function Explanation**
  - RAND is used to generate a random number and the result is output to *Out*, within the range 0~32767.
  - The input value does not have any effect on the random number to generate. But the value must be input for *In*.
  - To get the random number within a specific range, users just need perform the MOD calculation over the generated value and get the remainder. For example, the random number between 0 and10 can be generated by writing the program RAND(0) MOD10.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**

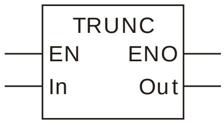
  - A random number is generated by writing RAND(0) as below.
    The variable table and program

Variable name	Data type	Current value
RAND_EN	BOOL	TRUE
RAND_In	INT	0
Out1	DINT	256



8.8.24 TRUNC

FB/FC	Explanation	Applicable model
FC	TRUNC is used to get the integral part of a real number.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Real number to convert	Input	Real number whose integer part is got	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Integral part of a real number	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														●	●					
Out													●							

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - TRUNC is used to get the integral part of a real number and the result is output to *Out*.
  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is only LINT. An error will occur during the compiling of the software if the data type of the output parameter is not LINT.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

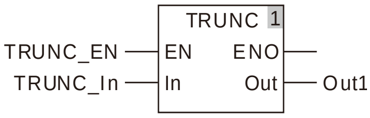
- **Programming Example**
  - The data type of *TRUNC _In* is REAL with the value -5.6. The data type of *Out1* is LINT. Then the value of *Out1* is -5 when *TRUNC _EN* changes to TRUE. And the value of *Out1* is 10 as the values of *TRUNC _In* 10.8.
    - Variable 1

Variable name	Data type	Current value
TRUNC_EN	BOOL	TRUE
TRUNC _In	REAL	-5.6
Out1	LINT	-5

    - Variable 2

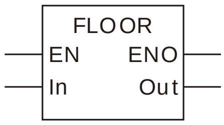
Variable name	Data type	Current value
TRUNC_EN	BOOL	TRUE
TRUNC _In	REAL	10.8
Out1	LINT	10

    - The program



8.8.25 FLOOR

FB/FC	Explanation	Applicable model
FC	FLOOR is used to get the integral part of a real number. The output value is the integral part of the real number subtracted by 1 if the input real number is a negative number.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Real number to convert	Input	Real number whose integer part is got	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Integer part of a real number	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														●	●					
Out													●							

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ FLOOR is used to get the integral part of a real number and the result is output to *Out*. The output value is the integral part of the real number if the input real number is a positive number. For example, the output value is 3 if the input value is 3.5. The output value is the integral part of the real number subtracted by 1 if the input real number is a negative number. For example, the output value is -4 if the input value is -3.5.

  ■ Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LINT. An error will occur during the compiling of the software if the data type of the output parameter is not LINT.

- **Precautions for Correct Use**

  ■ The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**

  - The data type of variable *FLOOR _In* is REAL with the value 5.6*.* The data type of *Out1* is LINT. Then the value of *Out1* is 5 when *FLOOR _EN* changes to TRUE. And the value of *Out1* is -11 as the values of *FLOOR _In* -10.2.

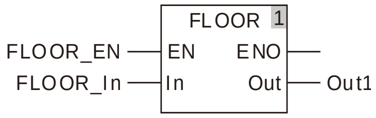
    - Variable 1

Variable name	Data type	Current value
FLOOR_EN	BOOL	TRUE
FLOOR _In	REAL	5.6
Out1	LINT	5

    - Variable 2

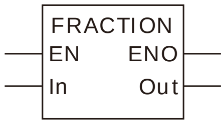
Variable name	Data type	Current value
FLOOR_EN	BOOL	TRUE
FLOOR _In	REAL	-10.2
Out1	LINT	-11

    - The program



8.8.26 FRACTION

FB/FC	Explanation	Applicable model
FC	FRACTION is used to get the fraction part of a real number.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Real number to convert	Input	Real number whose fraction part is got	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Fraction part of a real number	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														●	●					
Out															●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - FRACTION is used to get the fraction part of a real number and the result is output to *Out*. The sign of the result value should be the same as that of the input value.
  - Users can choose different data types for the input parameter in this instruction. But the data type of the output parameter is restricted to LREAL. An error will occur during the compiling of the software if the data type of the output parameter is not LREAL.

- **Precautions for Correct Use**
  - The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

- **Programming Example**

  - The data type of variable *FRACTION _In* is REAL with the value -5.6. The data type of *Out1* is LREAL. Then the value of *Out1* is -0.6 when *FRACTION _EN* changes to TRUE. And the value of *Out1* is 0.8 as the values of *FRACTION _In* 10.8.

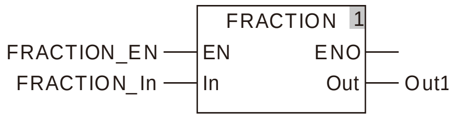
    - Variable 1

Variable name	Data type	Current value
FRACTION_EN	BOOL	TRUE
FRACTION _In	REAL	-5.6
Out1	LREAL	-0.6

    - Variable 2

Variable name	Data type	Current value
FRACTION_EN	BOOL	TRUE
FRACTION _In	REAL	10.8
Out1	LREAL	0.8

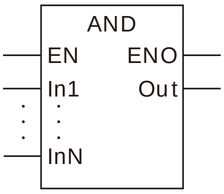
    - The program



8.9 Bit String Instructions

8.9.1 AND

FB/FC	Explanation	Applicable model
FC	AND is used for performing a logical AND operation of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Operands	Input	The number of operands can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	AND operation result of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●											
Out	●	●	●	●	●	●	●	●	●											

Note:

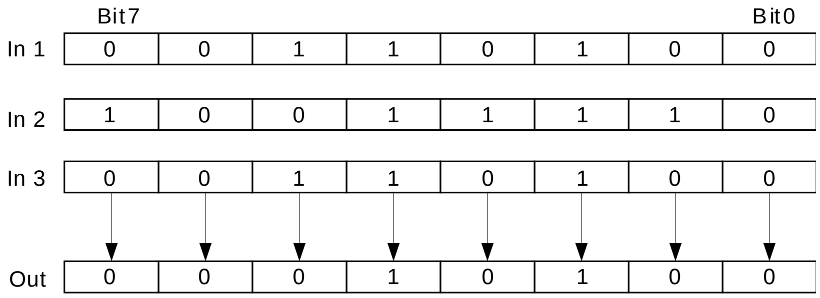
The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

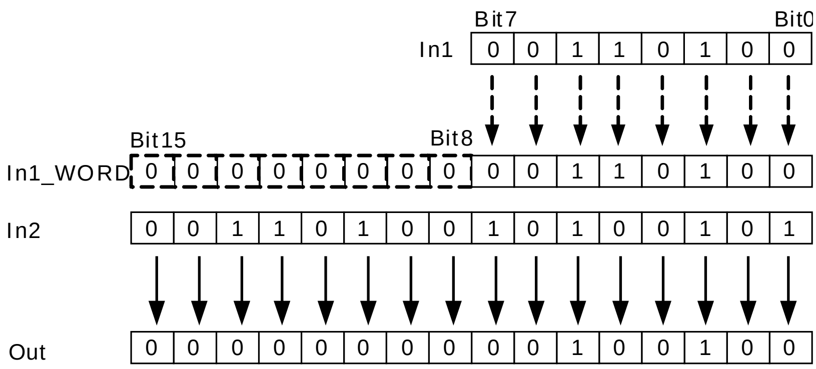
  ■ AND is used for performing a bitwise logical AND operation of two or more variables or constants and the result is output to *Out*. That is *Out* = *In1* & *In2* &...& *InN*

  The operational rule:

  The corresponding bit of the output variable is TRUE when corresponding bits of input variables are all TRUE as shown below. Otherwise, the corresponding bit of the output variable is FALSE.



- *In1~InN* are allowed to be the variables of different data types when none of the data types of input variables are BOOL.
  When *In1* to *InN* are the variables of different data types, take the data type which can include all ranges of the values of *In1~InN* for the operation.
  For example, if the data type of *In1* is BYTE and *In2* is WORD, the data type of *Out* is WORD. In operation, the value of *In1* is converted from BYTE to WORD as shown in the following figure. Bit8~ Bit 15 are complemented and their values are all 0. And then the logical AND of the bit values of *In1* and *In2* is conducted as below.



- If the data type of an input variable is BOOL, the data types of all input and output variables are required to be BOOL. Otherwise, an error will occur in the compiling of the software.

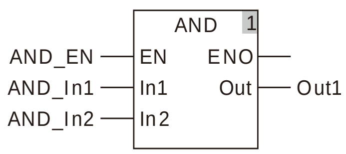
● **Precautions for Correct Use**

The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.

Programming Example

- The data types of AND_In1, AND_In2 and Out1 are all BYTE. The values of AND_In1 and AND_In2 are 10 and 50 respectively and the value of Out1 is 2 when AND_EN is TRUE.
  - **The variable table and program**

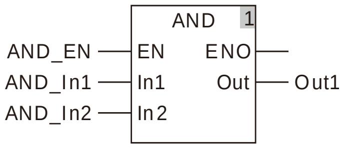
Variable name	Data type	Current value
AND_EN	BOOL	TRUE
AND_In1	BYTE	10
AND_In2	BYTE	50
Out1	BYTE	2



- The data types of AND_In1, AND_In2 and Out1 are BYTE, WORD and WORD respectively. The values of AND_In1 and AND_In2 are 255 and 256 respectively and the value of Out1 is 0 when AND_EN is TRUE.

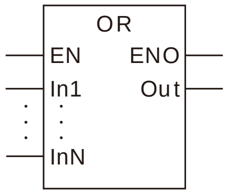
- **The variable table and program**

Variable name	Data type	Current value
AND_EN	BOOL	TRUE
AND_In1	BYTE	255
AND_In2	WORD	256
Out1	WORD	0



8.9.2 OR

FB/FC	Explanation	Applicable model
FC	OR is used for performing a logical OR operation of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Operand	Input	The number of operands can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	OR operation result of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●											
Out	●	●	●	●	●	●	●	●	●											

Note:

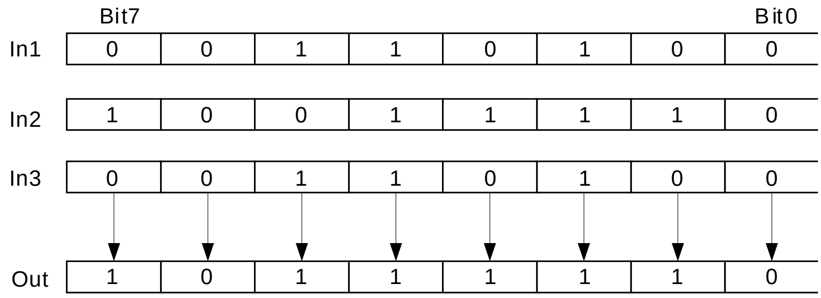
The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ OR is used for performing a bitwise logical OR operation of two or more variables or constants and the result is output to *Out*. That is *Out*= *In1* OR *In2* OR…OR *InN.*

  The operational rule:

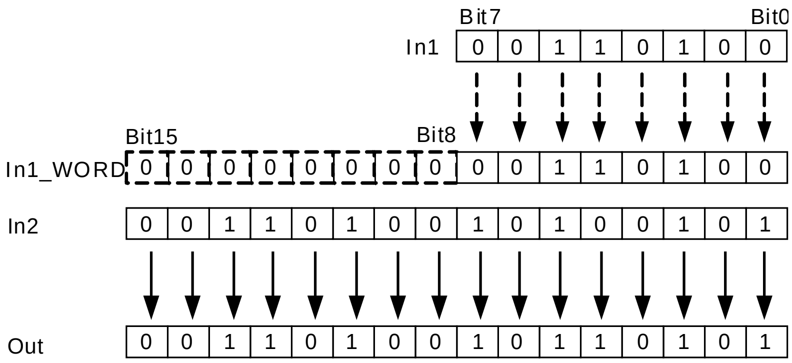
  When corresponding bits of all input variables are all FALSE, the corresponding bit of the output variable is FALSE. Otherwise, the corresponding bit of the output variable is TRUE.



- *In1~InN* are allowed to be the variables of different data types when none of the data types of input variables are BOOL.

  When *In1* to *InN* are the variables of different data types, take the data type which can include all ranges of the values of *In1~InN* for the operation.

  For example, if the data type of *In1* is BYTE and *In2* is WORD, the data type of *Out* is WORD. In operation, the value of *In1* is converted from BYTE to WORD as shown in the following figure. Bit8~ Bit 15 are complemented and their values are all 0. And then the logical OR of the bit values of *In1* and *In2* is conducted as below.



- If the data type of an input variable is BOOL, the data types of all input and output variables are required to be BOOL. Otherwise, an error will occur in the compiling of the software.

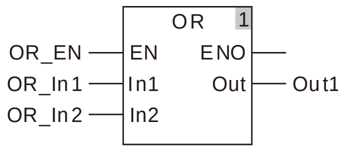
- **Precautions for Correct Use**

  The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.

Programming Example

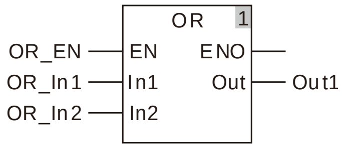
- The data types of OR_In1, OR_In2 and Out1 are all BYTE. The values of OR_In1 and OR_In2 are 10 and 50 respectively and the value of Out1 is 58 when OR_EN is TRUE.
  - **The variable table and program**

Variable name	Data type	Current value
OR_EN	BOOL	TRUE
OR_In1	BYTE	10
OR_In2	BYTE	50
Out1	BYTE	58



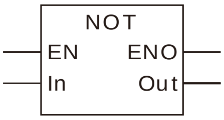
- The data types of OR_In1, OR_In2 and Out1 are BYTE, WORD and WORD respectively. The values of OR_In1 and OR_In2 are 255 and 256 respectively and the value of Out1 is 511 when OR_EN is TRUE.
  - **The variable table and program**

Variable name	Data type	Current value
OR_EN	BOOL	TRUE
OR_In1	BYTE	255
OR_In2	WORD	256
Out1	WORD	511



8.9.3 NOT

FB/FC	Explanation	Applicable model
FC	NOT is used for the NOT operation taking the inverse of a variable or constant.	DVP15MC11T



- **Parameters**

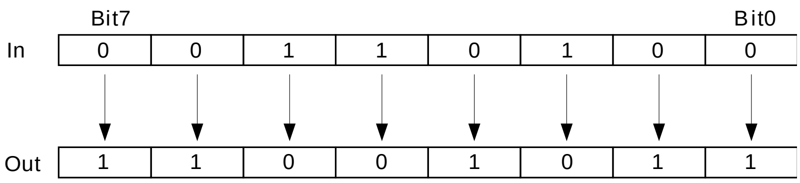
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Operand	Input	Input parameter to take the inverse	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	Not operation result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	●	●	●	●	●	●	●	●	●											
Out	●	●	●	●	●	●	●	●	●											

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - NOT is used for the bitwise NOT operation taking the inverse of the value of a variable or constant and the result is output to *Out*.
    The operational rule:
    If one bit of the input variable is TRUE, the corresponding bit of the output variable is FALSE. If one bit of the input variable is FALSE, the corresponding bit of the output variable is TRUE.



  - The data type of *Out* must be the same as *In*.

- **Precautions for Correct Use**

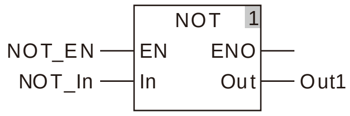
The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.

Programming Example

- The data types of NOT _In and Out1 are both BYTE. The value of In1 is 10 and the value of Out1 is 245 when NOT_EN is TRUE.

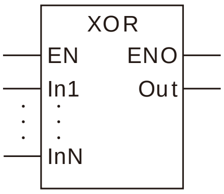
  - **The variable table and program**

Variable name	Data type	Current value
NOT_EN	BOOL	TRUE
NOT _In	BYTE	10
Out1	BYTE	245



8.9.4 XOR

FB/FC	Explanation	Applicable model
FC	XOR is used for the XOR operation of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Operand	Input	The number of operands can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	XOR operation result of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●											
Out	●	●	●	●	●	●	●	●	●											

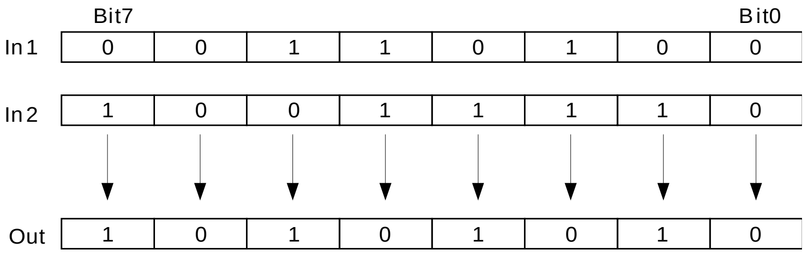
Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

XOR is used for the bitwise XOR operation of two or more variables or constants and the result is output to *Out*. That is *Out*= *In1* XOR *In2* XOR…XOR *InN.*

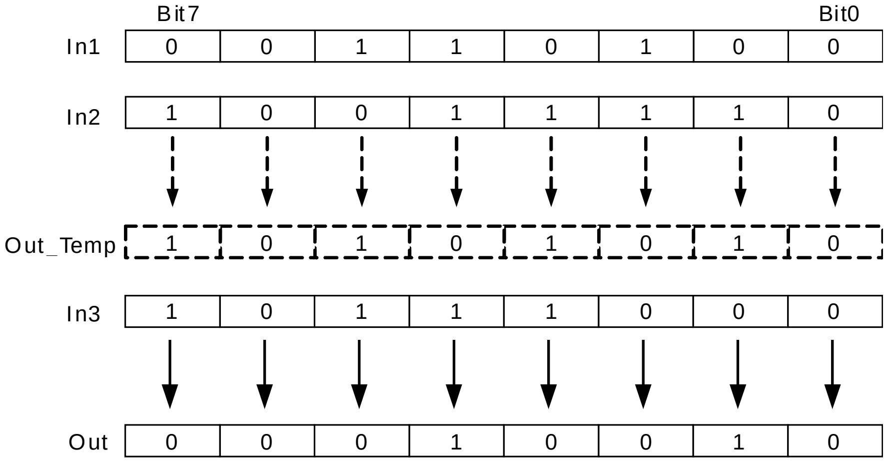
The operational rule for XOR of In1 and In2 is shown in the following figure.



- The steps for XOR operation when more than 2 input parameters exist are:

  The XOR result of In1 and In2 is got first; then the XOR operation of the previous result and In3 is conducted and so on. Finally, the XOR operation of the previous XOR result and InN is processed.

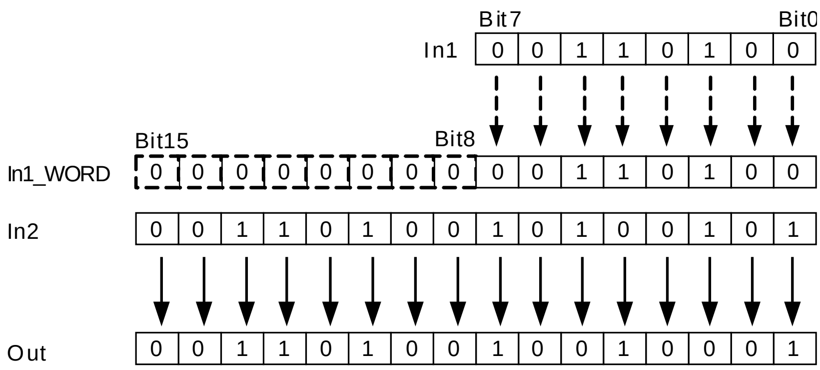
  The XOR result of In1 and In2 is Out_Temp and the XOR result of Out_Temp and In3 is Out as shown below.



- *In1~InN* are allowed to be the variables of different data types when none of the data types of input variables are BOOL.

  When *In1* to *InN* are the variables of different data types, take the data type which can include all ranges of the values of *In1~InN* for the XOR operation.

  For example, if the data type of *In1* is BYTE and *In2* is WORD, the data type of *Out* is WORD. In operation, the value of *In1* is converted from BYTE to WORD as shown in the following figure. (Bit8~ Bit 15 are supplemented and their values are all 0.) And then the logical XOR of the bit values of *In1* and *In2* is conducted as below.



- If the data type of an input variable is BOOL, the data types of all input and output variables are required to be BOOL. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**

  The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.

Programming Example

- The data types of XOR_In1, XOR_In2 and Out1 are all BYTE. The values of XOR_In1 and XOR_In2 are 10 and 50 and the value of Out1 is 56 when XOR_EN is TRUE as shown in Variable 1.

  The data types of XOR_In1, XOR_In2 and Out1 are BYTE, WORD and WORD. The values of XOR_In1 and XOR_In2 are 255 and 256 and the value of Out1 is 511 when XOR_EN is TRUE as shown in Variable 2.

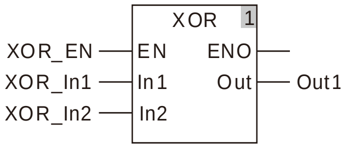
- **Variable 1**

Variable name	Data type	Current value
XOR_EN	BOOL	TRUE
XOR _In1	BYTE	10
XOR _In2	BYTE	50
Out1	BYTE	56

- **Variable 2**

Variable name	Data type	Current value
XOR_EN	BOOL	TRUE
XOR_In1	BYTE	255
XOR_In2	WORD	256
Out1	WORD	511

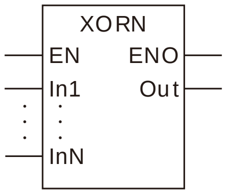
- **The program**



8

8.9.5 XORN

FB/FC	Explanation	Applicable model
FC	XORN is used for an XORN operation of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Operand	Input	The number of operands can be increased or decreased through the programming software. Maximum: 8. Minimum: 2. That is N=2 ~ 8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Operation result	Output	XORN operation result of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●											
Out	●	●	●	●	●	●	●	●	●											

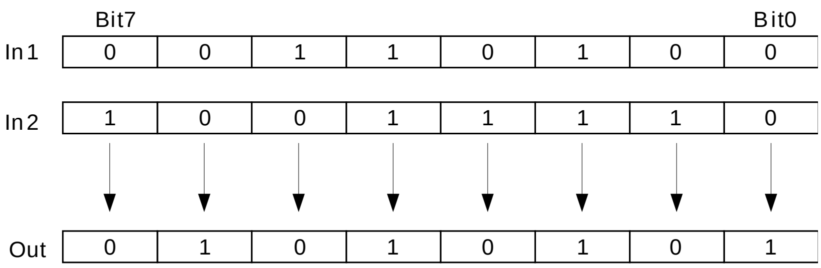
Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

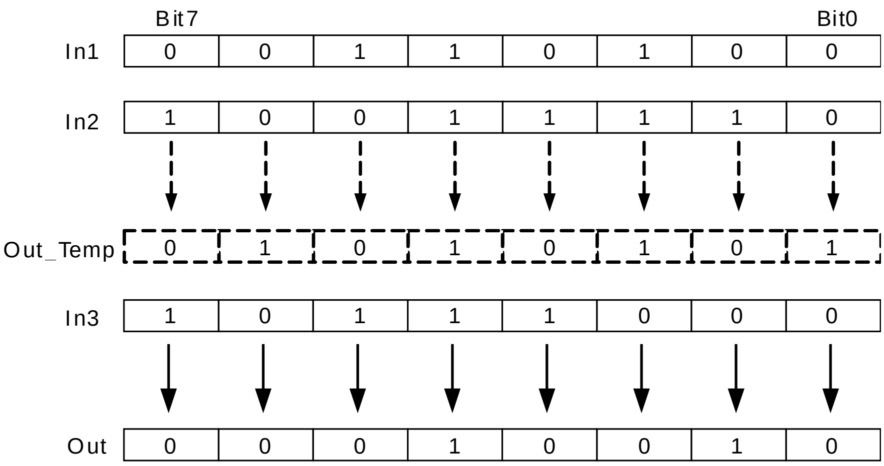
- **Function Explanation**

XORN is used for a bitwise XORN of two or more variables or constants and the result is output to *Out*. That is *Out*= *In1* XORN *In2* XORN…XORN *InN*.

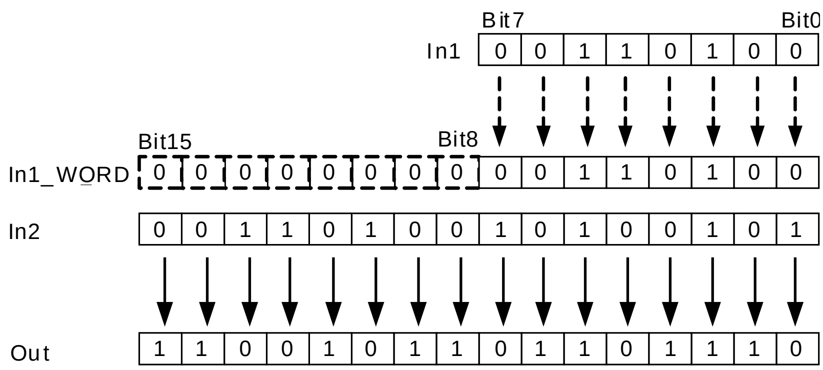
The operational rule for XORN of In1 and In2 is shown in the following figure.



- The steps for XORN operation is for when more than 2 input parameters exist:
  The XORN result of In1 and In2 is got first; then the XORN of the previous result and In3 is conducted and so on. Finally, the XORN of the previous XORN result and InN is processed.
  The XORN result of In1 and In2 is Out_Temp and the XORN result of Out_Temp and In3 is Out as shown below.



- *In1~InN* are allowed to be the variables of different data types when none of the data types of input variables are BOOL.
  When *In1* to *InN* are the variables of different data types, take the data type which can include all ranges of the values of *In1~InN* for the operation.
  For example, if the data type of *In1* is BYTE and *In2* is WORD, the data type of *Out* is WORD. In operation, the value of *In1* is converted from BYTE to WORD as shown in the following figure. (Bit8~ Bit 15 are supplemented and their values are all 0.) And then the logical XORN of the bit values of *In1* and *In2* is conducted as below.



- If the data type of an input variable is BOOL, the data types of all input and output variables are required to be BOOL. Otherwise, an error will occur in the compiling of the software.

- **Precautions for Correct Use**

  The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.

Programming Example

- The data types of XORN_In1, XORN_In2 and Out1 are all BYTE. The values of XORN_In1 and XORN_In2 are 10 and 50 and the value of Out1 is 199 when XORN_EN is TRUE as shown in Variable 1.
  The data types of XORN_In1, XORN_In2 and Out1 are BYTE, WORD and WORD. The values of XORN_In1 and XORN_In2 are 255 and 256 and the value of Out1 is 65535 when XORN _EN is TRUE as shown in Variable 2.

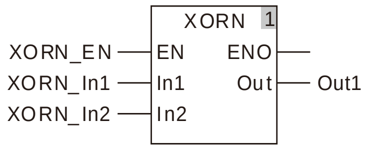
➢ **Variable 1**

Variable name	Data type	Current value
XORN_EN	BOOL	TRUE
XORN _In1	BYTE	10
XORN _In2	BYTE	50
Out1	BYTE	199

➢ **Variable 2**

Variable name	Data type	Current value
XORN _EN	BOOL	TRUE
XORN _In1	BYTE	255
XORN _In2	WORD	256
Out1	WORD	65535

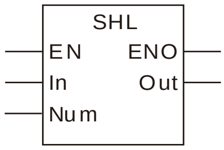
➢ **The program**



8.10 Shift Instructions

8.10.1 SHL

FB/FC	Explanation	Applicable model
FC	SHL is used to shift all bits of a variable or constant by the specified number of bits to the left and the result is output to Out.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to shift	Input	The original data to shift to the left	Depends on the data type of the variable that the input parameter is connected to.
Num	Number to shift	Input	The number of bits by which all bits of the original data are shifted to the left	Depends on the data type of the variable that the input parameter is connected to.
Out	Result	Output	Result from shifting all bits of the original data by the number of bits specified by Num to the left	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
Num						●														
Out	The data type of *Out* must be the same as *In*.																			

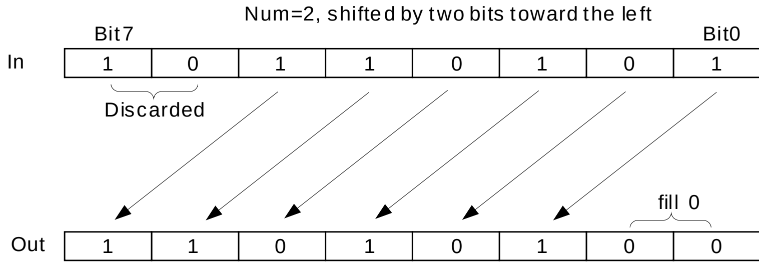
Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

SHL is used to shift all bits of the value of *In* by the number of bits specified by *Num* to the left and the result is output to *Out*.

When *Num*=2, all bits of the value of *In* are shifted by two bits to the left and the values of Bit0~Bit1 are supplemented with 0 and Bit6~Bit7 are discarded as shown in the following figure.



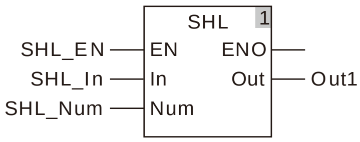
- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The value of *Out* is the same as *In* when the value of *Num* is 0.

Programming Example

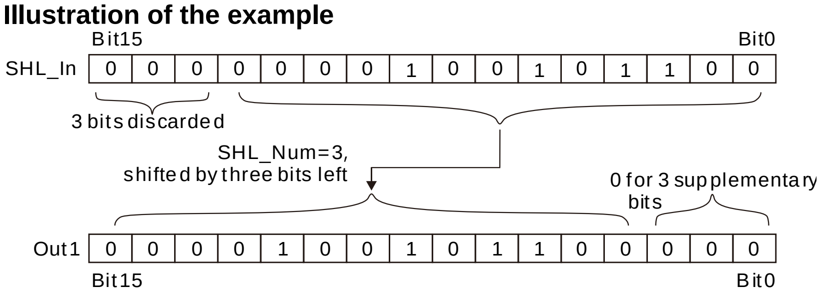
- The data types of SHL_In and SHL_Num are UINT and USINT respectively and their values are 300 and 3 respectively. The data type of Out1 is BYTE and the value of Out1 is 2400 when SHL_EN is TRUE.

➢ **The variable table and program**

Variable name	Data type	Current value
SHL_EN	BOOL	TRUE
SHL_In	UINT	300
SHL_Num	USINT	3
Out1	UINT	2400

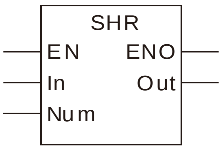


➢ **Illustration of the example**



8.10.2 SHR

FB/FC	Explanation	Applicable model
FC	SHR is used to shift all bits of a variable or constant by the specified number of bits to the right and the result is output to *Out*.	DVP15MC11T



- **Parameters**

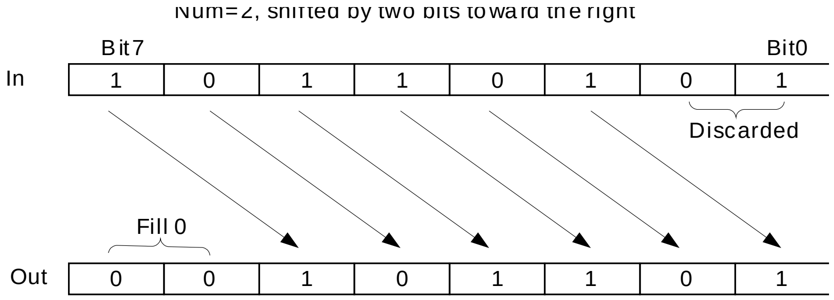
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to shift	Input	The original data to shift to the right	Depends on the data type of the variable that the input parameter is connected to.
Num	Number to shift	Input	The number of bits by which the bits of the original data are shifted to the right	Depends on the data type of the variable that the input parameter is connected to.
Out	Result	Output	Result from shifting all bits of the original data by the number of bits specified by Num to the right	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
Num						●														
Out	The data type of *Out* must be the same as *In*.																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - SHR is used to shift all bits of the value of *In* by the number of bits specified by *Num* to the right and the result is output to *Out*.
  - When *Num*=2, all bits of the value of *In* are shifted by two bits to the right and Bit0~Bit1 of *In* are discarded and the value of Bit6~Bit7 are supplemented with 0 as shown in the following figure.



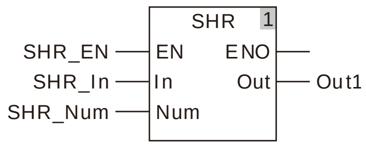
- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - When the value of Num is 0, the value of *Out* is the same as *In*.

Programming Example

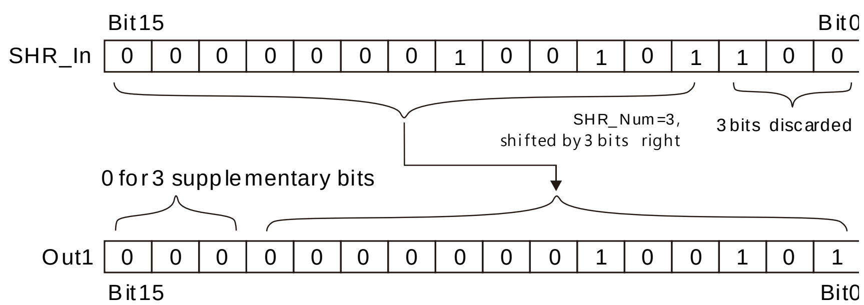
- The data types of SHR_In and SHR_Num are UINT and USINT respectively and their values are 300 and 3 respectively. The data type of Out1 is UINT and the value of Out1 is 37 when SHR_EN is TRUE.

➢ **The variable table and program**

Variable name	Data type	Current value
SHR_EN	BOOL	TRUE
SHR_In	UINT	300
SHR_Num	USINT	3
Out1	UINT	37

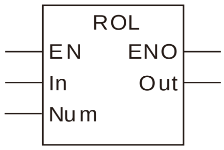


➢ **Illustration of the example**



8.10.3 ROL

FB/FC	Explanation	Applicable model
FC	ROL is used to rotate left all bits of a variable or constant by the specified number of bits and the result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to rotate	Input	The original data to rotate left	Depends on the data type of the variable that the input parameter is connected to.
Num	Number of bits	Input	The number of bits by which the bits of the original data are rotated to the left	Depends on the data type of the variable that the input parameter is connected to.
Out	Result	Output	Result from rotating all bits of the original data by the number of bits specified by Num to the left	Depends on the data type of the variable that the output parameter is connected to.

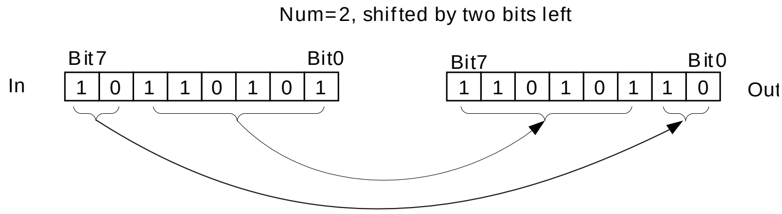
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
Num						●														
Out	The data type of *Out* must be the same as *In*.																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ROL is used to rotate all bits of the value of *In* by the number of bits specified by *Num* to the left and the result is output to *Out*.
  Via ROL, the bits shifted out of the left will shift to the null bits in the right one by one. When *Num*=2, all bits of the value of *In* rotates by two bits to the left. The rotation method is that Bit0~Bit5 are shifted to Bit2~Bit7 respectively, Bit 7 is shifted to Bit1 and Bit 6 is shifted to Bit0.



- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The value of *Out* is the same as *In* when the value of *Num* is 0.
  - The number of bits by which the bits of original data are rotated left is equal to the value of Num MOD In when the value of *Num* is greater than the number of bits of the value of *In*.

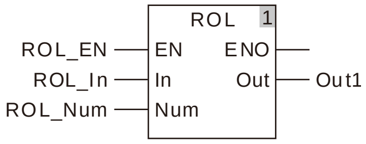
    For example, if the data type of *In* is BYTE, the value of *out* when Num=USINT#1 is the same for when Num=USINT#9.

Programming Example

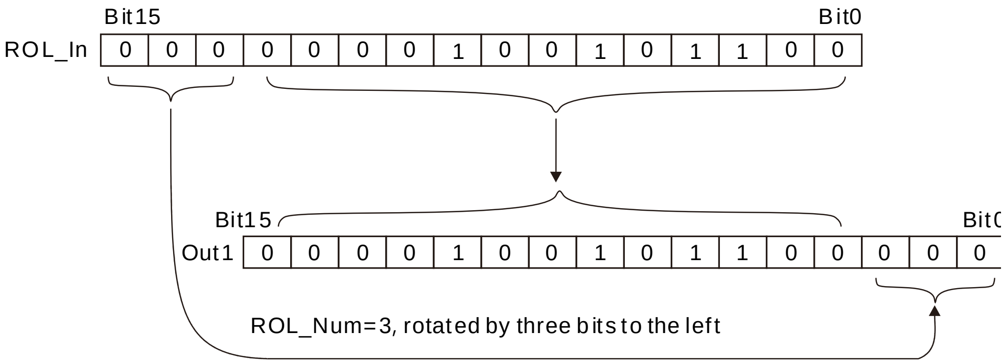
- The data types of ROL_In and ROL_Num are UINT and USINT respectively and their values are 300 and 3 respectively. The data type of Out1 is BYTE and the value of Out1 is 2400 when ROL_EN is TRUE.

  - **The variable table and program**

Variable name	Data type	Current value
ROL_EN	BOOL	TRUE
ROL _In	UINT	300
ROL _Num	USINT	3
Out1	UINT	2400

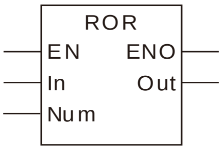


  - **Illustration of the example**



8.10.4 ROR

FB/FC	Explanation	Applicable model
FC	ROR is used to rotate all bits of a variable or constant by the specified number of bits to the right and the result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to rotate	Input	The original data to rotate to the right	Depends on the data type of the variable that the input parameter is connected to.
Num	Number of bits	Input	The number of bits by which the bits of data are rotated to the right	Depends on the data type of the variable that the input parameter is connected to.
Out	Result	Output	Result from rotating all bits of the original data by the number of bits specified by Num to the right	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●	●	●	●	●											
Num						●														
Out	The data type of *Out* must be the same as *In*.																			

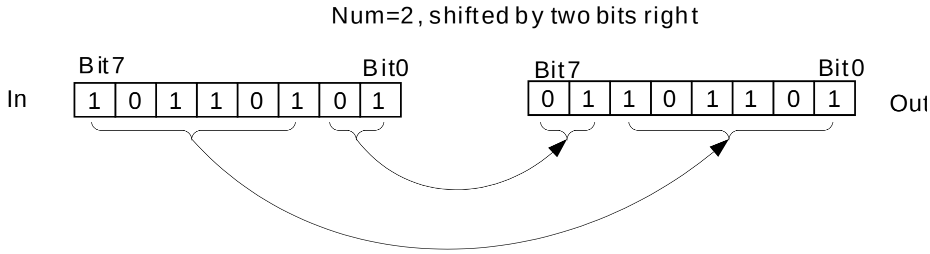
Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ROR is used to rotate all bits of the value of *In* by the number of bits specified by *Num* to the right and the result is output to *Out*.

  Via ROR, the bits shifted out of the right will shift to the null bits in the left one by one. When *Num*=2, all bits of the value of *In* rotates by two bits to the right. The rotation method is that Bit2~Bit7 are shifted to Bit0~Bit5 respectively, Bit0 is shifted to Bit6 and Bit1 is shifted to Bit7.

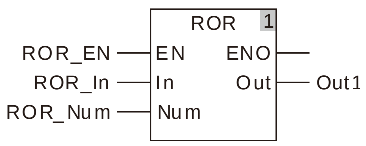


- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if any input variable is omitted. But the output variable is allowed to omit.
  - The value of *Out* is the same as *In* when the value of *Num* is 0.
  - The number of bits by which the bits of data are rotated to the right is equal to the value of Num MOD In when the value of *Num* is greater than the number of bits of the value of *In*.
    For example, if the data type of *In* is BYTE, the value of *out* when Num=USINT#1 is the same for when Num=USINT#9.

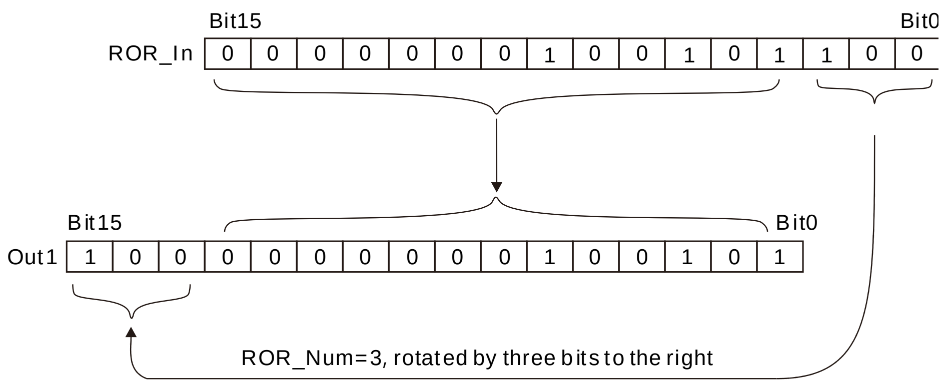
Programming Example

- The data types of ROR_In and ROR_Num are UINT and USINT respectively and their values are 300 and 3 respectively. The data type of Out1 is BYTE and the value of Out1 is 32805 when ROR_EN is TRUE.
  - **The variable table and program**

Variable name	Data type	Current value
ROR_EN	BOOL	TRUE
ROR_In	UINT	300
ROR_Num	USINT	3
Out1	UINT	32805



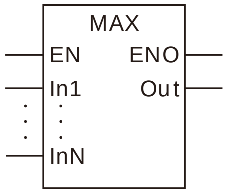
  - **Illustration of the example**



8.11 Selection Instructions

8.11.1 MAX

FB/FC	Explanation	Applicable model
FC	Max is used for finding the largest value of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The comparison data can be added or removed while the program is being written. The maximum number of comparison data is 8. N=2~8	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	The largest value of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

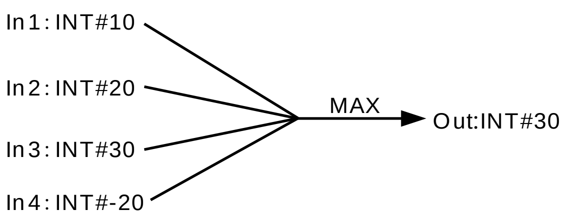
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ The Max instruction finds the largest value of two or more variables or constants and the largest value is output to *Out*.



- When the data types of input variables are not BOOL, TIME, DATE, TOD or STRING, the input parameters *In1~InN* are allowed to be the variables of different data types.

- When the data types of input variables are one of BOOL, TIME, DATE, TOD and STRING, all the input variables and output variable should be of the data type. For example, if the data type of *In1* is TIME, the data type of *In2~InN* must be TIME. Otherwise, an error will occur during the compiling of the software.

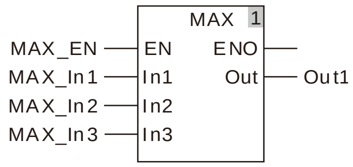
- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.
  - The length of the data type of the output variable must contain the length of all input parameters. Otherwise, an error will occur during the compiling of the software

Programming Example

- The data types of MAX_In1, MAX_In2 and MAX_In3 are INT, UINT and DINT respectively. The data type of Out1 is DINT. If the values of MAX_In1, MAX_In2 and MAX_In3 are -10, 50 and 100 respectively, the value of Out1 is 100 when MAX_EN is TRUE.

  ➢ **The variable table and program**

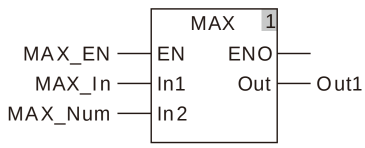
Variable name	Data type	Current value
MAX_EN	BOOL	TRUE
MAX_In1	INT	- 10
MAX_In2	UINT	50
MAX_In3	DINT	100
Out1	DINT	100



- The data types of MAX_In1 and MAX_In2 are TIME. The data type of Out1 is TIME.
  If the values of MAX_In1 and MAX_In2 are T#1ms and T#50ms respectively, the value of Out1 is T#50ms when MAX_EN is TRUE.

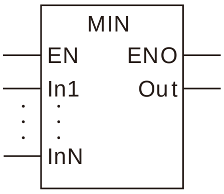
  ➢ **The variable table and program**

Variable name	Data type	Current value
MAX_EN	BOOL	TRUE
MAX_In1	TIME	T#1ms
MAX_In2	TIME	T#50ms
Out1	TIME	T#50ms



8.11.2 MIN

FB/FC	Explanation	Applicable model
FC	MIN is used for finding the smallest value of two or more variables or constants.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Comparison data	Input	The comparison data can be added or removed while the program is being written. The maximum number of comparison data is 8. N=2~8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Comparison result	Output	The smallest value of In1 ~ InN	Depends on the data type of the variable that the output parameter is connected to.

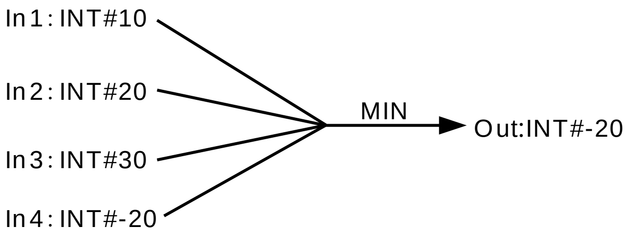
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

Function Explanation

- The MIN instruction finds the smallest value of two or more variables and constants and the smallest value is output to *Out*.



- When the data types of input variables are not BOOL, TIME, DATE, TOD or STRING, the input parameters *In1~InN* are allowed to be the variables of different data types.

- When the data types of input variables are one of BOOL, TIME, DATE, TOD and STRING, all the input variables and output variable should be of the data type. For example, if the data type of *In1* is TIME, the data type of *In2~InN* must be TIME. Otherwise, an error will occur during the compiling of the software.

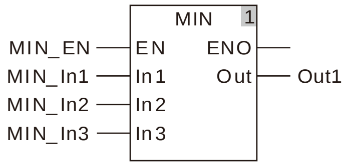
- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.
  - The length of the data type of the output variable must contain the length of all input parameters. Otherwise, an error will occur during the compiling of the software.

Programming Example

- The data types of MIN_In1, MIN_In2 and MIN_In3 are INT, UINT and DINT respectively. The data type of Out1 is DINT. If the values of MIN_In1, MIN_In2 and MIN_In3 are -10, 50 and 100 respectively, the value of Out1 is -10 when MIN_EN is TRUE.

➢ **The variable table and program**

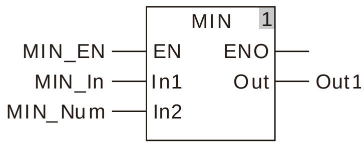
Variable name	Data type	Current value
MIN_EN	BOOL	TRUE
MIN_In1	INT	- 10
MIN_In2	UINT	50
MIN_In3	DINT	100
Out1	DINT	- 10



- The data types of MIN_In1 and MIN_In2 are TIME. The data type of Out1 is TIME.
  If the values of MIN_In1 and MIN_In2 are T#1ms and T#50ms respectively, the value of Out1 is T#1ms when MIN_EN is TRUE.

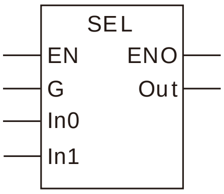
➢ **The variable table and program**

Variable name	Data type	Current value
MIN_EN	BOOL	TRUE
MIN_In1	TIME	T#1ms
MIN_In2	TIME	T#50ms
Out1	TIME	T#1ms



8.11.3 SEL

FB/FC	Explanation	Applicable model
FC	SEL is used for selecting one of two variables or constants and the selected value is output to *Out*.	DVP15MC11T



- **Parameters**

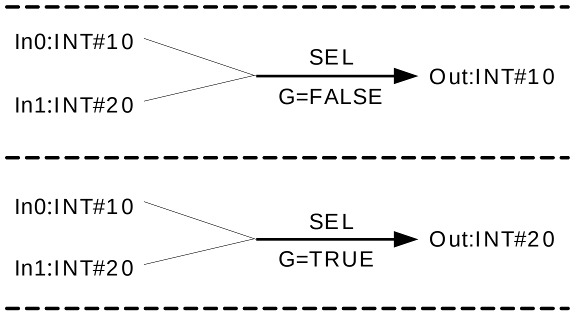
Parameter name	Meaning	Input/ Output	Description	Valid range
G	Gate	Input	In0 is selected when G is FALSE; In1 is selected when G is TRUE.	Depends on the data type of the variable that the input parameter is connected to.
In0 and In1	Selections	Input	Data to be selected	Depends on the data type of the variable that the input parameter is connected to.
Out	Selection result	Output	Selection result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
G	●																			
In0 and In1	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - According to the selection condition G, the SEL instruction selects one of two variables or constants and the selection result is output to *Out*.



- When the data types of input variables are not BOOL, TIME, DATE, TOD or STRING, the input parameters *In0~In1* are allowed to connect the variables of different data types.
- When the data types of input variables are one of BOOL, TIME, DATE, TOD and STRING, all the input variables and output variable should be of the data type. For example, if the data type of the variable connected to *In0* is TIME, the data types of the variables connected to *In1* and *Out* must be TIME. Otherwise, an error will occur during the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.
  - The length of the data type of the output variable must contain the length of the variables that the input parameters *In0* and *In1* connect. Otherwise, an error will occur during the compiling of the software.

Programming Example

- The data types of SEL_G, SEL_In0 and SEL_In1 are BOOL, UINT and DINT and the data type of Out1 is DINT. When SEL_EN is TRUE, the value of Out1 is 50 if the values of SEL_G, SEL_In0 and SEL_In1 are FALSE, 50 and 100 respectively as shown in the following table Variable 1. If the values of SEL_G, SEL_In0 and SEL_In1 are TRUE, 50 and 100 respectively, the value of Out1 is 100 as shown in the following table Variable 2.

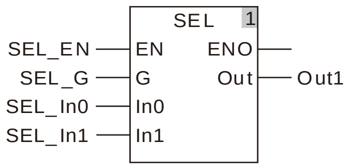
➢ **Variable 1**

Variable name	Data type	Current value
SEL_EN	BOOL	TRUE
SEL_G	BOOL	FALSE
SEL_In0	UINT	50
SEL_In1	DINT	100
Out1	DINT	50

➢ **Variable 2**

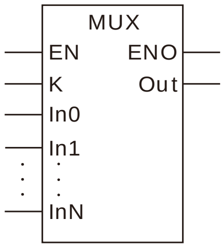
Variable name	Data type	Current value
SEL_EN	BOOL	TRUE
SEL_G	BOOL	TRUE
SEL_In0	UINT	50
SEL_In1	DINT	100
Out1	DINT	100

- **The program**



8.11.4 MUX

FB/FC	Explanation	Applicable model
FC	MUX is used for selecting one of two or more variables or constants and the result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
K	Gate	Input	Gate	Depends on the data type of the variable that the input parameter is connected to.
In0, In1 to InN	Selections	Input	The selections can be added or removed while the program is being written. The maximum number of selections is 8. N=2~8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Selection result	Output	Selection result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
K						●														
In0, In1 to InN	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
Out	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ Based on the selection condition *K*, the MUX instruction selects one of *In0~InN* and the selection result is output to *Out*. The value of *Out* corresponds to the value of *K* as shown in the following table.

The value of K	The value of Out
0	In0
1	In1
2	In2
3	In3
4	In4
5	In5
6	In6
7	In7

- When the data types of input variables are not BOOL, TIME, DATE, TOD or STRING, the input parameters *In0~InN* are allowed to connect the variables of different data types.
- When the data types of input variables are one of BOOL, TIME, DATE, TOD and STRING, all the input variables and output variable should be of the data type. For example, if the data type of *In0* is TIME, the data types of the variables connected to *In1~InN* and *Out* must be TIME. Otherwise, an error will occur during the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.
  - The length of the data type of the output variable must contain the length of the variables that the input parameters *In0* ~ *InN* connect. Otherwise, an error will occur during the compiling of the software.

Programming Example

- The data types of MUX_K, MUX_In0 and MUX_In1 are UINT, UINT and DINT and the data type of Out1 is DINT. When MUX_EN is TRUE, the value of Out1 is 50 if the values of MUX_K, MUX_In0 and MUX_In1 are 0, 50 and 100 as shown in the following table Variable 1. If the values of MUX_K, MUX_In0 and MUX_In1 are 1, 50 and 100, the value of Out1 is 100 as shown in the following table Variable 2.

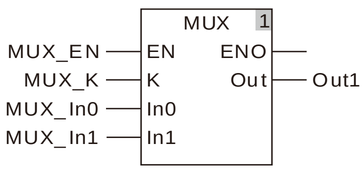
  - **Variable 1**

Variable name	Data type	Current value
MUX_EN	BOOL	TRUE
MUX_K	USINT	0
MUX_In0	UINT	50
MUX_In1	DINT	100
Out1	DINT	50

  - **Variable 2**

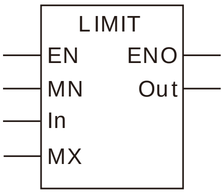
Variable name	Data type	Current value
MUX_EN	BOOL	TRUE
MUX_K	UINT	1
MUX_In0	UINT	50
MUX_In1	DINT	100
Out1	DINT	100

  - **The program**



8.11.5 LIMIT

FB/FC	Explanation	Applicable model
FC	LIMIT is used for limiting the output value within the zone between the specified minimum and maximum values.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
MN	Minimum value	Input	Minimum value	Depends on the data type of the variable that the input parameter is connected to.
In	Data to limit	Input	Data to limit	Depends on the data type of the variable that the input parameter is connected to.
MX	Maximum value	Input	Maximum value	Depends on the data type of the variable that the input parameter is connected to.
Out	Processing result	Output	Processing result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
In		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
MX		●	●	●	●	●	●	●	●	●	●	●	●	●	●					
Out		●	●	●	●	●	●	●	●	●	●	●	●	●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  ■ The LIMIT instruction limits the value within range between MN and MX and the result is output to *Out*.

The value of In	The value of Out
In < MN	MN
MN ≤ In ≤MX	In
MX < In	MX

- The instruction allows input parameters *MN*, *In* and *MX* to connect the variables of different data types. When *MN*, *In* and *MX* are the variables of different data types, the calculation is performed with the data type which can contain the range of the values of *MN*, *In* and *MX*. For example, if the data type of *MN* is INT and the data types of *In* and *MX* are DINT, the data type of *Out* is DINT.
- The instruction allows the input parameters and the output parameter to connect the variables of different data types. But the length of the data type of the output variable must contain the length of the variables that the input parameters *In0* ~ *InN* connect. Otherwise, an error will occur during the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

- The data types of LIMIT_MN, LIMIT_In and LIMIT_MX are UINT, UINT and DINT and the data type of Out1 is DINT. When LIMIT_EN is TRUE, the value of Out1 is 50 if the values of LIMIT_MN, LIMIT_In and LIMIT_MX are 1, 50 and 100 as shown in the following table Variable 1. If the values of LIMIT_MN, LIMIT_In and LIMIT_MX are 2, 200 and 100, the value of Out1 is 100 as shown in the following table Variable 2. If the values of LIMIT_MN, LIMIT_In and LIMIT_MX are 50, 10 and 100, the value of Out1 is 50 as shown in the following table Variable 3.

➢ **Variable 1**

Variable name	Data type	Current value
LIMIT_EN	BOOL	TRUE
LIMIT_MN	UINT	1
LIMIT_In	UINT	50
LIMIT_MX	DINT	100
Out1	DINT	50

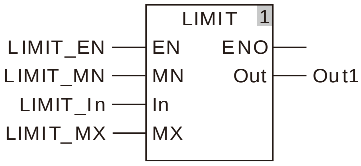
➢ **Variable 2**

Variable name	Data type	Current value
LIMIT_EN	BOOL	TRUE
LIMIT_MN	UINT	2
LIMIT_In	UINT	200
LIMIT_MX	DINT	100
Out1	DINT	100

➢ **Variable 3**

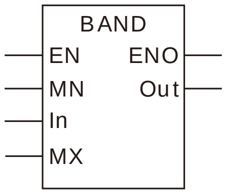
Variable name	Data type	Current value
LIMIT_EN	BOOL	TRUE
LIMIT_MN	UINT	50
LIMIT_In	UINT	10
LIMIT_MX	DINT	100
Out1	DINT	50

- **The program**


LIMIT
1
LIMIT_EN
EN
ENO
LIMIT_MN
MN
Out
Out1
LIMIT_In
In
LIMIT_MX
MX

8.11.6 BAND

FB/FC	Explanation	Applicable model
FC	BAND performs the deadband control and the processing result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
MN	Minimum value	Input	Minimum value	Depends on the data type of the variable that the input parameter is connected to.
In	Data to limit	Input	Data to limit	Depends on the data type of the variable that the input parameter is connected to.
MX	Maximum value	Input	Maximum value	Depends on the data type of the variable that the input parameter is connected to.
Out	Processing result	Output	Processing result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN														●	●					
In														●	●					
MX														●	●					
Out														●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The BAND instruction performs the dead band control of the value of *In* according to the maximum value, *MX* and the minimum value, *MN* and the processing result is output to *Out*.

The value of In	The value of Out
In < MN	In - MN
MN ≤In ≤MX	0
MX < In	In - MX

- The instruction allows input parameters *MN*, *In* and *MX* to connect the variables of different data types. When *MN*, *In* and *MX* are the variables of different data types, the calculation is performed with the data type which can contain the range of the values of *MN*, *In* and *MX*. For example, if the data type of *MN* is REAL and the data types of *In* and *MX* are LREAL, the data type of *Out* is LREAL.
- The instruction allows the input parameters and the output parameter to connect the variables of different data types. But the length of the data type of the output variable must contain the length of the variables that the input parameters connect. Otherwise, an error will occur during the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.
  - When the value of *MN* is greater than that of *MX*, the instruction will still be executed normally and the value of *Out* will be equal to that of *MX*.

Programming Example

- The data types of BAND_MN, BAND_In and BAND_MX are REAL and the data type of Out1 is LREAL. When BAND_EN is TRUE, the value of Out1 is 0 if the values of BAND_MN, BAND_In and BAND_MX are 1, 50 and 100 as shown in the following table Variable 1. If the values of BAND_MN, BAND_In and BAND_MX are 2, 250 and 100, the value of Out1 is 150 (150=250-100) as shown in the following table Variable 2. If the values of BAND_MN, BAND_In and BAND_MX are 50, 10 and 100, the value of Out1 is - 40 (- 40 = 10 – 50) as shown in the following table Variable 3.

➢ **Variable 1**

Variable name	Data type	Current value
BAND_EN	BOOL	TRUE
BAND_MN	REAL	1
BAND_In	REAL	50
BAND_MX	REAL	100
Out1	LREAL	0

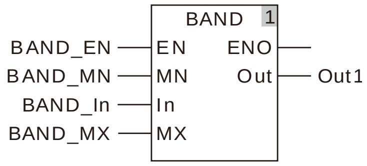
➢ **Variable 2**

Variable name	Data type	Current value
BAND_EN	BOOL	TRUE
BAND_MN	REAL	2
BAND_In	REAL	250
BAND_MX	REAL	100
Out1	LREAL	150

➢ **Variable 3**

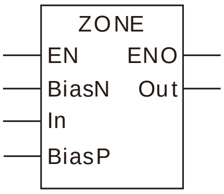
Variable name	Data type	Current value
BAND_EN	BOOL	TRUE
BAND_MN	REAL	50
BAND_In	REAL	10
BAND_MX	REAL	100
Out1	LREAL	-40

➢ **The program**



8.11.7 ZONE

FB/FC	Explanation	Applicable model
FC	ZONE is used for adding a bias value to the input value and the processing result is output to *Out*.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
BiasN	Negative bias value	Input	Negative bias	Depends on the data type of the variable that the input parameter is connected to.
In	Data to control	Input	Data to control	Depends on the data type of the variable that the input parameter is connected to.
BiasP	Positive bias value	Input	Positive bias	Depends on the data type of the variable that the input parameter is connected to.
Out	Processing result	Output	Processing result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
BiasN														●	●					
In														●	●					
BiasP														●	●					
Out														●	●					

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The ZONE instruction adds the set bias value to the value of *In* and the processing result is output to *Out*. When the value of *In* is a negative value, *BiasN* is the bias value. When the value of *In* is a positive value, *BiasP* is the bias value.

The value of In	The value of Out
In<0	In+BiasN
In=0	0
In>0	In+BiasP

- The instruction allows input parameters *BiasN*, *In* and *BiasP to* connect the variables of different data types. When *BiasN*, *In* and *BiasP* are the variables of different data types, the calculation is performed with the data type which can contain the range of the values of *BiasN*, *In* and *BiasP*. For example, if the data type of *BiasN* is INT and the data types of *In* and *BiasP* are DINT, the data type of *Out* is DINT.
- The instruction allows the input parameters and the output parameter to connect the variables of different data types. But the length of the data type of the output variable must contain the length of the variables that the input parameters connect. Otherwise, an error will occur during the compiling of the software.

- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.
  - When the value of *BiasN* is larger than *BiasP,* the instruction will still be executed normally.

Programming Example

- The data types of ZONE_BiasN, ZONE_In and ZONE_BiasP are INT, INT and DINT and the data type of Out1 is DINT. When ZONE_EN is TRUE, the value of Out1 is 0 if the values of ZONE_BiasN, ZONE_In and ZONE_BiasP are 1, 0 and 100 as shown in the following table Variable 1. If the values of ZONE_BiasN, ZONE_In and ZONE_BiasP are 2, 50 and 100, the value of Out1 is 150 (150 = 50 + 100) as shown in the following table Variable 2. If the values of ZONE_BiasN, ZONE_In and ZONE_BiasP are 50, -10 and 100, the value of Out1 is 40 (40 = - 10 + 50) as shown in the following table Variable 3.

➢ **Variable 1**

Variable name	Data type	Current value
ZONE_EN	BOOL	TRUE
ZONE_BiasN	INT	1
ZONE_In	INT	0
ZONE_BiasP	DINT	100
Out1	DINT	0

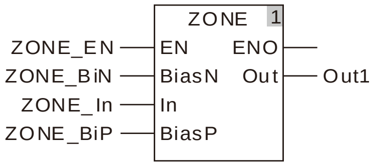
➢ **Variable 2**

Variable name	Data type	Current value
ZONE_EN	BOOL	TRUE
ZONE_BiasN	INT	2
ZONE_In	INT	50
ZONE_BiasP	DINT	100
Out1	DINT	150

➢ **Variable 3**

Variable name	Data type	Current value
ZONE_EN	BOOL	TRUE
ZONE_BiasN	INT	50
ZONE_In	INT	- 10
ZONE_BiasP	DINT	100
Out1	DINT	40

- **The program**



8

8.12 Data Type Conversion Instructions

8.12.1 BOOL_TO_***

FB/FC	Explanation	Applicable model
FC	BOOL_TO_*** instructions convert boolean data into the data of basic data types. “***” can be any basic data type.	DVP15MC11T

- **Parameters**

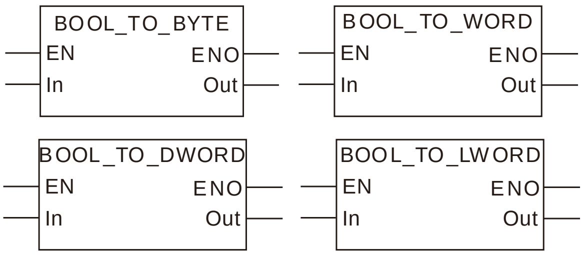
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to convert	Input	Data to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Conversion result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	●																			
Out		The data type of *Out* must be the same as “***” of the instruction name.																		

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - **BOOL to Bit String**
    - Relevant instructions:

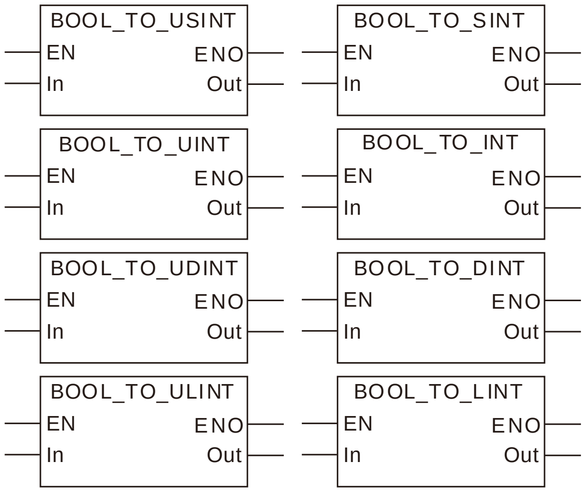


➢ The rule for the conversion from Boolean to Bit-String is shown in the following table. (The format of the bit-string value and the hexadecimal expression are to be confirmed.)

Boolean	Bit String			
	BYTE	WORD	DWORD	LWORD
FALSE	16#00	16#0000	16#0000_0000	16#0000_0000_0000_0000
TRUE	16#01	16#0001	16#0000_0001	16#0000_0000_0000_0001

■ **BOOL to Integer**

➢ Relevant instructions:

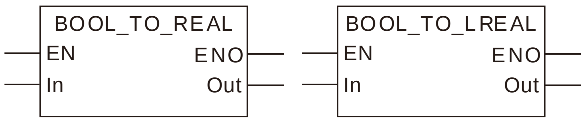


➢ The rule that Boolean data are converted into Integer data is as the following table shows.

Boolean	Integer							
	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT
FALSE	0	0	0	0	0	0	0	0
TRUE	1	1	1	1	1	1	1	1

■ **BOOL to Real number**

➢ Relevant instructions:

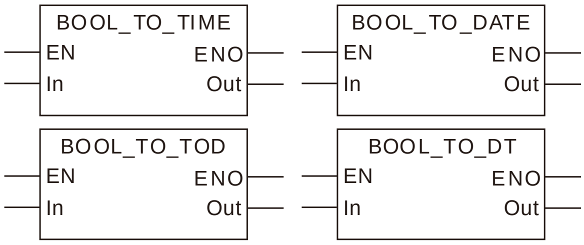


➢ The rule that Boolean data are converted into Real-number data is as the following table shows.

Boolean	Real	
	REAL	LREAL
FALSE	0	0
TRUE	1	1

■ BOOL to Time and Date

- Relevant instructions:

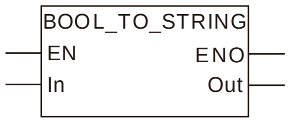


- The rule that Boolean data are converted into Time or Date data is as the following table shows.

Boolean	Time and Date			
	TIME	DATE	TOD	DT
FALSE	T#0ms	D#1970-1-1	TOD#	DT#
TRUE	T#1ms	D#1970-1-1	TOD#	DT#

■ BOOL to String

- Relevant instructions:



- The rule that Boolean data are converted into String data is as the following table shows. (The string format is to be confirmed.)

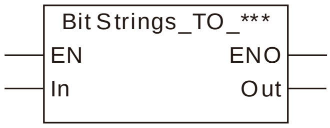
Boolean	String
	STRING
FALSE	‘FALSE’
TRUE	‘TRUE’

● Precautions for Correct Use

The input variables are not allowed to omit. If the input variables are omitted, an error will occur during the compiling of the software. The output variable is allowed to omit.

8.12.2 Bit strings_TO_***

FB/FC	Explanation	Applicable model
FC	Bit strings_TO_*** instructions convert bit-string data into the data of basic data types. "***" can be any basic data type.	DVP15MC11T



- **Parameters**

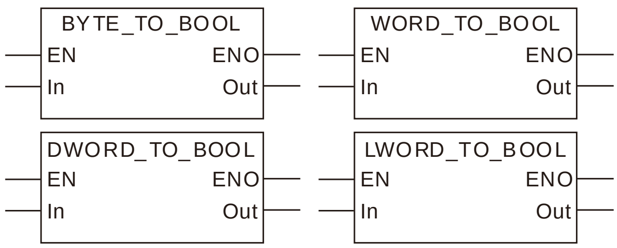
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to convert	Input	Data to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Conversion result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		●	●	●	●															
Out	The data type of *Out* must be the same as "***" of the instruction name.																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - **Bit string to BOOL**
    - Relevant instructions:



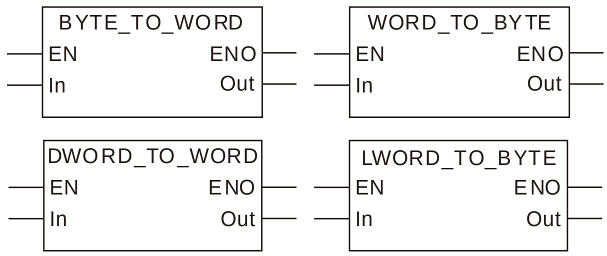
8

➢ The rule that Bit-string data are converted into Boolean data is as the following table shows.

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
BYTE	BOOL	16#00	FALSE
		16#01~16#FF	TRUE
WORD	BOOL	16#0000	FALSE
		16#0001~16#FFFF	TRUE
DWORD	BOOL	16#0000_0000	FALSE
		16#0000_0001~16#FFFF_FFFF	TRUE
LWORD	BOOL	16#0000_0000_0000_0000	FALSE
		16#0000_0000_0000_0001~ 16#FFFF_FFFF_FFFF_FFFF	TRUE

- **Bit string to Bit string**

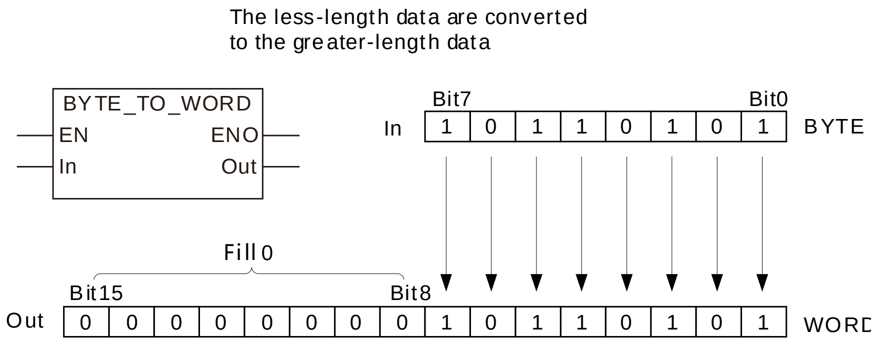
➢ Bit-string data can be converted to Bit-string data. And some instructions are shown below.



There are two kinds of conversion for different types of bit-string data. One is the conversion of the less-length data to the greater-length data. The other is the conversion of the greater-length data to the less-length data.

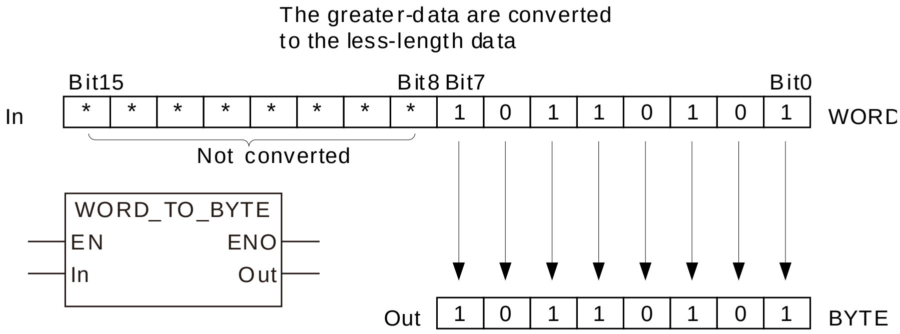
The less-length data is converted to the greater-length data by writing the values of all bits of the less-length data to corresponding bits of the greater-length data and setting the values of the remaining bits of the greater-length data to 0.

See the following example that the Byte data in *In* is converted to the Word data in *Out*. The values of Bit0~Bit7 of *In* are copied and pasted to Bit0~Bit7 of *Out*. And the values of Bit8~Bit15 of *Out* are set to 0.



The greater-length data are converted to the less-length data by revising the values of all bits of the less-length data into the values of the corresponding bits of the greater-length data and the values of the remaining bits of the greater-length data are not converted and have no impact on the conversion.

See the following example that the Word data *In* is converted to the Byte data *Out*. The values of Bit0~Bit7 of *In* are copied and pasted to Bit0~Bit7 of *Out*. And the values of Bit8~Bit15 of *In* are not converted and have no impact on the conversion.

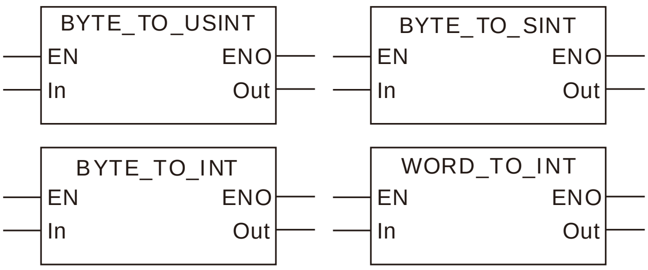


➢ The Bit-string data are converted into the Bit-string data as the following table shows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
BYTE	WORD	16#00~16#FF	16#0000~16#00FF
	DWORD	16#00~16#FF	16#0000_0000~16#0000_00FF
	LWORD	16#00~16#FF	16#0000_0000_0000_0000~ 16#0000_0000_0000_00FF
WORD	BYTE	16#**00~16#**FF	16#00~16#FF
	DWORD	16#0000~16#FFFF	16#0000_0000~16#0000_FFFF
	LWORD	16#0000~16#FFFF	16#0000_0000_0000_0000~ 16#0000_0000_0000_FFFF
DWORD	BYTE	16#****_**00~16#****_**FF	16#00~16#FF
	WORD	16#****_0000~16#****_FFFF	16#0000~16#FFFF
	LWORD	16#0000_0000~16#FFFF_FFFF	16#0000_0000_0000_0000~ 16#0000_0000_FFFF_FFFF
LWORD	BYTE	16#****_****_****_**00~ 16#****_****_****_**FF	16#00~16#FF
	WORD	16#****_****_****_0000~ 16#****_****_****_FFFF	16#0000~16#FFFF
	DWORD	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	16#0000_0000~16#FFFF_FFFF

■ **Bit string to Integer**

➢ The Bit-string data can be converted to the Integer data. And some instructions are shown below.



The rule for the conversion of bit-string data into integer data is consistent with that for the conversion of bit-string data into bit-string data.

The less-length data is converted to the greater-length data by writing the values of all bits of the less-length data to corresponding bits of the greater-length data and setting the values of the remaining bits of the greater-length data to 0.

8

The greater-length data is converted to the less-length data by revising the values of all bits of the less-length data into the values of the corresponding bits of the greater-length data and the values of the remaining bits of the greater-length data are not converted and have no impact on the conversion.

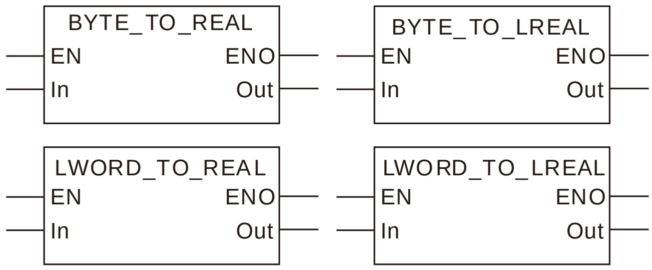
If the lengths of the two data to convert are equal, all values of all bits of *In* are copied and pasted to the corresponding bits of *Out*.

➢ The Bit-string data are converted into the Integer data as the following table shows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
BYTE	USINT	16#00~16#FF	0~255
	UINT	16#00~16#FF	0~255
	UDINT	16#00~16#FF	0~255
	ULINT	16#00~16#FF	0~255
	SINT	16#00~16#7F	0~127
		16#80~16#FF	-128~-1
	INT	16#00~16#FF	0~255
	DINT	16#00~16#FF	0~255
	LINT	16#00~16#FF	0~255
WORD	USINT	16#**00~16#**FF	0~255
	UINT	16#0000~16#FFFF	0~65535
	UDINT	16#0000~16#FFFF	0~65535
	ULINT	16#0000~16#FFFF	0~65535
	SINT	16#**00~16#**7F	0~127
		16#**80~16#**FF	-128~-1
	INT	16#0000~16#7FFF	0~32767
		16#8000~16#FFFF	-32768~-1
	DINT	16#0000~16#FFFF	0~65535
	LINT	16#0000~16#FFFF	0~65535
DWORD	USINT	16#****_**00~16#****_**FF	0~255
	UINT	16#****_0000~16#****_FFFF	0~65535
	UDINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
	ULINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
	SINT	16#****_**00~16#****_**7F	0~127
		16#****_**80~16#****_**FF	-128~-1
	INT	16#****_0000~16#****_7FFF	0~32767
		16#****_8000~16#****_FFFF	-32768~-1
	DINT	16#0000_0000~16#7FFF_FFFF	0~2147483647
		16#8000_0000~16#FFFF_FFFF	-2147483648~-1
	LINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
LWORD	USINT	16#****_****_****_**00~ 16#****_****_****_**FF	0~255
	UINT	16#****_****_****_0000~ 16#****_****_****_FFFF	0~65535

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
	UDINT	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	0~4294967295
	ULINT	16#0000_0000_0000_0000~ 16# FFFF_FFFF_FFFF_FFFF	0~18446744073709551645
	SINT	16#****_****_****_**00~ 16#****_****_****_**7F	0~127
		16#****_****_****_**80~ 16#****_****_****_**FF	-128~-1
	INT	16#****_****_****_0000~ 16#****_****_****_7FFF	0~32767
		16#****_****_****_8000~ 16#****_****_****_FFFF	-32768~-1
	DINT	16#****_****_0000_0000~ 16#****_****_7FFF_FFFF	0~2147483647
		16#****_****_8000_0000~ 16#***_*****_FFFF_FFFF	-2147483648~-1
	LINT	16#0000_0000_0000_0000~ 16#7FFF_FFFF_FFFF_FFFF	0~9223372036854775807
		16#8000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	-9223372036854775808~0

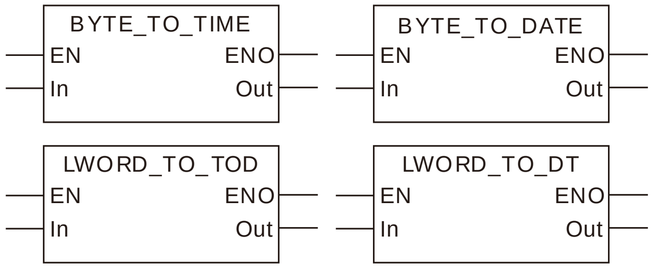
- **Bit string to Real number**
  - The Bit-string data can be converted to the Real-number data. And some instructions are shown below.



  - The Bit-string data are converted into the Real-number data as the following table shows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
BYTE	REAL	16#00~16#FF	0~2.55e+2
	LREAL	16#00~16#FF	0~2.55e+2
WORD	REAL	16#0000~16#FFFF	0~6.5535e+4
	LREAL	16#0000~16#FFFF	0~6.5535e+4
DWORD	REAL	16#0000_0000~ 16#FFFF_FFFF	0~4.294967e+9
	LREAL	16#0000_0000~ 16#FFFF_FFFF	0~4.294967295e+9
LWORD	REAL	16#0000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	0~1.844674e+19
	LREAL	16#0000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	0~1.84467440737095e+19

- **Bit string to Time and Date**
  - The Bit-string data can be converted to the Time or Date data. And some instructions are shown below.



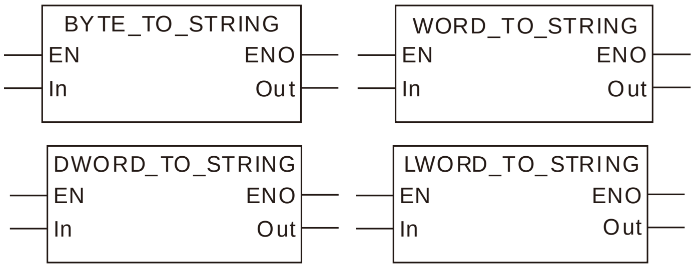
The rule for the conversion of Bit-string data into Time or Date data is the same as that for the conversion of the Bit-string data into unsigned integer data.

  - The Bit-string data are converted into the Time and Date data as the following table shows.

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
BYTE	TIME	16#00~16#FF	T#0ns~T#255ns
	DATE	16#00~16#FF	D#1970-1-1
	TOD	16#00~16#FF	TOD#0:0:0~ TOD#0:0:0.255
	DT	16#00~16#FF	DT#1970-1-1-0:0:0~ DT#1970-1-1-0:4:15
WORD	TIME	16#0000~16#FFFF	T#0ns~T#65us535ns
	DATE	16#0000~16#FFFF	D#1970-1-1
	TOD	16#0000~16#FFFF	TOD#0:0:0~ TOD#0:1:5.535
	DT	16#0000~16#FFFF	DT#1970-1-1-0:0:0~ DT#1970-1-1-18:12:15
DWORD	TIME	16#0000_0000~16#FFFF_FFFF	T#0ns~ T#4s294ms967us295ns
	DATE	16#0000_0000~16#FFFF_FFFF	D#1970-1-1~D#2016-2-7
	TOD	16#0000_0000~16#0526_5BFF	TOD#0:0:0~ TOD#23:59:59.999
		16#0526_5C00~16#0A4C_B7FF	
		…….	
		16#FC57_9C00~16#FFFF_FFFF	TOD#0:0:0~ TOD#17:2:47.295
	DT	16#0000_0000~16#FFFF_FFFF	DT#1970-1-1-0:0:0~ DT#2016-2-7-6:28:15
LWORD	TIME	16#0000_0000_0000_0000~ 16# FFFF_FFFF_FFFF_FFFF	T#213503d23h34m33s709ms551us615ns
	DATE	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	D#1970-1-1~D#2016-2-7
	TOD	16#****_****_0000_0000~ 16#****_****_0A4C_B7FF	TOD#0:0:0~ TOD#23:59:59.999
		16#****_****_0526_5C00~ 16#****_****_0A4C_B7FF	
		……	
		16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	TOD#0:0:0~ TOD#17:2:47.295
	DT	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	DT#1970-1-1-0:0:0~ DT#2016-2-7-6:28:15

■ Bit string to String

➢ The Bit-string data can be converted to the String data. And some instructions are shown below.



➢ The Bit-string data are converted into the String data as the following table shows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
BYTE	STRING	16#00~16#FF	‘00’~’FF’
WORD	STRING	16#0000~16#FFFF	‘0000’~’FFFF’
DWORD	STRING	16#0000_0000~16#FFFF_FFFF	‘00000000’~’FFFFFFFF’
LWORD	STRING	16#0000_0000_0000_0000~ 16# FFFF_FFFF_FFFF_FFFF	‘0000000000000000’~ ‘FFFFFFFFFFFFFFFF’

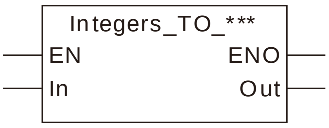
When the Bit-string data are converted to the String data, the length of the output String data must meet the length of the input parameter. For example, during the use of the BYTE_TO_STRING instruction, the output String data must contain more than 2 characters. Otherwise, an error will occur during the compiling of the software.

● Precautions for Correct Use

The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

8.12.3 Integers_TO_***

FB/FC	Explanation	Applicable model
FC	Integers_TO_*** instructions convert integers into the data of basic data types. "***" can be any basic data type.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to convert	Input	Data to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Conversion result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						●	●	●	●	●	●	●	●							
Out	The data type of *Out* must be the same as "***" of the instruction name.																			

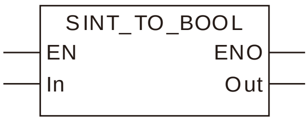
Note:

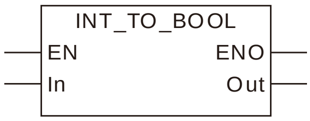
The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

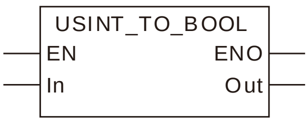
- **Function Explanation**

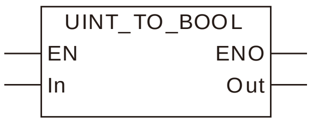
  - **Integer to BOOL**

    - Some instructions are shown below.







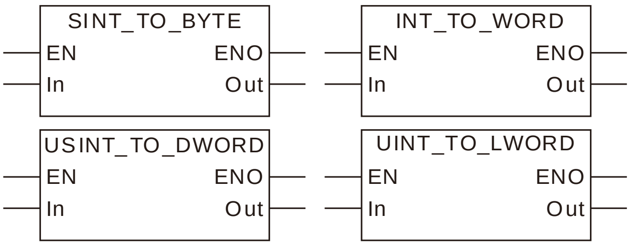


➢ The Integer data are converted into the Boolean data as the following table shows. If the Integer value is 0, the conversion result is FALSE. If not 0, the result is TRUE. For details on the conversion rule, see the table as follows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
USINT	BOOL	0	FALSE
		1~255	TRUE
UINT	BOOL	0	FALSE
		1~65535	TRUE
UDINT	BOOL	0	FALSE
		1~4294967295	TRUE
ULINT	BOOL	0	FALSE
		1~18446744073709551645	TRUE
SINT	BOOL	0	FALSE
		-128~-1, 1~127	TRUE
INT	BOOL	0	FALSE
		-32768~-1, 1~32767	TRUE
DINT	BOOL	0	FALSE
		-2147483648~-1, 1~2147483647	TRUE
LINT	BOOL	0	FALSE
		-9223372036854775808~-1, 1~9223372036854775807	TRUE

■ **Integer to Bit string**

➢ The Integer data can be converted to the Bit-string data. And some instructions are shown below.



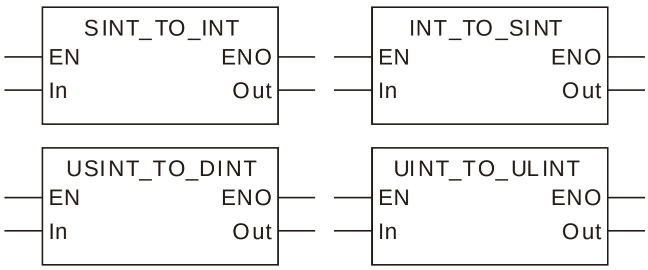
The rule for the conversion of the Integer data into the Bit-string data is the same as that for the conversion of the Bit-string data into the Bit-string data. Refer to section 8.13.2 for details.

➢ The Integer data are converted into the Bit-string data as the following table shows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
USINT	BYTE	16#00~16#FF	16#00~16#FF
	WORD	16#00~16#FF	16#0000~16#00FF
	DWORD	16#00~16#FF	16#0000_0000~16#0000_00FF
	LWORD	16#00~16#FF	16#0000_0000_0000_0000~ 16#0000_0000_0000_00FF
UINT	BYTE	16#**00~16#**FF	16#00~16#FF
	WORD	16#0000~16#FFFF	16#0000~16#FFFF
	DWORD	16#0000~16#FFFF	16#0000_0000~16#0000_FFFF
	LWORD	16#0000~16#FFFF	16#0000_0000_0000_0000~ 16#0000_0000_0000_FFFF

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
UDINT	BYTE	16#****_**00~16#****_**FF	16#00~16#FF
	WORD	16#****_0000~16#****_FFFF	16#0000~16#FFFF
	DWORD	16#0000_0000~16#FFFF_FFFF	16#0000_0000~16#FFFF_FFFF
	LWORD	16#0000_0000~16#FFFF_FFFF	16#0000_0000_0000_0000~ 16#0000_0000_FFFF_FFFF
ULINT	BYTE	16#****_****_****_**00~ 16#****_****_****_**FF	16#00~16#FF
	WORD	16#****_****_****_0000~ 16#****_****_****_FFFF	16#0000~16#FFFF
	DWORD	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	16#0000_0000~16#FFFF_FFFF
	LWORD	16#0000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	16#0000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF
SINT	BYTE	16#00~16#FF	16#00~16#FF
	WORD	16#00~16#FF	16#0000~16#00FF
	DWORD	16#00~16#FF	16#0000_0000~16#0000_00FF
	LWORD	16#00~16#FF	16#0000_0000_0000_0000~ 16#0000_0000_0000_00FF
INT	BYTE	16#**00~16#**FF	16#00~16#FF
	WORD	16#0000~16#FFFF	16#0000~16#FFFF
	DWORD	16#0000~16#FFFF	16#0000_0000~16#0000_FFFF
	LWORD	16#0000~16#FFFF	16#0000_0000_0000_0000~ 16#0000_0000_0000_FFFF
DINT	BYTE	16#****_**00~16#****_**FF	16#00~16#FF
	WORD	16#****_0000~16#****_FFFF	16#0000~16#FFFF
	DWORD	16#0000_0000~16#FFFF_FFFF	16#0000_0000~16#FFFF_FFFF
	LWORD	16#0000_0000~16#FFFF_FFFF	16#0000_0000_0000_0000~ 16#0000_0000_FFFF_FFFF
LINT	BYTE	16#****_****_****_**00~ 16#****_****_****_**FF	16#00~16#FF
	WORD	16#****_****_****_0000~ 16#****_****_****_FFFF	16#0000~16#FFFF
	DWORD	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	16#0000_0000~16#FFFF_FFFF
	LWORD	16#0000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	16#0000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF

- **Integer to Integer**
  - The Integer data can be converted to the Integer data. And some instructions are shown below.



1. The rule for the conversion of the Integer data into the Integer data is the same as that for the conversion of the Bit-string data into the Bit-string data.
2. The less-length data are converted to the greater-length data by writing the values of all bits of the less-length data to corresponding bits of the greater-length data and setting the values of the remaining bits of the greater-length data to 0.
3. The data of greater length is converted to the data of less length by revising the values of all bits of the less-length data into the values of the corresponding bits of the greater-length data and the values of the remaining bits of the greater-length data are not converted and have no impact on the conversion.
4. If the lengths of the two data to convert are equal, all values of all bits of *In* are copied and pasted to the corresponding bits of *Out*.

➢ The Bit-string data are converted into the Integer data as the following table shows.

Data type		**The value of *In* corresponds to the value of *Out***	
In	Out	In	Out
USINT	USINT	16#00~16#FF	0~255
	UINT	16#00~16#FF	0~255
	UDINT	16#00~16#FF	0~255
	ULINT	16#00~16#FF	0~255
	SINT	16#00~16#7F	0~127
		16#80~16#FF	- 128~ - 1
	INT	16#00~16#FF	0~255
	DINT	16#00~16#FF	0~255
	LINT	16#00~16#FF	0~255
UINT	USINT	16#**00~16#**FF	0~255
	UINT	16#0000~16#FFFF	0~65535
	UDINT	16#0000~16#FFFF	0~65535
	ULINT	16#0000~16#FFFF	0~65535
	SINT	16#**00~16#**7F	0~127
		16#**80~16#**FF	- 128~ -1
	INT	16#0000~16#7FFF	0~32767
		16#8000~16#FFFF	- 32768~ -1
	DINT	16#0000~16#FFFF	0~65535
	LINT	16#0000~16#FFFF	0~65535
UDINT	USINT	16#****_**00~16#****_**FF	0~255
	UINT	16#****_0000~16#****_FFFF	0~65535
	UDINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
	ULINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
	SINT	16#****_**00~16#****_**7F	0~127
		16#****_**80~16#****_**FF	-128~-1
	INT	16#****_0000~16#****_7FFF	0~32767
		16#****_8000~16#****_FFFF	-32768~-1
	DINT	16#0000_0000~16#7FFF_FFFF	0~2147483647
		16#8000_0000~16#FFFF_FFFF	-2147483648~-1
	LINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
ULINT	USINT	16#****_****_****_**00~ 16#****_****_****_**FF	0~255
	UINT	16#****_****_****_0000~ 16#****_****_****_FFFF	0~65535

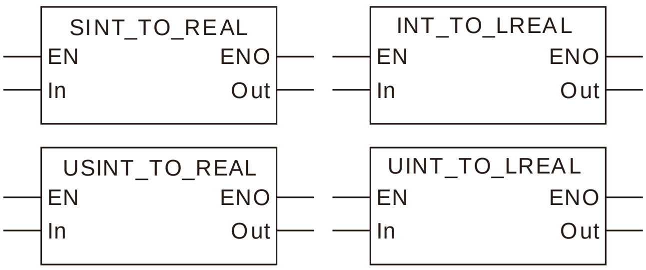
8

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
	UDINT	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	0~4294967295
	ULINT	16#0000_0000_0000_0000~ 16# FFFF_FFFF_FFFF_FFFF	0~18446744073709551645
	SINT	16#****_****_****_**00~ 16#****_****_****_**7F	0~127
		16#****_****_****_**80~ 16#****_****_****_**FF	-128~-1
	INT	16#****_****_****_0000~ 16#****_****_****_7FFF	0~32767
		16#****_****_****_8000~ 16#****_****_****_FFFF	-32768~-1
	DINT	16#****_****_0000_0000~ 16#****_****_7FFF_FFFF	0~2147483647
		16#****_****_8000_0000~ 16#****_****_FFFF_FFFF	-2147483648~-1
	LINT	16#0000_0000_0000_0000~ 16#7FFF_FFFF_FFFF_FFFF	0~9223372036854775807
		16#8000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	-9223372036854775808~0
SINT	USINT	16#00~16#FF	0~255
	UINT	16#00~16#FF	0~255
	UDINT	16#00~16#FF	0~255
	ULINT	16#00~16#FF	0~255
	SINT	16#00~16#7F	0~127
		16#80~16#FF	-128~-1
	INT	16#00~16#FF	0~255
	DINT	16#00~16#FF	0~255
	LINT	16#00~16#FF	0~255
INT	USINT	16#**00~16#**FF	0~255
	UINT	16#0000~16#FFFF	0~65535
	UDINT	16#0000~16#FFFF	0~65535
	ULINT	16#0000~16#FFFF	0~65535
	SINT	16#**00~16#**7F	0~127
		16#**80~16#**FF	-128~-1
	INT	16#0000~16#7FFF	0~32767
		16#8000~16#FFFF	-32768~-1
	DINT	16#0000~16#FFFF	0~65535
	LINT	16#0000~16#FFFF	0~65535
DINT	USINT	16#****_**00~16#****_**FF	0~255
	UINT	16#****_0000~16#****_FFFF	0~65535
	UDINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
	ULINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
	SINT	16#****_**00~16#****_**7F	0~127
		16#****_**80~16#****_**FF	-128~-1
	INT	16#****_0000~16#****_7FFF	0~32767

8

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
DINT		16#****_8000~16#****_FFFF	-32768~-1
	DINT	16#0000_0000~16#7FFF_FFFF	0~2147483647
		16#8000_0000~16#FFFF_FFFF	-2147483648~-1
	LINT	16#0000_0000~16#FFFF_FFFF	0~4294967295
LINT	USINT	16#****_****_****_**00~ 16#****_****_****_**FF	0~255
	UINT	16#****_****_****_0000~ 16#****_****_****_FFFF	0~65535
	UDINT	16#****_****_0000_0000~ 16#****_****_FFFF_FFFF	0~4294967295
	ULINT	16#0000_0000_0000_0000~ 16# FFFF_FFFF_FFFF_FFFF	0~18446744073709551645
	SINT	16#****_****_****_**00~ 16#****_****_****_**7F	0~127
		16#****_****_****_**80~ 16#****_****_****_**FF	-128~-1
	INT	16#****_****_****_0000~ 16#****_****_****_7FFF	0~32767
		16#****_****_****_8000~ 16#****_****_****_FFFF	-32768~-1
	DINT	16#****_****_0000_0000~ 16#****_****_7FFF_FFFF	0~2147483647
		16#****_****_8000_0000~ 16#****_****_FFFF_FFFF	-2147483648~-1
	LINT	16#0000_0000_0000_0000~ 16#7FFF_FFFF_FFFF_FFFF	0~9223372036854775807
		16#8000_0000_0000_0000~ 16#FFFF_FFFF_FFFF_FFFF	-9223372036854775808~0

- **Integer to Real number**
  - The Integer data can be converted to the Real-number data. And some instructions are shown below.

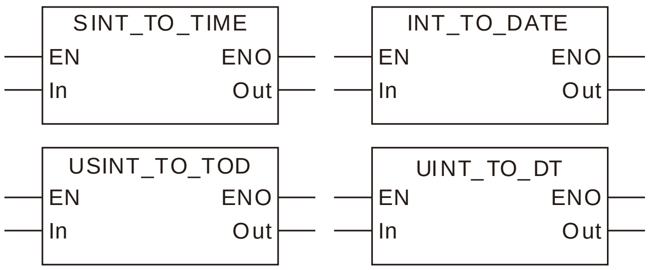


- The Integer data are converted into the Real-number data as the following table shows.

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
USINT	REAL	0~255	0~2.55e+2
	LREAL	0~255	0~2.55e+2
UINT	REAL	0~65535	0~6.5535e+4
	LREAL	0~65535	0~6.5535e+4
UDINT	REAL	0~4294967295	0~4.294967e+9
	LREAL	0~4294967295	0~4.294967295e+9
ULINT	REAL	0~18446744073709551615	0~1.844674e+19
	LREAL	0~18446744073709551615	0~1.84467440737095e+19
SINT	REAL	-128~127	-1.28e+2~1.27e+2
	LREAL	-128~127	-1.28e+2~1.27e+2
INT	REAL	-32768~32767	-3.2768e+4~3.2767e+4
	LREAL	-32768~32767	-3.2768e+4~3.2767e+4
DINT	REAL	-2147483648~2147483647	-2.147483e+9~2.147483e+9
	LREAL	-2147483648~2147483647	-2.147483e+9~2.147483e+9
LINT	REAL	-9223372036854775808~ 9223372036854775807	-9.223372e+18~9.223372e+18
	LREAL	-9223372036854775808~ 9223372036854775807	-9.22337203685477e+18~ 9.22337203685477e+18

Integer to Time or Date

- The Integer data are converted into the Time or Date data and some instructions are shown as below.



The rule for the conversion of the Integer data into the Time or Date data is the same as that for the conversion of the Integer data into the unsigned integer data.

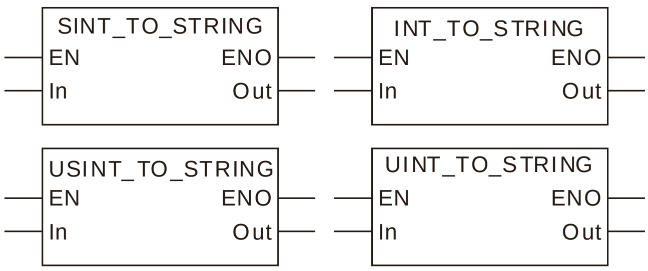
- The Integer data are converted into the Time or Date data as the following table shows.

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
USINT	TIME	16#00~16#FF	T#0ns~T#255ns
	DATE	16#00~16#FF	D#1970-1-1
	TOD	16#00~16#FF	TOD#0:0:0~ TOD#0:0:0.255
	DT	16#00~16#FF	DT#1970-1-1-0:0:0~ DT#1970-1-1-0:4:15
UINT	TIME	16#0000~16#FFFF	T#0ns~T#65us535ns
	DATE	16#0000~16#FFFF	D#1970-1-1
	TOD	16#0000~16#FFFF	TOD#0:0:0~ TOD#0:1:5.535
	DT	16#0000~16#FFFF	DT#1970-1-1-0:0:0~ DT#1970-1-1-18:12:15

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
UDINT	TIME	16#00000000~16#FFFFFFFF	T#0ns~ T#4s294ms967us295ns
	DATE	16#00000000~16#FFFFFFFF	D#1970-1-1~D#2016-2-7
	TOD	16#00000000~16#05265BFF	TOD#0:0:0~ TOD#23:59:59.999
		16#05265C00~16#0A4CB7FF	
		…….	
		16#FC579C00~16#FFFFFFFF	TOD#0:0:0~ TOD#17:2:47.295
	DT	16#00000000~16#FFFFFFFF	DT#1970-1-1-0:0:0~ DT#2016-2-7-6:28:15
ULINT	TIME	16#0000000000000000~ 16# FFFFFFFFFFFFFFFF	T#213503d23h34m33s709ms551us615ns
	DATE	16#********00000000~ 16#********FFFFFFFF	D#1970-1-1~D#2016-2-7
	TOD	16#********00000000~ 16#********0A4CB7FF	TOD#0:0:0~ TOD#23:59:59.999
		16#********05265C00~ 16#********0A4CB7FF	
		……	
		16#********00000000~ 16#********FFFFFFFF	TOD#0:0:0~ TOD#17:2:47.295
	DT	16#********00000000~ 16#********FFFFFFFF	DT#1970-1-1-0:0:0~ DT#2016-2-7-6:28:15
SINT	TIME	16#00~16#FF	T#0ns~T#255ns
	DATE	16#00~16#FF	D#1970-1-1
	TOD	16#00~16#FF	TOD#0:0:0~ TOD#0:0:0.255
	DT	16#00~16#FF	DT#1970-1-1-0:0:0~ DT#1970-1-1-0:4:15
INT	TIME	16#0000~16#FFFF	T#0ns~T#65us535ns
	DATE	16#0000~16#FFFF	D#1970-1-1
	TOD	16#0000~16#FFFF	TOD#0:0:0~ TOD#0:1:5.535
	DT	16#0000~16#FFFF	DT#1970-1-1-0:0:0~ DT#1970-1-1-18:12:15
DINT	TIME	16#00000000~16#FFFFFFFF	T#0ns~ T#4s294ms967us295ns
	DATE	16#00000000~16#FFFFFFFF	D#1970-1-1~D#2016-2-7
	TOD	16#00000000~16#05265BFF	TOD#0:0:0~ TOD#23:59:59.999
		16#05265C00~16#0A4CB7FF	
		…….	
		16#FC579C00~16#FFFFFFFF	TOD#0:0:0~ TOD#17:2:47.295
	DT	16#00000000~16#FFFFFFFF	DT#1970-1-1-0:0:0~ DT#2016-2-7-6:28:15
LINT	TIME	16#0000000000000000~ 16# FFFFFFFFFFFFFFFF	T#213503d23h34m33s709ms551us615ns
	DATE	16#********00000000~ 16#********FFFFFFFF	D#1970-1-1~D#2016-2-7
	TOD	16#********00000000~ 16#********0A4CB7FF	TOD#0:0:0~ TOD#23:59:59.999

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
		16#********05265C00~ 16#********0A4CB7FF	
		……	
		16#********00000000~ 16#********FFFFFFFF	TOD#0:0:0~ TOD#17:2:47.295
	DT	16#********00000000~ 16#********FFFFFFFF	DT#1970-1-1-0:0:0~ DT#2016-2-7-6:28:15

- **Integer to String**
  - The Integer data can be converted to the String data and some instructions are shown as below.



  - The Integer data are converted into the String data as the following table shows.

Data type		The value of *In* corresponds to the value of *Out*	
In	Out	In	Out
USINT	STRING	0~255	'0'~'255'
UINT	STRING	0~65535	'0'~'65535'
UDINT	STRING	0~4294967295	'0'~'4294967295'
ULINT	STRING	0~18446744073709551615	'0'~'18446744073709551615'
SINT	STRING	-128~127	'-128'~'127'
INT	STRING	-32768~32767	'-32768'~'32767'
DINT	STRING	-2147483648~2147483647	'-2147483648'~'2147483647'
LINT	STRING	-9223372036854775808~ 9223372036854775807	'-9223372036854775808'~ '9223372036854775807'

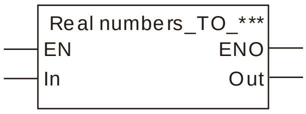
When the Bit-string data are converted to the String data, the length of the output String data must meet the length of the input parameter.

- **Precautions for Correct Use**

The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

8.12.4 Real numbers_TO_***

FB/FC	Explanation	Applicable model
FC	Real numbers_TO_*** instructions convert real numbers into the data of basic data types. “***” can be any basic data type.	DVP15MC11T



- **Parameters**

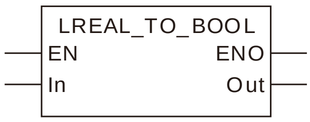
Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to convert	Input	Data to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Conversion result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														●	●					
Out	The data type of *Out* must be the same as “***” of the instruction name.																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

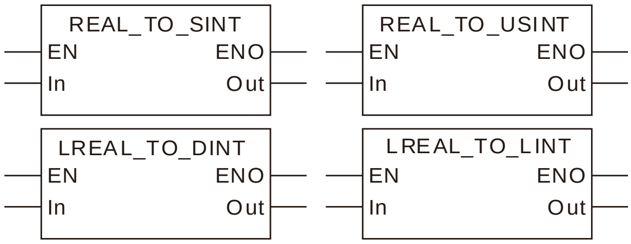
- **Function Explanation**
  - **Real Number to BOOL**
    - Relevant instructions:



- The real numbers are converted into the Boolean data as the following table shows. If the real number is 0, the conversion result is FALSE. If the real number is not 0, the conversion result is TRUE. For details on the rule, see the table as follows.

Data type		The value of *In* corresponds to the value of *Out*	
In	**Out**	**In**	**Out**
REAL	BOOL	-3.402823E+38~-1.175495E-38	TRUE
		0	FALSE
		1.175495E-38~3.402823E+38	TRUE
LREAL	BOOL	-1.79769313486231E+308~ -2.22507385850721E-308	TRUE
		0	FALSE
		2.22507385850721E-308~ 1.79769313486231E+308	TRUE

- **Real Number to Integer**
  - Real numbers can be converted to integers. And some instructions are shown below.



  - For the real number-to-integer conversion, there are two cases in which the fractional part is truncated and rounded up as follows.

    Case 1: If the first digital number of the fractional part is less than 5, the fractional part will be truncated and the integer part will not change.

    Case 2: If the first digital number of the fractional part is greater than or equal to 5, the fractional part will be truncated and the integer part will add by 1.

Input value		**Output result**	
		Data type	**Output value**
Case 1	1.36	SINT	1
		USINT	1
	-2.4	SINT	-2
		USINT	254
Case 2	1.6	SINT	2
		USINT	2
	-2.6	SINT	-3
		USINT	253

Note:

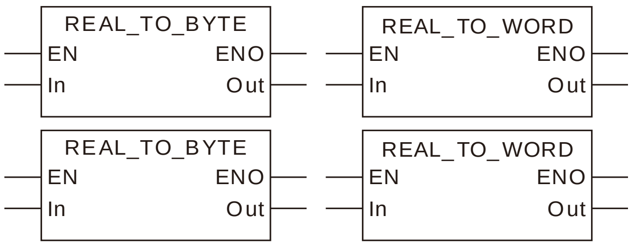
For the Real Number-to-Integer Conversion, there are two cases for the value of a real number.

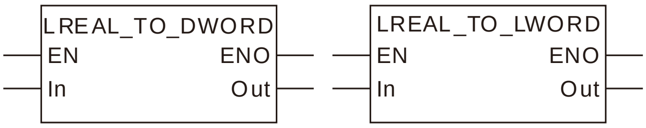
1. If the number of input digits of a real number exceeds what is allowed, the result will be an unsure value. Please set a limit in the user program in order to get a correct value.
   For example: Then the input value is 123456789 and the number of its digits exceeds the set limit 7. The digits which go beyond the limit are abnormal. Then the output value is 1234567***92***.

2. If the number of input digits does not exceed the set limit, the result is calculated based on the conversion rule.

8

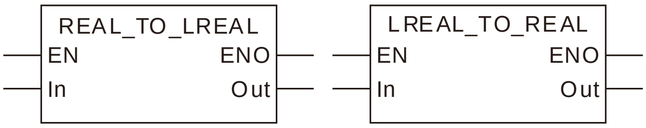
- **Real Number to Bit string**
  - Real numbers can be converted to bit strings. And some instructions are shown below.



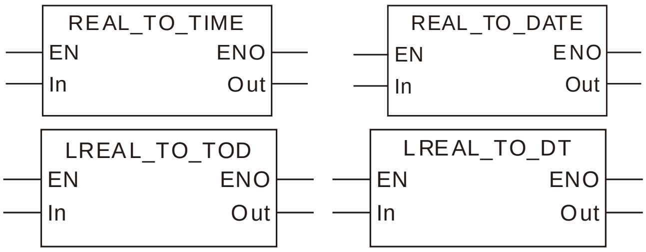


The rule for the conversion of real numbers into bit strings is the same as that for the conversion of real numbers into unsigned integers.

- **Real Number to Real Number**
  - Real numbers can be converted to real numbers. And some instructions are shown below.

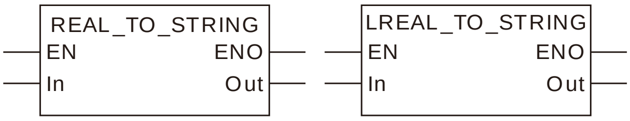


- **Real Number to Time or Date**
  - Real numbers can be converted to times or dates. And some instructions are shown below.



For the real number-to-time or date conversion, the real number is converted to the integer first and then the integer is converted to the time or date. For relevant contents, refer to the real number-to- integer conversion and integer-to-time or date conversion.

- **Real Number to String**
  - Real numbers can be converted to strings. And some instructions are shown below.



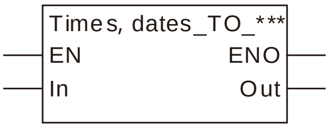
The rule for the real number-to-string conversion is the same as that for the integer-to-string conversion. Refer to section 8.13.3 for details.

- **Precautions for Correct Use**

The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

8.12.5 Times,dates_TO_***

FB/FC	Explanation	Applicable model
FC	Times, dates_TO_*** instructions convert Time or date data into the data of basic data types. “***” can be any basic data type.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to convert	Input	Data to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Conversion result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																●	●	●	●	
Out	The data type of *Out* must be the same as “***” of the instruction name.																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

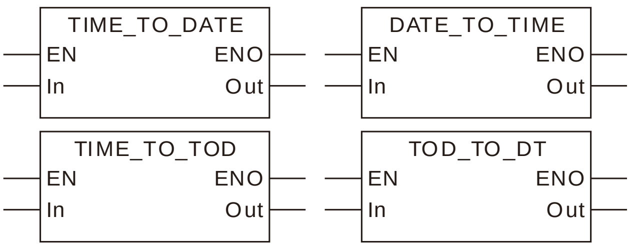
- **Function Explanation**

  - **Time and Date to Bool, Bit String, Integer, Real Number and String**

    The rule for the conversion of the time and date into the bool, bit string, integer, real number and string is the same as that for the conversion of the unsigned integer into bool, bit string, integer, real number and string. Refer to section 8.13.5 for details.

  - **Time and Date to Time and Date**

    ➢ The time and date data can be converted to each other. And some instructions are shown below.



The rule for the conversion of the time and date data into the time and date data is the same as that for the conversion of unsigned integers into unsigned integers. The units must be uniform during the conversion. The unit of TIME is ns (nanosecond) and the unit of others is ms (millisecond).

- **Precautions for Correct Use**

  The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

8.12.6 Strings_TO_***

FB/FC	Explanation	Applicable model
FC	Strings_TO_*** instructions convert String data into the data of basic data types. "***" can be any basic data type.	DVP15MC11T

```
      ┌──────────────┐
      │ String_TO_***│
  ────┤EN         ENO├────
  ────┤In         Out├────
      └──────────────┘
```

- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Data to convert	Input	Data to convert	Depends on the data type of the variable that the input parameter is connected to.
Out	Conversion result	Output	Conversion result	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				●
Out	The data type of *Out* must be the same as "***" of the instruction name.																			

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - **String to Bool**
    - Relevant instructions:

```
      ┌───────────────┐
      │STRING_TO_BOOL │
  ────┤EN          ENO├────
  ────┤In          Out├────
      └───────────────┘
```

The rule for the String-to-Bool conversion is that the output Bool value is TRUE only when the string value is TRUE or true. Otherwise, the output is FALSE.

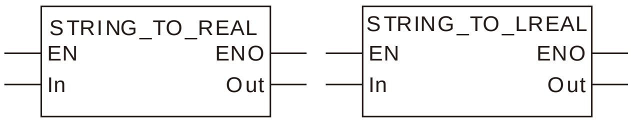
  - **String to Integer**
    - Strings can be converted to integers. And some instructions are shown below.

```
      ┌───────────────┐         ┌────────────────┐
      │STRING_TO_SINT │         │STRING_TO_USINT │
  ────┤EN          ENO├────  ───┤EN           ENO├────
  ────┤In          Out├────  ───┤In           Out├────
      └───────────────┘         └────────────────┘
      ┌───────────────┐         ┌────────────────┐
      │STRING_TO_DINT │         │STRING_TO_ULINT │
  ────┤EN          ENO├────  ───┤EN           ENO├────
  ────┤In          Out├────  ───┤In           Out├────
      └───────────────┘         └────────────────┘
```

- For the string-to-integer conversion, the string is required to be the integer value such as '123', '-123' and '+123'. The string like 'M123' is not allowed to convert to the integer. The conversion examples are shown in the following table.

Input value	Output result	
	Data type	Output value
'123'	SINT	123
'+123'	SINT	123
'-123'	SINT	-123
'M123'	SINT	The conversion is not allowed and the original value of the output variable is retained.

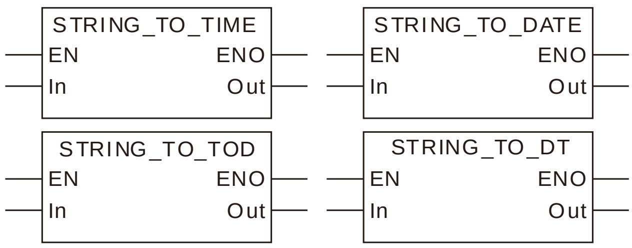
- **String to Real Number**
  - Strings can be converted to real numbers. And some instructions are shown below.



- For the string-to-real number conversion, the string is required to be the real number value such as '123', '-123.123' and '1.23e+5'. The conversion examples are shown in the following table.

Input value	Output result	
	Data type	Output value
'123'	REAL	123
'-123.123'	REAL	-123.123
'1.23e+5'	REAL	-1.23e+5
'M123.123'	REAL	The conversion is not allowed and the original value of the output variable is retained.

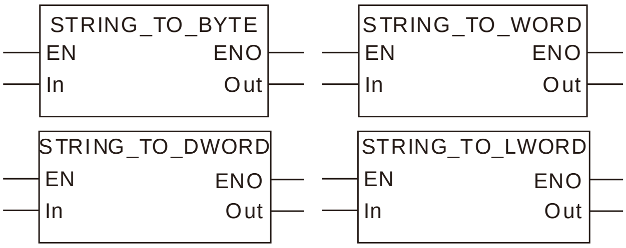
- **String to Time or Date**
  - Strings can be converted to times and dates. And some instructions are shown below.



- For the string-to-time or date conversion, the string is required to represent the time or date value such as 'T#1ns', 'D#1970-1-1', 'TOD#0:0:0' and 'DT#1970-1-1-0:0:0'. The conversion examples are shown in the following table.

Input value	Output result	
	Data type	Output value
'T#1ns'	TIME	T#1ns
'D#1970-1-1'	DATE	D#1970-1-1
'TOD#0:0:0'	TOD	TOD#0:0:0
'DT#1970-1-1-0:0:0'	DT	DT#1970-1-1-0:0:0

- **String to Bit String**
  - Strings can be converted to bit strings. And some instructions are shown below.



The rule for the string-to-bit string conversion is the same as that for the string-to integer conversion.

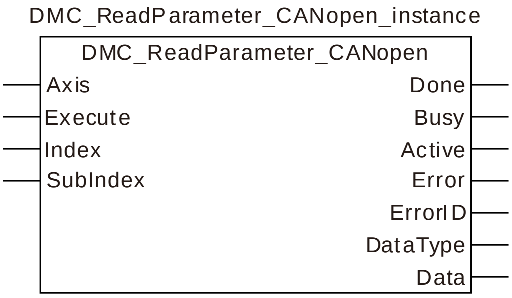
- **Precautions for Correct Use**

The input variable is not allowed to omit. An error will occur during the compiling of the software if the input variable is omitted. But the output variable is allowed to omit.

8.13 CANopen Communication Instructions

8.13.1 DMC_ReadParameter_CANopen

FB/FC	Explanation	Applicable model
FB	DMC_ReadParameter_CANopen is used to read a parameter value of a slave.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the slave which is to be controlled by the instruction	USINT	1~127 (0)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
Index	The index of a parameter to be read	UINT	0	When *Execute* changes from FALSE to TRUE
SubIndex	The subindex of a parameter to be read	USINT	0	When *Execute* changes from FALSE to TRUE

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
Error	TRUE when there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	
Data Type	The data type of the read parameter. 1: Byte, 2: Word, 4: Double Word	USINT	
Data	The value of the parameter which has been read	UDINT	

- **The index and subindex of the slave parameter to be read:**

1. The user-defined parameter is a servo drive parameter to be read. The data length is specified by users according to the data type of the read parameter. The data length of the byte

parameter is 1, the data length of the word parameter is 2 and the data length of the double-word parameter is 4.

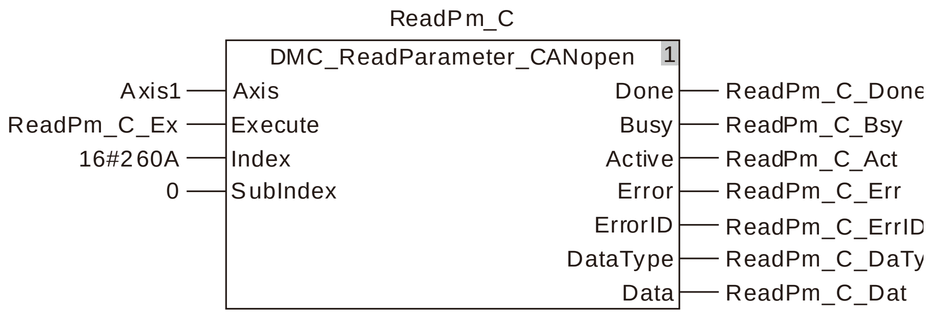
The method of calculating the index and subindex of a servo drive parameter:
Index= a servo drive parameter value (Hex) + 2000 (Hex)
Subindex= 0.

Example:

The index and subindex of the servo drive parameter P6-10 are [2000 + 060A (the hexdecimal value of P6-10)] 260A and 0 respectively.

➢ **The variable table and program**

Variable name	Data type	Initial value
ReadPm_C	DMC_ReadParameter_CANopen	
ReadPm_C_Ex	BOOL	FALSE
ReadPm_C_Done	BOOL	
ReadPm_C_Bsy	BOOL	
ReadPm_C_Act	BOOL	
ReadPm_C_Err	BOOL	
ReadPm_C_ErrID	WORD	
ReadPm_C_DaTy	USINT	
ReadPm_C_Dat	UDINT	

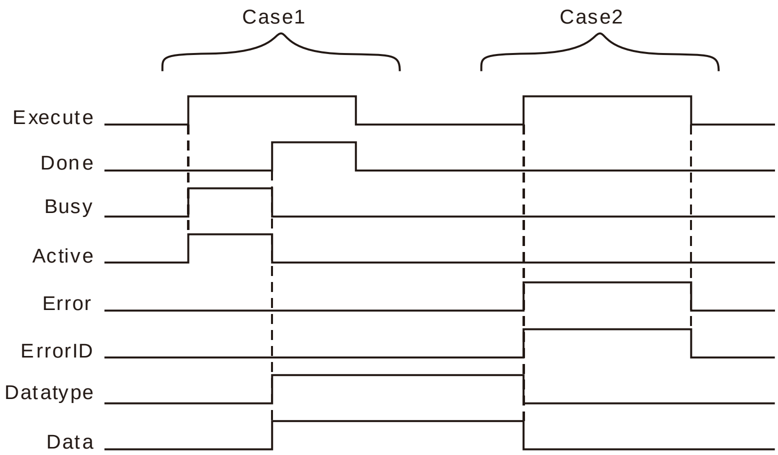


2. For the index and subindex of other slave parameters, refer to CANopen-related manual of the slave.

● **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the reading of the parameter content is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed.
Busy	◆ When *Execute* changes to TRUE	◆ When *Error* changes to TRUE ◆ When *Done* changes from FALSE to TRUE
Active	◆ When the slave starts being controlled by the instruction	◆ When Error changes to TRUE ◆ When Done changes from FALSE to TRUE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Output Timing Chart**



Case 1 : *Busy* and *Active* change to TRUE when *Execute* changes from FALSE to TRUE and one period later, *Done* changes to TRUE and *Datatype* and *Data* show corresponding data. When *Done* changes to TRUE, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *Done* changes from TRUE to FALSE and *Datatype* and *Data* retain original values.

Case 2 : Before DMC_ReadParameter_CANopen is executed, the input parameter value such as axis No: 0 is illegal. When *Execute* changes from FALSE to TRUE, *Error* changes from FALSE to TRUE, the values of Datatype and Data are cleared to 0 and *ErrorID* shows corresponding error codes. As *Execute* changes from TRUE to FALSE, *Error* changes from TRUE to FALSE and the content of *ErrorID* is cleared to 0.

- **Functions**

DMC_ReadParameter_CANopen is used to read the parameter value of a slave. Users can specify the index and subindex of the parameter to be read.

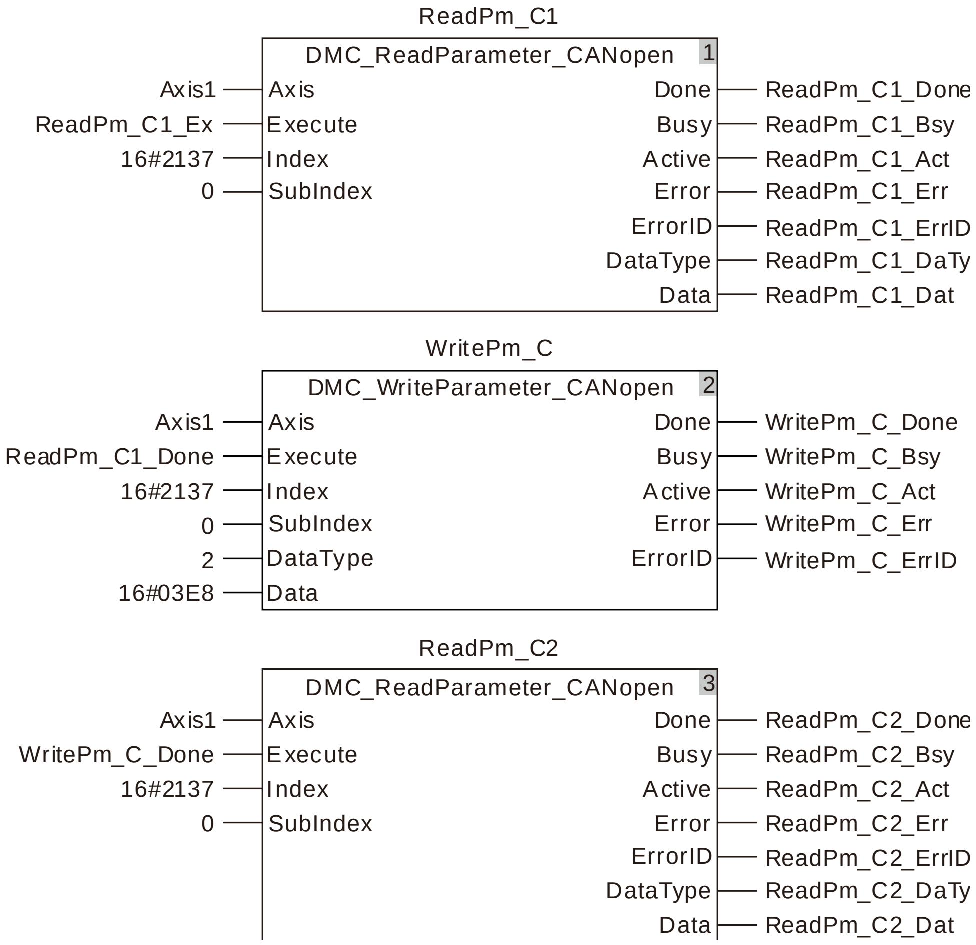
Programming Example

Below is an example of DMC_ReadParameter_CANopen instruction execution.

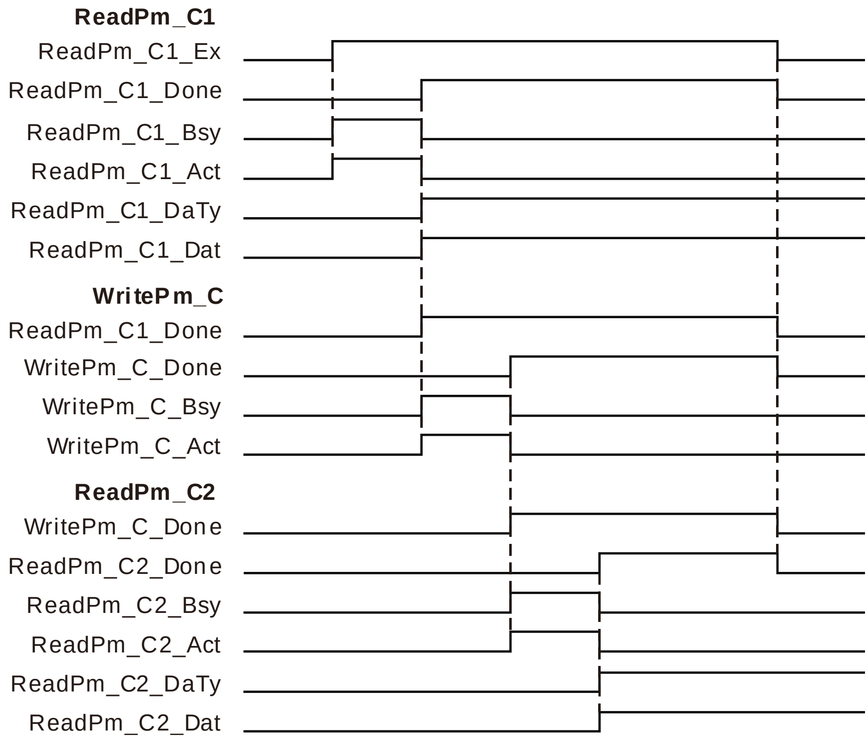
- **The variable table and program**

Variable name	Data type	Current value
ReadPm_C1	DMC_ReadParameter_CANopen	
Axis1	USINT	1
ReadPm_C1_Ex	BOOL	TRUE
ReadPm_C1_Done	BOOL	TRUE
ReadPm_C1_Bsy	BOOL	FALSE
ReadPm_C1_Act	BOOL	FALSE
ReadPm_C1_Err	BOOL	FALSE
ReadPm_C1_ErrID	WORD	FALSE
ReadPm_C1_DaTy	USINT	2
ReadPm_C1_Dat	UDINT	5000
WritePm_C	DMC_WriteParameter_CANopen	
WritePm_C_Done	BOOL	TRUE
WritePm_C_Bsy	BOOL	FALSE

Variable name	Data type	Current value
WritePm_C_Act	BOOL	FALSE
WritePm_C_Err	BOOL	FALSE
WritePm_C_ErrID	WORD	FALSE
ReadPm_C2	DMC_ReadParameter_CANopen	
ReadPm_C2_Done	BOOL	TRUE
ReadPm_C2_Bsy	BOOL	FALSE
ReadPm_C2_Act	BOOL	FALSE
ReadPm_C2_Err	BOOL	FALSE
ReadPm_C2_ErrID	WORD	FALSE
ReadPm_C2_DaTy	USINT	2
ReadPm_C2_Dat	UDINT	1000



■ Timing Chart



❖ The first DMC_ReadParameter_CANopen starts being executed as ReadPm_C1_Ex changes from FALSE to TRUE. When the execution of the first DMC_ReadParameter_CANopen is completed, ReadPm_C1_Done changes to TRUE, ReadPm_C1_DaTy = 2 and ReadPm_C1_Dat=5000.

That is, the content of the servo slave parameter P1-55 which is read is 5000. (The maximum speed of the servo is limited to 5000rpm.)

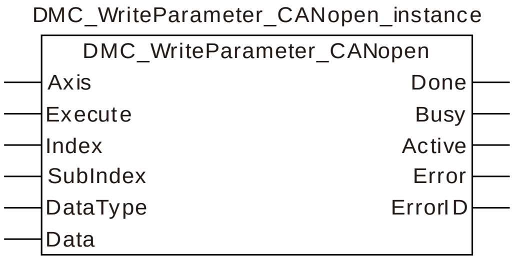
❖ As ReadPm_C1_Done changes from FALSE to TRUE, DMC_WriteParameter_CANopen starts being executed. When the DMC_WriteParameter_CANopen instruction execution is completed, WritePm_C_Done changes to TRUE. That is, 1000 is written as the content of the servo slave parameter P1-55. (The maximum speed of the servo is limited to 1000rpm.)

❖ The second DMC_ReadParameter_CANopen is executed as WritePm_C_Done changes from FALSE to TRUE. When the execution of the second DMC_ReadParameter_CANopen is completed, ReadPm_C2_Done changes to TRUE, ReadPm_C2_DaTy = 2 and ReadPm_C2_Dat=1000. That is, the read content of the servo slave parameter P1-55 is 1000. (The maximum speed of the servo is limited to 1000rpm.)

8

8.13.2 DMC_WriteParameter_CANopen

FB/FC	Explanation	Applicable model
FB	DMC_WriteParameter_CANopen is used to set a parameter value of a slave.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the slave which is to be controlled by the instruction	USINT	1~127 (0)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
Index	The index of a parameter which is set	UINT		
SubIndex	The subindex of a parameter which is set	USINT		
DataType	The data type of the parameter which is set 1： Byte, 2： Word, 4： Double Word.	USINT		
Data	The content value of the parameter which is set	UDINT		

Notes:

1. The value of *DataType* must indicate the data type of the parameter which is set. If the filled value is incorrect, an error will occur in the instruction.
2. For the method of calculating the index and subindex of CANopen slave parameter, refer to Introduction of Axis Parameters in Chapter 9.

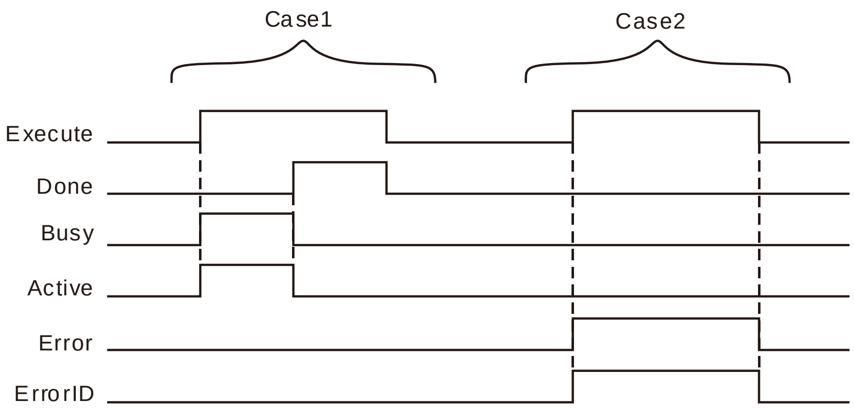
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
Error	TRUE when there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the writing of the parameter content is completed	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed
Busy	◆ When *Execute* changes to TRUE	◆ When *Error* changes to TRUE ◆ When *Done* changes from FALSE to TRUE
Active	◆ When the slave starts being controlled by the instruction	◆ When *Error* changes to TRUE ◆ When *Done* changes from FALSE to TRUE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Execute* changes from TRUE to FALSE

- **Timing Chart**



Case 1： *Busy* and *Active* change to TRUE when *Execute* changes from FALSE to TRUE and one period later, *Done* changes to TRUE. When *Done* changes to TRUE, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *Done* changes from TRUE to FALSE.

Case 2： Before DMC_WriteParameter_CANopen is executed, the input parameter value such as axis No: 0 is illegal. After *Execute* changes from FALSE to TRUE, *Error* changes from FALSE to TRUE and *ErrorID* shows corresponding error codes. As *Execute* changes from TRUE to FALSE, *Error* changes from TRUE to FALSE and the content of *ErrorID* is cleared to 0.

8

- **Function**

DMC_WriteParameter_CANopen is used to set the parameter value of a slave. Users can specify the index and subindex of the parameter which is to be set.

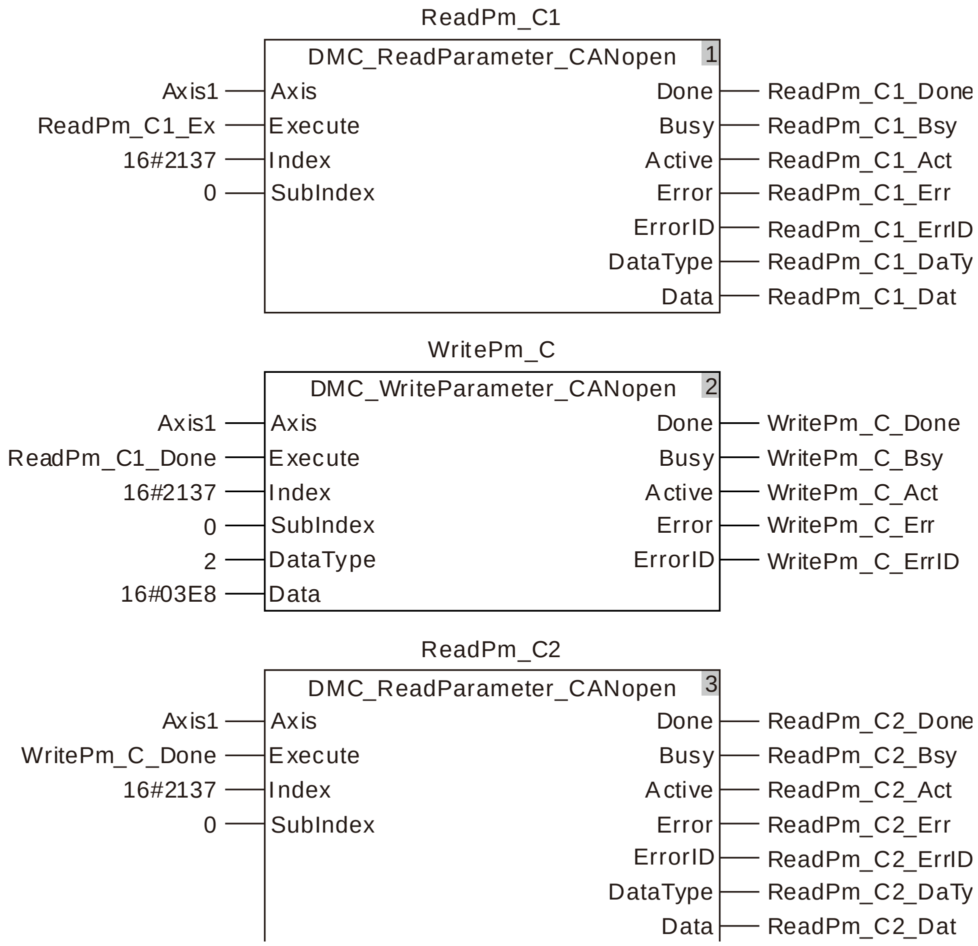
Programming Example

Below is an example of one DMC_WriteParameter_CANopen instruction execution.

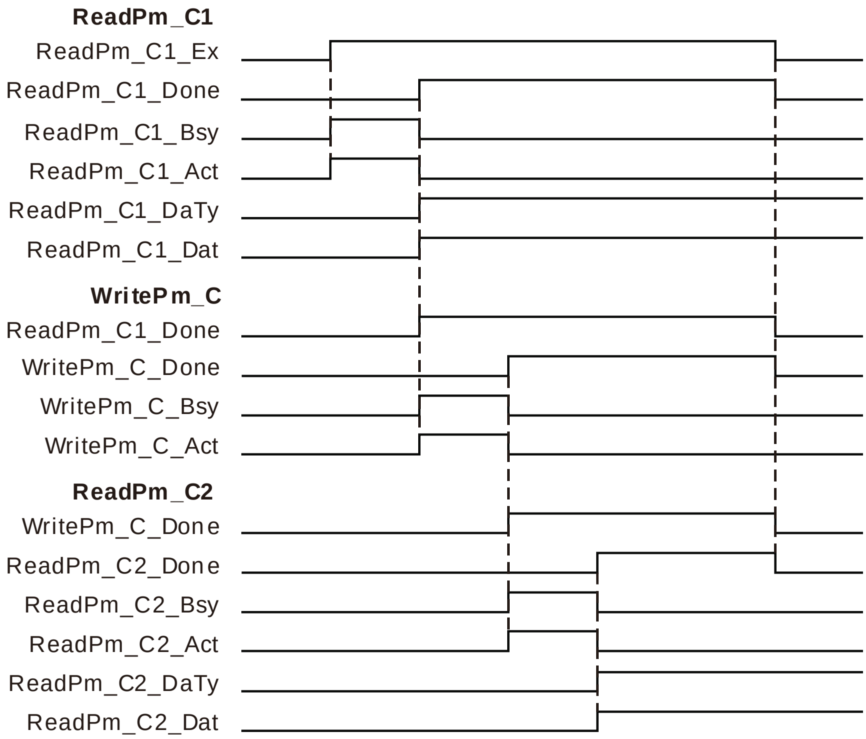
- **The variable table and program**

Variable name	Data type	Initial value
ReadPm_C1	DMC_ReadParameter_CANopen	

Variable name	Data type	Initial value
Axis1	USINT	1
ReadPm_C1_Ex	BOOL	TRUE
ReadPm_C1_Done	BOOL	TRUE
ReadPm_C1_Bsy	BOOL	FALSE
ReadPm_C1_Act	BOOL	FALSE
ReadPm_C1_Err	BOOL	FALSE
ReadPm_C1_ErrID	WORD	FALSE
ReadPm_C1_DaTy	USINT	2
ReadPm_C1_Dat	UDINT	5000
WritePm_C	DMC_WriteParameter_CANopen	
WritePm_C_Done	BOOL	TRUE
WritePm_C_Bsy	BOOL	FALSE
WritePm_C_Act	BOOL	FALSE
WritePm_C_Err	BOOL	FALSE
WritePm_C_ErrID	WORD	FALSE
ReadPm_C2	DMC_ReadParameter_CANopen	
ReadPm_C2_Done	BOOL	TRUE
ReadPm_C2_Bsy	BOOL	FALSE
ReadPm_C2_Act	BOOL	FALSE
ReadPm_C2_Err	BOOL	FALSE
ReadPm_C2_ErrID	WORD	FALSE
ReadPm_C2_DaTy	USINT	2
ReadPm_C2_Dat	UDINT	1000


ReadPm_C1
DMC_ReadParameter_CANopen
1
Axis1
ReadPm_C1_Ex
16#2137
0
Axis
Execute
Index
SubIndex
Done
Busy
Active
Error
ErrorID
DataType
Data
ReadPm_C1_Done
ReadPm_C1_Bsy
ReadPm_C1_Act
ReadPm_C1_Err
ReadPm_C1_ErrID
ReadPm_C1_DaTy
ReadPm_C1_Dat
WritePm_C
DMC_WriteParameter_CANopen
2
Axis1
ReadPm_C1_Done
16#2137
0
2
16#03E8
Axis
Execute
Index
SubIndex
DataType
Data
Done
Busy
Active
Error
ErrorID
WritePm_C_Done
WritePm_C_Bsy
WritePm_C_Act
WritePm_C_Err
WritePm_C_ErrID
ReadPm_C2
DMC_ReadParameter_CANopen
3
Axis1
WritePm_C_Done
16#2137
0
Axis
Execute
Index
SubIndex
Done
Busy
Active
Error
ErrorID
DataType
Data
ReadPm_C2_Done
ReadPm_C2_Bsy
ReadPm_C2_Act
ReadPm_C2_Err
ReadPm_C2_ErrID
ReadPm_C2_DaTy
ReadPm_C2_Dat

■ Timing chart

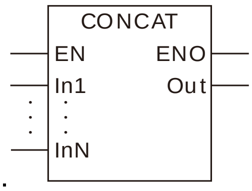


- ❖ When ReadPm_C1_Ex changes from FALSE to TRUE, the first DMC_ReadParameter_CANopen starts being executed. After the execution of the first DMC_ReadParameter_CANopen is completed, ReadPm_C1_Done changes to TRUE, ReadPm_C1_DaTy =2 and ReadPm_C1_Dat=5000. That is, the content of the servo slave parameter P1-55 which is read is 5000. (The maximum speed of the servo is limited to 5000rpm.)
- ❖ When ReadPm_C1_Done changes from FALSE to TRUE, the DMC_ WriteParameter _CANopen instruction starts being executed. After the execution of the DMC_WriteParameter_CANopen instruction is completed, WritePm_C_Done changes to TRUE. That is, the content of the servo slave parameter P1-55 which is written is 1000. (The maximum speed of the servo is limited to 1000rpm.)
- ❖ When WritePm_C_Done changes from FALSE to TRUE, the second DMC_ ReadParameter _CANopen instruction starts being executed. After the execution of the second DMC_ ReadParameter _CANopen instruction is completed, ReadPm_C2_Done changes to TRUE, ReadPm_C2_DaTy =2 and ReadPm_C2_Dat=1000. That is, the content of the servo slave parameter P1-55 which is read is 1000. (The maximum speed of the servo is limited to 1000rpm.)

8.14 String Processing Instructions

8.14.1 CONCAT

FB/FC	Explanation	Applicable model
FC	CONCAT joins two or more string variables or constants together to form a new string.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1 to InN	Strings to join	Input	The joined parameter can be added or removed while the program is being written. The maximum number of joined parameters is 8 and the minimum number is 2. N=2~8.	Depends on the data type of the variable that the input parameter is connected to.
Out	Result of joining	Output	String resulted from joining	Depends on the data type of the variable that the output parameter is connected to.

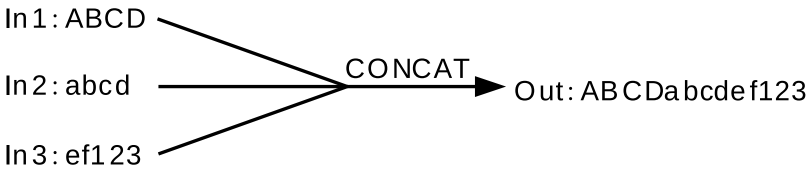
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1 to InN																				●
Out																				●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  The CONCAT instruction joins two or more strings to form a new string and the new string is output to *Out*. The parameters from In1 to InN are joined in order as shown in the following figure.



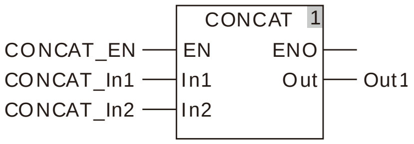
- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

- The data types of CONCAT_In1, CONCAT_In2 and Out1 are strings and the values of CONCAT_In1 and CONCAT_In2 are ’Asasz’ and ’B1255’ respectively. When CONCAT_EN is TRUE, the value of Out1 is ’AsaszB1255’.

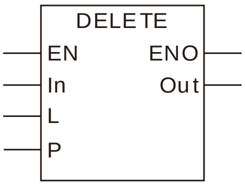
  - **The variable table and program**

Variable name	Data type	Current value
CONCAT_EN	BOOL	TRUE
CONCAT_In1	STRING	‘Asasz’
CONCAT_In2	STRING	‘B1255’
Out1	STRING	‘AsaszB1255’



8.14.2 DELETE

FB/FC	Explanation	Applicable model
FC	DELETE deletes the specified-length string from the specified position from the string variable or constant.	DVP15MC11T



- **Parameters**

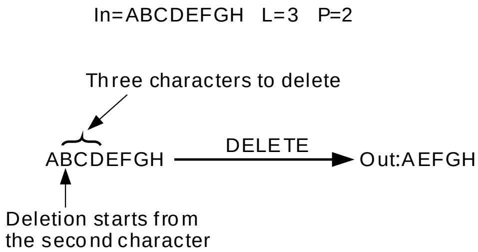
Parameter name	Meaning	Input/ Output	Description	Valid range
In	String for deletion	Input	String for deletion	Depends on the data type of the variable that the input parameter is connected to.
L	Number of characters to delete	Input	Number of characters to delete	0~ maximum length of the string
P	Deletion start position	Input	Deletion start position	1~ maximum length of the string
Out	Deletion result	Output	String after deletion	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				●
L							●													
P							●													
Out																				●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The DELETE instruction deletes L characters starting from the position specified by *P* of the *In* string and the characters after deletion will be output to *Out*. The deletion way is illustrated as below.

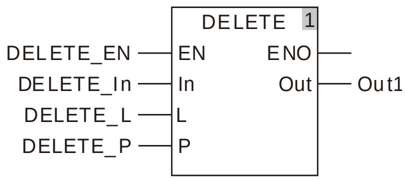


- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

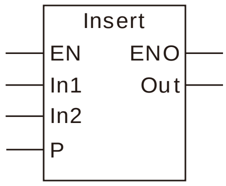
- DELETE_In is 'AaBbCcDd', DELETE_L= 2 and DELETE_P = 3. When DELETE_EN is TRUE, Out1 is'AaCcDd'.
  - **The variable table and program**

Variable name	Data type	Current value
DELETE_EN	BOOL	TRUE
DELETE_In	STRING	'AaBbCcDd'
DELETE_L	UINT	2
DELETE_P	UINT	3
Out1	STRING	'AaCcDd'



8.14.3 INSERT

FB/FC	Explanation	Applicable model
FC	INSERT inserts a string to the specified position in the string variable or constant.	DVP15MC11T



- **Parameters**

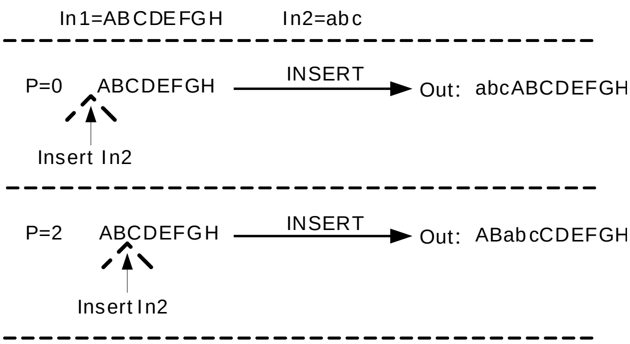
Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Original string	Input	Original string	Depends on the data type of the variable that the input parameter is connected to.
In2	String to insert	Input	String to insert	Depends on the data type of the variable that the input parameter is connected to.
P	Insertion start position	Input	Insertion start position	0~ maximum length of the string
Out	Insertion result	Output	Staring after insertion	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				●
In2																				●
P							●													
Out																				●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The INSERT instruction inserts the *In2* string into the *In1* string and the new string is output to *Out.* The insertion position is between the position specified by *P* and the position specified by *P+1* of the characters in *In1*. If P =0, the *In2* string is inserted at the start of the *In1* string. The insertion way is illustrated as below.

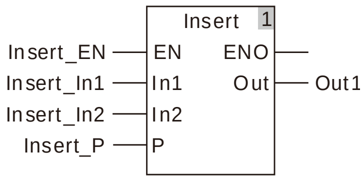


- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

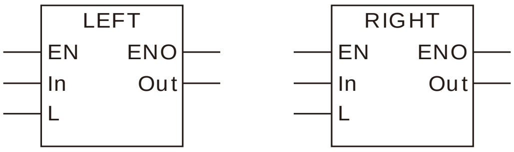
- Insert_In1 is 'AaBbCcDd', Insert_In2 is 'Ee' and Insert_P=2. When Insert_EN is TRUE, Out1 is 'AaEeBbCcDd'.
  - **The variable table and program**

Variable name	Data type	Current value
Insert_EN	BOOL	FALSE
Insert_In1	STRING	'AaBbCcDd'
Insert_In2	STRING	'Ee'
Insert_P	UINT	2
Out1	STRING	'AaEeBbCcDd'



8.14.4 LEFT / RIGHT

FB/FC	Explanation	Applicable model
FC	LEFT/RIGHT extracts a specified-length string from the string variable or constant.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Original string	Input	Original string	Depends on the data type of the variable that the input parameter is connected to.
L	Number of characters to get	Input	Number of characters to get	0~maximum number of characters
Out	Extraction result	Output	Extraction result	Depends on the data type of the variable that the output parameter is connected to.

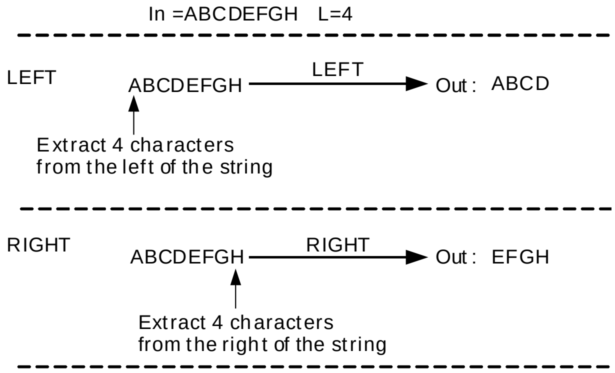
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				●
L							●													
Out																				●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

8

- **Function Explanation**
  - The LEFT/RIGHT instruction extracts a specified-length string from the string *In* and the extracted string is output to *Out*. The LEFT instruction extracts characters from the left of the string *In* and the RIGHT instruction extracts characters from the right of the string. The way of extracting characters is illustrated as below.

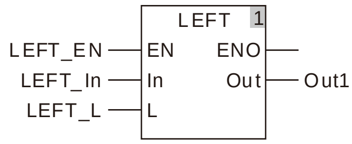


- **Precautions for Correct Use**
  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

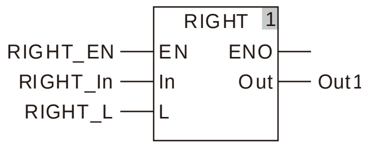
- When the LEFT_In string is 'AaBbCcDd', LEFT_L=2 and LEFT_EN is TRUE, Out1 is 'Aa' as shown in the following table 1. When the RIGHT_In string is 'AaBbCcDd', RIGHT_L=2 and RIGHT_EN is TRUE, Out1 is 'Dd' as shown in the following table 2.
  - **The variable table and program 1**

Variable name	Data type	Current value
LEFT_EN	BOOL	TRUE
LEFT_In	STRING	'AaBbCcDd'
LEFT_L	UINT	2
Out1	STRING	'Aa'



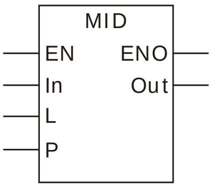
  - **The variable table and program 2**

Variable name	Data type	Current value
RIGHT_EN	BOOL	TRUE
RIGHT_In	STRING	'AaBbCcDd'
RIGHT_L	UINT	2
Out1	STRING	'Dd'



8.14.5 MID

FB/FC	Explanation	Applicable model
FC	MID extracts a specified-length string from the specified character position of a string variable or constant.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	Original string	Input	Original string	Depends on the data type of the variable that the input parameter is connected to.
L	Length of characters to extract	Input	Number of characters to extract	0~ maximum number of characters
P	Extraction start position	Input	Extraction start position	1~ maximum number of characters
Out	Extraction result	Output	Extraction result	Depends on the data type of the variable that the output parameter is connected to.

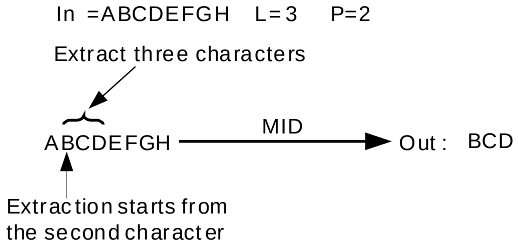
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				●
L							●													
P							●													
Out																				●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

  - The MID instruction extracts *L* characters starting from the number-P character of the *In* string. The extracted string is output to *Out*. The extraction way is illustrated as below.



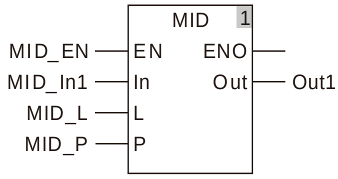
- **Precautions for Correct Use**

  - The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

- The MID_In string is ’AaBbCcDd’, MID_L=2 and MID_LP=3. When MID_EN is TRUE, Out1 is ’Bb’.
  - **The variable table and program**

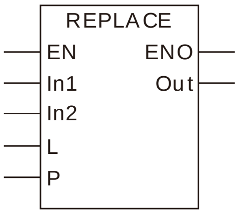
Variable name	Data type	Current value
MID_EN	BOOL	TRUE
MID_In	STRING	‘AaBbCcDd’
MID_L	UINT	2
MID_P	UINT	3
Out1	STRING	‘Bb’



8

8.14.6 REPLACE

FB/FC	Explanation	Applicable model
FC	The REPLACE instruction replaces the specified-length string starting from the specified position with another string.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	Original string	Input	Original string	Depends on the data type of the variable that the input parameter is connected to.
In2	Insert string	Input	String to insert	Depends on the data type of the variable that the input parameter is connected to.
L	Number of characters	Input	Number of characters to delete	0~ maximum number of characters
P	Replacement start position	Input	Replacement start position	1~ maximum number of characters
Out	Replacement result	Output	Replacement result	Depends on the data type of the variable that the output parameter is connected to.

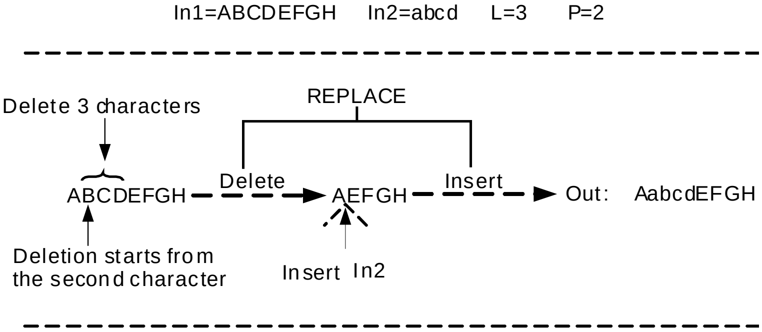
	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				●
In2																				●
L							●													
P							●													
Out																				●

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

The REPLACE instruction replaces L characters starting from the number-P character of the *In1* string by inserting another string *In2*. And the replacement result is output to *Out*. The replacement process is illustrated as below.



- **Precautions for Correct Use**

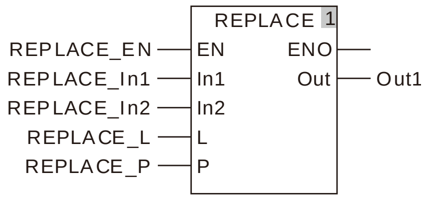
  ■ The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

■ The REPLACE_In1 string is ’AaBbCcDd’, the REPLACE_In2 string is ’DELTA’, REPLACE_L=2 and REPLACE_LP=3. When REPLACE_EN is TRUE, Out1 is ’AaDELTACcDd’.

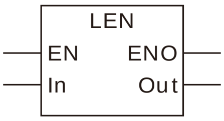
➢ **The variable table and program**

Variable name	Data type	Current value
REPLACE_EN	BOOL	TRUE
REPLACE_In1	STRING	‘AaBbCcDd’
REPLACE_In2	STRING	‘DELTA’
REPLACE_L	UINT	2
REPLACE_P	UINT	3
Out1	STRING	‘AaDELTACcDd’



8.14.7 LEN

FB/FC	Explanation	Applicable model
FC	LEN calculates the number of characters in a string.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In	String	Input	String	Depends on the data type of the variable that the input parameter is connected to.
Out	Number of characters	Output	Number of characters	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				●
Out							●													

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**

The LEN instruction finds the number of characters in a string and the result is output to *Out*. For example, when the string is ABCDEFGH, the value of *Out* is 8.

- **Precautions for Correct Use**

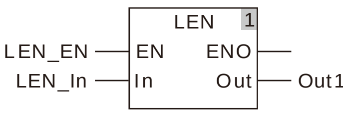
The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

■ The LEN_In string is ’AaBbCcDd’. As LEN_EN is TRUE, the value of Out1 is 8.

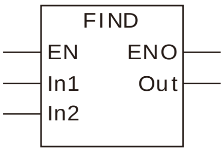
➢ **The variable table and program**

Variable name	Data type	Current value
LEN_EN	BOOL	TRUE
LEN_In	STRING	AaBbCcDd
Out1	UINT	8



8.14.8 FIND

FB/FC	Explanation	Applicable model
FC	FIND searches for the position of a specified string in another string.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
In1	String	Input	String	Depends on the data type of the variable that the input parameter is connected to.
In2	Key characters to search for	Input	Key characters to search for	Depends on the data type of the variable that the input parameter is connected to.
Out	Number of characters	Output	Number of characters	Depends on the data type of the variable that the output parameter is connected to.

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																				●
In2																				●
Out							●													

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The Find instruction takes the characters in *In2* as key characters and searches for the position of key characters in the string *In1*. For example, as *In1* is ABCDEFGH and *In2* is DE, the value of *Out* is 4.
  - The search starts from the first character in the string *In1.*
  - If multiple *In2* strings exist in *In1*, the value of *Out* is the position of the first *In2* from the beginning of *In1*.

- **Precautions for Correct Use**

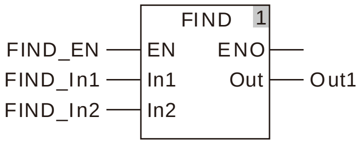
The input variables are not allowed to omit. An error will occur during the compiling of the software if the input variables are omitted. But the output variable is allowed to omit.

Programming Example

- The FIND_In1 string is ’AaBbCcDd’ and the FIND_In2 string is ’Cc’. As FIND_EN is TRUE, the value of Out1 is 5.

➢ **The variable table and program**

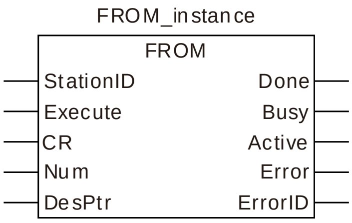
Variable name	Data type	Current value
FIND_EN	BOOL	TRUE
FIND_In1	STRING	‘AaBbCcDd’
FIND_In2	STRING	‘Cc’
Out1	UINT	5



8.15 Immediate Refresh Instructions

8.15.1 FROM

FB/FC	Explanation	Applicable model
FB	The FROM instruction reads the values in CR registers of the left-side and righ-side extension modules.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
StationID	The position of the extension module connected to the left side or right side of DVP15MC11T	USINT	Position range of left-side module: 100~107 Position range of right-side special module: 0~7 The position of the first module at the left side of DVP15MC11T is 100 and the position of the first module at the right side of DVP15MC11T is 0. (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (The variable value must be set)	-
CR	The number of the first CR (Controlled Register) to be read	UINT	0~max. CR number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Num	Number of CR registers which are to be read	USINT	1~64 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
DesPtr	The CR values read by the instruction	INT or DINT	The range of the data type of the read CR value (The variable value must be set)	When *Execute* changes from FALSE to TRUE

- **Output Parameters**

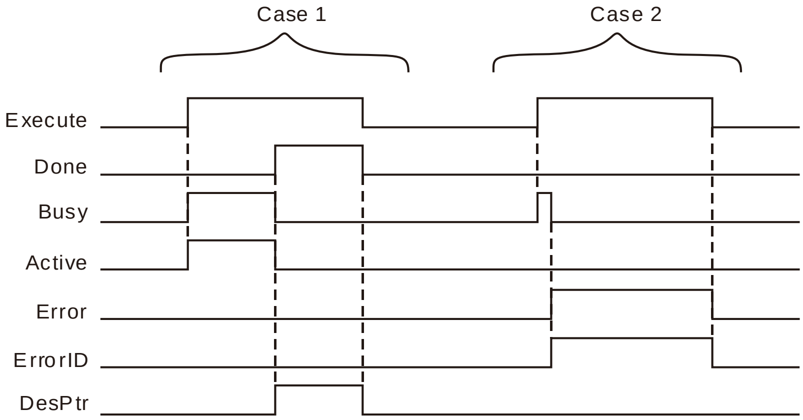
Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE

Parameter name	Function	Data type	Valid range
Active	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	-

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the reading of the parameter values is finished.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed.
Busy	◆ When *Execute* changes to TRUE	◆ When *Done* changes from FALSE to TRUE ◆ When *Error* changes to TRUE.
Active	◆ When the instruction execution begins	◆ When *Error* changes to TRUE. ◆ When *Done* changes from FALSE to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1 : When *Execute* changes from FALSE to TRUE, *Busy* and *Active* change to TRUE and one period later, *Done* changes to TRUE. Meanwhile *Busy* and *Active* change to FALSE and *DesPtr* shows the corrsponding data in CR registers of the extension module. When *Execute* changes from TRUE to FALSE, *Done* changes from TRUE to FALSE and the value of *DesPtr* is cleared to *0*.

Case 2 : When an error occurs as *Execute* is TRUE, *Error* changes from FALSE to TRUE and *ErrorID* shows corresponding error codes. *Error* changes from TRUE to FALSE and the value in ErrorID is cleared to 0 after *Execute* changes from TRUE to FALSE.

- **Function Explanation**

The FROM instruction can be applied to read the values in the registers of the left-side and righ-side extension modules.

The position of the left-side and right-side module is specified by *StationID*. The Station ID range of right-side module is 0~7. 0 represents the first extension analog module at the right side and 7 means the eight extension analog module at the right side. The Station ID range of left-side modules is 100~107. 100 represents the first extension module at the left side and 107 means the eight extension module at the left

8

side. If the Standard ID range exceeds the specified range of the left-side and right side module, an error will occur in the instruction execution.

If more than one CR register need be read by the instruction, the parameter *DesPtr* need be defined as the N^{th} element of an array. The data in the first CR register will be read to the N^{th} element of the array, the data in the second CR register will be read to the $N+1^{th}$ element and so on. By doing so, the data in mutiple CR registers will be all read to the array. Refer to Programming Example 2 for details.

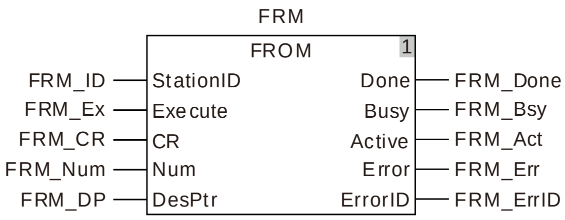
Precaution

Maximum 8 extension modules are connectable to the left side and Maximum 8 special modules are connectable to the right side of DVP15MC11T. Digital modules have no position number. For example, if DVP04AD-S, DVP16SP11T and DVP04DA-S are connected to the right side of DVP15MC11T one after another, the *StationID* value of DVP04AD-S is 0 and the *StationID* value of DVP04DA-S is 1.

Programming Example

- **The variable table and program**

Variable name	Data type	Current value
FRM	FROM	
FRM_ID	USINT	0
FRM _Ex	BOOL	FALSE
FRM _CR	UINT	0
FRM _Num	USINT	1
FRM _DP	INT	
FRM _Done	BOOL	
FRM _Bsy	BOOL	
FRM _Act	BOOL	
FRM _Err	BOOL	
FRM _ErrID	WORD	



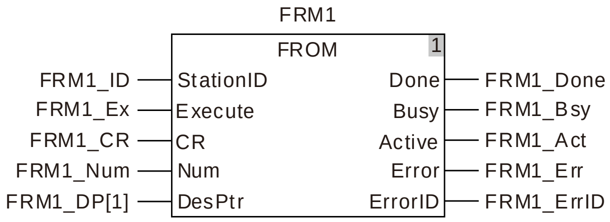
DVP-04AD is connected to the right side of DVP15MC11T. When FRM _Ex changes from FALSE to TRUE and FRM _Bsy and FRM _Act change to TRUE simultaneously, FROM instruction starts to execute. When FRM _Done changes to TRUE, the instruction execution is finished. FRM _DP displays that the value in CR0 read by the instruction is 136 and thus the version of DVP-04AD is 1.36.

Programming Example

- **The variable table and program**

Variable name	Data type	Current value
FRM1	FROM	
FRM1_ID	USINT	0
FRM1_Ex1	BOOL	FALSE
FRM1_CR1	UINT	2
FRM1_Num1	USINT	4
FRM1_DP	Array[1..4] of INT	
FRM1_Done	BOOL	
FRM1_Bsy	BOOL	
FRM1_Act	BOOL	

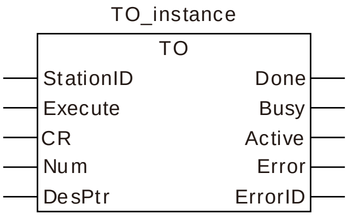
Variable name	Data type	Current value
FRM1_Err	BOOL	
FRM1_ErrID	WORD	



DVP-04AD is connected to the right side of DVP15MC11T. When FRM1_Ex changes from FALSE to TRUE and FRM1_Bsy and FRM1_Act change to TRUE simultaneously, FROM instruction starts to execute. When FRM1_Done changes to TRUE, the instruction execution is finished. The values read from CR2, CR3, CR4 and CR5 are stored in the four elements FRM1_DP[1], FRM1_DP[2], FRM1_DP[3] and FRM1_DP[4] of the FRM1_DP array.

8.15.2 TO

FB/FC	Explanation	Applicable model
FB	The TO instruction writes data to the specified CR registers of the left-side module and right-side module.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
StationID	The position of the extension module connected to the left side or right side of DVP15MC11T	USINT	Position range of left-side module: 100~107 Position range of right-side special module: 0~7 The position of the first module at the left side of DVP15MC11T is 100 and the position of the first module at the right side of DVP15MC11T is 0. (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
CR	The number of the first CR (Controlled Register) to be read	UINT	0~ max. CR number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Num	Number of CR registers which are to be read	USINT	1~64 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
DesPtr	The CR value written by the instruction	INT or DINT	The range of the data type of the written CR value (The variable value must be set)	When *Execute* changes from FALSE to TRUE

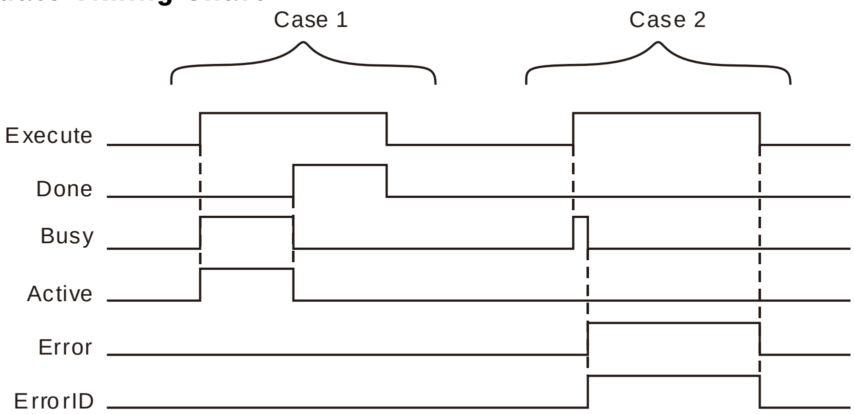
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	-

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the writing of the parameter values is finished.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed
Busy	◆ When *Execute* changes to TRUE	◆ When *Done* changes from FALSE to TRUE ◆ When *Error* changes to TRUE.
Active	◆ When the instruction execution begins	◆ When *Error* changes to TRUE. ◆ When *Done* changes from FALSE to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1：When *Execute* changes from FALSE to TRUE, *Busy* and *Active* change to TRUE. One period later, *Done* changes to TRUE. Meanwhile *Busy* and *Active* changes from TRUE to FALSE. After *Execute* changes from TRUE to FALSE, *Done* changes from TRUE to FALSE.

Case 2：When an error occurs as *Execute* changes from FALSE to TRUE, *Error* changes from FALSE to TRUE and *ErrorID* shows corresponding error codes. *Error* changes from TRUE to FALSE and the value in ErrorID is cleared to 0 after *Execute* changes from TRUE to FALSE.

- **Function Explanation**

The TO instruction is used to write data to the specified CR registers of the left-side module and right-side module.

The positions of left-side and right-side modules are specified by *StationID*. The *StationID* range of right-side module is 0~7. 0 represents the first extension analog module at the right side. 7 is the eighth extension analog module at the right side. The *StationID* range of left-side module is 100~107. 100 is the first extension module at the left side. 107 is the eighth extension analog module at the left side. If *StationID* value exceeds the specified range for left-side and right-side modules, an error will occur in execution of the instruction.

If the instruction is used to write values to multiple CR registers, *DesPtr* need be defined as the N^{th} element of the array. Then multiple values will be written to multiple CR registers by writing the N^{th} element value to the first CR, the $N+1^{th}$ element value to the second CR and so on after execution of the instruction.

Refer to the following program examples for more details on the usage.

⚠ **Precaution**

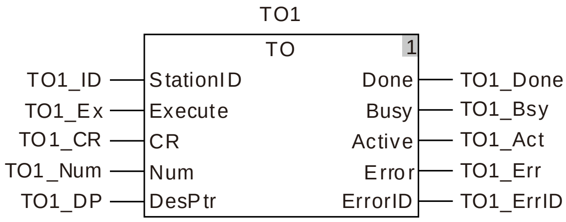
Maximum 8 extension modules are connectable to the left side and Maximum 8 special modules are connectable to the right side of DVP15MC11T. The right-side digital modules have no position number. For example, if DVP04AD-S, DVP16SP11T and DVP04DA-S are connected to the right side of DVP15MC11T one after another, the *StationID* value of DVP04AD-S is 0 and the *StationID* value of DVP04DA-S is 1.

8

Programming Example 1

■ **The variable table and program**

Variable name	Data type	Current value
TO1	TO	
TO1_ID	USINT	0
TO1_Ex	BOOL	FALSE
TO1_CR	UINT	2
TO1_Num	USINT	1
TO1_DP	INT	10
TO1_Done	BOOL	
TO1_Bsy	BOOL	
TO1_Act	BOOL	
TO1_Err	BOOL	
TO1_ErrID	WORD	

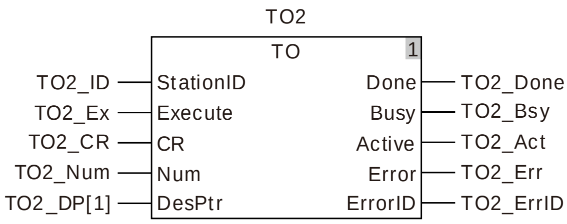


DVP-04AD is connected to the right side of DVP15MC11T. When TO1_Ex changes from FALSE to TRUE, TO1_Bsy and TO1_Act change to TRUE simultaneously and the TO instruction execution starts. When TO1_Done changes to TRUE, the instruction execution is finished and the value which is written to CR2 in DVP-04AD is 10.

Programming Example 2

■ **The variable table and program**

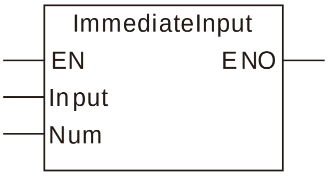
Variable name	Data type	Current value
TO2	TO	
TO2_ID	USINT	0
TO2_Ex	BOOL	FALSE
TO2_CR	UINT	2
TO2_Num	USINT	4
TO2_DP	Array[1..4] of INT	
TO2_Done	BOOL	
TO2_Bsy	BOOL	
TO2_Act	BOOL	
TO2_Err	BOOL	
TO2_ErrID	WORD	



DVP-04AD is connected to the right side of DVP15MC11T. When TO2_Ex changes from FALSE to TRUE, TO2_Bsy and TO2_Act change to TRUE simultaneously and the TO instruction execution starts. As TO2_Done changes to TRUE, the instruction execution is completed and the values written in CR2, CR3, CR4 and CR5 in DVP-04AD are the values written in the four elements TO2_DP[1], TO2_DP[2], TO2_DP[3] and TO2_DP[4] of the TO2_DP array respectively.

8.15.3 ImmediateInput

FB/FC	Explanation	Applicable model
FC	ImmediateInput is used for the immediate refresh of input points.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
Input	Start input point	Input	Start input point	0~15
Num	Number	Input	Number of input points for immediate refresh	1~16

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Input											●									
Num						●														

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

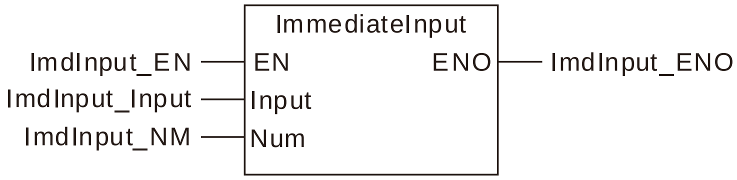
- **Function Explanation**
  - The ImmediateInput instruction is used for refreshing external input point status to %IX0.0~%IX0.15. If the ImmediateInput instruction does not exist, the controller refreshes external input point status to %IX0.0~%IX0.15 once only every time the program scan starts.
  - The Input parameter value 0~15 corresponds to %IX0.0~%IX0.15. Num represents the quantity of consecutive devices starting from the one specified by Input. E.g. when Input value is 0 and Num is 2, it indicates that the external input point status is refreshed to %IX0.0 and %IX0.1.

- **Precautions for Correct Use**
  - The instruction is only used for the immediate refresh of local input points instead of extension input points.

Programming Example

- **The variable table and program**

Variable name	Data type	Current value
ImdInput_EN	BOOL	FALSE
ImdInput_Input	INT	2
ImdInput_NM	USINT	2
ImdInput_ENO	BOOL	

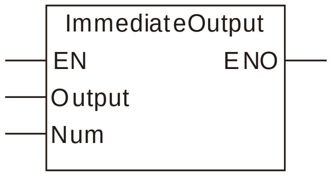


- **Program explanation**

  When the input variable ImdInput_EN is TRUE, the external hardware input points status will be refreshed to %IX0.2 and %IX0.3.

8.15.4 ImmediateOutput

FB/FC	Explanation	Applicable model
FC	ImmediateOutput is used for the immediate refresh of output points.	DVP15MC11T



- **Parameters**

Parameter name	Meaning	Input/ Output	Description	Valid range
Output	Start output point	Input	Start output point	0~7
Num	Number	Input	Number of output points for immediate refresh	1~8

	Boolean	Bit string				Integer								Real number		Time, date				String
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Output											●									
Num						●														

Note:

The symbol ● indicates that the parameter is allowed to connect to the variable or constant of the data type.

- **Function Explanation**
  - The ImmediateOutput instruction is used for refreshing current status of internal output point %QX0.0~%QX0.7 to external hardware output point. If the ImmediateOutput instruction does not exist, the controller refreshes internal output point status to external hardware output point. The status of %QX0.0~%QX0.7 is decided by other instructions. The ImmediateOutput instruction is only used for refreshing the status of %QX0.0~%QX0.7 to external hardware output points. The ImmediateOutput instruction does not control the TRUE or FALSE of %QX0.0~%QX0.7.
  - The *Output* parameter value 0~7 of the ImmediateOutput instruction corresponds to %QX0.0~%QX0.7. *Num* represents the quantity of consecutive devices starting from the one specified by *Output*. E.g. when *Output* value is 0 and *Num* is 2, it indicates that the status of %QX0.0 and %QX0.1 is refreshed to the external hardware output point.

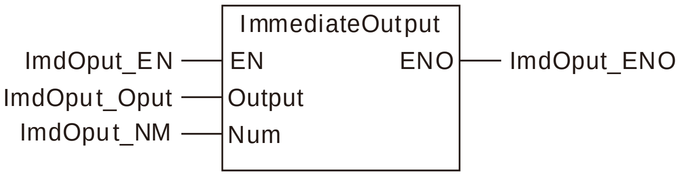
- **Precautions for Correct Use**

  The instruction is only used for the immediate refresh of local output points instead of extension output points.

Programming Example

- **The variable table and program**

Variable name	Data type	Current value
ImdOput_EN	BOOL	FALSE
ImdOput_Oput	INT	2
ImdOput_NM	USINT	2
ImdOput_ENO	BOOL	



- **Program Explanation**

  When the input variable ImdOput_EN is TRUE, the status of %QX0.2 and %QX0.3 will be refreshed to the external hardware output point.

Memo

Chapter 9 Introductions of Axis Parameters

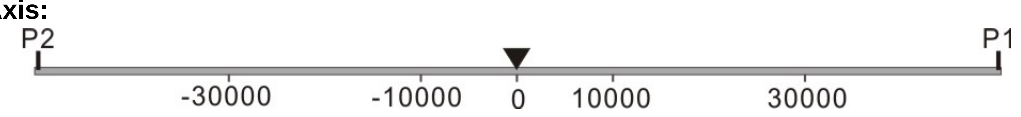
Table of Contents

9.1 Description of Axis Parameters 9-2

9.1 Description of Axis Parameters

Serial No	Parameter Name	Function	Data Type	Default Value
1	Name	Axis name	STRING	-
"Name" is a remark word only used for naming the servo drive without actual meaning.				
2	Node ID	CANopen node ID of an axis; range:1-32	USINT	-
"Node ID" is the CANopen station address of a servo drive.				
3	Axis type&unit	Axis type: linear axis/ rotary axis Unit: the unit of pitch (***UnitsPerRotation***). E.g. Users can fill mm (millimeter) or ° (degree) as a unit.	-	Linear axis

Linear Axis:

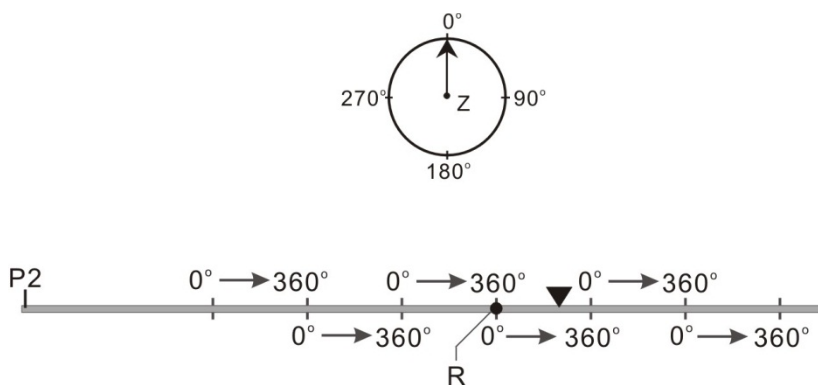


Linear Axis Model

Note:

P1	Positive Limit
P2	Negative Limit
▼	Servo Position

Rotary Axis :



Rotary Axis Model（"Modulo": 360）

Note:

P1	Positive Limit
P2	Negative Limit
▼	Servo Position
R	Home Position
Z	Axis of the servo motor

Difference between linear axis and rotary axis:

The rotary axis regards modulo as its cycle, which is the difference between linear axis and rotary axis. The position of the terminal actuator of the linear axis is 500 and the corresponding position of the rotary axis is 140 which is the remainder of 500 divided by modulo (360).

9

Serial No	Parameter Name	Function	Data Type	Default Value
4	Modulo	The cycle used for equally dividing the actual position of the terminal actuator.	LREAL	360
5	Software Limitation	Enables software limitation; If the item is not selected, the maximum/ minimum position of the axis which software limits is invalid. If the item is selected, the maximum/ minimum position of the axis limited by software is valid.	BOOL	0
6	Maximum Position	The maximum position of the axis limited by software	LREAL	-
7	Minimum Position	The minimum position of the axis limited by software	LREAL	-
8	Maximum Resolution	Maximum resolution for the number of servo pulses	UDINT	1280000
9	Unit Numerator	To set the number of pulses needed when the motor runs one rotation by adjusting the parameter and *Unit Denominator*.	UINT	128
10	Unit Denominator	To set the number of pulses needed when the motor runs one rotation by adjusting *Unit Numerator* and the parameter.	UINT	1
11	Pulses/rotation	How many pulses are needed when the servo motor runs one rotation.	UINT	10000

Unit Numerator and *Unit Denominator* jointly set the electronic gear ratio of the servo drive. The electronic gear ratio is used to set how many pulses the servo drive receives for one rotation that the servo motor runs.

The resolution of the servo motor is 1,280,000 pulses/rotation. Suppose the value of parameter 11 (Pulses/rotation) is N. So N*(Unit Numerator / Unit Denominator) = 1,280,000.

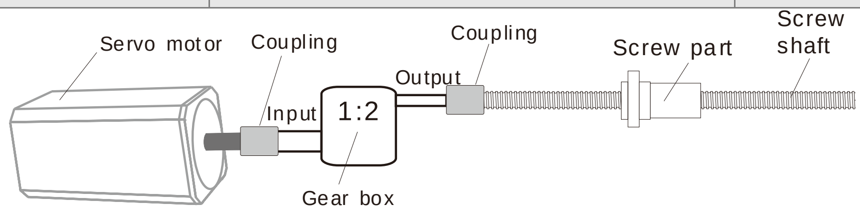
Serial No	Parameter Name	Function	Data Type	Default Value
12	InputRotation	This parameter and *OutputRotation* decide the mechanical gear ratio.	UINT	1
13	OutputRotation	*InputRotation* and this parameter decide the mechanical gear ratio.	UINT	1
14	UnitsPerRotation	The number of units which the terminal actuator moves while output end of the gear rotates for one circle.	UINT	10000

As illustrated below, ***InputRotation*** =1, ***OutputRotation*** =2, it means the input mechanism of gear box rotates for one circle and the output mechanism of gear box rotates for 2 circles. ***UnitsPerRotation*** represents the corresponding position (units) that ball screw moves while the output mechanism of gear box rotates for one circle.

E.g. If output mechanism of gear rotates for one circle and ball screw moves 1mm and ***UnitsPerRotation*** is set to 1, through the relative position motion instruction the ball screw will move 1 unit, i.e. the ball screw will move 1mm;

If ***UnitsPerRotation*** is set to 1000, the ball screw will move 1 unit through the MC_MoveRelative motion instruction, i.e. 1/1000mm actually. The unit of the position in the motion control instruction, G codes and electronic cam is Unit.

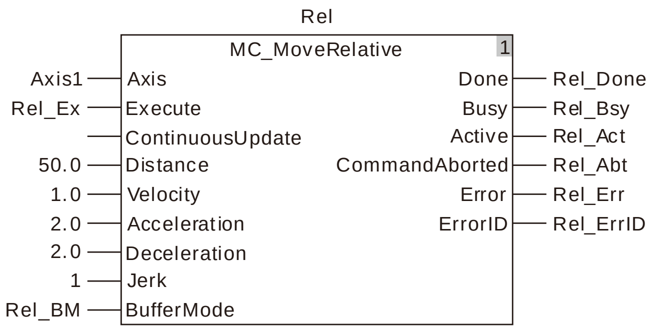
Serial No	Parameter Name	Function	Data Type	Default Value



As mentioned above, ***UnitsPerRotation*** is set to 1 and the ball screw will move 50 mm at the velocity 1mm/s, acceleration 2mm/ s^2 and change rate of acceleration 1mm/s^3 after the following MC_MoveRelative is executed.

The variable table and program

Variable name	Data type	Initial value
Rel	MC_MoveRelative	
Rel_Ex	BOOL	FALSE
Rel_BM	MC_Buffer_Mode	0
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	



15	Homing Mode	Set the homing mode of the servo drive; range: 1~ 35. See appendix D for more details.	UINT	1
16	Speed 1	The speed from starting homing to finding the home switch; Unit: rpm, setting range: 1-2000 rpm	UDINT	20
	Speed 2	The speed from finding the home switch to reaching the mechanical home; Unit: rpm, setting range: 1-500 rpm	UDINT	10

9

10

Chapter 10 Motion Control Function

Table of Contents

10.1 EN and ENO 10-2

10.2 Relation among Velocity, Acceleration and Jerk 10-3

10.3 Introduction of BufferMode 10-6

10.4 The State Machine 10-32

DVP15MC11T is a motion controller which is developed in compliance with CANopen DSP402 motion control protocol and the motion control instructions defined as function blocks are needed for it.
The motion control instructions for the MC module are based on the technical specifications of motion control function blocks in the PLCopen.
Below is the introduction of what need be known about while the motion instructions are used.

10.1 EN and ENO

When one instruction which is used has EN and ENO and EN is FALSE (0), the function defined by instruction will not be performed and the output values of the instruction will not be refreshed. On the contrary, the function defined by the instruction will be performed and the output values will be refreshed if EN is TRUE (1).

The output of ENO and the input of EN keep consistent with each other. ENO changes to TRUE while EN is TRUE. ENO changes to FALSE while EN is FALSE.

For the FB instruction, the instruction execution will continue as its EN changes from TRUE to FALSE after being executed. But the output values of the FB instruction will not be refreshed.

10.2 Relation among Velocity, Acceleration and Jerk

DVP15MC11T adopts the method of the quadratic-curve acceleration and deceleration. By means of the method, the S-type velocity waveform which is generated can reduce the mechanical shock effectively. In addition, at least the velocity (v), acceleration (Acc) or deceleration (Dec) and change rate of the acceleration (Jerk) need be specified while the motion control instructions are used.

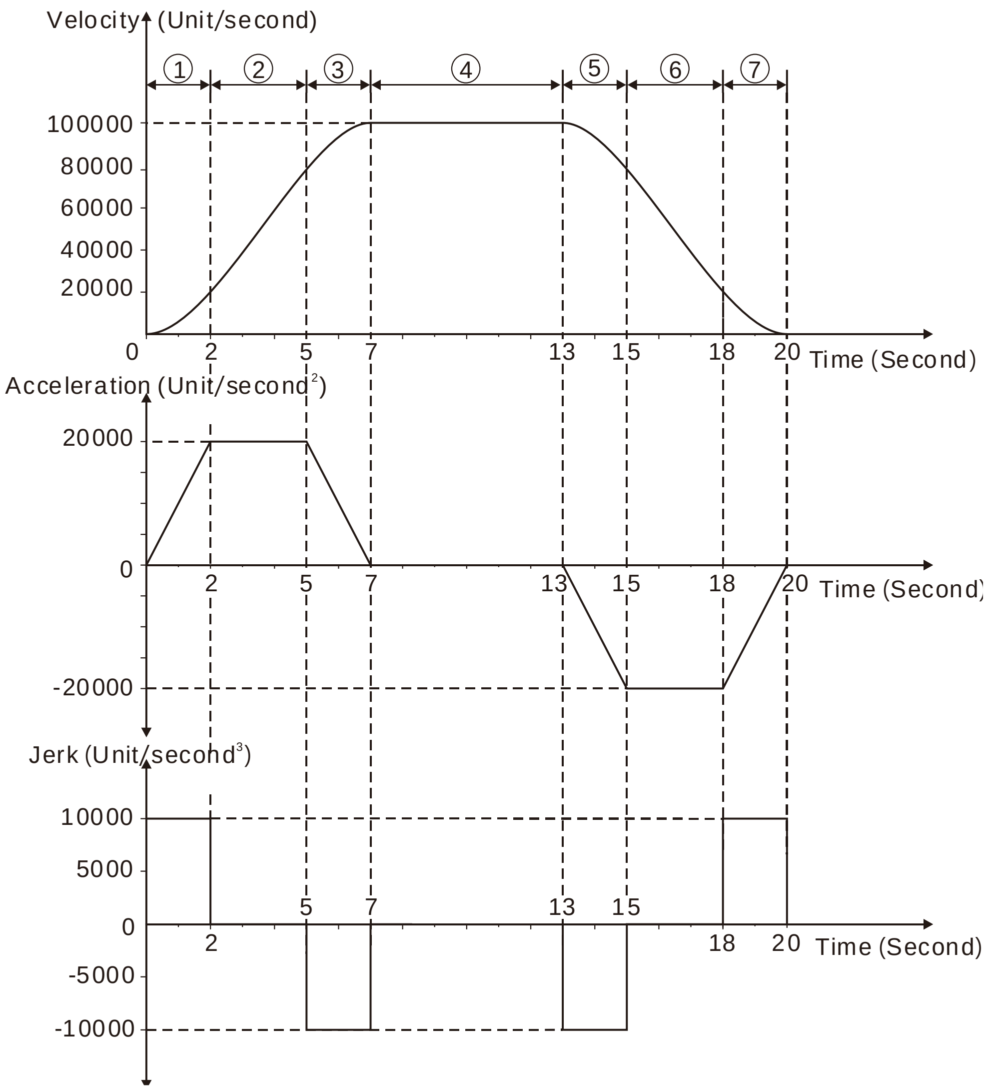
Velocity: Indicates the maximum velocity in the motion of an axis with the unit of unit/second.
Acceleration: Indicates the maximum acceleration in the motion of an axis with the unit of unit/second^2.
Jerk: Indicates the maximum change rate of the acceleration or deceleration in the motion of an axis with the unit of unit/second^3. The value of *Jerk* can be specified in the instruction and the value will be used for the axis in the acceleration and deceleration. The smoothness of the velocity can be improved by modifying the value of *Jerk*.

- **The relation among the velocity, acceleration (deceleration) and jerk:**

$$Acc(Dec) = \frac{dv}{dt}$$

$$Jerk = \frac{dAcc}{dt}$$

The acceleration (deceleration) is the variation of the velocity per unit time. The change rate of acceleration is the variation of the acceleration per unit time. For example, one MC_MoveRelative instruction is be used to express the relation among the three elements. The distance is 1300000 units; the velocity is 100000units/second; the acceleration is 20000units/second^2 and the jerk is 10000units/second^3.
See the following chart for the relation among these elements.



- **The relations among Velocity, Acceleration and Jerk are explained in the following table.**

Stage No.	Time (second)	Jerk (Unit/second^3)	Acceleration/ Deceleration (Unit/second^2)	Velocity (Unit/second)	Motion type
1	0~2	10000 units/second^3	Acceleration is increased to 20000 units/second^2	Increasing	The acceleration motion with an increasing acceleration
2	2~5	0	Acceleration stays at 20000 units/second^2	Increasing	The acceleration motion with a constant acceleration
3	5~7	-10000 units/second^3	Acceleration is decreased to 0.	Increases to 100000 unit/second	The acceleration motion with an decreasing acceleration
4	7~13	0	Acceleration has been decreased to 0unit/second^2 and it has been 0unit/second^2	100000 unit/second	The motion at a constant speed

Stage No.	Time (second)	Jerk (Unit/second^3)	Acceleration/ Deceleration (Unit/second^2)	Velocity (Unit/second)	Motion type
			during this stage.		
5	13~15	-10000 units/second^3	Deceleration is increased to 20000unit/second^2.	Decreasing	The deceleration motion with an increasing deceleration
6	15~18	0	Deceleration has been increased to 20000units/second^2 and it has been 20000units/second^2 during this stage.	Decreasing	The deceleration motion with a constant deceleration
7	18~20	10000 units/second^3	Deceleration is decreased to 0.	Decreases to 0	The deceleration motion with a decreasing deceleration

10.3 Introduction of BufferMode

For the same axis, another motion instruction can be started while one motion instruction is controlling the axis motion. There are 6 buffer modes for selection to switch from one motion instruction being executed to another motion instruction. The buffer mode can be selected through the *BufferMode* parameter of the buffered motion instruction.

The terms about *BufferMode* are explained as below.

1. Current instruction: The motion instruction which is controlling the axis currently.
2. Buffered instruction: The instruction which is waiting to be executed.
3. Transit velocity: The speed at which the axis moves at the moment when the currently being executed instruction is switched to the buffered instruction.
4. Target velocity: The *Velocity* parameter of an instruction
5. Target position: The *Position* or *Distance* parameter of the position-related instructions

- **Six Buffer Modes for Selection**

Buffer Mode	Description
0: mcAborting **(Aborting)**	The instruction being executed currently is aborted immediately.
1: mcBuffered **(Buffered)**	The buffered instruction just starts to control the axis after the current instruction execution is completed.
2: mcBlendingLow **(Blend with low)**	The buffered instruction just starts to control the axis after the target position of the current instruction is reached. The transit velocity is the lower of the target velocities of the current instruction and buffered instruction.
3: mcBlendingPrevious **(Blend with previous)**	The buffered instruction just starts to control the axis after the target position of the current instruction is reached. The transit velocity is the target velocity of the current instruction.
4: mcBlendingNext **(Blend with next)**	The buffered instruction just starts to control the axis after the target position of the current instruction is reached. The transit velocity is the target velocity of the buffered instruction.
5: mcBlendingHigh **(Blend with high)**	The buffered instruction just starts to control the axis after the target position of the current instruction is reached. The transit velocity is the higher of the target velocities of the current instruction and buffered instruction.

Notes:

1. The same axis only supports one buffer mode. An error will occur if multiple buffer modes are performed for the same axis.
   For example, the *BufferMode* parameters of instruction 2 and instruction 3 are not mcAborting. Instruction 2 (the buffered instruction) will be switched to from instruction 1 (current instruction). Instruction 3 will report an error if instruction 3 is switched to from instruction 2 when the execution of instruction 1 is not completed. If the *BufferMode* parameter of Instruction 3 is mcAborting, instruction 1 and instruction 2 will be aborted immediately and instruction 3 will be executed right away.
2. When the MC_MoveSuperimposed instruction controls the axis alone, the buffered instruction excluding MC_MoveAdditive is executed and the MC_MoveSuperimposed instruction is aborted no matter what the value of the *BufferMode* parameter is.
   While the current instruction and MC_MoveSuperimposed or MC_HaltSuperimposed jointly control the axis and then another motion instruction is executed, all the being executed previously instructions will be aborted if *BufferMode*=mcAborting; if *BufferMode*=mcBuffered, mcBlendingLow, mcBlendingPrevious, mcBlendingNext and mcBlendingHigh, the current instruction and buffered

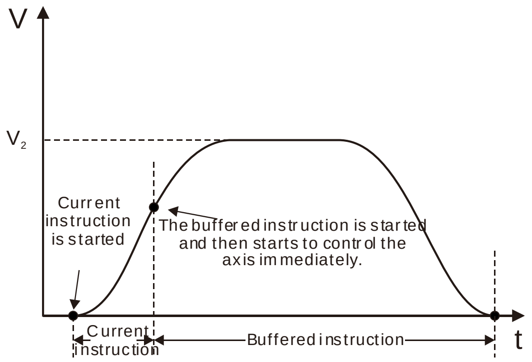
instruction will be blended according to the setting value of *BufferMode* without any impact on the execution of MC_MoveSuperimposed or MC_HaltSuperimposed.

- **Example: Using two MC_MoveRelative instructions for explanation.**

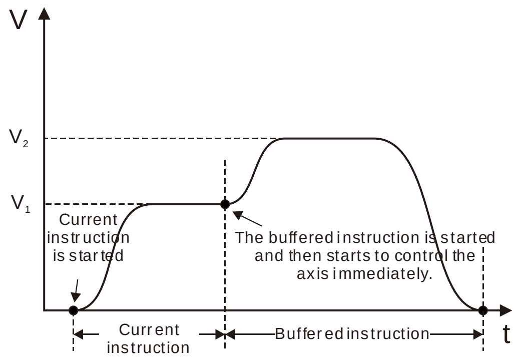
The maximum velocity of the first MC_MoveRelative instruction is V_1 and distance is S_1. The maximum velocity of the second MC_MoveRelative instruction is V_2 and distance is S_2. Modifying the value of *BufferMode* of the second MC_MoveRelative instruction, you can get different blending processes of the two instructions. See details as below.

- **Aborting: Buffermode=mcAborting. See the examples of four cases as below.**

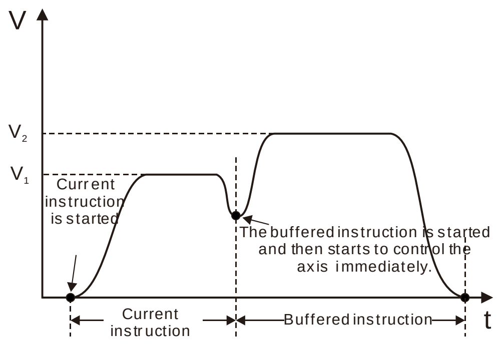
1. Current instruction is aborted while the controlled axis is accelerating.



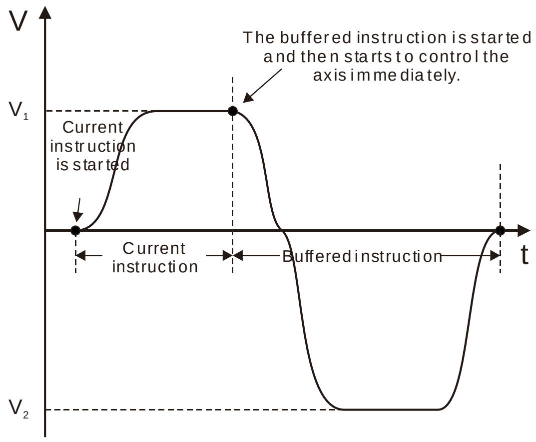
2. Current instruction is aborted while the controlled axis is moving at a constant velocity.



3. Current instruction is aborted while the controlled axis is decelerating.

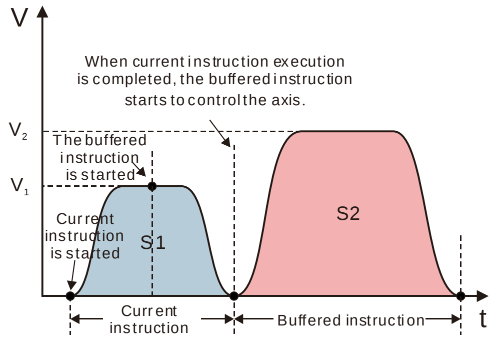


4. The velocity direction of the buffered instruction is opposite to the current instruction.

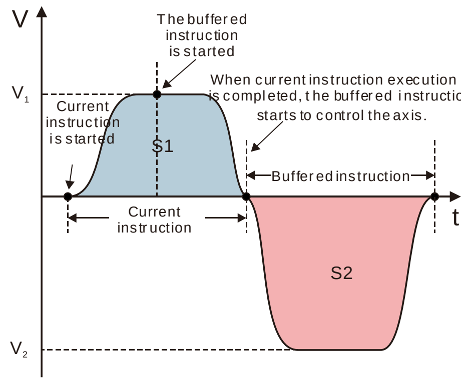


■ **Buffered: Buffermode=**mcBuffered**. See two cases as below.**

1. When the direction of the buffered instruction is the same as that of the current instruction

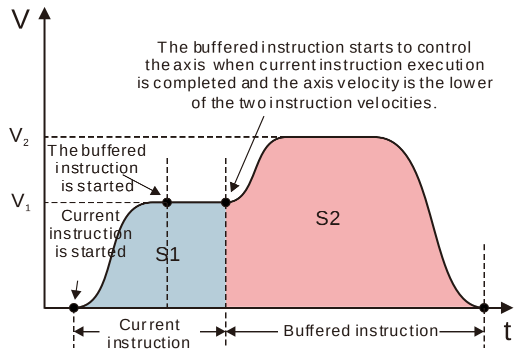


2. When the direction of the buffered instruction is opposite to that of the current instruction.

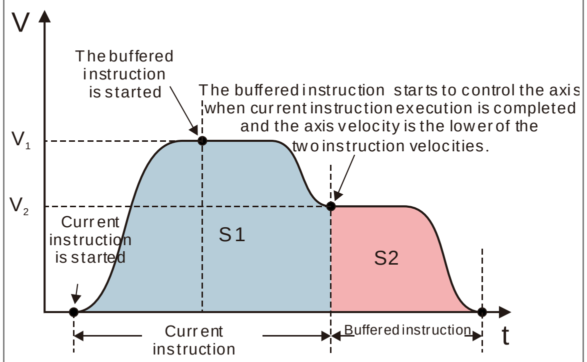


■ **Blending with low velocity: Buffermode=**mcBlendingLow**. See three cases as below.**

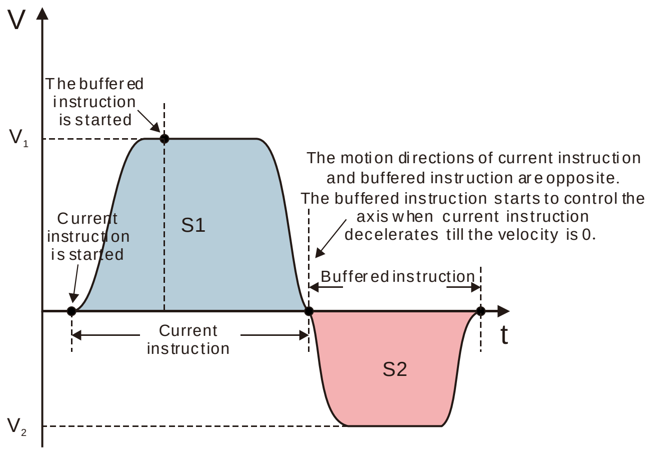
1. When the velocity of the current instruction is less than that of the buffered instruction.



2. When the velocity of the current instruction is greater than that of the buffered instruction.

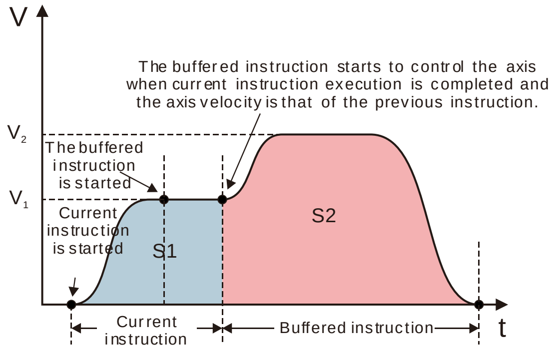


3. When the velocity direction of the current instruction is opposite to that of the buffered instruction.

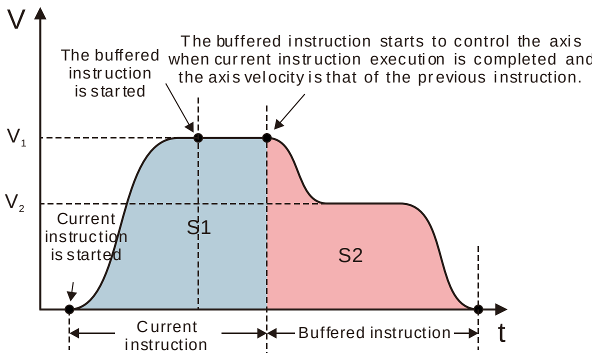


■ **Blending with previous velocity: Buffermode=**mcBlendingPrevious**. See three cases as below.**

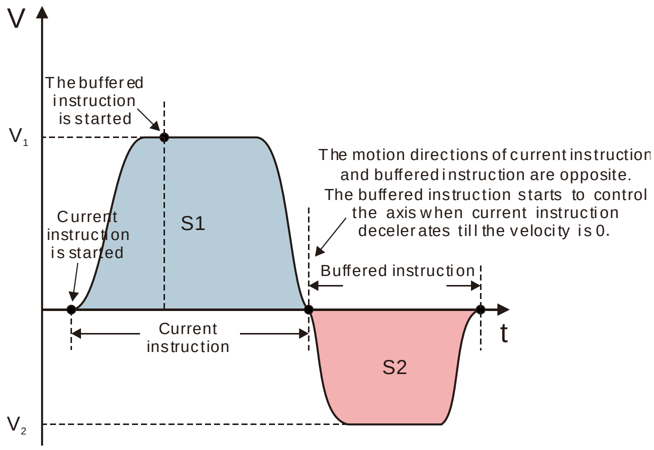
1. When the target velocity of the current instruction is less than that of the buffered instruction.



2. When the target velocity of the current instruction is greater than that of the buffered instruction.

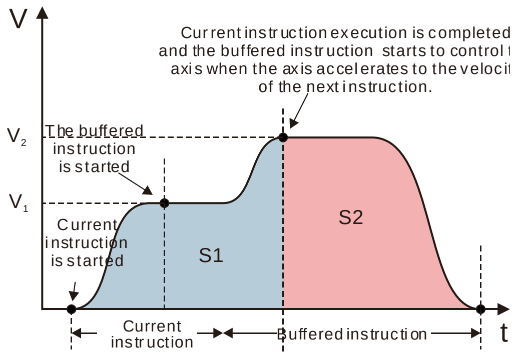


3. When the velocity direction of the current instruction is opposite to that of the buffered instruction.

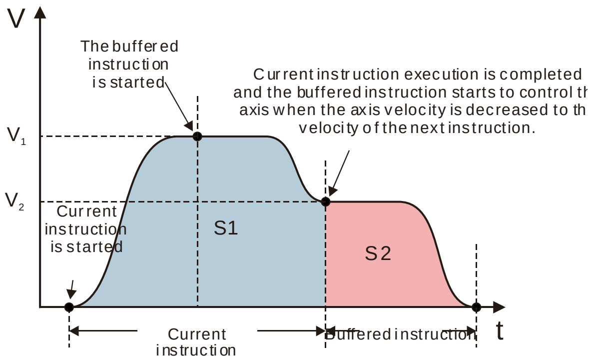


■ **Blending with next velocity: Buffermode=**mcBlendingNext**. See three cases as below.**

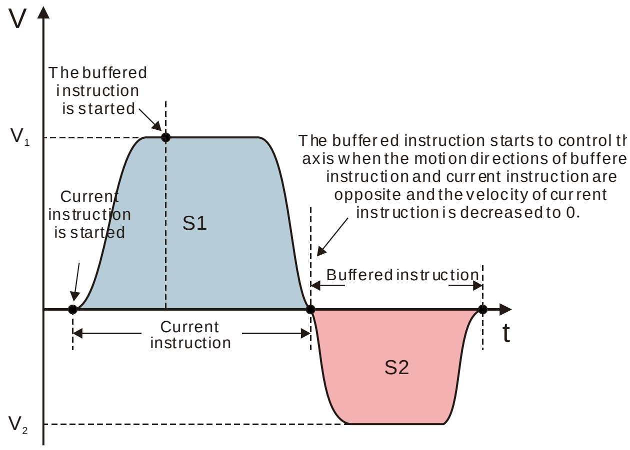
1. When the target velocity of the current instruction is less than that of the buffered instruction



2. When the target velocity of the current instruction is greater than that of the buffered instruction

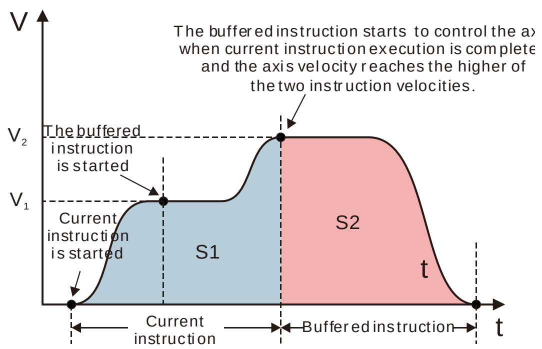


3. When the velocity direction of the current instruction is opposite to that of the buffered instruction

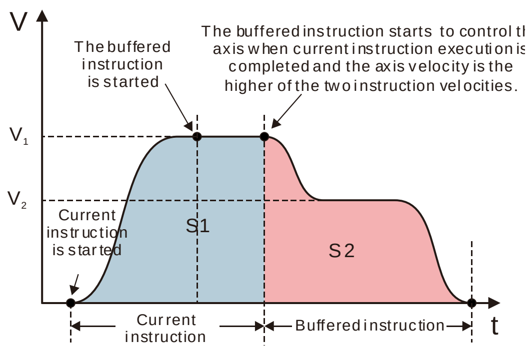


■ **Blending with high velocity: Buffermode=**mcBlendingHigh**. See three cases as below.**

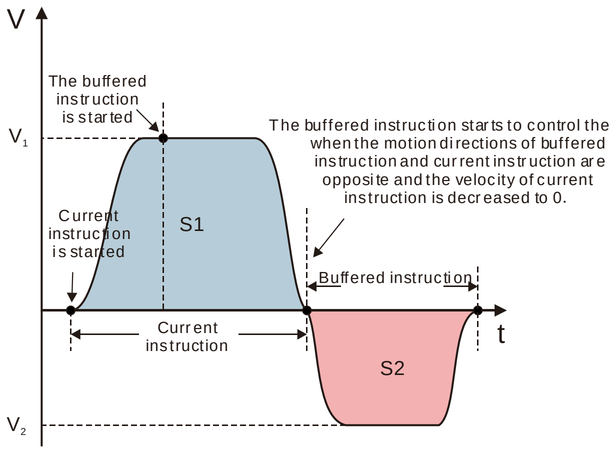
1. When the target velocity of the current instruction is less than that of the buffered instruction



2. When the target velocity of the current instruction is greater than that of the buffered instruction



3. When the velocity direction of the current instruction is opposite to that of the buffered instruction



10

- **Buffer Modes that various instructions support**

The buffer mode of the current instruction and buffered instruction is set by modifying the value of the *BufferMode* parameter. The value of BufferMode of the buffered instruction is selected according to the buffer mode that current instruction supports and the B*ufferMode* parameter of the current instruction is invalid.

For example:

The *BufferMode* of MC_MoveRelative supports mcAborting, mcBuffered, mcBlendingLow, mcBlendingPrevious, mcBlendingNext and mcBlendingHigh. The *BufferMode* of MC_MoveVelocity supports mcAborting and mcBuffered.

Case 1： If MC_MoveRelative is the current instruction and MC_MoveVelocity is the buffered instruction. The *BufferMode* parameter of MC_MoveVelocity can select one of mcAborting, mcBuffered, mcBlendingLow, mcBlendingPrevious, mcBlendingNext and mcBlendingHigh.

Case 2： If MC_MoveVelocity is the current instruction and MC_MoveRelative is the buffered instruction. The *BufferMode* parameter of MC_MoveRelative can select one of mcAborting and mcBuffered.

The buffer mode of the buffered instruction can be selected according to the current instruction as listed below.

Current instruction	The selectable *BufferMode* value of the buffered instruction
MC_MoveAbsolute	【mcAborting, mcBuffered, mcBlendingLow, mcBlendingPrevious, mcBlendingNext, mcBlendingHigh】*[1]
MC_MoveRelative	【mcAborting, mcBuffered, mcBlendingLow, mcBlendingPrevious, mcBlendingNext, mcBlendingHigh】*[1]
MC_MoveAdditive	【mcAborting, mcBuffered, mcBlendingLow, mcBlendingPrevious, mcBlendingNext, mcBlendingHigh】*[1]
MC_MoveSuperimposed	mcAborting
MC_HaltSuperimposed	mcAborting
MC_MoveVelocity	mcAborting, mcBuffered
MC_Home	Only the MC_Stop instruction can abort the MC_Home instruction.
MC_Halt	mcAborting, mcBuffered
MC_ GearIn	mcAborting, mcBuffered
MC_ GearOut	mcAborting, mcBuffered
MC_CombineAxes	mcAborting, mcBuffered
MC_ CamIn	mcAborting, mcBuffered
MC_ CamOut	mcAborting, mcBuffered

*[1]: The *BufferMode* parameter of the buffered instructions MC_GearIn, MC_CamIn and MC_CombineAxes can only choose mcAborting and mcBuffered.

Whether the current instruction execution has been completed or not depends on the completion output parameter of the instruction. As the completion output parameter is TRUE, it indicates that the instruction execution is completed and the buffered instruction execution starts.

See the completion output parameters of instructions in the following table so as to judge the instruction execution state in a buffer mode.

Instruction name	Is it a buffered instruction? (Yes or No)	Can it be followed by a buffered instruction? (Yes or No)	Completion output parameter of an instruction
MC_Home	No	No	Done
MC_Stop	No	No	Done
MC_Halt	Yes	Yes	Done
MC_MoveAbsolute	Yes	Yes	Done
MC_MoveRelative	Yes	Yes	Done
MC_MoveAdditive	Yes	Yes	Done
MC_MoveSuperimposed	No	No	——
MC_HaltSuperimposed	No	No	——
MC_MoveVelocity	Yes	Yes	InVelocity
MC_CamIn	Yes	Yes	EndOfProfile
MC_CamOut	No	Yes	Done
MC_GearIn	Yes	Yes	InGear
MC_GearOut	No	Yes	Done
MC_CombineAxes	Yes	Yes	InSync

- **Examples of Buffer Modes**

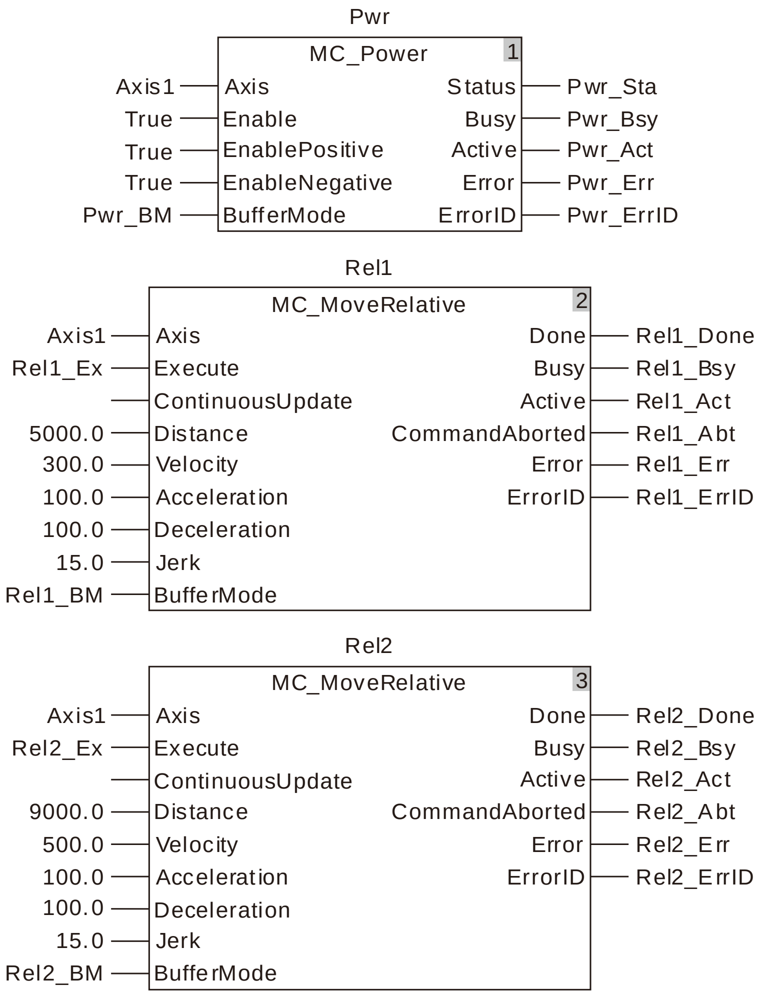
Example 1

The following example explains six buffer modes for the switch from the execution of one MC_MoveRelative instruction to the other MC_MoveRelative instruction.

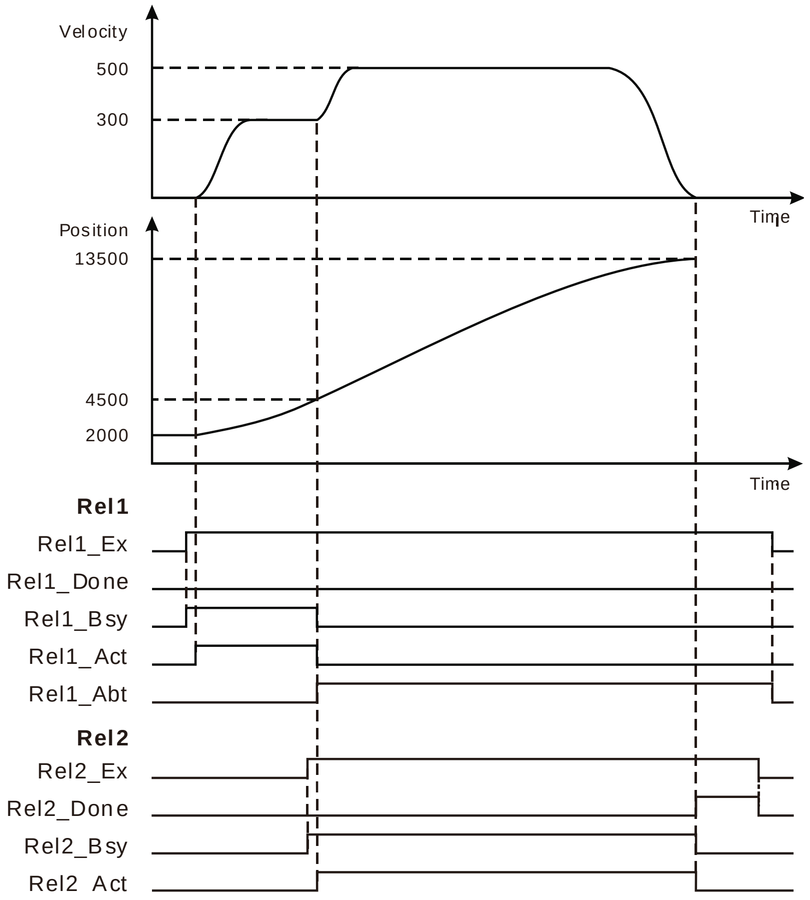
The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel1	MC_MoveRelative	
Rel1_Ex	BOOL	FALSE
Rel1_BM	MC_Buffer_Mode	0
Rel1_Done	BOOL	
Rel1_Bsy	BOOL	
Rel1_Act	BOOL	
Rel1_Abt	BOOL	
Rel1_Err	BOOL	
Rel1_ErrID	WORD	
Rel2	MC_MoveRelative	

Variable name	Data type	Initial value
Rel2_Ex	BOOL	FALSE
Rel2_BM	MC_Buffer_Mode	
Rel2_Done	BOOL	
Rel2_Bsy	BOOL	
Rel2_Act	BOOL	
Rel2_Abt	BOOL	
Rel2_Err	BOOL	
Rel2_ErrID	WORD	



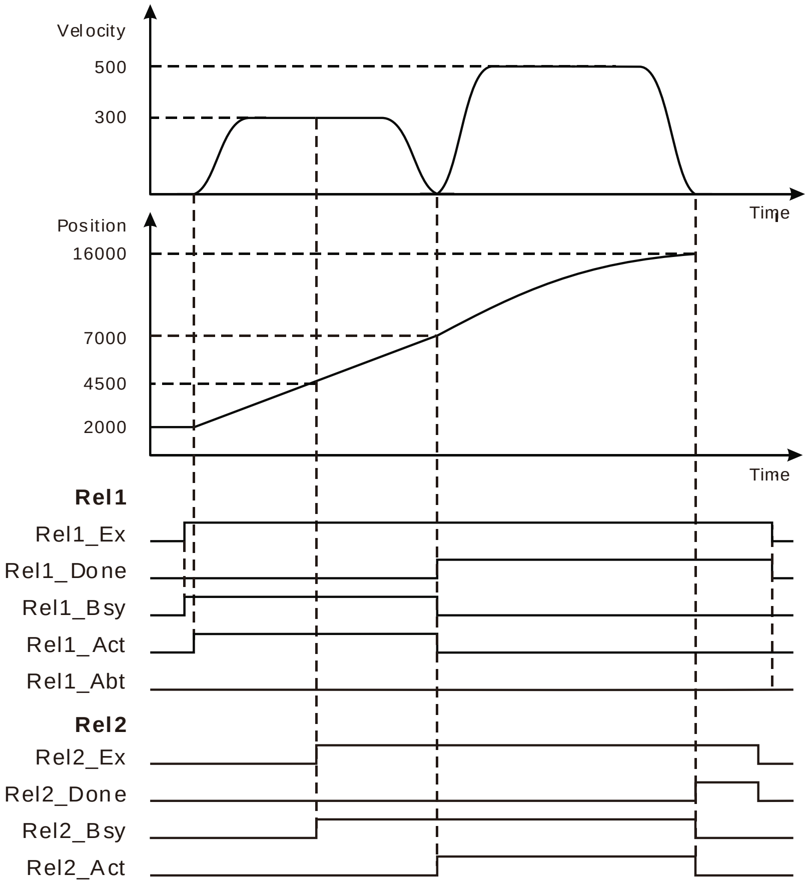
■ **Rel2_BM=mcAborting**



❖ As Rel1_Ex changes from FALSE to TRUE, Rel1_Bsy changes to TRUE. One period later, Rel1_Act changes to TRUE and the first MC_MoveRelative instruction execution starts. While the target position is not reached yet, Rel2_Ex changes from FALSE to TRUE and Rel2_Bsy changes to TRUE. One period later, Rel1_Abt and Rel2_Act change to TRUE and Rel1_Bsy and Rel1_Act change to FALSE. Meanwhile the first MC_MoveRelative instruction is aborted and the second MC_MoveRelative instruction execution starts. As the target position is reached, Rel2_Done changes to TRUE and meanwhile Rel2_Bsy and Rel2_Act change to FALSE.

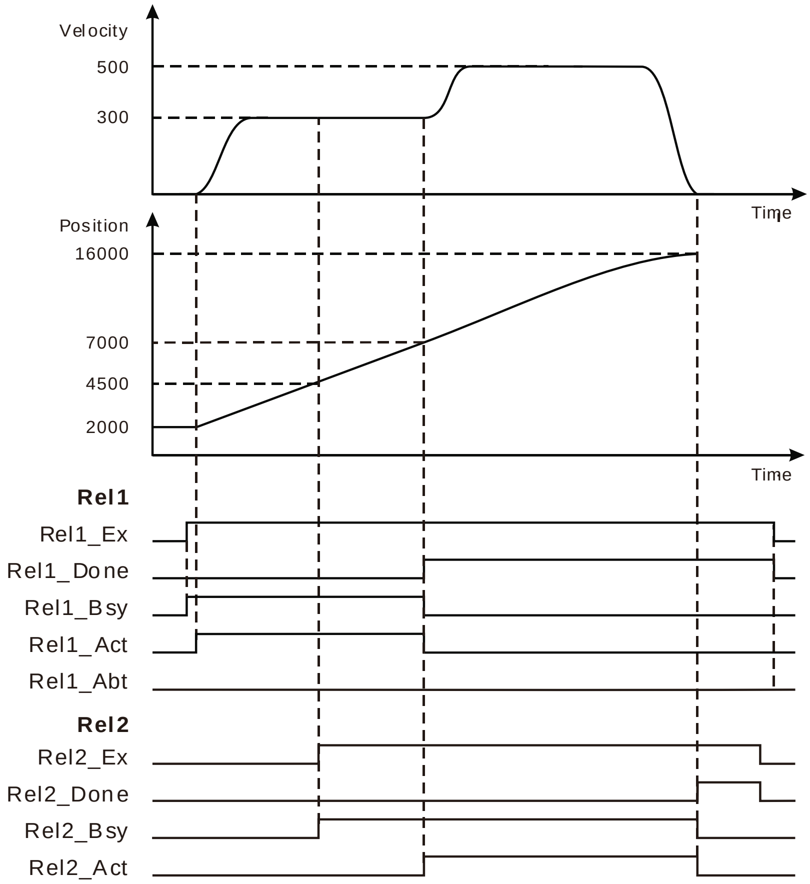
❖ As Rel2_Ex changes from TRUE to FALSE, Rel2_Done changes to FALSE.

■ Rel2_BM =mcMcBuffered



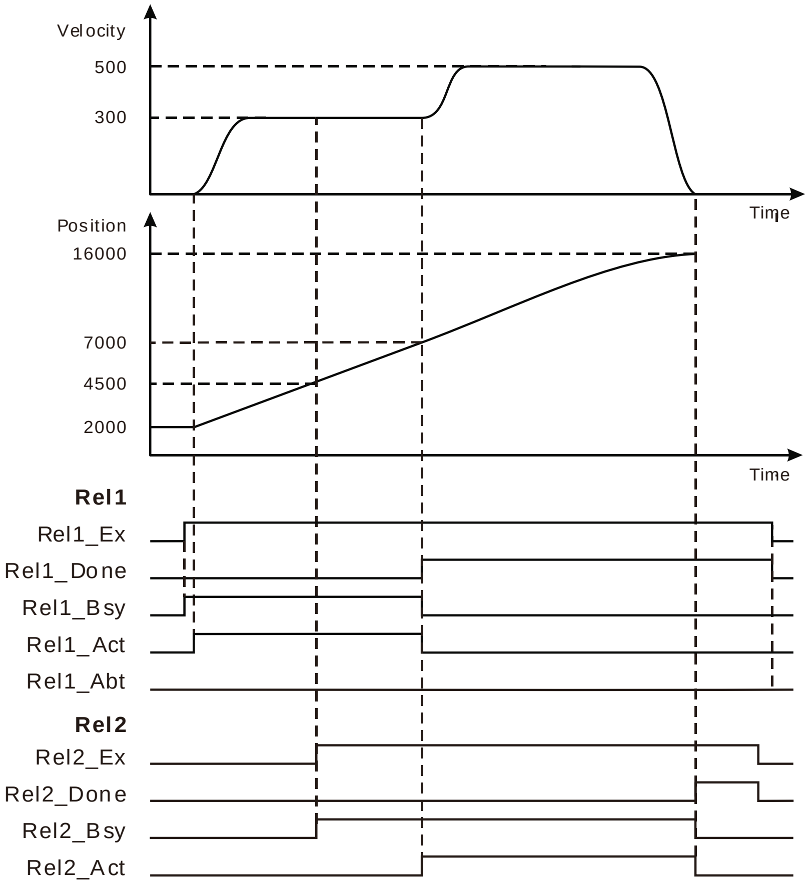
- ❖ As Rel1_Ex changes from FALSE to TRUE, Rel1_Bsy changes to TRUE. One period later, Rel1_Act changes to TRUE and the first MC_MoveRelative instruction execution starts. While the target position is not reached yet and Rel2_Ex changes from FALSE to TRUE, Rel2_Bsy changes to TRUE, Rel1_Bsy and Rel1_Act remain TRUE and the first MC_MoveRelative instruction execution continues. As the target position is reached, Rel1_Done changes to TRUE, Rel1_Bsy and Rel1_Act change to FALSE. Rel2_Act changes to TRUE and the second MC_MoveRelative instruction execution starts immediately. When the target position is reached, Rel2_Done changes to TRUE and meanwhile Rel2_Bsy and Rel2_Act change to FALSE.
- ❖ As Rel1_Ex changes from TRUE to FALSE, Rel1_Done changes to FALSE. As Rel2_Ex changes from TRUE to FALSE, Rel2_Done changes to FALSE.

■ **Rel2_BM =mcBlendingLow**



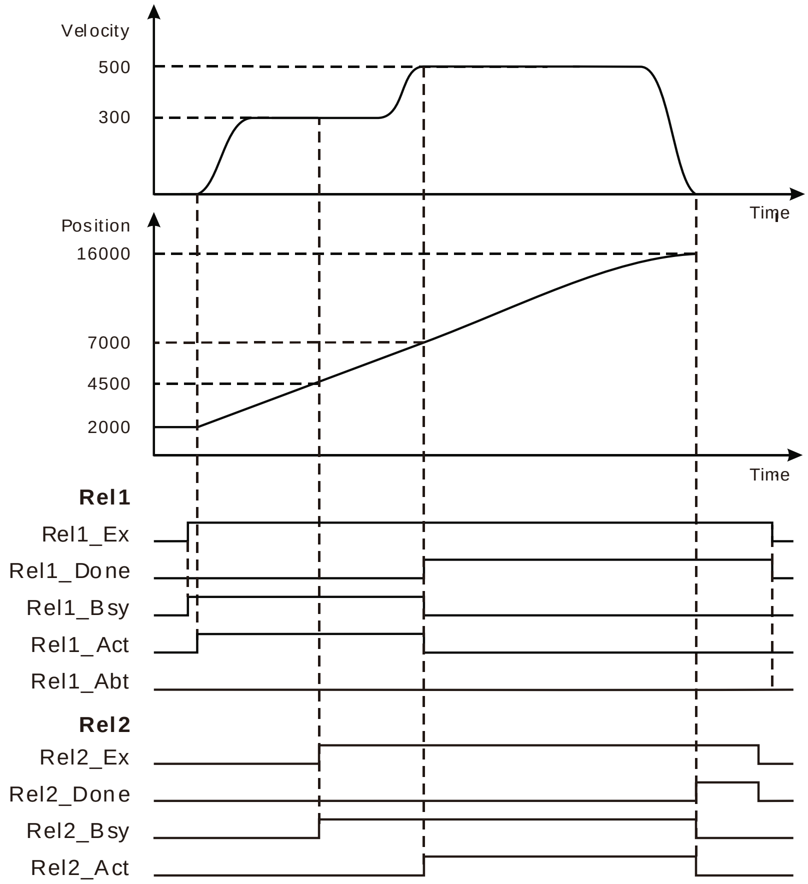
- ❖ As Rel1_Ex changes from FALSE to TRUE, Rel1_Bsy changes to TRUE. One period later, Rel1_Act changes to TRUE and the first MC_MoveRelative instruction execution starts. While the target position is not reached yet and Rel2_Ex changes from FALSE to TRUE, Rel2_Bsy changes to TRUE, Rel1_Bsy and Rel1_Act remain TRUE and the first MC_MoveRelative instruction execution continues. As the target position is reached, Rel1_Done changes to TRUE. At the moment, the velocity is 300 units /second which is the lower one of the target velocities of the current instruction and buffered instruction, Rel1_Bsy and Rel1_Act change to FALSE, Rel2_Act changes to TRUE and the second MC_MoveRelative instruction execution starts immediately. As the target position is reached, Rel2_Done changes to TRUE and meanwhile Rel2_Bsy and Rel2_Act change to FALSE.
- ❖ As Rel1_Ex changes from TRUE to FALSE, Rel1_Done changes to FALSE. As Rel2_Ex changes from TRUE to FALSE, Rel2_Done changes to FALSE.

■ **Rel2_BM =mcBlending _Previous**



- ❖ As Rel1_Ex changes from FALSE to TRUE, Rel1_Bsy changes to TRUE. One period later, Rel1_Act changes to TRUE and the first MC_MoveRelative instruction execution starts. While the target position is not reached yet and Rel2_Ex changes from FALSE to TRUE, Rel2_Bsy changes to TRUE, Rel1_Bsy and Rel1_Act remain TRUE and the first MC_MoveRelative instruction execution continues. As the target position is reached, Rel1_Done changes to TRUE. At the moment, the velocity is 300 units /second which is the target velocity of the current instruction, Rel1_Bsy and Rel1_Act change to FALSE, Rel2_Act changes to TRUE and the second MC_MoveRelative instruction execution starts immediately. As the target position is reached, Rel2_Done changes to TRUE and meanwhile Rel2_Bsy and Rel2_Act change to FALSE.
- ❖ As Rel1_Ex changes from TRUE to FALSE, Rel1_Done changes to FALSE. As Rel2_Ex changes from TRUE to FALSE, Rel2_Done changes to FALSE.

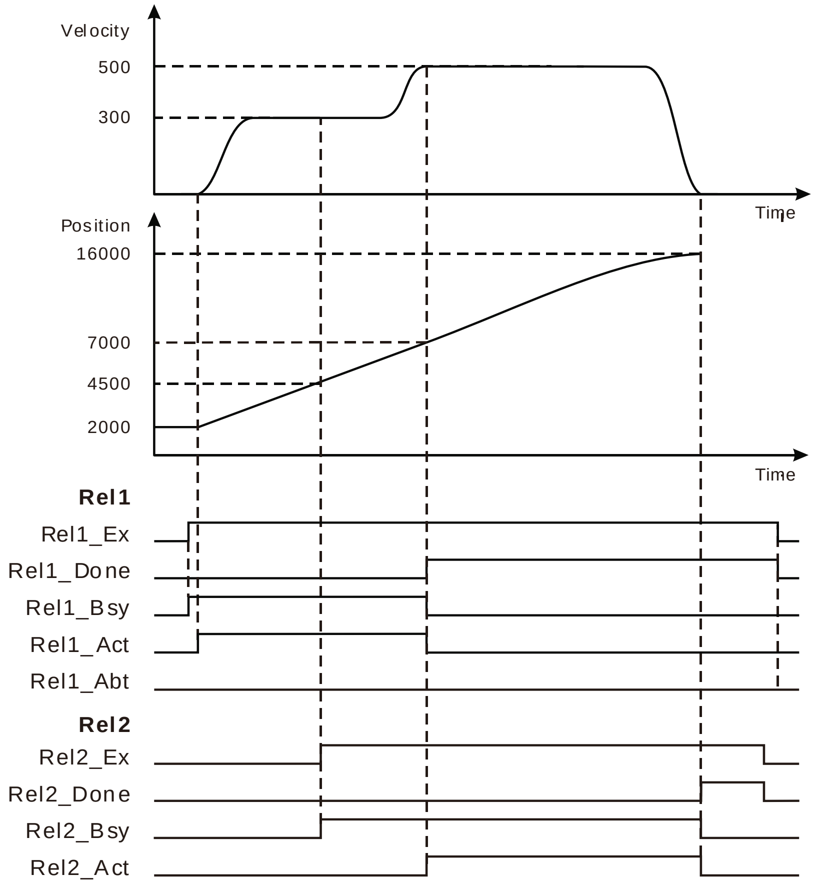
- **Rel2_BM =mcBlending _Next**



❖ As Rel1_Ex changes from FALSE to TRUE, Rel1_Bsy changes to TRUE. One period later, Rel1_Act changes to TRUE and the first MC_MoveRelative instruction execution starts. While the target position is not reached yet and Rel2_Ex changes from FALSE to TRUE, Rel2_Bsy changes to TRUE, Rel1_Bsy and Rel1_Act remain TRUE and the first MC_MoveRelative instruction execution continues. As the target position is reached, Rel1_Done changes to TRUE. At the moment, the velocity is 500 units /second which is the target velocity of the buffered instruction; Rel1_Bsy and Rel1_Act change to FALSE; Rel2_Act changes to TRUE and the second MC_MoveRelative instruction execution starts. As the target position is reached, Rel2_Done changes to TRUE and meanwhile Rel2_Bsy and Rel2_Act change to FALSE.

❖ As Rel1_Ex changes from TRUE to FALSE, Rel1_Done changes to FALSE. As Rel2_Ex changes from TRUE to FALSE, Rel2_Done changes to FALSE.

■ **Rel2_BM =mcBlending _High**



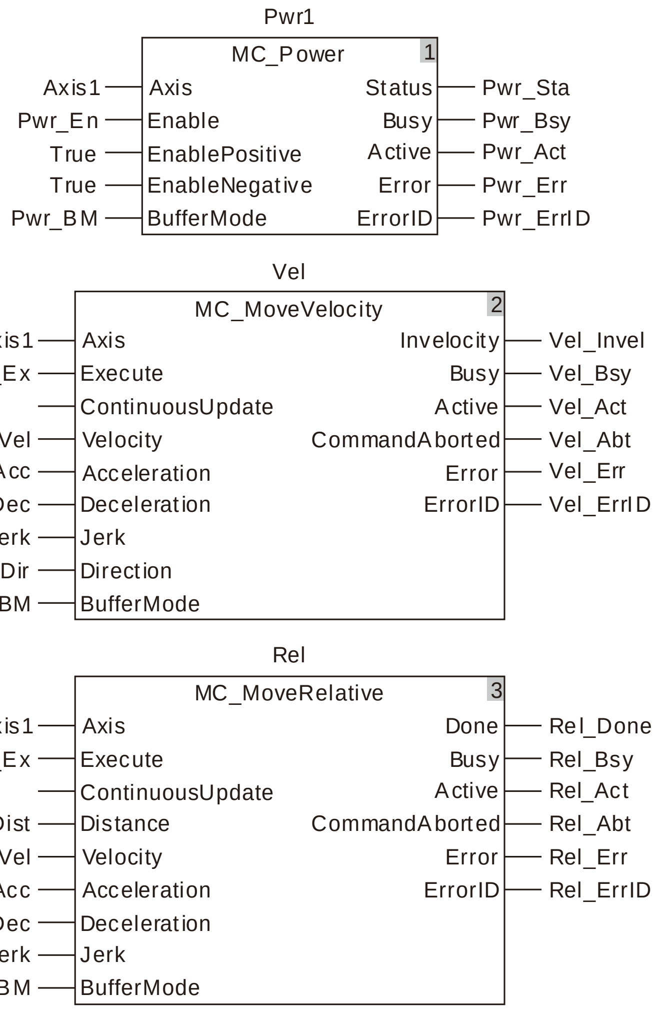
- ❖ As Rel1_Ex changes from FALSE to TRUE, Rel1_Bsy changes to TRUE. One period later, Rel1_Act changes to TRUE and the first MC_MoveRelative instruction execution starts. While the target position is not reached yet and Rel2_Ex changes from FALSE to TRUE, Rel2_Bsy changes to TRUE, Rel1_Bsy and Rel1_Act remain TRUE and the first MC_MoveRelative instruction execution continues. As the target position is reached, Rel1_Done changes to TRUE. At the moment, the velocity is 500 units /second which is the higher one of the target velocities of the current instruction and buffered instruction; Rel1_Bsy and Rel1_Act change to FALSE; Rel2_Act changes to TRUE and the second MC_MoveRelative instruction execution starts. As the target position is reached, Rel2_Done changes to TRUE and meanwhile Rel2_Bsy and Rel2_Act change to FALSE.
- ❖ As Rel1_Ex changes from TRUE to FALSE, Rel1_Done changes to FALSE. As Rel2_Ex changes from TRUE to FALSE, Rel2_Done changes to FALSE.

- **Example 2**

The following example explains the axis states for different *BufferMode* values with a MC_MoveVelocity instruction and a MC_MoveReltave instruction which is the buffered instruction.

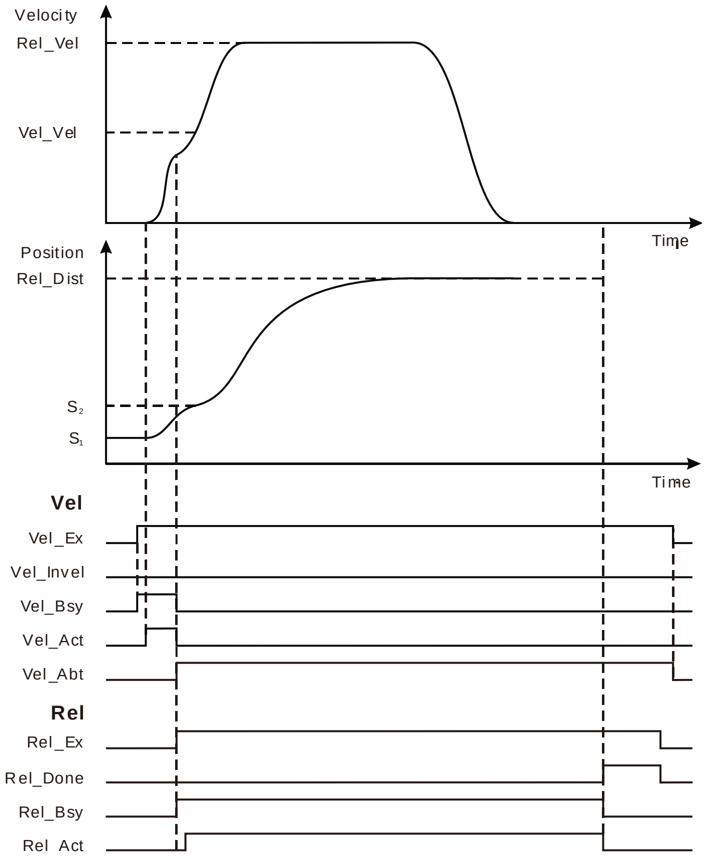
The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel _Ex	BOOL	FALSE
Vel _Vel	LREAL	10000.0
Vel _Acc	LREAL	10000.0
Vel _Dec	LREAL	10000.0
Vel _Jerk	LREAL	10000.0
Vel _Dir	MC_DIRECTION	1
Vel _BM	MC_Buffer_Mode	
Vel _Invel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ErrID	WORD	
Rel	MC_MoveRelative	
Rel_Ex	BOOL	FALSE
Rel_Dist	LREAL	100000.0
Rel_Vel	LREAL	20000.0
Rel_Acc	LREAL	10000.0
Rel_Dec	LREAL	10000.0
Rel_Jerk	LREAL	10000.0
Rel_BM	MC_Buffer_Mode	0
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	


Pwr1
MC_Power
1
Axis1
Pwr_En
True
True
Pwr_BM
Axis
Enable
EnablePositive
EnableNegative
BufferMode
Status
Busy
Active
Error
ErrorID
Pwr_Sta
Pwr_Bsy
Pwr_Act
Pwr_Err
Pwr_ErrID
Vel
MC_MoveVelocity
2
Axis1
Vel_Ex
Vel_Vel
Vel_Acc
Vel_Dec
Vel_Jerk
Vel_Dir
Vel_BM
Axis
Execute
ContinuousUpdate
Velocity
Acceleration
Deceleration
Jerk
Direction
BufferMode
Invelocity
Busy
Active
CommandAborted
Error
ErrorID
Vel_Invel
Vel_Bsy
Vel_Act
Vel_Abt
Vel_Err
Vel_ErrID
Rel
MC_MoveRelative
3
Axis1
Rel_Ex
Rel_Dist
Rel_Vel
Rel_Acc
Rel_Dec
Rel_Jerk
Rel_BM
Axis
Execute
ContinuousUpdate
Distance
Velocity
Acceleration
Deceleration
Jerk
BufferMode
Done
Busy
Active
CommandAborted
Error
ErrorID
Rel_Done
Rel_Bsy
Rel_Act
Rel_Abt
Rel_Err
Rel_ErrID

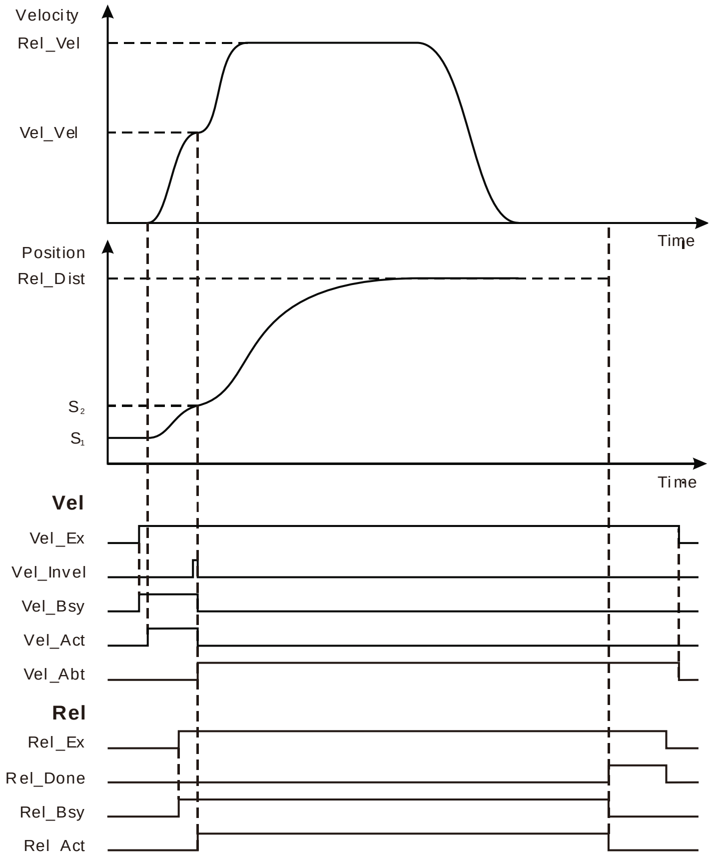
10

■ **Rel_BM =mcAborting**



❖ As Vel_Ex changes from FALSE to TRUE, Vel_Bsy changes to TRUE. One period later, Vel_Act changes to TRUE. Before the target velocity is reached, the axis moves at the velocity and acceleration specified by the MC_MoveRelative instruction as Rel_Ex changes from FALSE to TRUE. As Vel_Abt changes to TRUE, Vel_Bsy and Vel_Act change to FALSE, the velocity instruction is aborted, the MC_MoveRelative instruction is executed and Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. As the positioning is completed, Rel_Done changes to TRUE.

■ **Rel_BM =mcBuffered**



❖ As Vel_Ex changes from FALSE to TRUE, Vel_Bsy changes to TRUE. One period later, Vel_Act changes to TRUE. Rel_Ex changes from FASLE to TRUE when the target velocity is not reached. The axis will not execute the MC_MoveRelatvie instruction till the velocity instruction execution is completed. At the moment, Rel_Bsy changes to TRUE. When the velocity instruction execution is completed, Vel_Invel changes to TRUE and one period later, the MC_MoveRelatvie instruction starts to control the axis. Vel_Abt changes to TRUE and the velocity instruction is aborted. Rel_Act is TRUE, which means that the MC_MoveRelative instruction starts to control the motion of the axis. Rel_Done changes to TRUE as the positioning is completed.

■ **(The effect of Rel_BM = mcBlendingLow, mcBlendingPrevious, mcBlendingNext or mcBlendingHigh is the same as that of Rel_BM = mcBuffered.)**

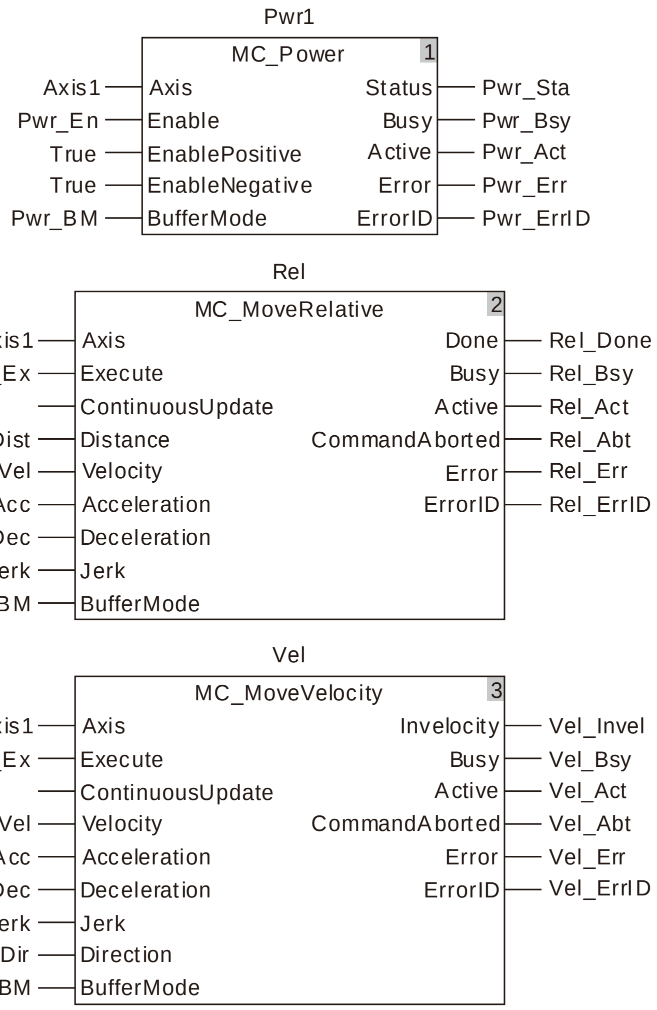
● **Example 3**

The example explains the axis states for different *BufferMode* value with a MC_MoveRelative instruction and a MC_MoveVelocity instruction which is the buffered instruction.

The variable table and program

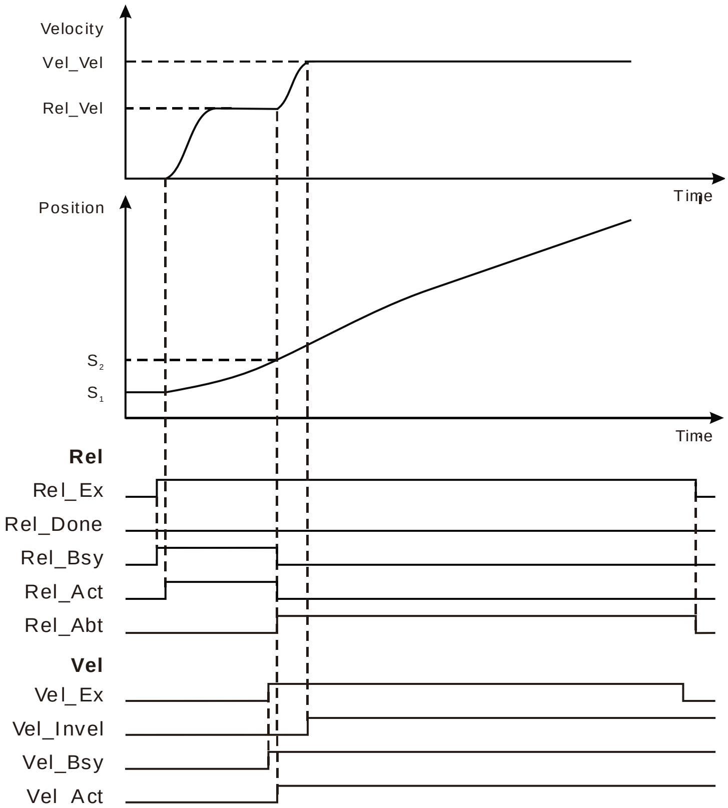
Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	

Variable name	Data type	Initial value
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel	MC_MoveRelative	
Rel_Ex	BOOL	FALSE
Rel_Dist	LREAL	100000.0
Rel_Vel	LREAL	10000.0
Rel_Acc	LREAL	10000.0
Rel_Dec	LREAL	10000.0
Rel_Jerk	LREAL	10000.0
Rel_BM	MC_Buffer_Mode	0
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel _Ex	BOOL	FALSE
Vel _Vel	LREAL	20000.0
Vel _Acc	LREAL	10000.0
Vel _Dec	LREAL	10000.0
Vel _Jerk	LREAL	10000.0
Vel _Dir	MC_DIRECTION	1
Vel _BM	MC_Buffer_Mode	
Vel _Invel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ErrID	WORD	


Pwr1
MC_Power
1
Axis1
Axis
Pwr_En
Enable
True
EnablePositive
True
EnableNegative
Pwr_BM
BufferMode
Status
Pwr_Sta
Busy
Pwr_Bsy
Active
Pwr_Act
Error
Pwr_Err
ErrorID
Pwr_ErrID
Rel
MC_MoveRelative
2
Axis1
Axis
Rel_Ex
Execute
ContinuousUpdate
Rel_Dist
Distance
Rel_Vel
Velocity
Rel_Acc
Acceleration
Rel_Dec
Deceleration
Rel_Jerk
Jerk
Rel_BM
BufferMode
Done
Rel_Done
Busy
Rel_Bsy
Active
Rel_Act
CommandAborted
Rel_Abt
Error
Rel_Err
ErrorID
Rel_ErrID
Vel
MC_MoveVelocity
3
Axis1
Axis
Vel_Ex
Execute
ContinuousUpdate
Vel_Vel
Velocity
Vel_Acc
Acceleration
Vel_Dec
Deceleration
Vel_Jerk
Jerk
Vel_Dir
Direction
Vel_BM
BufferMode
Invelocity
Vel_Invel
Busy
Vel_Bsy
Active
Vel_Act
CommandAborted
Vel_Abt
Error
Vel_Err
ErrorID
Vel_ErrID

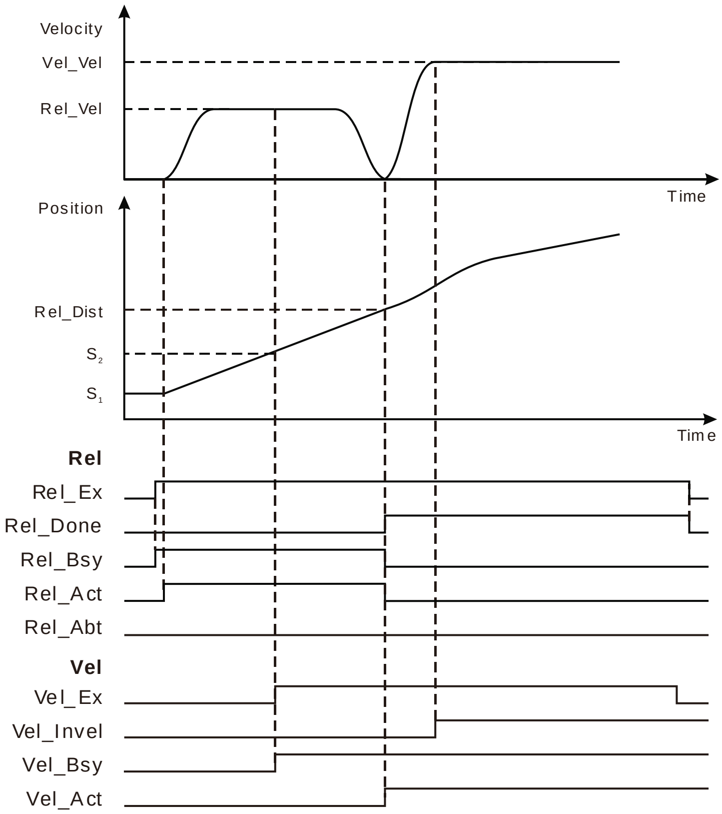
10

■ **Vel _BM =mcAborting**



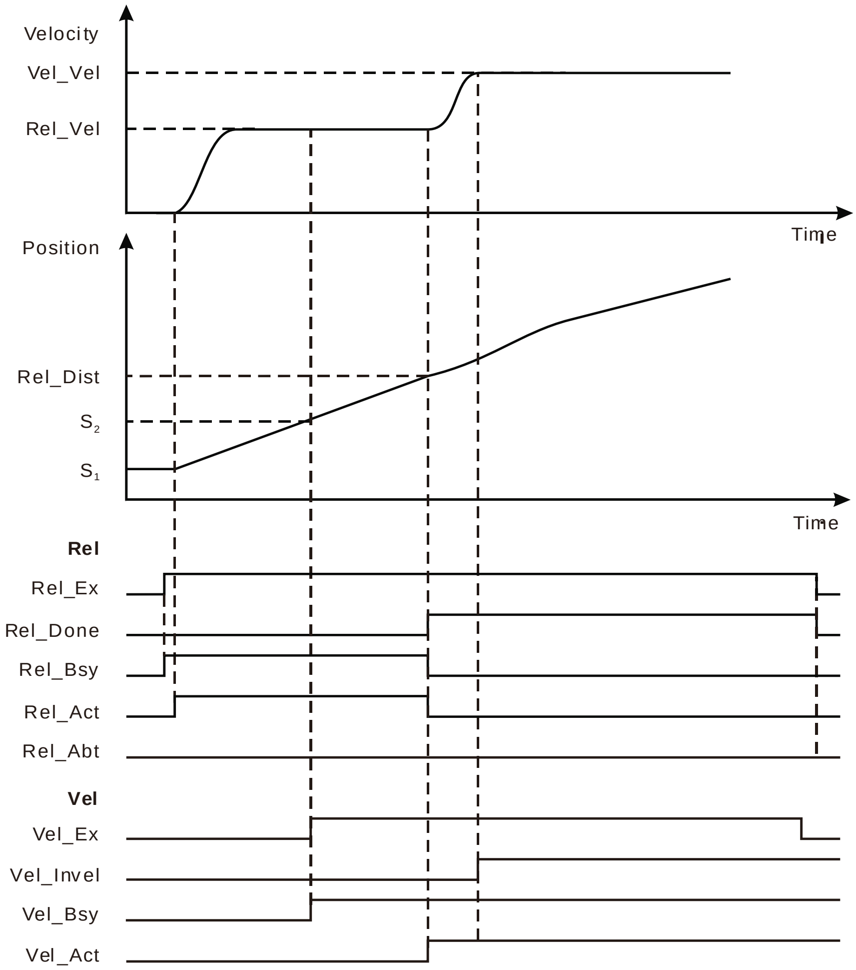
❖ As Rel_Ex changes from FALSE to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. When the target position is not reached, Vel_Ex changes from FALSE to TRUE, the axis moves at the velocity and acceleration specified by the velocity instruction. When Rel_Abt changes to TRUE, Rel_Bsy and Rel_Act change to FALSE, the MC_MoveRelative instruction is aborted and the velocity instruction is executed. Vel_Bsy is TRUE and one period later, Vel_Act changes to TRUE. As the velocity is reached, Vel_Invel changes to TRUE.

■ **Vel _BM =mcBuffered**



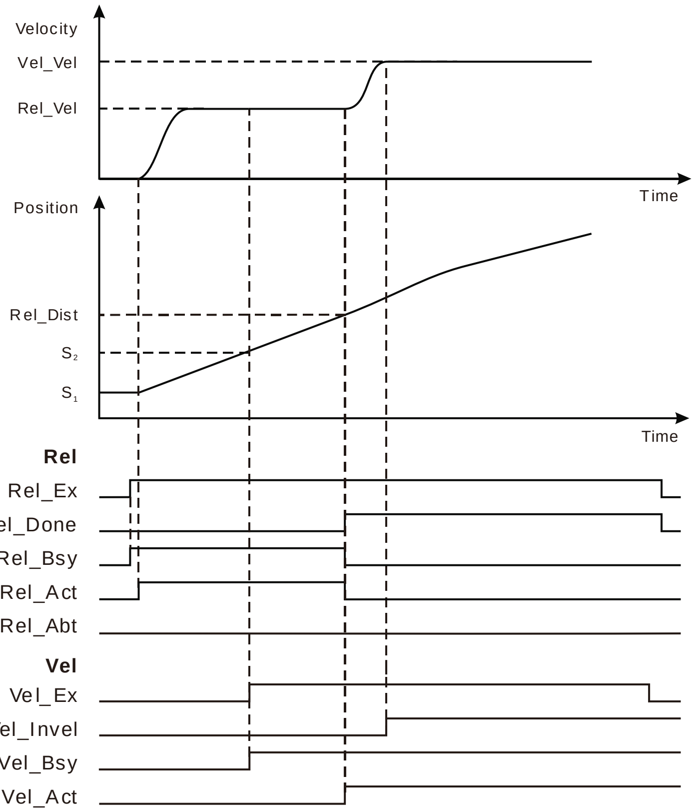
❖ As Rel_Ex changes from FALSE to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. When the target position is not reached, Vel_Ex changes from FALSE to TRUE. The axis decelerates to 0 when the execution of the MC_MoveRelative instruction is completed. Then Rel_Done changes to TRUE, Rel_Bsy and Rel_Act change to FALSE and the axis moves at the velocity and acceleration specified by the velocity instruction. Vel_Bsy changes to TRUE and one period later, Vel_Act changes to TRUE. Rel_Invel changes to TRUE as the target velocity is reached.

■ **Vel _BM =mcBlendingLow**



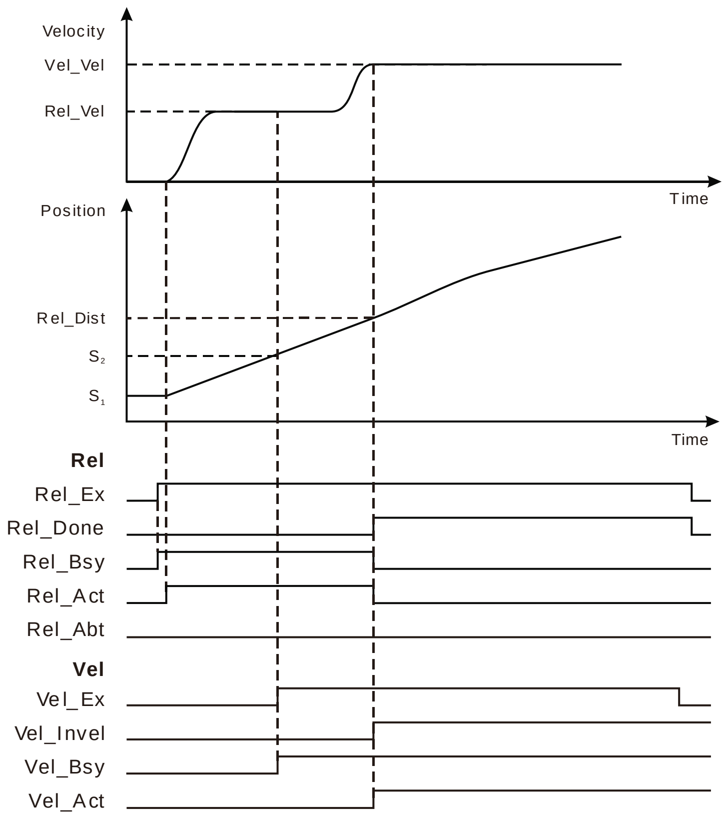
❖ As Rel_Ex changes from FALSE to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. When the target position is not reached, Vel_Ex changes from FALSE to TRUE and Vel_Bsy changes to TRUE. The axis will wait for the completion of MC_MoveRelative execution. After MC_MoveRelative execution is completed, Rel_Done changes to TRUE, Rel_Bsy changes to FALSE and Rel_Act changes to FALSE. Meanwhile Vel_Act changes to TRUE. At the moment, the velocity is 10000units/second, which is the lower one of the target speeds of the current instruction and the buffered instruction. The velocity instruction execution starts after MC_MoveRelative instruction execution is completed. Vel_Invel changes to TRUE when the target velocity is reached.

■ **Vel _BM =mcBlendingPrevious**



❖ As Rel_Ex changes from FALSE to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. When the target position is not reached, Vel_Ex changes from FALSE to TRUE and Vel_Bsy changes to TRUE. The axis will wait for the completion of MC_MoveRelative execution. After MC_MoveRelative execution is completed, Rel_Done changes to TRUE, Rel_Bsy changes to FALSE, Rel_Act changes to FALSE and meanwhile Vel_Act changes to TRUE. At the moment, the velocity is 10000units/second (which is the target speed of the current instruction). Vel_Invel changes to TRUE when the target velocity is reached.

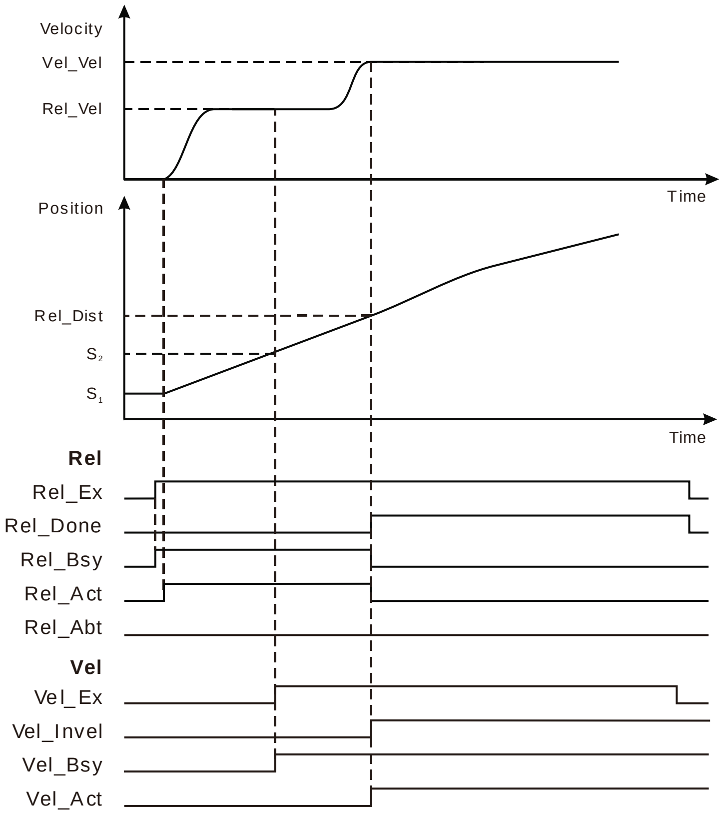
■ **Vel _BM =mcBlendingNext**



❖ As Rel_Ex changes from FALSE to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. When the target position is not reached, Vel_Ex changes from FALSE to TRUE and Vel_Bsy changes to TRUE. The axis will wait for the completion of MC_MoveRelative execution. After MC_MoveRelative execution is completed, Rel_Done changes to TRUE, Rel_Bsy changes to FALSE, Rel_Act changes to FALSE and meanwhile Vel_Act changes to TRUE. At the moment, the velocity is 20000units/second (which is the target speed of the buffered instruction). Vel_Invel changes to TRUE when the target velocity is reached.

10

■ **Vel _BM =mcBlendingHigh**

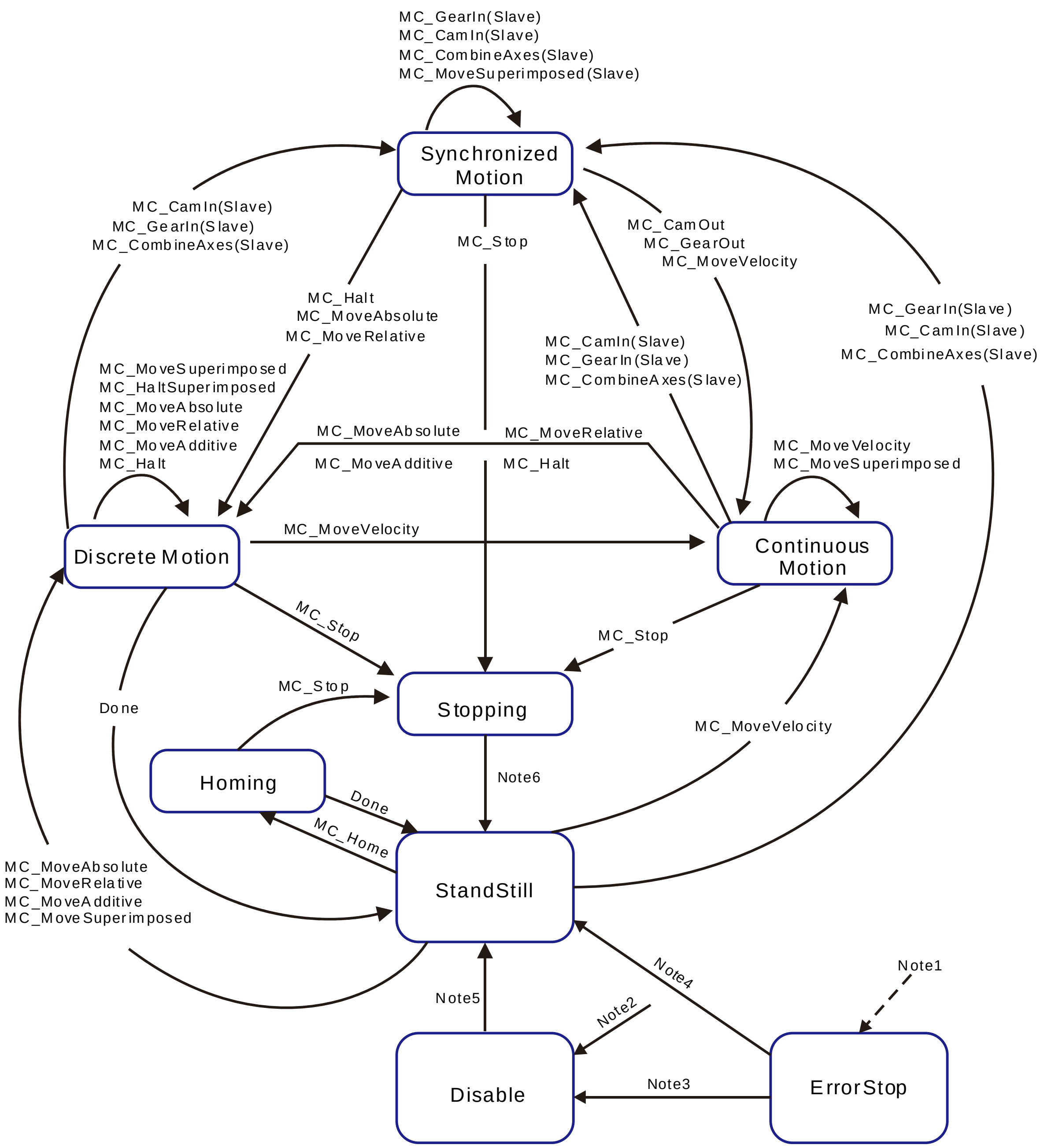


❖ As Rel_Ex changes from FALSE to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE. When the target position is not reached, Vel_Ex changes from FALSE to TRUE and Vel_Bsy changes to TRUE. The axis will wait for the completion of MC_MoveRelative execution. After MC_MoveRelative execution is completed, Rel_Done changes to TRUE, Rel_Bsy changes to FALSE and Rel_Act changes to FALSE. At the moment, the velocity is 20000units/second (which is the higher one of the target speeds of the current instruction and the buffered instruction). And then the axis runs according to the velocity, acceleration and deceleration specified by the velocity instruction. Vel_Invel changes to TRUE when the target velocity is reached.

10.4 The State Machine

When DVP15MC11T utilizes the motion control instruction to control every axis, there is one internal-run state for every axis and axis states are switched by following the state machine instructions below. The state machine defines the motion instructions that can be executed in all states and the states after the motion instructions are executed. Using the motion instructions, users could judge if a certain instruction could be used in current state through the state machine.

The state machine of DVP15MC11T is illustrated as below and the arrow points to the axis status.



Note1 : The axis in any state will enter the ErrorStop state as long as an error occurs in the axis.

Note2 : The axis enters the Disabled state when no axis error occurs in any state and *Enable* of MC_Power is FALSE.

Note3：When *Status* of MC_Power is FALSE, the MC_Reset instruction is used to reset the axis to the Disabled state.

Note4：When *Enable* and *Status* of MC_Power are TRUE, the MC_Reset instruction is used to reset the axis to the Standstill state.

Note5：The axis enters from Disabled to *Standstill* state when the MC_Power instruction is used to enable the axis and *Status* of MC_Power is TRUE.

Note6：The axis enters from Stopping to *Standstill* state when *Done* of MC_Stop is TRUE and *Execute* of MC_Stop is FALSE.

No.	Axis state	Indication
1	StandStill	Pre-execution state
2	Disabled	No-execution state
3	ErrorStop	Error state
4	Stopping	Stop state
5	Homing	Homing state
6	Discrete Motion	Discrete motion state
7	Continuous Motion	Continuous motion state
8	Synchronized Motion	Synchronized motion state

Note:

Axis state can be judged according to the output parameters of the MC_ReadStatus instruction. Refer to section 11.3.17 for details on the MC_ReadStatus instruction.

MEMO

11

Chapter 11 Motion Control Instructions

Table of Contents

11.1 Table of Motion Control Instructions 11-3

11.2 About Motion Control Instructions 11-4
11.2.1 Composition of A Motion Control Instruction 11-4
11.2.2 Program Languages that Motion Control Instructions Support 11-4
11.2.3 Configuration of Motion Control Instructions 11-4

11.3 Single-axis Instructions 11-5
11.3.1 MC_Power 11-5
11.3.2 MC_Home 11-14
11.3.3 MC_MoveVelocity 11-19
11.3.4 MC_Halt 11-26
11.3.5 MC_Stop 11-31
11.3.6 MC_MoveRelative 11-36
11.3.7 MC_MoveAdditive 11-44
11.3.8 MC_MoveAbsolute 11-52
11.3.9 MC_MoveSuperimposed 11-61
11.3.10 MC_HaltSuperimposed 11-68
11.3.11 MC_SetPosition 11-73
11.3.12 MC_SetOverride 11-83
11.3.13 MC_Reset 11-87
11.3.14 DMC_SetTorque 11-90
11.3.15 MC_ReadAxisError 11-94
11.3.16 MC_ReadActualPosition 11-96
11.3.17 MC_ReadStatus 11-101
11.3.18 MC_ReadMotionState 11-106
11.3.19 DMC_ReadParameter_Motion 11-111
11.3.20 DMC_WriteParameter_Motion 11-115
11.3.21 DMC_TouchProbe 11-119

11.4 Multi-axis Instructions 11-128
11.4.1 MC_GearIn 11-128
11.4.2 MC_GearOut 11-134
11.4.3 MC_CombineAxes 11-139
11.4.4 Introduction of Electronic Cam 11-147
11.4.5 MC_CamIn 11-148
11.4.6 MC_CamOut 11-169

11.5 Application Instructions 11-175
11.5.1 Rotary Cut Technology 11-175
11.5.2 Rotary Cut Parameters 11-176

11.5.3 Control Feature of Rotary Cut Function 11-177
11.5.4 Introduction to Cam Curve with Rotary Cut Function 11-178
11.5.5 Rotary-cut Instructions 11-182
11.5.5.1 APF_RotaryCut_Init 11-182
11.5.5.2 APF_RotaryCut_In 11-185
11.5.5.3 APF_RotaryCut_Out 11-187
11.5.6 Application Example of Rotary Cut Instructions 11-189

11.1 Table of Motion Control Instructions

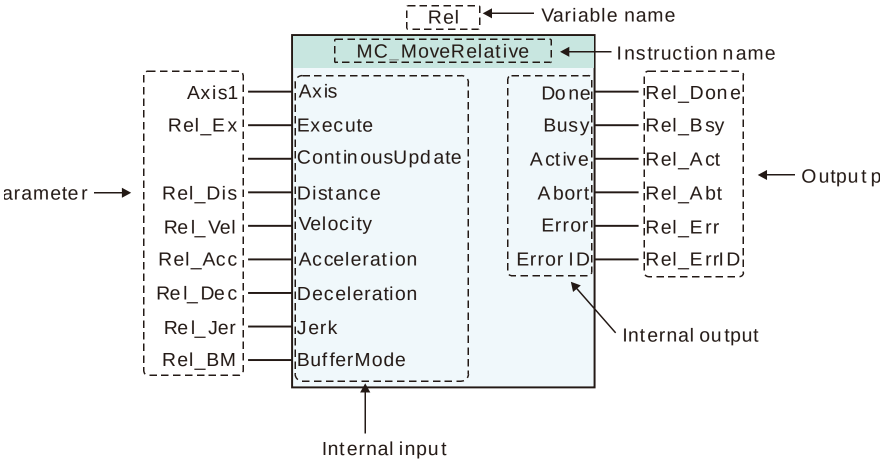
Instruction set	Instruction code	Instruction name
Single-axis instructions	MC_Power	Power Servo
	MC_Home	Homing
	MC_MoveVelocity	Velocity
	MC_Halt	Temporary Stop
	MC_Stop	Stop
	MC_MoveRelative	Relative Positioning
	MC_MoveAdditive	Additive Positioning
	MC_MoveAbsolute	Absolute Positioning
	MC_MoveSuperimposed	Superimposed Positioning
	MC_HaltSuperimposed	Halt Superimposing
	MC_SetPosition	Set Position
	MC_SetOverride	Set Override Factors
	MC_Reset	Reset
	DMC_SetTorque	Set Torque
	MC_ReadAxisError	Read Axis Error
	MC_ReadActualPosition	Read Actual Position
	MC_ReadStatus	Reead Axis Status
	MC_ReadMotionState	Read Motion State
	DMC_ReadParameter_Motion	Read A Parameter
	DMC_WriteParameter_Motion	Write A Parameter Value
	DMC_TouchProbe	Capture Axis Position
Multi-axis instructions	MC_GearIn	Start E-Gear Operation
	MC_GearOut	End E-Gear Operation
	MC_CombineAxes	Combine Axes
	MC_CamIn	Start E-Cam Operation
	MC_CamOut	End E-Cam Operation
Application instructions	APF_RotaryCut_Init	Initialize Rotary Cut
	APF_RotaryCut_In	Rotary Cut In
	APF_RotaryCut_Out	Rotary Cut Out

11

11.2 About Motion Control Instructions

11.2.1 Composition of A Motion Control Instruction

The instructions starting with “MC_” or “DMC” belong to motion instructions.



11.2.2 Program Languages that Motion Control Instructions Support

The motion instructions support the following two types of program languages.

For details, refer to the software help file.

- Ladder diagram (LD)
- Structured text (ST)

11.2.3 Configuration of Motion Control Instructions

Motion instructions can only be added to the motion event task. Otherwise, they can not be executed if they are added to other types of tasks.

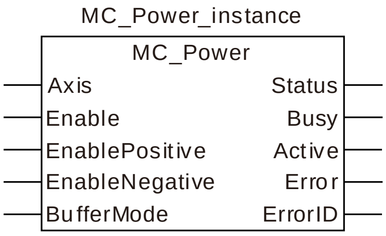
The following table shows task types and whether motion instruction can be added to these tasks.

Task type		Whether motion intructions can be added or not
Cyclic		No
Freewheeling		No
Triggered by event	Motion event	Yes
	Non-motion event	No

11.3 Single-axis Instructions

11.3.1 MC_Power

FB/FC	Explanation	Applicable model
FB	MC_Power is used to enable or disable the corresponding servo axis.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Enable* changes to TRUE
Enable	The instruction is executed when *Enable* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	When *Enable* changes to TRUE
EnablePositive	The specified axis is allowed to move forward only under the condition that *Enable* is TRUE and *EnablePositive* is also TRUE.	BOOL	TRUE or FALSE (FALSE)	When *Enable* changes to TRUE
EnableNegative	The specified axis is allowed to move reversely only under the condition that *Enable* is TRUE and *EnableNegative* is also TRUE.	BOOL	TRUE or FALSE (FALSE)	When *Enable* changes to TRUE
Buffermode	Specify the behavior of MC_Power when *Enable* changes to FALSE	MC_Buffer Mode	0: mcAborting 1: mcBuffered (0)	When *Enable* changes to TRUE

Note:

Motion control instructions can control servo axes for corresponding motions only after Power ON. When Power OFF, no motion control instructions can be executed.

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Status	TRUE when the axis is enabled.	BOOL	TRUE/FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE

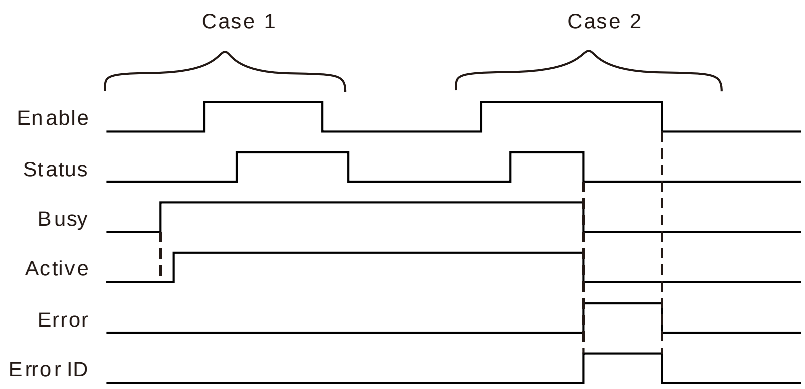
Parameter name	Function	Data type	Valid range
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	

11

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Status	◆ When the axis is enabled.	◆ When *Enable* changes to FALSE. ◆ When *Error* changes to TRUE.
Busy	◆ When the instruction is being executed.	◆ When *Error* changes to TRUE.
Active	◆ The instruction starts controlling the axis.	◆ When *Error* changes to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When an abnormal situation is cleared.

- **Output Update Timing Chart**



Case 1 : When MC_Power instruction is executed for the first time, *Busy* changes to TRUE and one cycle later, *Active* changes to TRUE. After *Enable* changes from FALSE to TRUE and the axis is enabled, *Status* changes to TRUE. After *Enable* changes from TRUE to FALSE and the axis is disabled, *Status* changes from TRUE to FALSE.

Case 2 : When an error occurs in the execution of the instruction, *Error* changes to TRUE, the corresponding error code is contained in *ErrorID* and meanwhile *Status*, *Busy* and *Active* all change to FALSE. *Error* changes to FALSE when the error is cleared.

- **Function**

This instruction is used to enable or disable the corresponding servo axis.

1. *Status* will not change to TRUE if the axis is not enabled yet after *Enable* is set to TRUE. Please make sure that *Status* has already changed to TRUE before the axis is started to move.
2. When *Enable* and *EnablePositive* are both TRUE, the axis specified by a motion instruction is allowed to move in the positive direction.
3. When *Enable* is TRUE and *EnablePositive* is FALSE, the axis specified by a motion instruction is prohibited to move in the positive direction. In this case, there will be an error in existence if some motion instruction is used to move the axis forward. If the axis moves from forward to backward, the instruction which is controlling the motion of the axis will be aborted and the axis will stop moving and enter the state of Standstill.
4. When *Enable* and *EnableNegative* are both TRUE, the axis specified by a motion instruction can move in the negative direction.

5. When *Enable* is TRUE and *EnableNegative* is FALSE, the axis specified by a motion instruction is prohibited to move in the negative direction. In this case, there will be an error in existence if some motion instruction is used to move the axis backward. If the axis moves from backward to forward, the instruction which is controlling the motion of the axis will be aborted and the axis will stop moving and enter the state of Standstill.
6. When the axis moves in the positive direction and *EnablePositive* changes from TRUE to FALSE, the axis will decelerate its speed at the deceleration rate specified by the current motion instruction controlling the axis and finally stop at the velocity of 0. When the axis moves in the negative direction and *EnableNegative* changes from TRUE to FALSE, the axis will decelerate its speed at the deceleration rate specified by the current motion instruction controlling the axis and finally stop at the velocity of 0.
7. In principle, only one MC_Power can be used for one axis. If there are two MC_Power instructions in the program where the same axis is controlled, please refer to the execution result of the MC_Power which is executed late.
8. While a motion instruction is controlling the axis, *Enable* of MC_Power changes from TRUE to FALSE and whether the axis enters the Disable state depends on the value of Buffermode.
9. Buffermode

   BufferMode specifies the behavior of MC_Power when *Enable* changes from TRUE to FALSE.

Input	BufferMode selection	Function
Enable	0: mcAborting (Interrupt)	When *Enable* changes from TRUE to FALSE, the axis will stop moving immediately and become disabled (The state machine enters the state of Disabled). **Precaution: Be cautious during operation in case of any danger to personnel or devices!**
	1: mcBuffered (Waiting)	When *Enable* changes from TRUE to FALSE, the axis will not enter the Disabled state immediately. Only when the axis stops moving, the state machine goes to the Standstill state first and one cycle later, it enters the Disabled state.

Programming Example 1

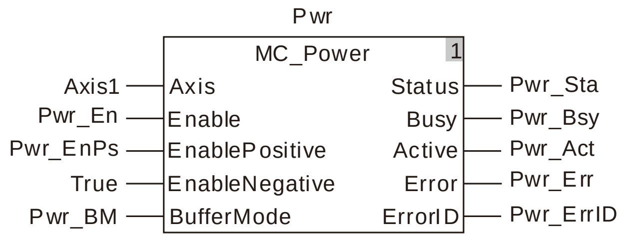
The example of MC_Power instruction execution

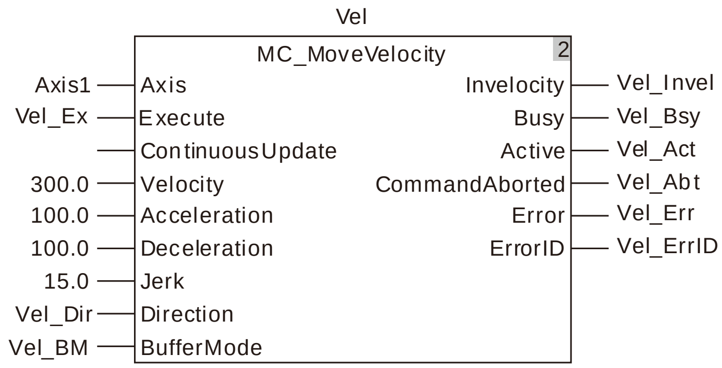
When Pwr_En is TRUE and Pwr_EnPs is FALSE, the axis specified by the motion instruction is forbidden to move in the positive direction. While the axis is moving in the positive direction and Pwr_EnPs changes from TRUE to FALSE, the axis will decelerate its speed at the deceleration rate specified by the current motion instruction controlling the axis till the velocity of the axis reaches 0.

1. The variables and program

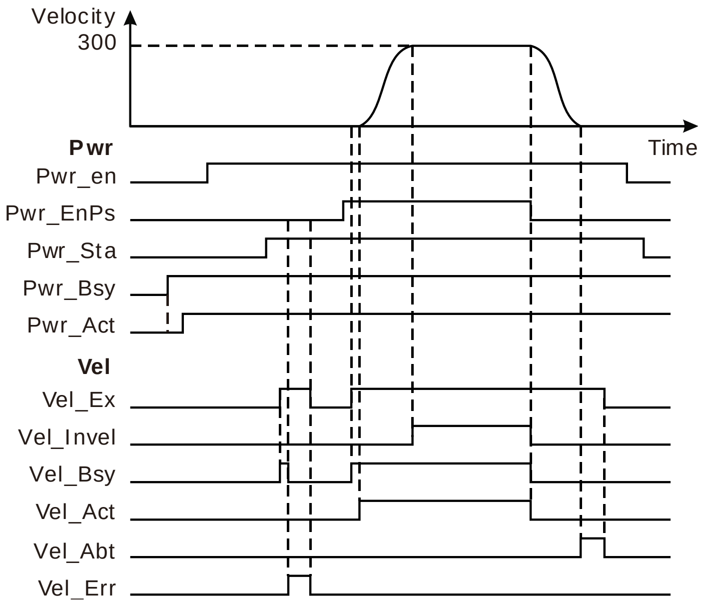
Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_EnPs	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	

Variable name	Data type	Initial value
Vel _Ex	BOOL	FALSE
Vel _Dir	MC_DIRECTION	1
Vel _BM	MC_Buffer_Mode	0
Vel _Invel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ErrID	WORD	





2. Motion Curve and Timing Chart



❖ When Vel _Ex changes to TRUE for the first time, Vel _Bsy changes to TRUE and one cycle later, Vel _Err changes to TRUE. At this moment, the servo motor could not move because Pwr_EnPs is FALSE.

- ❖ When Pwr_EnPs is TRUE and Vel _Ex changes to TRUE for the second time, Vel _Bsy changes to TRUE; one cycle later, Vel _Act changes to TRUE and the servo motor starts moving in the positive direction. When the servo motor reaches the target velocity, Vel _Invel changes to TRUE.
- ❖ When Pwr_EnPs changes to FALSE, MC_Velocity instruction is aborted and the servo motor begins to decelerate its speed at the deceleration rate specified by MC_Velocity instruction. When the velocity is decreased to 0, CommandAborted changes to TRUE.
- ❖ When Vel _Ex changes to FALSE, Vel _Abt changes to FALSE.
- ❖ When Pwr_En changes to FALSE, Pwr_Sta change to FALSE after the axis is disabled.

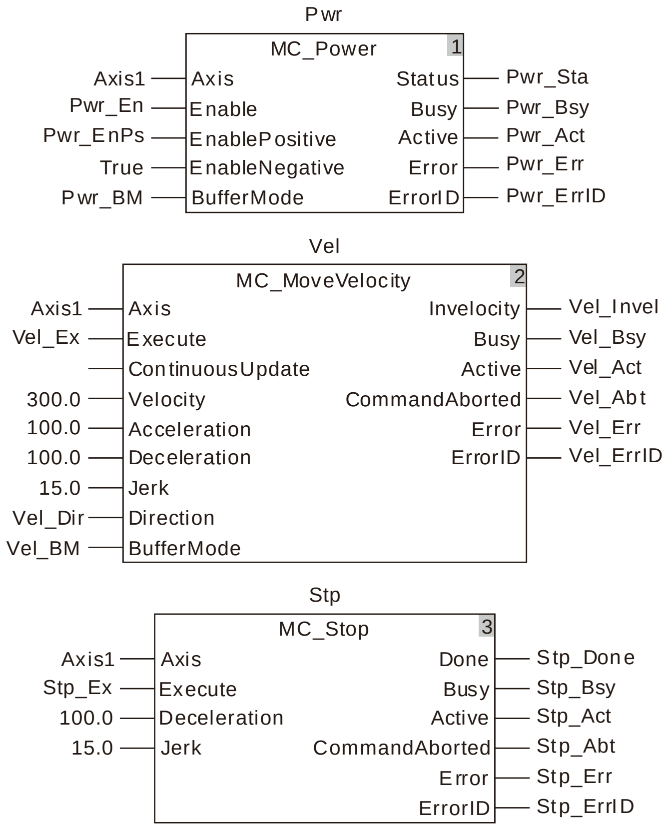
Programming Example 2

The example of Vel _BM =0

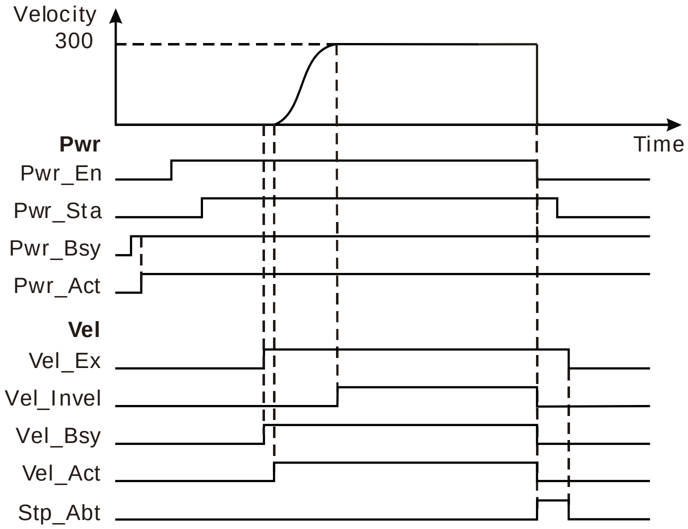
When the value of Vel _BM is set to 0 and Pwr_En changes from TRUE to FALSE, the axis will enter the Disabled state and the velocity will be decreased to 0 immediately.

1. The variables and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_EnPs	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	1
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel _Ex	BOOL	FALSE
Vel _Dir	MC_DIRECTION	1
Vel _BM	MC_Buffer_Mode	0
Vel _Invel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ErrID	WORD	
Stp	MC_Stop	
Stp _Ex	BOOL	FALSE
Stp _Done	BOOL	
Stp _Bsy	BOOL	
Stp _Act	BOOL	
Stp _Abt	BOOL	
Stp _Err	BOOL	
Stp _ErrID	WORD	



2. Motion Curve and Timing Chart



- ❖ When Vel _Ex changes to TRUE, the servo motor starts moving in the positive direction. When the speed of the servo motor reaches target velocity, Vel _Invel changes to TRUE.
- ❖ When Pwr_En changes to FALSE, the speed of the servo motor is decreased to 0 and the axis enters the Standstill state right away. At the same time, Vel _Abt changes to TRUE and Vel _Bsy and Vel _Act change to FALSE. Pwr_Sta changes to FALSE after the axis is disabled.
- ❖ When Vel _Ex changes to FALSE, Vel _Abt changes to FALSE.

Programming Example 3

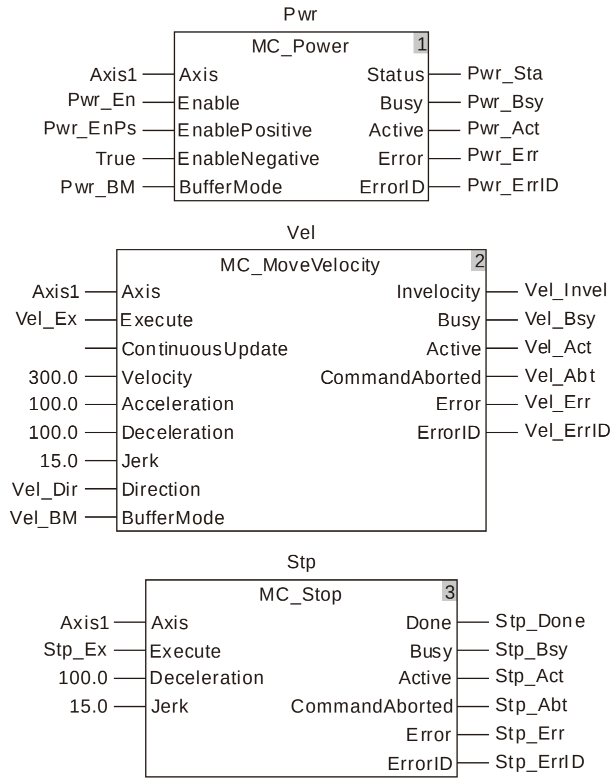
The example of Vel_BM =1

When the value of *Buffermode* is set to 1 and *Enable* changes from TRUE to FALSE, there will be no change in *Status* of MC_Power unless the axis stops moving. When the axis stops moving, the axis will enter the Standstill state first and one cycle later, it will go to the Disabled state.

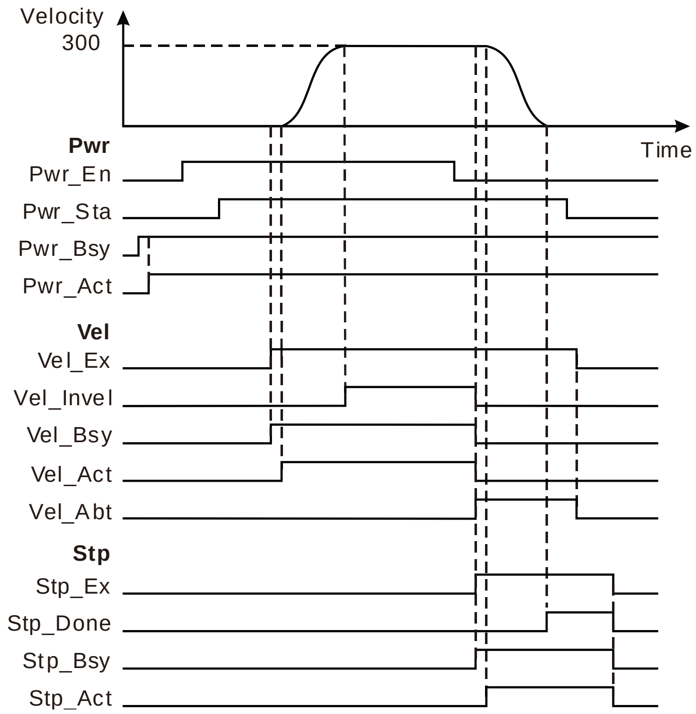
11

1. The variables and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_EnPs	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel _Ex	BOOL	FALSE
Vel _Dir	MC_DIRECTION	1
Vel _BM	MC_Buffer_Mode	0
Vel _Invel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ErrID	WORD	
Stp	MC_Stop	
Stp _Ex	BOOL	FALSE
Stp _Done	BOOL	
Stp _Bsy	BOOL	
Stp _Act	BOOL	
Stp _Abt	BOOL	
Stp _Err	BOOL	
Stp _ErrID	WORD	



2. Motion Curve and Timing Chart



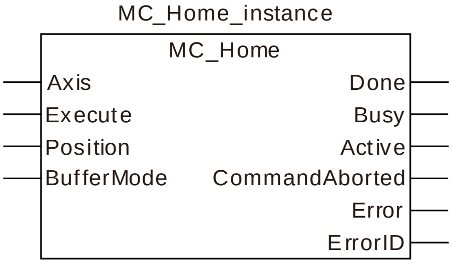
- ❖ When Vel _Ex changes to TRUE, Vel _Bsy changes to TRUE; one cycle later, Vel _Act changes to TRUE and the servo motor starts moving in the positive direction. When the speed of the servo motor reaches the target velocity, Vel _Invel changes to TRUE.
- ❖ When Pwr_En changes to FALSE, the axis will not enter the Standstill state immediately. When Stp _Ex changes to TRUE, Stp _Bsy changes to TRUE; one cycle later, Stp _Act changes to TRUE and the servo motor begins to decelerate. When the speed of the servo motor drops to 0,

Stp _Done changes to TRUE. Meanwhile, the axis enters the Standstill state and Pwr_Sta changes to FALSE. One cycle later, the axis goes to the Disabled state.

- When Vel _Ex changes to FALSE, Vel _Abt changes to FALSE.
- When Stp _Ex changes to FALSE, Stp _Done, Stp _Bsy and Stp _Act change to FALSE.

11.3.2 MC_Home

FB/FC	Explanation	Applicable model
FB	MC_Home controls the servo motor to perform the homing action according to the set mode and velocity.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
Position	The servo home point offset, Unit: unit.	LREAL	Negative number, positive number and 0 (0)	When *Execute* changes from FALSE to TRUE
BufferMode	Reserved	-	-	-

- **Output Parameters**

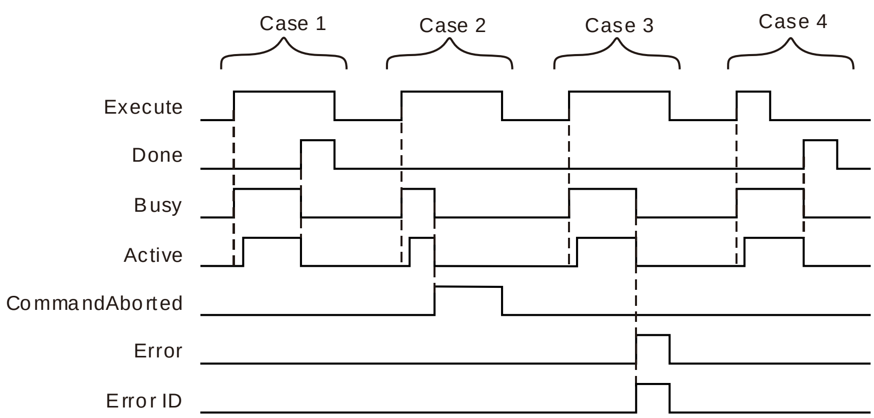
Parameter name	Function	Data type	Valid range
Done	TRUE when the homing is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When homing is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
		to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* is set to TRUE when the instruction is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1 : When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one cycle later, *Active* changes to TRUE. When the positioning is completed, *Done* changes to TRUE and meanwhile *Busy* and *Active* change to FALSE.

Case 2 : When the instruction is aborted by other instruction after *Execute* changes from FALSE to TRUE, *CommandAborted* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3 : When an error occurs such as axis alarms or Offline after *Execute* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error code. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

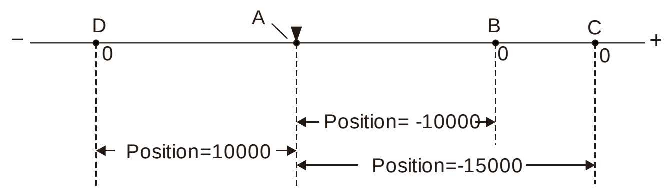
Case 4：*Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE in the course of execution of the instruction. Meanwhile, *Busy* and *Active* change to FALSE and one cycle later, *Done* changes to FALSE.

- **Function**

1. According to the set homing mode, the MC_Hme instruction is used for connecting the home switch and positive limit switch or negative limit switch to the external input points of the servo drive so as to achieve the homing function.
2. For real axes, the homing mode and phase-1 speed and phase-2 speed of the homing are set in the software axis parameter setting. See Appendix D for details on homing modes. For virtual axes, the homing mode can only be set to mode 35.
3. The instruction can be executed only while the axis is in Stanstill state. Otherwise, an error will occur.
4. Position parameter defines the offset between the mechanical zero point and servo reference zero point as the figure below:

A	Mechanical zero point, where the photoelectric sensor is.
▼	The position is where the servo is after execution of the instruction is finished.

For different *Position* value, the servo will eventually stop at the mechanical point A under the control of this instruction. But the reference zero point of the servo position will change as shown below.



As Position=10000, the reference zero point of the servo position is point D and point A position is 10000;

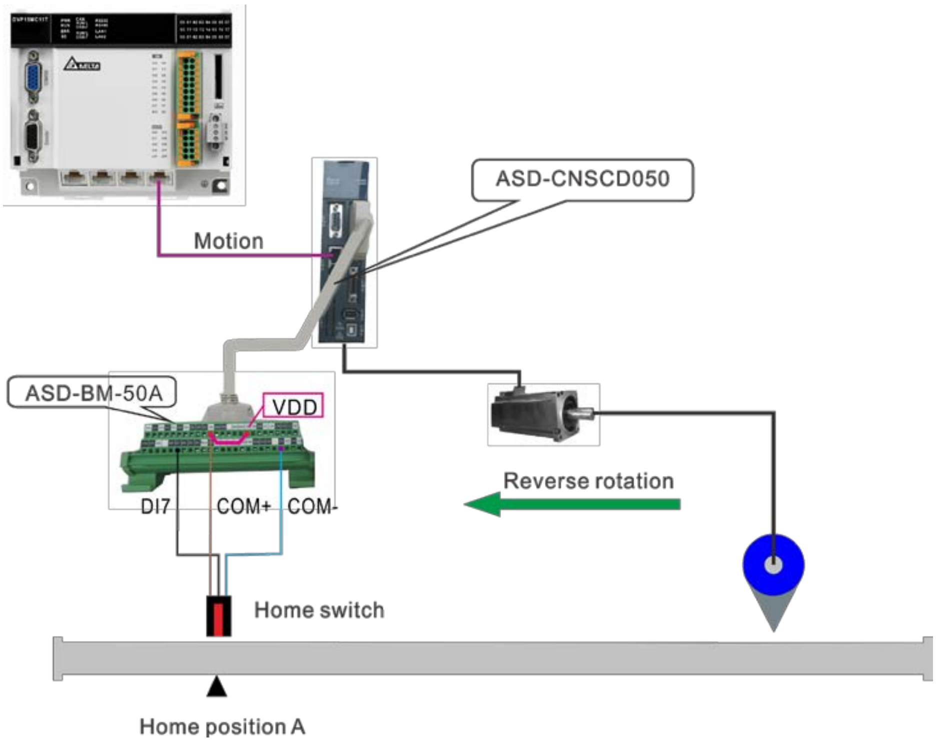
As Position=-15000, the reference zero point of the servo position is point C and point A position is -15000;

As Position=-10000, the reference zero point of the servo position is point B and point A position is -10000.

Programming Example

Select an appropriate homing mode via the positions of the mechanism and photoelectric switch. When Hom _Ex changes from FALSE to TRUE, the motion controller controls the servo motor to rotate and drive the mechanism to return to the mechanical zero point position A.

- Hardware wiring



Note:

- During wiring, COM+ and VDD must be shorted.
- Of the photoelectric switch, the brown terminal (24V+) is connected to COM+, the blue terminal (0V) is connected to COM- and the black terminal (Signal cable) is connected to DI7.
- The DI7 function is set to the home switch, i.e. P2-16 is set to 124.

- Homing mode selection

It can be seen from the hardware wiring figure that the mechanism regards the home switch position as the mechanical zero point position A. The home switch is OFF before searching for the home. While the mechanism is searching for the home point, the servo rotates reversely at beginning and homing mode 21 can be selected to achieve the homing.

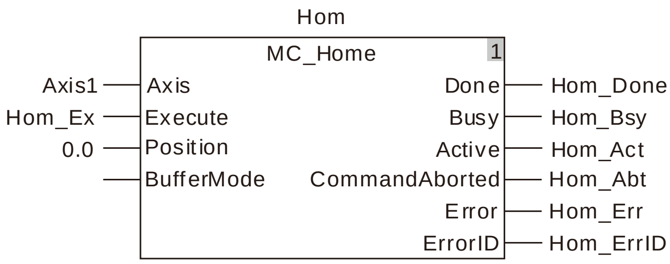
The settings for homing in the corresponding axis parameters are as follows.

Homing mode	21
The first-phase speed (the speed for finding the home switch, Unit: r/m)	100
The second-phase speed (The speed from finding the home switch to reaching the mechanical zero point, Unit: r/m)	10

Note: The set axis parameters are valid after being downloaded.

- **The variable table and program**

Variable name	Data type	Initial value
Hom	MC_Home	
Axis1	USINT	1
Hom_Ex	BOOL	FALSE
Hom_Done	BOOL	
Hom_Bsy	BOOL	
Hom_Act	BOOL	
Hom_Abt	BOOL	
Hom_Err	BOOL	
Hom_ErrID	WORD	



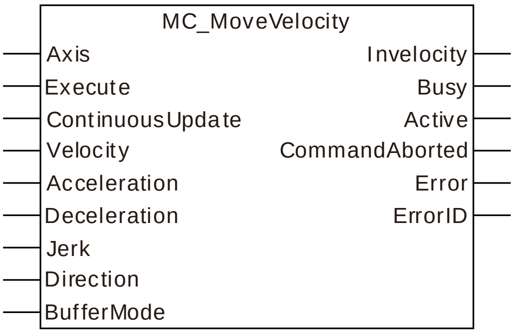
❖ When Hom_Ex changes from FALSE to TRUE, the motion controller controls the motion of the servo motor. The mechanism starts to run reversely, rotates forward after reaching the home switch and then stops at the mechanical zero point. And the mechanism is driven to return to the mechanical zero point A by doing so.

❖ When the home switch is met, the homing is completed and Hom_Done is set to ON.

11.3.3 MC_MoveVelocity

FB/FC	Explanation	Applicable model
FB	MC_MoveVelocity controls the axis motion based on the set acceleration and deceleration till the set target velocity is reached and then the axis moves at the set speed.	DVP15MC11T

MC_MoveVelocity_instance



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved	-	-	-
Velocity	Specify the target speed (Unit: unit/second)	LREAL	Positive number （The variable value must be set）	When *Execute* changes from FALSE to TRUE
Acceleration	Specify the target acceleration (Unit: unit/second^2)	LREAL	Positive number （The variable value must be set）	When *Execute* changes from FALSE to TRUE
Deceleration	Specify the target deceleration (Unit: unit/second^2)	LREAL	Positive number （The variable value must be set）	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of target acceleration or deceleration. (Unit: Unit/s^3)	LREAL	Positive number （The variable value must be set）	When *Execute* changes from FALSE to TRUE
Direction	Specify the rotation direction 1: Positive direction 3: Negative direction 4: Current direction (When the motor is in stop state, the current direction is the positive direction.)	MC_Direction	1: mcPositiveDirection, 3: mcNegativeDirection 4: mcCurrentDirection, （1）	When *Execute* changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
BufferMode	Specify the behavior when executing two instructions. 0: Aborting 1: Buffered 2: BlendingLow 3: BlendingPrevious 4: BlendingNext 5: BlendingHigh	MC_Buffer_Mode	0: mcAborting 1: mcBuffered 2: mcBlendingLow 3: mcBlendingPrevious 4: mcBlending _Next 5: mcBlending _High (0)	When *Execute* changes from FALSE to TRUE

Notes:

1. MC_MoveVelocity instruction is executed when *Execute* changes from FALSE to TRUE. The instruction can be re-executed when *Execute* of the instruction changes from FALSE to TRUE again no matter whether the instruction execution is completed. At the moment, the parameters including *Velocity*, *Acceleration*, *Deceleration, Jerk* and *Direction* are effective again and other parameters are ineffective. When the velocity instruction has the BufferMode relationship with other motion instruction, the parameters will be valid after the instruction parameters are changed and the instruction is re-triggered. The previous buffermode relation remains and the transition speed will be re-calculated.
2. *Invelocity* remains TRUE even if the target speed is changed through MC_SetOverride after the velocity instruction execution is completed (that is, *Invelocity* changes from FALSE to TRUE.) *Invelocity* will change from FALSE to TRUE when the new target speed is reached after the target speed is changed through MC_SetOverride before the execution of MC_MoveVelocity is completed (when *Invelocity* is FALSE.)
3. Refer to section 10.2 for the relation among *Position, Velocity, Acceleration and Jerk*.
4. Refer to section 10.3 for details on *BufferMode*.

- **Output Parameters**

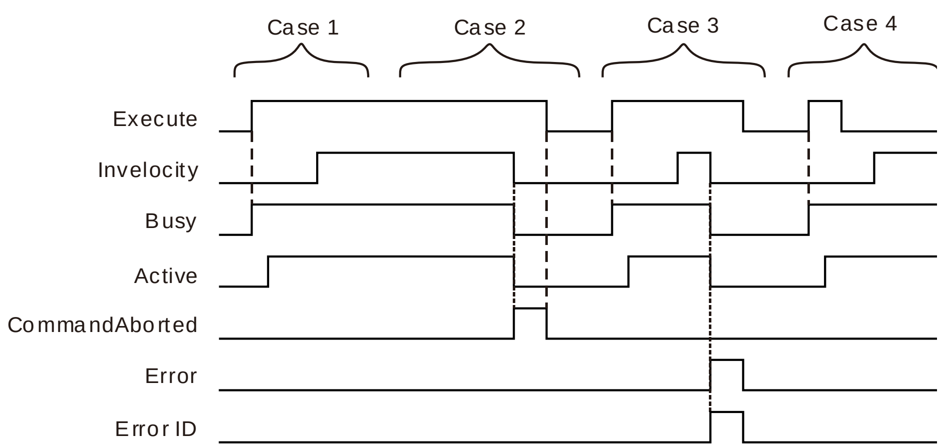
Parameter name	Function	Data type	Valid range
Invelocity	TRUE when the target velocity is reached.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Invelocity	◆ When the target velocity is reached.	◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE ◆ *Invelocity* changes to FALSE immediately when *Execute* changes from FALSE to TRUE again if the input parameter values are revised after the target velocity is reached. If the input parameter values are not changed after the instruction execution is completed and *Execute* changes from FALSE to TRUE again, *Invelocity* changes to FALSE immediately and *Invelocity* changes to TRUE in the

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
		next cycle.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* is set to TRUE when the instruction is aborted by other instruction after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1： When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one cycle later, *Active* changes to TRUE. When the target velocity is reached, *Invelocity* changes to TRUE and meanwhile, *Busy* and *Active* remain TRUE.

Case 2： When *Execute* is TRUE, the instruction is aborted by other instruction and *CommandAborted* changes to TRUE. Meanwhile, *Invelocity*, *Busy and Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3： When an error occurs such as parameter error while *Execute* is TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error code. Meanwhile, *Invelocity, Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4： In the course of execution of the instruction, *Invelocity* changes to TRUE when the target velocity is reached after *Execute* changes from TRUE to FALSE. Meanwhile, *Busy* and *Active* remain TRUE.

- **Function**

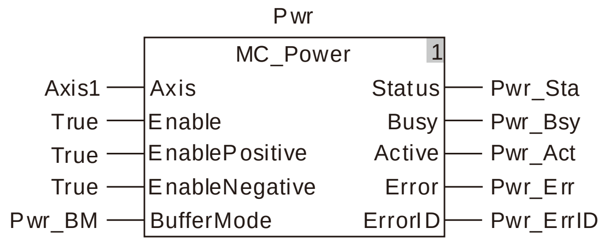
MC_MoveVelocity controls the axis to speed up or down according to the set acceleration, deceleration and jerk till the set target velocity is reached and after that the axis moves at the target speed. The direction of the uniform motion is determined by the input parameter *Direction*. The *Direction* value 1 indicates the positive direction, 3 is the negative direction and 4 is the current direction. If *Direction* value is set to 4 and the axis is in STOP state before the MC_MoveVelocity instruction is executed, the axis will move in the positive direction.

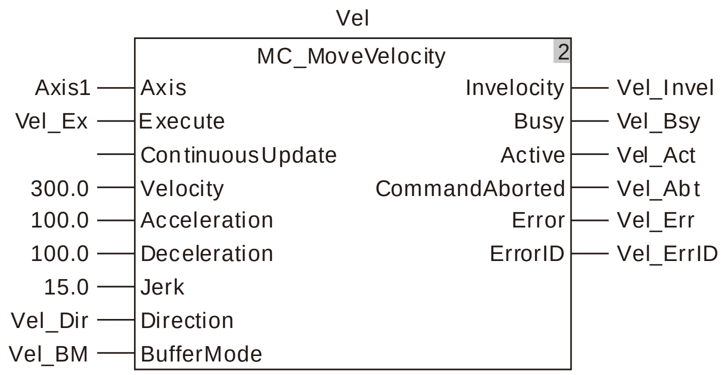
Programming Example 1

The programming example is as follows when one MC_ MoveVelocity instruction is used.

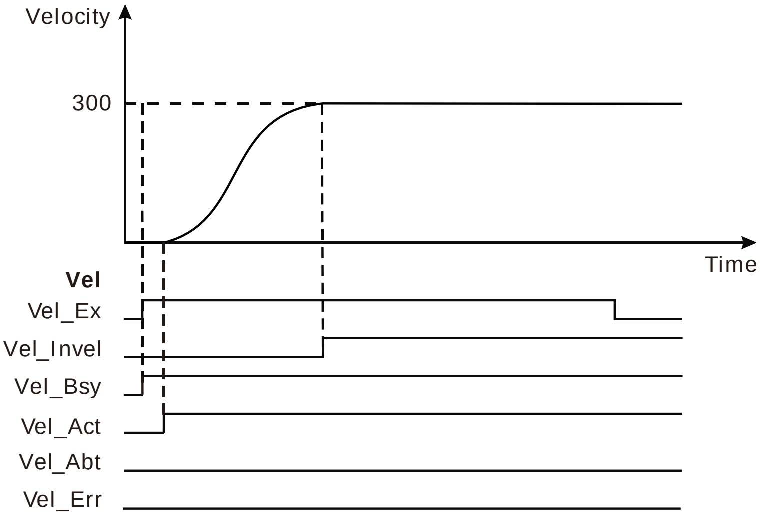
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_BM	MC_Buffer_Mode	1
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel_Ex	BOOL	FALSE
Vel_Dir	MC_DIRECTION	1
Vel_BM	MC_Buffer_Mode	0
Vel_Invel	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	





2. Motion Curve and Timing Chart



❖ When Vel_Ex changes from FALSE to TRUE, Vel_Bsy changes to TRUE. One cycle later, Vel_Act changes to TRUE and the execution of the velocity instruction starts. When the target velocity is reached, Vel_Invel changes to TRUE and Vel_Bsy and Vel_Act remain TRUE.

❖ When Vel_E changes from TRUE to FALSE, Vel_Inve, Vel_Bsy and Vel_Act remain TRUE.

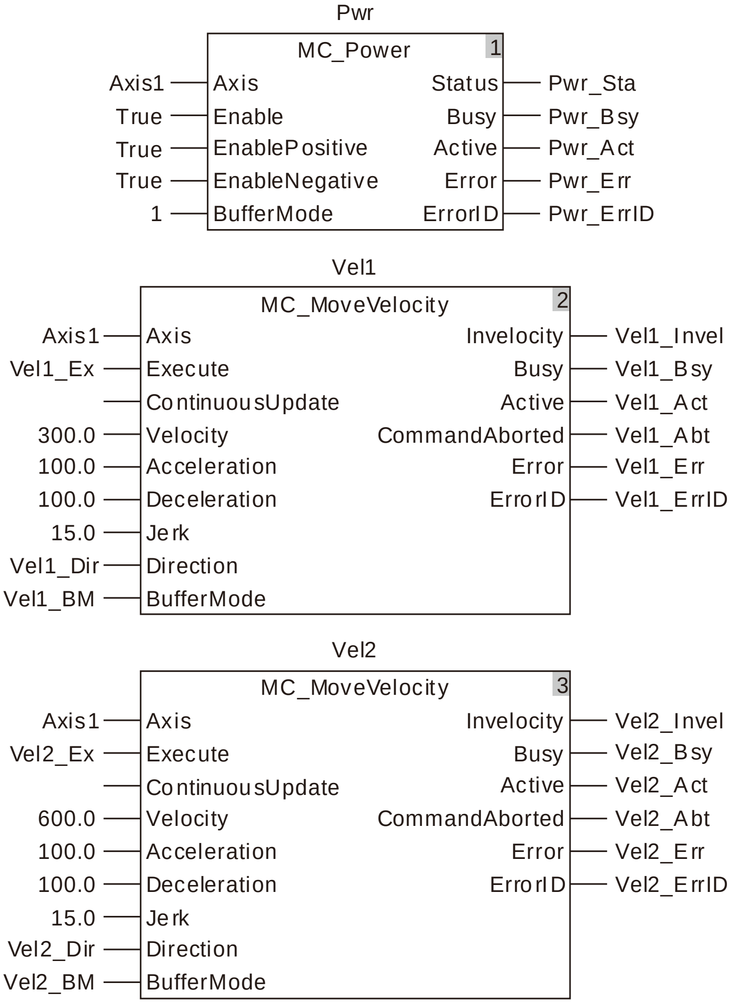
Programming Example 2

Below is the example that one MC_MoveVelocity instruction aborts another MC_MoveVelocity instruction.

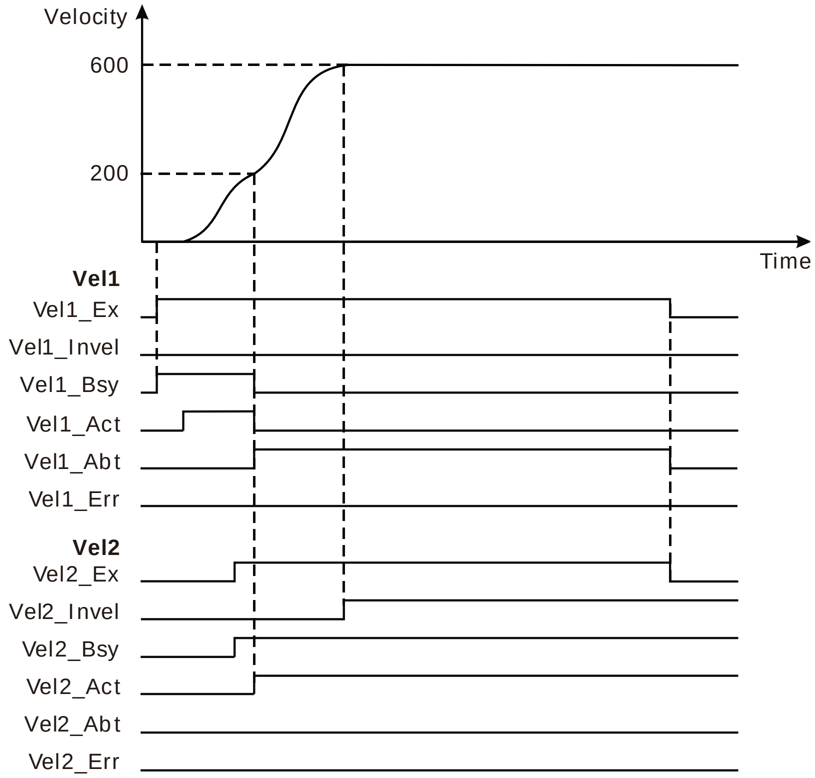
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_BM	MC_Buffer_Mode	1
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel1	MC_MoveVelocity	
Vel1_Ex	BOOL	FALSE
Vel1_Dir	MC_DIRECTION	1
Vel1_BM	MC_Buffer_Mode	0
Vel1_Invel	BOOL	
Vel1_Bsy	BOOL	
Vel1_Act	BOOL	
Vel1_Abt	BOOL	
Vel1_Err	BOOL	
Vel1_ErrID	WORD	
Vel2	MC_MoveVelocity	
Vel2_Ex	BOOL	FALSE
Vel2_Dir	MC_DIRECTION	1
Vel2_BM	MC_Buffer_Mode	0

Variable name	Data type	Initial value
Vel2_Invel	BOOL	
Vel2_Bsy	BOOL	
Vel2_Act	BOOL	
Vel2_Abt	BOOL	
Vel2_Err	BOOL	
Vel2_ErrID	WORD	



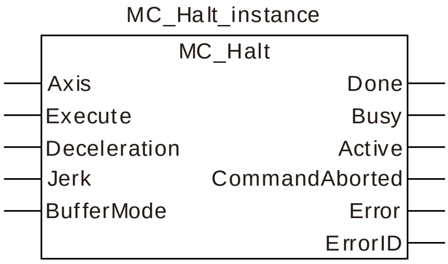
2. Motion Curve and Timing Chart



- ❖ When Vel1_Ex changes from FALSE to TRUE, Vel1_Bsy changes to TRUE. One cycle later, Vel1_Act changes to TRUE and the first MC_MoveVelocity instruction starts being executed. When the target velocity is not reached, Vel2_Ex changes from FALSE to TRUE and Vel2_Bsy changes to TRUE. One cycle later, Vel2_Act changes to TRUE, the first MC_MoveVelocity instruction is aborted, Vel1_Abt changes to TRUE and the axis starts to perform the second MC_MoveVelocity instruction. When the target velocity is reached, Vel2_Invel changes to TRUE and meanwhile, Vel2_Bsy and Vel2_Act remain TRUE.
- ❖ When Vel1_Ex changes from TRUE to FALSE, Vel1_Abt changes to FALSE. When Vel2_Ex changes from TRUE to FALSE, Vel2_Invel, Vel2_Bsy and Vel2_Act remain TRUE.

11.3.4 MC_Halt

FB/FC	Explanation	Applicable model
FB	MC_Halt is used to make the axis decelerate at a given deceleration rate till it stops.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
Deceleration	Specify the target deceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of the target acceleration or deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
BufferMode	Specify the behavior when executing two instructions. 0: Aborting 1: Buffered	MC_Buffer_Mode	0: mcAborting 1: mcBuffered (0)	When *Execute* changes from FALSE to TRUE

Note:

1. MC_Halt instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction changes from TRUE to FALSE in the course of the instruction execution.
2. While *Execute* changes from FALSE to TRUE once more in the course of execution of MC_Halt, there is no impact on the instruction execution and the instruction will continue being executed in the previous way. When *Execute* changes from FALSE to TRUE once again after the instruction execution is completed, the instruction can be re-executed.
3. Refer to section10.2 for the relation between *Deceleration and Jerk*.
4. Refer to section10.3 for details on *BufferMode*.

- **Output Parameters**

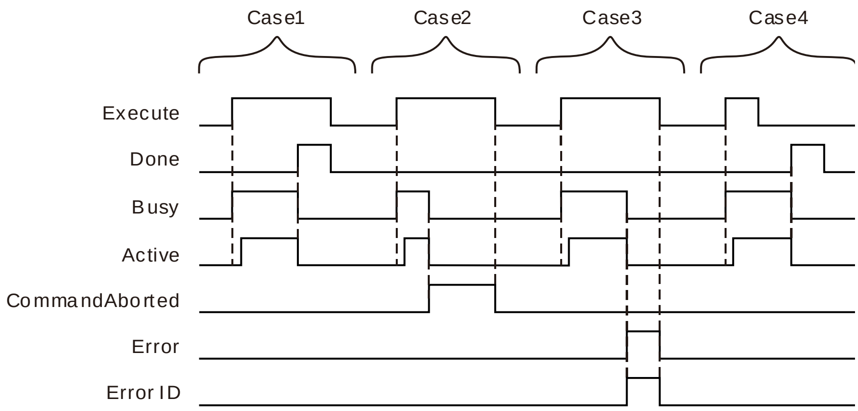
Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE/FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE/FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE/FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE/FALSE
Error	TRUE when there is an error.	BOOL	TRUE/FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	-

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the deceleration ends and the axis speed is decreased to 0.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When CommandAborted changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When CommandAborted changes to TRUE.
Command Aborted	◆ When the instruction execution is aborted by other motion instruction.	◆ When *Execute* changes from TRUE to FALSE. ◆ CommandAborted changes to TRUE when the instruction is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, CommandAborted changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

11

- **Output Update Timing Chart**



Case 1 : When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one period later, *Active* changes to TRUE. When the deceleration ends and the axis speed is decreased to 0, *Done* changes to TRUE and meanwhile *Busy* and *Active* change to FALSE.

Case 2 : After *Execute* changes from FALSE to TRUE and the instruction is aborted by other instruction, *CommandAborted* changes to TRUE and meanwhile *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3 : When an error occurs such as axis alarms or Offline after *Execute* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows the corresponding error code. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4 : In the course of execution of the instruction, *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE. Meanwhile, *Busy* and *Active* change to FALSE and one period later, *Done* changes to FALSE.

- **Function**

MC_Halt is used to make the axis decelerate at a given deceleration rate till it stops.

- The state machine enters DiscreteMotion as MC_Halt starts being executed. When the axis speed is decreased to 0, *Done* changes to TRUE and meanwhile, the state machine enters Standstill.
- Compared to MC_Stop instruction, MC_Halt instruction can not make the axis locked and thus the controller can perform other motion instruction on it.
  MC_Halt can be aborted through performing other motion instruction when the axis is decelerated during execution of MC_Halt. Other motion instruction can be executed by the controller to restart the axis after MC_Halt execution is over and the axis has stopped.

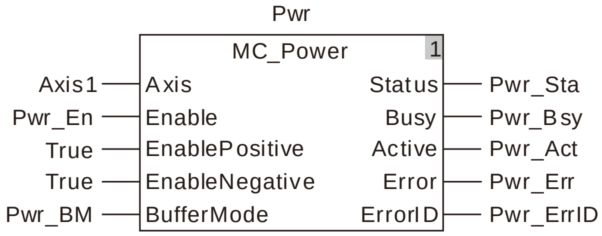
Programming Example

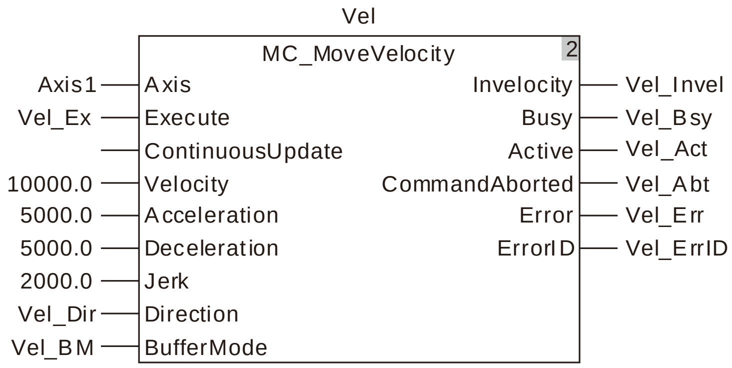
The example of MC_Halt execution is shown below.

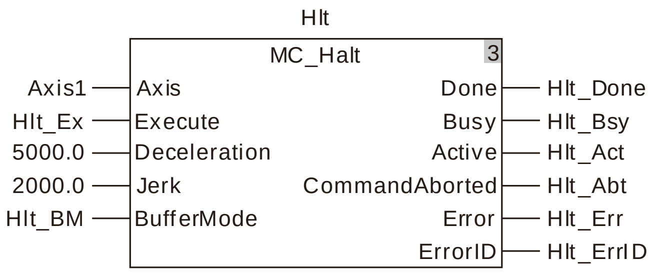
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	

Variable name	Data type	Initial value
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel_Ex	BOOL	FALSE
Vel_Dir	MC_DIRECTION	1
Vel_BM	MC_Buffer_Mode	0
Vel_Invel	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	
Hlt	MC_Halt	
Hlt_Ex	BOOL	FALSE
Hlt_BM	MC_Buffer_Mode	0
Hlt_Done	BOOL	
Hlt_Bsy	BOOL	
Hlt_Act	BOOL	
Hlt_Abt	BOOL	
Hlt_Err	BOOL	
Hlt_ErrID	WORD	

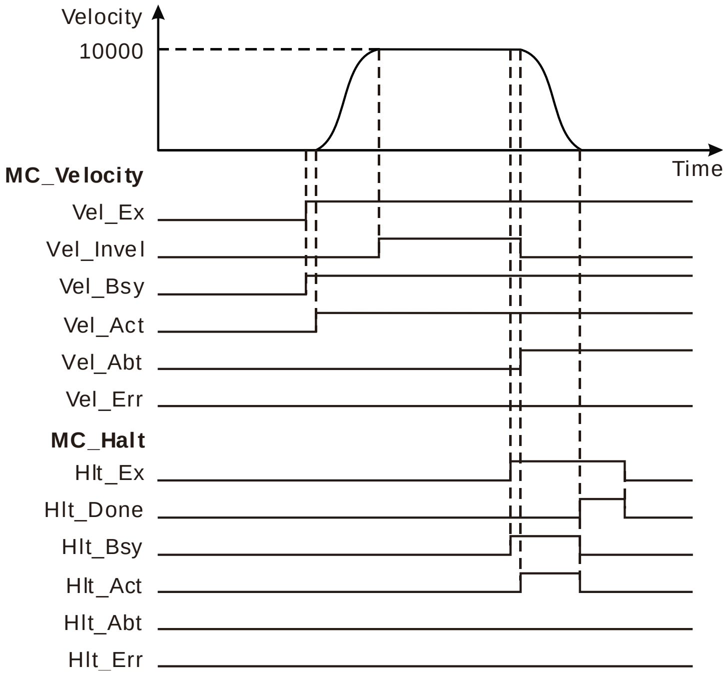






11

2. Motion Curve and Timing Charts:

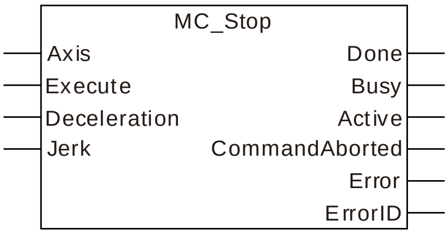


- ❖ When Vel_Ex changes to TRUE, Vel_Bsy changes to TRUE and one period later, Vel_Act changes to TRUE and the servo motor starts to move forward. Vel_Invel changes to TRUE as the servo motor reaches the target velocity.
- ❖ When Hlt_Ex changes to TRUE, Hlt_Bsy changes to TRUE and one period later, Hlt_Act changes to TRUE. Meanwhile, Vel_Invel changes to FALSE and Vel_Abt changes to TRUE and then the servo motor starts to decelerate.
- ❖ When the axis velocity is decreased to 0, Hlt_Done changes to TRUE and meanwhile, Hlt_Bsy and Hlt_Act change to FALSE.
- ❖ As Hlt_Ex changes to FALSE, Hlt_Done changes to FALSE.

11.3.5 MC_Stop

FB/FC	Explanation	Applicable model
FB	MC_Stop is used to make the axis decrease its speed at a given deceleration rate till it stops and then the axis goes into the Stopping state.	DVP15MC11T

MC_Stop_instance



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
Deceleration	Specify the target deceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of the target acceleration or deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE

Note:

1. MC_Stop instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction changes from TRUE to FALSE in the course of the instruction execution.
2. While *Execute* changes from FALSE to TRUE once more in the course of execution of MC_Halt, there is no impact on the instruction execution and the instruction will continue being executed in the previous way. When *Execute* changes from FALSE to TRUE once again after the instruction execution is completed, the instruction can be re-executed.
3. Refer to section 10.2 for the relation between *Deceleration and Jerk*.

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE/FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE/FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE/FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE/FALSE
Error	TRUE when there is an error.	BOOL	TRUE/FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

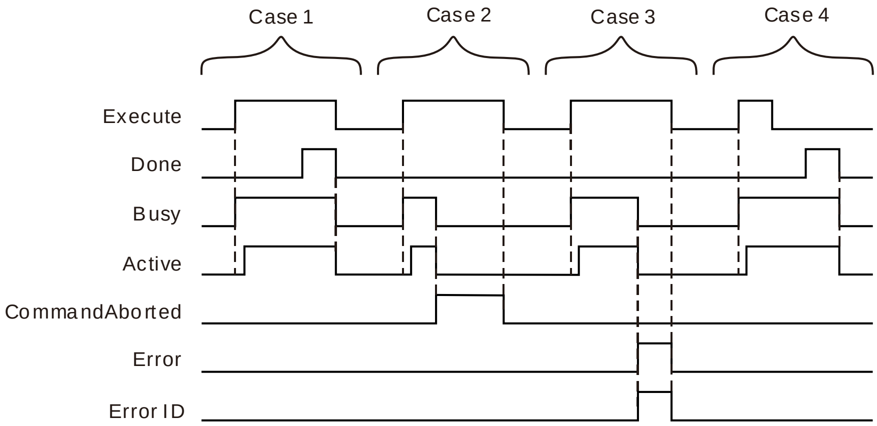
11

- **Output Update Timing**

Parameter name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the deceleration ends and the axis speed is decreased to 0.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE. ◆ When *Done* changes from TRUE to FALSE.
Active	◆ When the instruction starts to control the axis.	◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE. ◆ When *Done* changes from TRUE to FALSE.
CommandAborted	◆ When the instruction execution is aborted by another MC_Stop.	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* changes to TRUE when the instruction is aborted by another MC_Stop after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

11

- **Output Update Timing Chart**



Case 1 : When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one period later, *Active* changes to TRUE. When the deceleration ends and the axis speed is decreased to 0, *Done* changes to TRUE and *Busy* and *Active* remain TRUE.

Case 2 : When the MC_Stop instruction is aborted by another MC_Stop instruction after *Execute* changes from FALSE to TRUE, *CommandAborted* changes to TRUE and meanwhile *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3 : When an error occurs such as axis alarm or Offline after *Execute* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows the corresponding error code. And Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4 : In the course of execution of the instruction, *Done* changes to TRUE and *Busy* and *Active* remain TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE. One period later, *Done, Busy* and *Active* all change to FALSE.

- **Function**

MC_Stop is used to make the axis decrease its speed at a given deceleration rate till it stops.

- As long as *Execute* is TRUE after execution of MC_Stop is completed and the axis velocity is decreased to 0, the axis state will be in the Stopping state all the time. And during that period, other motion instruction can not be executed.
- If there are two MC_Stop instructions in the program for controlling the same axis, the previously being executed MC_Stop will be aborted by the later executed MC_Stop instruction.
- Compared to MC_Halt instruction, MC_Stop instruction will make the axis locked and thus the controller cannot perform other motion instruction excluding MC_Stop during MC_Stop execution. The controller still cannot perform other motion instructions when the execution of MC_Stop is finished and the axis has stopped. Other motion instruction can not be executed until *Execute* of MC_Stop changes from TRUE to FALSE.

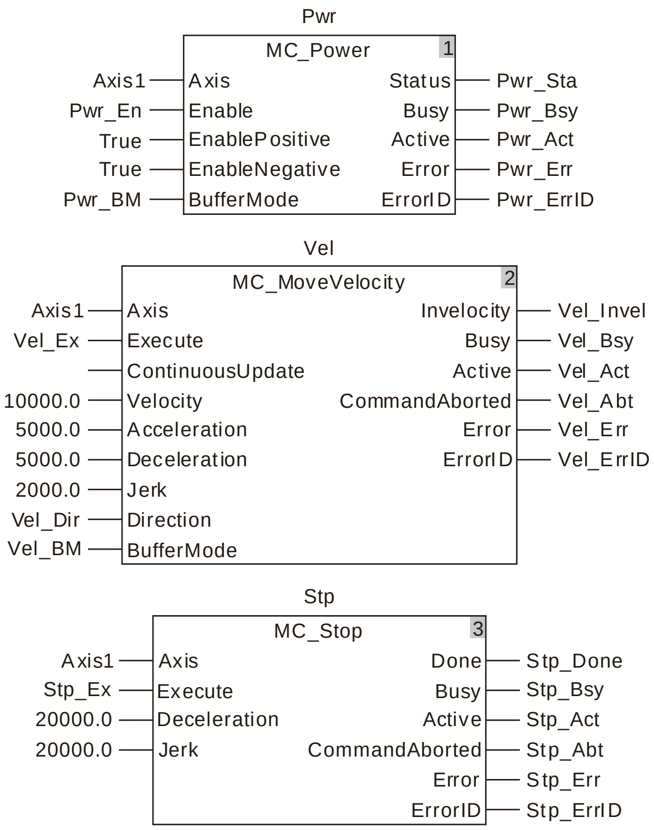
Programming Example 1

The example of MC_Stop execution is shown as below.

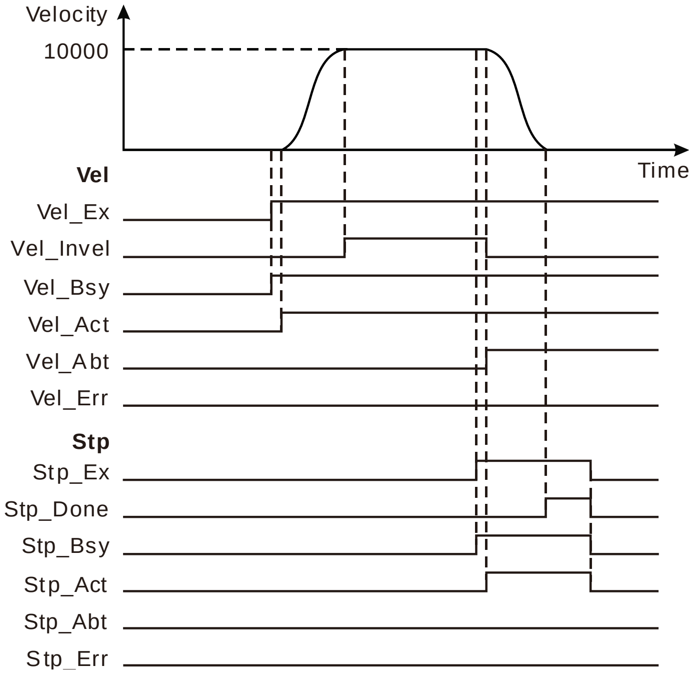
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0

Variable name	Data type	Initial value
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel_Ex	BOOL	FALSE
Vel_Dir	MC_DIRECTION	1
Vel_BM	MC_Buffer_Mode	0
Vel_Invel	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	
Stp	MC_Stop	
Stp_Ex	BOOL	FALSE
Stp_Done	BOOL	
Stp_Bsy	BOOL	
Stp_Act	BOOL	
Stp_Abt	BOOL	
Stp_Err	BOOL	
Stp_ErrID	WORD	



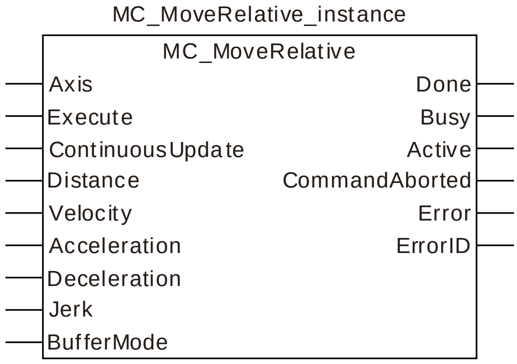
2. Motion Curve and Timing Charts:



- ❖ As Vel_Ex changes to TRUE, Vel_Bsy changes to TRUE. One period later, Vel_Act changes to TRUE and the servo motor starts to move forward. Vel_Invel changes to TRUE when the servo motor reaches the target velocity.
- ❖ As Stp_Ex changes to TRUE, Stp_Bsy changes to TRUE. One period later, Stp_Act changes to TRUE, meanwhile Vel_Invel changes to FALSE, Vel_Abt changes to TRUE and the servo motor starts to decelerate.
- ❖ When the axis velocity is decreased to 0, Stp_Done changes to TRUE and meanwhile Stp_Bsy, Stp_Act remain TRUE.
- ❖ As Stp_Ex changes to FALSE, Stp_Done, Stp_Bsy and Stp_Act change to FALSE simultaneously.

11.3.6 MC_MoveRelative

FB/FC	Explanation	Applicable model
FB	MC_MoveRelative is used to make the axis move a given distance by starting from the command current position at a given speed, acceleration and deceleration and Jerk.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved	-	-	-
Distance	Specify the motion distance from command current position. (Unit: Unit)	LREAL	Negative number, positive number or 0 (0)	When *Execute* changes from FALSE to TRUE
Velocity	Specify the target velocity. (Unit: Unit/second)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Acceleration	Specify the target acceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Deceleration	Specify the target deceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of the target acceleration or deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
BufferMode	Specify the behavior when executing two instructions. 0: Aborting 1: Buffered 2: BlendingLow 3: BlendingPrevious 4: BlendingNext	MC_Buffer_Mode	0 : mcAborting 1 : mcBuffered 2 : mcBlendingLow 3 : mcBlending_Previous	When *Execute* changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
	5: BlendingHigh		4 : mcBlending_Next 5 : mcBlending_High (0)	

Notes:

1. MC_MoveRelative instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction changes from TRUE to FALSE in the course of execution.
2. While the instruction is being executed and *Execute* changes from FALSE to TRUE again, there will be no impact on the instruction execution and the instruction will continue being executed in the previous way. When *Execute* changes from FALSE to TRUE again after the instruction execution is completed, the instruction can be re-executed and started in the conventional way.
3. Refer to section 10.2 for the relation among *Velocity, Acceleration and Jerk*.
4. Refer to section 10.3 for details on *BufferMode*.

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled by the instruction.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction execution is aborted.	BOOL	TRUE / FALSE
Error	TRUE while there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to the section 12.2 for corresponding error codes.	WORD	

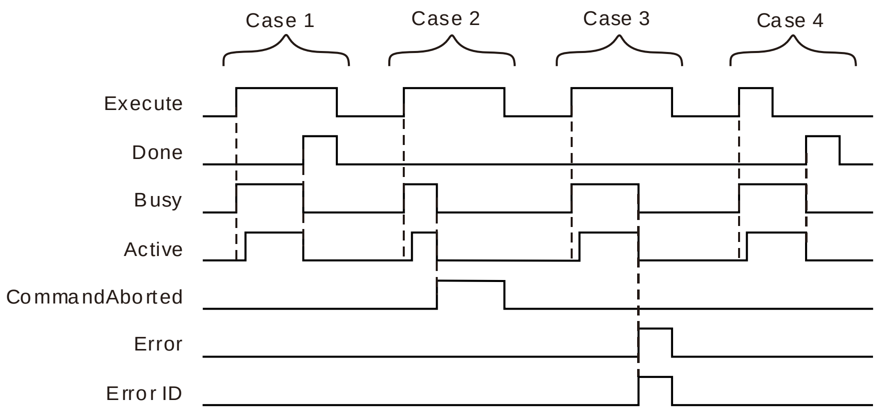
- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When positioning is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.

11

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* is set to TRUE when the instruction is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1：*Busy* changes to TRUE when *Execute* changes from FALSE to TRUE and one cycle later, *Active* changes to TRUE. When the positioning is finished, *Done* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE.

Case 2：When *Execute* changes from FALSE to TRUE and the instruction is aborted by other instruction, *CommandAborted* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE. *CommandAborted* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 3：When an error occurs such as axis alarm or Offline after *Execute* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows the corresponding error code. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4：In the course of execution of the instruction, *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE. Meanwhile, *Busy* and *Active* change to FALSE and one cycle later, *Done* changes to FALSE.

- **Function**

MC_MoveRelative is used to make the axis move for a given distance by starting from the command current axis position at a given speed, acceleration, deceleration and Jerk.

11

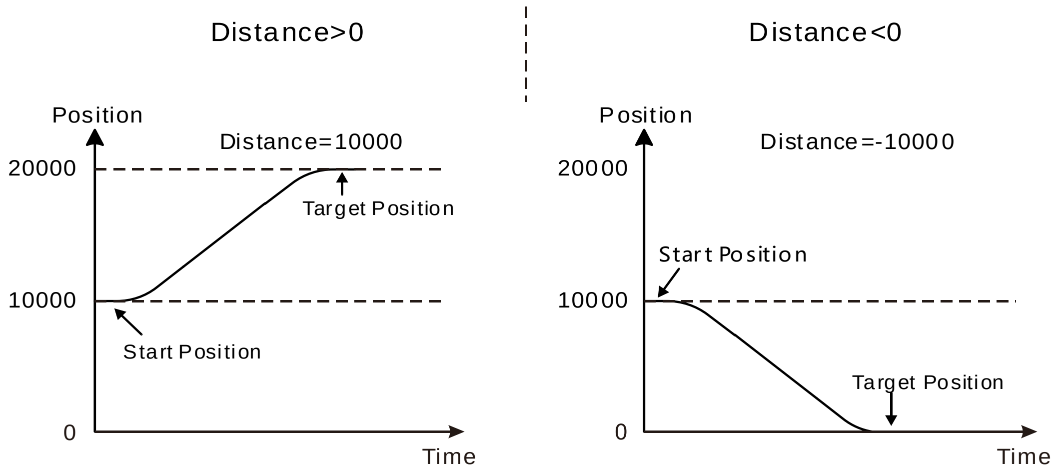
■ **Distance**

Distance and the start position for reference jointly determine the target position which the axis will reach under control of the instruction. The target position= the start position for reference + *Distance*.

When *Distance* is set to 0, the target position for the axis motion is set as current position. The instruction execution is finished in the next cycle since its execution and *Done* changes to TRUE.

As illustrated in the following left figure, the start position for reference is 10000. The axis moves in the positive direction and the target position is 20000 (10000+10000) when Distance>0 (10000).

In the following right figure, the axis moves in the negative direction and the target position is 0 (10000-10000) when Distance<0(-10000).

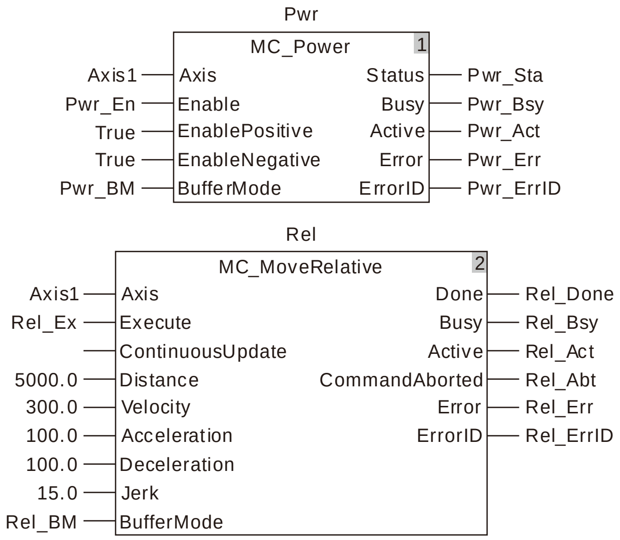


Programming Example 1

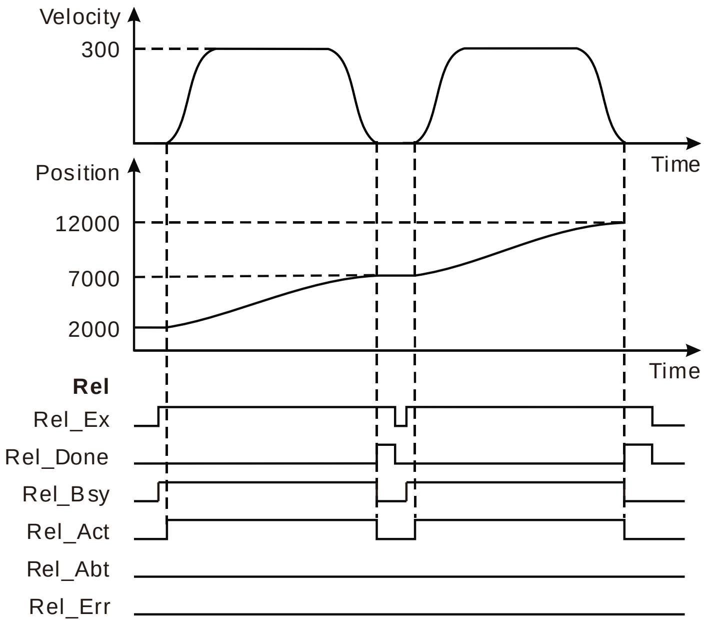
The programming example is as follows when one MC_MoveRelative instruction is used.

1. The variables and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel	MC_MoveRelative	
Rel _Ex	BOOL	FALSE
Rel _BM	MC_Buffer_Mode	0
Rel _Done	BOOL	
Rel _Bsy	BOOL	
Rel _Act	BOOL	
Rel _Abt	BOOL	
Rel _Err	BOOL	
Rel _ErrID	WORD	



2. **Motion Curve and Timing Chart**



❖ MC_MoveRelative instruction is executed for the first time when Rel_Ex changes from FALSE to TRUE for the first time. At the moment, the current position of the axis is 2000 and the target position is 7000 (7000=2000+5000).

❖ When the axis position of 7000 is reached, the instruction execution is finished and *Done* changes to TRUE.

❖ MC_MoveRelative instruction starts its second-time execution when Rel_Ex changes from FALSE to TRUE for the second time. At the moment, the current position of the axis is 7000 and the target position is 12000 (12000=7000+5000).

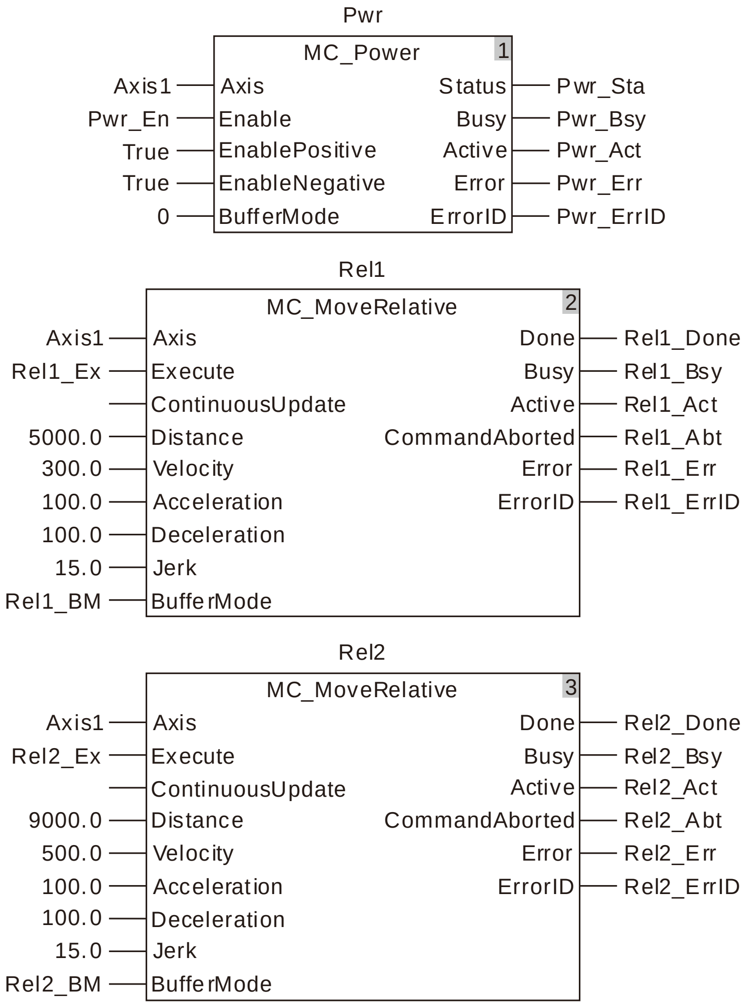
❖ When the axis position of 12000 is reached, the second-time execution of the instruction is completed and *Done* changes to TRUE for the second time.

Programming Example 2

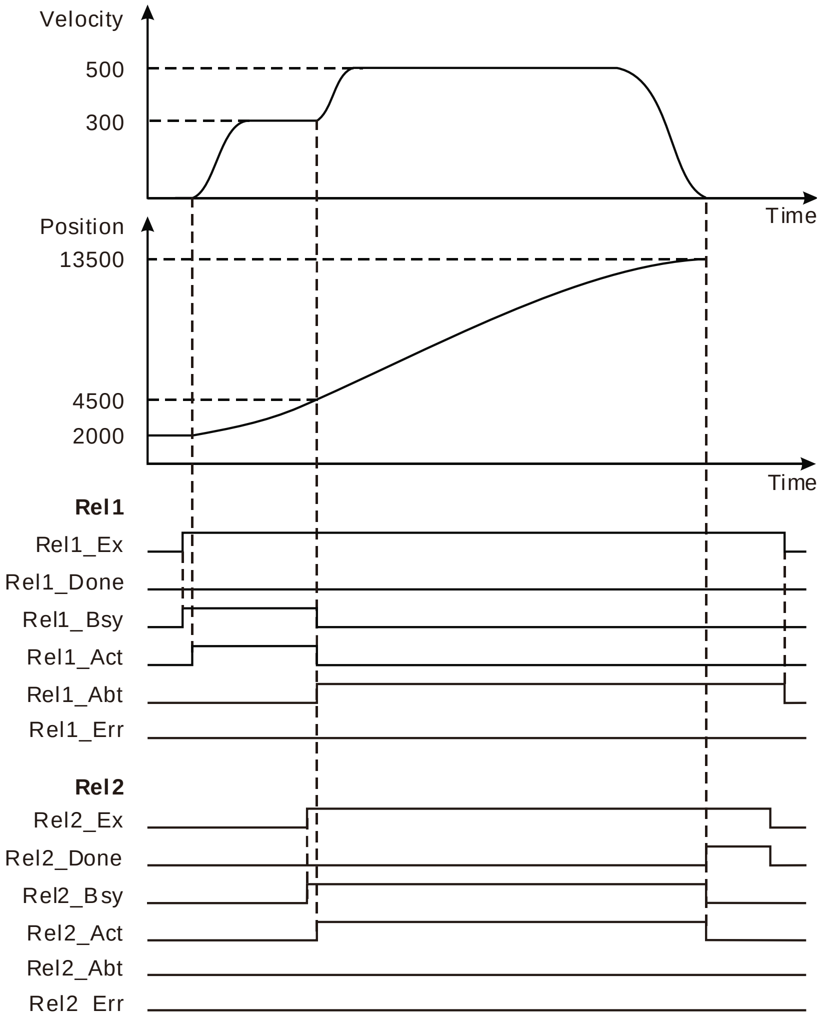
The example is shown below when MC_MoveRelative which is being executed is aborted.

1. The variables and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel1	MC_MoveRelative	
Rel1 _Ex	BOOL	FALSE
Rel1 _BM	MC_Buffer_Mode	0
Rel1 _Done	BOOL	
Rel1 _Bsy	BOOL	
Rel1 _Act	BOOL	
Rel1 _Abt	BOOL	
Rel1 _Err	BOOL	
Rel1 _ErrID	WORD	
Rel2	MC_MoveRelative	
Rel2 _Ex	BOOL	FALSE
Rel2 _BM	MC_Buffer_Mode	0
Rel2 _Done	BOOL	
Rel2 _Bsy	BOOL	
Rel2 _Act	BOOL	
Rel2 _Abt	BOOL	
Rel2 _Err	BOOL	
Rel2 _ErrID	WORD	


Pwr
MC_Power
1
Axis1
Axis
Status
Pwr_Sta
Pwr_En
Enable
Busy
Pwr_Bsy
True
EnablePositive
Active
Pwr_Act
True
EnableNegative
Error
Pwr_Err
0
BufferMode
ErrorID
Pwr_ErrID
Rel1
MC_MoveRelative
2
Axis1
Axis
Done
Rel1_Done
Rel1_Ex
Execute
Busy
Rel1_Bsy
ContinuousUpdate
Active
Rel1_Act
5000.0
Distance
CommandAborted
Rel1_Abt
300.0
Velocity
Error
Rel1_Err
100.0
Acceleration
ErrorID
Rel1_ErrID
100.0
Deceleration
15.0
Jerk
Rel1_BM
BufferMode
Rel2
MC_MoveRelative
3
Axis1
Axis
Done
Rel2_Done
Rel2_Ex
Execute
Busy
Rel2_Bsy
ContinuousUpdate
Active
Rel2_Act
9000.0
Distance
CommandAborted
Rel2_Abt
500.0
Velocity
Error
Rel2_Err
100.0
Acceleration
ErrorID
Rel2_ErrID
100.0
Deceleration
15.0
Jerk
Rel2_BM
BufferMode

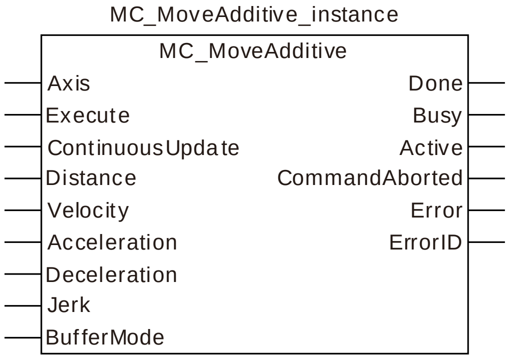
2. Motion Curve and Timing Chart



- ❖ The first MC_MoveRelative instruction starts being executed when Rel1_Ex changes from FALSE to TRUE. At the moment, the current position of the axis is 2000 and the target position is 7000 (7000=2000+5000).
- ❖ When the axis position of 4500 is reached, Rel2_Ex changes from FALSE to TRUE, the second MC_MoveRelative instruction starts being executed and the execution of the first MC_MoveRelative is aborted and Rel1_Abt changes to TRUE.
- ❖ When the axis position of 13500 (13500=4500+9000) is reached, the execution of the second MC_MoveRelative instruction is completed and Rel2_Done changes to TRUE.

11.3.7 MC_MoveAdditive

FB/FC	Explanation	Applicable model
FB	MC_MoveAdditive is used to make the axis move an additive distance at a given speed, acceleration and deceleration.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved	-	-	-
Distance	Specify the additive distance. (Unit: Unit)	LREAL	Negative number, positive number or 0 (0)	When *Execute* changes from FALSE to TRUE
Velocity	Specify the target velocity. (Unit: Unit/s)	LREAL	Positive number or 0 (0)	When *Execute* changes from FALSE to TRUE
Acceleration	Specify the target acceleration. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
Deceleration	Specify the target deceleration. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
Jerk	Specify the change rate of target acceleration and deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
BufferMode	Specify the behavior when executing two instructions. 0: Aborting 1: Buffered 2: BlendingLow 3: Blending Previous 4: BlendingNext 5: Blending High	MC_Buffer_Mode	0 : mcAborting 1 : mcBuffered 2 : mcBlendingLow 3 : mcBlending _Previous 4 : mcBlending _Next 5 : mcBlending _High (0)	When *Execute* changes from FALSE to TRUE

Notes:

1. MC_MoveAdditive instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction in the course of execution changes from TRUE to FALSE.
2. When *Execute* of the being executed instruction changes from FALSE to TRUE again, there is no impact on the instruction execution and the instruction will go on being executed in the previous way. When *Execute* changes from FALSE to TRUE again after the instruction execution is completed, the instruction can be re-executed and started in the conventional way.
3. Refer to section 10.2 for the relation among *Position, Velocity, Acceleration and Jerk*.
4. Refer to section 10.3 for details on *BufferMode.*

- **Output Parameters**

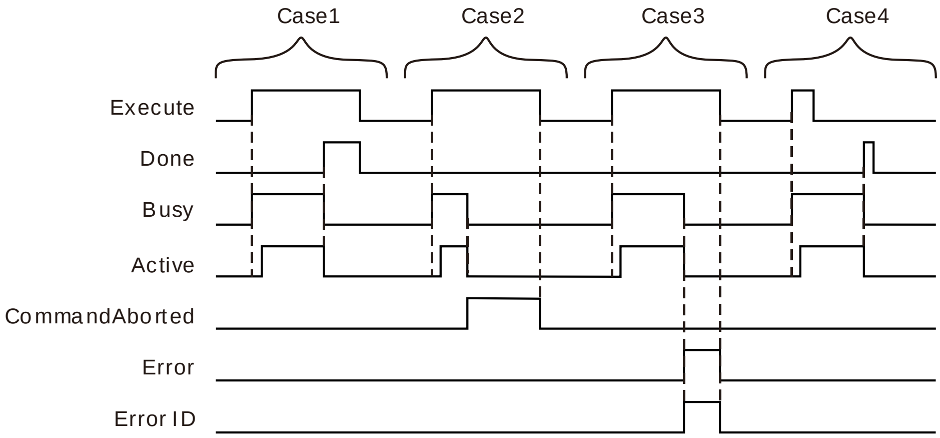
Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE while there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to the section 12.2.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When additive positioning is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is done. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts controlling the axis.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When the instruction execution is aborted by some other motion control instruction	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* is set to TRUE when the instruction execution is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

11

- **Output Update Timing Chart**



Case 1： When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one period later, *Active* changes to TRUE. When positioning is finished, *Done* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE.

Case 2： When *Execute* changes from FALSE to TRUE and the instruction execution is aborted by some other instruction, *CommandAborted* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3： When *Execute* changes from FALSE to TRUE and an error occurs such as axis alarm or Offline, *Error* changes to TRUE and *ErrorID* shows corresponding error codes. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4： In the course of execution of the instruction, *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE. Meanwhile, *Busy* and *Active* change to FALSE and one period later, *Done* changes to FALSE.

- **Function**

MC_MoveAdditive can control the actuator to move an additive distance at a given speed and acceleration.

The execution of the former instruction related with positioning has not been finished yet and the distance which the terminal actuator will move includes the uncompleted distance left by the former instruction and the given distance of this instruction when MC_MoveAdditive is executed. When the execution of MC_MoveAdditive is completed, the final position of the terminal actuator is the sum of the given distances of the former instruction and current instruction MC_MoveAdditive.
If the former instruction is a velocity instruction, MC_MoveAdditive will abort the execution of the velocity instruction and the terminal actuator will stop after moving a given distance of MC_MoveAdditive at a given speed, acceleration and deceleration.

If MC_MoveAdditive is executed while MC_MoveSuperimposed is individually executed, the instruction will abort MC_MoveSuperimposed immediately when the value of *BufferMode* of MC_MoveAdditive is 0. The distance which the terminal actuator will move includes the set distance of this instruction and the uncompleted distance left by MC_MoveSuperimposed while MC_MoveAdditive is executed.
An error will occur in the instruction right away if the value of *BufferMode* is in the range of 1~5 and the execution of MC_MoveSuperimposed instruction will continue.

If MC_MoveAdditive is executed when MC_MoveSuperimposed is used with a positioning instruction together, the instruction will abort MC_MoveSuperimposed and the positioning instruction when the value of *BufferMode* of MC_MoveAdditive is 0. The distance which the terminal actuator will move is the sum of the given distance of MC_MoveAdditive and the uncompleted distance left by the position instruction which is used with MC_MoveSuperimposed together, excluding the uncompleted distance left by MC_MoveSuperimposed while MC_MoveAdditive is executed. MC_MoveAdditive instruction will be executed after the execution of the positioning instruction which is used in conjunction with MC_MoveSuperimposed is completed if the value of *BufferMode* of MC_MoveAdditive is 1~5.

- **MC_MoveAdditive is started while MC_MoveSuperimposed is being executed.**

BufferMode of MC_MoveAdditive	Whether MC_MoveSuperimposed is being executed in conjunction with other position instruction	Description
0（Abort）	Yes	◆ The execution of MC_MoveSuperimposed and other position instruction will be aborted immediately. ◆ When MC_MoveAdditive is executed, the distance that the terminal actuator will travel is the set distance of MC_MoveAdditive plus the uncompleted distance left by MC_MoveSuperimposed plus the uncompleted distance left by the position instruction in conjunction with MC_MoveSuperimposed.
	No	◆ MC_MoveSuperimposed is aborted immediately. ◆ When MC_MoveAdditive is executed, the terminal actuator will travel the distance which is the sum of the uncompleted distance left by MC_MoveSuperimposed and the set distance of MC_MoveAdditive.
1~5（Buffered）	Yes	◆ MC_MoveSuperimposed will not be affected and keep being executed. ◆ After the execution of the position instruction in conjunction with MC_MoveSuperimposed ends, MC_MoveAdditive will start.
	No	◆ The execution of MC_MoveSuperimposed will continue. ◆ MC_MoveAdditive will report an error immediately.

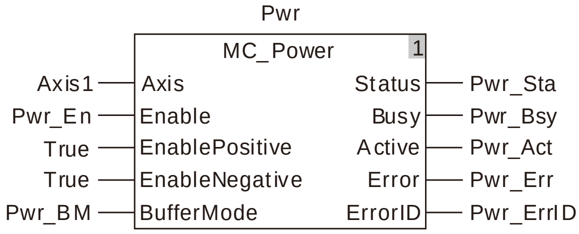
Programming Example 1

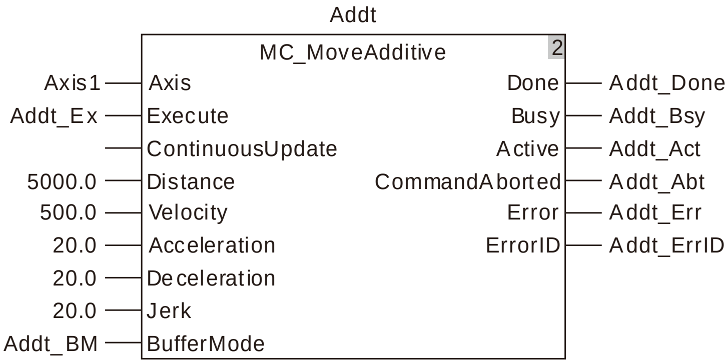
Below is an example of one single MC_MoveAbsolute instruction execution.

1. The variables and program

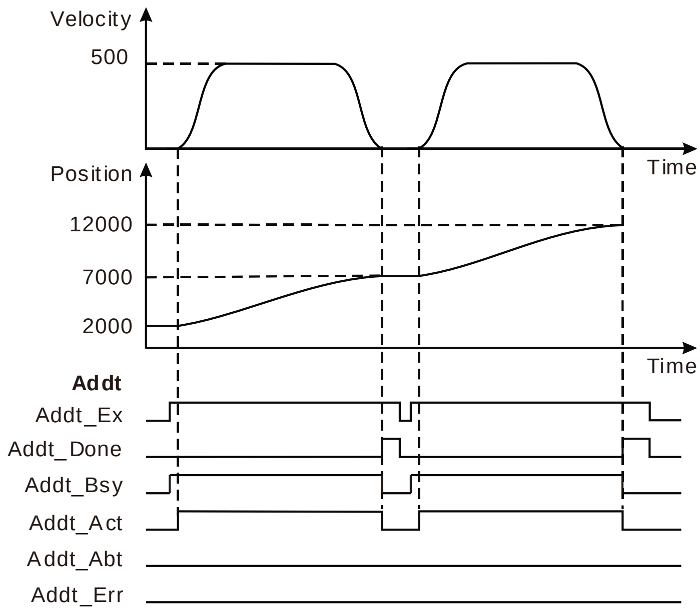
Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	

Variable name	Data type	Initial value
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Addt	MC_MoveAdditive	
Addt_Ex	BOOL	FALSE
Addt_BM	MC_Buffer_Mode	0
Addt_Done	BOOL	
Addt_Bsy	BOOL	
Addt_Act	BOOL	
Addt_Abt	BOOL	
Addt_Err	BOOL	
Addt_ErrID	WORD	





2. Motion Curve and Timing Charts:



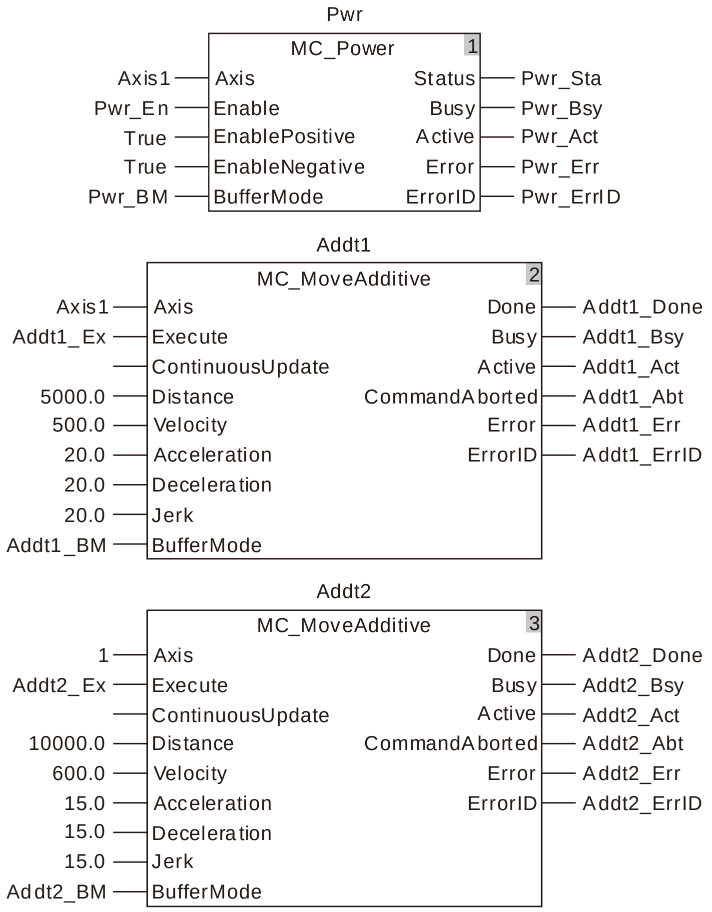
- ❖ When Addt_Ex changes from FALSE to TRUE, the motion controller controls the motion of the servo motor by taking current position as the reference point. Meanwhile, Addt_Bsy changes to TRUE and one period later, Addt_Act changes to TRUE. After the set distance is reached by the servo motor, Addt_Done changes from FALSE to TRUE and meanwhile Addt_Bsy and Addt_Act change from TRUE to FALSE.
- ❖ When Addt_Ex changes from TURE to FALSE, Addt_Done is reset.
- ❖ When Addt_Ex changes from FALSE to TRUE again after the servo motor reaches the set distance, the motion controller controls the motion of the servo motor and Addt_Done changes from FALSE to TRUE once again after the servo motor reaches the set distance.

Programming Example 2

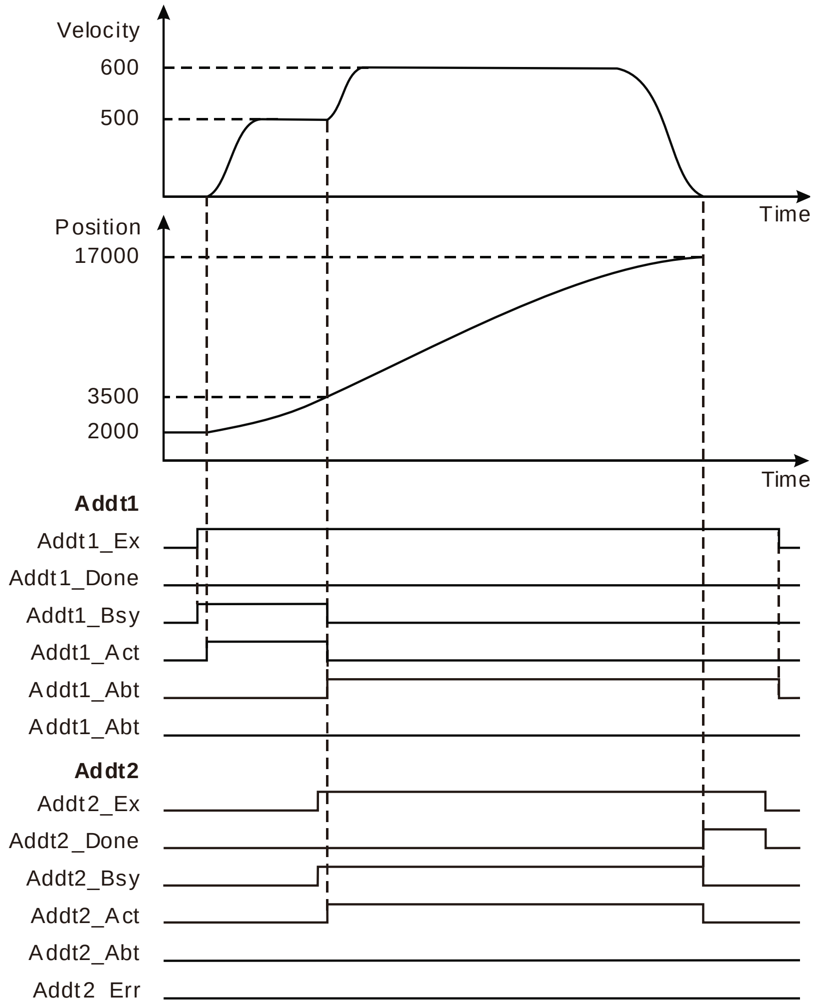
Below is an example on the execution of two MC_MoveAdditive instructions in the same task list.

1. The variables and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Addt1	MC_MoveAdditive	
Addt1_Ex	BOOL	FALSE
Addt1_BM	MC_Buffer_Mode	0
Addt1_Done	BOOL	
Addt1_Bsy	BOOL	
Addt1_Act	BOOL	
Addt1_Abt	BOOL	
Addt1_Err	BOOL	
Addt1_ErrID	WORD	
Addt2	MC_MoveAdditive	
Addt2_Ex	BOOL	FALSE
Addt2_BM	MC_Buffer_Mode	0
Addt2_Done	BOOL	
Addt2_Bsy	BOOL	
Addt2_Act	BOOL	
Addt2_Abt	BOOL	
Addt2_Err	BOOL	
Addt2_ErrID	WORD	


Pwr
MC_Power
1
Axis1
Axis
Status
Pwr_Sta
Pwr_En
Enable
Busy
Pwr_Bsy
True
EnablePositive
Active
Pwr_Act
True
EnableNegative
Error
Pwr_Err
Pwr_BM
BufferMode
ErrorID
Pwr_ErrID
Addt1
MC_MoveAdditive
2
Axis1
Axis
Done
Addt1_Done
Addt1_Ex
Execute
Busy
Addt1_Bsy
ContinuousUpdate
Active
Addt1_Act
5000.0
Distance
CommandAborted
Addt1_Abt
500.0
Velocity
Error
Addt1_Err
20.0
Acceleration
ErrorID
Addt1_ErrID
20.0
Deceleration
20.0
Jerk
Addt1_BM
BufferMode
Addt2
MC_MoveAdditive
3
1
Axis
Done
Addt2_Done
Addt2_Ex
Execute
Busy
Addt2_Bsy
ContinuousUpdate
Active
Addt2_Act
10000.0
Distance
CommandAborted
Addt2_Abt
600.0
Velocity
Error
Addt2_Err
15.0
Acceleration
ErrorID
Addt2_ErrID
15.0
Deceleration
15.0
Jerk
Addt2_BM
BufferMode

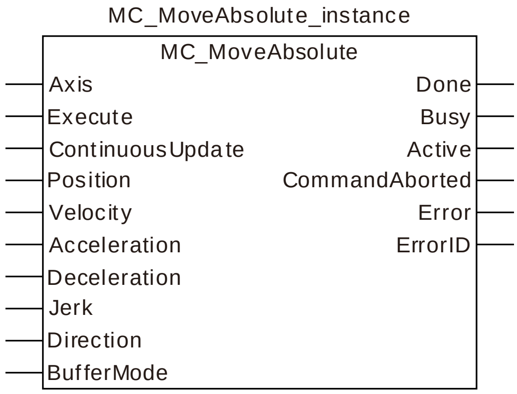
2. Motion Curve and Timing Charts:



- ❖ When Addt1_Ex changes from FALSE to TRUE, the motion controller controls the motion of the servo motor taking current position as the reference point. When Addt2_Ex changes from FALSE to TRUE, Addt2_Bsy changes from FALSE to TRUE and one period later, the first MC_MoveAdditive instruction is aborted and Addt1_Abt changes from FALSE to TRUE. Meanwhile, the servo motor moves according to the parameters of the second MC_MoveAdditive instruction. Addt2_Done changes from FALSE to TRUE when the servo motor completes the set distance which is the total sum of the two set distances of the two instructions.
- ❖ When Addt2_Ex changes from TRUE to FALSE, Addt2_Done is reset.

11.3.8 MC_MoveAbsolute

FB/FC	Explanation	FB/FC
FB	MC_MoveAbsolute is used to make the axis move to the specified absolute target position at the given speed, acceleration and deceleration.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved	-	-	-
Position	Specify the absolute target position. Rotary axis: 0≤ Position< Modulo Linear axis: No limit to Position. (Unit: Unit)	LREAL	Negative number, positive number or 0 (0)	When *Execute* changes from FALSE to TRUE
Velocity	Specify the target velocity. (Unit: Unit/s)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Acceleration	Specify the target acceleration. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Deceleration	Specify the target deceleration. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of target acceleration or deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Direction	Specify the rotation direction (which is valid only when the axis is the rotary axis).	MC_Direction	1: mcPositive-Direction, 2: mcShortestWay,	When *Execute* changes from FALSE to TRUE and the axis is in

Parameter name	Function	Data type	Valid range (Default)	Validation timing
	1: Positive direction 2: Shortest way 3: Negative direction 4: Current direction		3: mcNegative-Direction , 4: mcCurrent-Direction (1)	the mode of rotary axis
BufferMode	Specify the behavior when executing two instructions. 0 : McAborting 1 : McBuffered 2 : McBlendingLow 3 : McBlendingPrevious 4 : McBlending Next 5 : McBlendingHigh	MC_Buffer_ Mode	0 : mcAborting 1 : mcBuffered 2 : mcBlendingLow 3 : mcBlending _Previous 4 : mcBlending _Next 5 : mcBlending _High (0)	When Execute changes from FALSE to TRUE

Notes:

1. MC_MoveAbsolute instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction in the course of execution changes from TRUE to FALSE.
2. When *Execute* of the being executed instruction changes from FALSE to TRUE again, there is no impact on the instruction execution and the instruction will go on being executed in the previous way. When *Execute* changes from FALSE to TRUE again after the instruction execution is completed, the instruction can be re-executed.
3. When the axis is a rotary axis, Position can be the value within the range of 0~the value of modulo excluding the value of modulo. An error will occur in the instruction if the absolute value of Position is greater than or equal to the value of modulo. The value of Position is irrelevant to the value of modulo and it can be set to any constant if the axis is a linear axis.
4. *Direction* is valid only when the axis is the rotary axis. Refer to Direction in the following Function section for more details on *Direction*.
5. Refer to section 10.2 for the relation among *Position, Velocity, Acceleration and Jerk*.
6. Refer to section 10.3 for details on *BufferMode.*

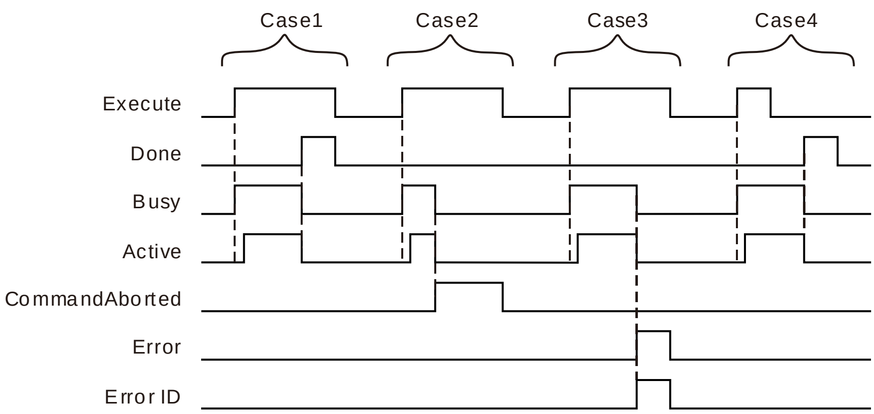
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled by the instruction.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction execution is aborted.	BOOL	TRUE / FALSE
Error	TRUE while there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for corresponding error codes.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When absolute positioning is completed	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is done. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts controlling the axis.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When the instruction execution is aborted by some other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* is set to TRUE when the instruction execution is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1: When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one period later, *Active* changes to TRUE. When positioning is completed, *Done* changes to TRUE and meanwhile, Busy and Active change to FALSE.

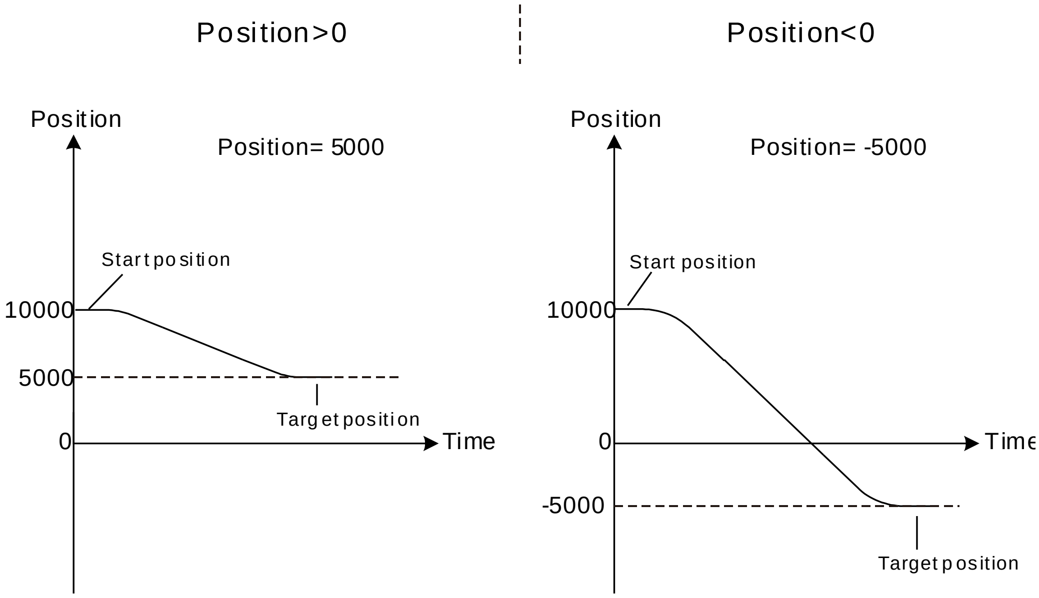
Case 2: When the instruction execution is aborted by some other motion instruction after *Execute* changes from FALSE to TRUE, *Abort* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3: When *Execute* changes from FALSE to TRUE and an error occurs such as axis alarm or Offline, *Error* changes to TRUE and *ErrorID* shows the corresponding error code. And Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4 : In the course of execution of the instruction, *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE. Meanwhile, *Busy* and *Active* change to FALSE and one period later, *Done* changes to FALSE.

- **Function**

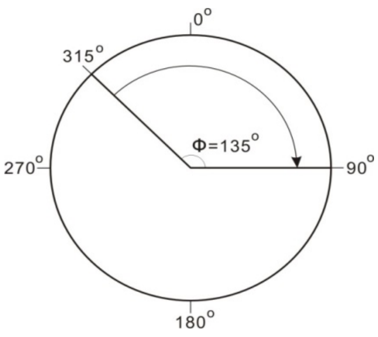
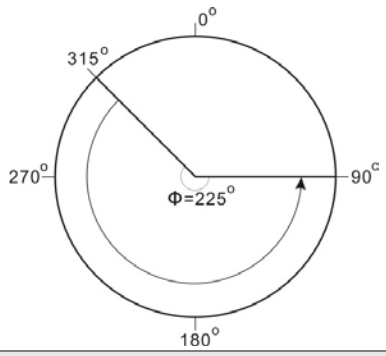
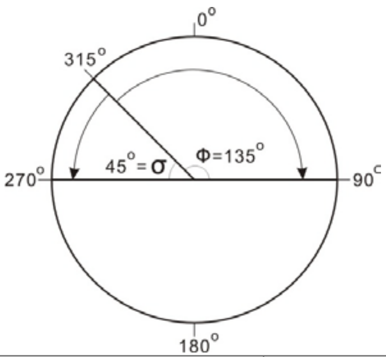
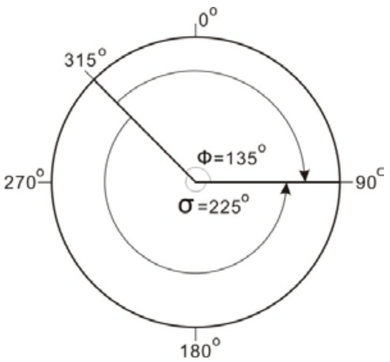
MC_MoveAbsolute is used to make the axis move to the specified absolute target position at the set speed, acceleration and deceleration.
The start axis position is 10000 when MC_MoveAbsolute instruction is executed. The axis will move reversely when *Position* >0 (5000). See the figure below when *Position* is 5000.
The axis will move reversely when *Position*<0 (-5000). See the figure below when *Position* is -5000.



Note: As long as MC_MoveAbsolute instruction which is being executed is aborted, its uncompleted distance will be discarded and the new instruction will be executed.

- **Direction**

Direction is valid when the axis is a rotary axis and different motion directions of the axis are listed in the following table based on different Direction value. (Modulo: 360)

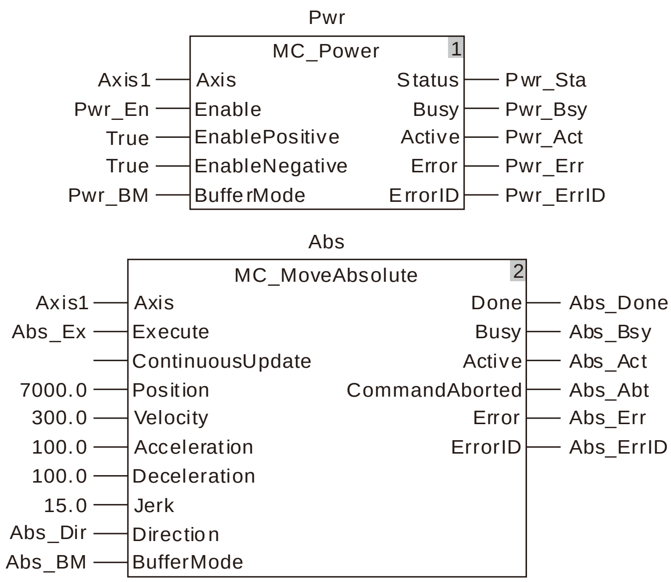
<table>
<tr><td>Direction: 1 (Positive direction)
Current position: 315°
Target position: 90°
Movement angle: 135°</td><td>Direction: 3 (Negative direction)
Current position: 315°
Target position: 90°
Movement angle: 225°</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Direction: 2 (Shortest way)
Current position: 315°
Target position: 90°
Movement angle: 135°</td><td>Direction: 2 (Shortest way)
Current position: 315°
Target position: 270°
Movement angle: 45°</td></tr>
<tr><td colspan="2">
</td></tr>
<tr><td>Direction: 4 (Current direction)
The status of the rotary axis before the instruction is executed: Moving in the negative direction.
Current position: 315°
Target position: 90°
Movement angle: 225°</td><td>Direction: 4 (Current direction)
The status of the rotary axis before the instruction is executed: motionless or moving in the positive direction.
Current position: 315°
Target position: 90°
Movement angle: 135°</td></tr>
<tr><td colspan="2">
</td></tr>
</table>

Programming Example 1

One MC_MoveAbsolute is executed as follows.

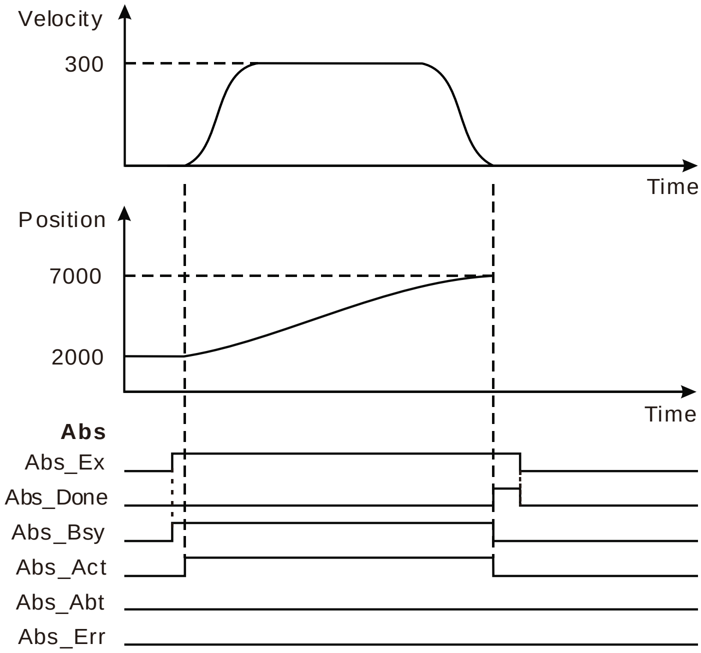
1. The variables and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Abs	MC_MoveAbsolute	
Abs_Ex	BOOL	FALSE
Abs_Dir	MC_DIRECTION	0
Abs_BM	MC_Buffer_Mode	0
Abs_Done	BOOL	
Abs_Bsy	BOOL	
Abs_Act	BOOL	
Abs_Abt	BOOL	
Abs_Err	BOOL	
Abs_ErrID	WORD	



11

2. Motion Curve and Timing Charts



❖ When Abs_Ex changes from FALSE to TRUE, MC_MoveAbsolute instruction starts being executed and at the moment, the current position of the axis is 2000 and target position is 7000.

❖ The execution of the instruction is completed when the axis reaches 7000.

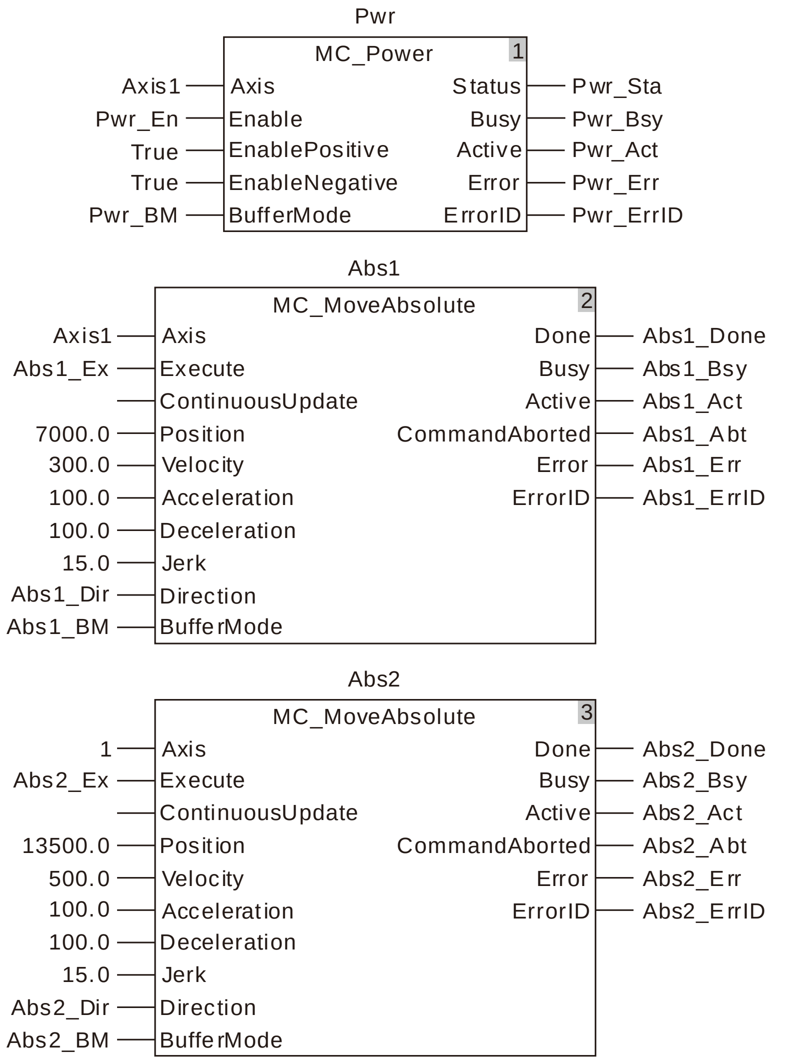
Programming Example 2

The example on how one MC_MoveAbsolute instruction aborts the execution of another MC_MoveAbsolute instruction is shown below.

1. The variables and program

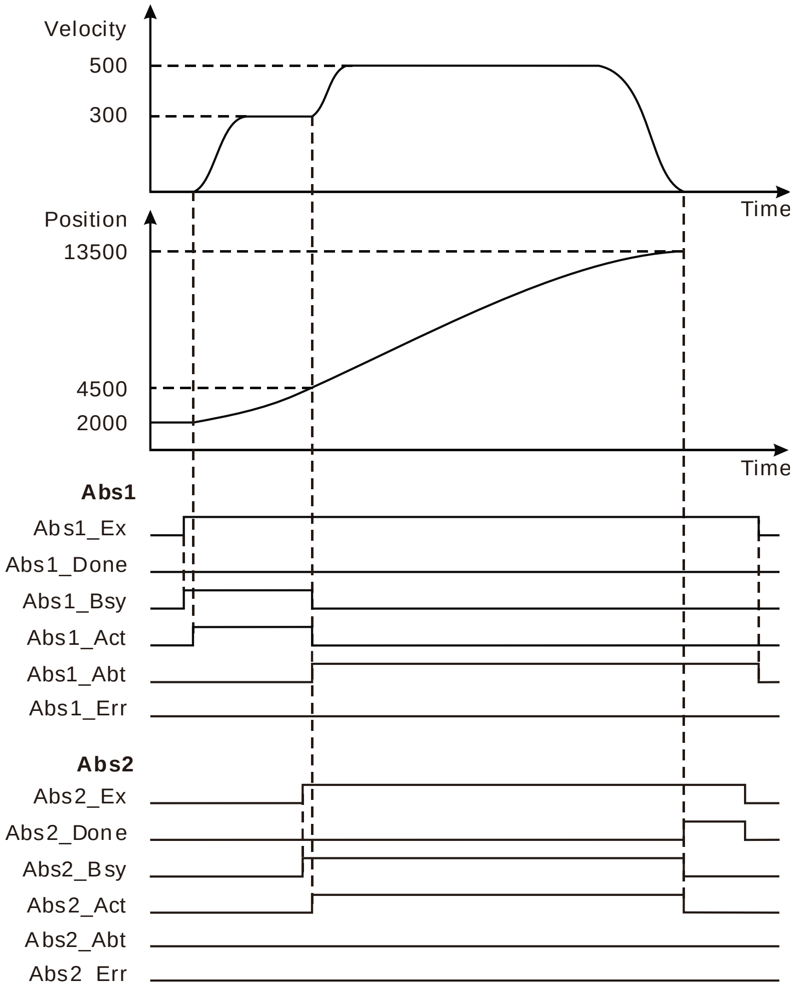
Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Abs1	MC_Move Absolute	
Abs1_Ex	BOOL	FALSE
Abs1_Dir	MC_DIRECTION	0
Abs1_BM	MC_Buffer_Mode	0
Abs1_Done	BOOL	
Abs1_Bsy	BOOL	
Abs1_Act	BOOL	
Abs1_Abt	BOOL	
Abs1_Err	BOOL	

Variable name	Data type	Initial value
Abs1_ErrID	WORD	
Abs2	MC_Move Absolute	
Abs2_Ex	BOOL	FALSE
Abs2_Dir	MC_DIRECTION	0
Abs2_BM	MC_Buffer_Mode	0
Abs2_Done	BOOL	
Abs2_Bsy	BOOL	
Abs2_Act	BOOL	
Abs2_Abt	BOOL	
Abs2_Err	BOOL	
Abs2_ErrID	WORD	



11

2. **Motion Curve and Timing Charts**



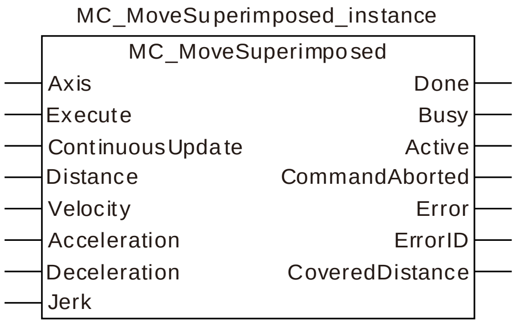
❖ When Abs1_Ex changes from FALSE to TRUE, the first MC_MoveAbsolute instruction starts being executed and at the moment, the current position of the axis is 2000 and target position is 7000.

❖ When the axis reaches 4500, Abs2_Ex changes from FALSE to TRUE; the second MC_MoveAbsolute instruction starts being executed and the first MC_MoveAbsolute instruction is aborted with its output parameter Abs1_Abt changing to TRUE.

❖ When the axis reaches 13500, the execution of the second MC_MoveAbsolute instruction is completed and its output parameter Abs2_Done changes to TRUE.

11.3.9 MC_MoveSuperimposed

FB/FC	Explanation	Applicable model
FB	MC_MoveSuperimposed controls the axis to superimpose the set distance on the current motion state according to the set velocity, acceleration and deceleration.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved	-	-	-
Distance	The distance to superimpose（Unit: Unit）	LREAL	Negative number, positive number and 0 (0)	When *Execute* changes from FALSE to TRUE
Velocity	Specify the target velocity. (Unit: Unit/second)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Acceleration	Specify the target acceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Deceleration	Specify the target deceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of the target acceleration or deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE

Notes:

1. MC_MoveSuperimposed instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction changes from TRUE to FALSE during execution of the instruction.
2. When *Execute* changes from FALSE to TRUE again during execution of the instruction, there is no impact on the instruction execution and the instruction will go on being executed in the previous way. When *Execute* changes from FALSE to TRUE again after the instruction execution is completed, the instruction can be re-executed.

3. Refer to section 10.2 for the relation among *Velocity, Acceleration, Deceleration and Jerk*.

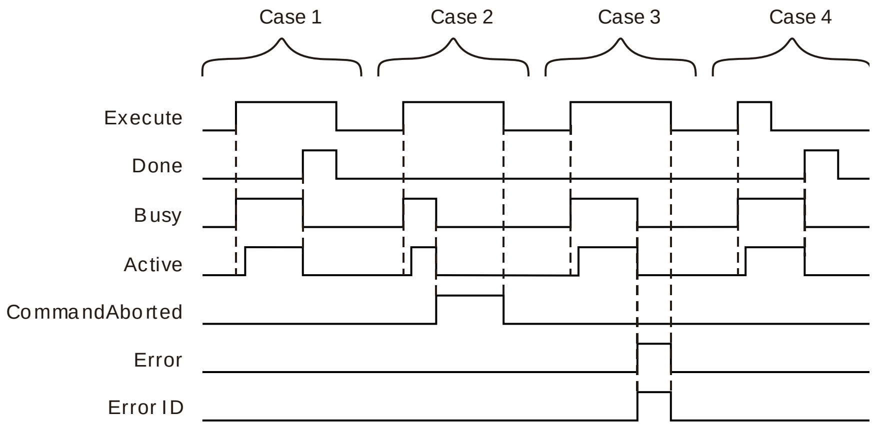
● **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE/FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	-
CoveredDistance	The totally superimposed distance since the instruction is started.	LREAL	Negative number, positive number and 0

● **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the superimposed positioning is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1： When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE. One cycle later, *Active* changes to TRUE. When the instruction execution is completed, *Done* changes to TRUE and *Busy* and *Active* change to FALSE.

Case 2： When *Execute* changes to TRUE and the instruction is aborted by other instruction, *CommandAborted* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE. *CommandAborted* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 3： When an error occurs such as disabled axis as *Execute* is TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error code. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE and the value of *ErrorID* is cleared to 0 when *Execute* changes from TRUE to FALSE.

Case 4： *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during execution of the instruction. Meanwhile, *Busy* and *Active* change to FALSE and one cycle later, *Done* changes to FALSE.

- **Function**

The MC_MoveSuperimposed instruction controls the axis to independently superimpose the set distance on the current motion state according to the set velocity, acceleration and deceleration.

1. When MC_MoveSuperimposed instruction is executed, the execution of the previous instruction excluding MC_MoveSuperimposed and MC_HaltSuperimposed instructions is not aborted. If the two instructions are executed simultaneously, their distances, velocities, accelerations and decelerations will be respectively added up in real time. When the set velocity of either of the instructions is reached, the acceleration of the instruction will be 0. If the previous instruction execution is finished, the velocities, accelerations and decelerations will not be added up any more and MC_MoveSuperimposed instruction continues running independently.
2. If MC_MoveSuperimposed instruction is executed when the axis is in Standstill state, the execution effect of MC_MoveSuperimposed instruction is equivalent to that of MC_MoveRelative instruction.
3. Execute another motion instruction excluding MC_MoveSuperimposed and MC_HaltSuperimposed instructions when MC_MoveSuperimposed instruction and one motion instruction jointly control the axis. If the *Buffermode* value of the lately executed motion instruction is 0, both of the MC_MoveSuperimposed instruction and the previously executed motion instruction will be aborted. If the *Buffermode* value of the lately executed motion instruction is another number except 0, the MC_MoveSuperimposed instruction and the previously executed motion instruction will not be aborted.
4. If another MC_MoveSuperimposed instruction is executed when one MC_MoveSuperimposed instruction and another motion instruction jointly control the axis, the previous MC_MoveSuperimposed instruction will be aborted but other motion instruction will not be affected.

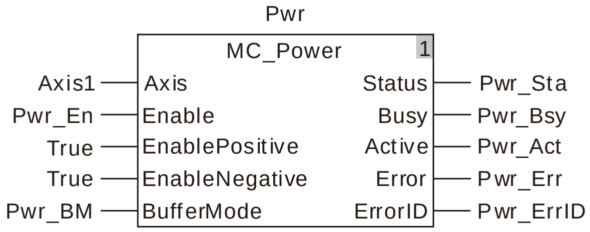
5. If another MC_MoveSuperimposed instruction is executed when one MC_MoveSuperimposed instruction controls the axis independently, the previous MC_MoveSuperimposed instruction will be aborted.
6. If the MC_HaltSuperimposed instruction is executed in the course of execution of MC_MoveSuperimposed instruction, the MC_MoveSuperimposed instruction will be aborted.
7. MC_MoveSuperimposed can be executed on the slave axis specified by MC_GearIn instruction and MC_ CamIn instruction.

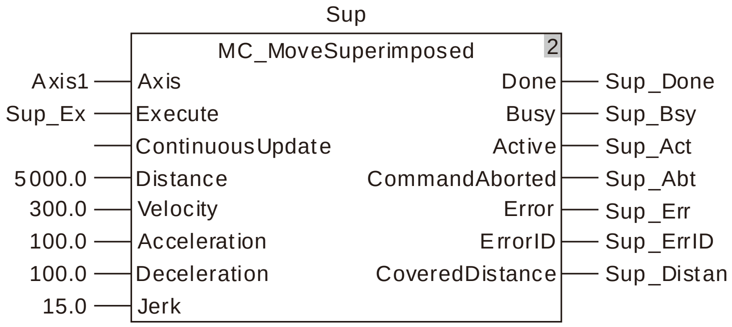
Programming Example 1

The programming example is as follows when one MC_MoveSuperimposed instruction is used.

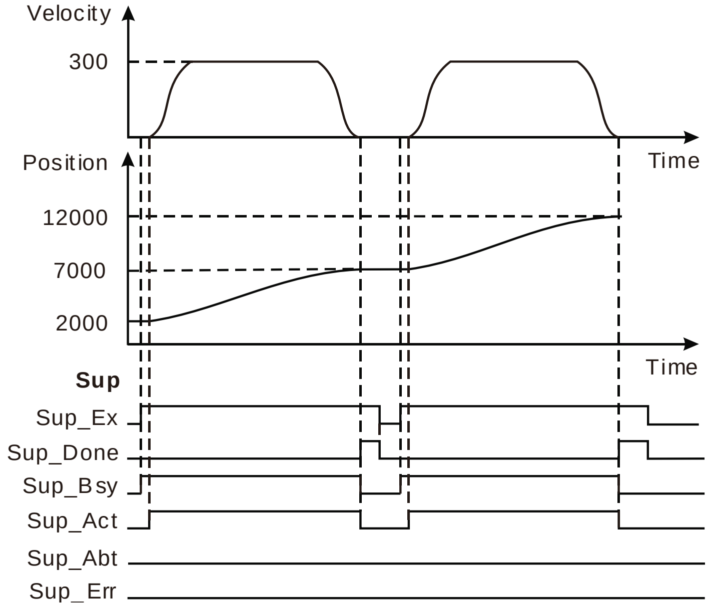
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Sup	MC_MoveSuperimposed	
Sup_Ex	BOOL	FALSE
Sup_Done	BOOL	
Sup_Bsy	BOOL	
Sup_Act	BOOL	
Sup_Abt	BOOL	
Sup_Err	BOOL	
Sup_ErrID	WORD	
Sup_Distan	LREAL	





2. Motion Curve and Timing Chart:



- When Sup_Ex changes to TRUE, Sup_Bsy changes to TRUE. One cycle later, Sup_Act changes to TRUE and the motion controller controls the servo motor to run by using current position as the reference point.
- After the servo motor completes the superimposed positioning, Sup_Done changes to TRUE and meanwhile Sup_Bsy and Sup_Act change to FALSE.
- When Sup_Ex changes to FALSE, Sup_Done changes to FALSE.
- When Sup_Ex changes to TRUE again after the servo motor completes the set distance, the motion controller controls the servo motor to run. When the servo motor completes the set distance, Sup_Done changes to TRUE again.

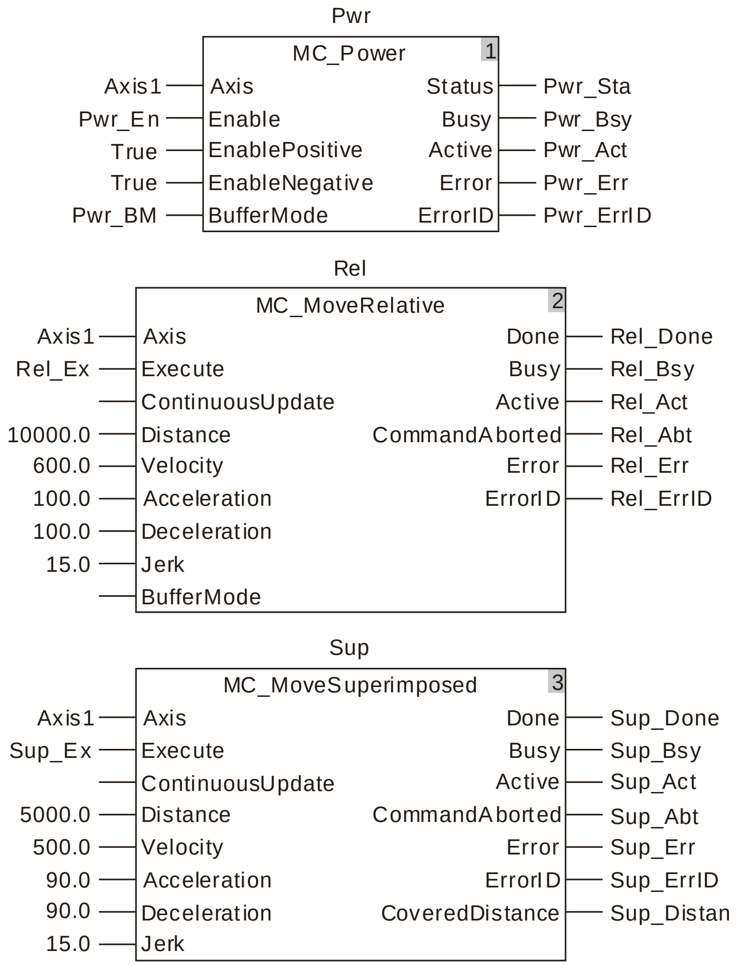
Programming Example 2

Below is the example that MC_MoveSuperimposed and MC_MoveRelative instructions are matched.

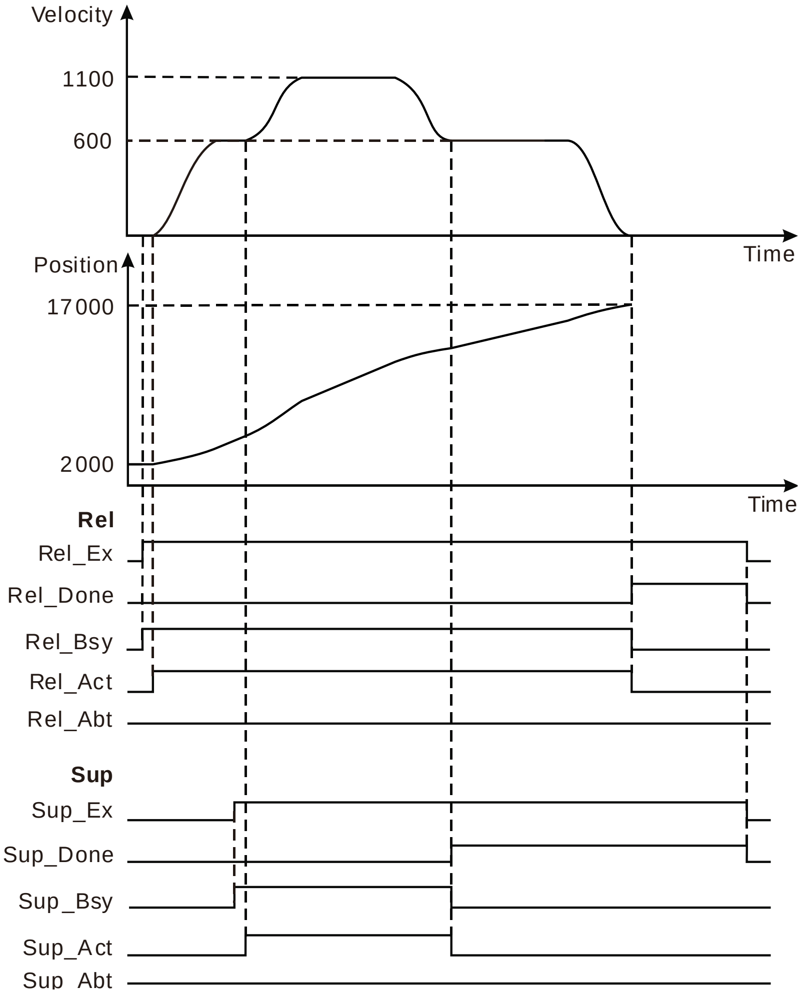
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel	MC_MoveRelative	
Rel_Ex	BOOL	FALSE
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	

Variable name	Data type	Initial value
Sup	MC_MoveSuperimposed	
Sup_Ex	BOOL	FALSE
Sup_Done	BOOL	
Sup_Bsy	BOOL	
Sup_Act	BOOL	
Sup_Abt	BOOL	
Sup_Err	BOOL	
Sup_ErrID	WORD	
Sup_Distan	LREAL	



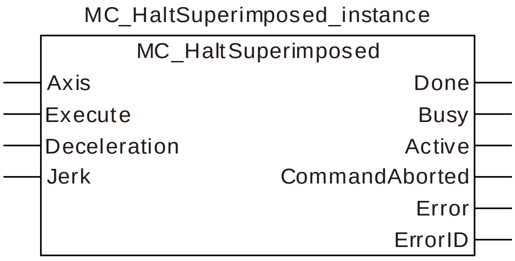
2. Motion Curve and Timing Chart:



- ❖ When Rel_Ex changes to TRUE, Rel_Bsy changes to TRUE. One period later, Rel_Act changes to TRUE and the motion controller controls the servo motor rotation by using the current position as the reference point.
- ❖ When Sup_Ex changes to TRUE, Sup_Bsy changes to TRUE. One cycle later, Sup_Act changes to TRUE and the the MC_MoveSuperimposed instruction starts to control the axis. The velocity and acceleration (0 at the moment) for the servo motor are the sums of the velocities and accelerations of the two instructions respectively.
- ❖ When the superimposed distance specified by the MC_MoveSuperimposed instruction is completed, Sup_Done changes to TRUE and Sup_Bsy and Sup_Act change to FALSE.
- ❖ When the distance specified by the MC_MoveRelative instruction is completed, Rel_Done changes to TRUE and Rel_Bsy and Rel_Act change to FALSE. The final position of the axis is the sum of the distances of the two instructions plus the start position.
- ❖ When Rel_Ex changes to FALSE, Rel_Done changes to FALSE. When Sup_Ex changes to FALSE, Sup_Done changes to FALSE.

11.3.10 MC_HaltSuperimposed

FB/FC	Explanation	Applicable model
FB	MC_HaltSuperimposed halts the execution of the MC_MoveSuperimposed instruction.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
Deceleration	Specify the target deceleration rate. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
Jerk	Specify the change rate of the target deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE

Notes:

1. MC_HaltSuperimposed instruction is executed when *Execute* changes from FALSE to TRUE. There is no impact on the instruction execution when *Execute* of the instruction changes from TRUE to FALSE during execution of the instruction.
2. Refer to section 10.2 for the relation between *Deceleration* and *Jerk*.

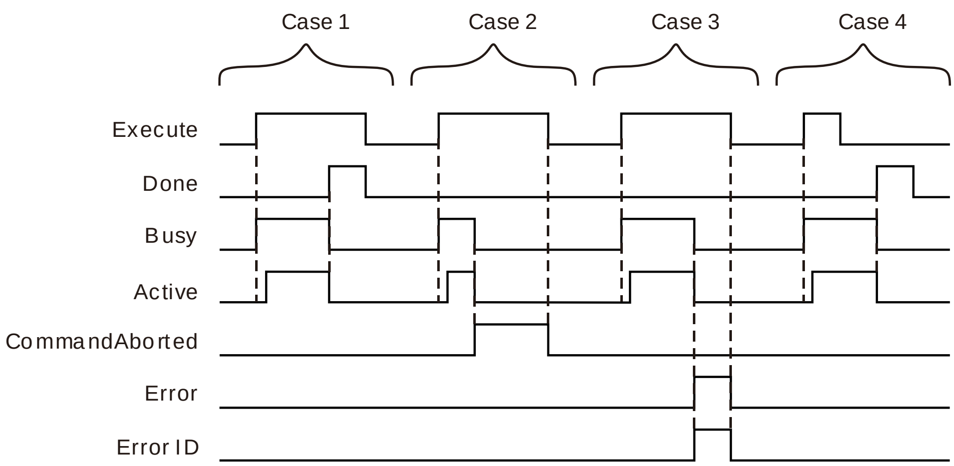
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE/FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	-

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the instruction execution is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1 : When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE. One cycle later, *Active* changes to TRUE. When the instruction execution is completed, *Done* changes to TRUE and *Busy* and *Active* change to FALSE.

Case 2： When *Execute* changes to TRUE and the instruction is aborted by other instruction, *CommandAborted* changes to TRUE and meanwhile, *Busy* and *Active* change to FALSE. *CommandAborted* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 3： When an error occurs such as axis disabled as *Execute* is TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error code. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4： *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE in the course of execution of the instruction. Meanwhile, *Busy* and *Active* change to FALSE and one cycle later, *Done* changes to FALSE.

- **Function**

The MC_HaltSuperimposed instruction is used to halt the execution of the MC_MoveSuperimposed instruction.

- The MC_HaltSuperimposed instruction cannot be executed alone and it can only be used with the MC_MoveSuperimposed instruction together.
- If the MC_HaltSuperimposed instruction is executed when the MC_MoveSuperimposed instruction and other motion instruction jointly control the axis, the MC_HaltSuperimposed instruction will abort the MC_MoveSuperimposed instruction but other motion instruction execution will not be affected.
- The MC_HaltSuperimposed instruction can halt the execution of the MC_HaltSuperimposed instruction.

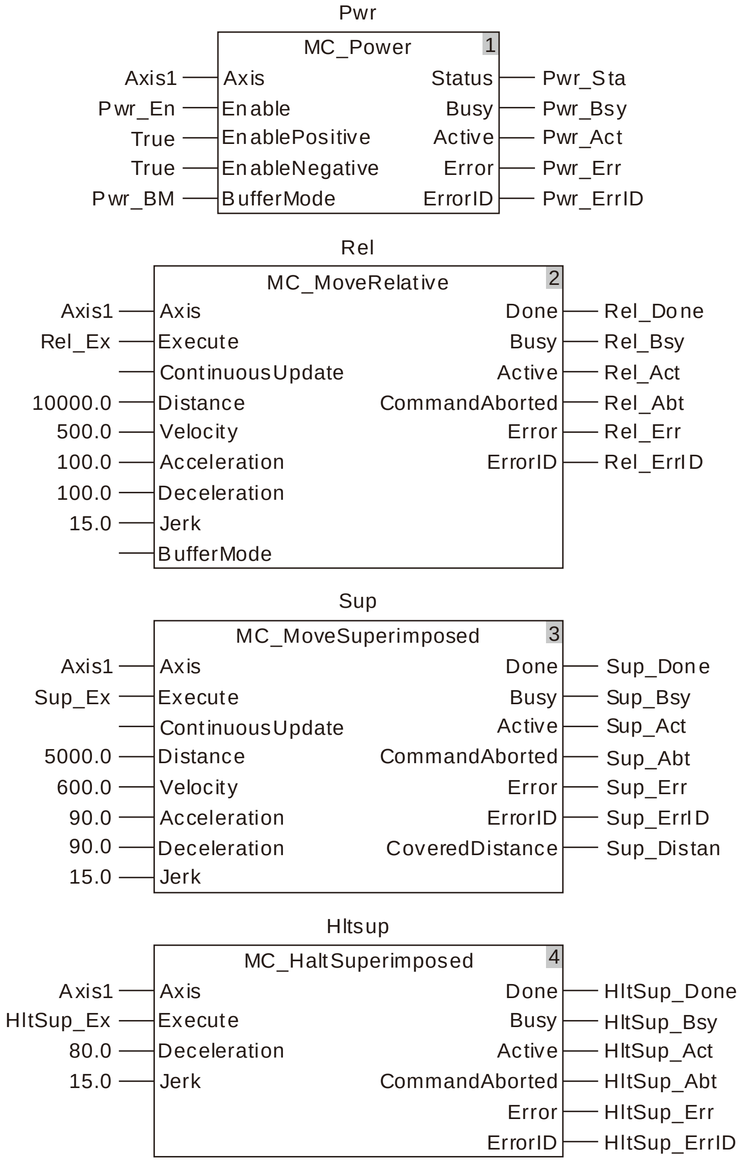
Programming Example

The programming example is as follows when one MC_HaltSuperimposed instruction is used.

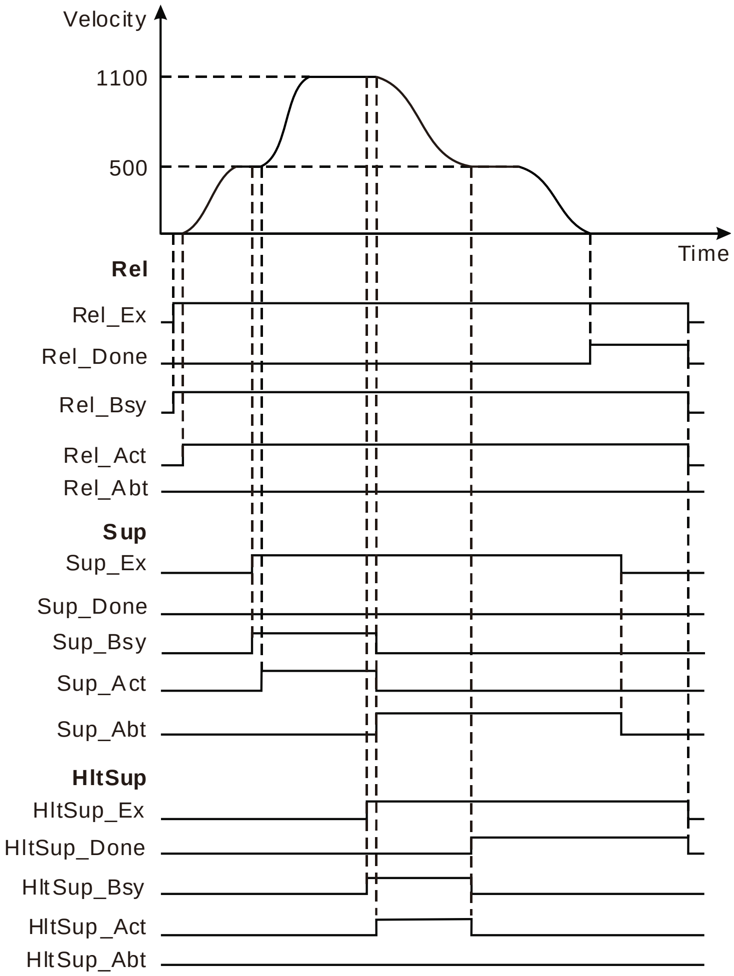
1. **The variable table and program**

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel	MC_MoveRelative	
Rel_Ex	BOOL	FALSE
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	
Sup	MC_MoveSuperimposed	
Sup_Ex	BOOL	FALSE
Sup_Done	BOOL	
Sup_Bsy	BOOL	
Sup_Act	BOOL	
Sup_Abt	BOOL	
Sup_Err	BOOL	

Variable name	Data type	Initial value
Sup_ErrID	WORD	
Sup_Distan	LREAL	
HltSup	MC_HaltSuperimposed	
HltSup_Ex	BOOL	FALSE
HltSup_Done	BOOL	
HltSup_Bsy	BOOL	
HltSup_Act	BOOL	
HltSup_Abt	BOOL	
HltSup_Err	BOOL	
HltSup_ErrID	WORD	



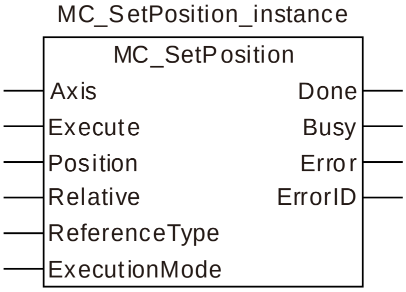
2. Motion Curve and Timing Chart



- When Rel_Ex changes to TRUE, Rel_Bsy changes to TRUE. One cycle later, Rel_Act changes to TRUE and the motion controller controls the servo motor rotation by using the current position as the reference point. When Sup_Ex changes to TRUE, Sup_Bsy changes to TRUE. One cycle later, Sup_Act changes to TRUE, the execution of the MC_MoveSuperimposed instruction starts and the velocities and accelerations (0 at the moment) for the servo motor will be added up respectively.
- When Hltsup_Ex changes to TRUE, Hltsup_Bsy changes to TRUE. One cycle later, Hltsup_Act changes to TRUE, the execution of the MC_HaltSuperimposed instruction starts, the MC_MoveSuperimposed instruction is aborted and Sup_Bsy and Sup_Act change to FALSE and meanwhile, Sup_Abt changes to TRUE. The execution of the MC_MoveSuperimposed instruction is halted by the MC_HaltSuperimposed instruction.
- When Hltsup_Done changes to TRUE, Hltsup_Bsy and Hltsup_Act change to FALSE.
- The execution of the MC_HaltSuperimposed instruction has no impact on the being executed MC_MoveRelative instruction.

11.3.11 MC_SetPosition

FB/FC	Explanation	Applicable model
FB	MC_SetPosition is used to set the position of the axis to a given value and no actual axis motion is brought accordingly.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
Position	Specify the target Position. (Unit: Unit)	LREAL	Negative number, positive number or 0 (0)	When *Execute* changes from FALSE to TRUE
Relative	Specify the relative mode or absolute mode for the target position and current position.	BOOL	TRUE or FALSE (FALSE)	When *Execute* changes from FALSE to TRUE
ReferenceType	Specify the position type for reference.	MC_ ReferenceType	0: mcCommand Position 1: mcActual Position (0)	When *Execute* changes from FALSE to TRUE
ExecutionMode	Reserved			

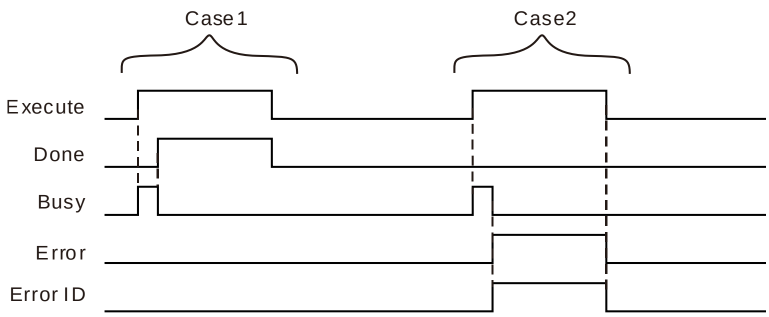
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction is completed.	BOOL	TRUE / FALSE
Busy	TRUE while the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When positioning is completed	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is finished. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal □.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**

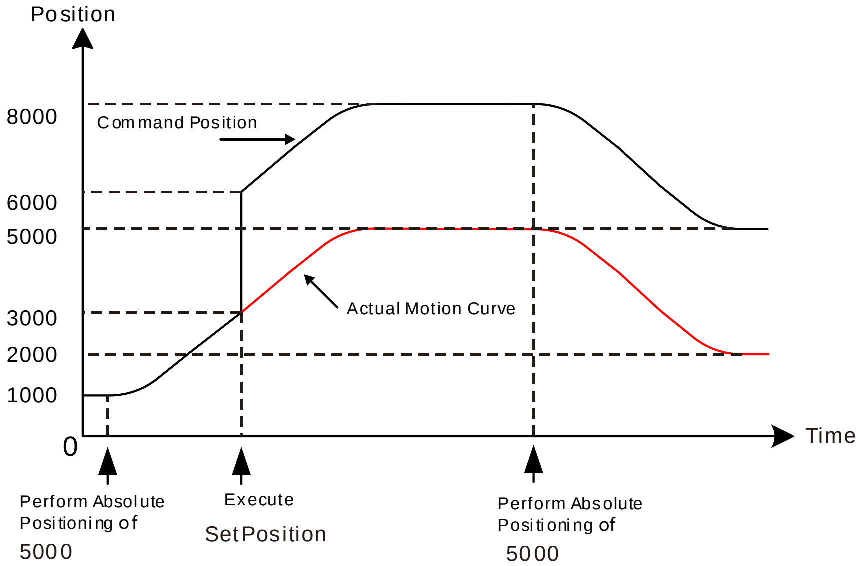


Case 1： When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one period later, *Done* changes to TRUE and meanwhile, *Busy* changes to FALSE.

Case 2： When an error occurs as *Execute* is TRUE, *Error* changes to TRUE and *ErrorID* shows the corresponding error code. And meanwhile, *Busy* changes to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

- **Function**

MC_SetPosition is used to set the position of the axis to a given value and no actual motion of the axis is incurred. MC_SetPosition execution does not affect the current motion. However, it has an impact on the actual execution effect of the instruction which is executed after MC_SetPosition execution is completed.

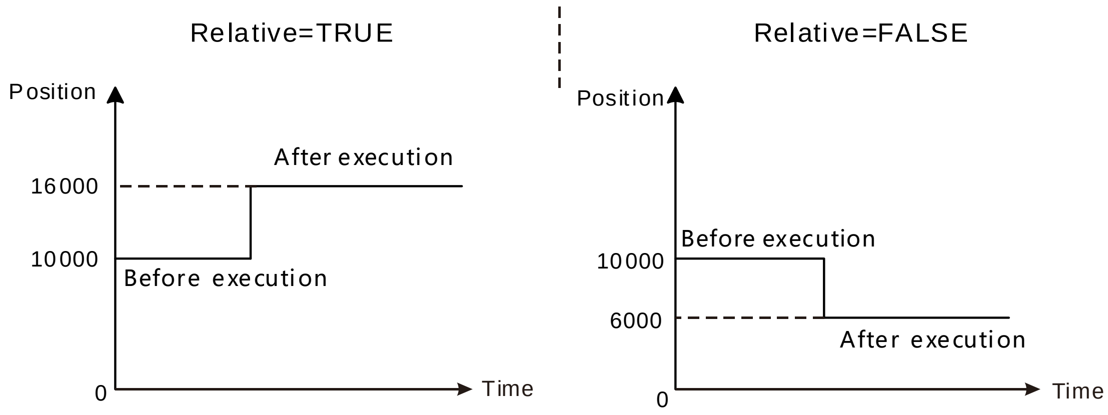


■ Relationship between *Position* and *Relative*

Position, Relative and reference position which stands for the axis position at the moment when the instruction starts being executed jointly determine the position setting value.

Relative is used to define the relationship between *Position* and reference position. When *Relative* is set to TRUE, it is a relative relationship between *Position* and reference position and the position setting value= reference position+ *Position.* When *Relative* is FALSE, it is an absolute relationship between *Position* and reference position and the position setting value equals *Position.*

As shown in the following figures, the reference position is set to10000 and the value of *Position* is 6000 for the instruction execution. The corresponding execution results are respectively illustrated for different *Relative* values as below.



■ ReferenceType

ReferenceType is used to select the command position or actual position as the reference position. When *ReferenceType* is 0, the reference position is the command position of the axis. When *ReferenceType* is 1, the reference position is the actual position of the axis.

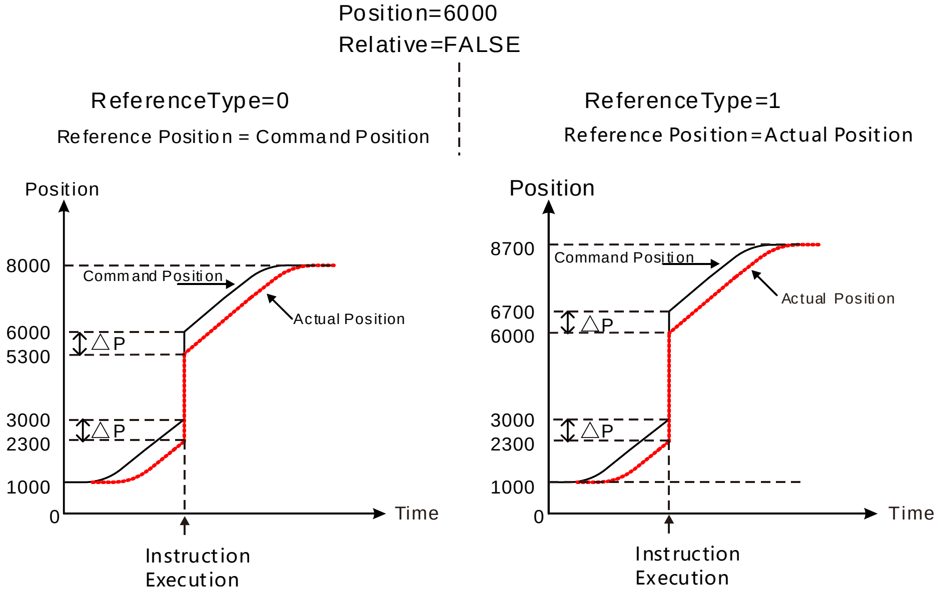
When the command position is taken as the reference position, the instruction calculates the target command position based on the current command position and the value of *Position* and it revises the command position value into the target position value. Meantime, the actual position of the axis will change accordingly. The law of the change is that the variation amount of the actual position is the same as that of the command position. That is to say that the deviation between the command position and actual position remains unchanged at the time when the instruction is executed and the instruction execution ends.

The solution for the actual position which is taken as the reference position is the same as that for the command position which is taken as the reference position.
There will be no difference in execution effect between the command position and actual position as the reference position if the axis is in Standstill state as MC_SetPosition is executed. That is because the difference is 0 between command position and actual position as the axis is still.

The differences in execution effect between command position and actual position as the reference position exist as illustrated below if the axis is in motion as MC_SetPosition is executed. If not zero, the difference between command position and actual position is caused by the command response time.

When MC_SetPosition is executed in absolute mode with *Position* set to 6000 while the axis is positioning with the target position of 5000, the command position and actual position of the axis are 3000 and 2300 respectively (difference value △P =700). The command position changes to 6000 and actual position becomes 5300 (5300=6000-△P) after the instruction is executed if the reference position is the command position as the following left figure shows.

The actual position of the axis changes to 6000 and command position becomes 6700 (6700=6000+△P) after the instruction is executed if the reference position is the actual position as the following right figure shows.



■ **Relationship between Axis Type and Reference Type**

Different axis types are applicable to different reference types as shown in the following table.

Axis type	Reference Type	
	Command Position	Actual Position
Real axis	YES	YES
Encoder axis	YES	YES
Virtual axis	YES	YES

There will be an error in the instruction execution if the axis on which MC_SetPosition is executed does not support the selected Reference Type.

■ Explanation of Instruction Application Situation

When MC_SetPosition is executed on the master axis which is in the built multi-axis relationship, the master axis position change incurred by the instruction does not affect the slave axis. That is, the slave axis will make any motion accordingly when the master axis position change incurred by MC_SetPosition.

When MC_SetPosition is executed on the slave axis, the slave axis position will change but the original relationship between slave axis and master axis will not be influenced.

MC_SetPosition will report an error when it is executed in the process of execution of MC_Stop. But MC_SetPosition can be executed normally after MC_Stop execution is completed.

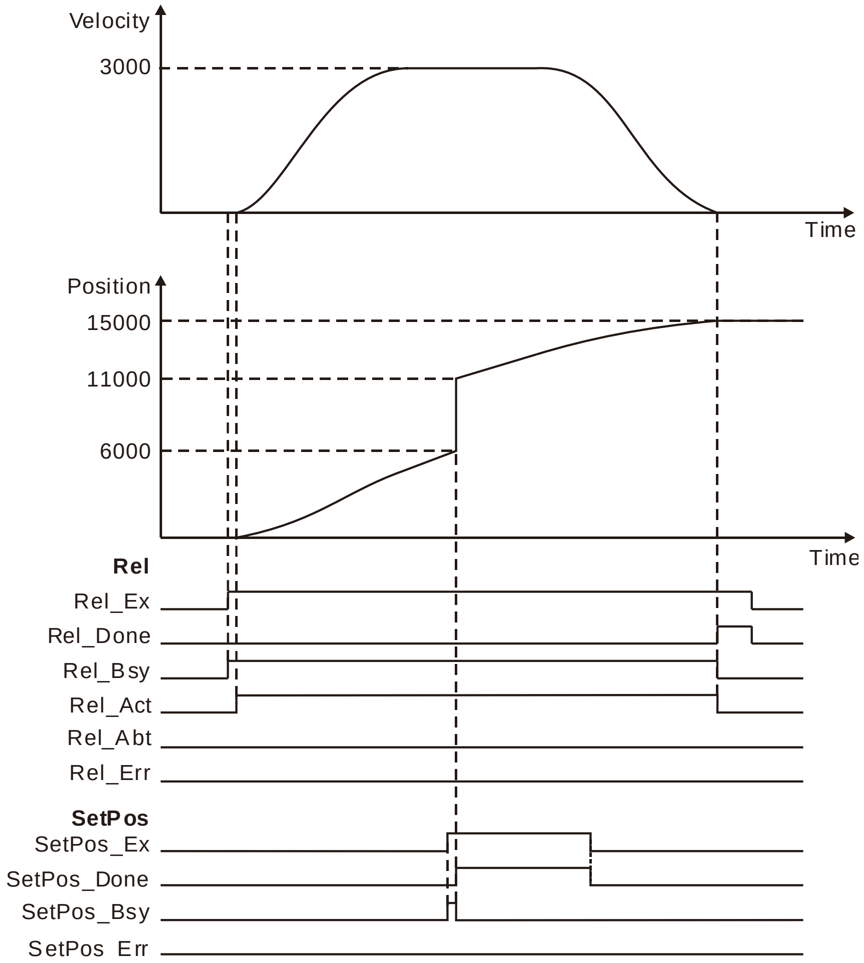
Programming Example 1

The following example shows the impact of MC_SetPosition execution on the positioning instruction when *Relative* of MC_SetPosition instruction is TRUE.

1. The variable table and program

Variable name	Data type	Initial value
Rel	MC_MoveRelative	
Axis1	USINT	1
Rel_Ex	BOOL	FALSE
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	
Tn	TON	
Tn_In	BOOL	FALSE
SRe	SR	
SRe_Q	BOOL	
SetPos	SetPosition	
SetPos_Ex	BOOL	FALSE
SetPos_RefTp	MC_REFERECNE TYPE	0
SetPos_Done	BOOL	
SetPos_Bsy	BOOL	
SetPos_Err	BOOL	
SetPos_ErrID	WORD	

2. **Motion Curve and Timing Charts:**



- ❖ As Rel_Ex changes from FALSE to TRUE, the execution of MC_MoveRelative instruction is started and MC_SetPosition is executed 3 seconds later after MC_MoveRelative is executed.
- ❖ The command position is 6000 as MC_SetPosition starts being executed and 11000 (11000=6000+5000) after the instruction execution ends. The position is 15000 as MC_MoveRelative execution ends.
- ❖ MC_SetPosition does not affect the motion which is being performed through observing the above velocity change curve.

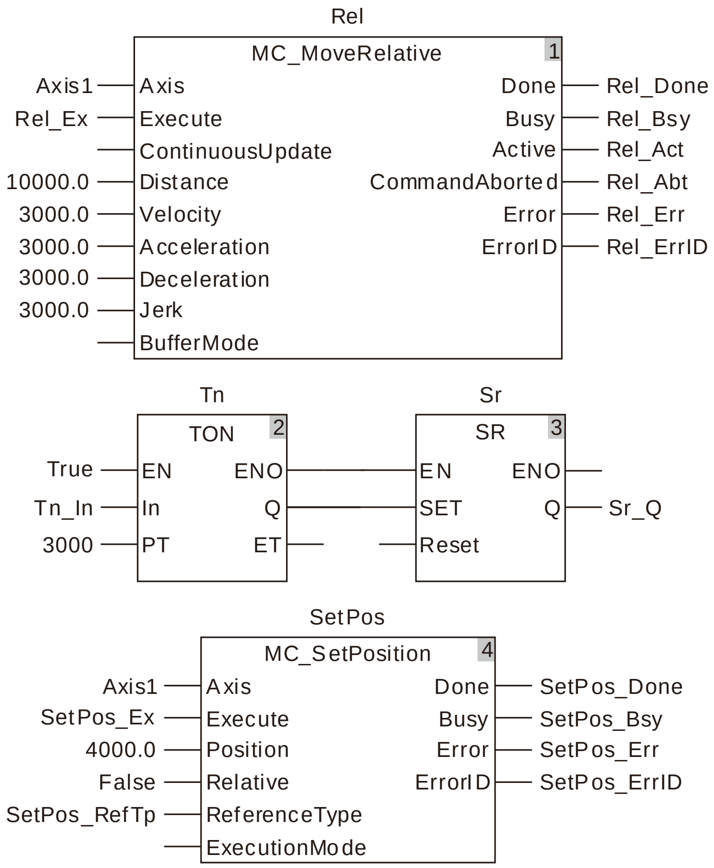
Programming Example 2

The following example describes the impact of MC_SetPosition execution on the axis position when *Relative* of MC_SetPosition instruction is FALSE (the absolute mode is chosen for MC_SetPosition).

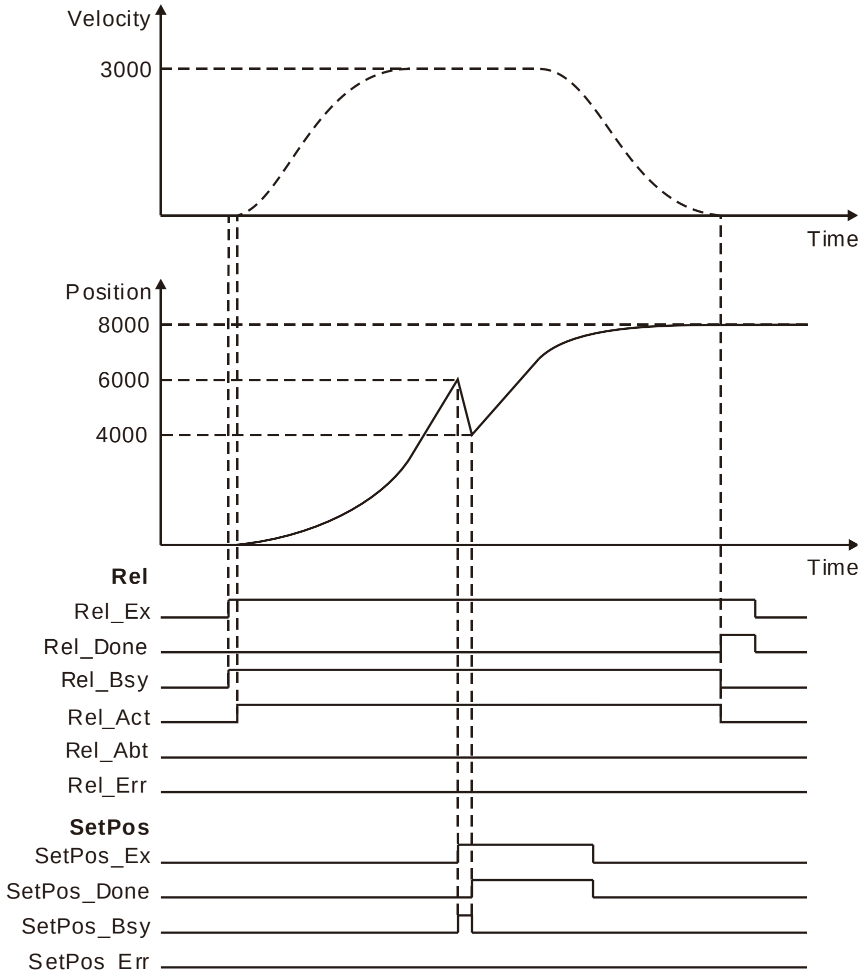
1. **The variable table and program**

Variable name	Data type	Initial value
Rel	MC_MoveRelative	
Axis1	USINT	1
Rel_Ex	BOOL	FALSE
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	

Variable name	Data type	Initial value
Rel_Err	BOOL	
Rel_ErrID	WORD	
Tn	TON	
Tn_In	BOOL	FALSE
SRe	SR	
SRe_Q	BOOL	
SetPos	SetPosition	
SetPos_Ex	BOOL	FALSE
SetPos_RefTp	MC_REFERECNETYPE	0
SetPos_Done	BOOL	
SetPos_Bsy	BOOL	
SetPos_Err	BOOL	
SetPos_ErrID	WORD	



2. Motion Curve and Timing Charts:



- As Rel_Ex changes from FALSE to TRUE, MC_MoveRelative instruction execution starts and MC_SetPosition is executed 3 seconds later after MC_MoveRelative is executed.
- The command position is 6000 as MC_SetPosition starts being executed and 4000 after the instruction execution is completed. The position is 8000 as MC_MoveRelative execution ends.
- MC_SetPosition does not affect the motion which is being performed through observing the above velocity change curve.

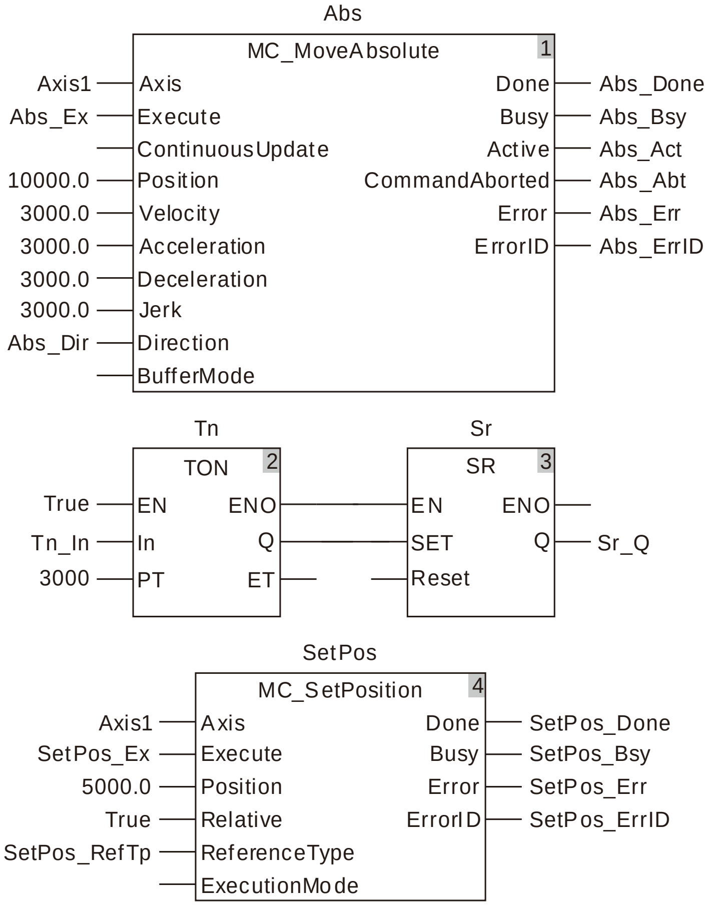
Programming Example 3

The following example shows how MC_SetPosition execution affects MC_MoveAbsolute instruction which is being executed. The actual execution effect of MC_MoveAbsolute which is being executed is not be impacted by MC_SetPosition.

1. The variable table and program

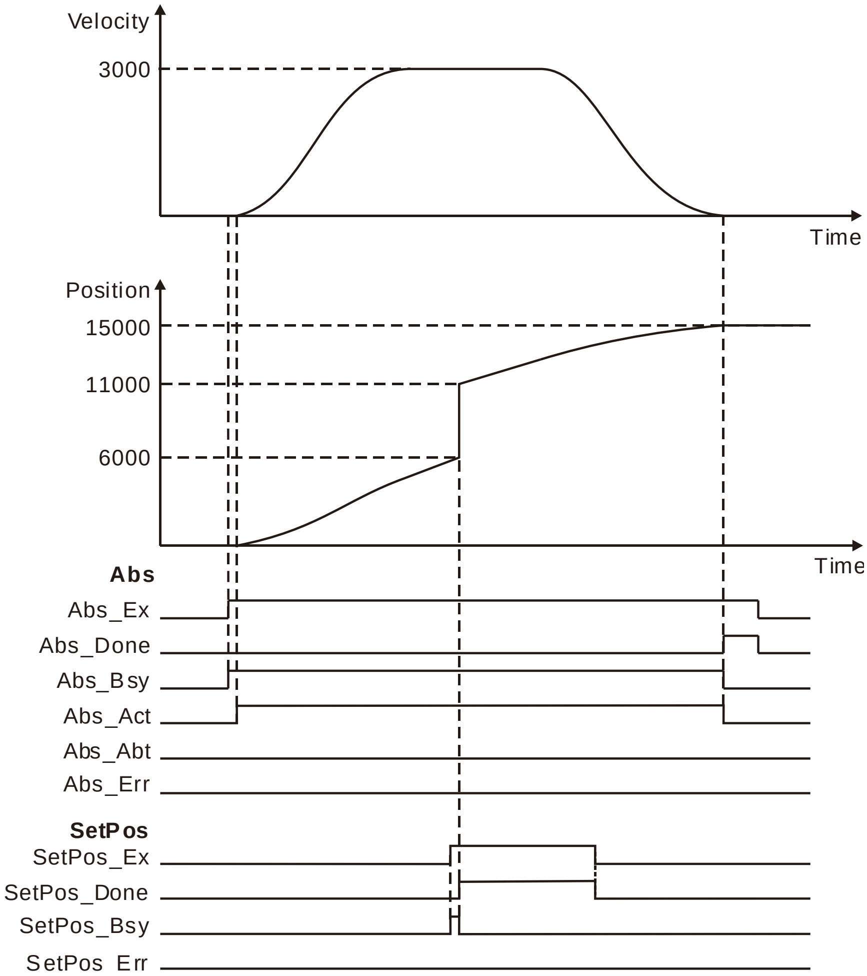
Variable name	Data type	Initial value
Abs	MC_MoveAbsolute	
Axis1	USINT	1
Abs_Ex	BOOL	FALSE
Abs_Dir	MC_DIRECTION	1
Abs_Done	BOOL	
Abs_Bsy	BOOL	
Abs_Act	BOOL	

Variable name	Data type	Initial value
Abs_Abt	BOOL	
Abs_Err	BOOL	
Abs_ErrID	WORD	
Tn	TON	
Tn_In	BOOL	FALSE
SRe	SR	
SRe_Q	BOOL	
SetPos	SetPosition	
SetPos_Ex	BOOL	FALSE
SetPos_RefTp	MC_REFERECNETYPE	0
SetPos_Done	BOOL	
SetPos_Bsy	BOOL	
SetPos_Err	BOOL	
SetPos_ErrID	WORD	



11

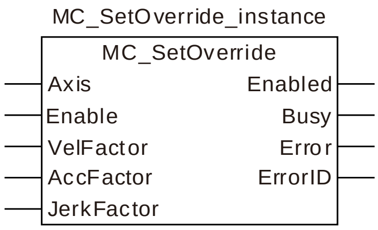
2. Motion Curve and Timing Charts:



- ❖ As Abs_Ex changes from FALSE to TRUE, the execution of MC_MoveAbsolute instruction is started and MC_SetPosition is executed 3 seconds later after MC_MoveAbsolute is executed.
- ❖ The command position is 6000 as MC_SetPosition starts being executed and 11000 after the instruction execution is completed. The position is 15000 as MC_MoveAbsolute execution ends.
- ❖ It can be seen that MC_SetPosition does not affect the actual execution effect of MC_MoveAbsolute which is being executed through observing the above velocity change curve.

11.3.12 MC_SetOverride

FB/FC	Explanation	Applicable model
FB	MC_SetOverride changes the target velocity for an axis.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Enable* changes to TRUE
Enable	The instruction is executed when *Enable* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
VelFactor	Velocity override factor (Unit: %)	LREAL	0~500 (100)	When *Enable* changes to TRUE
AccFactor	Reserved	-	-	-
JerkFactor	Reserved	-	-	-

- **Output Parameters**

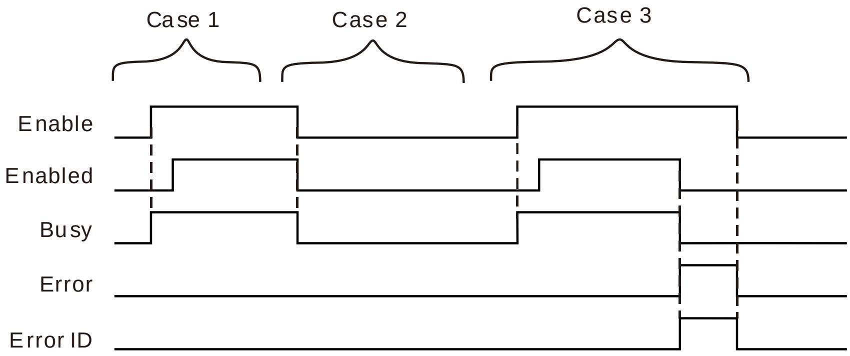
Parameter name	Function	Data type	Valid range
Enabled	TRUE when the instruction is controlling the axis.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE/FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	-

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Enabled	◆ When the instruction starts.	◆ When *Enable* changes to FALSE. ◆ When *Error* changes to TRUE.
Busy	◆ When *Enable* is TRUE.	◆ When *Enable* changes to FALSE. ◆ When *Error* changes to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Enable* changes from TRUE to FALSE

11

- **Output Update Timing Chart**



Case 1： When *Enable* changes from FALSE to TRUE, *Busy* changes to TRUE. *Enabled* changes to TRUE when the instruction execution is completed.

Case 2： When *Enable* changes from TRUE to FALSE, *Enabled* and *Busy* change to FALSE.

Case 3： When an error occurs after *Enable* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error code. Meanwhile, *Enabled* and *Busy* change to FALSE. *Error* changes to FALSE when *Enable* changes from TRUE to FALSE.

- **Function**

MC_SetOverride changes the target velocity for an axis.

1. If the target velocities of motion instructions are to be modified, use the MC_SetOverride instruction. Therefore, the instruction has no influence on the instructions without target velocities. However, *Enabled* remains TRUE even if the *Enable* of MC_SetOverride instruction is set to TRUE for the instructions which are not affected by MC_SetOverride.
2. The instructions of which the target velocities can be modified by MC_SetOverride are shown in the following table.

MC_MoveAbsolute（Absolute positioning）	MC_MoveRelative（Relative positioning）
MC_MoveAdditive（Additive positioning）	MC_MoveVelocity（Velocity instruction）
MC_MoveSuperimposed（Superimposed positioning）	

3. The new target velocity is calculated as below.
   The new target velocity after modification= Target velocity of currently executed instruction x Velocity override factor
4. The unit of *VelFactor* is %. “100” indicates “100%”. The valid range of *VelFactor* is between 0 and 500. An error will occur if the MC_SetOverride instruction is executed when *VelFactor* value exceeds the valid range.
5. The axis will speed up or down till the target velocity after modification is reached according to *Acceleration* or *Deceleration* of the currently executed instruction.
6. An error will occur when the target velocity after modification exceeds the maximum velocity in axis parameters.
7. If *VelFactor* value is set to 0, the target velocity changes to 0, the axis decelerates till the velocity is 0. If the axis operation state need be kept and axis operation need pause, set *VelFactor* value to 0. At the moment, the axis state will not change.
8. When motion instructions are executed or buffered, the VelFactor value can be modified to set the new target velocity.
9. If *VelFactor* value is modified when *Enable* is TRUE, the value will be effective immediately without restarting the MC_SetOverride instruction.

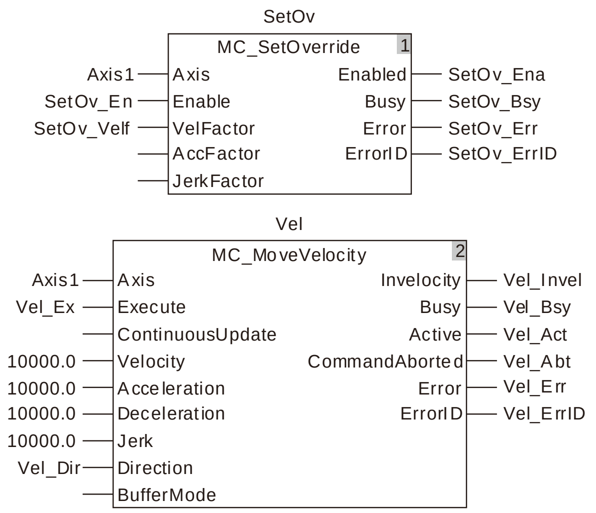
10. If *VelFactor* value is modified when *Enable* is TRUE and *VelFactor* value exceeds the valid range, an error will occur in MC_SetOverride and the target velocity will return to that as *VelFactor* value is 100%.
11. When *Enable* changes to FALSE, the axis will accelerate or decelerate by taking VelFactor=100 as the target.
12. If another MC_SetOverride instruction is started while one MC_SetOverride instruction is being executed on the axis, the execution result of the later executed MC_SetOverride instruction will be regarded as the reference result. The *Enabled* of the two instructions is TRUE.
13. If the MC_SetOverride instruction is used in the course of execution of the MC_MoveVelocity instruction, *InVelocity* remains TRUE even if MC_SetOverride is executed after *Invelocity* of MC_MoveVelocity changes to TRUE.

Programming Example

The example of how MC_MoveVelocity is affected by the execution of the MC_SetOverride instruction is described as below.

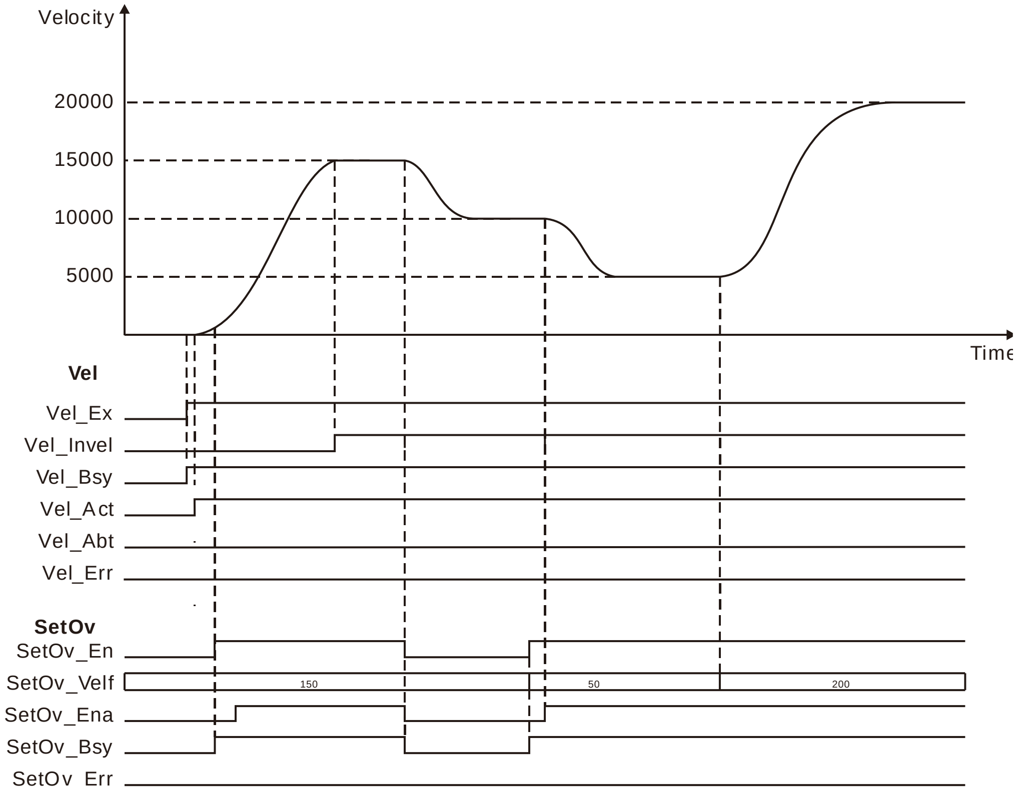
1. **The variable table and program**

Variable name	Data type	Initial value
SetOv	MC_SetOverride	
Axis1	USINT	1
SetOv_En	BOOL	FALSE
SetOv_Velt	LREAL	0.0
SetOv_Ena	BOOL	
SetOv_Bsy	BOOL	
SetOv_Err	BOOL	
SetOv_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel_Ex	BOOL	FALSE
Vel_Dir	MC_DIRECTION	1
Vel_Invel	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	



11

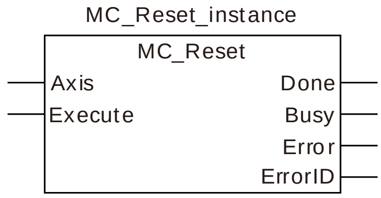
2. **Motion Curve and Timing Chart**



- ❖ When Vel_Ex changes to TRUE, Vel_Bsy changes to TRUE. One cycle later, Vel_Act changes to TRUE and the axis starts to run forward. When the target velocity is not reached (Vel_Invel is not TRUE), SetOv_En is set to TRUE, MC_SetOverride is effective and the target velocity of MC_MoveVelocity changes to the new target velocity. When the new target velocity of MC_MoveVelocity is reached, Vel_Invel changes to TRUE. After Vel_Invel changes to TRUE, Vel_Invel remains TRUE even if VelFactor value (SetOv_Velf) is modified.
- ❖ When SetOv_En changes to FALSE, it means the axis starts to decelerate with the velocity of when Vel_Invel value is 100 as the target velocity.
- ❖ SetOv_Velf value will come to effect immediately if SetOv_Velf value is modified in the course of execution of MC_SetOverride. And the target velocity of MC_MoveVelocity will change accordingly.

11.3.13 MC_Reset

FB/FC	Explanation	Applicable model
FB	MC_Reset clears the error states and axis alarm information inside DVP15MC11T.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-

- **Output Parameters**

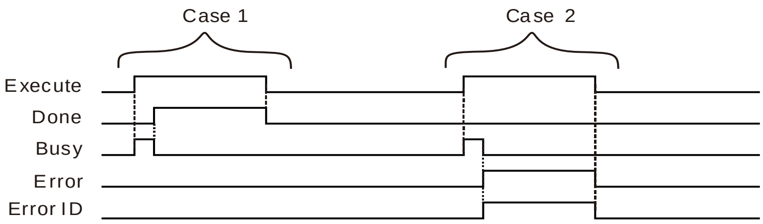
Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the instruction execution is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* is TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE.
Error	◆ When the input parameter values of the instruction are illegal or the mistake cannot be cleared.	◆ When *Execute* changes from TRUE to FALSE

11

- **Output Update Timing Chart**



Case 1： When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE. When the instruction execution is completed, *Done* changes to TRUE and *Busy* changes to FALSE. When *Execute* changes to FALSE, *Done* changes to FALSE.

Case 2： When an error occurs, *Error* changes to TRUE and *ErrorID* shows corresponding error code. When *Execute* changes from TRUE to FALSE, *Error* changes to FALSE and the value of *ErrorID* is cleared to 0.

- **Function**

MC_Reset clears the error state and axis alarm information about the real axis or virtual axis inside DVP15MC11T. The axis state can be observed via MC_ReadStatus.The MC_Reset instruction can be executed to clear the errors when the axis configured in DVP15MC11T enters the ErrorStop state. The instruction can be executed no matter whether the axis enters the ErrorStop state or not. When the errors such as axis alarms, axis offline or state machine switch problems occur, the axis enters the ErrorStop state and the motion instructions which are being executed stop. When the axis alarms, the execution of the instruction can clear the axis alarm information. After the execution of MC_Reset instruction is completed, the axis state will be determined by MC_Power instruction and the axis will be in Disabled or Standstill state.

Refer to chapter 9 for explanation of axis states.

After the axis alarm occurs, excluding the alarm which occurs when the axis meets the limit swtich in the course of homing, the alarm axis enters the ErrorStop state inside DVP15MC11T. The axis alarm can be eliminated if *Done* is TRUE after the instruction is executed. If *Error* is TRUE, the axis alarm cannot be eliminated and users should check if the cause of the error still exists.

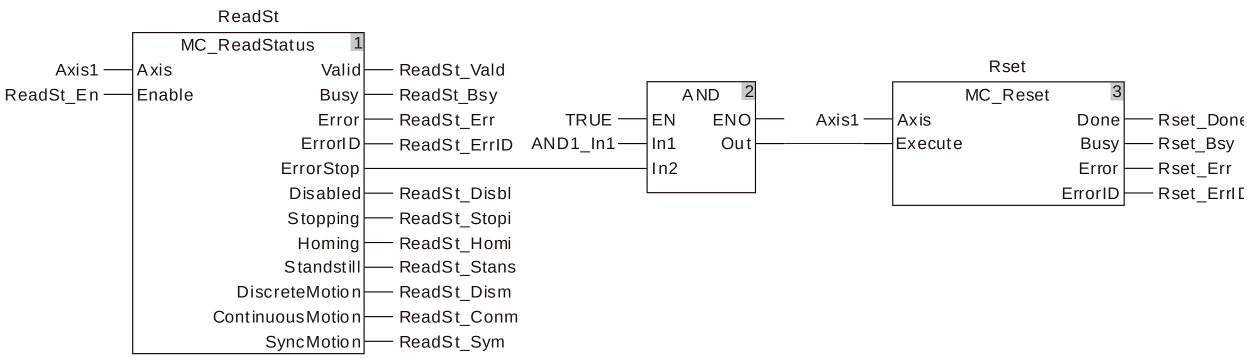
Programming Example

When ReadSt_En is TRUE, the MC_ReadStatus instruction will detect the status of axis 1. When axis 1 enters the ErrorStop state due to axis offline or alarm, *ErrorStop* of the MC_ReadStatus instruction will change to TRUE and the MC_Reset instruction will be executed.

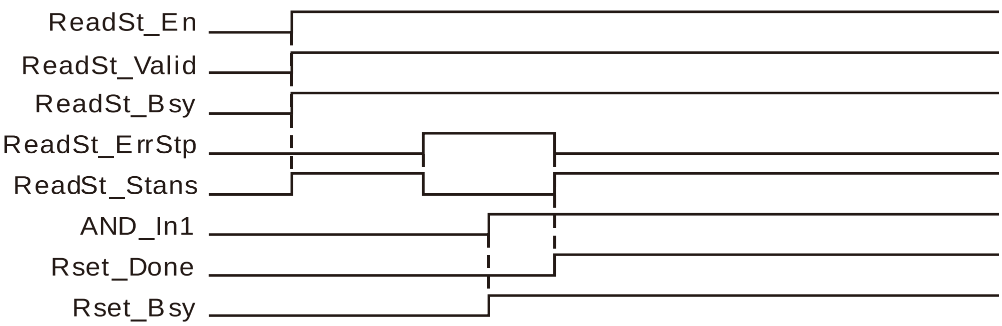
1. The variable table and program

Variable name	Data type	Initial value
ReadSt	MC_ReadStatus	
Axis1	USINT	1
ReadSt_En	BOOL	FALSE
ReadSt_Vald	BOOL	
ReadSt_Bsy	BOOL	
ReadSt_Err	BOOL	
ReadSt_ErrID	WORD	
ReadSt_Disbl	BOOL	
ReadSt_Stpin	BOOL	
ReadSt_Homi	BOOL	
ReadSt_Stans	BOOL	
ReadSt_Dism	BOOL	

Variable name	Data type	Initial value
ReadSt_Conm	BOOL	
ReadSt_Sym	BOOL	
AND1_In1	BOOL	FALSE
Rset	MC_Reset	
Rset_Done	BOOL	
Rset_Bsy	BOOL	
Rset_Err	BOOL	
Rset_ErrID	WORD	



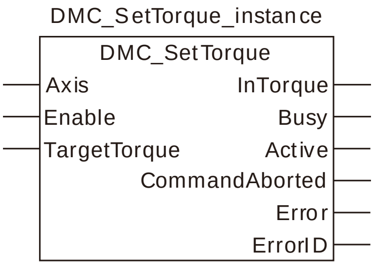
2. Timing Chart



- ❖ When ReadSt_En changes from FALSE to TRUE after the servo axis is enabled, ResdSt_Vald and ResdSt_Bsy change to TRUE and the axis is in Standstill state.
- ❖ AND_In1 is set from FALSE to TRUE when the axis enters the ErrorStop state and MC_Reset is executed. Rset_Busy is TRUE in the first cycle and Rset_Done is TRUE in the second cycle. Meanwhile, the axis enters the Standstill state from the ErrorStop state.

11.3.14 DMC_SetTorque

FB/FC	Explanation	Applicable model
FB	DMC_SetTorque sets the torque of the servo axis. The servo axis will work under the torque mode when the instruction is executed.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Enable* changes to TRUE
Enable	The instruction is executed when *Enable* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
TargetTorque	Specify the value of the target torque. The torque is expressed with the permillage of the rated torque of the servo axis. For example, the setting value 30 indicates that the set torque is 30‰ of the rated torque of the servo axis. While *Enable* is TRUE, modifying the parameter value will change the torque directly.	INT	Negative number, positive number and 0 (0)	When *Enable* changes to TRUE

Notes:

1. If the torque value is a positive number, the effection that the servo produces works in the positive direction. If the torque value is a negative number, the effection that the servo produces works in the negative direction.
2. When *Enable* is TRUE, the instruction is always valid and the torque changes accordingly as the torque value is modified. The instruction cannot be aborted by other instructions excluding MC_Stop. When *Enable* of the instruction is reset to FALSE, the instruction execution stops and other instruction can be executed.

- **Output Parameters**

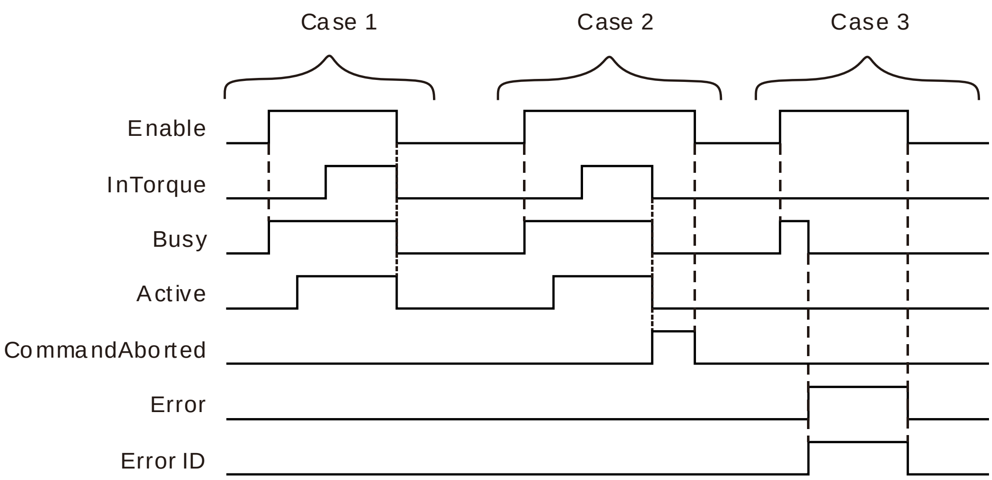
Parameter name	Function	Data type	Valid range
InTorque	TRUE when the target torque is reached.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs.	WORD	

Parameter name	Function	Data type	Valid range
	Please refer to section 12.2 for the corresponding error ID.		

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
InTorque	◆ When the target torque is reached.	◆ When *Error* changes to TRUE. ◆ When *Enable* changes from TRUE to FALSE
Busy	◆ When *Enable* changes to TRUE	◆ When *InTorque* changes to TRUE. ◆ When *Error* changes to TRUE.
Active	◆ When the instruction starts to control the axis	◆ When *InTorque* changes to TRUE. ◆ When *Error* changes to TRUE.
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Enable* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted after *Enable* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Enable* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1 : When *Enable* changes from FALSE to TRUE, *Busy* changes to TRUE in the same cycle. *Active* changes to TRUE in the next cycle and *InTorque* changes to TRUE in the 3rd cycle. When *Enable* changes from TRUE to FALSE, *Busy*, *Active* and *InTorque* change to FALSE in the same cycle.

Case 2 : When the DMC_SetTorque instruction is aborted by MC_Stop after *Enable* changes from FALSE to TRUE, CommandAborted changes to TRUE and meanwhile, *InTorque, Busy* and *Active* change to FALSE. When *Enable* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3 : The input parameter value is illegal such as the axis number: 0 before the DMC_SetTorque instruction is executed. *Busy* changes to TRUE when *Enable* changes from FALSE to TRUE. One cycle later, *Error* changes to TRUE, *Busy* changes to FALSE and *ErrorID* shows corresponding error codes. When *Enable* changes from TRUE to FALSE, *Error* changes from TRUE to FALSE and the content of *ErrorID* is cleared to 0.

11

- **Function**

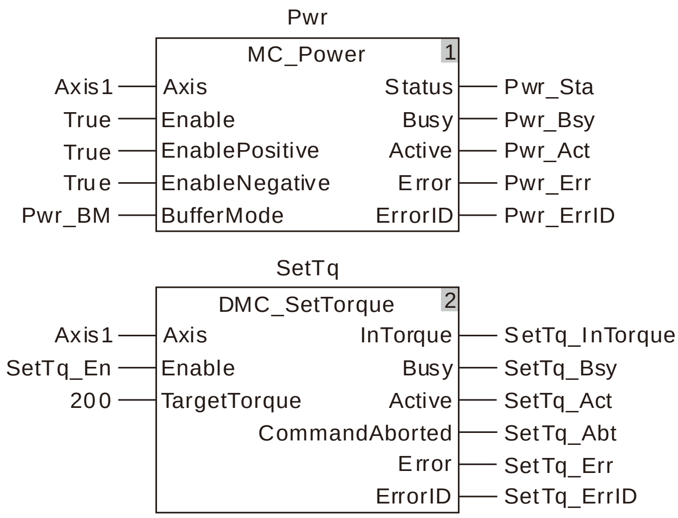
DMC_SetTorque sets the torque of the servo axis. The servo axis will work under the torque mode when the instruction is executed.

Programming Example

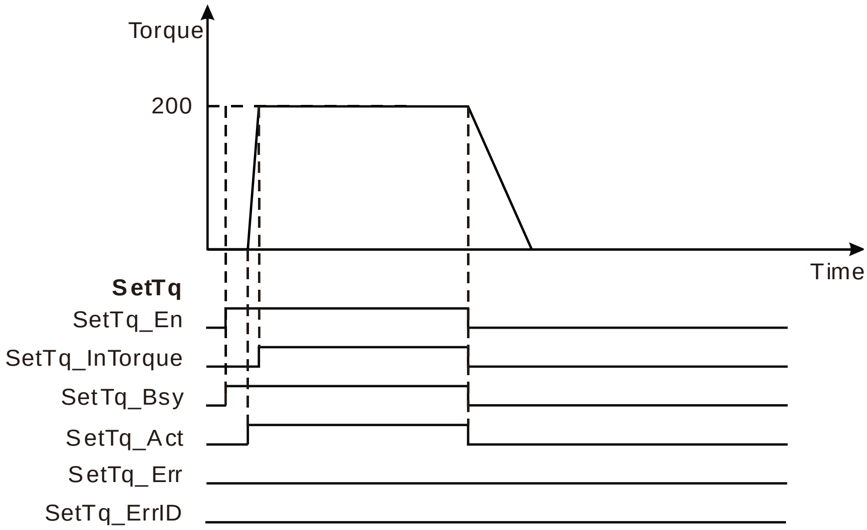
The example of executing the DMC_SetTorque instruction is decribed as follows.

1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
SetTq	DMC_SetTorque	
SetTq_En	BOOL	FALSE
SetTq_InTorque	BOOL	
SetTq_Bsy	BOOL	
SetTq_Act	BOOL	
SetTq_Abt	BOOL	
SetTq_Err	BOOL	
SetTq_ErrID	WORD	



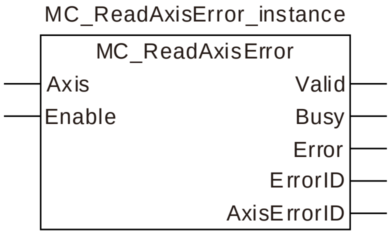
2. Motion Curve and Timing Chart



- ❖ When SetTq_En changes from FALSE to TRUE after the servo axis is enabled, SetTq_Bsy changes to TRUE. One cycle later, SetTq_Act changes to TRUE and the DMC_SetTorque instruction starts. When the torque is reached, SetTq_InTorque changes to TRUE and SetTq_Bsy and SetTq_Act remain TRUE.
- ❖ SetTq_InTorque, SetTq_Bsy and SetTq_Act change to FALSE when SetTq_En changes from FALSE to TRUE.

11.3.15 MC_ReadAxisError

FB/FC	Explanation	Applicable model
FB	MC_ReadAxisError is used to read the error information of a servo axis.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Enable* is TRUE
Enable	The instruction is executed when *Enable* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-

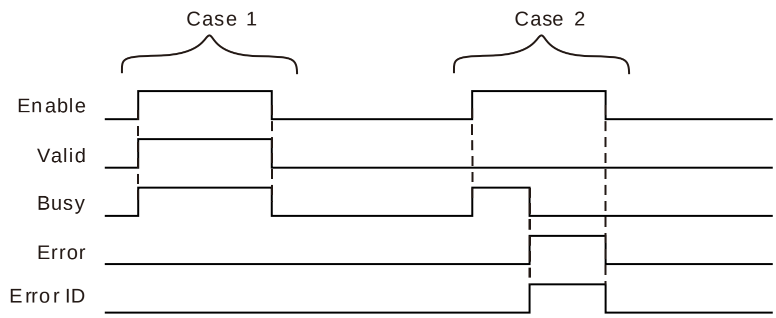
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Valid	TRUE when the output of the instruction is valid.	BOOL	TRUE / FALSE
Busy	TRUE while the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2.	WORD	
AxisErrorID	When *Valid* is TRUE, the value of *ErrorID*, xxx (hex) indicates that the servo drive releases an alarm and xxx is the alarm code that the servo drive reports. For example, AL303 of the servo drive means the value of *ErrorID* is 303 (hex).	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Valid	◆ When an axis error is read	◆ When *Enable* changes from TRUE to FALSE ◆ When Error changes from FALSE to TRUE
Busy	◆ When *Enable* changes to TRUE	◆ When *Enable* changes from TRUE to FALSE ◆ When Error changes from FALSE to TRUE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Enable* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1：When *Enable* changes from FALSE to TRUE, *Valid* and *Busy* change to TRUE. When *Enable* changes to FALSE, *Valid* and *Busy* change to FALSE.

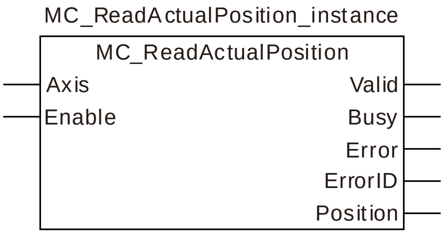
Case 2：When an error occurs, *Error* changes to TRUE and *ErrorID* shows corresponding error code. Meanwhile *Busy* changes to FALSE. When *Enable* changes from TRUE to FALSE, *Error* changes to FALSE and the value of *ErrorID* is cleared.

- **Function**

MC_ReadAxisError is used to read error information of a servo axis such as the alarm code which will show up on the panel of the servo drive and servo axis offline. The instruction is triggered by the high level. Axis errors will be read when *Valid* is TRUE.

11.3.16 MC_ReadActualPosition

FB/FC	Explanation	Applicable model
FB	MC_ReadActualPosition is used to read the actual position of an axis including real axes, virtual axes and encoder axes.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Enable* changes to TRUE
Enable	The instruction is executed when *Enable* changes to TRUE.	BOOL	TRUE or FALSE (FALSE)	-

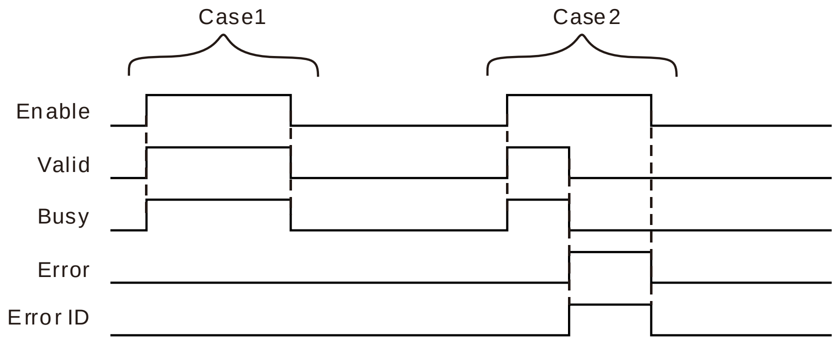
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Valid	TRUE when the output of the instruction is valid.	BOOL	TRUE / FALSE
Busy	TRUE while the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2.	WORD	
Position	The actual position of the axis.	LREAL	

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Valid	◆ When the actual position has been read.	◆ When *Enable* changes from TRUE to FALSE
Busy	◆ When *Enable* changes to TRUE.	◆ When Done changes to TRUE ◆ When Error changes to TRUE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Enable* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1 : When *Enable* changes from FALSE to TRUE, *Valid* and *Busy* change to TRUE simultaneously. When *Enable* changes to FALSE, *Valid* and *Busy* change to FALSE.

Case 2 : As an error occurs, *Error* changes to TRUE and *ErrorID* shows the corresponding error code. Meanwhile, *Busy* and Valid change to FALSE. When *Enable* changes from TRUE to FALSE, *Error* changes to FALSE and the value of *ErrorID* is cleared.

- **Function**

MC_ReadActualPosition is used to read the actual position of an axis including the real axis, virtual axis and encoder axis.

- ■ **Actual Position**

The unit of the actual position read by MC_ReadActualPosition is Unit and the unit of the feedback position that the servo drive gives to the controller is Pulse. Thus the actual position is acquired through conversion of the number of position feedback pulses of the servo drive. The servo gear ratio, mechanical gear ratio and units per output rotation among axis parameters are needed in the conversion.

The conversion formula is shown as below.

$$\text{ActualPosition} = \frac{\text{Units per output rotation}}{\begin{array}{c}\text{(the number of pulses/ rotation)}\\ \text{*mechanical gear ratio}\end{array}} * \begin{array}{c}\text{The number of servo positior}\\ \text{feeddback pulses}\end{array}$$

If the axis is a linear axis, its output *Position* equals ActualPosition above when the instruction is executed.

If the axis is a rotary axis, its output *Position* equals ActualPosition % modulo when the instruction is executed. (*Position* is the remainder got through dividing ActualPosition by the set modulo among the axes parameters) . So the value of *Position* varies between 0 and modulo.

- ■ **Timing for Updating Actual Position**

The timing for updating actual position is related to the cycle time of communication between the controller and servo drive because the actual position comes from the number of feedback position pulses that the servo drive gives. In one communication cycle, the servo sends the number of feedback position pulses to the controller only once. And thus the read actual position remains unchanged within one communication cycle.

For the reasons mentioned above, please use the position capturing function to acquire the more highly real-time position since the instruction reads the less highly real-time actual position of the axis than the position capturing function does.

- **The Impact of MC_SetPosition on Actual Position**

  The actual position that MC_ReadActualPosition reads should also include the position offset caused by MC_SetPosition after MC_SetPosition is executed.

  The conversion formula is shown as below.

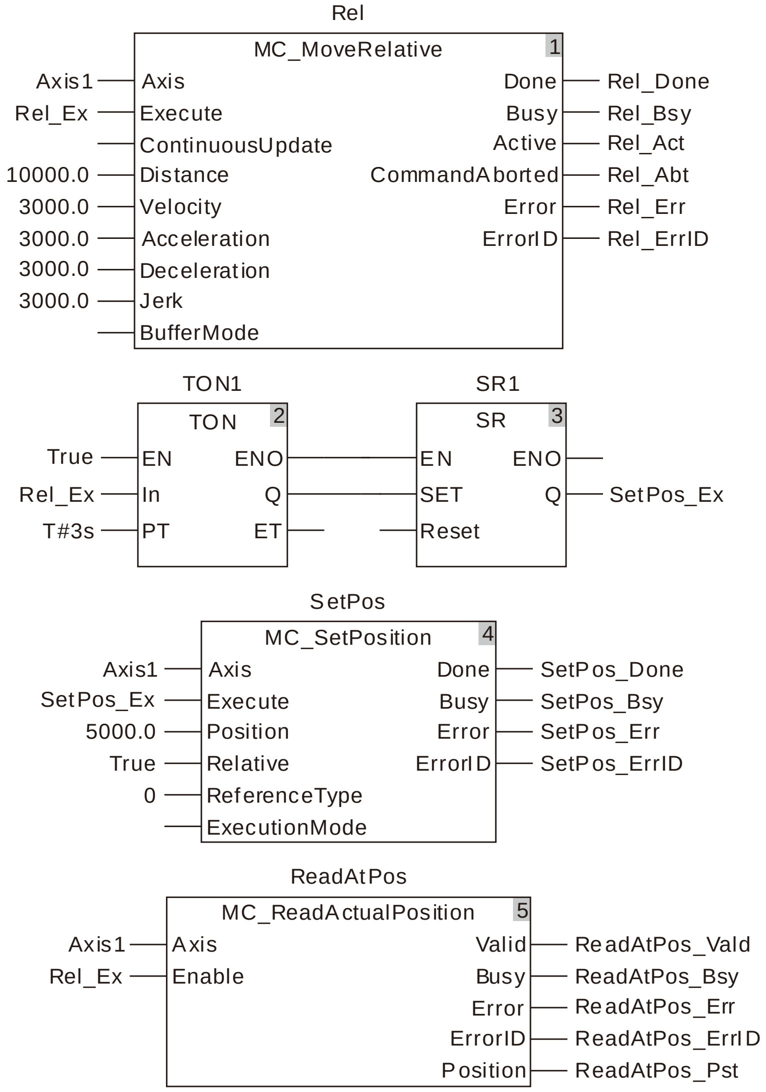
$$\text{ActualPosition} = \text{Position offset caused by MC\_SetPosition} + \frac{\text{Units per output rotation}}{(\text{The number of pulses/rotation}) * \text{mechanical gear ratio}} * \text{The number of servo position feeddback pulses}$$

Programming Example

This example shows the impact that MC_SetPosition has on the execution of MC_ReadActualPosition.

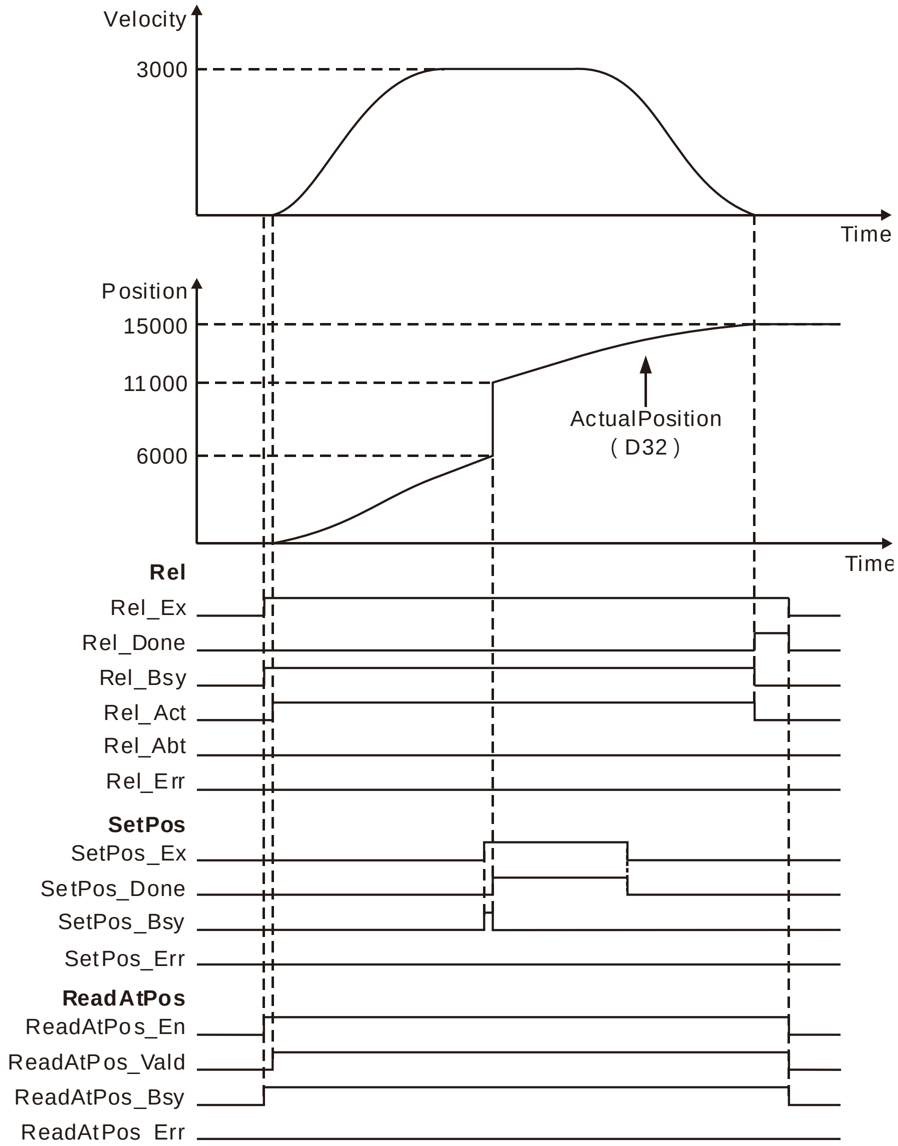
1. The variable table and program

Variable name	Data type	Initial value
Rel	MC_MoveRelative	
Axis1	USINT	1
Rel_Ex	BOOL	FALSE
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	
TON1	TON	
SR1	SR	
SetPos	SetPosition	
SetPos_Ex	BOOL	FALSE
SetPos_RefTp	MC_REFERECNETYPE	0
SetPos_Done	BOOL	
SetPos_Bsy	BOOL	
SetPos_Err	BOOL	
SetPos_ErrID	WORD	
ReadAtPos	ReadActualPosition	
ReadAtPos_Vald	BOOL	
ReadAtPos_Bsy	BOOL	
ReadAtPos_Err	BOOL	
ReadAtPos_ErrID	WORD	
ReadAtPos_Pst	LREAL	


Rel
MC_MoveRelative
1
Axis1
Axis
Rel_Ex
Execute
ContinuousUpdate
10000.0
Distance
3000.0
Velocity
3000.0
Acceleration
3000.0
Deceleration
3000.0
Jerk
BufferMode
Done
Rel_Done
Busy
Rel_Bsy
Active
Rel_Act
CommandAborted
Rel_Abt
Error
Rel_Err
ErrorID
Rel_ErrID
TON1
TON
2
True
EN
ENO
Rel_Ex
In
Q
T#3s
PT
ET
SR1
SR
3
EN
ENO
SET
Q
SetPos_Ex
Reset
SetPos
MC_SetPosition
4
Axis1
Axis
SetPos_Ex
Execute
5000.0
Position
True
Relative
0
ReferenceType
ExecutionMode
Done
SetPos_Done
Busy
SetPos_Bsy
Error
SetPos_Err
ErrorID
SetPos_ErrID
ReadAtPos
MC_ReadActualPosition
5
Axis1
Axis
Rel_Ex
Enable
Valid
ReadAtPos_Vald
Busy
ReadAtPos_Bsy
Error
ReadAtPos_Err
ErrorID
ReadAtPos_ErrID
Position
ReadAtPos_Pst

11

2. **Motion Curve and Timing Charts:**



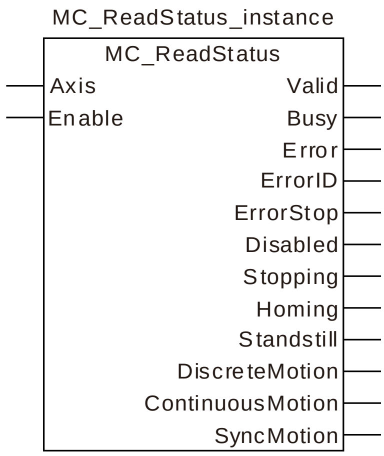
❖ When Rel_Ex changes from FALSE to TRUE, the execution of MC_MoveRelative and MC_ReadActualPosition is started simultaneously. MC_SetPosition is executed 3 seconds later after MC_MoveRelative is executed.

❖ The actual position is 6000 as MC_SetPosition starts being executed and 11000 (11000=6000+5000) after the execution is completed. The actual position is 15000 after MC_MoveRelative execution is completed.

❖ It can be seen from the above velocity curve chart that MC_SetPosition does not affect the ongoing motion. But the ActualPosition curve chart reflects that the actual position that MC_ReadActualPosition reads is affected by MC_SetPosition.

11.3.17 MC_ReadStatus

FB/FC	Explanation	Applicable model
FB	MC_ReadStatus is used to read the servo axis state in the controller.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When Enable changes to TRUE
Enable	The instruction is executed when *Enable* changes to TRUE.	BOOL	TRUE or FALSE (FALSE)	-

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Valid	TRUE when the output of the instruction is valid.	BOOL	TRUE / FALSE
Busy	TRUE while the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	
ErrorStop	Refer to section 10.4.	BOOL	TRUE / FALSE
Disabled		BOOL	TRUE / FALSE
Stopping		BOOL	TRUE / FALSE
Homing		BOOL	TRUE / FALSE
Standstill		BOOL	TRUE / FALSE
DiscreteMotion		BOOL	TRUE / FALSE
ContinuousMotion		BOOL	TRUE / FALSE
SyncMotion		BOOL	TRUE / FALSE

Notes:

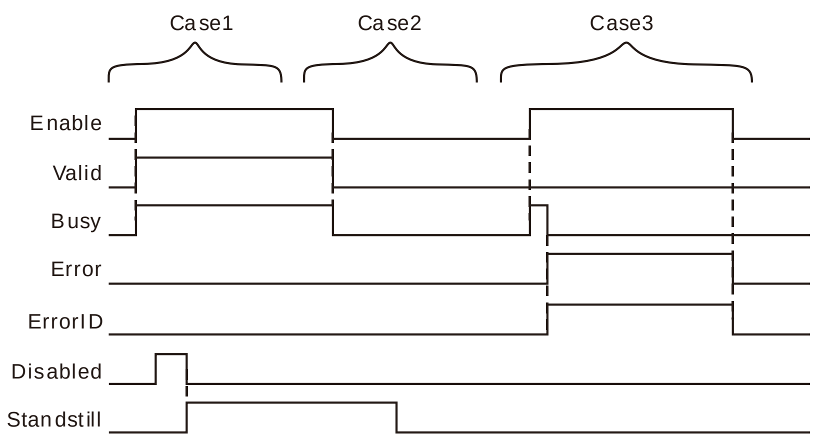
1. When *Enable* changes from FALSE to TRUE, the execution of MC_ReadStatus starts and the axis status is read.

2. When *Enable* changes from TRUE to FALSE, *Valid, Busy* and *Error* change to FALSE, meanwhile *ErrorID* changes to 0 and the outputs of *ErrorStop, Disabled, Stopping, Homing, Standstill, DiscreteMotion, ContinuousMotion* and *SyncMotion* keep the status as *Enable* is TRUE.

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Valid	◆ When *Enable* changes to TRUE	◆ When *Enable* changes from TRUE to FALSE ◆ When *Error* changes from FALSE to TRUE
Busy	◆ When *Enable* changes to TRUE	◆ When *Enable* changes from TRUE to FALSE ◆ When *Error* changes from FALSE to TRUE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Enable* changes from TRUE to FALSE
ErrorStop	◆ When the axis enters ErrorStop state	◆ When the axis is not in ErrorStop state
Disabled	◆ When the axis enters Disabled state	◆ When the axis is not in Disabled state
Stopping	◆ When the axis enters Stopping state	◆ When the axis is not in Stopping state
Homing	◆ When the axis enters Homing state	◆ When the axis is not in Homing state
Standstill	◆ When the axis enters Standstill state	◆ When the axis is not in Standstill
DiscreteMotion	◆ When the axis enters DiscreteMotion state	◆ When the axis is not in DiscreteMotion state
ContinuousMotion	◆ When the axis enters ContinuousMotion state	◆ When the axis is not in ContinuousMotion state
SyncMotion	◆ When the axis enters SyncMotion state	◆ When the axis is not in SyncMotion state

- **Output Update Timing Chart**



Case 1: When *Enable* changes from FALSE to TRUE, *Valid* and *Busy* change to TRUE simultaneously and *ErrorStop, Disabled, Stopping, Homing, Standstill, DiscreteMotion, ContinuousMotion* and *SyncMotion* will change to TRUE or FALSE according to the axis status.

Case 2: When *Enable* changes from TRUE to FALSE, *Valid* and *Busy* change to FALSE simultaneously and the outputs of *ErrorStop, Disabled, Stopping, Homing, Standstill, DiscreteMotion, ContinuousMotion* and *SyncMotion* will keep the same state as *Enable* is TRUE.

Case 3: When the value of the input parameter Axis is out of the valid range and *Enable* changes from FALSE to TRUE, *Busy* changes from FALSE to TRUE, one cycle later, *Error* changes from FALSE to TRUE and *ErrorID* shows corresponding error codes and *Busy* changes from TRUE to FALSE. When *Enable* changes from TRUE to FALSE, *Error* changes from TRUE to FALSE and meanwhile *ErrorID* changes to 0.

- **Function**

MC_ReadStatus is used to read the servo axis state in the controller. For the details on axis states, please refer to section 10.4.

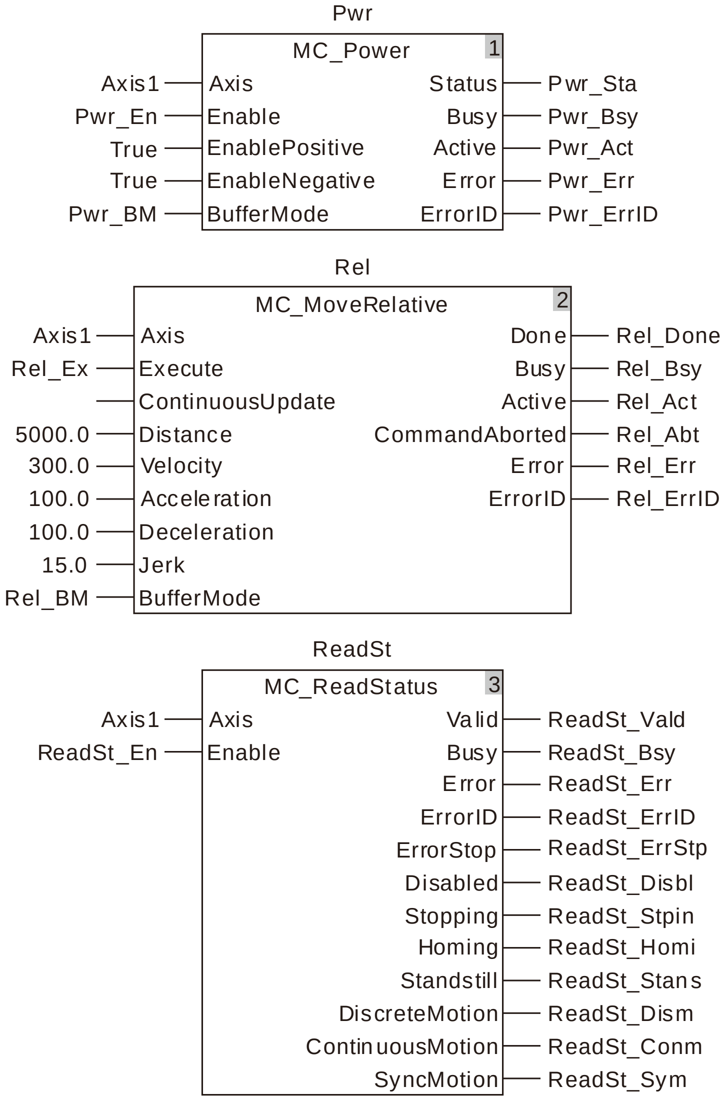
Programming Example

This example of the execution of MC_ReadStatus is shown as below.

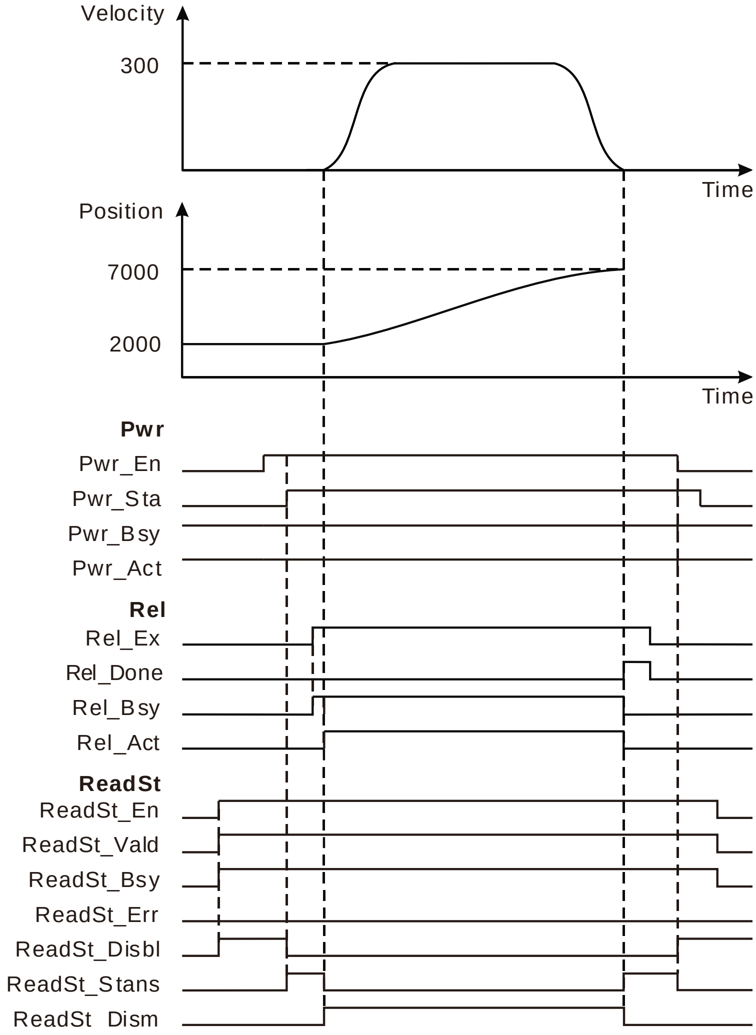
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_BM	MC_Buffer_Mode	0
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel	MC_MoveRelative	
Rel_Ex	BOOL	FALSE
Rel_BM	MC_Buffer_Mode	0
Rel_Done	BOOL	
Rel_Bsy	BOOL	
Rel_Act	BOOL	
Rel_Abt	BOOL	
Rel_Err	BOOL	
Rel_ErrID	WORD	
ReadSt	MC_ReadStatus	
ReadSt_En	BOOL	FALSE
ReadSt_Vald	BOOL	
ReadSt_Bsy	BOOL	
ReadSt_Err	BOOL	
ReadSt_ErrID	WORD	
ReadSt_ErrStp	BOOL	
ReadSt_Disbl	BOOL	
ReadSt_Stpin	BOOL	
ReadSt_Homi	BOOL	

Variable name	Data type	Initial value
ReadSt_Stans	BOOL	
ReadSt_Dism	BOOL	
ReadSt_Conm	BOOL	
ReadSt_Sym	BOOL	



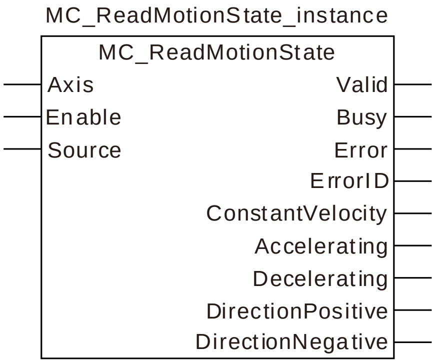
2. Motion Curve and Timing Charts:



- ❖ ReadSt_Vald, ReadSt_Bsy and ReadSt_Disbl change to TRUE as ReadSt_En changes from FALSE to TRUE.
- ❖ When Pwr_Sta changes from FALSE to TRUE, ReadSt_Stans changes to TRUE, ReadSt_Disbl changes to FALSE and the state of the axis changes from Disabled to Standstill.
- ❖ The motion controller controls the servo motor to move by starting from current position as Rel_Act changes from FALSE to TRUE. Meanwhile ReadSt_Stans changes to FALSE and ReadSt_Dism changes to TRUE. When the servo motor moves the target distance, Rel_Done and ReadSt_Stans change to TRUE; Rel_Bsy, Rel_Act and ReadSt_Dism change to FALSE.
- ❖ Rel_Done also changes to FALSE as Rel_Ex changes to FALSE.
- ❖ When Pwr_En changes to FALSE, ReadSt_Disbl changes to TRUE, ReadSt_Stans changes to FALSE and several cycles later Pwr_Sta also changes to FALSE.
- ❖ When ReadSt_En changes to FALSE, ReadSt_Vald and ReadSt_Bsy change to FALSE and ReadSt_Disbl remains TRUE.

11.3.18 MC_ReadMotionState

FB/FC	Explanation	Applicable model
FB	MC_ReadMotionState is used to read current motion state of the servo axis.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled	USINT	1~32 (The variable value must be set)	When *Enable* changes to TRUE
Enable	The instruction is executed when *Enable* changes to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
Source	Reserved	-	-	-

Notes:

1. When *Enable* changes from FALSE to TRUE, the execution of MC_ReadStatus starts.
2. When MC_ReadStatus is being executed and *Enable* changes from TRUE to FALSE, the instruction execution stops and the outputs of *ConstantVelocity*, *Accelerating*, *Decelerating*, *DirectionPositive* and *DirectionNegative* keep the status as *Enable* is TRUE.

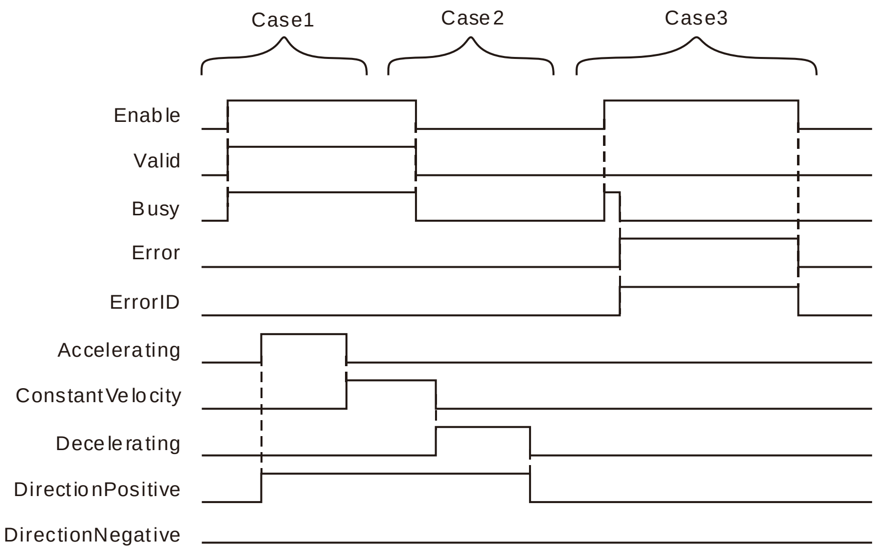
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Valid	TRUE when the output of the instruction is valid.	BOOL	TRUE / FALSE
Busy	TRUE while the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE while there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	
ConstantVelocity	TRUE when the axis moves at a constant speed	BOOL	TRUE / FALSE
Accelerating	TRUE when the absolute value of the axis velocity is increased.	BOOL	TRUE / FALSE
Decelerating	TRUE when the absolute value of the axis velocity is decreased.	BOOL	TRUE / FALSE
DirectionPositive	TRUE when the current position value is increased.	BOOL	TRUE / FALSE
DirectionNegative	TRUE when the current position value is decreased.	BOOL	TRUE / FALSE

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Valid	◆ When the actual velocity of the axis is read	◆ When *Enable* changes from TRUE to FALSE ◆ When *Error* changes from FALSE to TRUE
Busy	◆ When *Enable* changes to TRUE	◆ When *Enable* changes from TRUE to FALSE ◆ When *Error* changes from FALSE to TRUE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Enable* changes from TRUE to FALSE
ErrorID		
ConstantVelocity	◆ When the axis velocity is not changed	◆ When the axis velocity is changed and *Enable* is still TRUE
Accelerating	◆ When the absolute value of the axis velocity is increased	◆ When the axis velocity is not increased any more and *Enable* is still TRUE
Decelerating	◆ When the absolute value of the axis velocity is decreased	◆ When the axis velocity is not decreased any more and *Enable* is still TRUE
DirectionPositive	◆ When the current position value is increased	◆ When the current position value is not increased any more and *Enable* is still TRUE
DirectionNegative	◆ When the current position value is decreased	◆ When the current position value is not decreased any more and *Enable* is still TRUE

- **Output Update Timing Chart**



Case 1： When *Enable* changes from FALSE to TRUE, *Valid* and *Busy* change to TRUE and ConstantVelocity, Accelerating, Decelerating, DirectionPositive and DirectionNegative change to TRUE or FALSE according to the axis state.

Case 2 : When *Enable* changes from TRUE to FALSE, *Valid* and *Busy* change to FALSE and *ConstantVelocity, Accelerating, Decelerating, DirectionPositive* and *DirectionNegative* remain the state for when *Enable* is TRUE.

Case 3 : When the value of Axis is out of the valid range and *Enable* changes from FALSE to TRUE, *Busy* changes from FALSE to TRUE, one period later, *Error* changes from FALSE to TRUE and *ErrorID* shows corresponding error codes. Meanwhile, *Busy* changes from TRUE to FALSE. *Error* changes from TRUE to FALSE and the value of *ErrorID* becomes 0 as *Enable* changes from TRUE to FALSE.

- **Function**

MC_ReadMotionState is used to read current motion state of the servo axis. The motion state of the servo axis includes the constant motion, acceleration or deceleration, positive rotation and negative rotation.

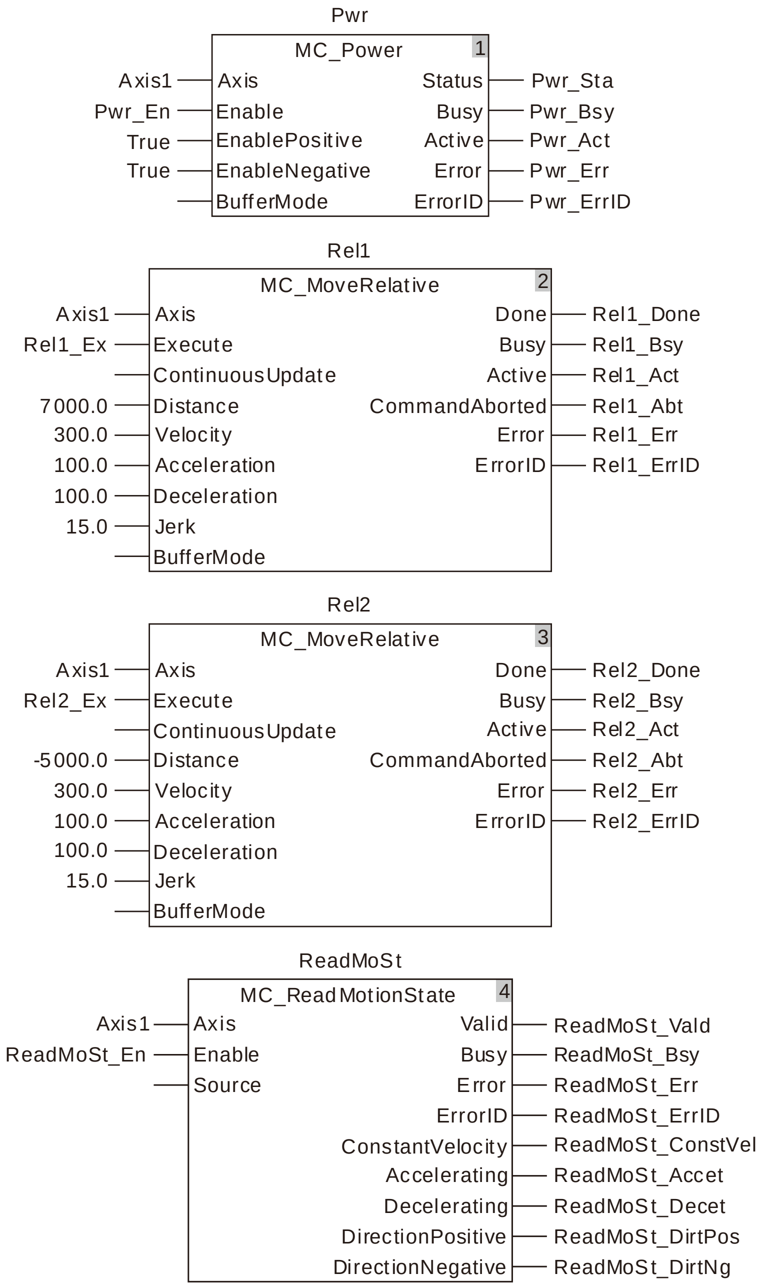
Programming Example

This example of the execution of MC_ ReadMotionState is shown as below.

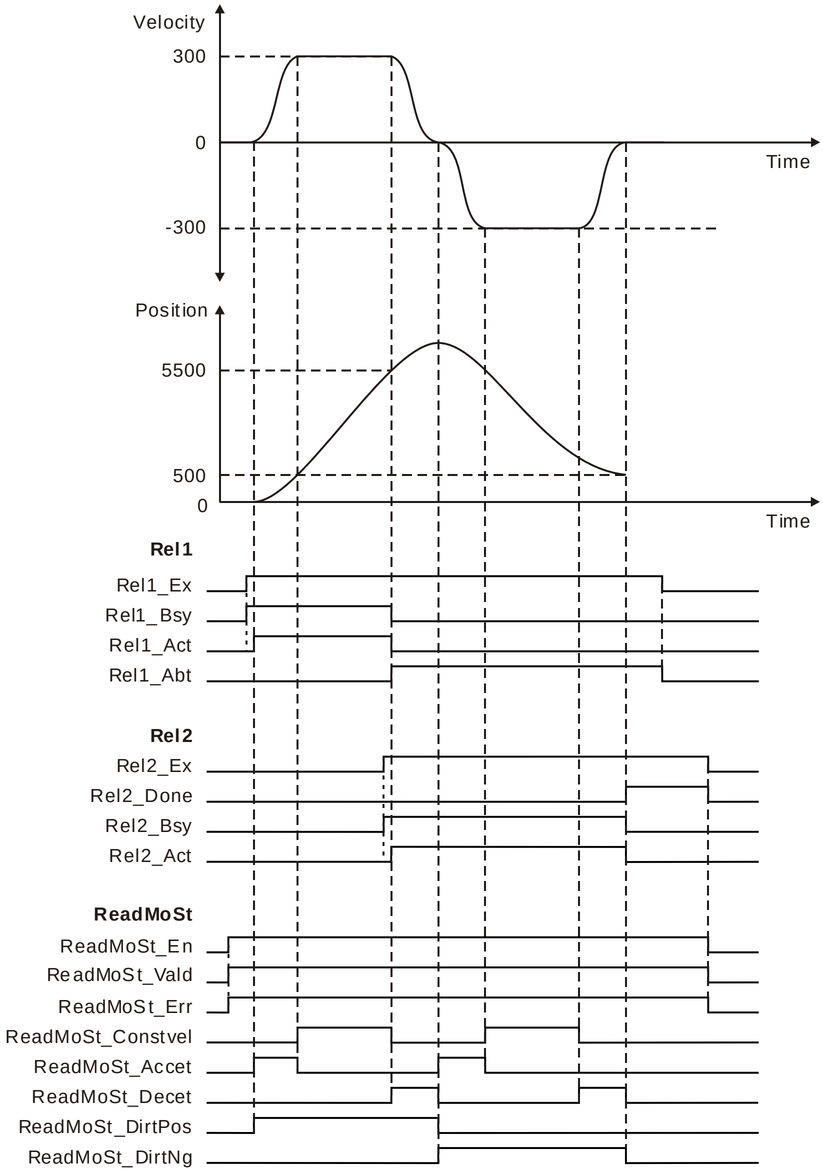
1. The variable table and program

Variable name	Data type	Initial value
Pwr	MC_Power	
Axis1	USINT	1
Pwr_En	BOOL	FALSE
Pwr_Sta	BOOL	
Pwr_Bsy	BOOL	
Pwr_Act	BOOL	
Pwr_Err	BOOL	
Pwr_ErrID	WORD	
Rel1	MC_MoveRelative	
Rel1_Ex	BOOL	FALSE
Rel1_Done	BOOL	
Rel1_Bsy	BOOL	
Rel1_Act	BOOL	
Rel1_Abt	BOOL	
Rel1_Err	BOOL	
Rel1_ErrID	WORD	
Rel2	MC_MoveRelative	
Rel2_Ex	BOOL	FALSE
Rel2_Done	BOOL	
Rel2_Bsy	BOOL	
Rel2_Act	BOOL	
Rel2_Abt	BOOL	
Rel2_Err	BOOL	
Rel2_ErrID	WORD	
ReadMoSt	MC_ReadMotionState	
ReadMoSt_En	BOOL	FALSE
ReadMoSt_Vald	BOOL	
ReadMoSt_Bsy	BOOL	
ReadMoSt_Err	BOOL	

Variable name	Data type	Initial value
ReadMoSt_ErrID	WORD	
ReadMoSt_ConstVel	BOOL	
ReadMoSt_Accet	BOOL	
ReadMoSt_Decet	BOOL	
ReadMoSt_DirtPos	BOOL	
ReadMoSt_DirtNg	BOOL	



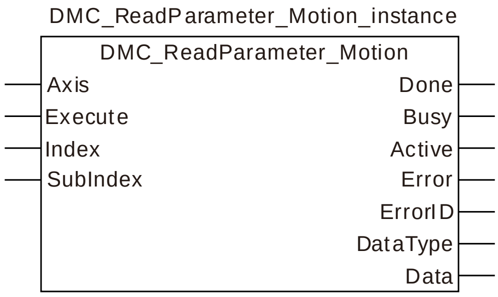
2. Motion Curve and Timing Charts:



- ❖ ReadMoSt_Vald and ReadMoSt_Bsy change from FALSE to TRUE as ReadMoSt_En changes from FALSE to TRUE.
- ❖ When Rel1_Act changes from FALSE to TRUE, the axis starts accelerating in the positive direction and meanwhile, ReadMoSt_Accet and ReadMoSt_DirtPos change to TRUE.
- ❖ When ReadMoSt_Constvel changes from FALSE to TRUE, ReadMoSt_Accet changes from TRUE to FALSE and the axis enters the state of moving at a constant velocity in the positive direction.
- ❖ When Rel2_Act changes from FALSE to TRUE, ReadMoSt_Decet changes from FALSE to TRUE and the axis starts decelerating in the positive direction.
- ❖ When ReadMoSt_Accet and ReadMoSt_DirtNg change from FALSE to TRUE, ReadMoSt_Decet and ReadMoSt_DirtPos change to FALSE simultaneously and the axis starts accelerating in the negative direction.
- ❖ When Rel2_Done changes from FALSE to TRUE, the axis stops moving and both of ReadMoSt_Decet and ReadMoSt_DirtNg change to FALSE.

11.3.19 DMC_ReadParameter_Motion

FB/FC	Explanation	Applicable model
FB	DMC_ReadParameter_Motion reads a slave parameter value.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the station address of the slave to control.	USINT	1~127 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
Index	Index of the parameter to read	UINT	0	When *Execute* changes from FALSE to TRUE
SubIndex	Subindex of the parameter to read	USINT	0	When *Execute* changes from FALSE to TRUE

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
Error	TRUE when an error occurs in execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2 for the corresponding error ID.	WORD	
DataType	The data type of the read parameter 1：Byte, 2：Word, 4：Double Word.	USINT	
Data	The read parameter value	UDINT	

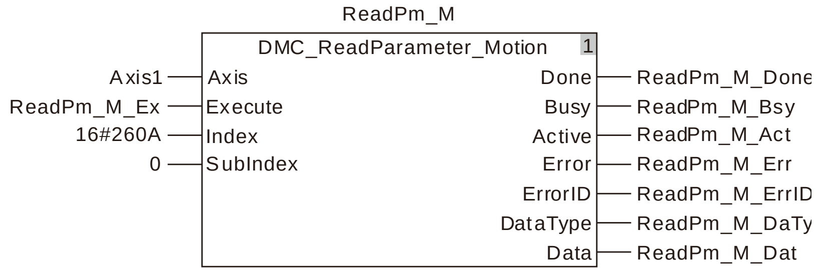
Note: The corresponding index and subindex of a salve parameter

1. User-defined parameter is the servo drive parameter which is to be read. The length is specified by users according to the data type of the parameter to read. The length of the byte parameter is 1. The length of the word parameter is 2. The length of the double-word parameter is 4.

11

The calculation of the index and subindex of a servo parameter is shown as follows.

Index = Servo drive parameter (Hex) + 2000 (Hex)
Subindex = 0.
Example: Calculation of the index of the servo parameter P6-10: 2000 + 060A (the hex. expression of P6-10) = 260A, subindex = 0.

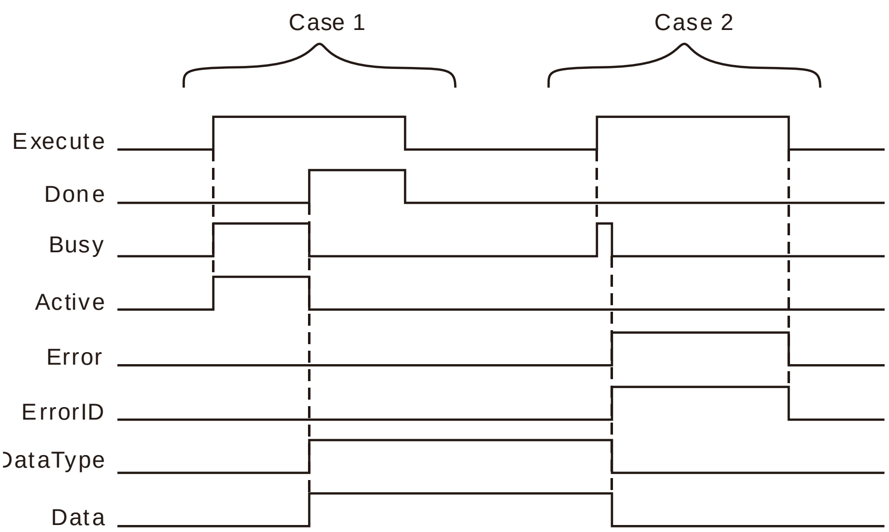


2. For the index and subindex of other slave parameters, refer to the product manual related to CANopen function.

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the reading is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Error* changes to TRUE. ◆ When *Done* changes from FALSE to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Error* changes to TRUE. ◆ When *Done* changes from FALSE to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 3 : When *Execute* changes from FALSE to TRUE, *Busy* and *Active* change to TRUE. One cycle later, *Done* changes to TRUE and *DataType* and *Data* show corresponding data values. After *Done* changes to TRUE, *Busy* and *Active* change to FALSE in the same cycle. When *Execute* changes from TRUE to FALSE, *Done* changes from TRUE to FALSE and *DataType* and *Data* retain original values. If *Error* changes to TRUE, the values of *DataType* and *Data* will be cleared to 0.

Case 4 : The input parameter value is illegal such as axis number: 0 before the DMC_ReadParameter_Motion instruction is executed. When *Execute* changes from FALSE to TRUE, *Error* changes to from FALSE to TRUE and *ErrorID* shows corresponding error code. When *Execute* changes from TRUE to FALSE, *Error* changes from TRUE to FALSE and the content of *ErrorID* is cleared to 0.

- **Function**

DMC_ReadParameter_Motion reads a slave parameter value. Users can specify the index and subindex of the parameter which is to be read.

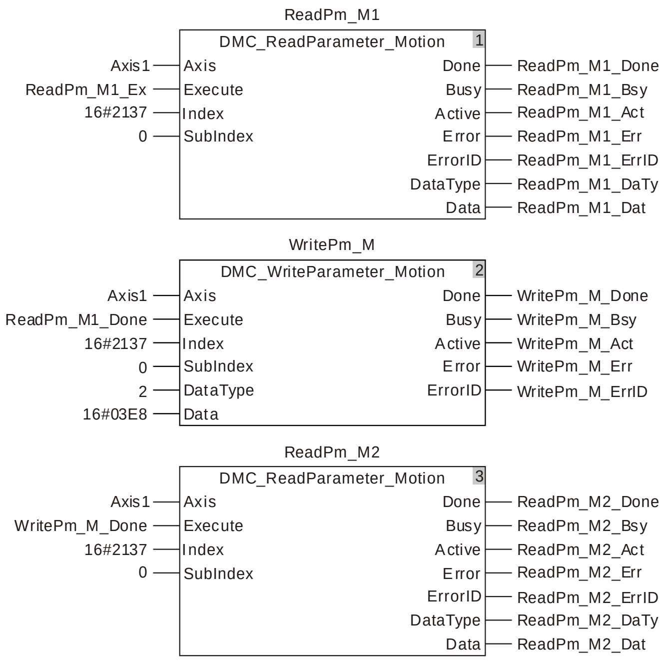
Programming Example

The example of executing the DMC_ReadParameter_ Motion instruction is described as follows.

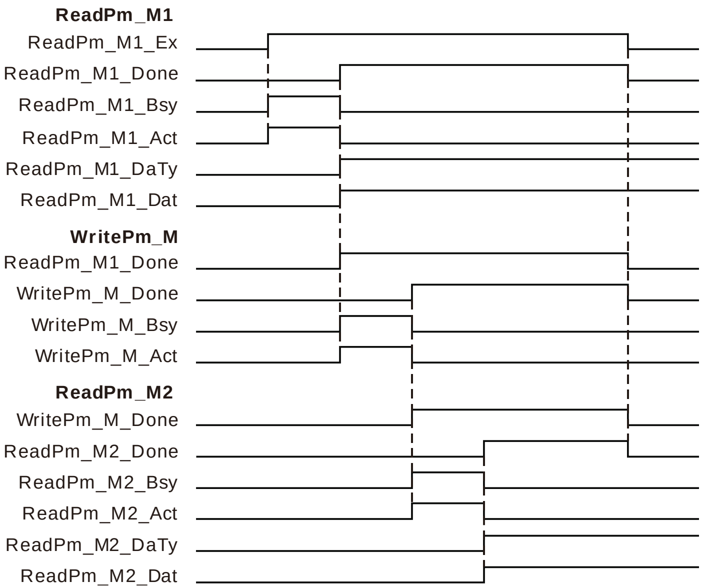
1. The variable table and program

Variable name	Data type	Initial value
ReadPm_M1	DMC_ReadParameter_ Motion	
Axis1	USINT	1
ReadPm_M1_Ex	BOOL	TRUE
ReadPm_M1_Done	BOOL	TRUE
ReadPm_M1_Bsy	BOOL	FALSE
ReadPm_M1_Act	BOOL	FALSE
ReadPm_M1_Err	BOOL	FALSE
ReadPm_M1_ErrID	WORD	FALSE
ReadPm_M1_DaTy	USINT	2
ReadPm_M1_Dat	UDINT	5000
WritePm_M	DMC_WriteParameter_Motion	
WritePm_M_Done	BOOL	TRUE
WritePm_M_Bsy	BOOL	FALSE
WritePm_M_Act	BOOL	FALSE
WritePm_M_Err	BOOL	FALSE
WritePm_M_ErrID	WORD	FALSE
ReadPm_M2	DMC_ReadParameter_ Motion	
ReadPm_M2_Done	BOOL	TRUE
ReadPm_M2_Bsy	BOOL	FALSE
ReadPm_M2_Act	BOOL	FALSE
ReadPm_M2_Err	BOOL	FALSE
ReadPm_M2_ErrID	WORD	FALSE
ReadPm_M2_DaTy	USINT	2
ReadPm_M2_Dat	UDINT	1000

11



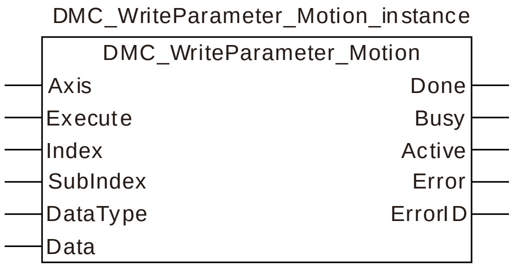
2. Timing Chart



- ❖ When ReadPm_M1_Ex changes from FALSE to TRUE, executing the first DMC_ReadParameter_Motion instruction starts. When the instruction execution is completed, ReadPm_M1_Done changes to TRUE, the value of ReadPm_M1_DaTy is 2 and ReadPm_M1_Dat is 5000. That is, the content of the servo parameter P1-55 which is read is 5000 (The maximum velocity of the servo is limited to 5000rpm.)
- ❖ When ReadPm_M1_Done changes from FALSE to TRUE, executing the DMC_WriteParameter_Motion instruction starts. When the instruction execution is completed, WritePm_M_Done changes to TRUE. That means writing 1000 to the servo slave parameter P1-55 is successful. (The maximum velocity of the servo is limited to 1000rpm)
- ❖ When WritePm_M_Done changes from FALSE to TRUE, executing the second DMC_ReadParameter_Motion instruction starts. When the instruction execution is completed, ReadPm_M2_Done changes to TRUE, ReadPm_M2_DaTy is 2 and ReadPm_M2_Dat is 1000. That is, the content of the servo slave parameter P1-55 which is read is 1000. (The maximum velocity of the servo is limited to 1000rpm.)

11.3.20 DMC_WriteParameter_Motion

FB/FC	Explanation	Applicable model
FB	DMC_WriteParameter_Motion sets a slave parameter value.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the slave to control.	USINT	1~127 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
Index	The index of the parameter to write.	UINT		
SubIndex	The subindex of the parameter to write.	USINT		
DataType	The data type of the parameter to write 1：Byte, 2：Word, 4：Double Word.	USINT		
Data	The value of the parameter to write	UDINT		

Notes:

1. DataType must be the data type of the parameter to write. An error will occur if the filled value is incorrect.
2. Refer to Chapter 9 for the calculation of the index and subindex of servo parameters.

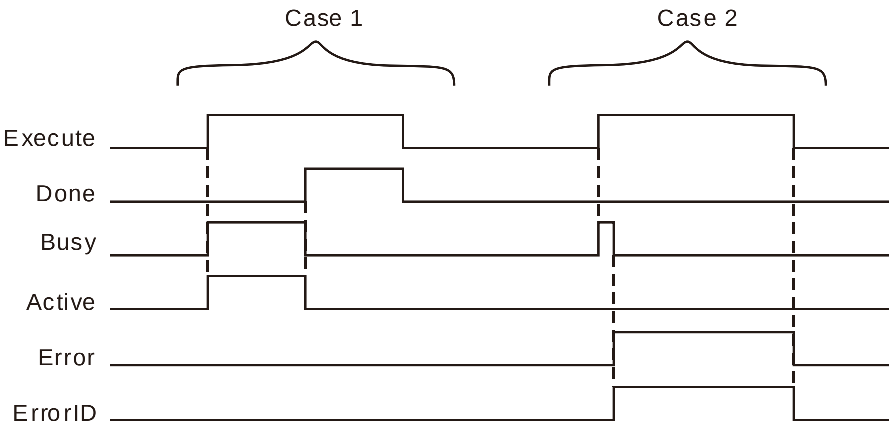
- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the writing is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Error* changes to TRUE. ◆ When *Done* changes from FALSE to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *Error* changes to TRUE. ◆ When *Done* changes from FALSE to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1： When *Execute* changes from FALSE to TRUE, *Busy* and *Active* change to TRUE. One cycle later, *Done* changes to TRUE. After *Done* changes to TRUE, *Busy* and *Active* change to FALSE in the same cycle. When *Execute* changes from TRUE to FALSE, *Done* changes from TRUE to FALSE.

Case 2： The input parameter value is illegal such as axis number: 0 before the DMC_WriteParameter_Motion instruction is executed. When *Execute* changes from FALSE to TRUE, *Error* changes to from FALSE to TRUE and *ErrorID* shows corresponding error code. When *Execute* changes from TRUE to FALSE, *Error* changes from TRUE to FALSE and the content of *ErrorID* is cleared to 0.

- **Function**

DMC_WriteParameter_Motion sets a slave parameter value. Users can specify the index and subindex of the parameter which is to be set.

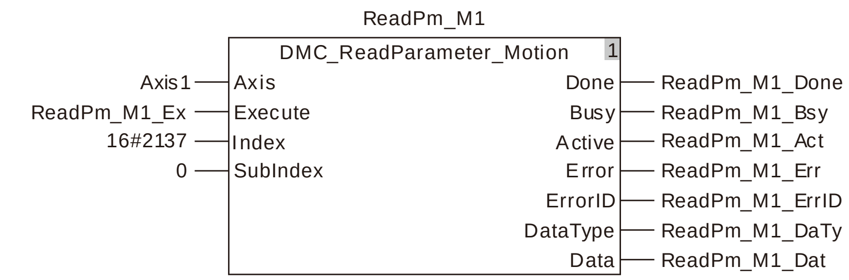
Programming Example

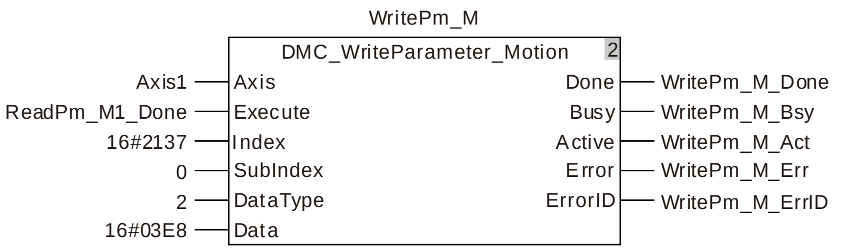
The example of executing the DMC_WriteParameter_ Motion instruction is described as follows.

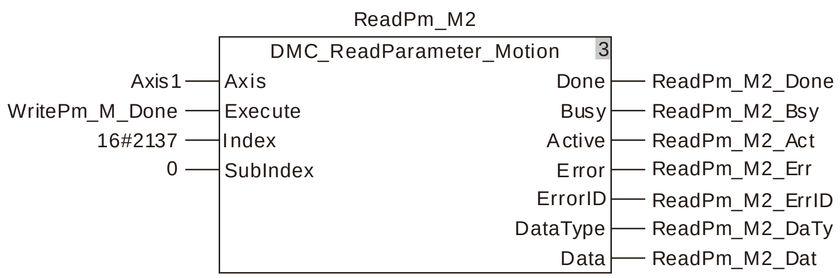
1. The variable table and program

Variable name	Data type	Initial value
ReadPm_M1	DMC_ReadParameter_Motion	
Axis1	USINT	1
ReadPm_M1_Ex	BOOL	TRUE
ReadPm_M1_Done	BOOL	TRUE
ReadPm_M1_Bsy	BOOL	FALSE

Variable name	Data type	Initial value
ReadPm_M1_Act	BOOL	FALSE
ReadPm_M1_Err	BOOL	FALSE
ReadPm_M1_ErrID	WORD	FALSE
ReadPm_M1_DaTy	USINT	2
ReadPm_M1_Dat	UDINT	5000
WritePm_M	DMC_WriteParameter_Motion	
WritePm_M_Done	BOOL	TRUE
WritePm_M_Bsy	BOOL	FALSE
WritePm_M_Act	BOOL	FALSE
WritePm_M_Err	BOOL	FALSE
WritePm_M_ErrID	WORD	FALSE
ReadPm_M2	DMC_ReadParameter_Motion	
ReadPm_M2_Done	BOOL	TRUE
ReadPm_M2_Bsy	BOOL	FALSE
ReadPm_M2_Act	BOOL	FALSE
ReadPm_M2_Err	BOOL	FALSE
ReadPm_M2_ErrID	WORD	FALSE
ReadPm_M2_DaTy	USINT	2
ReadPm_M2_Dat	UDINT	1000

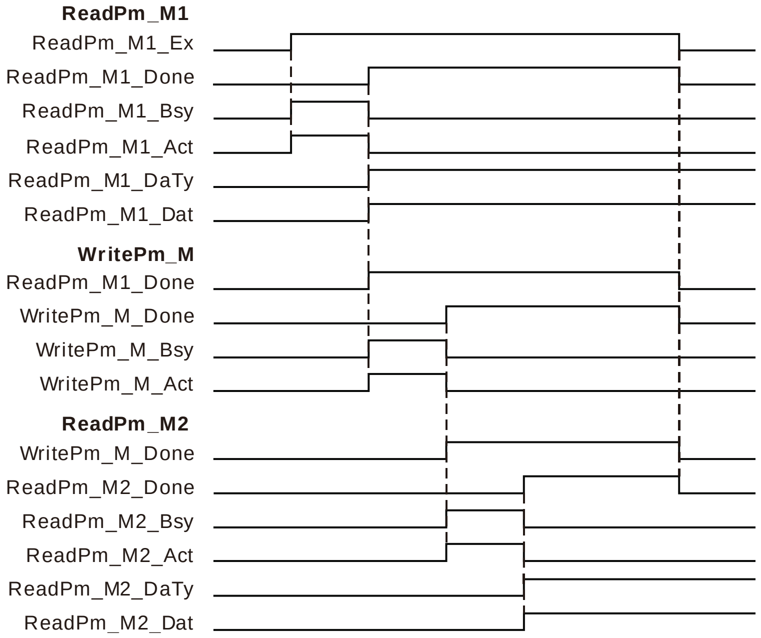






11

2. **Timing Chart**



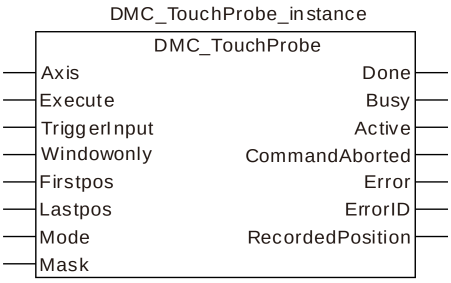
❖ When ReadPm_M1_Ex changes from FALSE to TRUE, executing the first DMC_ReadParameter_Motion instruction starts. After the instruction execution is completed, ReadPm_M1_Done changes to TRUE, ReadPm_M1_DaTy is 2 and ReadPm_M1_Dat is 5000. That is, the content of the servo slave parameter P1-55 which is read is 5000. (The maximum velocity of the servo is limited to 5000rpm.)

❖ When ReadPm_M1_Done changes from FALSE to TRUE, executing DMC_WriteParameter_Motion starts. When the instruction execution is completed, WritePm_M_Done changes to TRUE. That means the content of the servo slave parameter P1-55 which is set is 1000. (The maximum velocity of the servo is limited to 1000rpm.)

❖ When WritePm_M_Done changes from FALSE to TRUE, executing the second DMC_ReadParameter_Motion instruction starts. When the instruction execution is completed, ReadPm_M2_Done changes to TRUE, ReadPm_M2_DaTy is 2 and ReadPm_M2_Dat is 1000. That is, the content of the servo slave parameter P1-55 which is read is 1000. (The maximum velocity of the servo is limited to 1000rpm.)

11.3.21 DMC_TouchProbe

FB/FC	Explanation	Applicable model
FB	DMC_TouchProbe is used for capturing the position of an axis.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Axis	Specify the number of the axis which is to be controlled.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
TriggerInput	Specify one of the input points I0~I7, I10~I17 of DVP15MC11T as the bit for triggering position capture. The values of the parameter 0~15 correspond to input points I0~I7 and I10~I17. The parameter is valid when *Mode* is 0 and 1 and invalid when *Mode* is 2, 3 and 4.	MC_Triggerinput	0:mcTriggerinputI0 ... 7: mcTriggerinputI7 8:mcTriggerinputI10 ... 15: mcTriggerinputI17 (0)	
Windowonly	Reserved	-	-	-
Firstops	Reserved	-	-	-
Lastops	Reserved	-	-	-
Mode	Mode 0: The trigger signal comes from the rising edge of the input points: I0~I7 and I10~I17 of DVP15MC11T. The input point which is used is specified by *TriggerInput*. The position is captured through the rising edge of the trigger bit. The captured position is converted from the number of pulses that the external encoder port of the controller receives through axis parameters.	INT		

Parameter name	Function	Data type	Valid range (Default)	Validation timing
	Mode 1: The trigger signal comes from the falling edge of one of the input points: I0~I7 and I10~I17 of DVP15MC11T, which is specified by *TriggerInput*. The captured position is converted from the number of pulses that the external encoder port of the controller receives through axis parameters. Mode 2: The trigger signal comes from the rising edge of the high-speed input point: DI7 of the servo drive. The captured position is converted from the number of pulses which the servo motor feeds back to the servo drive through axis parameters. Mode 3: The trigger signal comes from the rising edge of the high-speed input point: DI7 of the servo drive. The captured position is converted from the number of pulses that CN1 port of the servo drive receives through axis parameters. Mode 4: The trigger signal comes from the rising edge of the high-speed input point: DI7 of the servo drive. The captured position is converted from the number of pulses that CN5 port of the servo drive receives through axis parameters.			
Mask	Reserved	-	-	-

Notes:

1. In Mode 0 and mode 1, the same input point cannot be used for the position capture simultaneously.
2. In Mode 2, mode 3 and mode 4, the position capture cannot be performed for the same axis simultaneously.

- **Output Parameters**

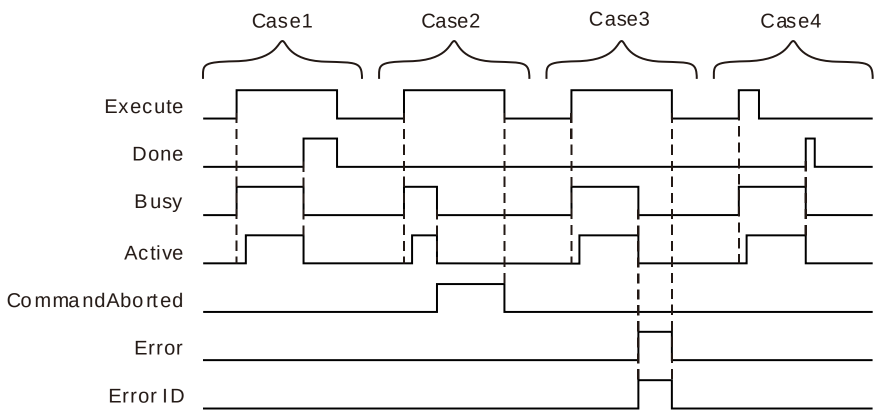
Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

Parameter name	Function	Data type	Valid range
RecordedPosition	The captured position after the completion of the instruction execution. Refer to the following Function for details.	LREAL	

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the instruction execution is completed.	◆ When Execute changes from TRUE to FALSE
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Active	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
CommandAborted	◆ When the instruction execution is aborted by some other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction execution is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One period later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1 : When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE and one period later, *Active* changes to TRUE. When positioning is completed, *Done* changes to TRUE and meanwhile *Busy* and *Active* change to FALSE.

Case 2 : When *Execute* changes from FALSE to TRUE and the instruction is aborted by other instruction, *Commandaborted* changes to TRUE and meanwhile *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3： After *Execute* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error codes when an error occurs such as axis alarm or Offline. Meanwhile, *Busy* and *Active* change to FALSE. *Error* changes to FALSE when *Execute* changes from TRUE to FALSE.

Case 4： During execution of the instruction, *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE. Meanwhile, *Busy* and *Active* change to FALSE and one period later, *Done* changes to FALSE.

- **Function**
  - (*RecordedPosition*) the position that DMC_TouchProbe captures is converted from other value based on axis parameters. The data sources for conversion are listed in the following table.

Mode	Data source
Mode 0 and mode 1	The number of pulses that the external encoder port of DVP15MC11T receives
Mode 2	The number of pulses that the servo motor feeds back to the servo drive
Mode 3	The number of pulses that the pulse, /pulse, sign and /sign input terminals of CN1 port of the servo drive receive.
Mode 4	The number of pulses that A, /A, B and /B input terminals of CN5 port of the servo drive receive.

  - For mode 0, 1 or 2, the range of the data source value is -2147483648~2147483647. When the data source value exceeds 2147483647, it will become -2147483648. With the changing + or – sign of the data source value, the + or - sign of the value of *RecordedPosition* will not change but the value of *RecordedPosition* will continue to increase.
  - For mode 3 or 4, the range of the data source value is -2147483648~2147483647. When the data source value exceeds 2147483647, it will become -2147483648. The value of *RecordedPosition* will change from a positive number to a negative number.

- The position captured by the DMC_TouchProbe instruction is calculated according to axis parameters. For different modes, the data sources are different. “Servo gear ratio setting” and “Mechanism gear ratio setting” in axis parameters are shown in the following table. When *Mode* value of the instruction is equal to 3 (which you can refer to the introduction of mode 3 below), the number of pulses received at pulse, /pulse, sign and /sign of CN1 is 435 and the position captured by the instruction is 65.25. The calculation formula: 435×（3×1000）÷（2×10000）=65.25. 10000, 2, 3 and1000 in the formula correspond to 10000, 2, 3, and 1000 in the following table respectively. For other mode, the calculation method for the position captured by the instruction is the same as that described above but only the data source is different.

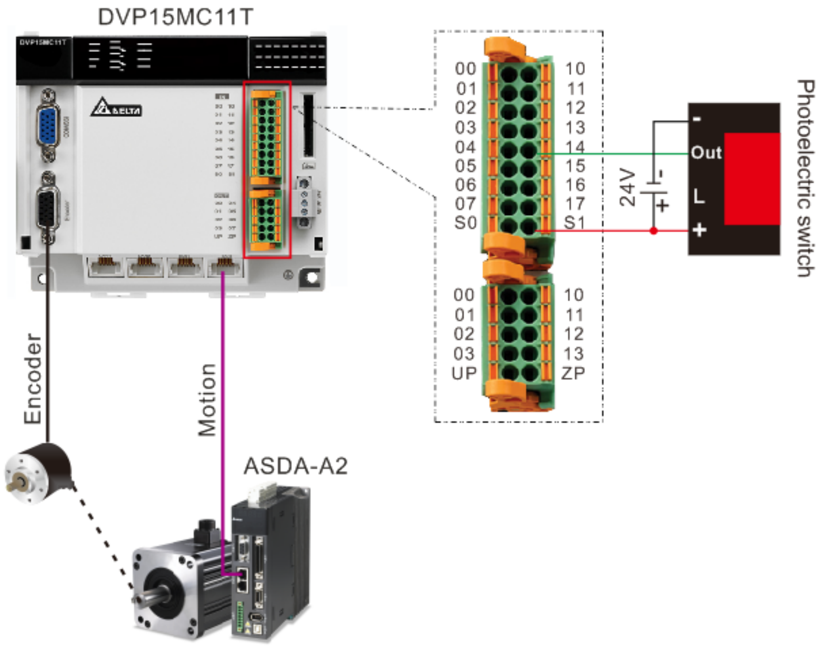
Servo gear ratio setting	Mechanism gear ratio setting
Unit Numerator: 128	Output rotations of gear: 3
Unit Denominator: 1	Input rotations of gear: 2
Pulses per rotation:10000	Units per output rotation: 1000 units/rotation

- When *Mode*=0 or 1 in DMC_TouchProbe, the captured position can be calculated according to the method mentioned above as well. In actual application, the position capture is generally performed by building an external encoder axis. When the number of pulses received at the external encoder interface of DVP15MC11T is 638, the position captured by DMC_TouchProbe is 95.4. The calculation formula: 638×（3×1000）÷（2×10000）=95.4. In the formula, 1000 is *Units per output rotation*, 2 is *Input rotations of gear*; 3 is *Output rotations of gear* and 10000 is *number of pulses per rotation*). When I0 changes from OFF to ON once, the position capture is performed once.

- **Wiring Figure**
  - **Mode 0 and mode 1**

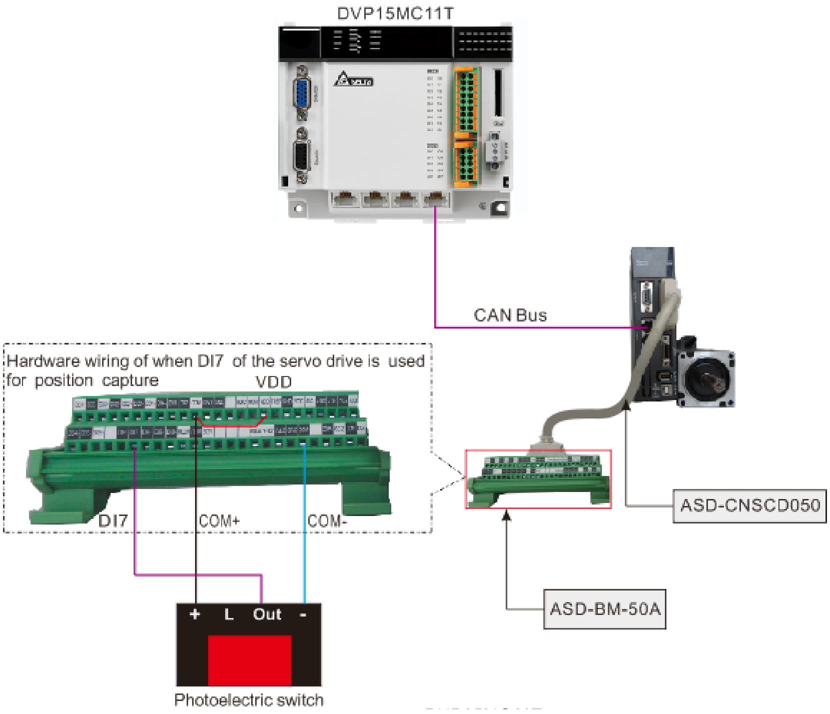
Mode 0: The external signal triggers I point of DVP15MC11T and the position capture is conducted through the rising edge of the input point specified by *TriggerInput*. The captured position is converted from the number of pulses the external encoder port of the controller receives through axis parameters.

Mode 1: The external signal triggers I point of DVP15MC11T and the position capture is conducted through the falling edge of the input point specified by *TriggerInput*. The captured position is converted from the number of pulses the external encoder port of the controller receives through axis parameters.



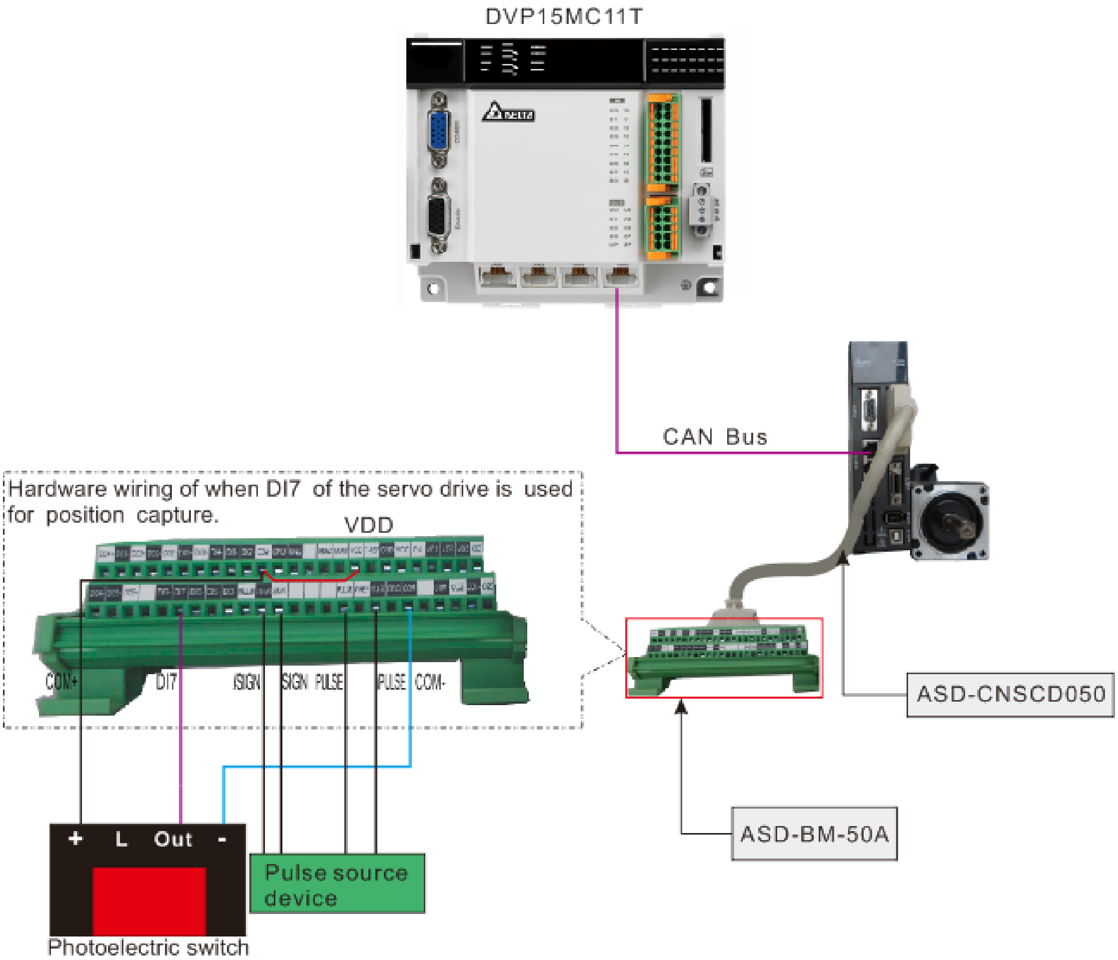
- **Mode 2**

The external signal triggers the high-speed input point: DI7 of the servo drive. The position captured is converted from the number of pulses which the servo motor feeds back to the servo drive through axis parameters.



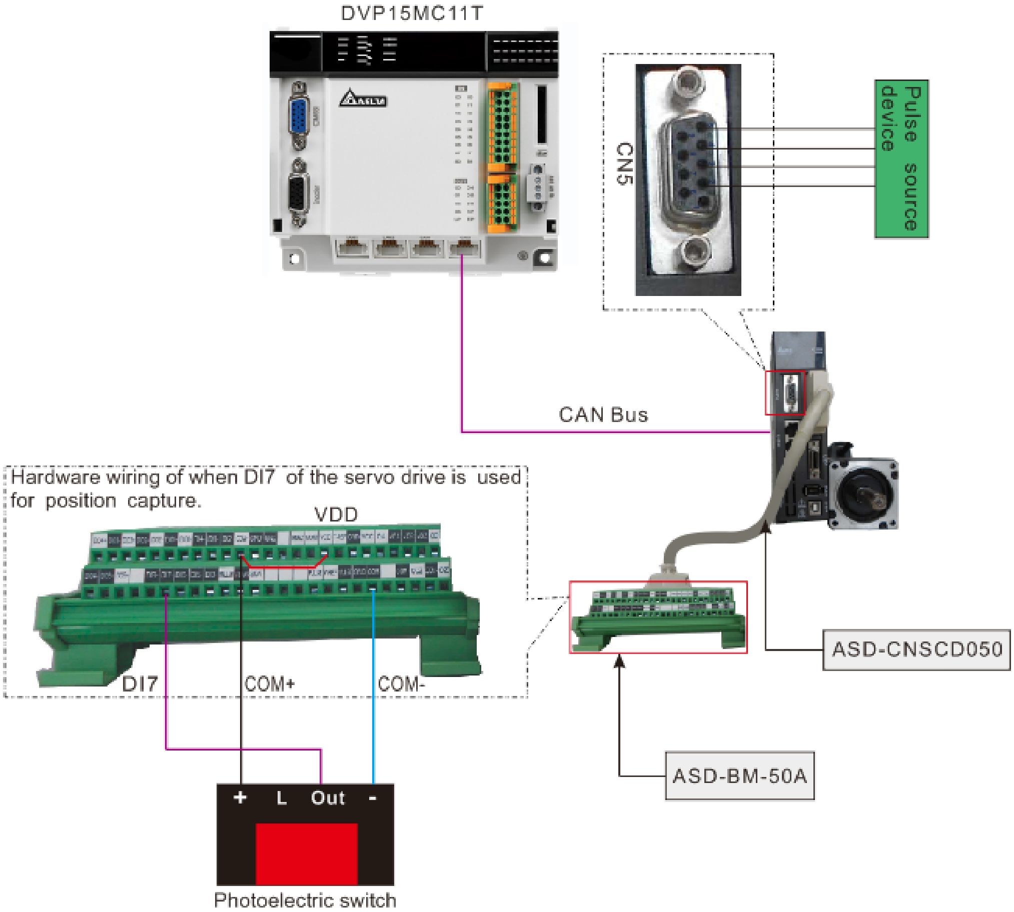
■ **Mode 3**

The external signal triggers the high-speed input point: DI7 of the servo drive. The captured position is converted from the number of pulses CN1 port of the servo drive receives through axis parameters.



■ **Mode 4**

The external signal triggers the high-speed input point: DI7 of the servo drive. The captured position is converted from the number of pulses CN5 port of the servo drive receives through axis parameters.

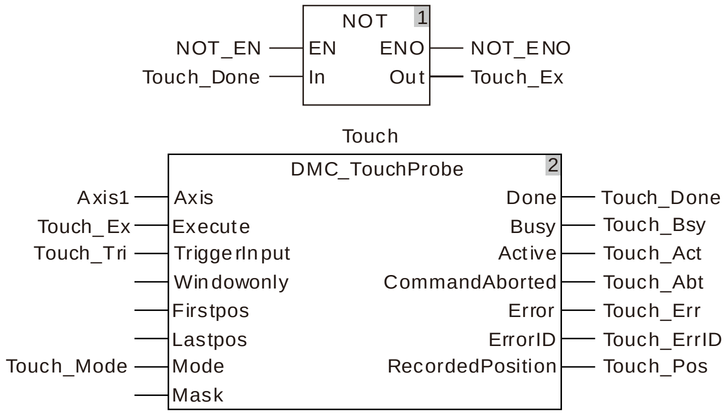


Programming Example 1

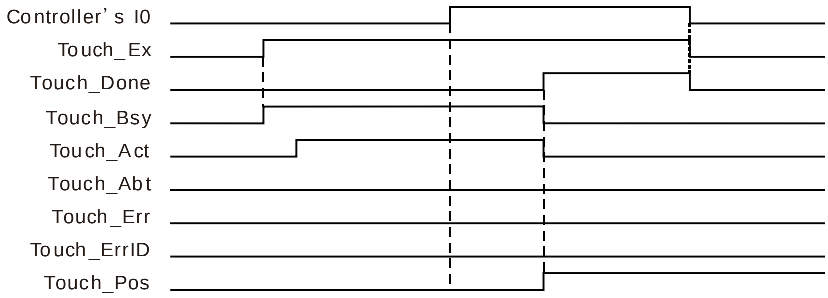
Capture the position of the external encoder axis by using the rising edge of I0 under mode 0.

1. The variable table and program

Variable name	Data type	Initial value
NOT_EN	BOOL	FALSE
NOT_ENO	BOOL	
Touch	DMC_TouchProbe	
Axis1	USINT	3
Touch_Ex	BOOL	FALSE
Touch_Tri	MC_Triggerinput	0
Touch_Mode	INT	0
Touch_Done	BOOL	
Touch_Bsy	BOOL	
Touch_Act	BOOL	
Touch_Abt	BOOL	
Touch_Err	BOOL	
Touch_ErrID	UINT	
Touch_Pos	LREAL	



2. **Timing Chart**



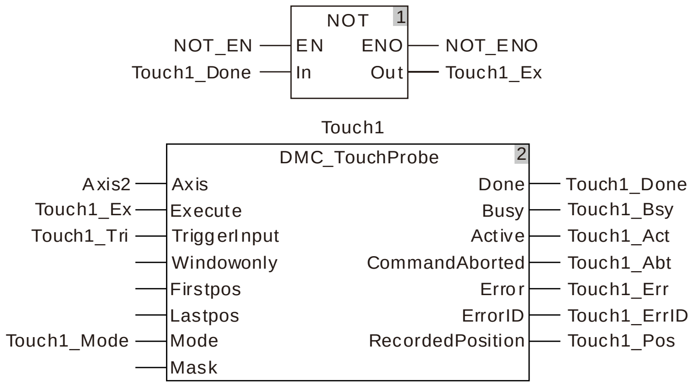
- When Touch_Ex changes from FALSE to TRUE, Touch_Bsy changes from FALSE to TRUE in the first cycle and Touch_Act changes from FALSE to TRUE in the second cycle.
- When the external signal triggers controller's I0, DMC_TouchProbe starts to execute. When Touch_Done changes from FALSE to TRUE, the position Touch_Pos outputs is converted from the number of pulses that the externam encoder port of the controller receives through axis parameters. Meantime Touch_Bsy and Touch_Act change from TRUE to FALSE. When Touch_Ex changes from TRUE to FALSE, Touch_Done changes from TRUE to FALSE and the position that Touch_Pos captures will not be cleared to 0

Programming Example 2

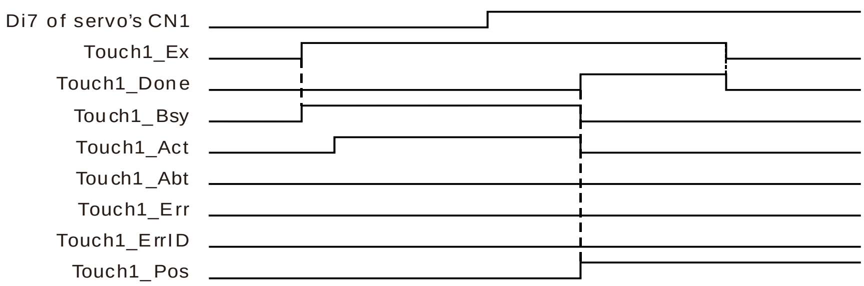
Capture the position converted from the number of pulses that the servo motor feeds back to the servo drive according to axis parameters when the external signal triggers DI7 of servo's CN1 under Mode 2.

1. The variable table and program

Variable name	Data type	Initial value
NOT_EN	BOOL	FALSE
NOT_ENO	BOOL	
Touch1	DMC_TouchProbe	
Axis2	USINT	1
Touch1_Ex	BOOL	FALSE
Touch1_Tri	MC_Triggerinput	
Touch1_Mode	INT	2
Touch1_Done	BOOL	
Touch1_Bsy	BOOL	
Touch1_Act	BOOL	
Touch1_Abt	BOOL	
Touch1_Err	BOOL	
Touch1_ErrID	UINT	
Touch1_Pos	LREAL	



2. Timing Chart

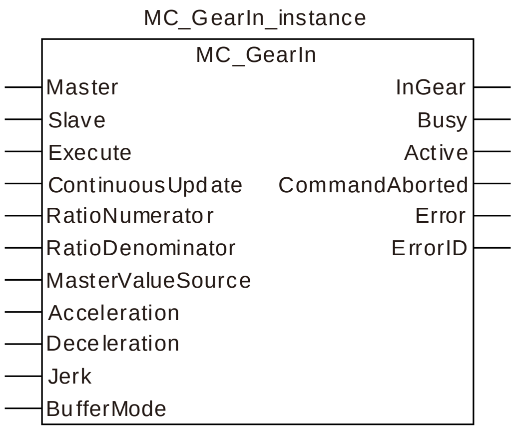


- ❖ When Touch1_Ex changes from FALSE to TRUE, Touch1_Bsy changes from FALSE to TRUE in the first cycle and Touch1_Act changes from FALSE to TRUE in the second cycle.
- ❖ When the execution of DMC_TouchProbe is finished after the external signal triggers DI7 of servo's CN1, Touch1_Done changes from FALSE to TRUE and Touch1_Pos outputs the position converted from the number of pulses which the servo motor feeds back to the servo drive according to the axis parameters. Meantime Touch1_Bsy and Touch1_Act change from TRUE to FALSE. When Touch1_Ex changes from TRUE to FALSE, Touch1_Done changes from TRUE to FALSE and the position that Touch1_Pos captures will not be cleared to 0.

11.4 Multi-axis Instructions

11.4.1 MC_GearIn

FB/FC	Explanation	Applicable model
FB	MC_GearIn is used for establishing the electronic gear relationship between two axes.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Master	Specify the number of the master axis which is to be controlled by the instruction	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Slave	Specify the number of the slave axis which is to be controlled by the instruction	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved	-	-	-
RatioNumerator	Gear ratio Numerator	LREAL	Positive number and negative number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
RatioDenominator	Gear ratio Denominator	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
MasterValueSource	Command source selection 0 : Command position of the master axis which the slave axis follows 1 : Actual position of the master axis which the slave axis follows	MC_Source	0:mcSetValue 1:mcActualValue (0)	When *Execute* changes from FALSE to TRUE
Acceleration	Specify the target acceleration. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Deceleration	Specify the target deceleration. (Unit: Unit/s^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of target acceleration and deceleration. (Unit: Unit/s^3)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
BufferMode	Specify the behavior when executing two instructions. 0: Aborting 1: Buffered	MC_Buffer_Mode	0 : mcAborting 1 : mcBuffered (0)	When *Execute* changes from FALSE to TRUE

Notes:

1. The execution of MC_GearIn is started when *Execute* changes from FALSE to TRUE. No matter whether the execution of the instruction is completed or not, the instruction can be re-executed when *Execute* changes from FALSE to TRUE once again. And meanwhile only *Velocity, Acceleration, Deceleration* and *Jerk* parameters will be effective again.
2. The slave axis specified by MC_GearIn instruction can execute other motion instruction while MC_GearIn is being executed. While other motion instruction aborts the MC_GearIn instruction, the gear relationship between the master axis and slave axis will disconnected. MC_Halt or MC_Stop can abort the motion of the slave axis.
3. Refer to section 10.2 for the relation among *Acceleration*, *Deceleration* and *Jerk*.
4. Refer to section 10.3 for details on *BufferMode*.

- **Output Parameters**

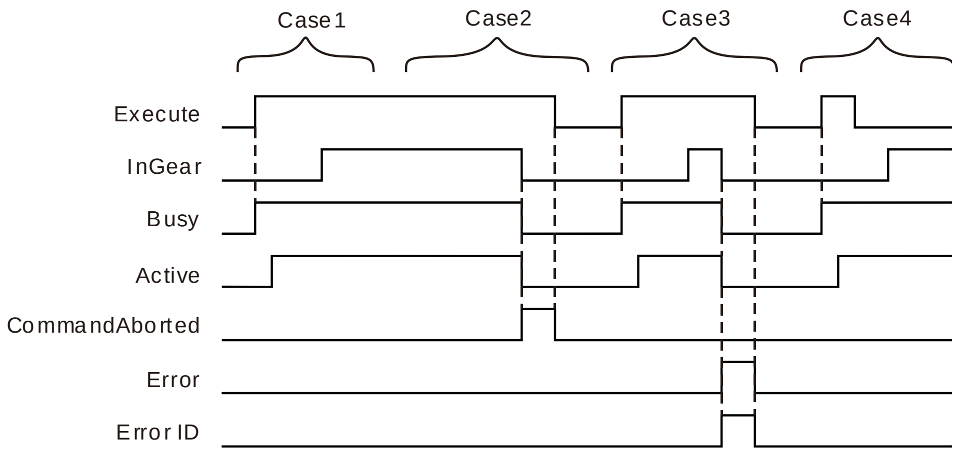
Parameter name	Function	Data type	Valid range
InGear	TRUE when the slave axis reaches the synchronous state.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2.	WORD	

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
InGear	◆ When the slave axis enters the synchronous state.	◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE ◆ *InGear* will change to FALSE immediately when the input parameter is modified after the synchronous state is reached and *Execute* changes from FALSE to TRUE once more. ◆ *InGear* will change to FALSE immediately when the input parameter is not modified after the instruction execution is finished and *Execute* changes from FALSE to TRUE once more. And in the next period, *InGear* changes

Name	Timing for changing to TRUE	Timing for changing to FALSE
		to TRUE.
Busy	◆ When *Execute* changes to TRUE	◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE
Active	◆ When the axis starts being controlled by the instruction	◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE
CommandAborted	◆ When the instruction execution is aborted by other motion instruction	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted by other instruction after *Execute* changes from TRUE to FALSE in the course of the instruction execution. One period later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Execute* changes from TRUE to FALSE

- **Output Update Timing Chart**



Case 1： *Busy* changes to TRUE as *Execute* changes from FALSE to TRUE. One period later, *Active* changes to TRUE. When the synchronous state is reached, *InGear* changes to TRUE and meanwhile *Busy* and *Active* remain TRUE.

Case 2： When *Execute* changes to TRUE and the slave axis is controlled by other instruction, MC_GearIn instruction is aborted by other instruction and *CommandAborted* changes to TRUE. Meanwhile *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3： When *Execute* changes from FALSE to TRUE and an error such as a parameter mistake occurs, *Error* changes to TRUE and *ErrorID* shows corresponding error codes. Meanwhile *InGear*, *Busy* and *Active* change to FALSE. As *Execute* changes from TRUE to FALSE, *Error* changes to FALSE.

Case 4： After *Execute* changes from TRUE to FALSE in the process of execution of MC_GearIn, *InGear* changes to TRUE and meanwhile *Busy* and *Active* remain TRUE.

- **Function**

1. MC_GearIn is used for establishing an electronic gear relationship between two axes. After the MC_GearIn instruction is executed, the slave axis performs the gear operation with the master axis according to the parameters, *RatioNumerator*, *RatioDenominator*, *Acceleration*, *Deceleration*, *Jerk*

and *BufferMode*. The master axis can be a real axis, virtual axis or encoder axis. The salve axis can be a real axis or virtual axis.

2. In the instruction execution, the slave axis need be enabled and the master axis can be enabled or disabled.
3. If the MC_GearIn instruction is executed when the e-gear relationship between two axes has not been built yet, the velocity of the slave axis will reach the target velocity according to the values of *RatioNumerator*, *RatioDenomenator*, *Acceleration*, *Deceleration* and *Jerk* specified by the instruction.

$$\text{Acceleration (or Deceleration) of Slave axis} = \text{Acceleration (or Deceleration) of Master axis} \times \frac{\text{RatioNumerator}}{\text{RatioDenominator}}$$

After the e-gear relationship between two axes has been built (when *InGear* of the instruction changes to TRUE), the relationship among the velocity of the slave axis, gear ratio numerator, gear ratio denominator and the velocity of the master axis is shown as below.

$$\text{Target velocity of Slave axis} = \text{Velocity of Master axis} \times \frac{\text{Gear ratio numeberator}}{\text{Gear ratio denominator}}$$

4. E-gear ratio

$$\text{E-gear ratio} = \frac{\text{RatioNumerator}}{\text{RatioDenominator}}$$

If the e-gear ratio is a positive number, the motion directions of the slave axis and master axis are same.

If the e-gear ratio is a negative number, the motion directions of the slave axis and master axis are opposite.

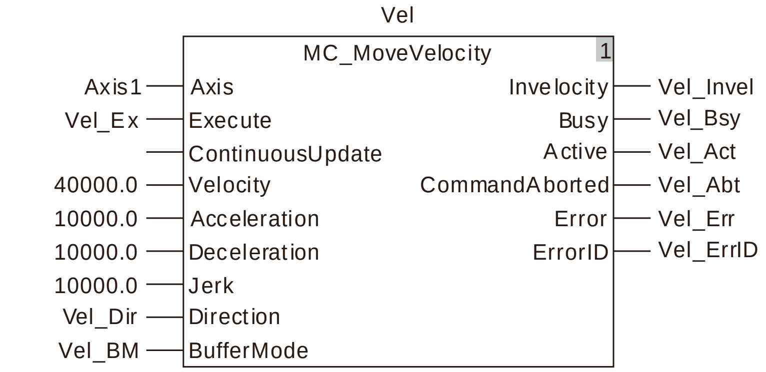
Programming Example

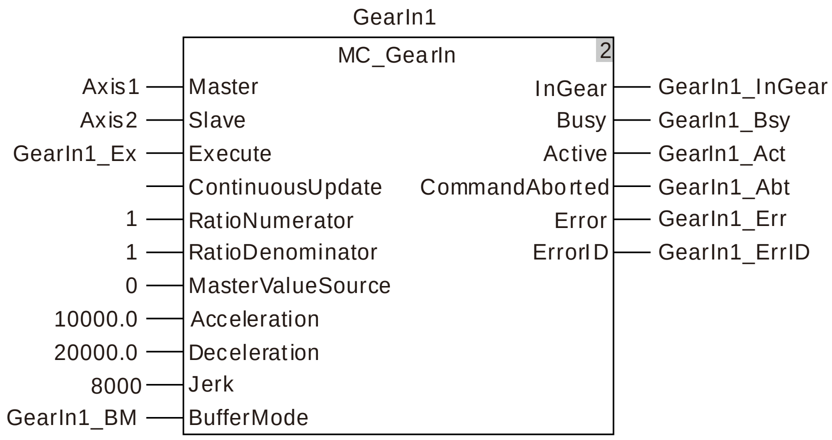
Below is the example of execution of MC_GearIn instructions.

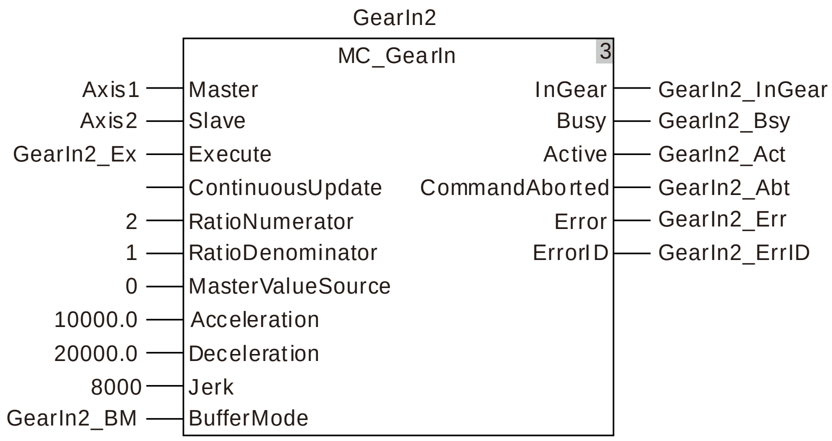
1. The variable table and program

Variable name	Data type	Initial value
Vel	MC_MoveVelocity	
Axis1	USINT	1
Axis2	USINT	2
Vel_Ex	BOOL	FALSE
Vel_Dir	MC_DIRECTION	1
Vel_BM	MC_Buffer_Mode	0
Vel_Invel	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	
GearIn1	MC_GearIn	
GearIn1_Ex	BOOL	FALSE
GearIn1_BM	MC_Buffer_Mode	0
GearIn1_InGear	BOOL	
GearIn1_Bsy	BOOL	
GearIn1_Act	BOOL	
GearIn1_Abt	BOOL	
GearIn1_Err	BOOL	
GearIn1_ErrID	WORD	

Variable name	Data type	Initial value
GearIn2	MC_GearIn	
GearIn2_Ex	BOOL	FALSE
GearIn2_BM	MC_Buffer_Mode	0
GearIn2_InGear	BOOL	
GearIn2_Bsy	BOOL	
GearIn2_Act	BOOL	
GearIn2_Abt	BOOL	
GearIn2_Err	BOOL	
GearIn2_ErrID	WORD	

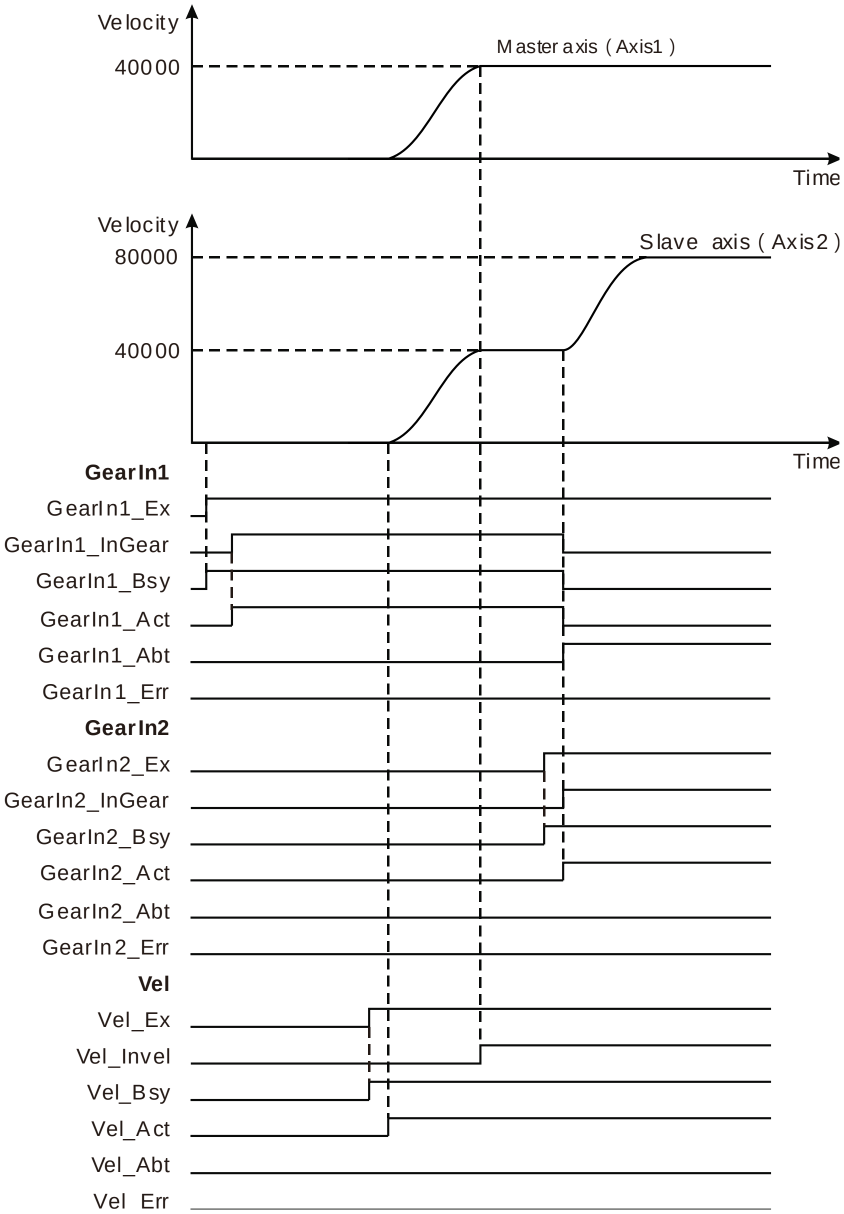






11

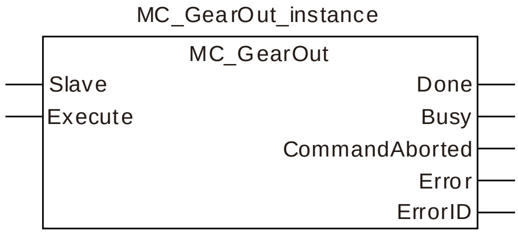
2. Motion Curve and Timing Charts:



- ❖ In GearIn1, the values of *RatioNumerator* and *RatioDenomenator* are both 1. GearIn1_Ex changes from FALSE to TRUE and meanwhile GearIn1 _Bsy changes to TRUE. One period later, GearIn1_InGear changes to TRUE and the e-gear relationship between the master axis and the slave axis is built.
- ❖ Vel_Ex changes from FALSE to TRUE after the e-gear relationship between the master axis and slave axis is built. One period later, Vel_Act changes to TRUE, the master axis performs the velocity instruction and the slave axis follows the master axis for motion.
- ❖ In GearIn2, the values of *RatioNumerator* and *RatioDenomenator* are 2 and 1 respectively. GearIn2_Ex changes from FALSE to TRUE and meanwhile GearIn2 _Bsy changes to TRUE. One period later, GearIn2_Act and GearIn1_Abt change to TRUE and the slave axis gets to the target velocity based on the values of RatioNumberator, Ratio Denomenator, MasterValueSource, Acceleration, Jerk and BufferMode specified by the GearIn2 instruction. Since the values of *RatioNumerator* and *RatioDenomenator* in GearIn2 are 2 and 1 respectively, the target velocity of the slave axis is twice that of the master axis. When GearIn2_InGear changes to TRUE, the velocity of the slave axis will be twice that of the master axis.

11.4.2 MC_GearOut

FB/FC	Explanation	Applicable model
FB	MC_GearOut is used for ending the established electronic gear relationship between the master axis and slave axis.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Slave	Specify the number of the slave axis which is to disconnect from the e-gear relationship.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-

Notes:

1. The slave axis will continue to move at the speed of disconnection if the slave axis disconnects from the e-gear relationship through the MC_GearOut instruction after the two axes has built the e-gear relationship through the MC_GearIn instruction.
2. The slave axis can execute other motion instructions after the MC_GearOut instruction execution is completed.
3. The relationship between the master axis and slave axis is disconnected through the MC_GearOut instruction. To stop the motion of the slave axis, MC_Halt or MC_Stop instruction can be used.

- **Output Parameters**

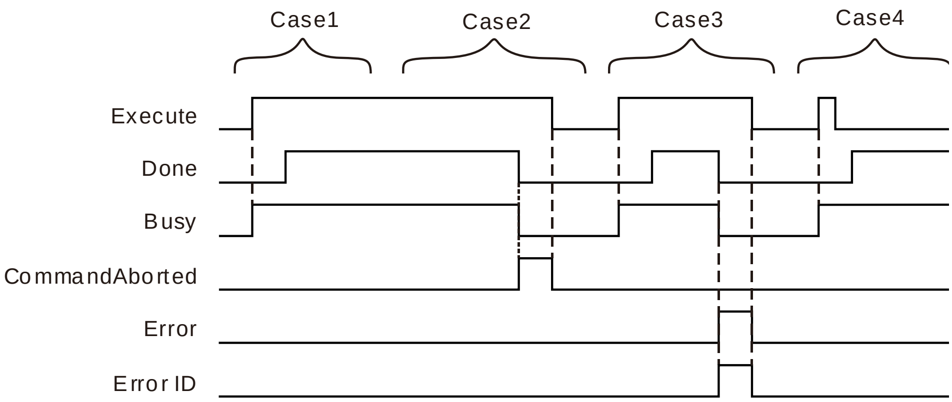
Parameter name	Function	Data type	Valid range
Done	TRUE when the e-gear relationship between the slave axis and master axis is disconnected and the MC_GearOut instruction is controlling the axes.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2.	WORD	

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the electronic gear relationship between the slave axis and master axis is disconnected	◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE
Busy	◆ When *Execute* changes to TRUE	◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE

Name	Timing for changing to TRUE	Timing for changing to FALSE
CommandAborted	◆ When the instruction execution is aborted by other motion instruction	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted by other instruction after *Execute* changes from TRUE to FALSE in the course of the instruction execution. One period later, *CommandAborted* changes to FALSE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1： *Busy* changes to TRUE as *Execute* changes from FALSE to TRUE. One period later, *Done* changes to TRUE. *Busy* and *Done* remain TRUE after *Execute* changes from TRUE to FALSE.

Case 2： If the MC_GearOut instruction is aborted by other instruction as *Execute* changes to TRUE, *CommandAborted* changes to TRUE and meanwhile *Busy* and *Done* change to FALSE. *CommandAborted* changes to FALSE as *Execute* changes from TRUE to FALSE.

Case 3： When an error occurs (e.g. the axis is disabled), *Error* changes to TRUE and *ErrorID* shows corresponding error codes after *Execute* changes from FALSE to TRUE. Meanwhile, *Busy* and *Done* change to FALSE. As *Execute* changes from TRUE to FALSE, *Error* changes to FALSE.

Case 4： *Execute* changes from TRUE to FALSE before a period is reached during execution of the MC_GearOut instruction. *Done* changes to TRUE and *Busy* remains TRUE as a period is reached.

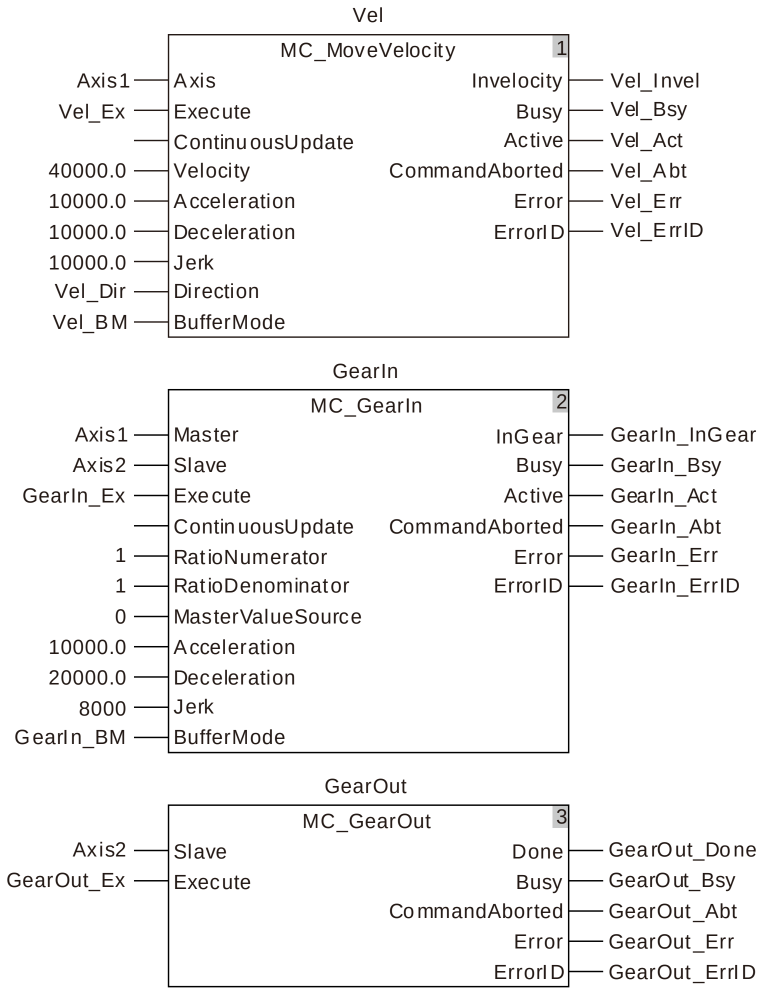
Programming Example

Below is the example of the execution of the MC_GearOut instruction.

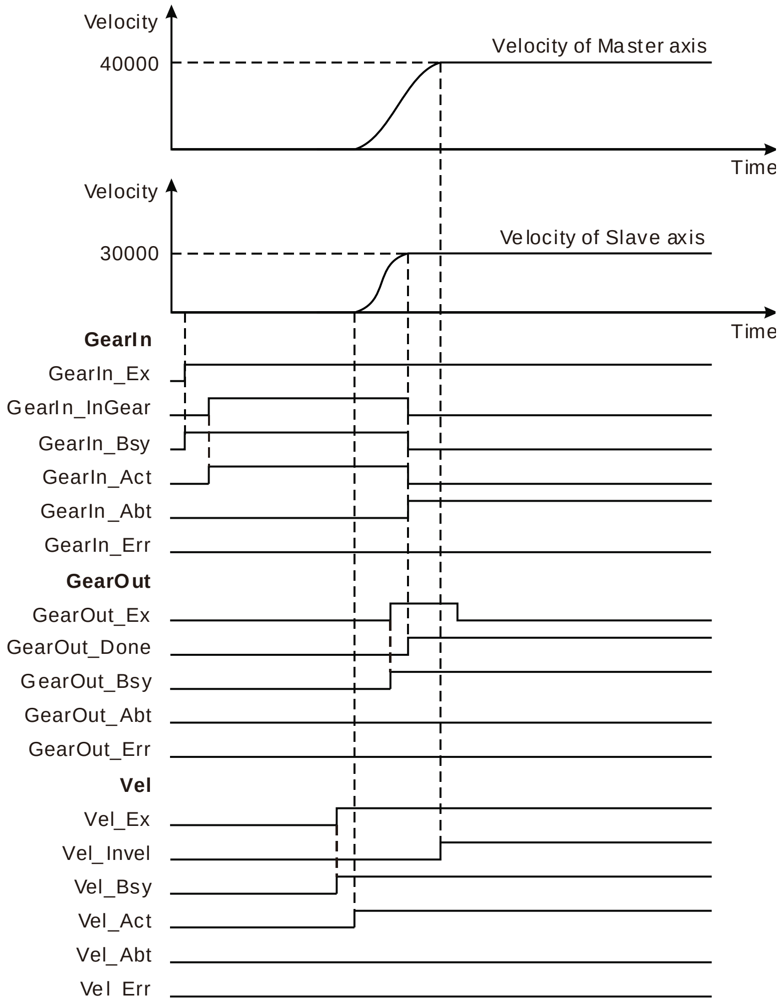
1. The variable table and program

Variable name	Data type	Initial value
Vel	MC_MoveVelocity	
Axis1	USINT	1
Axis2	USINT	2
Vel_Ex	BOOL	FALSE
Vel_Dir	MC_DIRECTION	1
Vel_BM	MC_Buffer_Mode	0
Vel_Invel	BOOL	

Variable name	Data type	Initial value
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	
GearIn	MC_ GearIn	
GearIn_Ex	BOOL	FALSE
GearIn_BM	MC_Buffer_Mode	0
GearIn_InGear	BOOL	
GearIn_Bsy	BOOL	
GearIn_Act	BOOL	
GearIn_Abt	BOOL	
GearIn_Err	BOOL	
GearIn_ErrID	WORD	
GearOut	MC_ GearOut	
GearOut_Ex	BOOL	FALSE
GearOut_Done	BOOL	
GearOut_Bsy	BOOL	
GearOut_Act	BOOL	
GearOut_Abt	BOOL	
GearOut_Err	BOOL	
GearOut_ErrID	WORD	


Vel
MC_MoveVelocity
1
Axis1
Axis
Vel_Ex
Execute
ContinuousUpdate
40000.0
Velocity
10000.0
Acceleration
10000.0
Deceleration
10000.0
Jerk
Vel_Dir
Direction
Vel_BM
BufferMode
Invelocity
Vel_Invel
Busy
Vel_Bsy
Active
Vel_Act
CommandAborted
Vel_Abt
Error
Vel_Err
ErrorID
Vel_ErrID
GearIn
MC_GearIn
2
Axis1
Master
Axis2
Slave
GearIn_Ex
Execute
ContinuousUpdate
1
RatioNumerator
1
RatioDenominator
0
MasterValueSource
10000.0
Acceleration
20000.0
Deceleration
8000
Jerk
GearIn_BM
BufferMode
InGear
GearIn_InGear
Busy
GearIn_Bsy
Active
GearIn_Act
CommandAborted
GearIn_Abt
Error
GearIn_Err
ErrorID
GearIn_ErrID
GearOut
MC_GearOut
3
Axis2
Slave
GearOut_Ex
Execute
Done
GearOut_Done
Busy
GearOut_Bsy
CommandAborted
GearOut_Abt
Error
GearOut_Err
ErrorID
GearOut_ErrID

2. Curve and Timing Charts:



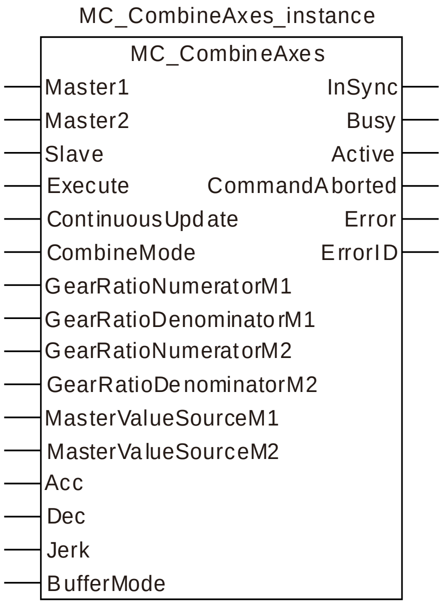
❖ As GearIn_Ex changes from FALSE to TRUE, GearIn_Bsy changes to TRUE. And one period later, GearIn_InGear changes to TRUE and the gear relationship between the master axis and slave axis is built.

❖ After the gear relationship between the two axes is built, Vel_Ex changes from FALSE to TRUE. One period later, Vel_Act changes to TRUE. The master axis executes the velocity instruction and the slave axis moves by following the motion of the master axis.

❖ While the master axis is executing the velocity instruction, GearOut_Ex changes from FALSE to TRUE and GearOut_Bsy changes to TRUE. One period later, GearOut_Done and GearIn_Abt change to TRUE. And the slave axis will continue to move at the current speed.

11.4.3 MC_CombineAxes

FB/FC	Explanation	Applicable model
FB	MC_CombineAxes outputs the sum or difference of the position variations of two master axes as the slave position variation.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Master1	The position source of axis 1	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Master2	The position source of axis 2	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Slave	The controlled slave	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	--
ContinuousUpdate	Reserved	-	-	-
CombineMode	Combining method selection. 0: Sum of two master axis position variations 1: Difference of two master axis position	MC_CombineMode	0: mcAddAxes、1: mcSubAxes (0)	When *Execute* changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
	variations			
GearRatioNumeratorM1	Specify the master axis1 gear ratio numerator.	LREAL	Positive number or negative number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
GearRatioDenominatorM1	Specify the master axis1 gear ratio denominator.	LREAL	Positive number or negative number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
GearRatioNumeratorM2	Specify the master axis2 gear ratio numerator.	LREAL	Positive number or negative number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
GearRatioDenominatorM2	Specify the master axis2 gear ratio denominator.	LREAL	Positive number or negative number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
MasterValueSourceM1	Specify the synchronization position source of master axis 1. 0 :Command position 1：Actual position	MC_SOURCE	0:mcSetValue 1:mcActualValue (0)	When *Execute* changes from FALSE to TRUE
MasterValueSourceM2	Specify the synchronization position source of master axis 2. 0 :Command position 1：Actual position	MC_SOURCE	0:mcSetValue 1:mcActualValue (0)	When *Execute* changes from FALSE to TRUE
Acc	Specify the acceleration for the slave axis.	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Dec	Specify the deceleration for the slave axis.	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Specify the change rate of the acceleration for the slave axis.	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
BufferMode	Specify the behavior when executing two instructions. 0：Aborted 1：Buffered	MC_Buffer_Mode	0 : mcAborting 1 : mcBuffered (0)	When *Execute* changes from FALSE to TRUE

Notes:

1. The instruction execution starts when *Execute* changes from FALSE to TRUE. When *Execute* changes from FALSE to TRUE again no matter whether the instruction execution is completed or not, the instruction cannot be re-executed and the previous setting values will be kept.
2. Refer to section 10.2 for the relation among *Position*, *Velocity*, *Acceleration* and *Jerk*.
3. Refer to section 10.3 for the details about *BufferMode*.

- **Output Parameters**

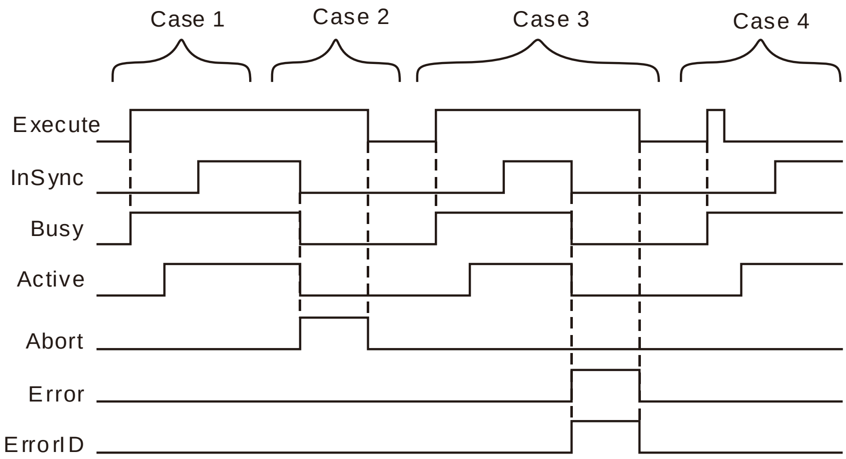
Parameter name	Function	Data type	Valid range
InSync	TRUE when the slave axis has completed the synchronization action.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to section 12.2.	WORD	

- **Output Update Timing**

Parameter Name	Timing for changing to TRUE	Timing for changing to FALSE
InSync	◆ When the slave axis completes the synchronization action.	◆ When *Error* changes to TRUE. ◆ When *CommandAborted* changes to TRUE.
Busy	◆ When *Execute* is TRUE.	◆ When *CommandAborted* changes to TRUE. ◆ When *Error* changes to TRUE.
Active	◆ When the instruction starts to control the axis.	◆ When *CommandAborted* changes to TRUE. ◆ When *Error* changes to TRUE.
CommandAborted	◆ When this instruction execution is aborted by other motion control instruction.	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

11

- **Output Update Timing Chart**



Case 1: When *Execute* changes from FALSE to TRUE, *Busy* changes to TRUE. One cycle later, *Active* changes to TRUE. When the slave axis has synchronized with the two master axes, *InSync* changes to TRUE and *Busy* and *Active* remain TRUE.

Case 2: When *Execute* is TRUE, *Busy* is TRUE and *Active* is TRUE. When the slave have synchronized with the two master axes, *InSync* is TRUE. At the moment, the instruction is aborted by another instruction, *CommandAborted* changes to TRUE and meanwhile *Invelocity*, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3: When *Execute* changes from FALSE to TRUE, *Error* changes to TRUE and *ErrorID* shows corresponding error codes when an error occurs such as axis alarms or offline. Meanwhile, *InSync*, *Busy* and *Active* change to FALSE. When *Execute* changes from TRUE to FALSE, *Error* changes to FALSE.

Case 4: The instruction is still executed and the states of *Busy* and *Active* do not change after *Execute* changes from TRUE to FALSE during execution of the instruction. When the slave axis has been synchronized with the two master axes, *InSync* changes to TRUE and *Busy* and *Active* remain TRUE.

- **Function**

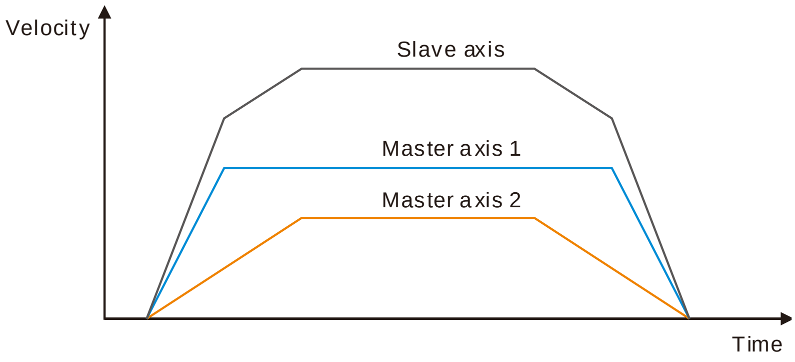
MC_CombineAxes outputs the sum or difference of the position variations of two master axes as the slave position variation.

- **Combine modes: Addition or Subtraction**

  The addition or subtraction of the position variations of master axis 1 and master axis 2 are conducted and the calculation result is output as slave axis position variation.

- ***CombineMode* is set to 0**

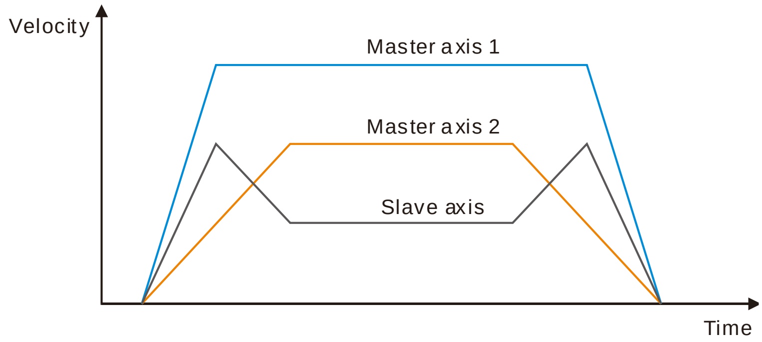
$$\textbf{Position variation of Slave axis} = \text{Position variation of Master axis1} \times \frac{\text{GearRatioNumeratorM1}}{\text{GearRatioDenominatorM1}} + \text{Position variation of Master axis2} \times \frac{\text{GearRatioNumeratorM2}}{\text{GearRatioDenominatorM2}}$$



- ***CombineMode* is set to 1**

Position variation of Slave axis $=$

$$= \text{Position variation of Master axis1} \times \frac{\text{GearRatioNumeratorM1}}{\text{GearRatioDenominatorM1}} - \text{Position Variation of Master axis2} \times \frac{\text{GearRatioNumeratorM2}}{\text{GearRatioDenominatorM2}}$$



- The master gear ratio numerator and denominator are the factors to adjust the position variations of two master axes. See the formula above.
- *MasterValueSource* can be set to 0 (command position) and 1 (actual position) so as to specify the source of the position variation. If the value is set to 0, add up the master axis command position variations. If the value is set to 1, subtract one master axis actual position variation from another master axis actual position variation.
- The *Acc*, *Dec* and *Jerk* indicate that the master axis has been in motion before the instruction is executed. If the instruction is executed at the moment, the slave axis will speed up or down according to the set acceleration, deceleration and jerk so as to realize the synchronization with the master position variations. When the synchronization is achieved, *InSync* is TRUE and the instruction execution is completed.
- Use other motion instruction (such as MC_Stop instruction) for the control over the slave axis so as to end the master-slave axis relationship in the instruction. Set the value of *BufferMode* of other motion instruction which has the *Buffermode* parameter to 0 in order to abort the MC_CombineAxes instruction and disconnect the master-slave axis relationship.
- If the master axis gear ratio is to be switched during the motion, use another MC_CombineAxes instruction to abort the MC_CombineAxes instruction which is being executed.

Programming Example

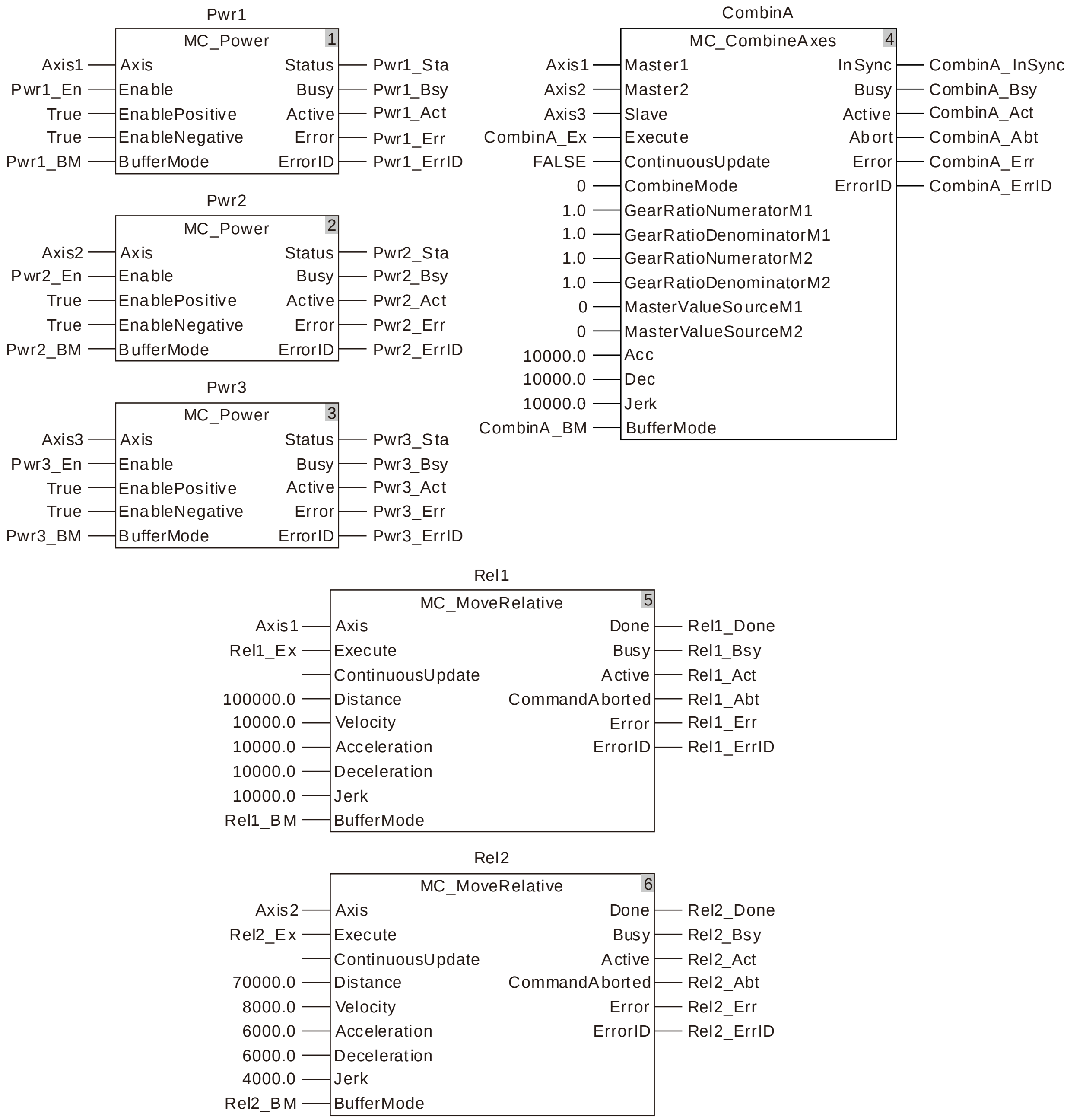
The example of executing the MC_CombineAxes instruction is described as below.

1. The variable table and program

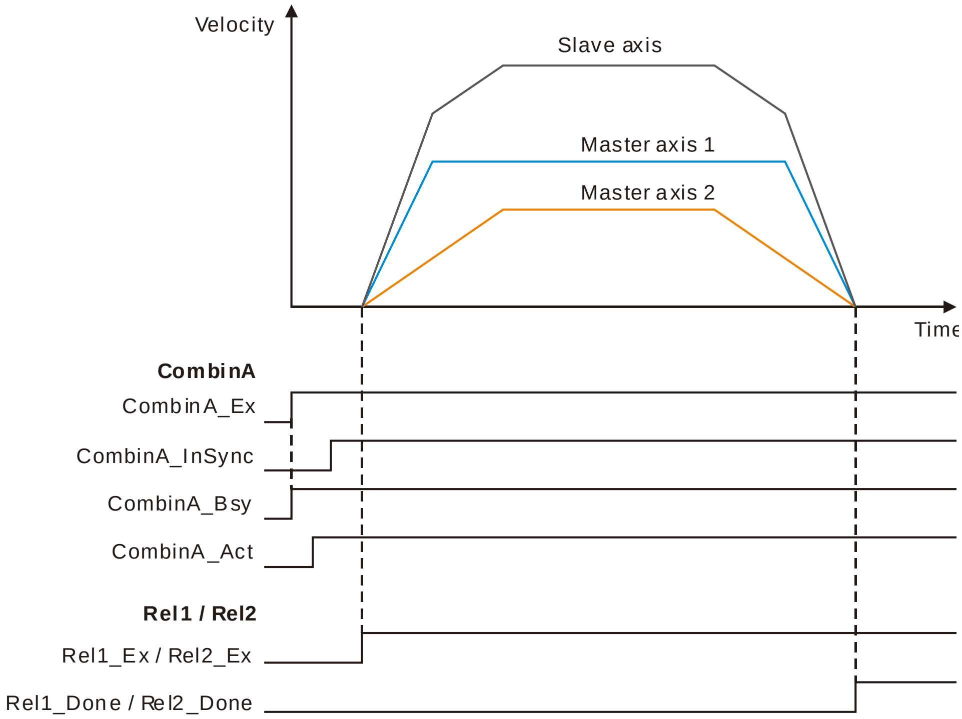
Variable name	Data type	Initial value
Pwr1	MC_Power	
Axis1	USINT	1
Pwr1_BM	MC_Buffer_Mode	1
Pwr1_Sta	BOOL	

Variable name	Data type	Initial value
Pwr1_Bsy	BOOL	
Pwr1_Act	BOOL	
Pwr1_Err	BOOL	
Pwr1_ErrID	WORD	
Pwr2	MC_Power	
Axis2	USINT	1
Pwr2_BM	MC_Buffer_Mode	1
Pwr2_Sta	BOOL	
Pwr2_Bsy	BOOL	
Pwr2_Act	BOOL	
Pwr2_Err	BOOL	
Pwr2_ErrID	WORD	
Pwr3	MC_Power	
Axis3	USINT	1
Pwr3_BM	MC_Buffer_Mode	1
Pwr3_Sta	BOOL	
Pwr3_Bsy	BOOL	
Pwr3_Act	BOOL	
Pwr3_Err	BOOL	
Pwr3_ErrID	WORD	
CombinA	MC_CombineAxes	
CombinA_Ex	BOOL	FALSE
CombinA_BM	MC_Buffer_Mode	1
CombinA_InSync	BOOL	
CombinA_Bsy	BOOL	
CombinA_Act	BOOL	
CombinA_Abt	BOOL	
CombinA_Err	BOOL	
CombinA_ErrID	WORD	
Rel1	MC_MoveRelative	
Rel1_Ex	BOOL	FALSE
Rel1_Dir	MC_DIRECTION	1
Rel1_BM	MC_Buffer_Mode	0
Rel1_Done	BOOL	
Rel1_Bsy	BOOL	
Rel1_Act	BOOL	
Rel1_Abt	BOOL	
Rel1_Err	BOOL	
Rel1_ErrID	WORD	
Rel2	MC_MoveRelative	
Rel2_Ex	BOOL	FALSE
Rel2_Dir	MC_DIRECTION	1
Rel2_BM	MC_Buffer_Mode	0
Rel2_Done	BOOL	
Rel2_Bsy	BOOL	

Variable name	Data type	Initial value
Rel2_Act	BOOL	
Rel2_Abt	BOOL	
Rel2_Err	BOOL	
Rel2_ErrID	WORD	



2. **Motion Curve and Timing Chart**



❖ When CombinA_Ex change from FALSE to TRUE, the execution of the MC_CombineAxes instruction starts. After a period of time, the instruction execution succeeds, CombinA_InSync changes to TRUE and three axes can go into the synchronized motion as required. At the moment, *Execute*s of MC_MoveRelatives for the two master axes are set to TRUE and then the two master axes start to run and meanwhile the slave also starts to run according to the sum of two master axis position variations. The slave axis position variation is the sum of the two master axis position variations in the unit time.

After the instructions executed for the master axes are completed, the three axes remain in the synchronized state. To abort the synchronization state of the three axes, use MC_Stop instruction to abort the slave axis motion and disconnect the synchronization state.

11.4.4 Introduction of Electronic Cam

The cam is the component with the curve profile or grooves. It transmits the motion to the follower near its edge and the rack will turn periodically following the follower. The cam mechanism consists of a cam, follower and rack. The following figure shows the cam profile made up of point A, B, C, and D. AB' is a follower which is connected to the rack. δ4 is an inner angle of repose; δ2 is an external angle of repose. The radius of the base circle is r0 and S is the cam curve.

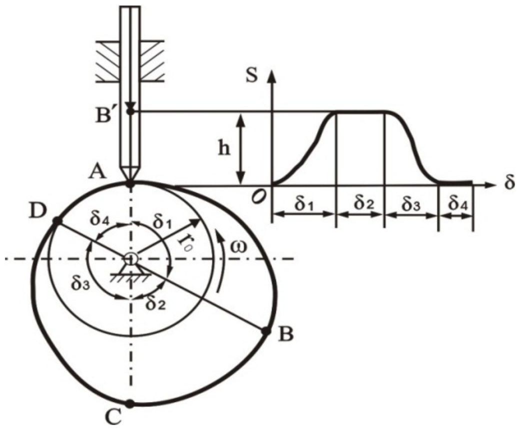


Figure 11.4.4.1

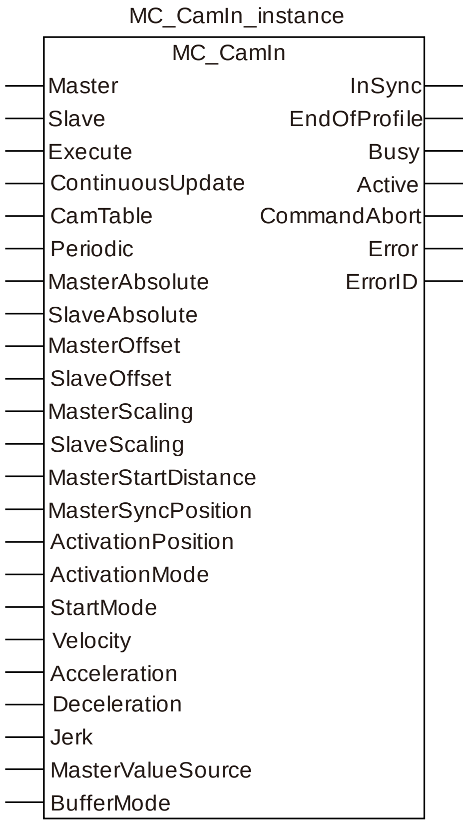
The electronic cam is an analog cam of the mechanical cam through applying computer technology. Compared with the mechanical cam, the electronic cam has many advantages of being easy to design and modify; cost saving; higher efficiency and preciseness. Because the electronic cam is an analog cam, the defects of a mechanical cam like being easy to be damaged and not fit for high-speed rotation and transmission can be avoided for the electronic cam.

DVP15MC11T controller supports the function of the electronic cam. User can edit the cam curve in the corresponding cam editor software.

The cam curve need be called in the motion control program after being edited. The motion control program can call the cam curve by using the MC_CamIn instruction.

11.4.5 MC_CamIn

FB/FC	Explanation	Applicable model
FB	MC_CamIn is used to build the cam relationship between two axes according to the set parameters.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Master	Specify the number of the master axis in the electronic cam operation.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Slave	Specify the number of the slave axis in the electronic cam operation.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-
ContinuousUpdate	Reserved			
CamTable	Specify the cam table used for building a cam relationship between the master axis and slave axis	USINT	1~64 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Periodic	Specify whether to execute the specified cam table periodically or just one period.	BOOL	TRUE or FALSE (FALSE)	When *Execute* changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
MasterAbsolute	Specify the position mode of the master axis. TRUE: Absolute position FALSE: Relative position	BOOL	TRUE or FALSE (FALSE)	When *Execute* changes from FALSE to TRUE
SlaveAbsolute	Specify the position mode of the slave axis. TRUE: Absolute position FALSE: Relative position	BOOL	TRUE or FALSE (FALSE)	When *Execute* changes from FALSE to TRUE
MasterOffset	Specify how many units the master axis position shifts by. (Unit: Unit)	LREAL	Negative number, positive number and 0 (0)	When *Execute* changes from FALSE to TRUE
SlaveOffset	Specify how many units the slave axis position shifts by. (Unit: Unit)	LREAL	Negative number, positive number and 0 (0)	When Execute changes from FALSE to TRUE
MasterScaling	Specify the scaling of the master axis position.	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
SlaveScaling	Specify the scaling of the slave axis position.	LREAL	Positive number or negative number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
MasterStartDistance	Reserved			
MasterSyncPosition	Reserved			
ActivationPosition	Specify the position of the master axis as the engagement begins. In other words, when the master axis passes the position, the slave axis starts to perform the engagement action.	LREAL	Negative number, positive number and 0 (0)	When *Execute* changes from FALSE to TRUE
ActivationMode	Specify the mode of the position where to start the engagement	MC_ACTIVATION_MODE	0: mcRelative (Relative axis position) 1: mcAbsolute (Absolute axis position) 2: mcPhase_Axis (Absolute axis phase) 3: mcPhase_CAM (Absolute cam phase) (0)	When Execute changes from FALSE to TRUE
StartMode	Specify the way how the slave axis performs the engagement action.	MC_START_MODE	0: mcRampInShortest (The shortest way) 1: mcRampInPositive (Positive direction) -1: mcRampInNegative (Negative direction) (0)	When *Execute* changes from FALSE to TRUE
Velocity	Specify the maximum stacking velocity of the slave axis during the period when	LREAL	Positive number (The variable value must	When *Execute* changes from

Parameter name	Function	Data type	Valid range (Default)	Validation timing
	the slave axis performs the engagement action. (Unit: Unit/second)		be set)	FALSE to TRUE
Acceleration	Specify the maximum acceleration of the slave axis during the period when the slave axis performs the engagement action. (Unit: Unit/second^2)	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Deceleration	Specify the maximum deceleration of the slave axis during the period when the slave axis performs the engagement action. (Unit: Unit/second^2.	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Jerk	Reserved	-	-	-
MasterValueSource	Specify the type of the master axis position in the electronic cam calculation.	MC_SOURCE	0:mcSetValue 1:mcActualValue (0)	When Execute changes from FALSE to TRUE
BufferMode	Specify the behavior when executing two instructions.	MC_Buffer_Mode	0: mcAborting 1: mcBuffered (0)	When *Execute* changes from FALSE to TRUE

Note:

1. The MC_CamIn instruction execution starts when *Execute* changes from FALSE to TRUE. Changing *Execute* from TRUE to FALSE does not influence the instruction execution during execution of the instruction.
2. Changing *Execute* from FALSE to TRUE again does not influence the instruction execution during execution of the instruction. The instruction will keep going in the previous way.
3. Refer to Section 10.3 for details on *BufferMode*.

- **Output Parameters**

Parameter name	Function	Data type	Valid range
InSync	TRUE when the master axis and slave axis move synchronously based on the cam curve.	BOOL	TRUE / FALSE
EndOfProfile	TRUE when the cam motion reaches the end point.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Active	TRUE when the axis is being controlled.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to the section 12.2.	WORD	

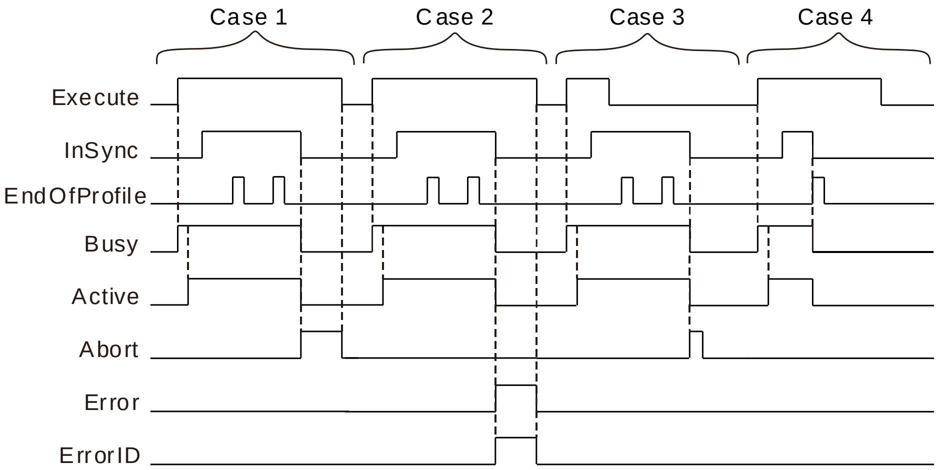
- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
InSync	◆ When the slave axis and master axis	◆ When the cam relationship between the slave axis and master axis is

Name	Timing for changing to TRUE	Timing for changing to FALSE
	are synchronous in the cam motion.	disconnected. ◆ When the acyclic cam motion is performed (*Periodic*=FALSE) and *EndOfProfile* changes to TRUE ◆ When *CommandAborted* changes to TRUE ◆ When *Error* changes to TRUE
EndOfProfile	◆ When the cam motion reaches the end point in the cam table.	◆ One period after *EndOfProfile* changes to TRUE
Busy	◆ When *Execute* changes to TRUE	◆ When the acyclic cam motion is performed (*Periodic*=FALSE) and *EndOfProfile* changes to TRUE ◆ When *Error* changes to TRUE ◆ When *CommandAborted* changes to TRUE
Active	◆ When the instruction starts to control axes	◆ When the acyclic cam motion is performed (*Periodic*=FALSE) and *EndOfProfile* changes to TRUE ◆ When *Error* changes to TRUE ◆ When *CommandAborted* changes to TRUE
CommandAborted	◆ When the instruction execution is aborted by other motion instruction	◆ When *Execute* changes from TRUE to FALSE ◆ *CommandAborted* is set to TRUE when the instruction is aborted by other instruction after *Execute* changes from TRUE to FALSE in the course of the instruction execution. One period later, *CommandAborted* changes to FALSE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal	◆ When *Execute* changes from TRUE to FALSE

11

- **Output Update Timing Chart**



Case 1： *Busy* changes to TRUE as *Execute* changes from FALSE to TRUE. And one period later, *Active* changes to TRUE. When the slave axis and master axis are in the synchronous motion, *InSync* changes from FALSE to TRUE. When the final point of the cam cycle is reached, *EndOfProfile* changes from FALSE to TRUE and changes to FALSE one cycle later. When the cam relationship between the slave axis and master axis is disconnected (e.g. by executing the MC_CamOut instruction), *CommandAborted* changes from FALSE to TRUE and *InSync, Busy* and *Active* all change from TRUE to FALSE. After that, *CommandAborted* changes from TRUE to FALSE as *Execute* changes from TRUE to FALSE.

Case 2： As an error occurs in the execution of the instruction, *Error* changes from FALSE to TRUE, *ErrorID* shows corresponding error codes and *InSync*, *Busy* and *Active* all change from TRUE to FALSE. After that, *Error* changes from TRUE to FALSE and the value of *ErrorID* changes to 0 as *Execute* changes from TRUE to FALSE.

Case 3： The instruction execution still continues after *Execute* changes from TRUE to FALSE during execution of the instruction. The timing for changing the state of *InSync*, *EndOfProfile, Busy* and *Active* is consistent with what state they are in as *Execute* is TRUE. After that, *InSync*, *Busy* and *Active* all change from TRUE to FALSE after the cam relationship between the slave axis and master axis is disconnected. Meanwhile CommandAborted changes from FALSE to TRUE and changes to FALSE one cycle later.

Case 4： If the cam motion is performed in the acyclic way (*Periodic*=FALSE), *EndOfProfile* changes from FALSE to TRUE when the end point of the cam cycle is reached. Meanwhile *InSync*, *Busy* and *Active* all change from TRUE to FALSE and *EndOfProfile* changes from TRUE to FALSE one cycle later.

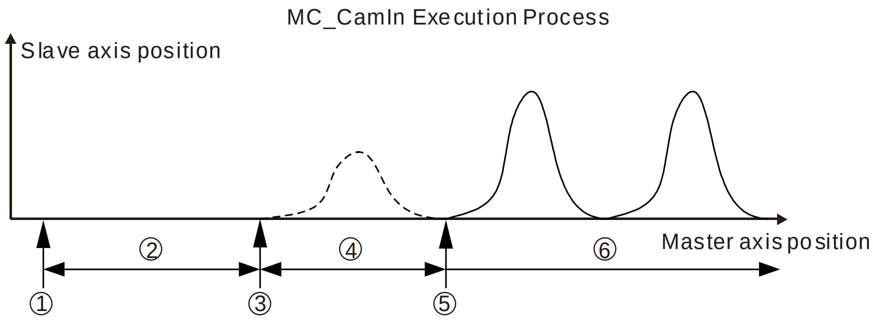
- **Function**

The *MC_CamIn* instruction is used for making the slave axis and master axis move synchronously according to the planned cam relationship. The *MC_CamOut* instruction is used for disconnecting the cam relationship between the two axes.

About MC_CamIn Instruction

MC_CamIn Execution Process

The MC_CamIn execution process figure:



Stage 1: Trigger and execute the MC_CamIn instruction.
Stage 2: Wait for the start of the engagement.
Stage 3: The slave axis starts to perform the engagement action as the master axis reaches the position where the engagement starts.
Stage 4: The engagement is ongoing.
Stage 5: The master axis and slave axis achieve the synchronization as the engagement is completed.
Stage 6: The master axis and slave axis are in the synchronous motion.

Stage 1: Trigger and execute the MC_CamIn instruction.
The *MC_CamIn* instruction is executed at this time and then the slave will enter the state of waiting for the start of the engagement immediately.

NOTE: If *ActivationPosition*=0 and *ActivationMode*=0 (relative axis position), the slave axis will move from current speed to SYNC speed. Except in the case above, the slave axis will stop moving immediately! All set input parameters of the *MC_CamIn* instruction will be read and retained for use in the execution.

Stage 2: Wait for the start of the engagement.
The slave axis waits for the timing for performing the engagement action in the standstill state. The time to start the engagement is when the master axis passes the position specified by the parameter *ActivationPosition*. In different circumstances, the period of time the slave axis waits for is different. If the master axis is at the position specified by *ActivationPosition* as the *MC_CamIn* instruction is executed, the slave axis starts the engagement action immediately. If the master axis never reaches the position specified by *ActivationPosition,* the slave axis will never start to perform the engagement action and the cam synchronization will never come true. The parameters *ActivationPosition* and *ActivationMode* are used at this stage.
Stage 3: The slave axis starts to perform the engagement action when the master axis passes the position specified by *ActivationPosition.* The parameters, *MasterAbsolute, SlaveAbsolute*, *MasterOffset*, *SlaveOffset*, *MasterScaling* and *SlaveScaling* will work at the moment for making sure of the corresponding relationship between the master axis position and slave axis position and the cam phase.
Stage 4: The engagement is ongoing.
The slave axis performs the engagement in the way specified by the *StartMode* parameter. Besides *StartMode,* the parameters *Velocity, Acceleration* and *Deceleration* also works at this stage. The motion features about velocity, acceleration/ deceleration of the slave axis are determined by these parameters in the engagement.

Stage 5: The engagement is completed and the master axis and slave axis achieve the synchronization.

The engagement is completed and the slave axis and master axis achieve the cam synchronization if the cam phase that the master axis and slave axis correspond to meets the planned cam relationship after the slave axis starts to perform the engagement action.

NOTE: In the figure above, the set master axis position at the time when the engagement begins is greater than the master position at the time when the *MC_CamIn* instruction execution starts. The similar way is also applied to the circumstance that the set master axis position at the time when the engagement begins is less than or equal to the master position at the time when the *MC_CamIn* instruction execution starts.

- **ActivationPosition**

The *ActivationPosition* parameter is the start position of the cam engagement, (which is the master axis position). In other words, the slave axis starts to perform the engagement when the master axis reaches the position specified by *ActivationPosition* after the *MC_CamIn* instruction is triggered and executed.

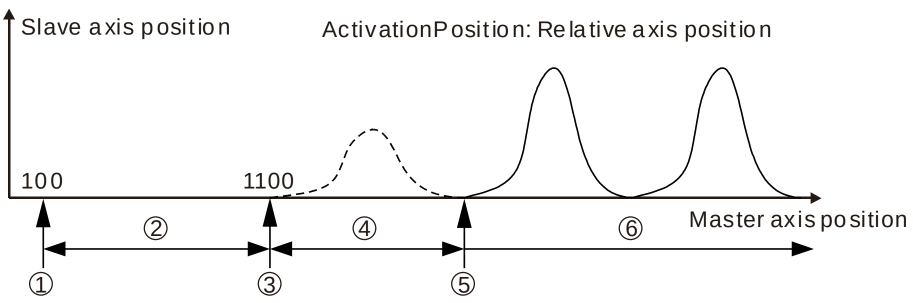
ActivationPosition can be the master axis position, master axis phase, master axis cam phase, which can be selected through the *ActivationMode* parameter.

- ***ActivationPosition:* Relative axis position**

As *ActivationMode*=0, *ActivationPosition* is an axis position which is relative to the master axis position at the time when the *MC_CamIn* instruction is executed. The master axis position as the actual engagement starts is the value of *ActivationPosition* plus the master position of when the *MC_CamIn* instruction execution begins.

For example: The master axis position is 100 and *ActivationPosition* 1000 at the time when the *MC_CamIn* instruction execution starts. The master axis position is 1100 (1100=100+1000) as the actual engagement begins.

MC_CamIn Execution Process



Stage 1: Trigger and execute the MC_CamIn instruction. The master axis absolute position is 100 at the moment.

Stage 2: Wait for the start of the engagement.

Stage 3: The master axis reaches the position for starting the engagement (1100) and the slave axis starts to perform the engagement action.

Stage 4: The engagement is ongoing.

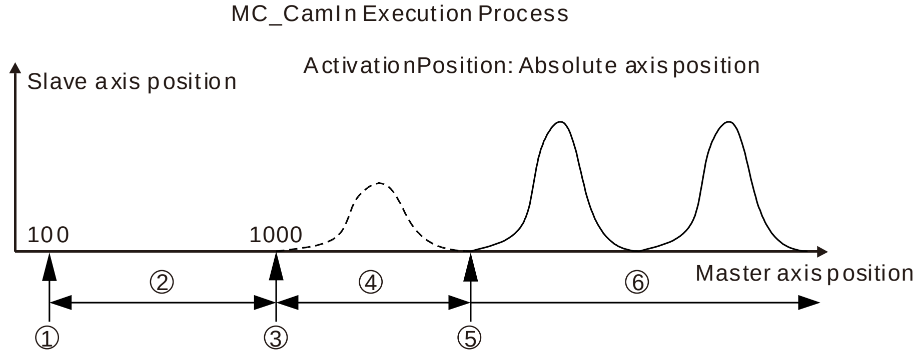
Stage 5: The engagement is completed and the master axis and slave axis achieve the synchronization.

Stage 6: The master axis and slave axis are in the synchronous motion.

- ***ActivationPosition*: Absolute axis position**

When *ActivationMode* =1, *ActivationPosition* is an axis position which is absolute to the master axis position at the time when the *MC_CamIn* instruction is executed. The master axis position as the actual engagement starts is *ActivationPosition*.

For example: The master axis position is 100 and *ActivationPosition* 1000 at the time when the *MC_CamIn* instruction execution starts. The master axis position is 1000 (1000= *ActivationPosition*) as the actual engagement begins.



Stage 1: Trigger and execute the MC_CamIn instruction. The master axis absolute position is 100 at the moment.

Stage 2: Wait for the start of the engagement.

Stage 3: The master axis reaches the position for starting the engagement (1000) and the slave axis starts to perform the engagement action.

Stage 4: The engagement is being conducted.

Stage 5: The engagement is completed and the master axis and slave axis achieve the synchronization.

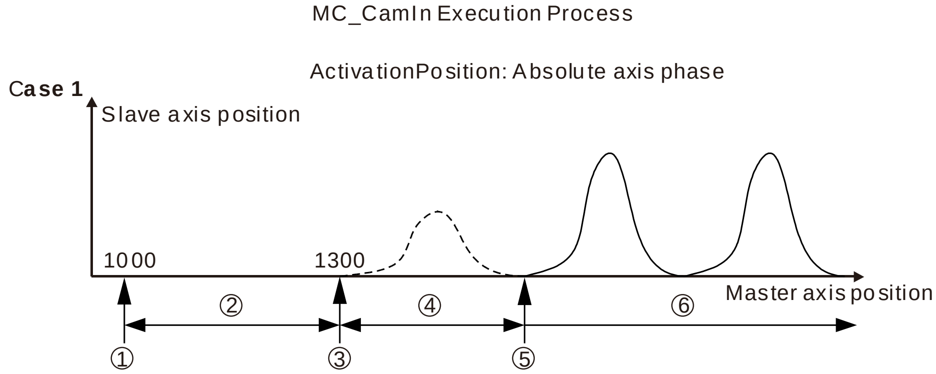
Stage 6: The master axis and slave axis are in the synchronous motion.

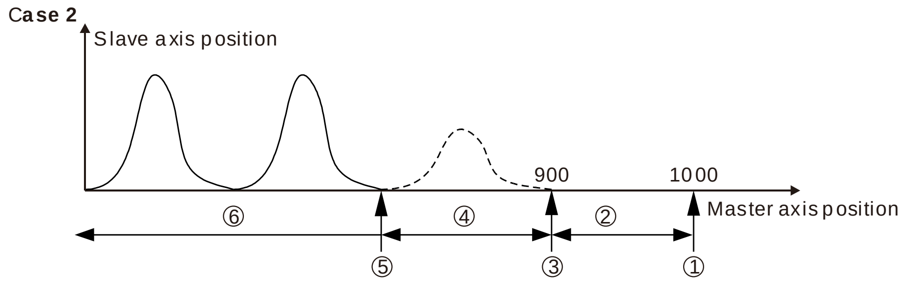
- ***ActivationPosition*: Absolute axis phase**

When *ActivationMode* =2, *ActivationPosition* is an absolute axis phase which is the remainder got by dividing the axis absolute position by modulo. The slave axis starts to perform the engagement action as the master axis absolute phase is *ActivationPosition*.

The absolute axis phase is cyclic. Its absolute axis phase may be equal to *ActivationPosition* many times in the motion of the master axis. But the slave axis starts to perform the engagement action only when the absolute axis phase of the master axis is equal to *ActivationPosition* for the first time after the MC_CamIn instruction is executed.

For example, the master axis modulo is 400, *ActivationPosition*=100 and the master axis position is 1000 at the time when the *MC_CamIn* instruction is executed. The slave axis will not perform the engagement action because the absolute axis phase of the master axis is 200 (200=1000%400) at the time when the *MC_CamIn* instruction is executed. The slave axis starts to perform the engagement action as the master axis position is 1300 (Absolute axis phase is 100=1300%400) or 900 (Absolute axis phase is 100=900%400). (% means the mathematic operation to find the remainder)





Stage 1: Trigger and execute the MC_CamIn instruction. The master axis absolute position is 1000 at the moment. (The absolute axis phase is 200)

Stage 2: Wait for the start of the engagement.

Stage 3: The master axis reaches the position for starting the engagement (1300 in circumstance 1 and 900 in circumstance 2) and the slave axis starts to perform the engagement action.

Stage 4: The engagement is being conducted.

Stage 5: The engagement is completed and the master axis and slave axis achieve the synchronization.

Stage 6: The master axis and slave axis are in the synchronous motion.

NOTE: As *ActivationPosition* is the absolute axis phase, the range of the *ActivationPosition* parameter value is 0~modulo (excluding modulo). If the value of *ActivationPosition* exceeds the valid range, an error will occur and the instruction execution will fail as the *MC_CamIn* instruction is executed.

- ***ActivationPosition*: Absolute cam phase**

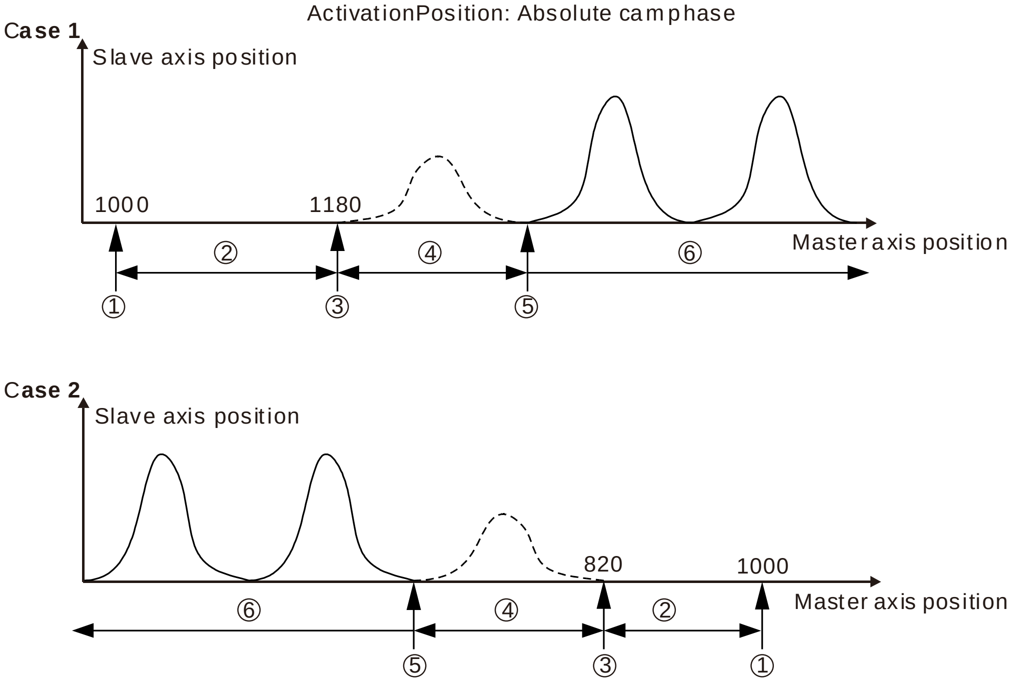
When *ActivationMode* =3, *ActivationPosition* is the absolute cam phase which is the remainder got by dividing the axis absolute position by its cam cycle. The slave axis starts to perform the engagement action as the cam phase of the master axis is *ActivationPosition*.

The cam phase is cyclic. Its cam phase may be equal to *ActivationPosition* many times in the motion of the master axis. But the slave axis starts to perform the engagement action only when the cam phase of the master axis is equal to *ActivationPosition* for the first time after the MC_CamIn instruction is executed.

For example, the maximum position of the master axis in the cam table is 360, *ActivationPosition*=100 and the master axis position is 1000 at the time when the *MC_CamIn* instruction is executed. The slave axis will not perform the engagement action because the absolute cam phase of the master axis is 280 (280=1000%360) at the time when the *MC_CamIn* instruction execution begins. Then the slave axis starts to perform the engagement action as the master axis position is 1180 (Absolute cam phase is 100=1180%360) or 820 (Absolute cam phase is 100=820%360).

MC_CamIn Execution Process

ActivationPosition: Absolute cam phase



Stage 1: Trigger and execute the MC_CamIn instruction. The master axis absolute position is 1000 at the moment. (The absolute cam phase is 280)

Stage 2: Wait for the start of the engagement.

Stage 3: The master axis reaches the position for starting the engagement (The master axis position is 1180 in circumstance 1 and 820 in circumstance 2) and the slave axis starts to perform the engagement action.

Stage 4: The engagement is being conducted.

Stage 5: The engagement is completed and the master axis and slave axis achieve the synchronization.

Stage 6: The master axis and slave axis are in the synchronous motion.

Note: As *ActivationPosition* is the absolute cam phase, the range of the *ActivationPosition* parameter value is 0~ cam cycle value (excluding the cam cycle value). If the value of *ActivationPosition* exceeds the valid range, an error will occur and the execution will fail as the *MC_CamIn* instruction is executed.

- **Relationship between the master axis position and slave axis position**

The cam relationship which is planned in the software is the position relationship between the master axis and slave axis. The “position” mentioned here is the cam phase of the master axis / slave axis instead of the actual axis position. If the cam relationship which is planned is seen as the function CAM as below, the input of the function CAM is the master axis cam phase and the output is the slave axis cam phase. The formula is shown as below.

y = CAM (x)

x：The master axis cam phase

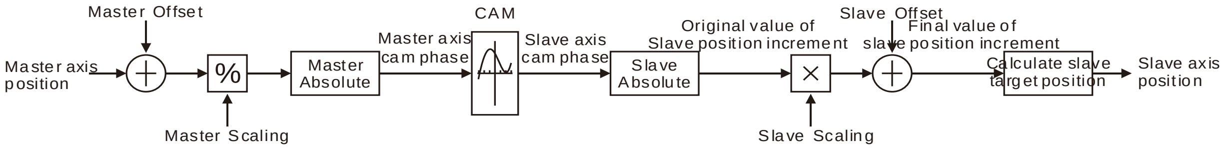
y：The slave axis cam phase

The cam phase comes from the axis positions and there is a conversion between them. The conversion between the axis position and cam phase is related with the *MasterAbsolute*, *SlaveAbsolute*, *MasterOffset*, *SlaveOffset*, *MasterScaling* and *SlaveScaling* parameters.

For details, refer to relevant sections.

11

The slave axis follows the master axis to make the synchronous cam motion by using the MC_*CamIn* instruction. In the synchronous cam motion, the corresponding relationship between the master axis position and slave axis position is based on the pre-planned cam relationship (the cam curve or cam table). The process in which the slave axis position is calculated through the master axis position is illustrated as follows.



■ ***MasterAbsolute* and *SlaveAbsolute***

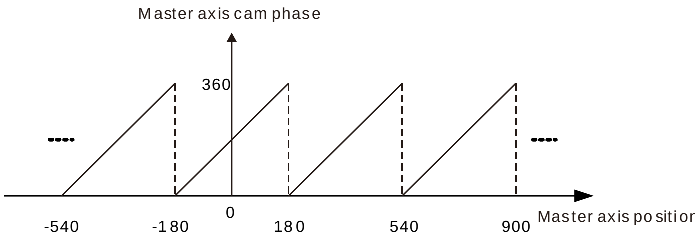
The *MasterAbsolute* parameter is used for specifying the corresponding relationship between the master axis position and the cam phase. As *MasterAbsolute* is TRUE, the master axis position and the cam phase are in an absolute relationship. As *MasterAbsolute* is FALSE, the master axis position and the cam phase are in a relative relationship. For *SlaveAbsolute,* the explanation is similar to that of *MasterAbsolute*.

MasterAbsolute and *SlaveAbsolute* work at the moment when the engagement starts. That is to say that the corresponding relationship between the axis position and cam phase is built at the beginning of the engagement. (**NOTE:** The corresponding relationship is not built at the time when the *MC_CamIn* instruction execution begins but when the engagement begins.) After that, the cam phase is calculated according to the corresponding relationship.

➢ **Relative mode**

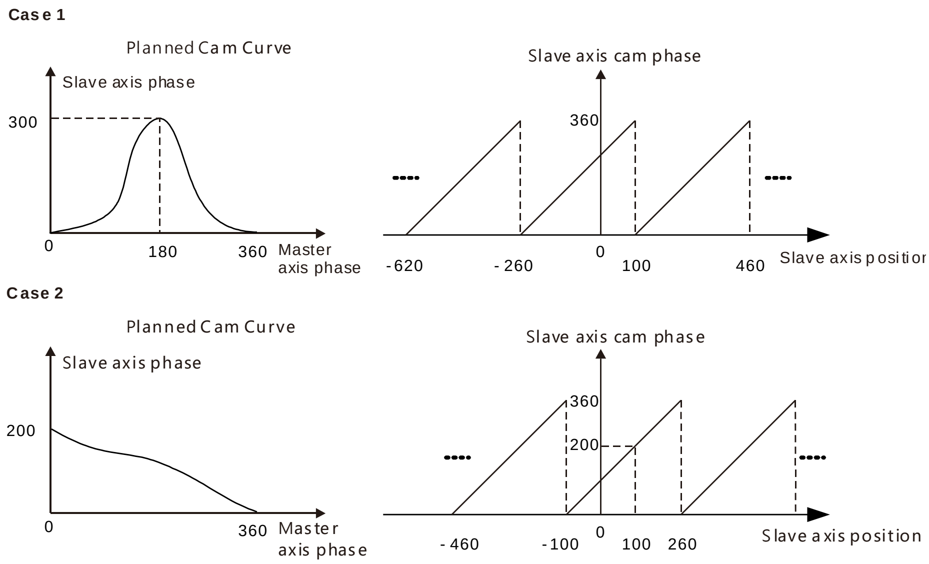
The master axis position and its cam phase are in the relative relationship as the *MasterAbsolute* parameter is FALSE. That is to say, the master axis position corresponds to its cam phase 0 at the time when the engagement starts. After that, the master cam phase will be calculated according to the corresponding relationship. For example, the master axis is in relative mode, the maximum value of the master axis cam phase in the cam relationship is 360 and the master axis position is 180 at the time when the engagement starts. So the master axis position 180 corresponds its cam phase 0; the master axis position 200 corresponds to its cam phase 20 (20= (200-180) %360) and so on.

In this circumstance, the master axis position corresponds to its cam phase as shown in the following figure.



As the *SlaveAbsolute* parameter is FALSE, the slave axis position and its cam phase are in the relative relationship. That is to say, the slave axis cam phase and the master axis cam phase meet the planned cam relationship at the time when the engagement starts. If the slave axis is in relative mode, the method of being sure of the slave axis cam phase is different from the master axis. When the slave axis cam phase is sure, it should meet the condition that the slave axis cam phase and the master axis cam phase meet the planned cam relationship at the time when the engagement starts.

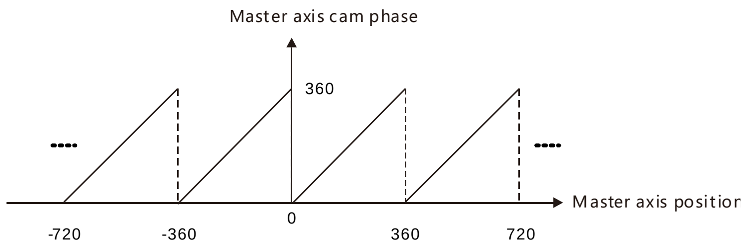
For example, the slave axis is in relative mode, the maximum value of the slave axis cam phase in the cam relationship is 360 and the slave axis position is 100 at the time when the engagement starts. If the master axis cam phase is 0 at the moment (and the slave axis cam phase is 0 as required in the cam relationship), the slave axis position 100 will correspond to its cam phase 0 as shown in the following circumstance 1. If the slave axis cam phase is 200 as required in the cam relationship, the slave axis position 100 will correspond to its cam phase 200 as shown in the following circumstance 2.



➢ **Absolute mode**

When the *MasterAbsolute* parameter is TRUE, the master axis position and its cam phase are in the absolute relationship. At any time, the master axis cam phase is equal to the remainder got by dividing the master axis position at that time by the maximum value of the master axis cam phase in the cam relationship.

For example, the master axis is in absolute mode and the maximum value of the master axis in the cam relationship is 360. So its cam phase is 100 as the master axis position is 100 (100=100%360); its cam phase is 140 (140=500%360) as the master axis position is 500 and so on. The master axis position corresponds to its cam phase as shown in the figure below.



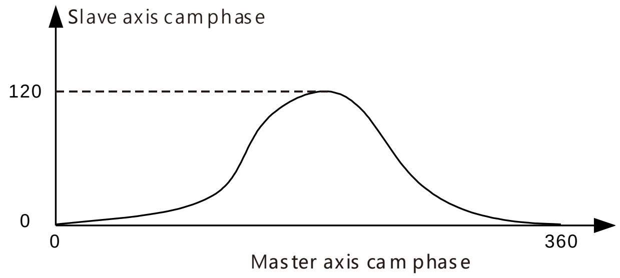
When the *SlaveAbsolute* parameter is TRUE, the slave axis position and its cam phase are in the absolute relationship. At any time, the slave axis cam phase is equal to the remainder got by dividing the slave axis position at that time by the maximum value of the slave axis cam phase in the cam relationship. When the slave axis is in absolute mode, the corresponding relationship between the slave axis position and its cam phase is consistent with that between the master axis position and its cam phase when the master axis is in absolute mode.

■ ***Offset* and *Scaling***

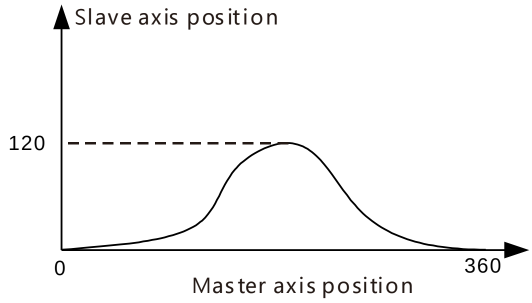
The cam relationship between the master axis and slave axis is preplanned. But as the cam motion is executed, the position offset or scaling based on the preplanned cam relationship can be performed through setting the *Offset* and *Scaling* parameters. For example, there are various sizes for the same product which is processed. Just one cam relationship need be planned and then changing the values of *Offset* and *Scaling* fits the processing of products of different sizes.

The *MasterOffset* parameter is valid only when the master axis is in absolute or relative mode. (*MasterAbsolute*=TRUE or FALSE). The *SlaveOffset* parameter is valid only as the slave axis is in absolute mode (*SlaveAbsolute*=TRUE). The *SlaveOffset* parameter is invalid as the slave axis is in relative mode (*SlaveAbsolute*=FALSE).

The position offset and scaling of the master axis and slave axis determine the actually executed cam relationship. The effect is described in the following example. The planned cam relationship is shown as the figure below.



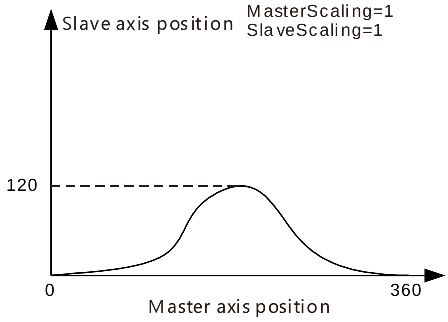
When the master axis and slave axis are both in absolute mode and the engagement begins, the master axis position and slave axis position are both 0. When there is no position offset and scaling (the offset and scaling are default values), the actual master axis position correspond to the actual slave axis position in the execution of the cam motion as shown in the following figure.



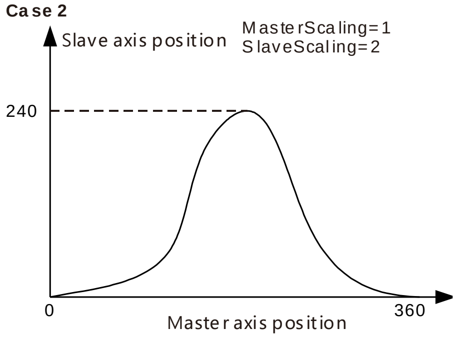
When the offset and scaling are not default values, the corresponding relationship between the actual master axis position and actual slave axis position are affected in the execution of the cam motion as below.

- **MasterOffset:0 and SlaveOffset:0 and the impact of MasterScaling and SlaveScaling on the cam relationship**

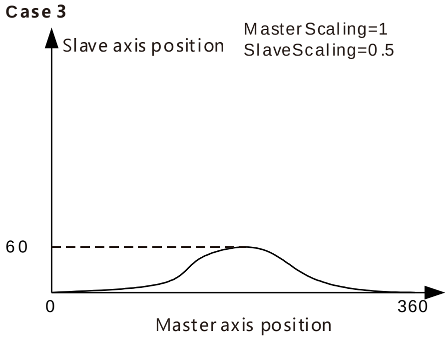
Case 1



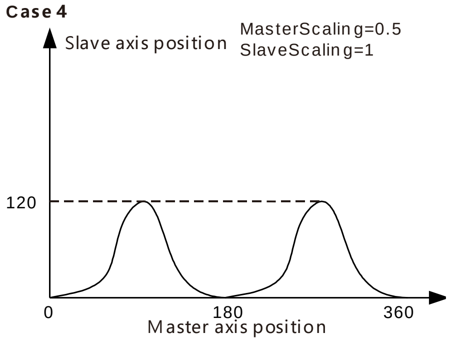
Case 2



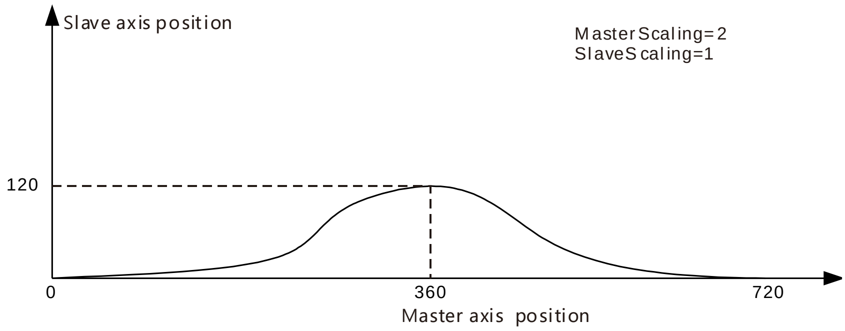
Case 3



Case 4



Case 5



Case 1： The actual cam relationship is consistent with the preplanned one as the values of MasterScaling and SlaveScaling are 1 and their offsets are 0.

Case 2： The slave position corresponding to the master axis position is two times what is planned in the cam relationship as the value of *MasterScaling* is 1, *SlaveScaling* is 2 and their offsets are 0.

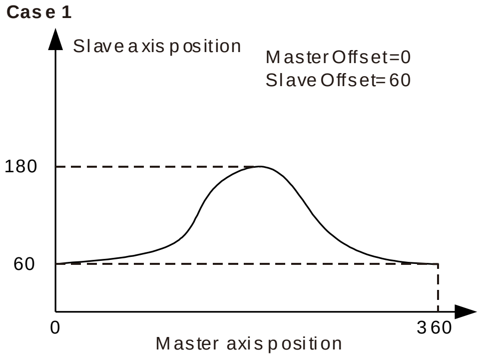
Case 3： The slave position corresponding to the master axis position is 1/2 that in the planned cam relationship as the value of MasterScaling is 1, SlaveScaling is 0.5 and their offsets are 0.

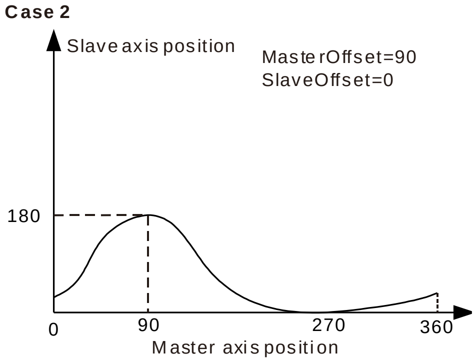
Case 4： The master axis position corresponding to the slave axis position is 1/2 what is planned as the value of *MasterScaling* is 0.5, *SlaveScaling* is 1 and their offsets are 0. If it is observed from the perspective of the cam phase, the master axis cam phase is 1/2 what is preplanned. That is, the master cam cycle changes from 360 to 180 (180=360*0.5) and the slave axis cam phase is unchanged.

Case 5： The master axis position corresponding to the slave axis position is 2 times what is planned as the value of *MasterScaling* is 2, *SlaveScaling* is 1 and their offsets are 0. If it is observed from the perspective of the cam phase, the master axis cam phase is two times the original. That is, the master axis cam cycle changes from 360 to 720 (720=360*2) and the slave axis cam phase is unchanged.

- **MasterScaling:1 and SlaveScaling:1 and the impact of MasterOffset and SlaveOffset on the actually executed cam relationship**

  MasterOffset means to make the actual axis position curve shifted horizontally in execution of the cam motion. *SlaveOffset* indicates to make the axis position curve shifted vertically in execution of the cam motion.





Case 1： The slave axis position corresponding to the master axis position will add by 60 based on the planned position as *MasterScaling* and *SlaveScaling* are both 1, *MasterOffset* is 0 and *SlaveOffset* is 60.

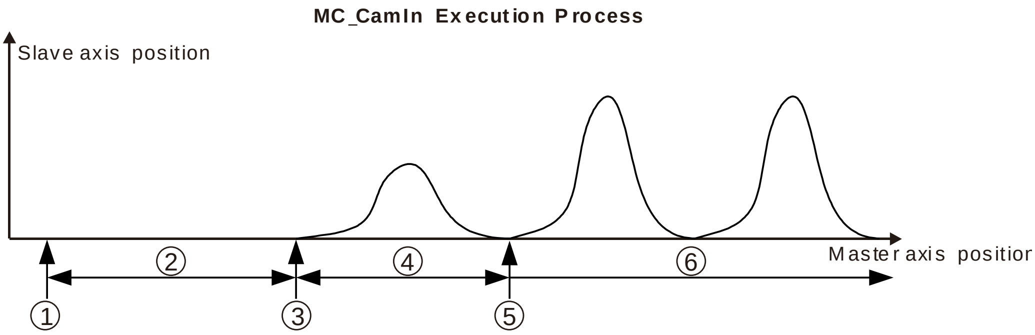
For example, in the planned cam relationship, the master axis position 180 corresponds to the slave axis position 180 and in the actual execution, the corresponding slave axis position is 240 (240=180+60).

Case 2： The master axis position corresponding to the slave axis position will shift (add) by 90 based on the planned position as *MasterScaling* and *SlaveScaling* are 1, *MasterOffset* is 90 and *SlaveOffset* is 0.

For example, in the planned cam relationship, the master axis position 180 corresponds to the slave axis position 180 and in the actual execution, the master axis position 90 corresponds to the slave axis position 180 which is the slave axis position corresponded to by the master axis position 180 (180=90+90) in the planned cam relationship.

- **StartMode**

In the engagement, the way how the slave axis moves is specified by the *StartMode* parameter. That is, *StartMode* works at stage 4 in the execution of the *MC_CamIn* instruction as shown in the following figure.



Stage 1: Trigger and execute the MC_CamIn instruction.
Stage 2: Wait for the start of the engagement.

Stage 3: The master axis reaches the position where the engagement begins and the slave axis starts to perform the engagement action.

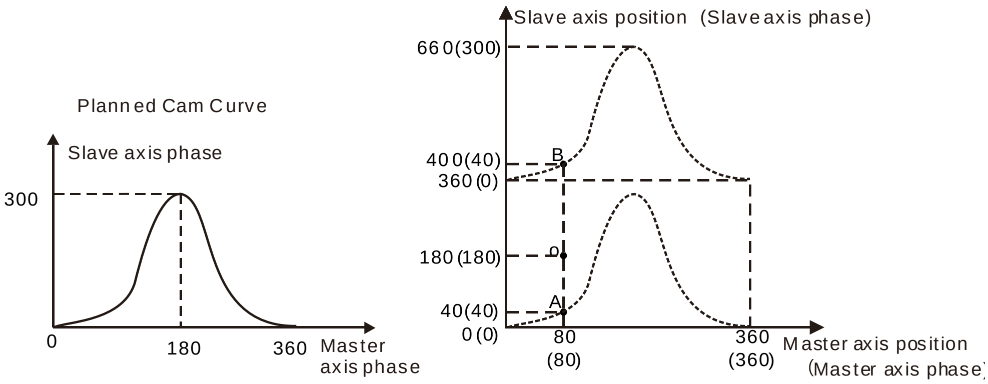
Stage 4: The engagement is ongoing.

Stage 5: The engagement is completed and the master axis and slave axis achieve the synchronization.

Stage 6: The master axis and slave axis are in the synchronous motion.

The cam synchronization requires that the master axis cam phase and the slave axis cam phase meet the defined cam relationship. The engagement process is the process in which the slave axis moves toward the synchronous phase. The synchronous phase and the master axis cam phase meet the defined cam relationship. Since the axis cam phase is cyclic, every cam phase is corresponded to by multiple axis positions. When the engagement occurs, there are many selections for the expected synchronization position. And thus there are several engagement ways for option.

For example, when the engagement starts, the master axis cam phase and slave axis cam phase are 80 and 180 respectively as point O in the following figure. But the defined cam relationship requires that the slave axis cam phase is 40 and thus the synchronous position that the slave axis expects is 40 or 400 (Point A or point B in the following figure) at the moment. The engagement process from Point O to A or Point O to B can be selected via the *StartMode* parameter.



There are three modes of *StartMode* for selection: the shortest way (mcRamplnShortest), positive direction (mcRamplnPositive) and negative direction (mcRamplnNegative). Users can select the right engagement mode according to actual need.

- ***StartMode=0*** **(The shortest way)**

  As *StartMode*=0, in the execution of the engagement action, the slave axis moves toward the position for synchronization by taking the shortest way. At the moment, the motion of the slave axis is affected by the *Velocity, Acceleration Deceleration* and *Jerk* parameters.

- ***StartMode=1*** **(Positive direction)**

  As *StartMode*=1, in the execution of the engagement action, the slave axis moves toward the position for synchronization in the positive direction. At the moment, the motion of the slave axis is affected by the *Velocity, Acceleration Deceleration* and *Jerk* parameters.

- ***StartMode=-1*** **(Negative direction)**

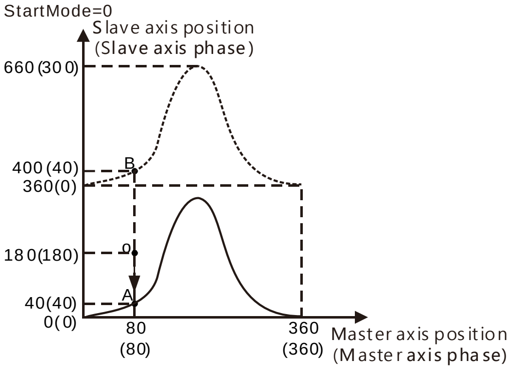
  As *StartMode*=-1, in the execution of the engagement action, the slave axis moves toward the position for synchronization in the negative direction. At the moment, the motion of the slave axis is affected by the *Velocity, Acceleration Deceleration* and *Jerk* parameters.

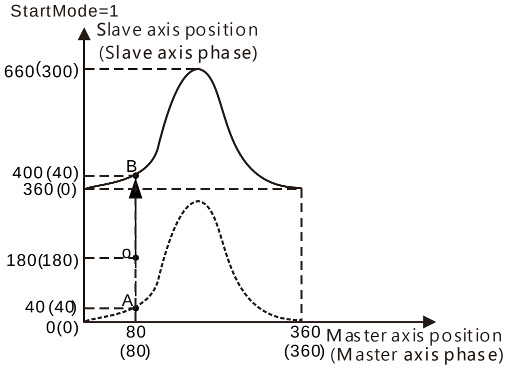
For example, as the engagement begins, the master axis cam phase and slave axis cam phase are 80 and 180 respectively (as point O below). According to the defined cam relationship, the master axis cam phase is 80 and the slave axis cam phase is 40 (as point A or B below). If the value of *StartMode* is different, the way the slave axis moves is different in the engagement process.

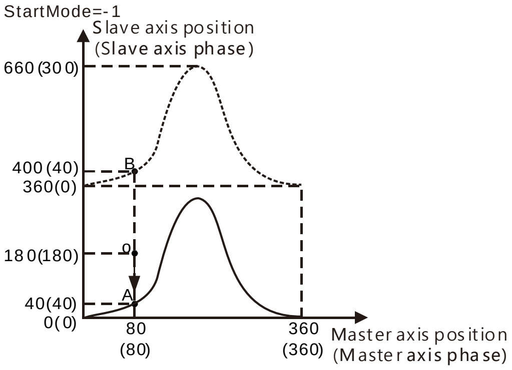
StartMode=0：The slave axis moves from point O to point A and the synchronization is achieved at point A since the distance from point O to point A is less than that from point O to point B.

StartMode=1：The slave axis gradually moves from point O to point B in the positive direction.

StartMode=-1：The slave axis gradually moves from point O to point A in the negative direction.







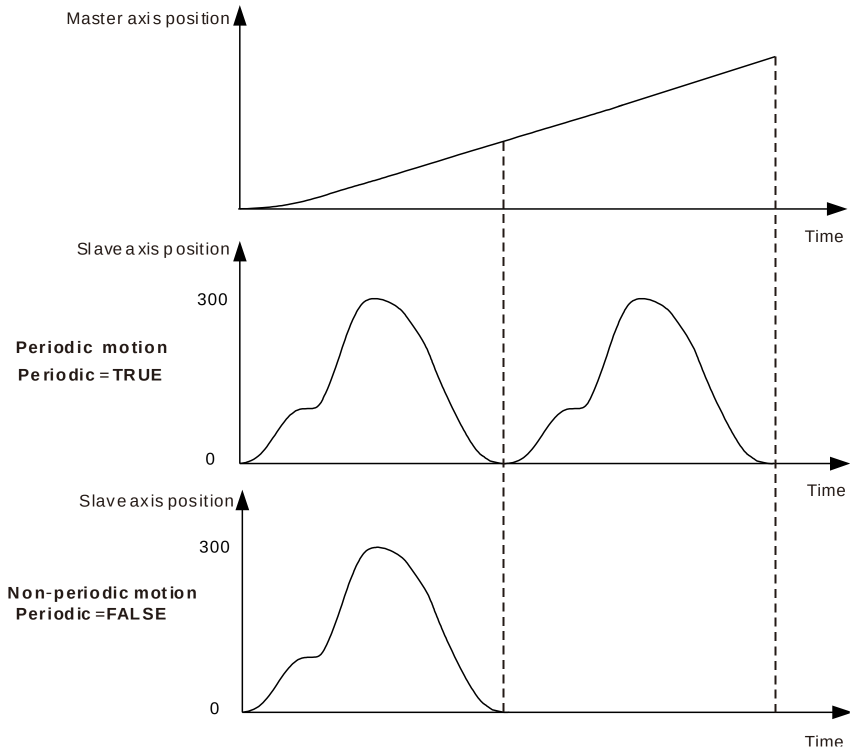
- **Periodic/Non-periodic Cam Operation（Periodic）**

In the actual application of electronic cams, some may be executed periodically and some just need be executed for one cycle. The *Periodic* parameter is used for choosing one of the two cases for the electronic cam motion.

As *Periodic*=TRUE, the slave axis follows the master axis to periodically perform the cam motion till the cam relationship is disconnected.

As *Periodic*=FALSE, when the end point of the cam cycle is reached after the slave axis and master axis enter the synchronous cam motion, the cam relationship between the slave axis and master axis will be disconnected and the slave axis will stop moving immediately.

If the velocity at the end point of the planned cam relationship is not 0, the slave axis will constantly move at the disconnection speed after the disconnection of the cam relationship.



- **The impact of other instructions on cam operation**

  - ***MC_CamOut***

    The *MC_CamOut* instruction can be used to end the cam operation which is being carried out.

  - ***MC_SetPosition***

    The *MC_SetPosition* instruction has no impact on the being executed motion instructions. Thus, during cam operation, the execution of *MC_SetPosition* instruction for the master axis and slave axis will not affect the cam operation. If the cam operation is triggered after the *MC_SetPosition* instruction is executed, the cam will be affected by the axis position change which is incurred by using the *MC_SetPosition* instruction.

  - ***MC_Stop* and *MC_Halt***

    As the *MC_Stop* and *MC_Halt* instructions are executed on the slave axis, the *MC_CamIn* instruction is aborted, the cam relationship is disconnected and the slave axis decelerates till it stops.

  - ***MC_Home***

    The *MC_Home* instruction cannot be executed on the slave axis but the master axis. As the *MC_Home* instruction is executed on the master axis, the master axis position may have a great change in a very short time, which may cause the vibration of the slave axis. Therefore, the *MC_Home* instruction is recommended to execute after the synchronous relationship between the two axes is disconnected.

11

■ **Other precautions**

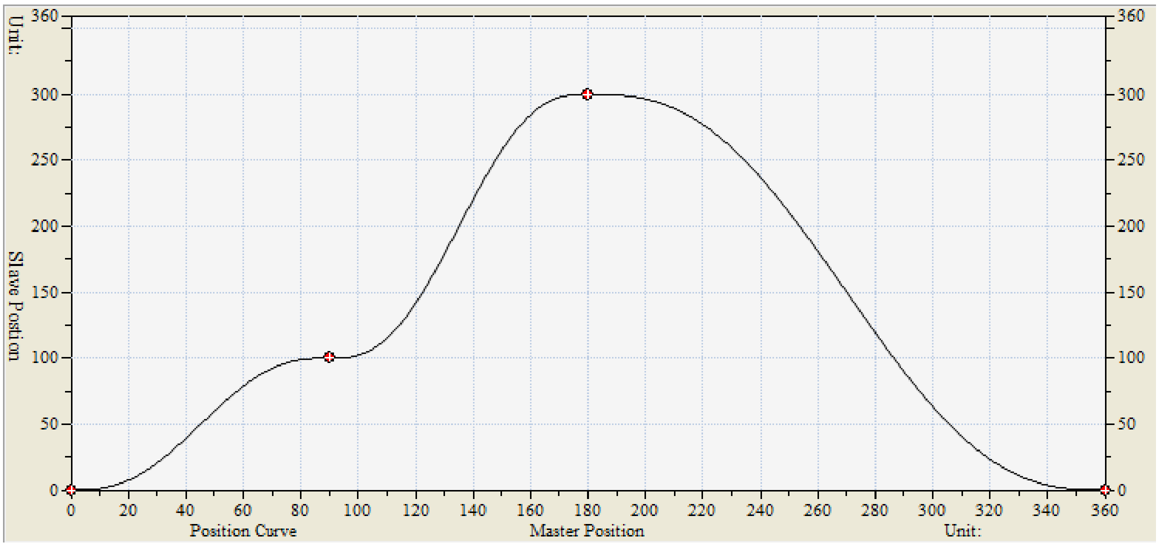
See the rule for different types of axes working as the master axis or slave axis in the cam relationship in the following table.

Axis type	As cam master axis	As cam slave axis
Servo real axis	OK	OK
Encoder	OK	NO
Virtual axis	OK	OK

Programming Example

- **The execution effect of the *MC_CamIn* instruction is described in the following example.**

■ **The cam curve is planned as below.**



■ **Key points of the cam curve**

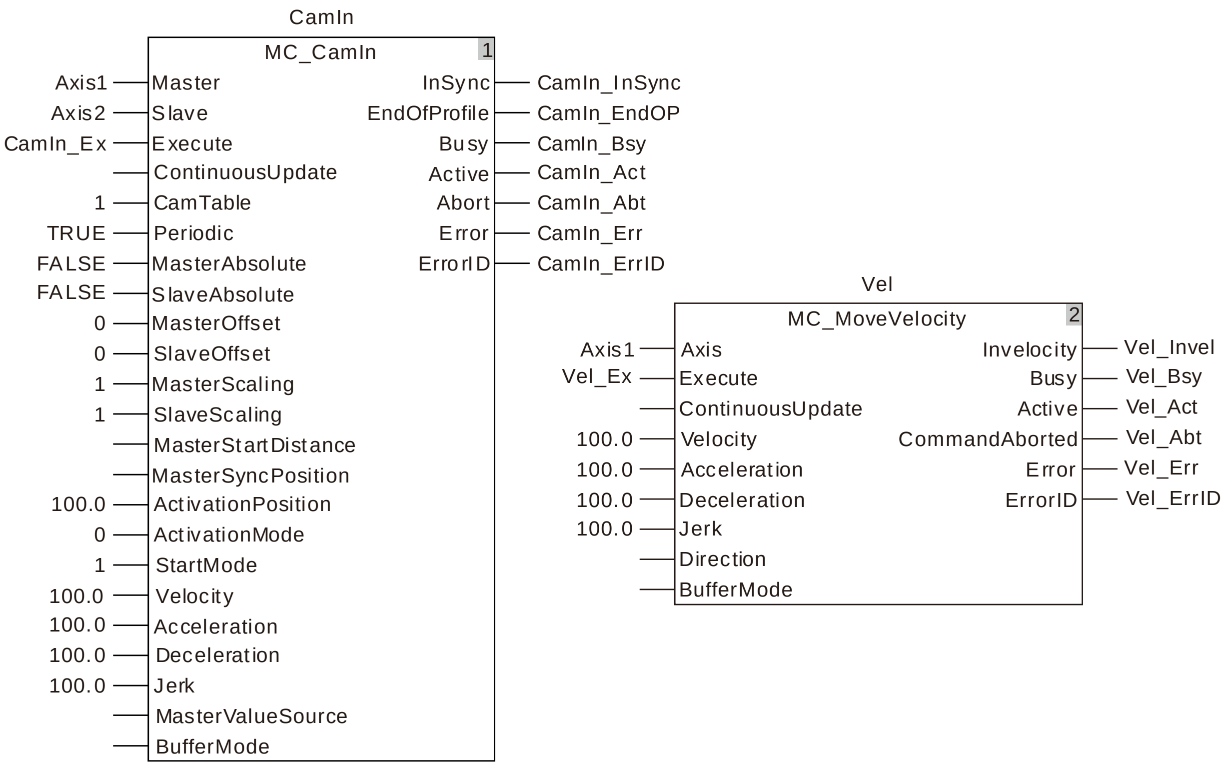
No.	Master axis position	Slave axis position	Velocity	Acceleration
1	0	0	0	0
2	90	100	0	0
3	180	300	0	0
4	360	0	0	0

■ **Explanation:**

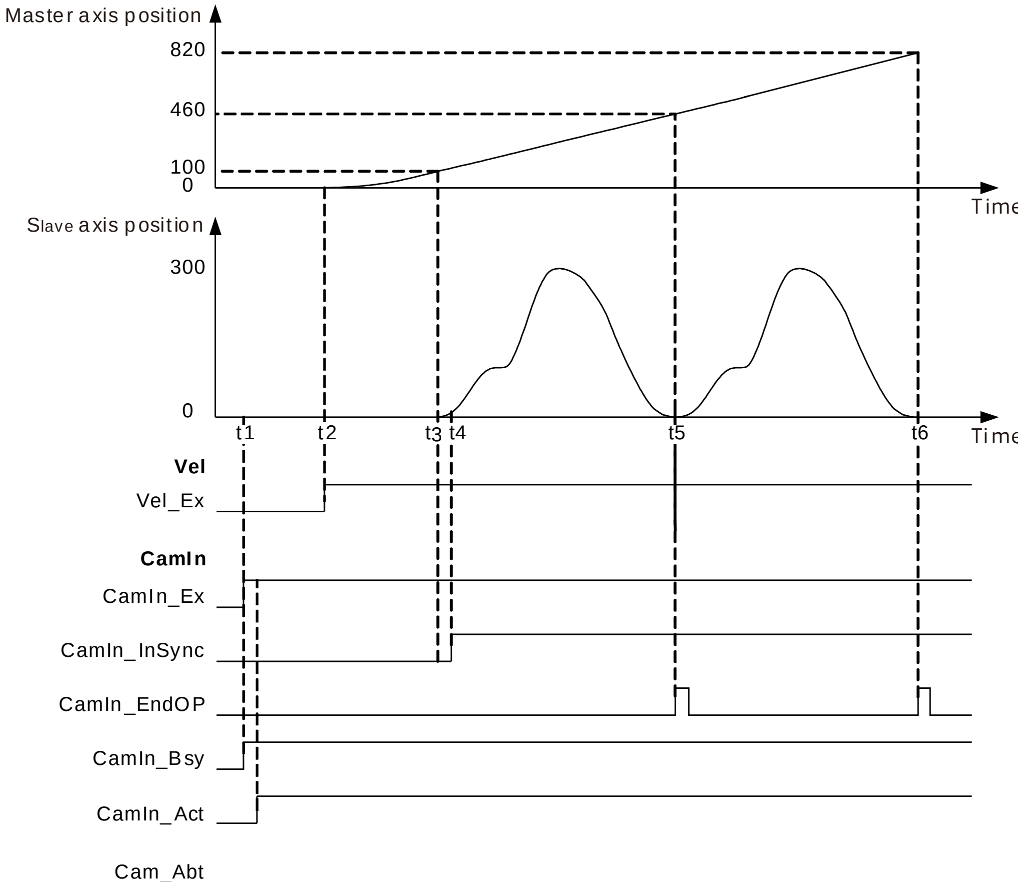
Cam period of the master axis and slave axis	360
Master Scaling and SlaveScaling	1
MasterOffset and SlaveOffset	0
MasterAbsolute	Relative
SlaveAbsolute	Relative
Periodic	Periodic
ActivationPosition	Relative axis position:100
StartMode	The shortest way

■ The variable table and program

Variable name	Data type	Initial value
CamIn	MC_CamIn	
CamIn_Ex	BOOL	
CamIn_InSync	BOOL	
CamIn_EndOP	BOOL	
CamIn_Bsy	BOOL	
CamIn_Act	BOOL	
CamIn_Abt	BOOL	
CamIn_Err	BOOL	
CamIn_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel _Ex	BOOL	
Vel _InVel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ ErrID	WORD	



- **Motion curve and timing chart**



❖ As CamIn_Ex changes from FALSE to TRUE, the MC_CamIn instruction is executed and at the moment of t1, both of the master axis and slave axis positions are 0. The value of *ActivationPosition* is 100 and *ActivationMode* is 0, so the slave will not start to execute the engagement action until the master axis position is 100 (the master axis position at the time of t1 + *ActivationPosition*).

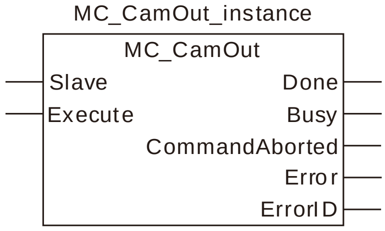
❖ As Vel_Ex changes from FALSE to TRUE, the MC_MoveVelocity instruction is executed and at the moment of t2, the master axis position is 0 and slave axis continues waiting for the start of the engagement. After that, the master axis will move from 0 in the positive direction under the control of the MC_MoveVelocity instruction.

❖ When the master axis passes 100, the position where the engagement begins is reached at the time of t3. The slave axis starts to perform the engagement action according to *StartMode* at the moment of t3. The synchronization is achieved at t4 and the *InSync* output parameter (CamIn1_InSync) changes from FALSE to TRUE.

❖ Whenever the synchronous motion reaches the end point in a cam period as shown at t5 and t6, the *EndOfProfile* output parameter (CamIn1_EndPro) will change to TRUE and it will change to FALSE after a program period.

11.4.6 MC_CamOut

FB/FC	Explanation	Applicable model
FB	MC_CamOut can disconnect the established electronic cam relationship.	DVP15MC11T



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Slave	Specify the number of the slave axis which is to be disconnected from the cam relationship.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	-

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction execution is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
CommandAborted	TRUE when the instruction is aborted.	BOOL	TRUE / FALSE
Error	TRUE when there is an error in the execution of the instruction.	BOOL	TRUE / FALSE
ErrorID	Contains the error code when an error occurs. Please refer to the section 12.2.	WORD	

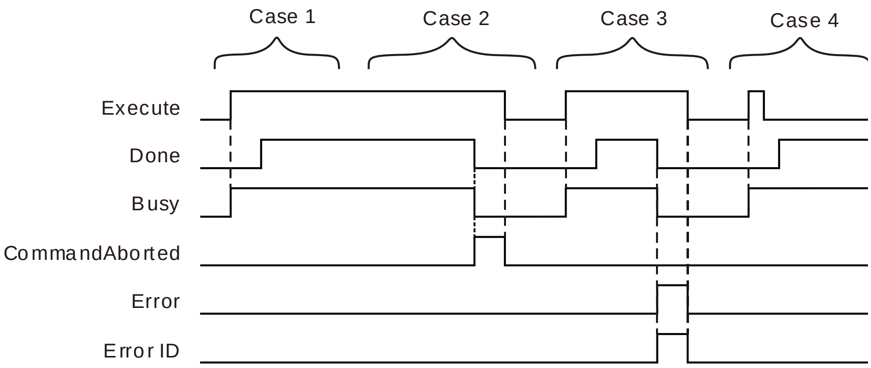
- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the electronic cam relationship between the slave axis and master axis is disconnected.	◆ When *CommandAborted* changes to TRUE. ◆ When *Error* changes to TRUE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *CommandAborted* changes to TRUE. ◆ When *Error* changes to TRUE.
CommandAborted	◆ When the instruction execution is aborted by other motion instruction.	◆ When *Execute* changes from TRUE to FALSE. ◆ *CommandAborted* is set to TRUE when the instruction is aborted by other instruction after *Execute* changes from TRUE to FALSE in the course of the instruction execution. One period later, *CommandAborted* changes to FALSE.

11

Name	Timing for changing to TRUE	Timing for changing to FALSE
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Output Update Timing Chart**



Case 1： *Busy* changes to TRUE as *Execute* changes from FALSE to TRUE. One period later, *Done* changes to TRUE. *Busy* and *Done* remain TRUE after *Execute* changes from TRUE to FALSE.

Case 2： When *Execute* is TRUE, *CommandAborted* changes to TRUE and meanwhile *Busy* and *Done* change to FALSE if the instruction is aborted by other instruction. When *Execute* changes from TRUE to FALSE, *CommandAborted* changes to FALSE.

Case 3： As *Execute* changes from FALSE to TRUE and an error occurs (e.g. an axis is disabled), *Error* changes to TRUE and *ErrorID* shows corresponding error codes. Meanwhile *Busy* and *Done* change to FALSE. As *Execute* changes from TRUE to FALSE, *Error* changes to FALSE.

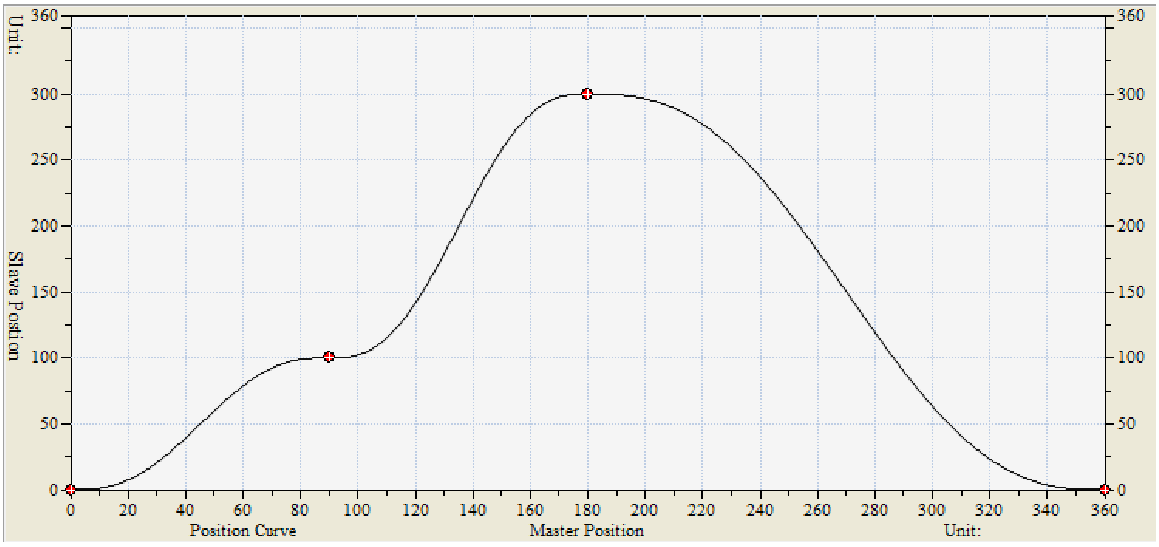
Case 4： *Execute* changes from TRUE to FALSE as the instruction execution lasts for less than one period. After that, *Done* changes to TRUE and *Busy* remain TRUE as one period is reached.

- **Functions**

MC_CamOut is used for disconnecting the established electronic cam relationship. The instruction works on the slave axis in the cam operation and the slave axis will continue moving at the speed of when it is disconnected from the cam relationship.
MC_Halt or MC_Stop instructions can be executed on the slave axis so as to stop the slave axis motion. The slave axis will stop moving and the cam relationship will be disconnected after the execution of the MC_Halt instruction or MC_Stop instruction is completed.

Programming Example

- **The execution effect of the *MC_CamOut* instruction is described in the following example. The cam curve is planned as below.**



- **The key points of the cam curve**

No.	Master axis position	Slave axis position	Velocity	Acceleration
1	0	0	0	0
2	90	100	0	0
3	180	300	0	0
4	360	0	0	0

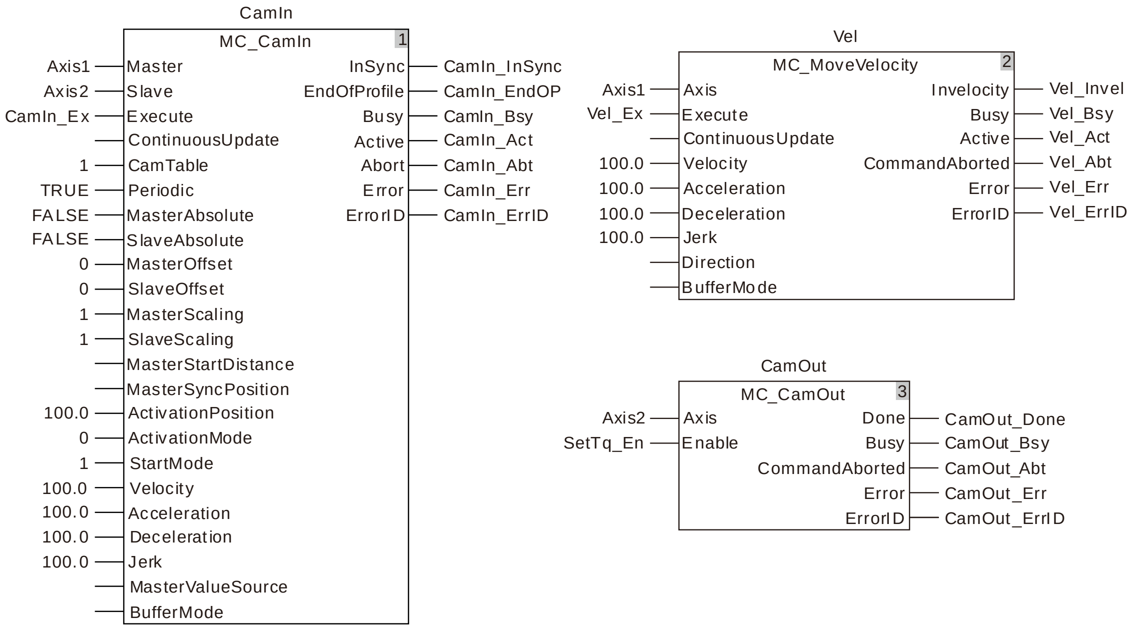
- **Explanation:**

Cam period of the master axis and slave axis	360
MasterScaling and SlaveScaling	1
MasterOffset and SlaveOffset	0
MasterAbsolute	Relative
SlaveAbsolute	Relative
Periodic	Periodic
ActivationPosition	Relative axis position: 100
StartMode	The shortest way

- **The variable table and program**

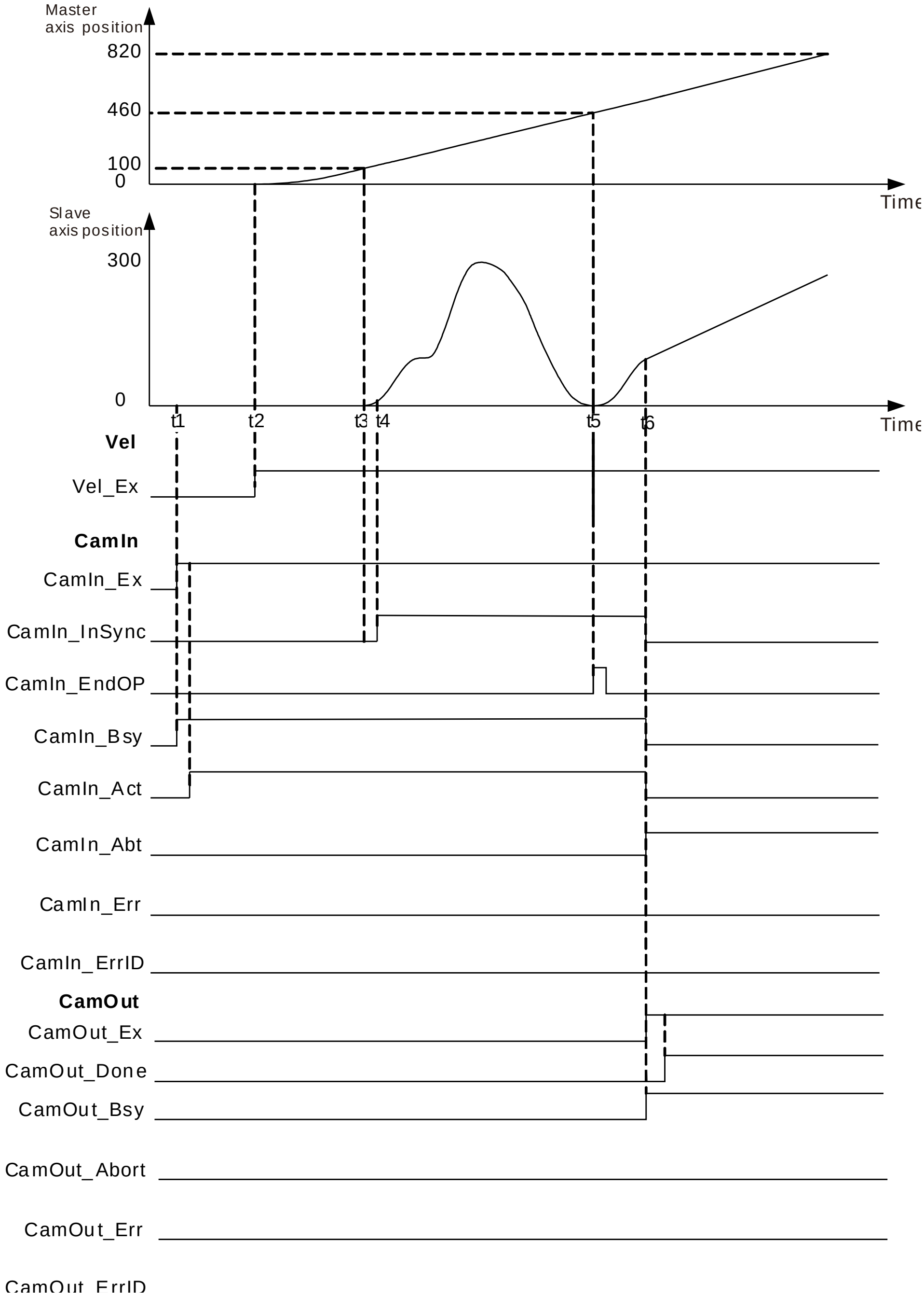
Variable name	Data type	Initial value
CamIn	MC_CamIn	
CamIn_Ex	BOOL	
CamIn_InSync	BOOL	
CamIn_EndOP	BOOL	
CamIn_Bsy	BOOL	
CamIn_Act	BOOL	
CamIn_Abt	BOOL	

Variable name	Data type	Initial value
CamIn_Err	BOOL	
CamIn_ErrID	WORD	
Vel	MC_MoveVelocity	
Vel_Ex	BOOL	
Vel_InVel	BOOL	
Vel_Bsy	BOOL	
Vel_Act	BOOL	
Vel_Abt	BOOL	
Vel_Err	BOOL	
Vel_ErrID	WORD	
CamOut	MC_CamOut	
CamOut_Ex	BOOL	
CamOut_Done	BOOL	
CamOut_Bsy	BOOL	
CamOut_Abt	BOOL	
CamOut_Err	BOOL	
CamOut_ErrID	WORD	



11

■ **Motion curve and timing chart**



❖ As CamIn_Ex changes from FALSE to TRUE at t1, the MC_CamIn instruction is executed and at the moment, both of the master axis and slave axis positions are 0. The value of *ActivationPosition* is 100 and *ActivationMode* is 0, so the slave axis will not start to execute the engagement action until the master axis position is 100 (the master axis position at t1 + *ActivationPosition*).

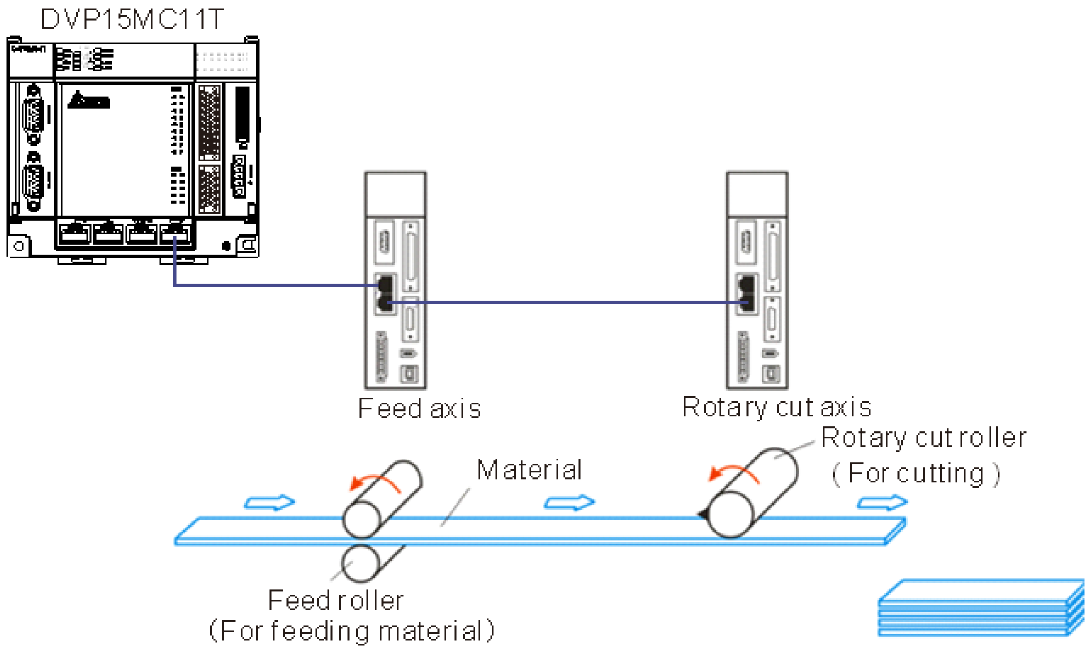
❖ As Vel_Ex changes from FALSE to TRUE at t2, the MC_MoveVelocity instruction execution starts. At the moment, the master axis position is 0 and the slave axis continues waiting for the execution of the engagement action. After that, the master axis moves from 0 in the positive direction under the control of the MC_MoveVelocity instruction.

- ❖ The position where the engagement starts is reached as the master axis passes 100 at t3. The slave axis starts to perform the engagement action according to *StartMode* at t3. The synchronization is achieved at t4 and the *InSync* output parameter (CamIn1_InSync) changes from FALSE to TRUE.
- ❖ During the synchronous cam motion in which the slave axis follows the motion of the master axis, by executing the MC_CamOut instruction, the cam relationship is disconnected at t6. After the MC_CamOut instruction is executed, the slave axis will keep moving at the speed it has when the slave axis is disconnected from the cam relationship.

11.5 Application Instructions

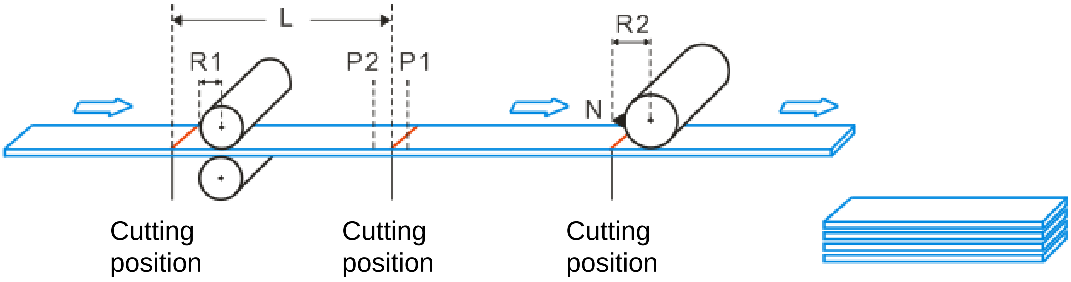
11.5.1 Rotary Cut Technology

Rotary cut is the technology to cut the material in transmission vertically. The knife conducts cutting on the cut surface periodically with the rotation of the rotary cut axis.



Note:
The feed axis is to control the feed roller; the rotary cut axis is to control rotary cut roller with the knife mounted on the rotary cut roller. The rotary cut function is usually used for cutting of the thin material or the material of medium thinness and can be applied in packaging machine, cutting machine, punching machine, printing machine etc.

11.5.2 Rotary Cut Parameters

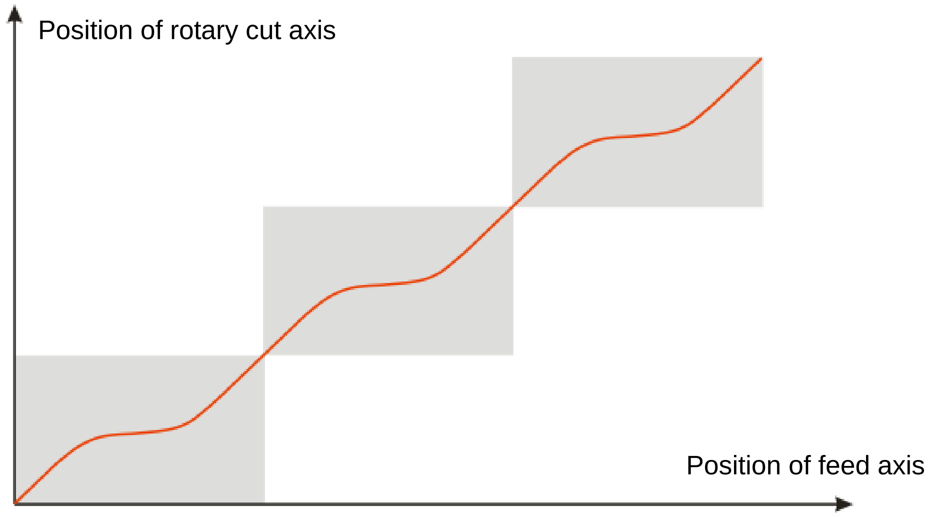


Parameter in the figure	Explanation	Corresponding parameter name of the instruction
L	The cutting length of the processed material	APF_RotaryCut_Init.CutLength
R1	The radius of the feed axis, i.e. the radius length of the feed roller.	APF_RotaryCut_Init.FeedRadius
R2	The radius of the rotary axis, i.e. the distance from the center of the rotary roller to the tool bit.	APF_RotaryCut_Init.RotaryRadius
N	The number of knives of the rotary roller. The number of knives is 1 in the figure above.	APF_RotaryCut_Init.KnifeNum
P1	The starting position of the synchronous area.	APF_RotaryCut_Init.SyncStartPos
P2	The end position of the synchronous area.	APF_RotaryCut_Init.SyncStopPos

11

11.5.3 Control Feature of Rotary Cut Function

Rotary cut function is a type of special electronic cam function. The figure of cam curve is shown below for continuous cutting.



- **Features**

1. Users can set the cutting length freely according to the technological requirement and the cutting length could be less or more than the circumference of the cutter.
2. In the SYNC area, the rotary cut axis and feed axis move at a certain speed rate. (Their velocities are usually equal.) And the cutting of material is conducted in the SYN area.
3. DVP15MC11T supports the rotary roller with multiple knives.
4. The feed axis is able to make the constant motion, acceleration, deceleration and irregular motion because the rotary cut axis moves according to the phase of the feed axis after the rotary cut function is enabled.
5. When rotary cut relation is broken off, the knife stops at the zero point of the system, i.e. the entry position for rotary cutting.

11.5.4 Introduction to Cam Curve with Rotary Cut Function

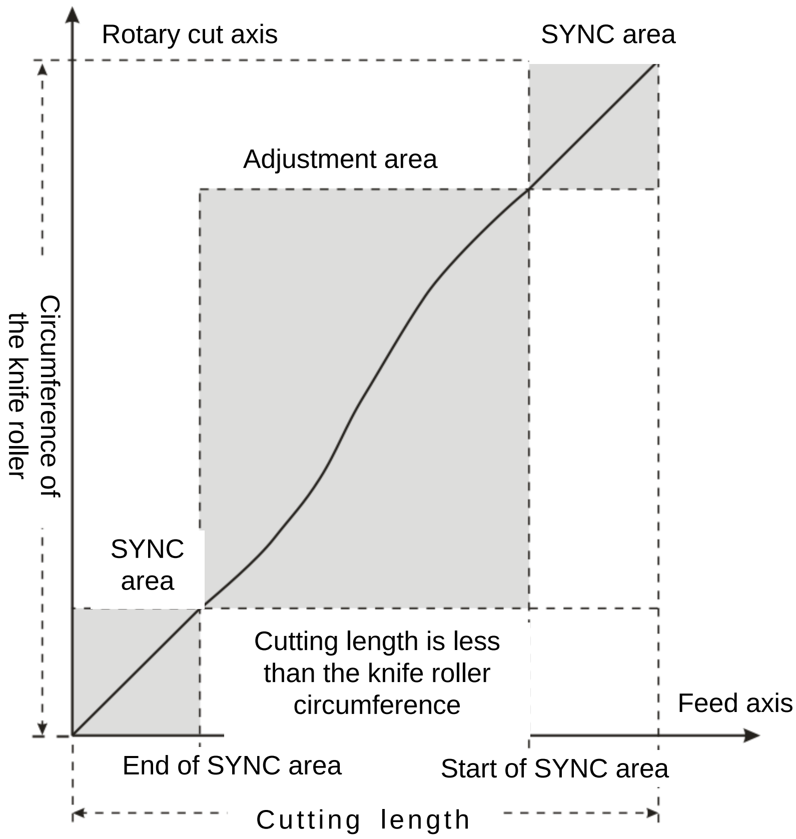
The cam curve with the rotary cut function could be divided into the SYNC area and adjustment area.

SYNC area: Feed axis and rotary-cut axis make the motion at a fixed speed ratio (Linear speed of the knife is usually equal to that of the cut surface), and material cutting takes place in SYNC area.

Adjustment area: Due to different cutting length, positioning need be adjusted accordingly. Adjustment area can be in the following three situations based on various cutting length.

1. **Short material cutting**

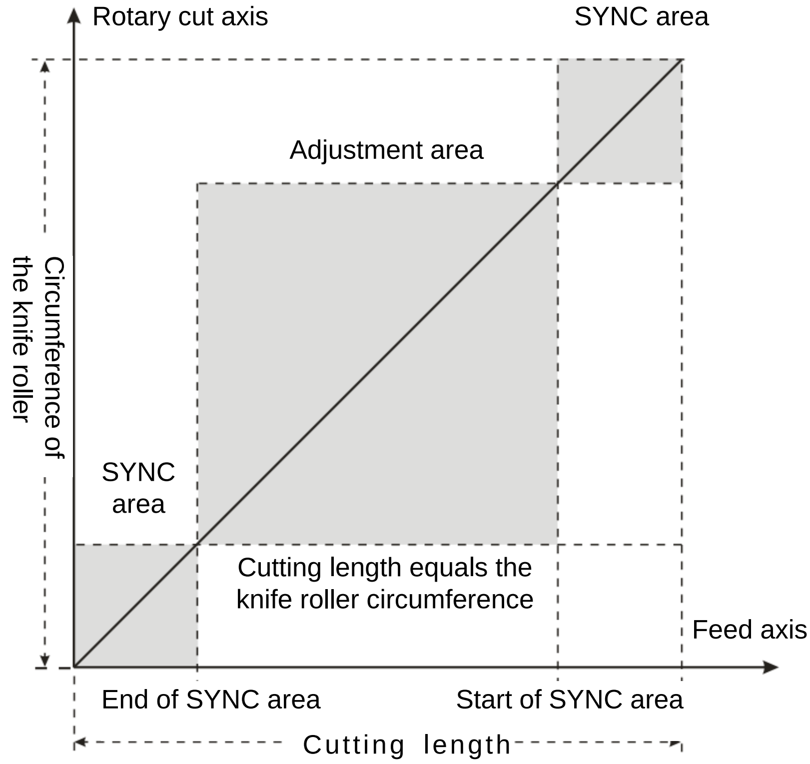
   When cutting length is less than the knife roller circumference, the rotary-cut curve for any cycle is shown below.



   For the cutting of short material, rotary cut axis must accelerate first in the adjustment area, and then decelerate to the synchronous speed.

2. **Equal-length cutting**

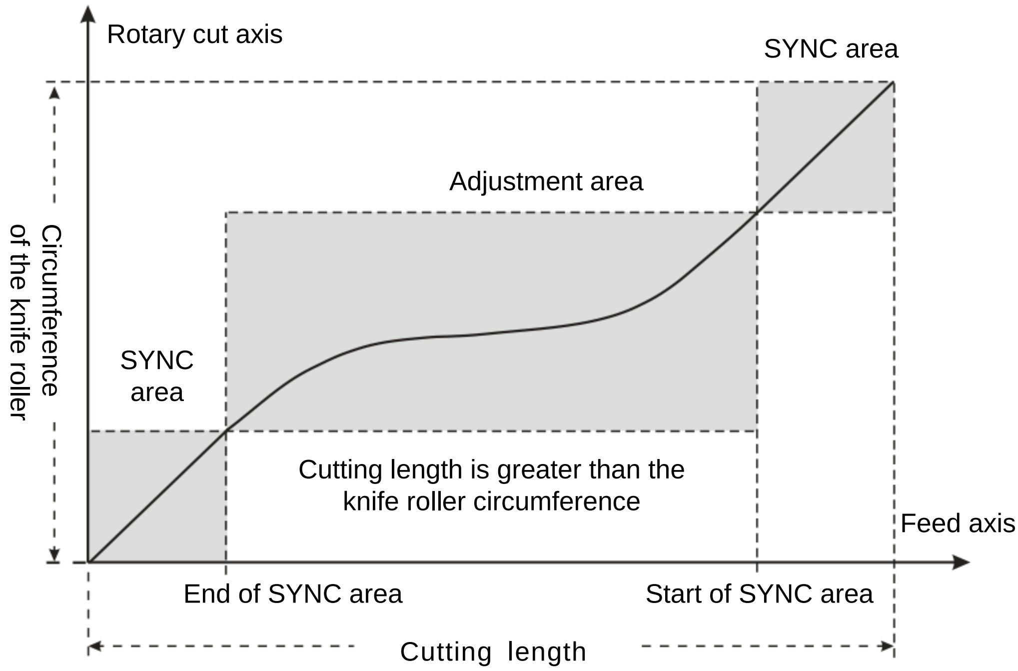
   When the cutting length is equal to the knife roller circumference, the rotary-cut curve for any cycle is shown below.



In this situation, the feed axis and rotary cut axis in SYNC area and non-SYNC area keep synchronous in speed. The rotary cut axis does not need to make any adjustment.

3. Long material cutting

When the cutting length is greater than the knife roller circumference, the rotary-cut curve for any cycle is shown below.

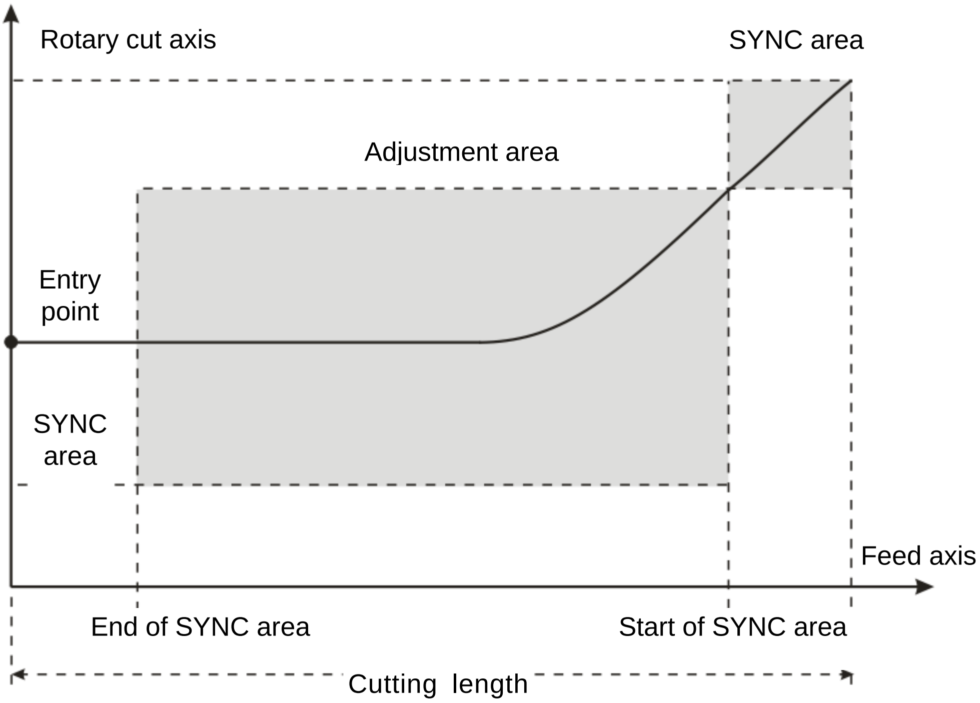


In this situation, the rotary cut axis should decelerate first in the adjustment area and then accelerate to the synchronous speed. If the cutting length is far greater than rotary cut roller circumference, the roller may decelerate to 0 and stay still for a while; and then accelerate up to the synchronous speed. The greater the cutting length is, the longer the roller stays.

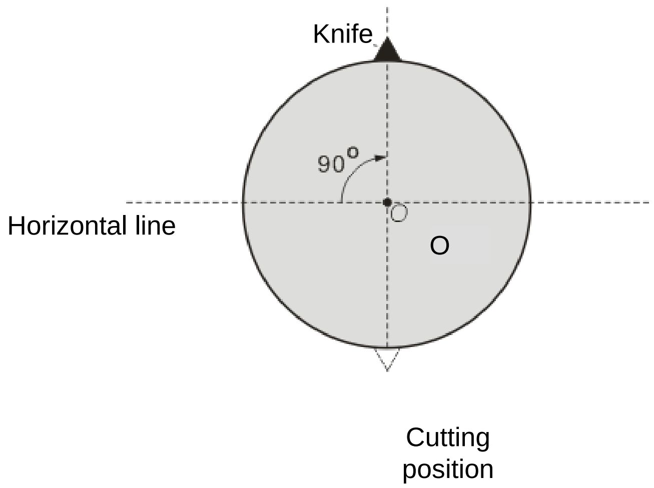
Additionally, when rotary cut function is started or broken off, the cam curves used are different.

4. The entry curve

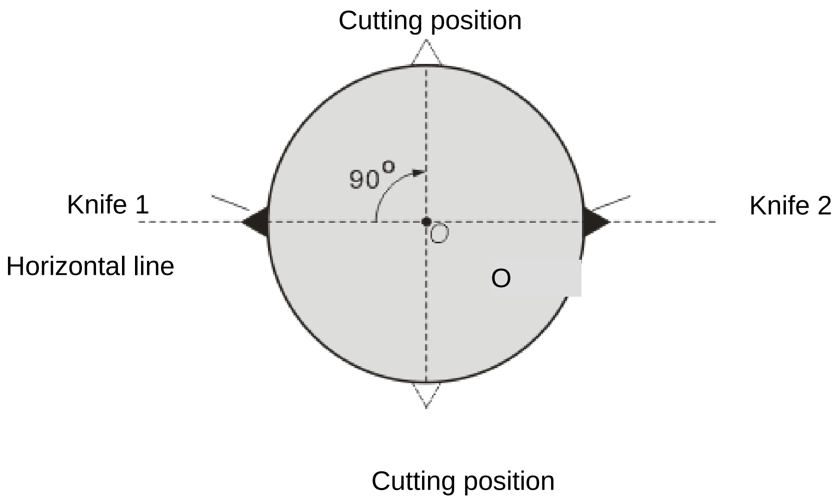
It is the rotary cut curve when rotary cut function is started.



The curve is the rotary cut function entry curve. When the rotary cut function is started up, the rotary cut axis will follow the feed axis to rotate according to the curve. The entry position is based on the rotary cut axis. For the single knife, the cutting position is directly below the rotary cut roller if the entry position is over the rotary cut roller in the following figure. Before the rotary cut function is started up, the knife must be turned to the upper of the rotary roller. Otherwise, the cutting may happen in the adjustment area.

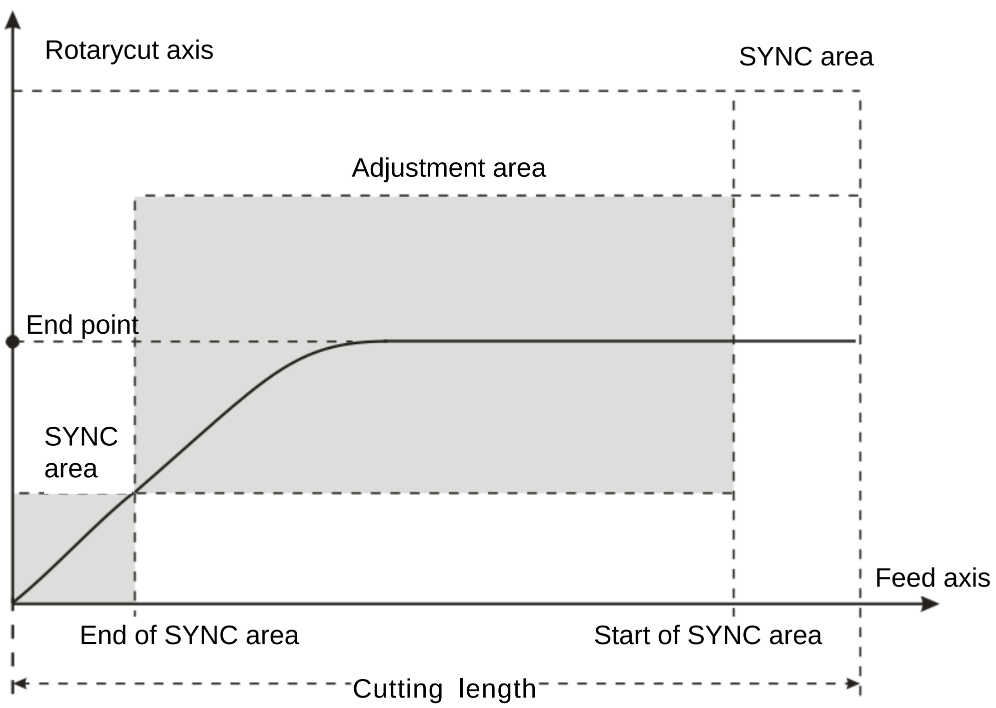


When the rotary roller is mounted with multiple knives, the distances between knives should be the same and the cutting position is at the center of the distance between knives. See the two-knife figure below.



5. **The end curve**

It is the rotary-cut curve when the rotary cut function is broken away.



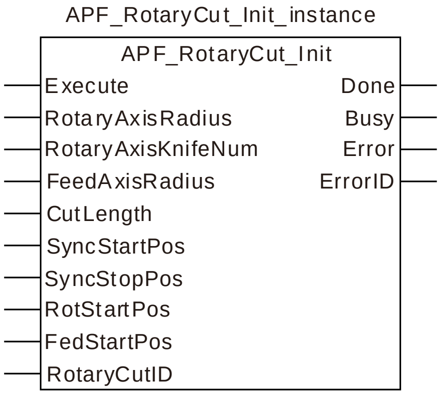
After the instruction “APF_RotaryCut_Out” is started up, the system will use the curve to make the rotary cut axis break away from the rotary cut state. Eventually, the knife stops at the end position as shown in the figure above.

The end position is based on the rotary cut axis. For the single knife, the end position is the entry position and it is also right above the rotary cut roller.

11.5.5 Rotary-cut Instructions

11.5.5.1 APF_RotaryCut_Init

FB/FC	Explanation	Applicable model
FB	APF_RotaryCut_Init is used for initializing the radius of the rotary-cut axis and feed axis, the cutting length, synchronous area and etc.	DVP15MC11T



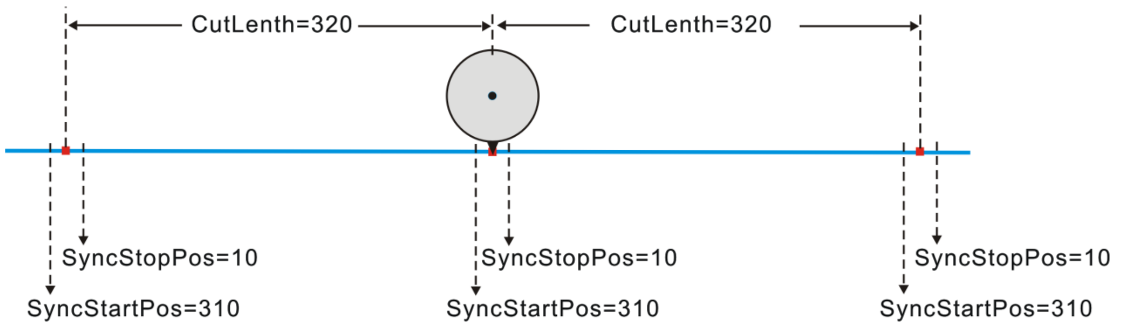
- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Execute	The instruction is executed when Execute changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
RotaryAxisRadius	The radius of the rotary cut axis, i.e. the distance from the center of the rotary cut roller to the knife.	LREAL	Positive number (The variable value must be set)	When *Execute* changes from FALSE to TRUE
RotaryAxisKnifeNum	The number of knives of the rotary cut axis, i.e. the number of knives mounted on the rotary cut roller	USINT	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
FeedAxisRadius	The radius of the feed axis; i.e. the radius length of the feed roller	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
CutLength	The cutting length of material	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
SyncStartPos	The start position of the sync area, i.e. the corresponding feed axis position when the sync area starts.	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE
SyncStopPos	The end position of the sync area, i.e. the corresponding feed axis position when the sync	LREAL	Positive number (The variable value must be set)	When Execute changes from FALSE to TRUE

Parameter name	Function	Data type	Valid range (Default)	Validation timing
	area ends.			
RotStartPos	Reserved	-	-	-
FedStartPos	Reserved	-	-	-
RotaryCutID	The number for a group of rotary cut instructions; a group of rotary cut instructions use the same number. Setting range: 1~8.	USINT	1~8 (The variable value must be set)	When *Execute* changes from FALSE to TRUE

Notes:

1. The value of "SyncStartPos" in SYNC area is always greater than that of "SyncStopPos" in SYNC area. As shown in the figure below, the cutting length is 320; "SyncStartPos" is 310 and "SyncStopPos" is 10.



2. The limit for SYNC area is that it must not be greater than the half of cutting length. In above figure, SYNC area is 20, and the half of the cutting length is 160.
3. The length parameters in the function are RotaryAxisRadius, FeedAxisRadius, CutLenth, SyncStartPos, and SyncStopPos with the same unit. For example, if the unit for one of the parameters is CM (centimeter), the units for other parameters must be CM as well.

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE when there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When initializing is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.

11

Name	Timing for changing to TRUE	Timing for changing to FALSE
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Function**

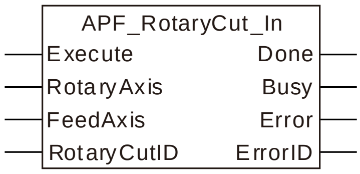
Before the rotary-cut relationoship is established, the instruction is used for initializing the radius of the rotary-cut axis and feed axis, cutting length, SYNC area and other parameters. After the instruction execution succeeds, relevant parameters will be downloaded so as to call for use in the established rotary-cut relationship.

After the rotary-cut relationship is established, the instruction can be used to modify the rotary-cut parameters. After the instruction execution is completed, the new parameters will be taken into effect in the next cycle.

11.5.5.2 APF_RotaryCut_In

FB/FC	Explanation	Applicable model
FB	APF_RotaryCut_In is used for establishing the rotary-cut relationship and specifying the axis No. of the rotary-cut axis and feed axis according to the application requirement.	DVP15MC11T

APF_RotaryCut_In_instance



- **Input Parameters**

Parameter name	Function	Data type	Valid range (Default)	Validation timing
Execute	The instruction is executed when *Execute* changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
RotaryAxis	The axis No. of the rotary-cut axis	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
FeedAxis	The axis No. of the feed axis. We suggest that the feed axis number should be less than the rotary cut axis number so that the rotary cut axis could better follow the feed axis for motion. The axis number can be set in order of 1~32 from small to large.	USINT	1~32 (The variable value must be set)	When *Execute* changes from FALSE to TRUE
RotaryCutID	The number for a group of rotary cut instructions; a group of rotary cut instructions use the same number. Setting range: 1~8.	USINT	1~8 (The variable value must be set)	When Execute changes from FALSE to TRUE

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE when there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

- **Output Update Timing**

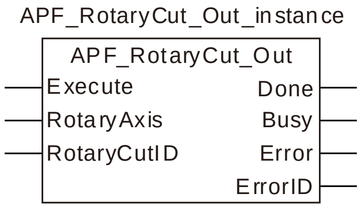
Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When the coupling between the rotary-cut axis and feed axis is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE.	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE.

- **Function**

APF_RotaryCut_In is used for building a rotary cut relationship and specifying the axis No. of the rotary-cut axis and feed axis according to the application requirement. The rotary cut axis will follow the feed axis for motion based on the rotary-cut curve after the instruction execution succeeds.

11.5.5.3 APF_RotaryCut_Out

FB/FC	Explanation	Applicable model
FB	APF_RotaryCut_Out is used for disconnecting the already established rotary-cut relationship between the rotary-cut axis and feed axis.	DVP15MC11T



- **Input Parameters**

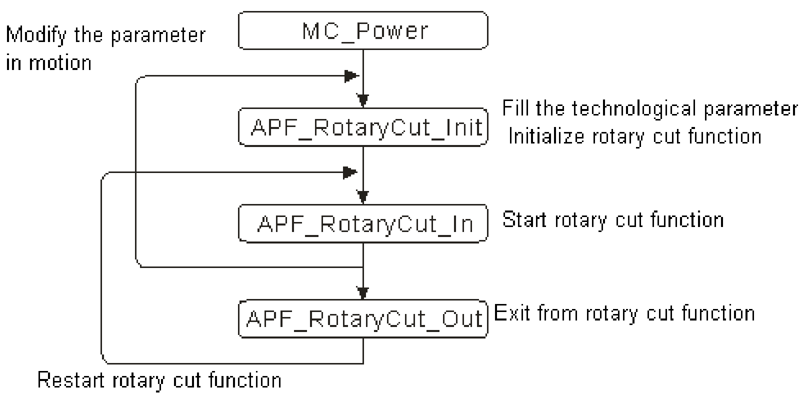
Parameter name	Function	Data type	Valid range (Default)	Validation timing
Execute	The instruction is executed when Execute changes from FALSE to TRUE.	BOOL	TRUE or FALSE (FALSE)	
RotaryAxis	The axis number of the rotary axis	USINT	1~32 (The variable value must be set)	When Execute changes from FALSE to TRUE
RotaryCutID	The number for a group of rotary cut instructions; a group of rotary cut instructions use the same number. Setting range: 1~8.	USINT	1~8 (The variable value must be set)	When Execute changes from FALSE to TRUE

- **Output Parameters**

Parameter name	Function	Data type	Valid range
Done	TRUE when the instruction is completed.	BOOL	TRUE / FALSE
Busy	TRUE when the instruction is being executed.	BOOL	TRUE / FALSE
Error	TRUE when there is an error.	BOOL	TRUE / FALSE
ErrorID	Contains error codes when an error occurs. Please refer to section 12.2 for the corresponding error code.	WORD	

Notes:

1. Control Sequence Chart of Rotary Cut Function



11

2. When the rotary cut function is performed, the rotary cut axis can only execute APF_RotaryCut_Out and MC_Stop instruction and other instructions are invalid.

- **Output Update Timing**

Name	Timing for changing to TRUE	Timing for changing to FALSE
Done	◆ When rotary-cut relationship disconnecting is completed.	◆ When *Execute* changes from TRUE to FALSE after the instruction execution is completed. ◆ *Done* changes to TRUE when the instruction execution is completed after *Execute* changes from TRUE to FALSE during the instruction execution. One cycle later, *Done* changes to FALSE.
Busy	◆ When *Execute* changes to TRUE	◆ When *Done* changes to TRUE. ◆ When *Error* changes to TRUE.
Error	◆ When an error occurs in the instruction execution or the input parameters for the instruction are illegal.	◆ When *Execute* changes from TRUE to FALSE

- **Function**

APF_RotaryCut_Out is used for disconnecting the already established rotary-cut relationship between the rotary-cut axis and feed axis. After the rotary-cut relationship is disconnected, the knife of the rotary-cut axis will stop at the entry position and will not follow the feed axis for motion any more.

The instruction has no impact on the motion of the feed axis.

11.5.6 Application Example of Rotary Cut Instructions

The section explains the setting of rotary cut parameters, establishment and disconnection of rotary cut relationship. The following is the programing example.

See the key parameters in the example as shown in the table below

Parameter name	Current value
RotaryAxis	2
FeedAxis	1
RotaryAxisRadius	10 (Unit: units)
RotaryAxisKnifeNum	1
FeedAxisRadius	20 (Unit: units)
CutLenth	30 (Unit: units)
SyncStartPos	19 (Unit: units)
SyncStopPos	1 (Unit: unit)

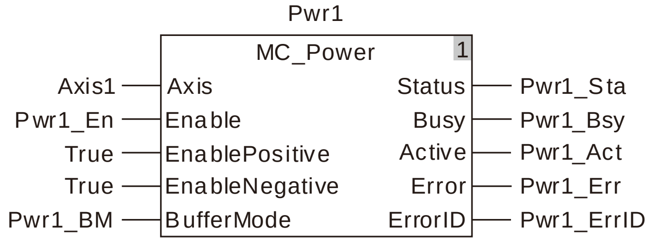
Programming Example

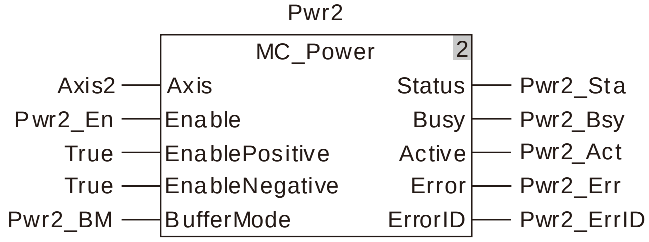
1. As Pwr1_En is TRUE, the servo of node address 1 turns "Servo On"; as Pwr2_En is TRUE, the servo of node address 2 turns "Servo On".

The variable table and program

Variable name	Data type	Initial value
Pwr1	MC_Power	
Axis1	USINT	1
Pwr1_En	BOOL	TRUE
Pwr1_BM	MC_Buffer_Mode	0
Pwr1_Sta	BOOL	TRUE
Pwr1_Bsy	BOOL	
Pwr1_Act	BOOL	
Pwr1_Err	BOOL	
Pwr1_ErrID	WORD	
Pwr2	MC_Power	
Axis2	USINT	1
Pwr2_En	BOOL	TRUE
Pwr2_BM	MC_Buffer_Mode	0
Pwr2_Sta	BOOL	TRUE
Pwr2_Bsy	BOOL	
Pwr2_Act	BOOL	
Pwr2_Err	BOOL	
Pwr2_ErrID	WORD	

11

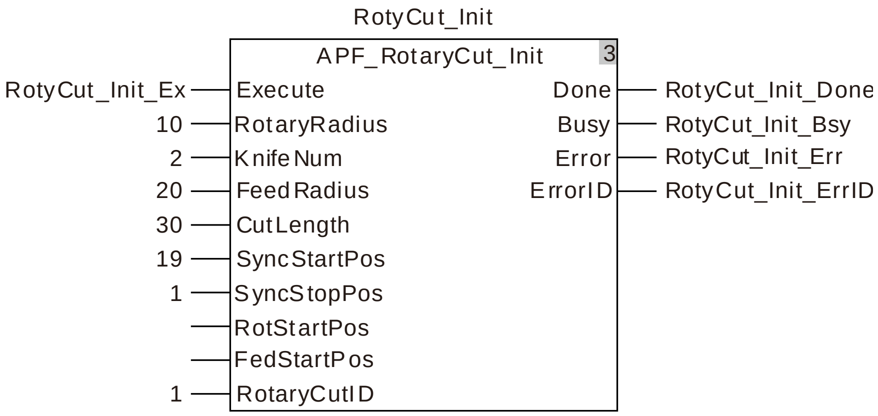




2. Set the rotary cut technology parameters. The radius of the rotary-cut axis is 10, knife quantity of the rotary-cut axis is 1, radius of the feed axis is 20 and cutting length of the feed axis is 30. The start position of SYNC area is 19, end position of SYNC area is 1, and the rotary cut group number is 1. When RotyCut_Init_Ex is TRUE, rotary cut technology parameters will be initialized.

The variable table and program

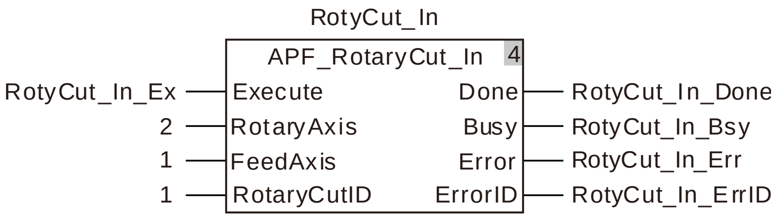
Variable name	Data type	Initial value
RotyCut_Init	APF_RotaryCut_Init	
RotyCut_Init_Ex	BOOL	TRUE
RotyCut_Init _Done	BOOL	TRUE
RotyCut_Init _Bsy	BOOL	
RotyCut_Init _Err	BOOL	
RotyCut_Init _ErrID	WORD	



3. When RotyCut_In_Ex is TRUE, the rotary-cut relationship starts being established. When RotyCut_In _Done is TRUE, it indicates the rotary-cut relationship between the rotary-cut axis and feed axis is made successfully. Servo 1 is the feed axis and servo 2 is the rotary-cut axis.

The variable table and program

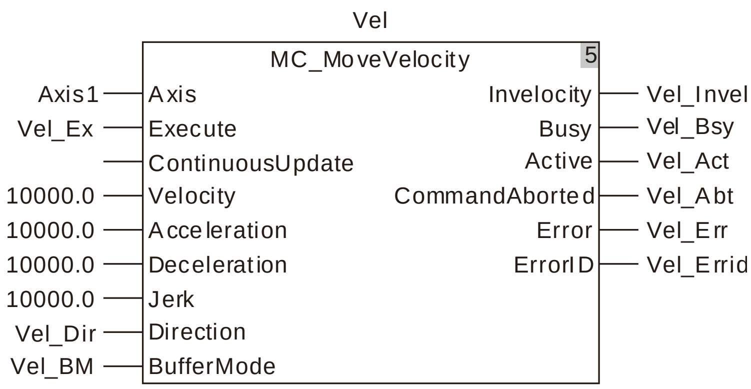
Variable name	Data type	Initial value
RotyCut_In	APF_RotaryCut_In	
RotyCut_In_Ex	BOOL	TRUE
RotyCut_In _Done	BOOL	TRUE
RotyCut_In _Bsy	BOOL	
RotyCut_In _Err	BOOL	
RotyCut_In _ErrID	WORD	



4. When Vel _Ex is TRUE, the feed axis starts to execute the velocity instruction. At the moment, the rotary-cut axis executes the rotary cut action based on the phase of the feed axis.

The variable table and program

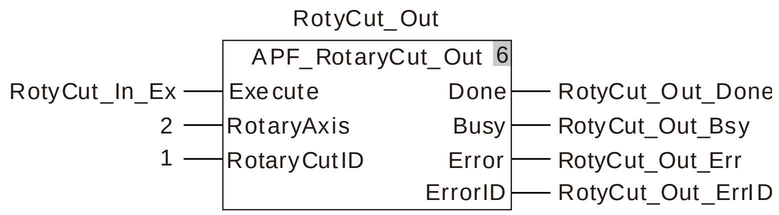
Variable name	Data type	Initial value
Vel	MC_MoveVelocity	
Axis1	USINT	1
Vel _Ex	BOOL	TRUE
Vel _Dir	MC_DIRECTION	1
Vel _BM	MC_Buffer_Mode	0
Vel _Invel	BOOL	
Vel _Bsy	BOOL	
Vel _Act	BOOL	
Vel _Abt	BOOL	
Vel _Err	BOOL	
Vel _ErrID	WORD	



5. When RotyCut_Out_Ex is TRUE, the rotary-cut axis starts to break away from the feed axis. When RotyCut_Out_Done is TRUE, it indicates that the rotary-cut axis breaks away successfully. After the rotary-cut axis breaks away from the feed axis, it will return to the entry point and the motion of the feed axis will not impact the rotary-cut axis any more.

The variable table and program

Variable name	Data type	Initial value
RotyCut_Out	APF_RotaryCut_Out	
RotyCut_Out_Ex	BOOL	TRUE
RotyCut_Out_Done	BOOL	TRUE
RotyCut_Out_Bsy	BOOL	
RotyCut_Out_Err	BOOL	
RotyCut_Out_ErrID	WORD	



12

Chapter 12 Troubleshooting

Table of Contents

12.1 Explanation of LED Indicators 12-2

12.2 Table of Error IDs in Motion Instructions 12-8

12.3 System Trouble Diagnosis through System Error Codes 12-12

This section mainly introduces the troubleshooting methods when any trouble occurs in DVP15MC11T.

12.1 Explanation of LED Indicators

- **PWR LED**

POWER LED indicates the state of the power supply for DVP15MC11T.

LED state	Explanation	How to deal with
Green light ON	Supply power is normal	No correction
LED OFF or blinking	Supply power is abnormal	Check if the supply power for DVP15MC11T is normal.

- **RUN LED**

RUN LED indicates the state of program execution in DVP15MC11T.

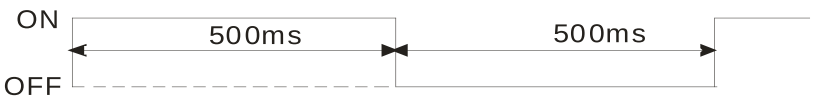
LED state	Explanation	How to deal with
Green light ON	DVP15MC11T is in RUN state.	No correction
LED OFF	DVP15MC11T is in STOP state.	Switch PLC to the RUN state according to demand

- **ERR LED**

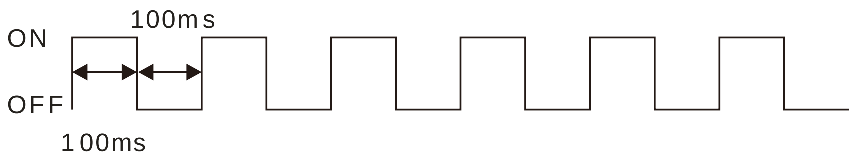
ERR LED indicates the error state of DVP15MC11T.

LED state	Explanation	How to deal with
LED OFF	DVP15MC11T is in the state of normal work.	No correction
Red light blinking	Errors in the program or configuration.	Get the detailed error information through the error diagnosis function.
Red light ON	Mistakes in hardware	Contact local technicians.

■ **ERR LED: Red light blinks.（1HZ）**



■ **ERR LED: Red light blinks quickly.（10HZ）**



- **SD LED**

SD LED is used for displaying the state of the SD card in DVP15MC11T.

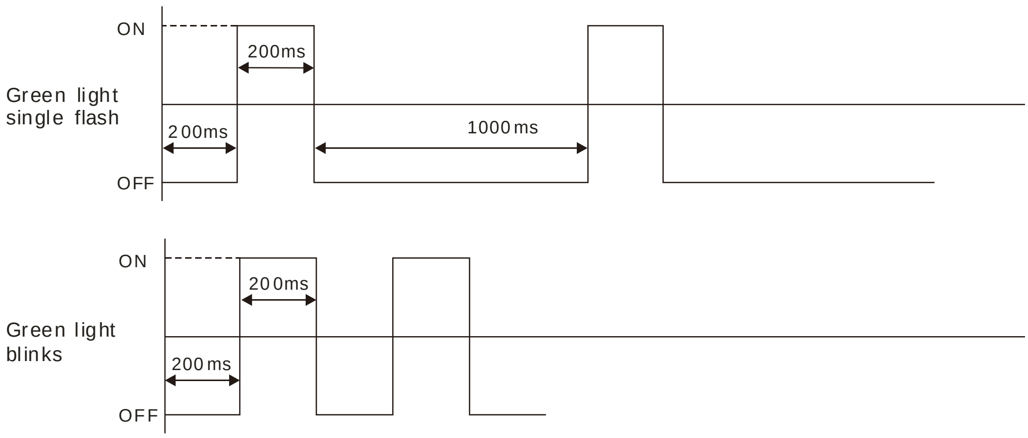
LED state	Explanation	How to deal with
LED OFF	1. No SD card is inserted to DVP15MC11T. 2. An error occurs in reading and writing the document	Insert the SD card or not according to the actual demand
Green light blinks quickly.	The SD card in DVP15MC11T is exchanging data	No correction
Green light ON	No data exchange for the SD card in DVP15MC11T.	

- **CAN1 (CANopen) LED**

■ **RUN LED**

LED state	Explanation	How to deal with
Green light single flash	CAN1 (CANopen) communication port is in STOP state.	PC is downloading the network configuration data. Wait till downloading is completed.
Green light blinking	CAN1 (CANopen) communication port is in Preoperational state.	1. Check if CANopen network bus cable connection is correct. 2. Check if the CANopen bus cable is Delta standard CANopen cable. 3. Check if the two ends of the CANopen bus have connected a terminal resistor respectively. 4. Check if the baud rate of the master is the same as that of other slaves. 5. Check if configured slaves have been actually connected to the network. 6. Check if some slave can not make the connection with the master. 7. Check if some slave is offline.
Green light ON	CAN1 (CANopen) communication port is in RUN state.	No correction

■ **RUN LED: Green light is in single flash and blinks as below.**



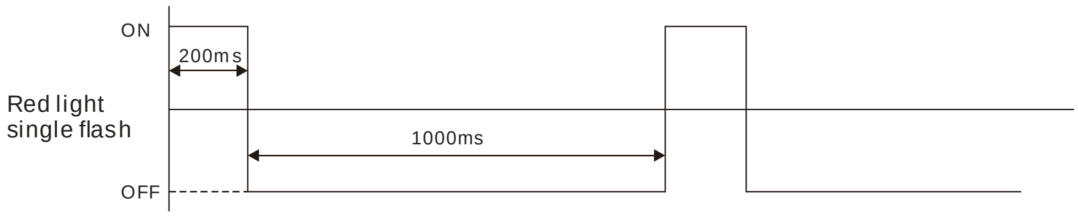
■ **ERR LED**

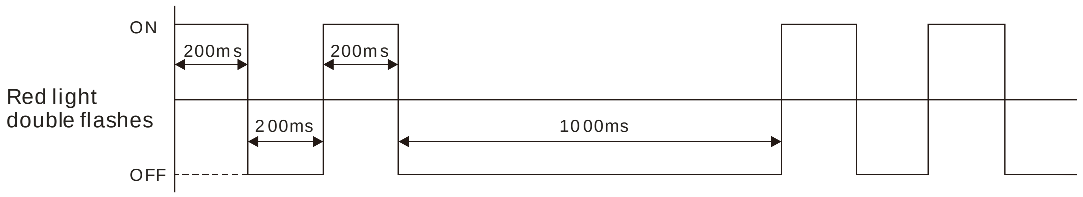
LED state	Explanation	How to deal with
LED OFF	PLC module is in the state of normal work.	No correction
Red light double flashes	Some slave is offline.	1. Check if the CANopen bus cable is Delta standard cable. 2. Check if the two ends of CANopen bus have connected a terminal resistor respectively. 3. Check if configured slaves have been actually connected to the network.

12

LED state	Explanation	How to deal with
		4. Check if the interference around CANopen bus cable is too strong.
Red light single flash	The bus error is out of the alert level.	1. Check if the CANopen bus cable connection is correct. 2. Check if the CANopen bus cable is Delta standard cable. 3. Check if the two ends of CANopen bus have connected a terminal resistor respectively. 4. Check if the baud rate of CANopen port is the same as that of other slaves. 5. Check if the interference around CANopen bus cable is too strong.
Red light ON	Bus-off	1. Check if the bus cable wiring in CANopen network is correct. 2. The baud rates of DVP15MC11T and other stations are same.

■ **ERR LED: Red light is in a single flash and double flashes as below.**





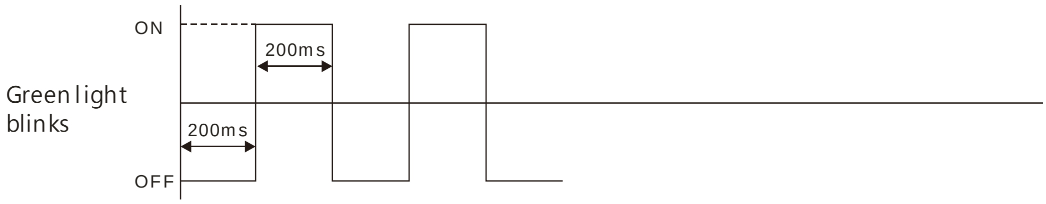
● **CAN2 (Motion) LED**

■ **RUN LED**

LED state	Explanation	How to deal with
LED OFF	No axis is configured for CAN2 (Motion) port in the software.	Configure the slave connected to Motion port to the master in the software.
Green light blinking	Not all of the axes configured for CAN2 (Motion) in the software have been online.	1. Check if the bus cable connection in the Motion network is correct. 2. Check if the Motion bus cable is Delta standard CANopen cable. 3. Check if the two ends of Motion bus have connected a terminal resistor respectively 4. Check if the baud rates of Motion port and slaves are same. 5. Check if the slaves configrued in the software have actually been connected to

		the network. 6. Check if some slave can not make the connection with the master. 7. Check if some slave is offline.
Green light ON	All of the axes configured for CAN2 (Motion) in the software have been online.	No correction

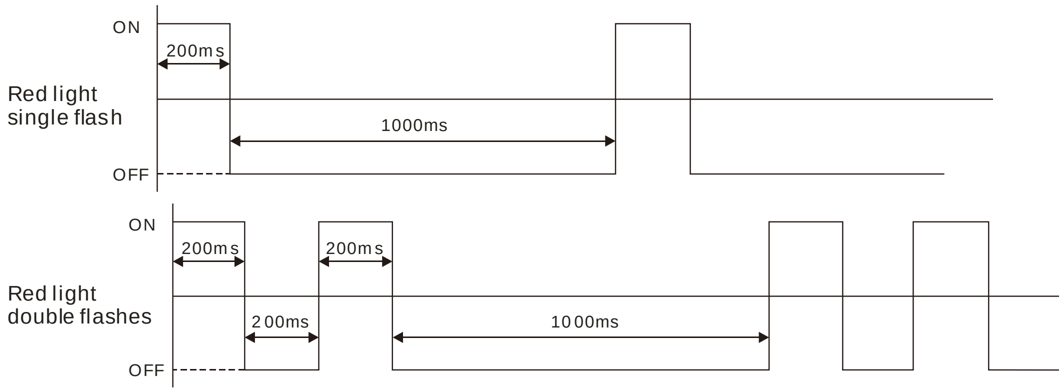
■ **RUN LED**



■ **ERR LED**

LED state	Explanation	How to deal with
LED OFF	PLC module is in the state of normal work.	No correction
Red light double flashes	Some slave is offline	1. Check if the Motion bus cable is Delta standard CANopen cable. 2. Check if the two ends of Motion bus have connected a terminal resistor respectively. 3. Check if all slaves configrued in the software have actually been connected to the network. 4. Check if the interference around CANopen bus cable is too strong.
Red light single flash	The bus error is out of the alert level.	1. Check if the bus cable connection in the Motion network is correct. 2. Check if the Motion bus cable is Delta standard CANopen cable. 3. Check if the two ends of Motion bus have connected a terminal resistor respectively. 4. Check if the baud rates of Motion port and slaves are same. 5. Check if the interference around Motion bus cable is too strong.
Red light ON	Bus-off	

■ **ERR LED: Red light is in a single flash and double flashes as below.**



12

- **LAN1 LED**

LAN1 LED indicates the network state of the first Ethernet communication port of the left side of DVP15MC11T.

LED	State	Indication
Yellow light	Blinking	Data are being sent or received via the first Ethernet port of DVP15MC11T.
	OFF	The first Ethernet port of DVP15MC11T is not connected to the Ethernet network.

- **LAN2 LED**

LAN2 LED indicates the network state of the second Ethernet communication port of the left side of DVP15MC11T.

LED	State	Indication
Yellow light	ON	Data are being sent or received via the second Ethernet port of DVP15MC11T.
	OFF	The second Ethernet port of DVP15MC11T is not connected to the Ethernet network.

- **RS232 LED**

RS232 LED, the RS-232 communication indicator of DVP15MC11T indicates the communication state of RS-232 port of DVP15MC11T.

LED state	Indication
Yellow light blinking	There are response data via RS-232 port.
LED OFF	There are no response data via RS-232 port.

- **RS485 LED**

RS485 LED, the RS-485 communication indicator of DVP15MC11T indicates the communication state of RS-485 port of DVP15MC11T.

LED state	Indication
Yellow light blinking	There are response data via RS-485 port.
LED OFF	There are no response data via RS-485 port.

- **Input point LED**

There are 16 input point LED indicators for showing if DVP15MC11T’s digital input points are ON or OFF.

LED state	Indication
Red light ON	Input point is ON.
LED OFF	Input point is OFF.

- **Output point LED**

  There are 8 output point LED indicators for showing if DVP15MC11T's digital output points are ON or OFF.

LED state	Indication
Red light ON	Output point is ON.
LED OFF	Output point is OFF.

12

12.2 Table of Error IDs in Motion Instructions

When an error occurs in the motion instruction, the value of ErrorID can be seen as follows for analysis of the cause and troubleshooting.

ErrorID		Meaning	How to deal with
Hex	Decimal		
0x1001	4097	The axis No. exceeds the valid range.	Make sure that the value of the input variable, *Axis* is within the range of 1~32.
0x1002	4098	The acceleration exceeds the valid range.	Make sure that the value of the input variable, *Acceleration* is a positive number.
0x1003	4099	The deceleration exceeds the valid range.	Make sure that the value of the input variable, *Deceleration* is a positive number.
0x1004	4100	The change rate of the acceleration exceeds the valid range.	Make sure that the value of the input variable, *Jerk* is a positive number.
0x1005	4101	The velocity exceeds the valid range.	Make sure that the value of the input variable, *Velocity* is a nonzero value.
0x1006	4102	The position value exceeds the valid range.	Make sure that the value of the input variable, *Position* of MC_MoveAbsolute is not greater than the value of Modulo among axis parameters.
0x1007	4103	The direction value exceeds the valid range.	Modify the value of the input variable, *Direction* into that which can be set in the instruction.
0x1008	4104	The buffermode value exceeds the valid range.	Modify the value of the input variable, *BufferMode* into that which can be set in the instruction.
0x1009	4105	The input value for reference position type is wrong.	Modify the value of the input variable, *ReferenceType* into that which can be set in the instruction.
0x100A	4106	The timing for executing MC_SetPosition is improper.	Change the timing of executing MC_SetPosition. Do not execute MC_SetPosition while MC_Home or MC_Stop is being executed.
0x100B	4107	The number of e-cam table is incorrect.	Modify the value of the input variable, *CamTable* into that of CamId set in the software.
0x100C	4108	The axis No. of the master axis is incorrect.	Make sure that the value of the input variable, *Axis* is within the range of 1~32.
0x100D	4109	The input value of the engagement mode is wrong.	Modify the value of the input variable, *StartMode* into that which can be set in the instruction.
0x100E	4110	The value of the master scaling is incorrect.	Make sure that the value of the input variable, *MasterScaling* is a positive number.
0x100F	4111	The value of the slave scaling is incorrect.	Make sure that the value of the input variable, *SlaveScaling* is a nonzero value.
0x1010	4112	The chosen position source of the master axis is wrong.	Modify the value of the input variable, *MasterValueSource* into that which can be set for the instruction.
0x1011	4113	Conflict in the axis No. of the master and slave axes.	Modify the values of the input variables, *Master* and *Slave* into different values.
0x1012	4114	Wrong e-gear numerator value	Modify the value of the input variable, *Numerator* into a nonzero value.
0x1013	4115	Wrong e-gear denominator value	Modify the value of the input variable, *Denominator* into a nonzero value.
0x1014	4116	The value of VelFactor is incorrect.	Modify the value of the input variable, *VelFactor* into that which can be set in the instruction.

ErrorID		Meaning	How to deal with
Hex	Decimal		
0x1015	4117	SDO Timeout in CANopen network (or Motion network)	1. Check if the slave specified in the instruction exists. 2. Check if the connection between the accessed slave and CANopen port or Motion port is normal. 3. Check if the baud rates of CANopen port or Motion port and the accessed slave are same.
0x1016	4118	The input parameter error of the SDO instruction	Check if the input parameter settings of the SDO instruction are reasonable. For example, see whether the accessed Index and Subindex exist or not and whether the value of DataType is legal or not.
0x1017	4119	Other faults in SDO in CANopen network (or Motion network).	Check if slaves are in normal work.
0x1018	4120	The value of TriggerInput of the position-capture instruction DMC_TouchProbe is wrong.	Modify the value of the input variable, *TriggerInput*. The value can be set within the range of 0~15 respectively representing I0~I7 and I10~I15.
0x1019	4121	The input point specified by TriggerInput of the instruction DMC_TouchProbe has been used in another DMC_TouchProbe.	Modify the value of *TriggerInput* of the instruction into one value which has not been used yet.
0x101A	4122	Windowonly of DMC_TouchProbe is abnormal.	Modify the values of *Firstops* and *Lastops* into those within the valid range.
0x101B	4123	Two DMC_TouchProbe instructions are performed for capturing the position of the same axis at the same time.	Prevent two DMC_TouchProbe instructions from capturing the position of the same axis at the same time.
0x101C	4124	The setting value of Mode of DMC_TouchProbe is incorrect.	Modify the value of the input variable, *Mode* into that which can be set in the instruction.
0x101D	4125	The axis specified by DMC_TouchProbe is not an encoder axis.	Modify the value of the input variable, *Axis* into the axis No. of the encoder axis which has been configured.
0x101E	4126	The value of ActivationPosition of MC_CamIn is incorrect.	Modify the value of the input variable, *ActivationPosition* into that which can be set in the instruction.
0x1020	4128	The used axis is not configured to the Motion network in the software.	Modify the value of the input variable, *Axis* into the axis No. which has been configured in the Motion network.
0x1021	4129	The radius of the rotary-cut axis is incorrect.	Modify the value of the input variable, *RotaryAxisRadius*. It should be greater than 0.
0x1022	4130	The radius of the feed axis is incorrect.	Modify the value of the input variable, *FeedAxisRadius*. It should be greater than 0.
0x1023	4131	The cutting length is incorrect.	Modify the value of the input variable, *CutLenth* of APF_RotaryCut_Init. It should be greater than 0.
0x1024	4132	The value of SyncStartPos is incorrect.	Modify the value of the input variable, *SyncStartPos* of APF_RotaryCut_Init. It should be between 0 and the cutting length.
0x1025	4133	The value of SyncStopPos is	Modify the value of the input variable,

ErrorID		Meaning	How to deal with
Hex	Decimal		
		incorrect.	*SyncStopPos* of APF_RotaryCut_Init. It should be between 0 and the cutting length.
0x1026	4134	The settings of SyncStopPos and SyncStartPos are incorrect.	The value of the input variable, *SyncStopPos* should be less than that of SyncStartPos of the instruction.
0x1027	4135	The value of RotCutID is incorrect.	The value of the input variable, *RotCutID* should be in the range of 1~8.
0x1028	4136	The value of RotaryAxisKnifeNum is incorrect.	The value of the input variable, *RotaryAxisKnifeNum* should be in the range of 1~16.
0x1029	4137	The inner state of rotary cut is illegal.	Modify the parameter values for initializing rotary cut.
0x103A	4154	Rotary cut initializing fails.	Since APF_RotaryCut_Init has not been executed, please execute APF_RotaryCut_Init first and then execute APF_RotaryCut_In.
0x103B	4155	The axis is offline and the capture function can not be performed	Execute the capture instruction after the axis is connected normally.
0x103C	4156	The value of *MasterOffset* of MC_CamIn is greater than the master axis cam cycle range.	Modify the value of *MasterOffset* into that between the negative number and positive number of the master axis cam cycle range. (The master axis cam cycle range= Maximum master axis cam cycle- Mimimum master axis cam cycle)
0x103D	4157	The value of *SlaveOffset* of MC_CamIn is greater than the slave axis cam cycle range.	Modify the value of *SlaveOffset* into that between the negative number and positive number of the slave axis cam cycle range. (The slave axis cam cycle range= Maximum slave axis cam cycle- Minimum slave axis cam cycle)
0x103E	4158	The *Depth* value of the instruction is out of the range.	Modify the value of the input *Depth* in order not to exceed the range.
0x103F	4159	The *VelOverride* value range of the instruction is illegal.	Modify the value of the input *VelOverride* in order not to exceed the range.
0x1040	4160	The file code is illegal.	Modify the value of the input *NCFile* into a proper code value.
0x1041	4161	DMC_SetTorque is executed when the axis is not in Standstill state.	Make sure that DMC_SetTorque is executed when the axis is in Standstill state.
0x1042	4162	The execution of MC_Reset fails.	1. Check if the axis specified by MC_Reset exists. 2. MC_Reset is executed after the servo alarm is cleared.
0x1043	4163	The execution of an instruction leads to the result that the axis position exceeds the range set in the software.	Modify the instruction to make sure that the final position does not exceed the software limit range.
0x1044	4164	The cam curve specified by MC_CamIn is not built in the software.	Check if the *CamTable* value of MC_CamIn can correspond to the cam curve built in the software.
0x1045	4165	Axis group ID error	Check if the value of *GroupID* is within the range of 1~8.
0x1046	4166	Mode input value error	The value of *Mode* for the instruction can only be set to 0
0x1047	4167	The number of the From/To	Check the value of *Station* and the number of the

ErrorID		Meaning	How to deal with
Hex	Decimal		
		instruction is wrong.	instruction are correct.
0x1048	4168	An error in the number of CR registers which are read and written by From/To.	Check if the value of *Num* is within the range of 1~64.
0x1049	4169	The input variable of the instruction is not set.	The input variable of the instruction must be set.
0x104A	4170	No response transmitted to From/To instruction	Check if the connection between modules is proper and if the extension module works normally.
0x104B	4171	Empty CNC file	Check if the value of NCFile is correct and the corresponding CNC file is empty.
0x104C	4172	CNC file analysis error	Check if there is any error in the CNC file compiling.
0x2001	8193	The axis is disabled by means of MC_Power instruction when it is not in Standstill state.	Make the axis disabled by using MC_Power instruction when the axis is in Standstill state.
0x2002	8194	The instruction cannot be executed due to the limitation of the motion direction.	Set *EnablePositive* and *EnableNegative* of MC_Power to TRUE to cancel the limitation of the motion direction of the axis.
0x2004	8196	MC_HaltSuperimposed cannot be performed when MC_MoveSuperimposed is not executed yet.	Modify the sequence of execution of MC_HaltSuperimposed. The execution of MC_HaltSuperimposed should be conducted in the process of performing MC_MoveSuperimposed.
0x2100	8448	The state machine limits that the function cannot be performed.	Modify the timing for execution of the instruction. Refer to the state machine in section 10.4 for the execution of motion instructions.
0x2101	8449	The buffer register is full.	The *BufferMode* of a motion control instruction only supports one switch for changing the time to execute current instruction and avoiding the circumstance that another instruction is also waiting to execute (*BufferMode* is not 0) while one instruction is waiting to execute (*BufferMode* is not 0).
0x2102	8450	Buffer function cannot be performed in the instruction.	The instruction cannot be operated in BufferMode.
0x3001	12289	An error in axis type setting	Modify the axis type on the axis configuration window.
0x3002	12290	Servo alarm	Have the control over the servo after clearing the servo alarm.
0x3003	12291	Servo Timeout	Check if the connection between the controller and servo is OK.
0x3004	12292	The command position exceeds the limit position set in the software.	Check if the set software limit position is proper or disable the software limit position.
0x3005	12293	The process from RUN to STOP occurs in the controller (during the execution of a motion instruction)	Clear the error with the MC_Reset instruction and then execute other motion instruction.

12

12.3 System Trouble Diagnosis through System Error Codes

When the ERR indicator of DVP15MC11T blinks or is always ON, users can get to know the cause of an error and shoot the trouble through selecting menu Device > Diagnosis Information… in the CANopen Builder software of version 6.0 or above.

System error code		Explanation	Correction
Hexadecimal	Decimal		
0x1000	4096	Internal RAM detection failed	Contact local technicians if the error still exists after repower on.
0x1001	4097	Internal Flash detection failed	
0x1002	4098	The extension port detection failed	
0x1003	4099	Internal voltage is abnormal (LV)	Adjust input voltage to 24V at the power port.
0x1004	4100	Flash initializing failed	Contact local technicians if the error still exists after repower on.
0x1005	4101	Flash ID detection failed.	
0x1007	4103	The access to flash failed in the Ethernet area.	Contact local technicians if the problem still exists after re-downloading the program and restoring the setting to the factory setting.
0x1008	4104	The access to flash failed in the extension area.	
0x1009	4105	The access to flash failed in the program area.	
0x100A	4106	The access to flash failed in the CAN motion area.	
0x100B	4107	The access to flash failed in the Task area.	
0x100C	4108	The access to flash failed in the CANopen communication.	
0x100D	4109	The access to flash failed in the hardware configuration.	
0x100E	4110	The access to flash faild in the CAM area.	
0x100F	4111	The access to flash (the flash management table) failed.	
0x1010	4112	The access to flash (sheet 1 in the flash management table) failed.	
0x1011	4113	The access to flash (sheet 2 in the flash management table) fails.	
0x1012	4114	The reading of flash failed.	
0x1013	4115	The writing in flash failed.	
0x1014	4116	The erasing of the content in flash failed.	
0x1015	4117	CNC file ID is out of the allowed range	Check if the CNC file ID is larger than 64. Update the software and redownload the program if the error still exists after redownloading the program.
0x1016	4118	The size of CNC file exceeds the range	CNC file is too large in size. Diminish the size and redownload the program.
0x1017	4119	The position of incremental encoder 1 changes dramatically in	Check if the input of the encoder is too fast or

System error code		Explanation	Correction
Hexadecimal	Decimal		
		short time.	enlarge the resolution of the encoder.
0x1018	4120	The position of incremental encoder 2 changes dramatically in short time.	Check if the input of the encoder is too fast or enlarge the resolution of the encoder.
0x1019	4121	System stack is used up.	There are too many intermediate variables in the program. Modify the program.
0x101A	4122	The Retain file is too large.	There are too many Retain variables. Decrease the number of Retain variables and then redownload the program.
0x101B	4123	The access to Retain file failed.	Redownload the program after restoring the system to the fatory setting.
0x1401	5121	The initializing of Ethernet LAN1 failed.	Contact local technicians if the error still exists after repower on.
0x1402	5122	The Ethernet LAN1 buffer overflows	
0x1403	5123	The data sending failed through the Ethernet LAN1.	
0x1404	5124	Sending the buffer memory distribution through Ethernet failed.	
0x1601	5633	The Ethernet LAN2 initializing failed.	Contact local technicians if the error still exists after repower on.
0x1602	5634	The Ethernet LAN2 buffer overflows.	
0x3000	12288	The number of inputs or the number of outputs is greater than the limit 32.	Reset the number of input and output variables in the self-defined POU and make sure the number of input or output variables does not exceed 32.
0x3001	12289	The capacity for one POU is more than 65535 bytes.	Change the capacity of variables in a POU to reduce the variable occupation in the memory.
0x3002	12290	The number of POUs is more than 1000.	Reduce the number of POUs called by the task and re-download the program.
0x3003	12291	The POU type is illegal.	Update the software if the error still exists after re-compiling and re-downloading the program and repowering the product.
0x3004	12292	The types of parameters in the program are illegal.	
0x3005	12293	Variable's offset address error in the program	
0x3006	12294	The data types of parameters are illegal in the program.	
0x3007	12295	The jump range in a program is illegal.	
0x3008	12296	Program memory allocation alignment is incorrect.	
0x3009	12297	Virtual axis encoder memory alignment is incorrect.	
0x300A	12298	The Bit accessed exceeds the range. (Only Bit0~Bit7 can be accessed.)	Update the software if the error still exists after re-compiling and re-downloading the program and repowering the product.

System error code		Explanation	Correction
Hexadecimal	Decimal		
0x300B	12299	It is detected that data types are illegal in the program.	Update the software if the error still exists after re-compiling and re-downloading the program and repowering the product.
0x300C	12300	The length of data type String is too large.	The number of characters in String data type is too large. Update the software if the error still exists after modifying the program, re-compiling and re-downloading the program.
0x300D	12301	Illegal addressing method for variables	Update the software if the error still exists after re-compiling and re-downloading the program and repowering the product.
0x3020	12320	The checksum of the downloaded CAN Motion configuration is illegal.	Update the software if the error still exists after re-compiling and re-downloading the program and repowering the product.
0x3021	12321	The checksum of the downloaded extension configuration is illegal.	
0x3022	12322	The checksum of the downloaded program is illegal.	
0x3023	12323	The checksum of the downloaded task data is illegal.	
0x3024	12324	The checksum of the downloaded CANopen data is illegal.	
0x3025	12325	The checksum of the downloaded hardware configuration is illegal.	
0x3026	12326	Watchdog timeout	Check if the program is correct or there is a loop of which the program execution can not get out when the program execution timeout occurs.
0x3027	12327	Calling the axis state machine failed.	Contact local technicians if the error still exists after redownloading the program and restoring to the factory setting.
0x3028	12328	CNC list analysis error	Check if the CNC file is correct and redownload the program.
0x3029	12329	CNC file analysis error	Check if the CNC file is correct and redownload the program.
0x3050	12368	The actual time for executing the priority 0 task exceeds the set watchdog timeout time.	Contact local distributors
0x3051	12369	The actual time for executing the priority 1 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3052	12370	The actual time for executing the priority2 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3053	12371	The actual time for executing the priority 3 task exceeds the set	1. Reset the watchdog time to a larger

System error code		Explanation	Correction
Hexadecimal	Decimal		
		watchdog timeout time.	value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3054	12372	The actual time for executing the priority 4 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3055	12373	The actual time for executing the priority 5 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3056	12374	The actual time for executing the priority 6 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Revise the program or re-download the revised program.
0x3057	12375	The actual time for executing the priority 7 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Revise the program or re-download the revised program.
0x3058	12376	The actual time for executing the priority 8 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3059	12377	The actual time for executing the priority 9 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x305A	12378	The actual time for executing the priority 10 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x305B	12379	The actual time for executing the priority 11 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the

12

System error code		Explanation	Correction
Hexadecimal	Decimal		
			program.
0x305C	12380	The actual time for executing the priority 12 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x305D	12381	The actual time for executing the priority 13 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x305E	12382	The actual time for executing the priority 14 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x305F	12383	The actual time for executing the priority 15 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3060	12384	The actual time for executing the priority 16 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3061	12385	The actual time for executing the priority 17 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3062	12386	The actual time for executing the priority 18 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3063	12387	The actual time for executing the priority 19 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3064	12388	The actual time for executing the priority 20 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop

System error code		Explanation	Correction
Hexadecimal	Decimal		
			in the program which the task calls. 3. Redownload it after modifying the program.
0x3065	12389	The actual time for executing the priority 21 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3066	12390	The actual time for executing the priority 22 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3067	12391	The actual time for executing the priority 23 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3068	12392	The actual time for executing the priority 24 task exceeds the set watchdog timeout time.	1. Reset the watchdog time to a larger value for the task. 2. Check whether there is any infinite loop in the program which the task calls. 3. Redownload it after modifying the program.
0x3069	12393	The actual time for executing the priority 25 task exceeds the set watchdog timeout time.	Contact local technicians.
0x306A	12394	The actual time for executing the priority 26 task exceeds the set watchdog timeout time.	Contact local technicians.
0x306B	12395	The actual time for executing the priority 27 task exceeds the set watchdog timeout time.	Contact local technicians.
0x306C	12396	The actual time for executing the priority 28 task exceeds the set watchdog timeout time.	Contact local technicians.
0x306D	12397	The actual time for executing the priority 29 task exceeds the set watchdog timeout time.	Contact local technicians.
0x306E	12398	The actual time for executing the priority 30 task exceeds the set watchdog timeout time.	Contact local technicians.
0x306F	12399	The actual time for executing the priority 31 task exceeds the set watchdog timeout time.	Contact local technicians.
0x5000	20480	Extension communication checking failed.	Contact local technicians if the error still exists after repower on.
0x5001	20481	Extension communication timeout	

System error code		Explanation	Correction
Hexadecimal	**Decimal**		
0x5100	20736	Extension configuration is inconsistent.	Contact local technicians if the error still exists after repower on.
0x5200	20992	The buffer for receiving CANopen data is full.	Adjust the CANopen configuration and check the task setup.
0x5201	20993	The buffer for sending CANopen data is full.	
0x5300	21248	The buffer for receiving CAN Motion data is full.	Adjust the CAN Motion configuration and check the task setup.
0x5301	21249	The buffer for sending CAN Motion data is full.	

A

Appendix A Modbus Communication

Table of Contents

A.1 Message Format in ASCII Mode A-2

A.2 Message Format in RTU Mode A-5

A.3 Modbus Function Codes Supported A-7

A.4 Modbus Exception Response Code Supported A-7

A.5 Introduction to Modbus Function Codes A-8

A.6 Table of Registers and Corresponding Modbus addresses A-15

A.1 Message Format in ASCII Mode

- Communication data structure

Field name	Components	Explanation
Start character	STX	Start character ":", the corresponding ASCII code: 0x3A
Communication address	ADR 1	Communication address consists of two ASCII codes.
	ADR 0	
Function code	CMD 1	Function code consists of two ASCII codes.
	CMD 0	
Data	DATA （0）	Data content consists of 2n ASCII codes, n≤205.
	DATA （1）	
	……….	
	DATA （n-1）	
LRC Check	LRC CHK 1	LRC check consists of two ASCII codes.
	LRC CHK 0	
End character	END1	End character consists of two ASCII codes. END1 = CR (0x0D), END0 = LF (0x0A)
	END0	

The corresponding relation between hexadecimal character and ASCII code:

Hexadecimal character	"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"
ASCII code	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37
Hexadecimal character	"8"	"9"	"A"	"B"	"C"	"D"	"E"	"F"
ASCII code	0x38	0x39	0x41	0x42	0x43	0x44	0x45	0x46

- ADR (Communication address)

The valid range of communication address: 0～254.

Communication address: 0 means the broadcast message is sent to all slaves and the slaves which have received the message will not make any response. If communication address is not 0, slaves will respond to master after receiving the message normally. For instance, ASCII codes for the communication address of 16 are denoted below.

Decimal 16 is equal to hexadecimal 10. (ADR 1, ADR 0) ='10', '1'=31H, '0' = 30H

- Function code and data

The data format is determined by function codes. For example, to read the two continuous address data with hexadecimal 0x0000 as the start address in DVP15MC11T. The communication address of DVP15MC11T is 1, 0x0000 is the Modbus address of %MW0 in DVP15MC11T PLC.

The data explanation is shown as below:

PC→DVP15MC11T

3A 30 31 30 33 30 30 30 30 30 30 30 32 46 41 0D 0A

DVP15MC11T→PC

3A 30 31 30 33 30 34 30 30 30 31 30 30 30 32 46 35 0D 0A

■ Request message:

Field name	Field character	ASCII code corresponding to field character
Start character	“ : ”	3A
Communication address: 01	“0”	30
	“1”	31
Function code: 03	“0”	30
	“3”	33
Start address: 0x0000	“0”	30
	“0”	30
	“0”	30
	“0”	30
Data number (Counted by word): 2	“0”	30
	“0”	30
	“0”	30
	“2”	32
LRC check code: 0xFA	“F”	46
	“A”	41
End character 1	CR	0D
End character 0	LF	0A

■ Response message:

Field name	Field character	ASCII code corresponding to field character
Start character	“ : ”	3A
Communication address: 01	“0”	30
	“1”	31
Function code: 03	“0”	30
	“3”	33
Data number (Counted by byte):	“0”	30
	“4”	34
Read content of 0x1000 address	“0”	30
	“0”	30
	“0”	30
	“1”	31
Read content of 0x1001 address	“0”	30
	“0”	30
	“0”	30
	“2”	32

Field name	Field character	ASCII code corresponding to field character
LRC check code: 0xF5	“F”	46
	“5”	35
End character 1	CR	0D
End character 0	LF	0A

- LRC check (Check sum)

LRC check code is the value by firstly getting the inverse values of every bit of the result value of addition operation of the data from communication ID to the last data content (Hex.) and then adding 1 to the final inverse value.

For instance, LRC check code value: 0xFA. The method of calculating LRC check code value: 0x01+ 0x03 + 0x00 + 0x00 + 0x00 + 0x02 = 0x06, the result 0xFA is got by getting the inverse values of every bit of 0x06 and then adding 1 to the final inverse value.

Field name	Field character	ASCII code corresponding to field character
Start character	“ : ”	3A
Communication address: 01	“0”	30
	“1”	31
Function code: 03	“0”	30
	“3”	33
Start data address: 0x0000	“0”	30
	“0”	30
	“0”	30
	“0”	30
Data number (Counted by word):2	“0”	30
	“0”	30
	“0”	30
	“2”	32
LRC check code: 0xFA	“F”	46
	“A”	41
End character 1: CR	CR	0D
End character 0: LF	LF	0A

A.2 Message Format in RTU Mode

- Communication data structure

Start	No input data for more than 10ms
Communication address	Slave address: 8-bit binary address
Function code	Function code: 8-bit binary address
Data (n-1)	Data content n × 8 bit binary data, n<=202
…….	
Data 0	
Low byte of CRC check	CRC check sum
High byte of CRC check	
End	CRC check sum is composed of two 8-bit binary data

- Communication address

  The range of a valid communication address is 0～254. The communication address 0 indicates to broadcast the message to all slaves and the slaves which have received the broadcast message do not make any response. If the communication address is not 0, slaves will reply to master as normal. For example, to communication with the slave with the communication address of 16, the address of the slave is set as 0x10 since decimal 16 is equal to hexadecimal 10.

- Function code and data

  The data format is determined by function codes.

  For example, to read the data of two continuous addresses with 0x0000 as start address in DVP15MC11T, the address of DVP15MC11T is 1, 0x0000 is the Modbus address of %MW0 in DVP15MC11T PLC.

  The data in the communication cable and the explanation on them are shown below:

  PC→DVP15MC11T: “01 03 00 00 00 02 C4 0B”

  DVP15MC11T→PC: “01 03 04 00 01 02 00 2A 32”

  ➢ Request message:

Field name	Character
Start	No input data for more than 10ms
Communication address	01
Function code	03
High byte of Modbus address	00
Low byte of Modbus address	00
Read high byte of data number	00
Read low byte of data number	02
Low byte of CRC check sum	C4
High byte of CRC check sum	0B
End	No input data for more than 10ms

➢ Response message:

Field name	Character
Start	No input data for more than 10ms
Communication address	01
Function code	03
Read data number（Counted by bytes）	04
Read high byte of data content	00
Read low byte of data content	01
Read high byte of data content	00
Read low byte of data content	02
Low byte of CRC check sum	2A
High byte of CRC check sum	32
End	No input data for more than 10ms

■ CRC check (check sum)

CRC check starts from “Communication address” to the last “Data content”. The calculation method is shown below.

Step 1: Download a 16-bit hex register (CRC register) with the content value FFFF.

Step 2: Make the XOR operation between the 8-bit data of the first byte in the command and the 8-bit data of the low byte in CRC register and then store the operation result in CRC register.

Step 3: Move the content value of CRC register by one bit towards the right and fill 0 in the highest bit.

Step 4: Check the value of the lowest bit in CRC register. If the value is 0, repeat the action of step 3; if 1, make XOR operation between the content in CRC register and hex. A001 and then store the result in CRC register.

Step 5: Repeat step 3 and step 4 till the content in CRC register is moved by 8 bits towards the right. At this moment, the processing of the first byte of the command message is finished.

Step 6: Repeat the action of step 2 to step 5 for the next byte in the command message till the processing of all bytes is finished. The last content in CRC register is CRC check value. When CRC check value in command message is transmitted, the high and low bytes in calculated CRC check value must exchange with each other, i.e. the low byte is transmitted first.

Example on calculation of CRC check value with C language

```
unsigned char* data     ← // Pointer of command message content
unsigned char length    ← // Length of command message content
unsigned int crc_chk ( unsigned char* data, unsigned char length )
{
int j;
unsigned int reg_crc=0Xffff;
while ( length-- )
{
reg_crc ^= *data++;
for （j=0;j<8;j++ )
{
If （( reg_crc & 0x01 )  reg_crc= ( reg_crc>>1 )  ^ 0Xa001; /* LSB ( b0 ) =1 */
else reg_crc=reg_crc >>1;
```

```
}
}
return reg_crc; // the value that sent back to the CRC register finally
}
```

A.3 Modbus Function Codes Supported

- The function codes which are supported by DVP15MC11T are listed in the following table when COM2 port is possessed by the motion control module.

Function code	Explanation	Available register
0x01	Read output bit register values; the data of 256 bits at most can be read at a time	%QX
0x02	Read bit register values; the data of 256 bits at most can be read at a time	%IX,%QX
0x03	Read one single or multiple word register value; the data of 100 words at most can be read at a time.	%MW,%QW,%IW
0x05	Write one single bit register value.	%QX
0x06	Write one single word register value.	%MW,%QW
0x0F	Write multiple bit register value; the data of 256 bits at most can be written at a time.	%QX
0x10	Write multiple word register value; the data of 100 words at most can be written at a time.	%MW,%QW

A.4 Modbus Exception Response Code Supported

- Exception response codes supported by DVP15MC11T are listed in the following table.

Exception response code	Explanation
0x01	Illegal command codes: the command codes in the command message which PLC receives are invalid.
0x02	Illegal register address: the address in the command message received is invalid.
0x03	Illegal register value: the data in the command message received by PLC are invalid.
0x07	◆ Check sum fault ✓ Check if the check sum is correct. ◆ Illegal command message ✓ Too short command message ✓ The length of the command message exceeds the valid range.

A.5 Introduction to Modbus Function Codes

- Function code 03 reads one single or multi word register values
  - Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	Read the start address of word registers in DVP15MC11T	High byte
Byte3		Low byte
Byte4	Read the number of addresses of word registers in DVP15MC11T (Counted by Word)	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

  - Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	Read the number of addresses of word registers in DVP15MC11T (Counted by Byte)	Single byte
Byte3	The address content of the word register in DVP15MC11T	High byte
Byte4		Low byte
…	The address content of the word register in DVP15MC11T	High byte
…		Low byte
Byte n	The address content of the word register in DVP15MC11T	High byte
Byte n+1		Low byte
Byte n+2	Low byte of CRC check sum	Low byte
Byte n+3	High byte of CRC check sum	High byte

  - Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ function code	Single byte
Byte2	Exception response code	Single byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

- **Example**

  To read the contents of address 0x0000 and 0x0001 in DVP15MC11T via function code 03.

  0x0000 and 0x0001 are the Modbus addresses of %MW0 and %MW1 in DVP15MC11T respectively.

  Suppose the value of %MW0 is 0x0001 and %MW1 is 0x0002:
  Request message: 01 03 00 00 00 02 C4 0B
  Response message: 01 03 04 00 01 00 02 2A 32

- Function code 06 writes one single word register value

  - Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	DVP15MC11T register address where to write the value	High byte
Byte3		Low byte
Byte4	The written value	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

  - Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	DVP15MC11T word register address where to write the value	High byte
Byte3		Low byte
Byte4	The written value	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

  - Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ function code	Single byte
Byte2	Exception response code	Single byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

  - **Example**

    Write 0x0100 to the address 0x0000 in DVP15MC11T via function code 06.
    Request message: 01 06 00 00 01 00 88 5A
    Response message: 01 06 00 00 01 00 88 5A

- Function code 0x10 writes multiple word register values

  - Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	The start address of DVP1515MCMC11T word registers where to write the value	High byte
Byte3		Low byte
Byte4	The number of addresses of DVP15MC11T word registers where to write the value. (Counted by word)	High byte
Byte5		Low byte
Byte6	The number of addresses of DVP15MC11T word registers where to write the value. (Counted by byte)	Single byte
Byte7	The address value written into DVP15MC11T word register	High byte
Byte8		Low byte
…	The address value written into DVP15MC11T word register	High byte
…		Low byte
Byte n	The address value written into DVP15MC11T word register	High byte
Byte n+1		Low byte
Byte n+2	Low byte of CRC check sum	Low byte
Byte n+3	High byte of CRC check sum	High byte

■ Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	The start address of DVP15MC11T word registers where to write the value	High byte
Byte3		Low byte
Byte4	The number of DVP15MC11T word registers where to write the value. (Counted by Word)	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

■ Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ function code	Single byte
Byte2	Exception response code	Single byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

- **Example**

  Write 0x0100 and 0x0200 to the addresses 0x0000 and 0x0001 in DVP15MC11T respectively via function code 0x10. 0x0000 and 0x0001 are Modbus addresses of %MW0 and %MW1 in DVP15MC11T respectively.

  Request message: 01 10 00 00 00 02 04 01 00 02 00 F3 33

  Response message: 01 10 00 00 00 02 41 C8

- Function code 0x01 reads multiple output bit register values
  - Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	The start address of DVP15MC11T bit registers to be read	High byte
Byte3		Low byte
Byte4	The number of DVP15MC11T bit registers to be read	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

  - Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	Read the number of bytes of bit registers.	Single byte
Byte3	Read the state value of the bit register.	Single byte
…	Read the state value of the bit register.	Single byte
Byte n	Read the state value of the bit register.	Single byte
Byte n+1	Low byte of CRC check sum	Low byte
Byte n+2	High byte of CRC check sum	High byte

  - Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ function code	Single byte
Byte2	Exception response message	Single byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

Note:

The value of Byte 2 in the response message is determined by the values of Byte 4 and Byte 5 in the request message. For example, the number of the read bit registers in the request

message is A. Dividing A by 8 produces B. If the quotient is an integer, the number of bytes of bit registers in the response message is B. Otherwise the number of bytes will be B + 1.
See the example below for details.

■ **Example**

Read the state value of %QX2.0~%QX3.4 in DVP15MC11T via function code 01. The address of %QX2.0 is 0xA010. Suppose the value of %QX2.0~%QX2.7 is 1000 0001 and %QX3.0~%QX3.4 is 1 0001.

Request message: 01 01 A0 10 00 0D DE 0A

Response message: 01 01 02 81 11 19 A0

● Function code 0x02 reads multiple bit register values

■ Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	The start address of DVP15MC11T bit registers where to read the state	High byte
Byte3		Low byte
Byte4	Read the number of bit registers.	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

■ Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	Read the number of bytes of bit registers.	Single byte
Byte3	Read the state value of the bit register.	Single byte
…	Read the state value of the bit register.	Single byte
Byte n	Read the state value of the bit register.	Single byte
Byte n+1	Low byte of CRC check sum	Low byte
Byte n+2	High byte of CRC check sum	High byte

■ Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ Function code	Single byte
Byte2	Exception response code	Single byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

Note:

The value of Byte 2 in the response message is determined by the values of Byte 4 and Byte 5 in the request message. For example, the number of the read bit registers in request message is A. Dividing A by 8 produces B. If the quotient is an integer, the number of bytes of bit registers in the response message is B. Otherwise the number of bytes will be B+ 1.
See the example below for details.

- **Example**

  Read the state value of %QX2.0~%QX3.4 in DVP15MC11T via function code 02. The address of %QX2.0 is 0xA010. Suppose %QX2.0~%QX2.7=1000 0001, %QX3.0~%QX3.4=1 0001.

  Request message: 01 02 A0 10 00 0D 9A 0A

  Response message: 01 02 02 81 11 19 E4

- Function code 0x05 writes one single bit register value
  - Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	Modbus address of the bit register	High byte
Byte3		Low byte
Byte4	The value written in the bit register	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

  - Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	Modbus address of the bit register	High byte
Byte3		Low byte
Byte4	The value written in the bit register	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

■ Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ Function code	Single byte
Byte2	Exception response code	Single byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

Note: The written value 0x0000 for the bit register in request message or response message indicates the value FALSE is written in the bit register; the written value 0xFF00 for the bit register indicates the value TRUE is written in the bit register.

A

■ **Example**

The value of %QX0.0 in DVP15MC11T is set to TRUE and the address of %QX0.0 is set to 0xA000 via function code 05.

Request message: 01 05 A0 00 FF 00 AE 3A

Response message: 01 05 A0 00 FF 00 AE 3A

- Function code 0x0F writes multiple bit register values

■ Data structure of a request message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	The start address of the bit registers where to write values	High byte
Byte3		Low byte
Byte4	The number of bit registers where to write values	High byte
Byte5		Low byte
Byte6	The number of bytes of bit registers where to write values	Single byte
Byte7	The value written to the bit register	Single byte
…	The value written to the bit register	Single byte
Byte n	The value written to the bit register	Single byte
Byte n+1	Low byte of CRC check sum	Low byte
Byte n+2	High byte of CRC check sum	High byte

- Data structure of a response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	Function code	Single byte
Byte2	The start address of bit registers where to write values	High byte
Byte3		Low byte
Byte4	The number of bit registers where to write values	High byte
Byte5		Low byte
Byte6	Low byte of CRC check sum	Low byte
Byte7	High byte of CRC check sum	High byte

A

- Data structure of an exception response message:

Byte NO.	Name	Byte
Byte0	Modbus ID	Single byte
Byte1	0x80+ Function code	Single byte
Byte2	Exception response code	High byte
Byte3	Low byte of CRC check sum	Low byte
Byte4	High byte of CRC check sum	High byte

Note: How many bytes of data in the request message depend on the number of bit registers in the request message.

- **Example**

The value of %QX0.0~%QX0.7 is set to 1000 0001 and the address of %QX0.0 is 0xA000 via function code 0F in DVP15MC11T.

Request message: 01 0F A0 00 00 08 01 81 26 55

Response message: 01 0F A0 00 00 08 76 0D

A.6 Table of Registers and Corresponding Modbus addresses

- Register numbers in the motion control module of DVP15MC11T and corresponding addresses are listed below:

Register name	Register number	Explanation	Address (hex)	Attribute
I	%IX0.0~%IX127.7	Bit registers	6000 ~ 63FF	Read only
Q	%QX0.0~%QX127.7		A000 ~ A3FF	Read/write
I	%IW0~%IW63	Word registers	8000 ~ 803F	Read only
Q	%QW0~%QW63		A000 ~ A03F	Read/write
M	%MW0~%MW32767		0000 ~ 7FFF	Read/write

MEMO

A

B

Appendix B Modbus TCP Communication

Table of Contents

B.1 Modbus TCP Message Structure B-2

B.2 Modbus Function Codes Supported in Modbus TCP B-2

B.3 Exception Response Code in Modbus TCP B-3

B.4 Modbus Function Codes in Modbus TCP B-3

B.5 Registers in DVP15MC11T and Corresponding Modbus Addresses B-12

B.1 Modbus TCP Message Structure

- **Modbus TCP message structure**

Byte NO.	Name		Explanation
Byte0	Transaction identifier	High byte	0
Byte1		Low byte	
Byte2	Protocol identifier	High byte	0
Byte3		Low byte	
Byte4	Modbus data length	High byte	The number of bytes of Modbus address and the data after it.
Byte5		Low byte	
Byte6	Modbus ID	Single byte	0 ~ 0xFF
Byte7	Function code	Single byte	
Byte8	Register address in DVP15MC11T	High byte	0~0xFFFF
Byte9		Low byte	
Byte10	Modbus data	High byte	The number of bytes of Modbus data is determined by function code.

B

B.2 Modbus Function Codes Supported in Modbus TCP

- **Modbus function codes which DVP15MC11T supports**

Function code	Function	Register
0x02	Read bit register value; maximum 256 bits of data could be read at a time.	%IX and %QX
0x03	Read one single or multiple word register values; maximum 100 words of data could be read at a time.	%IW, %QW and %MW
0x05	Write one single bit register value.	% QX
0x06	Write one single word register value.	%QW and %MW
0x0F	Write multiple bit register values; maximum 256 bits of data could be written at a time.	% QX
0x10	Write multiple word register values; maximum 100 words of data could be written at a time.	%QW and %MW

B.3 Exception Response Code in Modbus TCP

- **Modbus exception response codes that DVP15MC11T supports are shown in the table below.**

Exception response code	Indication
0x01	Unsupportive function code
0x02	Unsupportive Modbus address
0x03	Data length exceeds the range

B.4 Modbus Function Codes in Modbus TCP

- **Function code: 03 to read one single or multiple word register values**
  - **Request message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Low byte
Byte7	Function code	Single byte
Byte8	The start address of word registers to be read	High byte
Byte9		Low byte
Byte10	The number of word registers (Counted by Word)	High byte
Byte11		Low byte

  - **Response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte

Byte NO.	Name	Byte
Byte8	The number of read word registers. (Counted by Byte)	Single byte
Byte9	The content value in a word register	High byte
Byte10		Low byte
…	The content value in a word register	High byte
Byte n		Low byte

- **Exception response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	0x80+ function code	Single byte
Byte8	Exception response code	Single byte

- **Example**

To read the content value in the addresses 0x0000 and 0x0001 inside DVP15MC11T via function code 03. 0x0000 and 0x0001 are the Modbus address of %MW0 and %MW1 inside DVP15MC11T respectively. Suppose that the value of %MW0 is 0x0100 and the value of %MW1 is 0x0200.

Request message: 00 00 00 00 00 06 01 03 00 00 00 02

Response message: 00 00 00 00 00 07 01 03 04 01 00 02 00

- **Function code: 06 to write one single word register value**

- **Request message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte

B

Byte NO.	Name	Byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	The address of a word register where to write value	High byte
Byte9		Low byte
Byte10	The value written in the word register	High byte
Byte11		Low byte

- **Response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	The address of a word register where to write a value	High byte
Byte9		Low byte
Byte10	The value written in a word register	High byte
Byte11		Low byte

- **Exception response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	0x80+ function code	Single byte
Byte8	Exception response code	Single byte

- **Example:**

  To write the value 0x0100 to the address 0x0000 in DVP15MC11T via function code 06

  Request message: 00 00 00 00 00 06 01 06 00 00 01 00

  Response message: 00 00 00 00 00 06 01 06 00 00 01 00

- **Function code: 0x10 to write multiple word register values**

- **Request message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	The start address of word registers where to write values	High byte
Byte9		Low byte
Byte10	The number of word registers where to write values (Counted by Word)	High byte
Byte11		Low byte
Byte12	The number of word registers where to write values (Counted by Byte)	Single byte
Byte13	The value written in a word register	High byte
Byte14		Low byte
…	The value written in a word register	High byte
Byte n		Low byte

- **Response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte

Byte NO.	Name	Byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	The start address of word registers where to write values	High byte
Byte9		Low byte
Byte10	The number of word registers where to write values. (Counted by Word)	High byte
Byte11		Low byte

■ **Exception response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	0x80+ function code	Single byte
Byte8	Exception response code	Single byte

Note:

How many bytes of data in a response message depend on the number of read register addresses in DVP15MC11T in the request message. So the value of n in Byte n in the response message can be calculated through reading the number of register addresses in DVP15MC11T.

■ **Example**

To write 0x0100 and 0x0200 to the addresses 0x0000 and 0x0001 in DVP15MC11T via function code 06.

0x0000 and 0x0001 are the Modbus addresses of %MW0 and %MW1 in DVP15MC11T respectively.

Request message: 00 00 00 00 00 0B 01 10 00 00 00 02 04 01 00 02 00

Response message: 00 00 00 00 00 06 01 10 00 00 00 02

● **Function code: 0x02 to read multiple bit register values**

■ **Request message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte

Byte NO.	Name	Byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	The start address of the read bit registers	High byte
Byte9		Low byte
Byte10	The number of read bit registers	High byte
Byte11		Low byte

B

- **Response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	How many bytes for the read bit registers	Single byte
Byte9	The status value of a bit register which is read	Single byte
…	The status value of a bit register which is read	Single byte
Byte n	The status value of a bit register which is read	Single byte

- **Exception response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte

Byte NO.	Name	Byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	0x80+ function code	Single byte
Byte8	Exception response code	Single byte

- **Example**

  To read the state value of %QX2.0~%QX3.4 in DVP15MC11T via function code 02. 0xA010 is the address of %QX2.0. Suppose that %QX2.0~%QX2.7=1000 0001 and %QX3.0~%QX3.4=10001.

  Request message: 00 00 00 00 00 06 01 02 A0 10 00 0D

  Response message: 00 00 00 00 00 06 01 02 02 81 11

- **Function code: 0x05 to write one single bit register value**

  - **Request message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	Modbus address of a bit register	High byte
Byte9		Low byte
Byte10	The value written in the bit register	High byte
Byte11		Low byte

  - **Response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte

Byte NO.	Name	Byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	Modbus address of a bit register	High byte
Byte9		Low byte
Byte10	The value written in the bit register	High byte
Byte11		Low byte

■ **Exception response message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	0x80+ function code	Single byte
Byte8	Exception response code	Single byte

Note: The written value 0x0000 means that 0 is written to the bit register and 0xFF00 means that 1 is written to the bit register.

■ **Example**

Set the value of %QX0.0 in DVP15MC11T to 1 via function code 05; the address of %QX0.0 is 0xA000.

Request message: 00 00 00 00 00 06 01 05 A0 00 FF 00

Response message: 00 00 00 00 00 06 01 05 A0 00 FF 00

● **Function code: 0x0F to write multiple bit register values.**

■ **Request message data structure:**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte

Byte NO.	Name	Byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	Function code	Single byte
Byte8	The start address of the bit registers where to write values	High byte
Byte9		Low byte
Byte10	The number of bit registers where to write values	High byte
Byte11		Low byte
Byte12	How many bytes occupied by bit registers where to write values	Single byte
Byte13	The value written in a bit register	Single byte
Byte n	The value written in a bit register	Single byte

B

- **Response message data structure**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte
Byte3		Low byte
Byte4	Modbus data length	Single byte
Byte5	Modbus ID	High byte
Byte6		Low byte
Byte7	Function code	Single byte
Byte8	The start address of bit registers where to read status	High byte
Byte9		Low byte
Byte10	The number of bit registers where to write values	High byte
Byte 11		Low byte

- **Exception response message data structure**

Byte NO.	Name	Byte
Byte0	Transaction identifier	High byte
Byte1		Low byte
Byte2	Protocol identifier	High byte

Byte NO.	Name	Byte
Byte3		Low byte
Byte4	Modbus data length	High byte
Byte5		Low byte
Byte6	Modbus ID	Single byte
Byte7	0x80+ function code	Single byte
Byte8	Exception response code	Single byte

- **Example**

  Set %QX0.0~%QX0.7=1000 0001 via function code 0F and set the address of %QX0.0 to 0xA000 in DVP15MC11T.
  Request message: 00 00 00 00 00 0A 01 0F A0 00 00 08 01 81
  Response message: 00 00 00 00 00 06 01 0F A0 00 00 08

B.5 Registers in DVP15MC11T and Corresponding Modbus Addresses

B

Register name	Register no.	Explanation	Address (hex)	Attribute
I	%IX0.0~%IX127.7	Bit register	6000 ~ 63FF	Read only
Q	%QX0.0~%QX127.7		A000 ~ A3FF	Read/write
I	%IW0~%IW63	Word register	8000 ~ 803F	Read only
Q	%QW0~%QW63		A000 ~ A03F	Read/write
M	%MW0~%MW32767		0000 ~ 7FFF	Read/write

C

Appendix C CANopen Protocol

Table of Contents

C.1 Node States C-4

C.2 Network Management (NMT) C-7

C.3 PDO (Process Data Object) C-7

C.4 SDO (Service Data Object) C-9

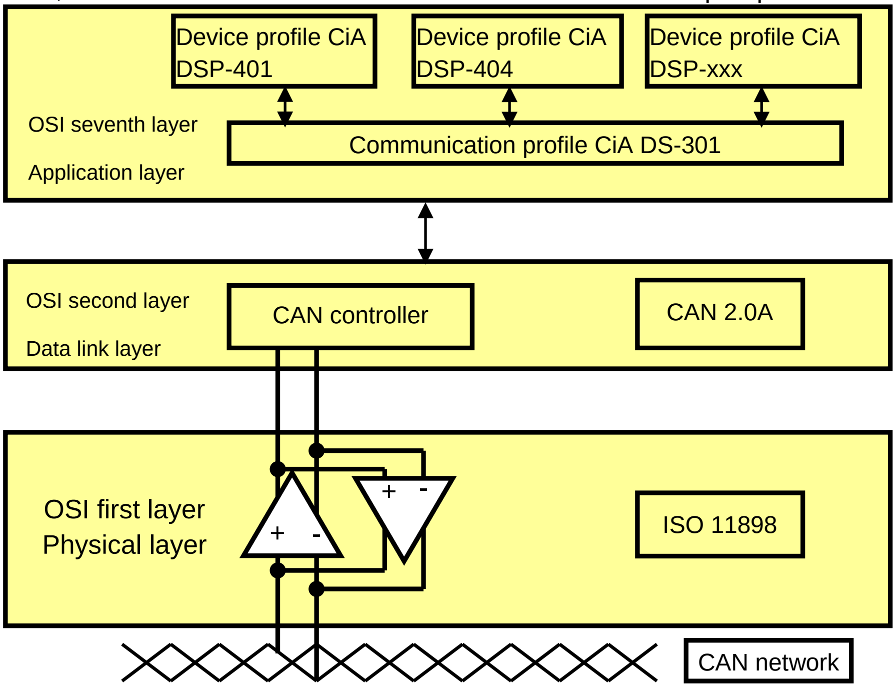
- **About CANopen protocol**

The CAN (controller area network) fieldbus only defines the physical layer and data link layer. (See ISO11898 standard.) It does not define the application layer. In the practical design, the physical layer and the data link layer are realized by the hardware. The CAN fieldbus itself is not complete. It needs the superior protocol to define the use of 11/29-bit identifier and 8-byte data.

The CANopen protocol is the CAN-based superior protocol. It is one of the protocols defined and maintained by CiA (CAN-in-Automation). It is developed on the basis of the CAL (CAN application layer) protocol, using a subset of the CAL communication and service protocols.

The CANopen protocol covers the application layer and the communication profile (CiA DS301). It also covers a framework for programmable registers (CiA 302), the recommendations for cables and connectors (CiA 303-1), and SI units and prefix representations (CiA 303-2).

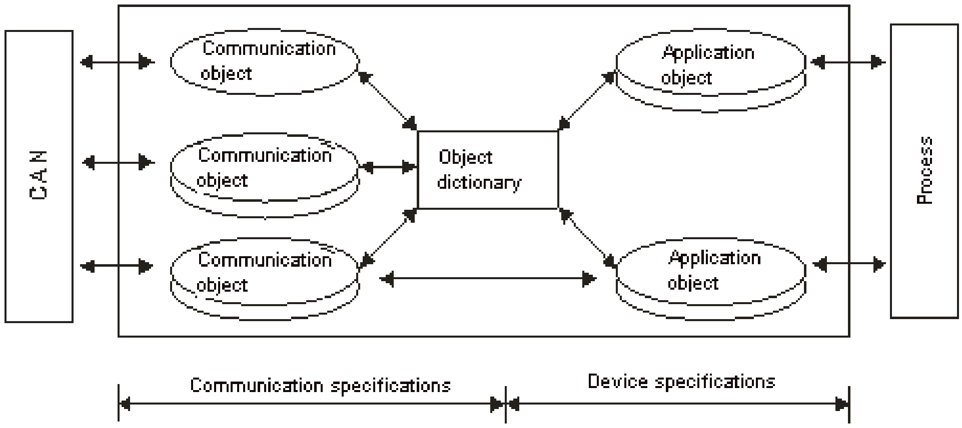
In the OSI model, the relation between the CAN standard and the CANopen protocol is as follows.



- **The object dictionary**

CANopen uses an object-based way to define a standard device. Every device is represented by a set of objects, and can be visited by the network. The model of the CANopen device is illustrated below. As the figure below shows, the object dictionary is the interface between the communication program and the superior application program.

The core concept of CANopen is the device object dictionary (OD). It is an orderly object set. Every object adopts a 16-bit index for addressing. In order to allow the visit to the single element in the data structure, it also defines an 8-bit subindex. Every node in the CANopen network has an object dictionary. The object dictionary includes the parameters which describe the device and the network behavior. The object dictionary of a node is described in the electronic data sheet.



- **The CANopen Communication Object**

  The CANopen communication protocol contains PDO, SDO, NMT and other predefined CANopen communication object.
  Refer to section C.3 for PDO introduction.
  Refer to section C.4 for SDO introduction.
  Refer to section C.2 for NMT introduction.

- **Other predefined CANopen communication objects (SYNC and EMCY)**

  - **SYNC Object (Synchronous object)**

    The synchronous object is the message broadcasted periodically by the master node in the CANopen network. This object is used to realize the network clock signal. Every device decides whether to use the event and undertake the synchronous communication with other network devices according to its configuration. For example, when controlling the driving device, the devices do not act immediately after they receive the command sent by the master. They do act until they receive the synchronous message. In this way, many devices can act synchronously.

    The format of the SYNC message:

COB-ID
80（hex）

  - **Emergency Object**

    The emergency object is used by the CANopen device to indicate an internal error. When an emergency error occurs in the device, the device sent the emergency message (including the emergency error code), and the device enters the error state. After the error is eliminated, the device sends the emergency message, the emergency error code is 0, and the device enters the normal state.
    The format of the emergency message:

<table>
<tr><th>COB-ID</th><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th><th>Byte 4</th><th>Byte 5</th><th>Byte 6</th><th>Byte 7</th></tr>
<tr><td rowspan="2">80（hex）
+Node-ID</td><td colspan="2">Emergency error code</td><td rowspan="2">Error register</td><td colspan="5" rowspan="2">Factory-defined error code</td></tr>
<tr><td>LSB</td><td>MSB</td></tr>
</table>

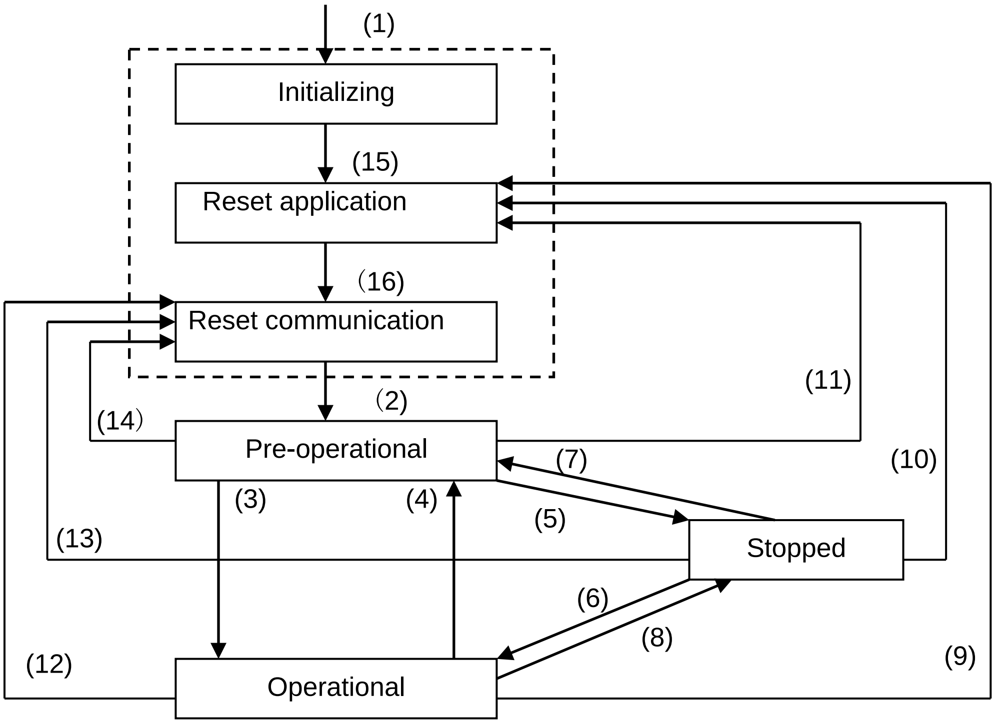
Note:
The value in the error register is mapped to index 1001 (hex) in the object dictionary. If the value is 0, no error occurs. If the value is 1, a general error occurs. If the value is H’80, an internal error occurs in the device.

C.1 Node States

- **Module control services**

The master node in the CANopen network controls the slave by sending the command. The slave executes the command after it receives the command and it does not need to reply. All CANopen nodes have internal NMT states. The slave node has four states, Initializing, Pre-operational, Operational, and Stop state.

The state of the device is illustrated below.



(1) After the power is supplied, the device automatically enters the initialization state.
(2) After the initialization is complete, the device automatically enters the Pre-operational state.
(3)(6) The remote node is started.
(4)(7) The device enters the Pre-operational state.
(5)(8) The remote node is stopped.
(9)(10)(11) The application layer is reset.
(12)(13)(14) The communication is reset.
(15) After the initializing is complete, the device automatically enters the “reset application” state.
(16) After the “reset application” state is complete, the device automatically enters the “reset communication” state.

The relation between the communication object and the state is shown below. The communication object service can be executed only in a proper state. For example, SDO can be executed only in the operational state and in the pre-operational state.

	Initialization	Pre-operational	Operational	Stopped
PDO			X	
SDO		X	X	
SYNC		X	X	
Time Stamp		X	X	
EMCY		X	X	
Boot-up	X			
NMT		X	X	X

The format of the control message for the node state:

COB-ID	Byte 0	Byte 1
0	Command specifier (CS)	Slave address (0: Broadcast)

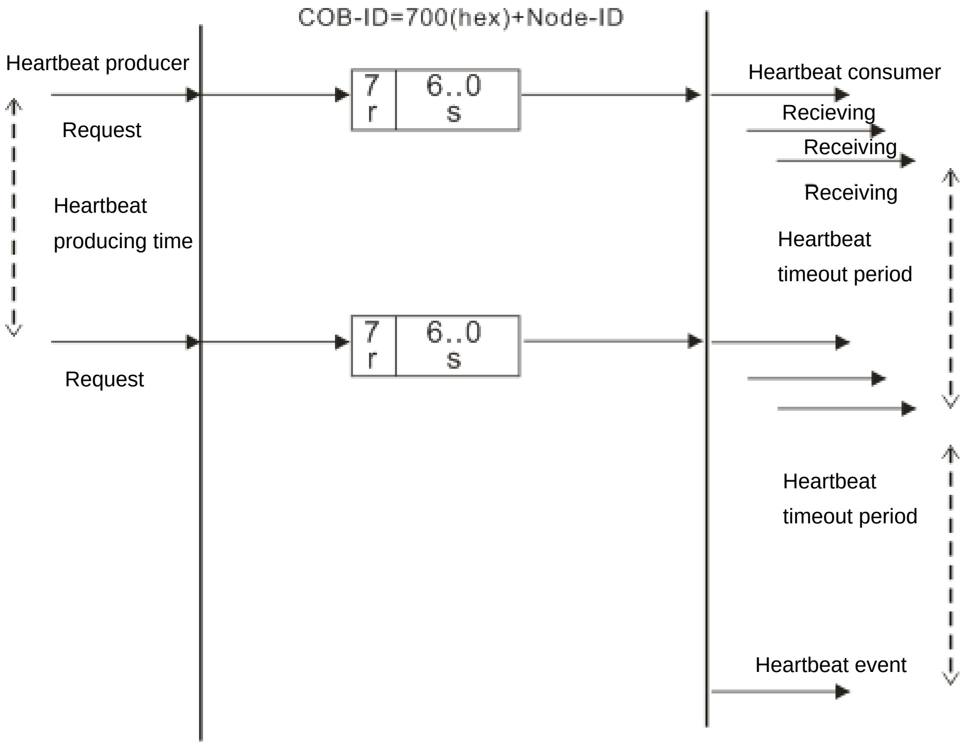
The command specifiers are listed below.

Command specifier (hex)	Function
01	Start the remote node
02	Stop the remote node
80	Enter the pre-operational state
81	Reset the application layer
82	Reset the communication

- **Error Control services**

The error control service is used to detect the disconnection of the node in the network. The error control services can be classified into two types, Heartbeat and Node Guarding. The PLC only supports Heartbeat. For example, the master can detect the disconnection of the slave only after the slave enables the Heartbeat service.

The Heartbeat principle is illustrated as follows. The Heartbeat producer transmits the Heartbeat message according to the Heartbeat producing time which is set. One or more Heartbeat consumers detect the message transmitted by the Heartbeat producer. If the consumer does not receive the message transmitted by the producer within the timeout period, the heartbeat event generated indicates that the CANopen communication is abnormal.



- **Boot-up services**

After the slave completes entering the pre-operational state, it will transmit a Boot-up message, which indicates the initializing is completed.

C

C.2 Network Management (NMT)

The CANopen network management complies with the Master/Slave mode. Only one NMT master can exist and other nodes are considered as slaves in a CANopen network. NMT contains three types of services, Module control services, Error Control services and Boot-up services. Please refer to section C.1 of the manual for more details.

C.3 PDO (Process Data Object)

- **PDO**

  - The PDO provides the direct visit channel for the device application object, is used to transmit the real-time data, and has high priority. Every byte in the PDO CAN message data list is used to transmit the data. The rate of making use of the message is high.
  - The PDO is described by means of the “producer/consumer mode”. The data is transmitted from one producer to one or many consumers. The data which can be transmitted are limited to 1-byte data to 8-byte data. After the data is transmitted by the producer, the consumer does not need to reply to the data. Every node in the network will detect the data information transmitted by the transmission node, and decides whether to process the data which is received.
  - There are two kinds of PDO services for every PDO: TxPDO and RxPDO. The PDO sent by the producer is called PDO (TxPDO) sent by the producer device. And the PDO the consumer receives is called PDO (RxPDO) which the consumer device receives.
  - Every PDO is described with two objects in the object dictionary: The PDO communication parameters and the PDO mapping parameters.

    The PDO communication parameters:

    Include the COB-ID which will be used by PDO, transmission type, prohibition time and the cycle of the counter.

    The PDO mapping parameters:

    Contain the object list in an object dictionary. These objects are mapped into the PDO, including the data length (in bits). To explain the contents of the PDO, the producer and the consumer have to understand the mapping.
  - The PDO transmission modes: synchronous and asynchronous

    Synchronous: Synchronous periodic and synchronous non-periodic

    Asynchronous: The PDO is transmitted when the data change, or it is transmitted after an event trigger.

    - The transmission modes supported by PDO are as follows.

Type			PDO transmission		
	Periodic	Non-periodic	Synchronous	Asynchronous	RTR
0		X	X		
1 – 240	X		X		
254				X	
255				X	

Mode 0: The PDO information is transmitted only when the PDO data change and the synchronous signal comes.

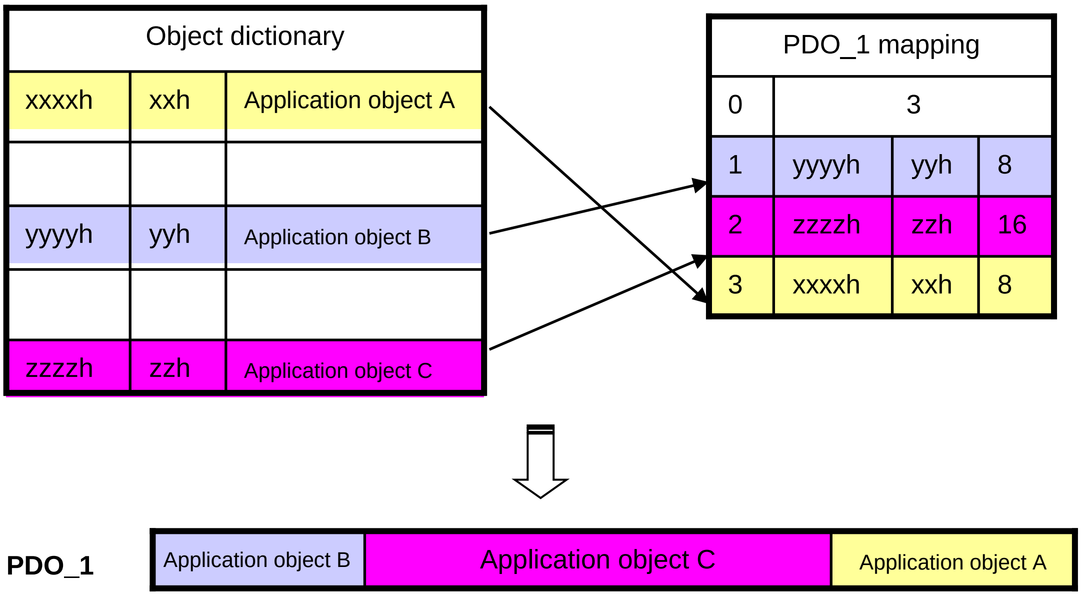
Modes 1~240: One piece of PDO information is transmitted every 1~240 synchronous signals.

Mode 254: The event trigger transmission is defined the manufacturer. For DVP15MC11T, the definition is the same as mode 255.

Mode 255: PDO is transmitted when the data change, or it is transmitted after an event trigger.

- All the data in the PDO has to be mapped from the object dictionary. The following is an example of the PDO mapping.

C



➢ The data format for RxPDO and TxPDO is as follows.

COB-ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Object identifier	Data							

C

C.4 SDO (Service Data Object)

- **SDO**

  - The SDO is used to build the client/server relation between two CANopen devices. The client device can read the data from the object dictionary of the server device, and write the data into the object dictionary of the server device. The access mode of the SDO is “client/server” mode. The mode which is accessed is the SDO server. Every CANopen device has at least one service data object which provides the access channel to the object dictionary of the device. SDO can read all objects in the object dictionary, and write all objects into the object dictionary.
  - The SDO message contains the index information and the subindex information which can be used to position the objects in the object dictionary, and the composite data structure can easily pass the SDO access. The trigger method of SDO belongs to the type of command response. In other words, the SDO server must reply after the SDO client sends a read/write request. The client and the server can stop the transmission of the SDO. The request message and response message can be differentiated according to their different COB-IDs.
  - The SDO can transmit the data in any length. If the data length is more than 4 bytes, the data has to be transmitted by segment. The last segment of the data contains an end flag.

  The structures of the SDO requested message and reply message are as follows.

  The formats of the request message and response message:

  - The format of the request message

COB-ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
600（hex）+Node-ID	Request code	Object index		Object index	Requested data			
		LSB	MSB		bit7-0	bit15-8	bit23-16	bit31-24

  - The definition of the request code in the request message:

Request code (hex)	Description
23	Writing the 4-byte data
2B	Writing the 2-byte data
2F	Writing the 1-byte data
40	Reading the data
80	Stopping current SDO function

  - The format of the response message

COB-ID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
580（hex）+Node-ID	Response code	Object index		Object subindex	Response data			
		LSB	MSB		bit7-0	bit15-8	bit23-16	bit31-24

  - The definition of the response code in the response message:

Response code (hex)	Description
43	Reading the 4-byte data
4B	Reading the 2-byte data
4F	Reading the 1-byte data
60	Writing the 1/2/4-byte data
80	Stopping the SDO function

Memo

Appendix D Explanation of Homing Modes

Table of Contents

D.1 Explanation of Homing Modes D-2

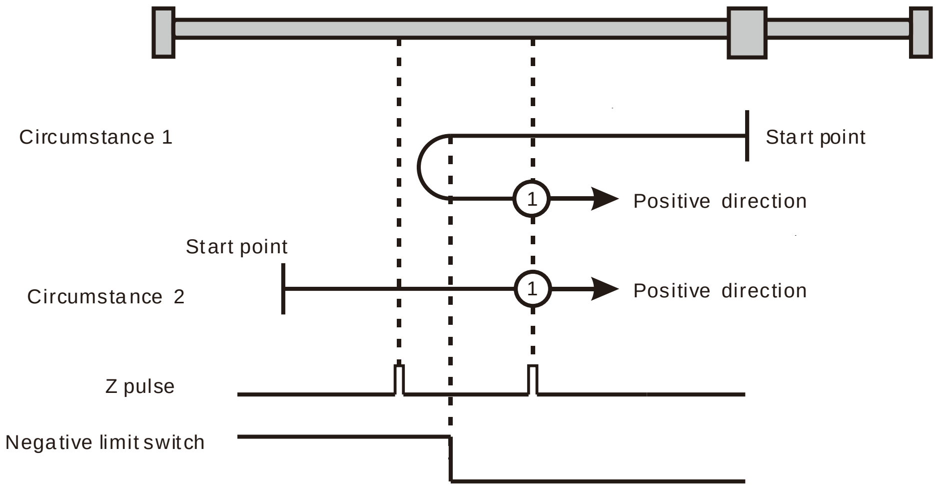
D.1 Explanation of Homing Modes

DVP15MC11T provides many homing modes from which user can choose the appropriate one in accordance with the field condition and technical requirement.

- Mode 1 Homing which depends on the negative limit switch and Z pulse.

Circumstance 1: MC_Home instruction is executed when the negative limit switch is OFF and the axis moves in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed when the axis encounters that the negative limit switch is ON. Where the first Z pulse is met is the home position when the negative limit switch is OFF.

Circumstance 2: MC_Home instruction is executed when the negative limit switch is ON and the axis moves in the positive direction at the second-phase speed. Where the first Z pulse is met is the home position when the negative limit switch is OFF.



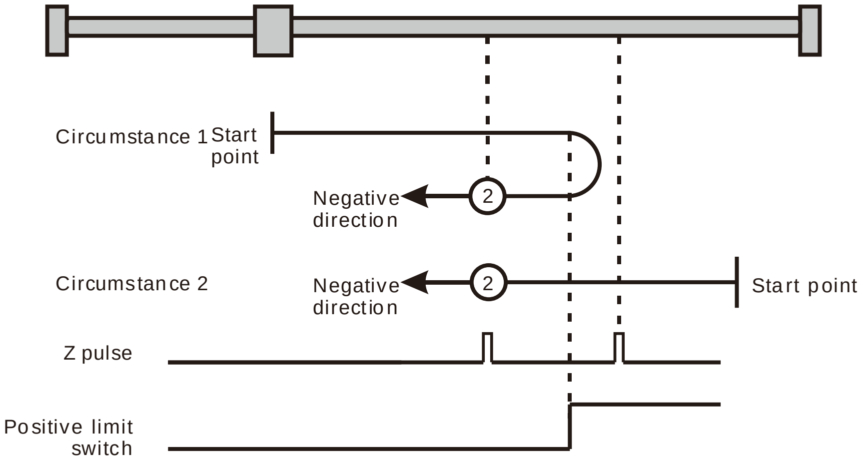
Homing depending on the negative limit switch and Z pulse (①: mode 1)

D

- Mode 2 Homing which depends on the positive limit switch and Z pulse

Circumstance 1: MC_Home instruction is executed when the positive limit switch is OFF and the axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed when the axis encounters that the positive limit switch is ON. Where the first Z pulse is met is the home position while the positive limit switch is OFF.

Circumstance 2: MC_Home instruction is executed when the positive limit switch is ON and the axis moves in the negative direction at the second-phase speed. Where the first Z pulse is met is the home position while the positive limit switch is OFF.



Homing depending on the positive limit switch and Z pulse (②: mode 2)

Mode 3 and mode 4 Homing which depends on the home switch and Z pulse

- Mode 3

Circumstance 1： When the home switch is OFF, MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed. When the axis encounters that the home switch is ON, the motion direction changes and the axis moves at the second-phase speed. Where the first Z pulse is met is the home position when the home switch is OFF.

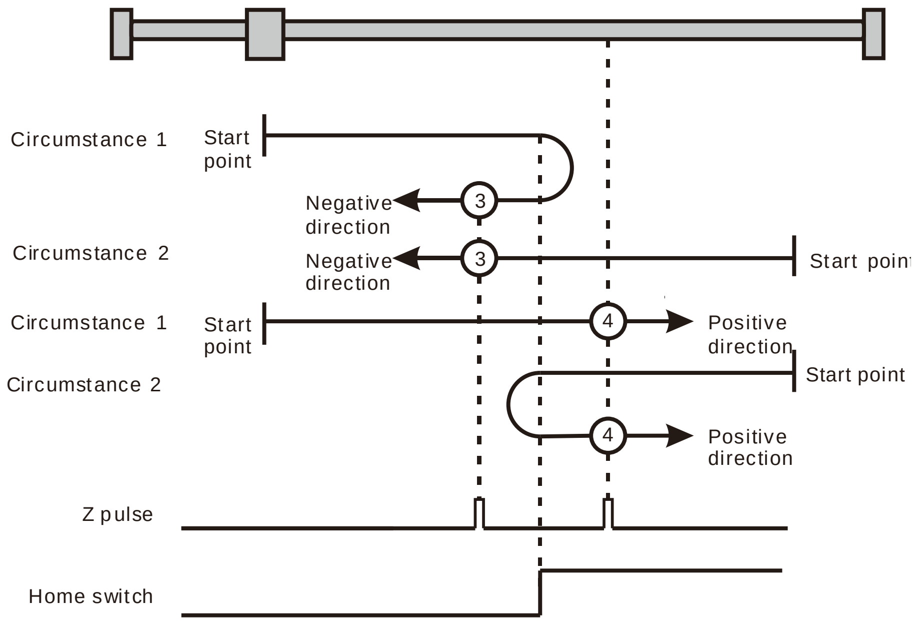
Circumstance 2： When the home switch is ON, MC_Home instruction is executed and the axis directly moves in the negative direction at the second-phase speed. Where the first Z pulse is met is the home position while the home switch is OFF.

- Mode 4

Circumstance 1： When the home switch is OFF, MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed. The axis moves at the second-phase speed when the axis encounters that the home switch is ON. Where the first Z pulse is met is the home position.

Circumstance 2： When the home switch is ON, MC_Home instruction is executed and the axis moves in the negative direction at the second-phase speed. When the axis encounters that the home switch is OFF, the motion direction changes and the axis moves at the second-phase speed. Where the first Z pulse is met is the home position.

D



Homing depending on the home switch and Z pulse (③: mode 3; ④: mode 4)

Mode 5 and mode 6 Homing which depends on the home switch and Z pulse

➢ Mode 5

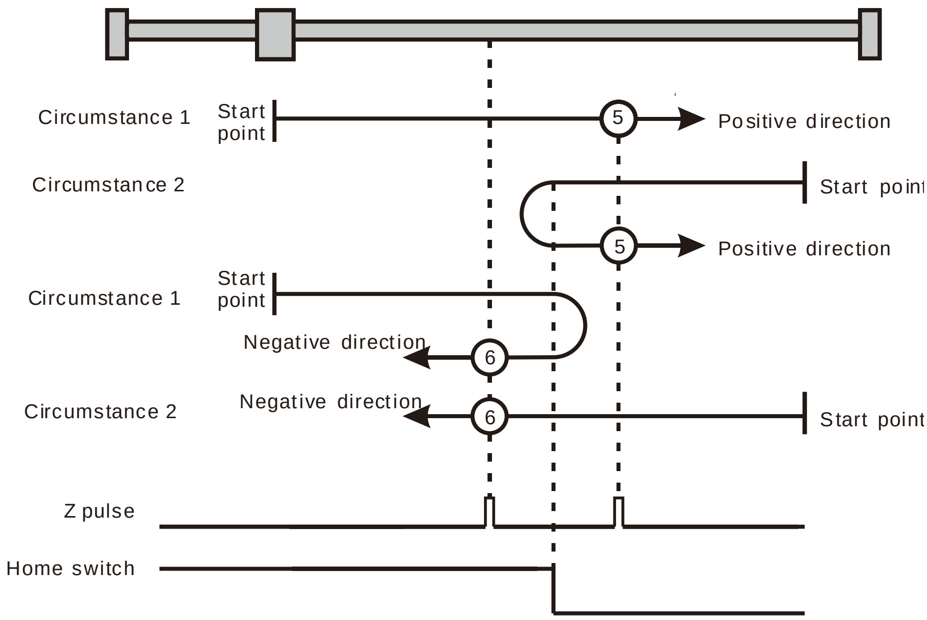
Circumstance 1： When the home switch is ON, MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed. Where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 2： When the home switch is OFF, MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed. When the home switch is ON, the motion direction changes and the axis moves at the second-phase speed. Where the first Z pulse is met is the home position when the home switch is OFF.

➢ Mode 6

Circumstance 1： When the home switch is ON, MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed. When the home switch is OFF, the motion direction changes and the axis moves at the second-phase speed. Where the first Z pulse is met is the home position.

Circumstance 2： When the home switch is OFF, MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed. While the home switch is ON, the axis moves at the second-phase speed and where the first Z pulse is met is the home position.



Homing depending on the home switch and Z pulse (⑤: mode 5, ⑥: mode 6)

Mode 7~ mode 10 Homing which depending on the home switch, positive limit switch and Z pulse

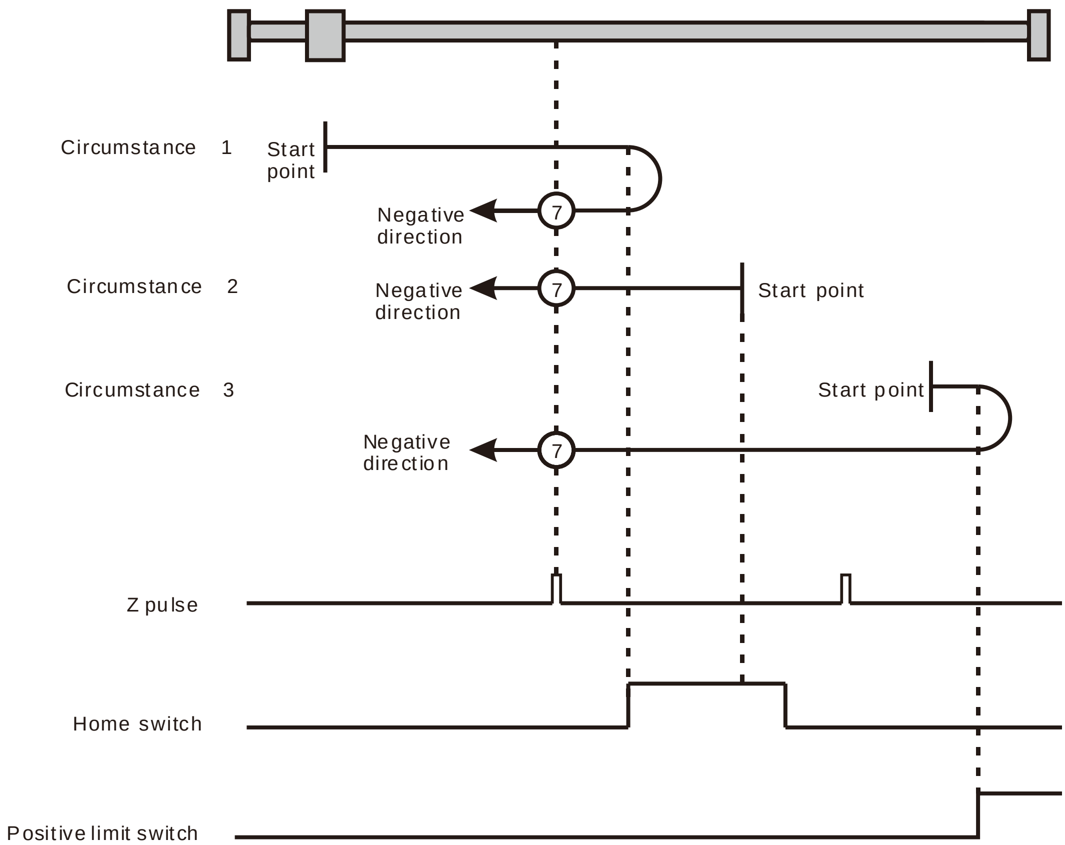
- Mode 7

Circumstance 1 : When the home switch is OFF, MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is ON. Where the first Z pulse is met is the home position when the home switch is OFF.

Circumstance 2 : When the home switch is ON, MC_Home instruction is executed and the axis moves in the negative direction at the second-phase speed. Where the first Z pulse is met is the home position when the home switch is OFF.

Circumstance 3 : When the home switch is OFF, MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. The axis starts to move at the second-phase speed when the home switch is ON. Where the first Z pulse is met is the home position when the home switch is OFF.

D



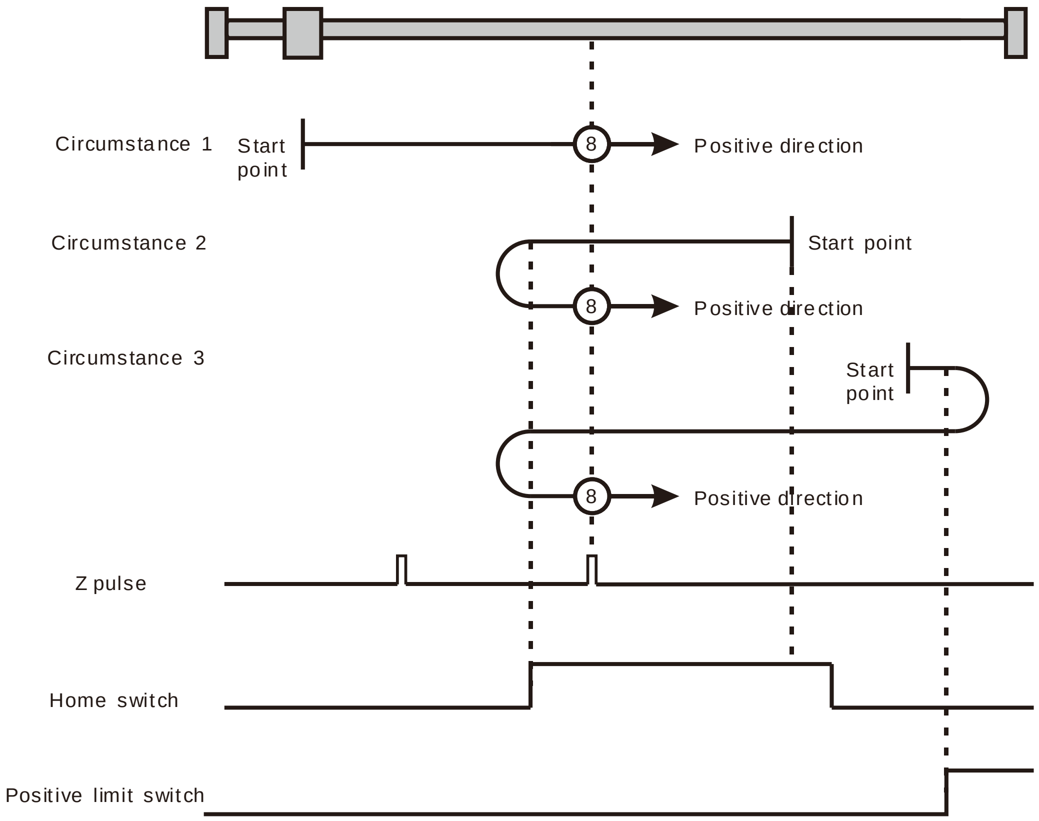
Homing depending on the home switch, positive limit switch and Z pulse (⑦: mode 7)

- Mode 8

Circumstance 1： When the home switch is OFF, MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed. The axis moves at the second-phase speed when the home switch is ON and where the first Z pulse is met is the home position.

Circumstance 2： MC_Home instruction is executed and the axis moves in the negative direction at the second-phase speed when the home switch is ON. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. And where the first Z pulse is met is the home position.

Circumstance 3： When the home switch is OFF, MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. The axis still moves at the first-phase speed when the home switch is ON. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF. The axis moves at the second-phase speed and where the first Z pulse is met is the home position when the home switch is ON.



Homing depending on the home switch, positive limit switch and Z pulse (⑧: mode 8)

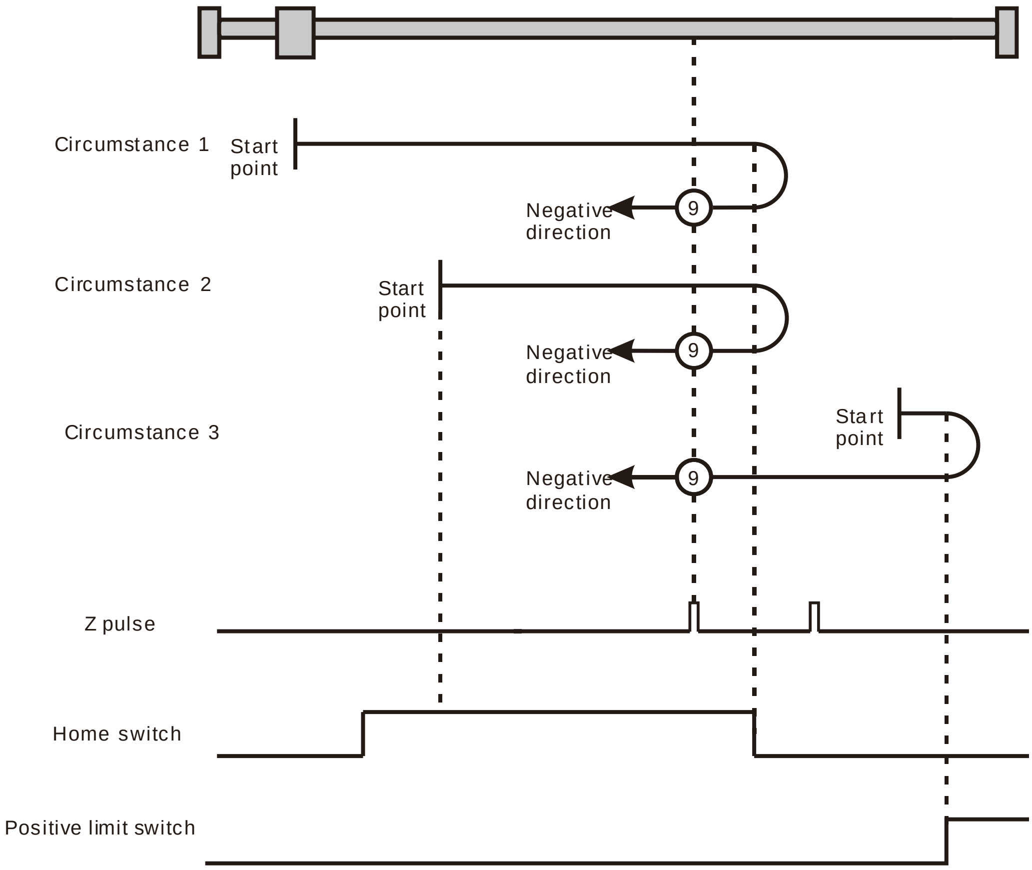
- Mode 9

Circumstance 1 : MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed when the home switch is OFF. The axis moves at the second-phase speed when the home switch is ON. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. And where the first Z pulse is met is the home position.

Circumstance 2 : When the home switch is ON MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. And where the first Z pulse is met is the home position.

Circumstance 3 : MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed when the home switch is OFF. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. The axis moves at the second-phase speed and where the first Z pulse is met is the home position when the home switch is ON.

D



Homing depending on the home switch, positive limit switch and Z pulse (⑨: mode 9)

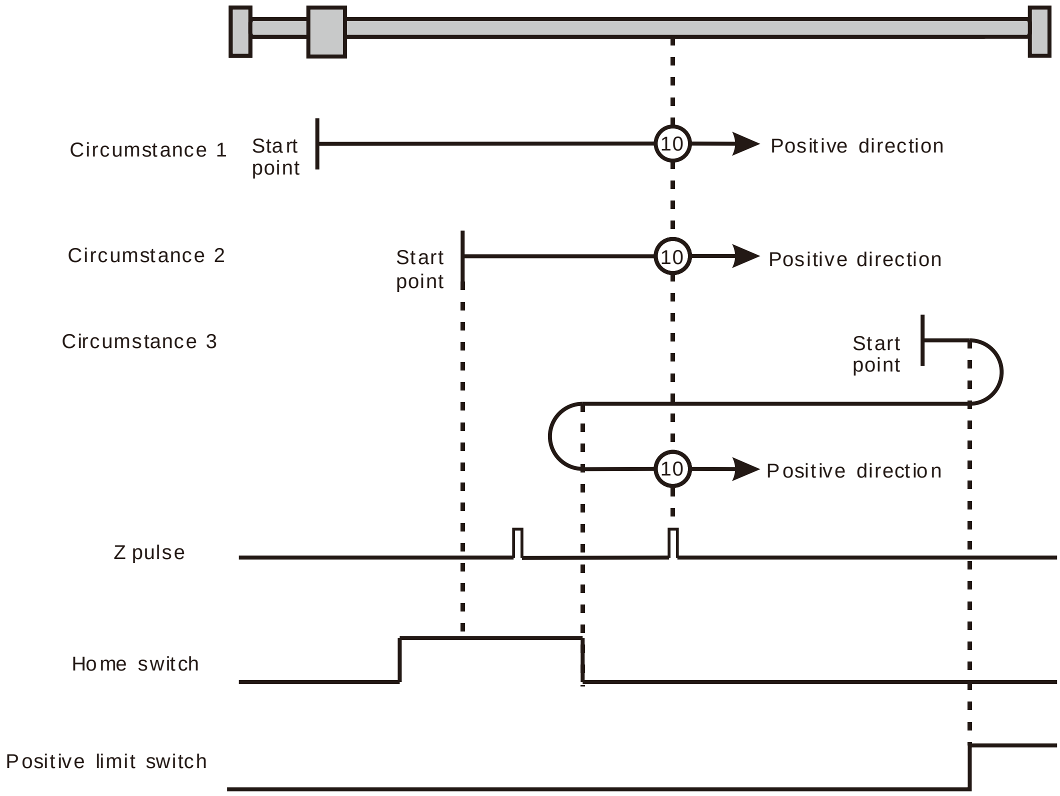
- Mode 10

Circumstance 1 : MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed when the home switch is OFF. The axis moves at the second-phase speed when the home switch is ON. And where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 2 : MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed when the home switch is ON. And where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 3 : MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed when the home switch is OFF. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. The motion direction changes again and the axis moves at the second-phase speed when the home switch is ON. Where the first Z pulse is met is the home position while the home switch is OFF.

D



Homing depending on the home switch, positive limit switch and Z pulse (⑩: mode 10)

Mode 11~ mode 14 Homing which depends on the home switch, negative limit switch and Z pulse

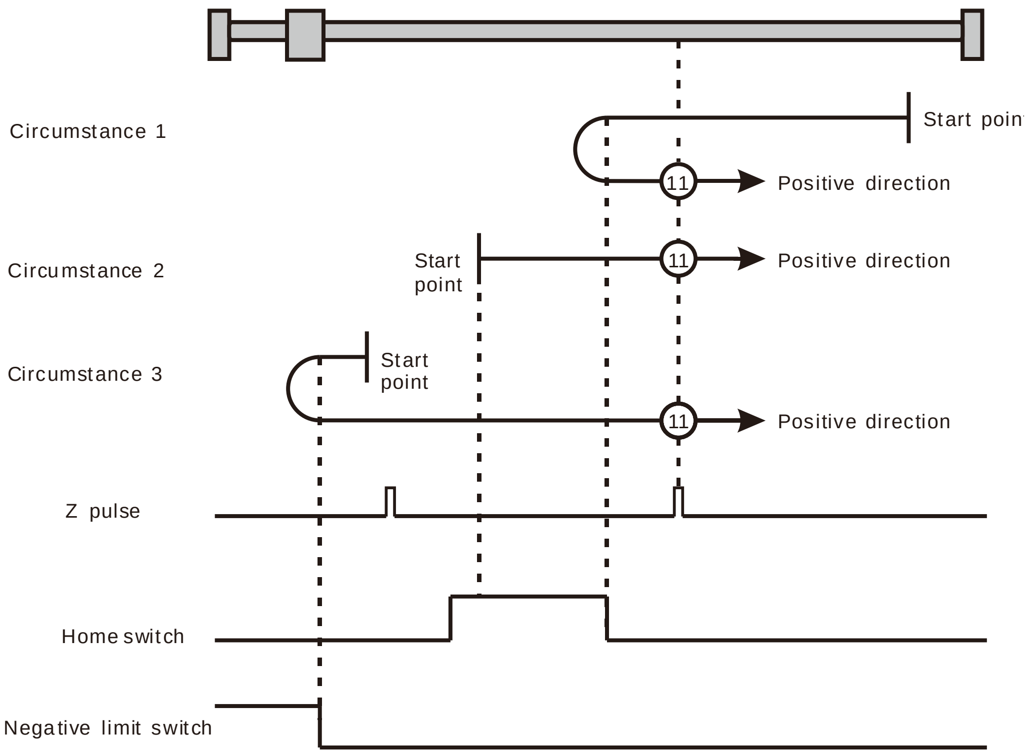
➢ Mode 11

Circumstance 1： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed when the home switch is OFF. The motion direction changes and the axis moves at the second-phase speed when the home switch is ON. And where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 2： MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed while the home switch is ON. And where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 3： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The motion direction changes and the axis moves at the first-phase speed while the home switch is OFF and the negative limit switch is ON. The axis moves at the second-phase speed when the home switch is ON. Where the first Z pulse is met is the home position while the home switch is OFF.

D



Homing depending on the home switch, negative limit switch and Z pulse (11: mode 11)

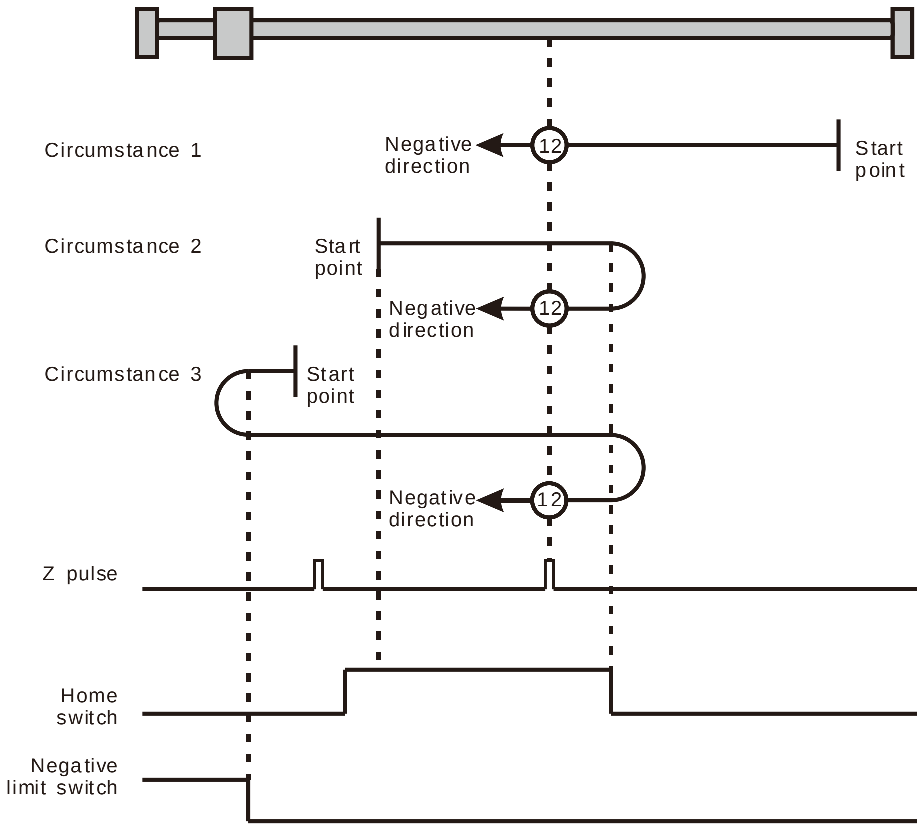
- Mode 12

Circumstance 1 : MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed when the home switch is OFF. The axis moves at the second-phase speed when the home switch is ON. And where the first Z pulse is met is the home position.

Circumstance 2 : MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed while the home switch is ON. The motion direction changes and the axis moves at the second-phase speed while the home switch is OFF. And where the first Z pulse is met is the home position.

Circumstance 3 : MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The motion direction changes and the axis moves at the first-phase speed while the home switch is OFF and the negative limit switch is ON. The axis still moves at the first-phase speed when the home switch is ON. The motion direction changes and the axis moves at the first-phase speed while the home switch is OFF. The axis moves at the second-phase speed while the home switch is ON. And where the first Z pulse is met is the home position.

D



Homing depending on the home switch, negative limit switch and Z pulse (12: mode 12)

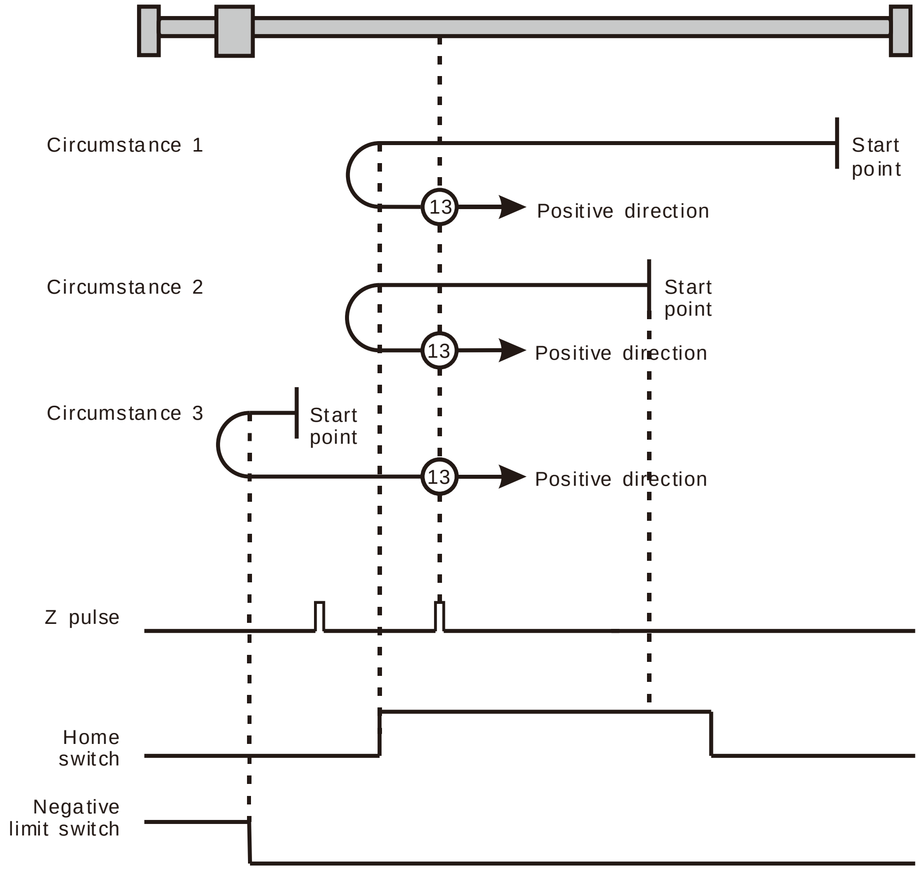
- Mode 13

Circumstance 1： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The axis moves at the second-phase speed while the home switch is ON. The motion direction changes and the axis moves at the second-phase speed while the home switch is OFF. And where the first Z pulse is met is the home position.

Circumstance 2： MC_Home instruction is executed and the axis moves in the negative direction at the second-phase speed while the home switch is ON. The motion direction changes and the axis moves at the second-phase speed while the home switch is OFF. And where the first Z pulse is met is the home position.

Circumstance 3： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The motion direction changes and the axis moves at the first-phase speed while the home switch is OFF and the negative limit switch is ON. The axis moves at the second-phase speed and where the first Z pulse is met is the home position when the home switch is ON.

D



Homing depending on the home switch, negative limit switch and Z pulse (13 : mode 13)

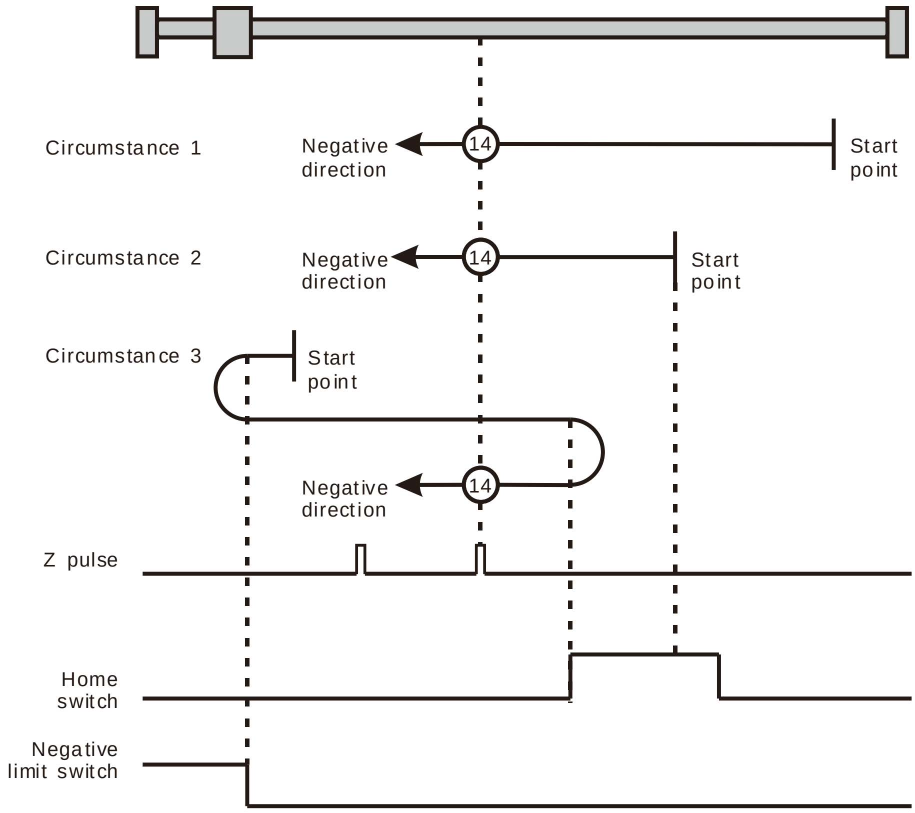
- Mode 14

Circumstance 1： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The axis moves at the second-phase speed once the home switch is ON. And where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 2： MC_Home instruction is executed and the axis moves in the negative direction at the second-phase speed while the home switch is ON. Where the first Z pulse is met is the home position while the home switch is OFF.

Circumstance 3： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The motion direction changes and the axis moves at the first-phase speed while the home switch is OFF and the negative limit switch is ON. The motion direction changes again and the axis moves at the second-phase speed when the home switch is ON. Where the first Z pulse is met is the home position while the home switch is OFF.

D



Homing depending on the home switch, negative limit switch and Z pulse (14 : mode 14)

Mode 15 and mode 16 are reserved for future development.

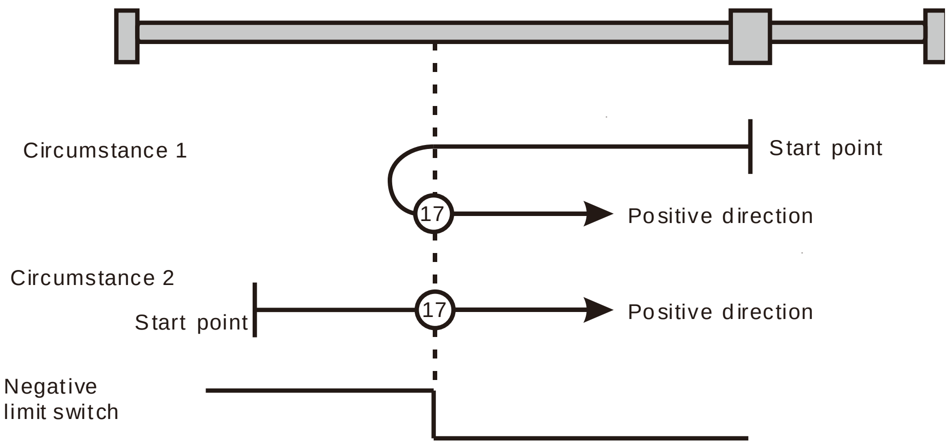
Mode 17~mode 30 Homing which has nothing to do with Z pulse

In mode 17~mode 30 which are respectively similar to mode1~mode 14 mentioned previously, the axis has nothing to do with Z pulse but the relevant home switch and limit switch status while returning to the home position. Mode 17 is similar to mode 1, mode 18 is similar to mode 2, mode 19 & mode 20 is similar to mode 3, mode 21 & mode 22 is similar to mode 5, mode 23 & mode 24 is similar to mode 7, mode 25 & mode 26 is similar to mode 9, mode 27 & mode 28 is similar to mode 11, and mode 29 & mode 30 are similar to 13.

- Mode 17 Homing which depends on the negative limit switch

Circumstance 1： MC_Home instruction is executed when the negative limit switch is OFF and the axis moves in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed when the axis encounters that the negative limit switch is ON. Where the servo is when the negative limit switch is OFF is the home position.

Circumstance 2： MC_Home instruction is executed when the negative limit switch is ON and the axis moves in the positive direction at the second-phase speed. Where the servo is is the home position when the negative limit switch is OFF.

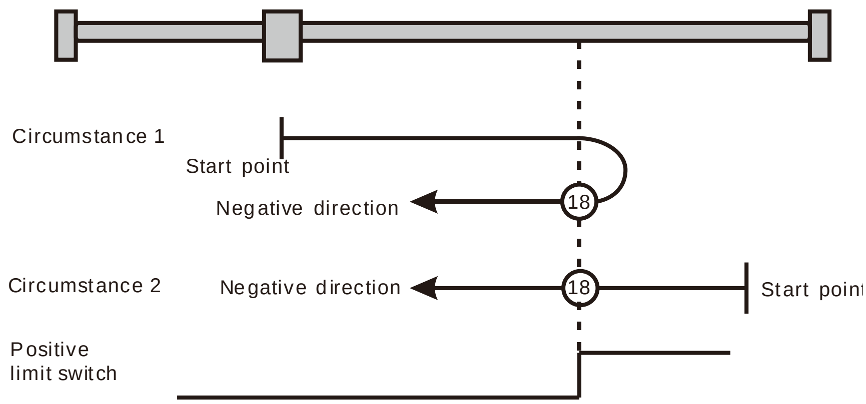


Homing depending on the negative limit switch (17: mode 17)

- Mode 18 Homing which depends on the positive limit switch

Circumstance 1： MC_Home instruction is executed when the positive limit switch is OFF and the axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed when the axis encounters that the positive limit switch is ON. Where the servo is is the home position while the positive limit switch is OFF.

Circumstance 2： MC_Home instruction is executed when the positive limit switch is ON and the axis moves in the negative direction at the second-phase speed. Where the servo is is the home position while the positive limit switch is OFF.



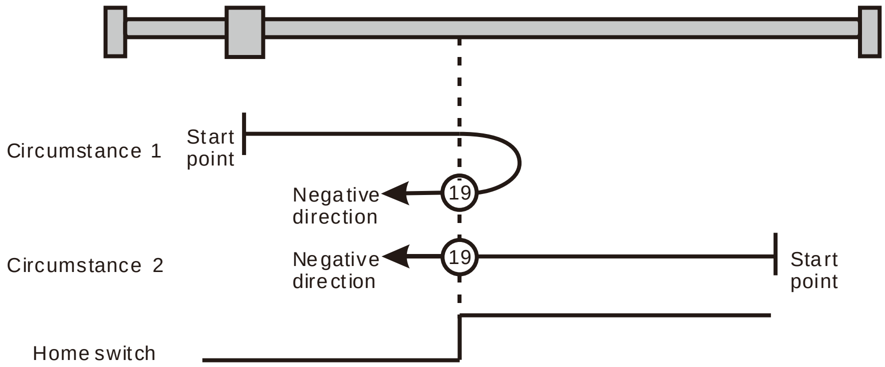
Homing depending on the positive limit switch (18: mode 18)

D

➢ Mode 19

Circumstance 1 : MC_Home instruction is executed and the axis moves in the positive direction at the first-phase speed while the home switch is OFF. The motion direction changes and the axis moves at the second-phase speed once the home switch becomes ON. And where the axis stands is the home position at the moment the home switch becomes OFF.

Circumstance 2 : MC_Home instruction is executed and the axis directly moves in the negative direction at the second-phase speed while the home switch is ON. And where the axis stands is the home position at the moment when the home switch becomes OFF.

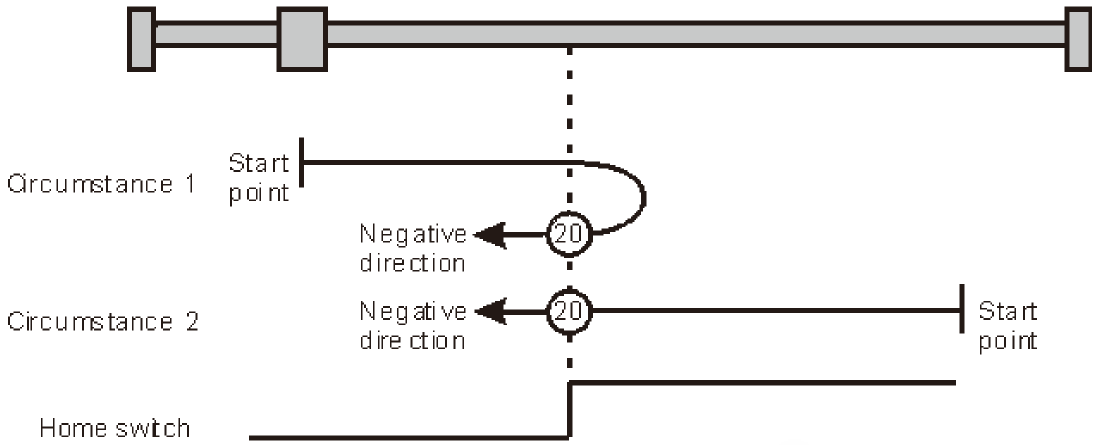


Homing depending on the home switch (19: mode 19)

➢ Mode 20

Circumstance 1 : MC_Home instruction is executed when the home switch is OFF and the axis moves in the positive direction at the first-phase speed. Where the servo is is the home position when the home switch is ON.

Circumstance 2 : MC_Home instruction is executed when the home switch is ON and the axis moves in the negative direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch becomes OFF. Where the servo is is the home position when the home switch is ON.



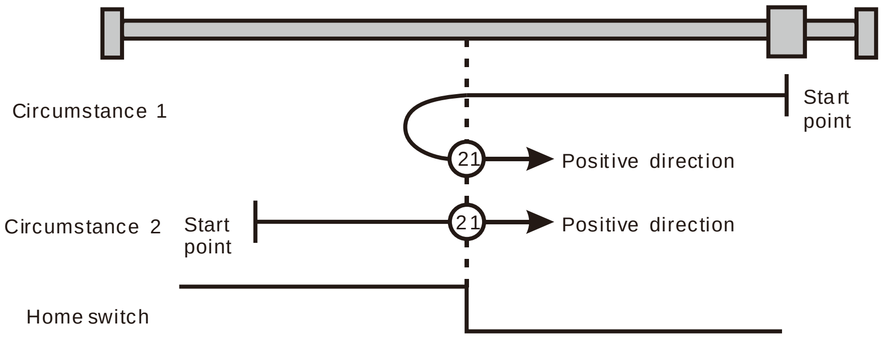
Homing depending on the home switch (20: mode 20)

D

- Mode 21

Circumstance 1： MC_Home instruction is executed and the axis moves in the negative direction at the first-phase speed while the home switch is OFF. The motion direction changes and the axis moves at the second-phase speed once the home switch becomes ON. And where the axis stands is the home position at the moment the home switch becomes OFF.

Circumstance 2： MC_Home instruction is executed and the axis moves in the positive direction at the second-phase speed while the home switch is ON. And where the axis stands is the home position at the moment the home switch becomes OFF.

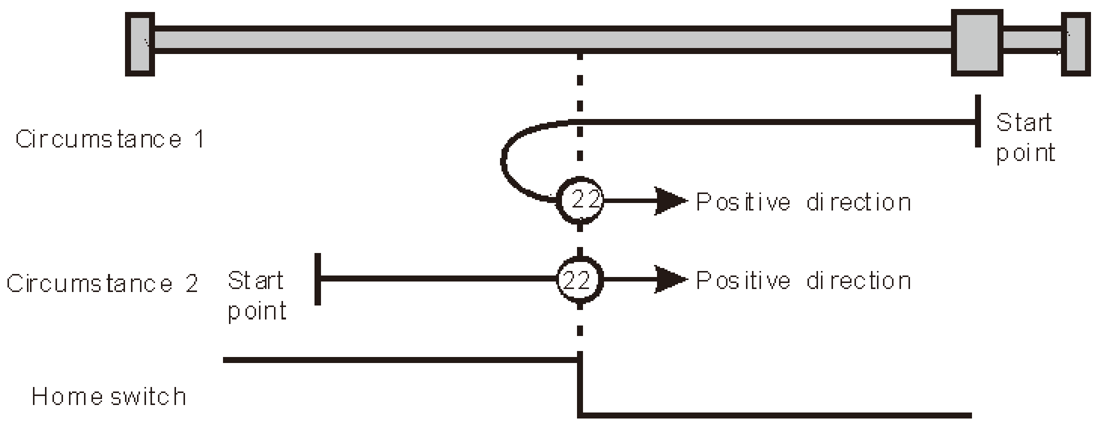


Homing depending on the home switch (21: mode 21)

- Mode 22

Circumstance 1： MC_Home instruction is executed while the home switch is ON and the axis moves in the positive direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed once the home switch becomes OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 2： MC_Home instruction is executed while the home switch is OFF and the axis moves in the negative direction at the first-phase speed. Where the axis stands is the home position when the home switch becomes ON.



Homing depending on the home switch (22: mode 22)

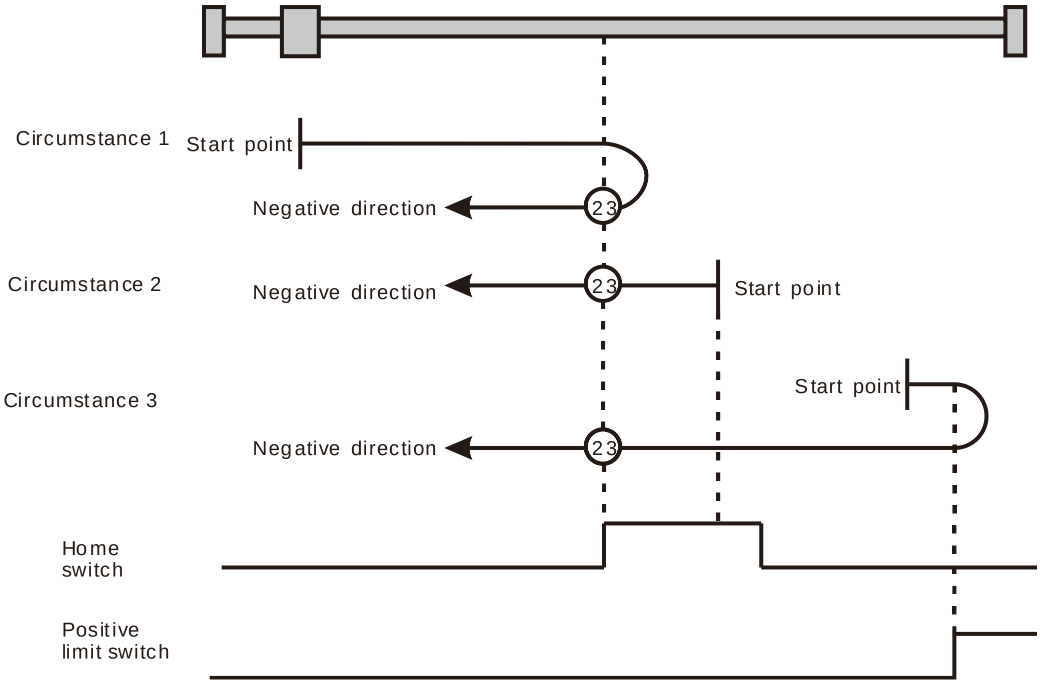
Homing depending on the home switch (22: mode 22)

➢ Mode 23

Circumstance 1 : MC_Home instruction is executed while the home switch is OFF and the axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed once the home switch becomes ON. Where the axis stands is the home position when the home switch is OFF.

Circumstance 2 : MC_Home instruction is executed while the home switch is ON and the axis moves in the negative direction at the second-phase speed. And where the axis stands is the home position when the home switch becomes OFF.

Circumstance 3 : MC_Home instruction is executed while the home switch is OFF. The axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. When the home switch is ON, the axis starts to move at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.



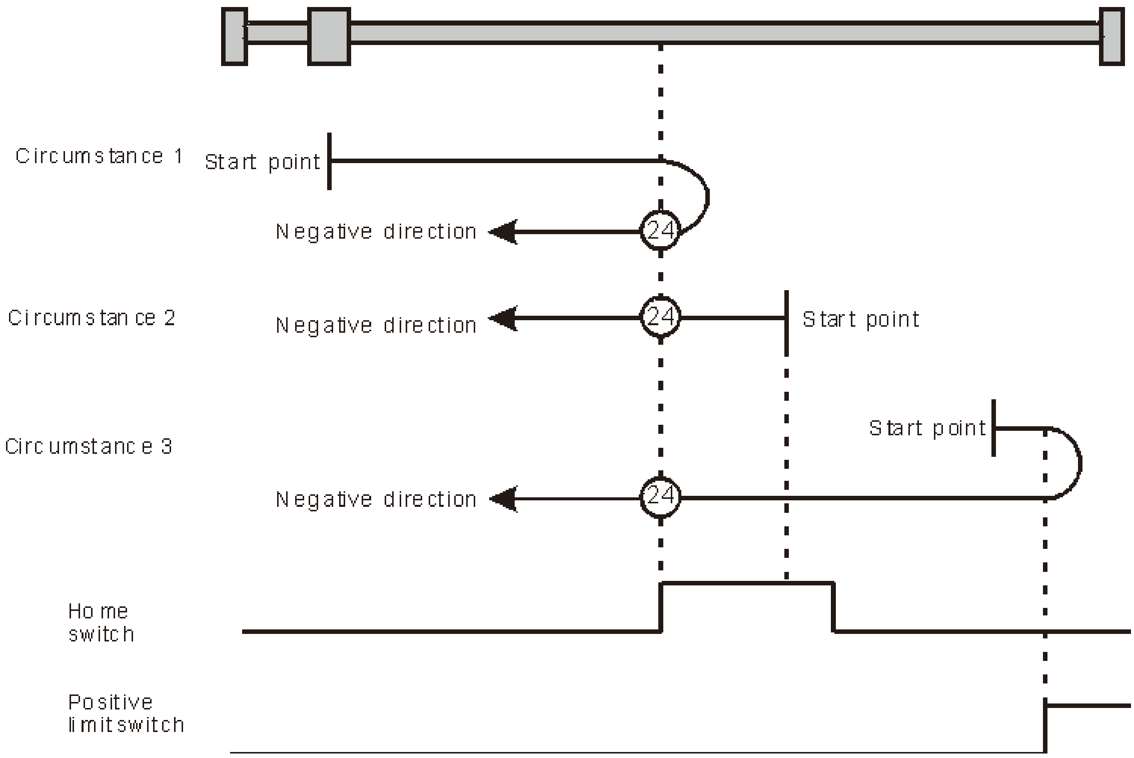
Homing depending on the home switch and positive limit switch (23 : mode 23)

➢ Mode 24

Circumstance 1 : MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the positive direction at the first-phase speed. Where the axis stands is the home position when the home switch is ON.

Circumstance 2 : MC_Home instruction is executed while the home switch is ON and the axis moves in the negative direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 3 : MC_Home instruction is executed while the home switch is OFF. The axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. When the home switch is ON, the axis still moves at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.



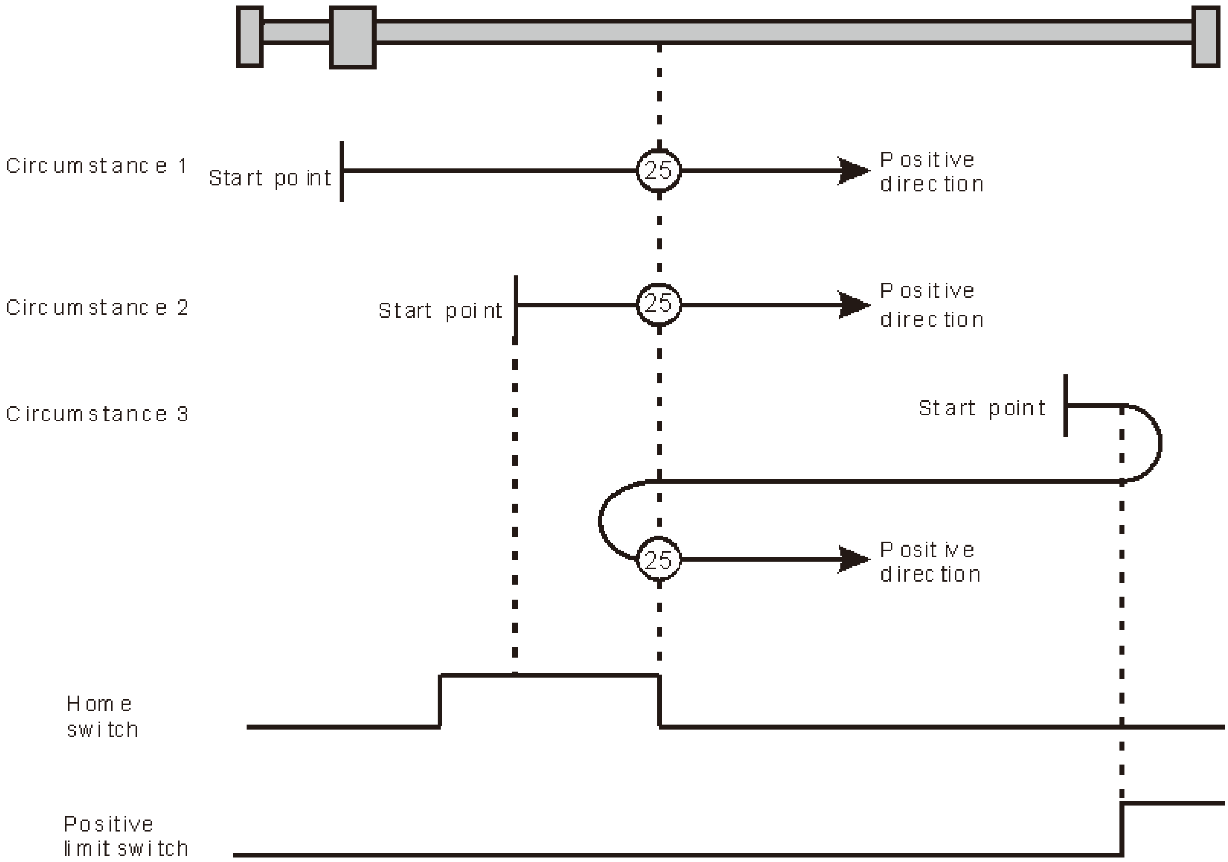
Homing depending on the home switch and positive limit switch (24: mode 24)

- Mode 25

Circumstance 1 : MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the positive direction at the first-phase speed. The axis moves at the second-phase speed when the home switch is ON. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 2 : MC_Home instruction is executed while the home switch is ON and the axis moves in the positive direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 3 : MC_Home instruction is executed while the home switch is OFF. The axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. Where the axis stands is the home position when the home switch is ON.



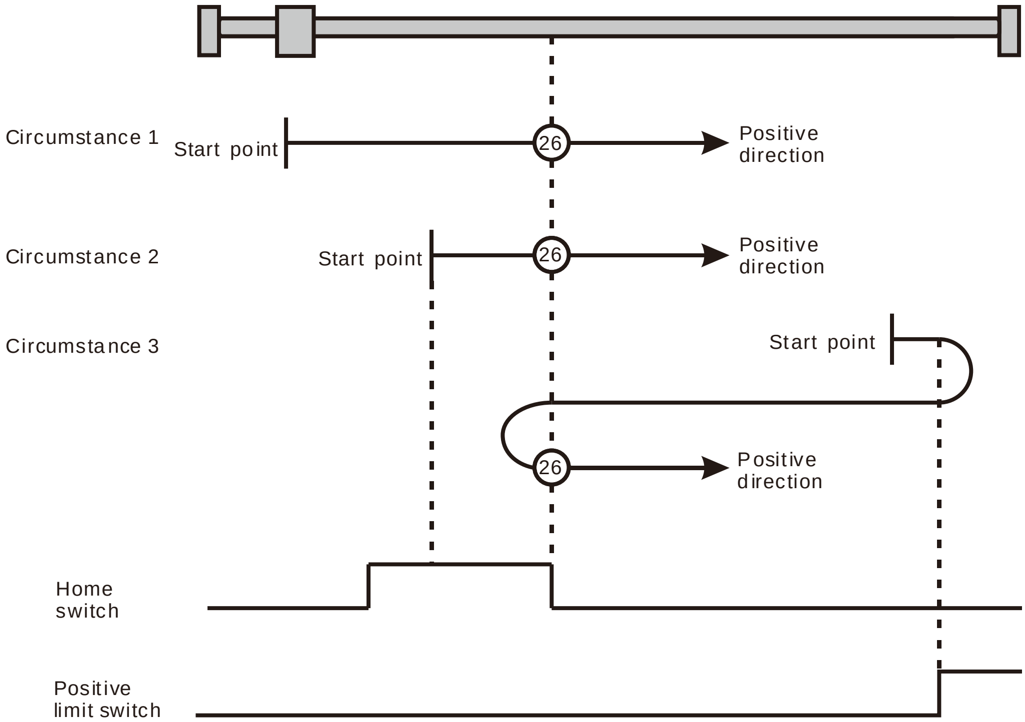
Homing depending on the home switch and positive limit switch (25: mode 25)

➢ Mode 26

Circumstance 1： MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the positive direction at the first-phase speed. The axis moves at the second-phase speed when the home switch is ON. Where the axis stands is the home position when the home switch is OFF.

Circumstance 2： MC_Home instruction is executed while the home switch is ON and the axis moves in the positive direction at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.

Circumstance 3： MC_Home instruction is executed while the home switch is OFF. The axis moves in the positive direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the positive limit switch is ON. The motion direction changes again and the axis moves at the second-phase speed when the home switch is ON. Where the axis stands is the home position when the home switch is OFF.



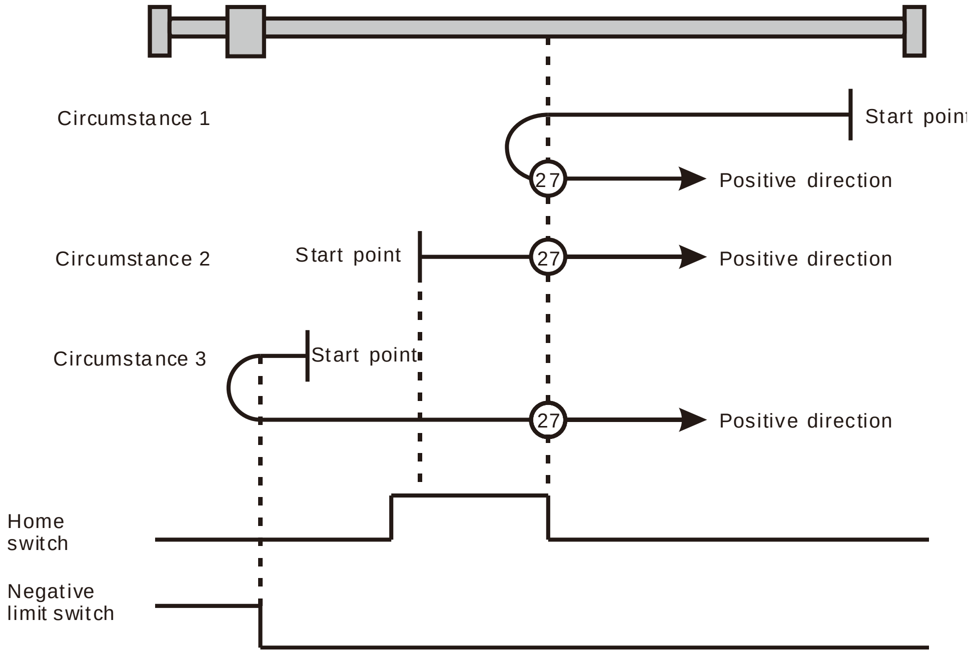
Homing depending on the home switch and positive limit switch (26: mode 26)

- Mode 27

Circumstance 1： MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is ON. Where the axis stands is the home position when the home switch is OFF.

Circumstance 2： MC_Home instruction is executed while the home switch is ON and the axis moves in the positive direction at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.

Circumstance 3： MC_Home instruction is executed while the home switch is OFF. The axis moves in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the negative limit switch is ON. When the home switch is ON, the axis starts to move at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.



Homing depending on the home switch and negative limit switch (27: mode 27)

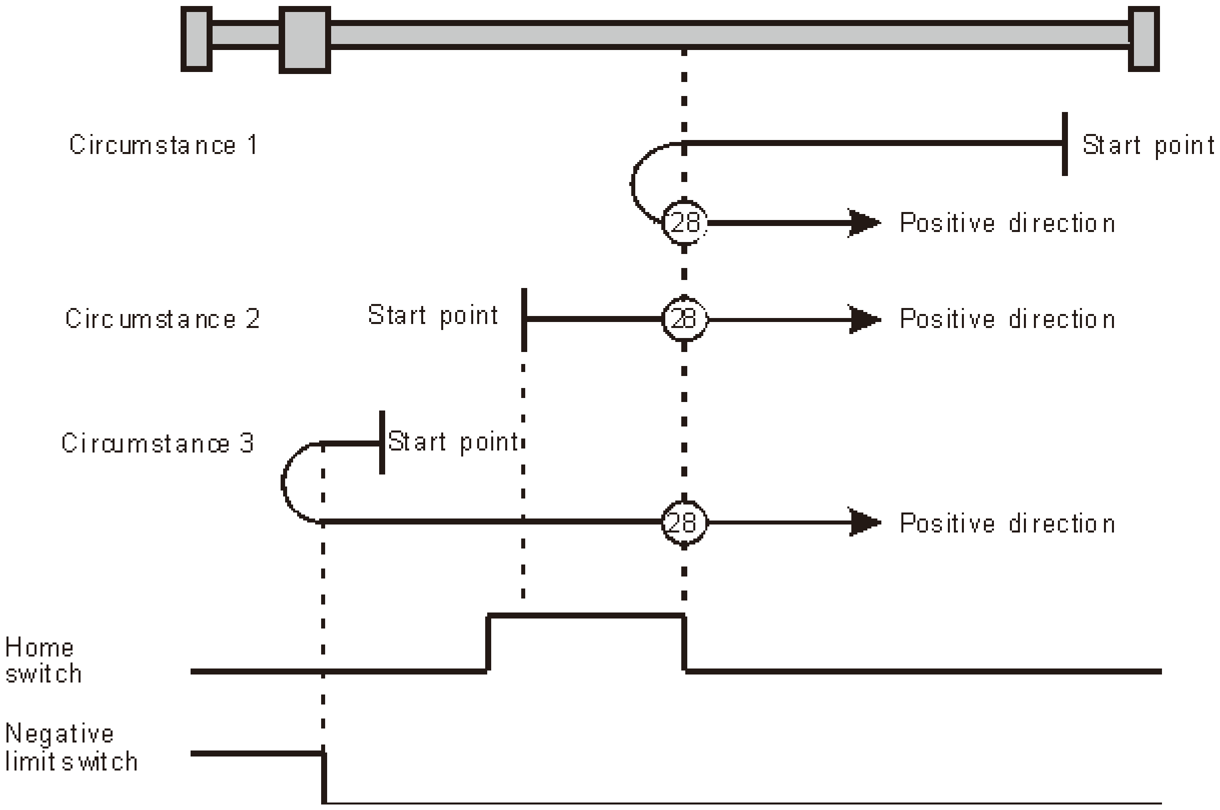
- Mode 28

Circumstance 1 : MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the negative direction at the first-phase speed. Where the axis stands is the home position when the home switch is ON.

Circumstance 2 : MC_Home instruction is executed while the home switch is ON and the axis moves in the positive direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 3 : MC_Home instruction is executed while the home switch is OFF. The axis moves in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the negative limit switch is ON. When the home switch is ON, the axis still moves at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

D



Homing depending on the home switch and negative limit switch (28: mode 28)

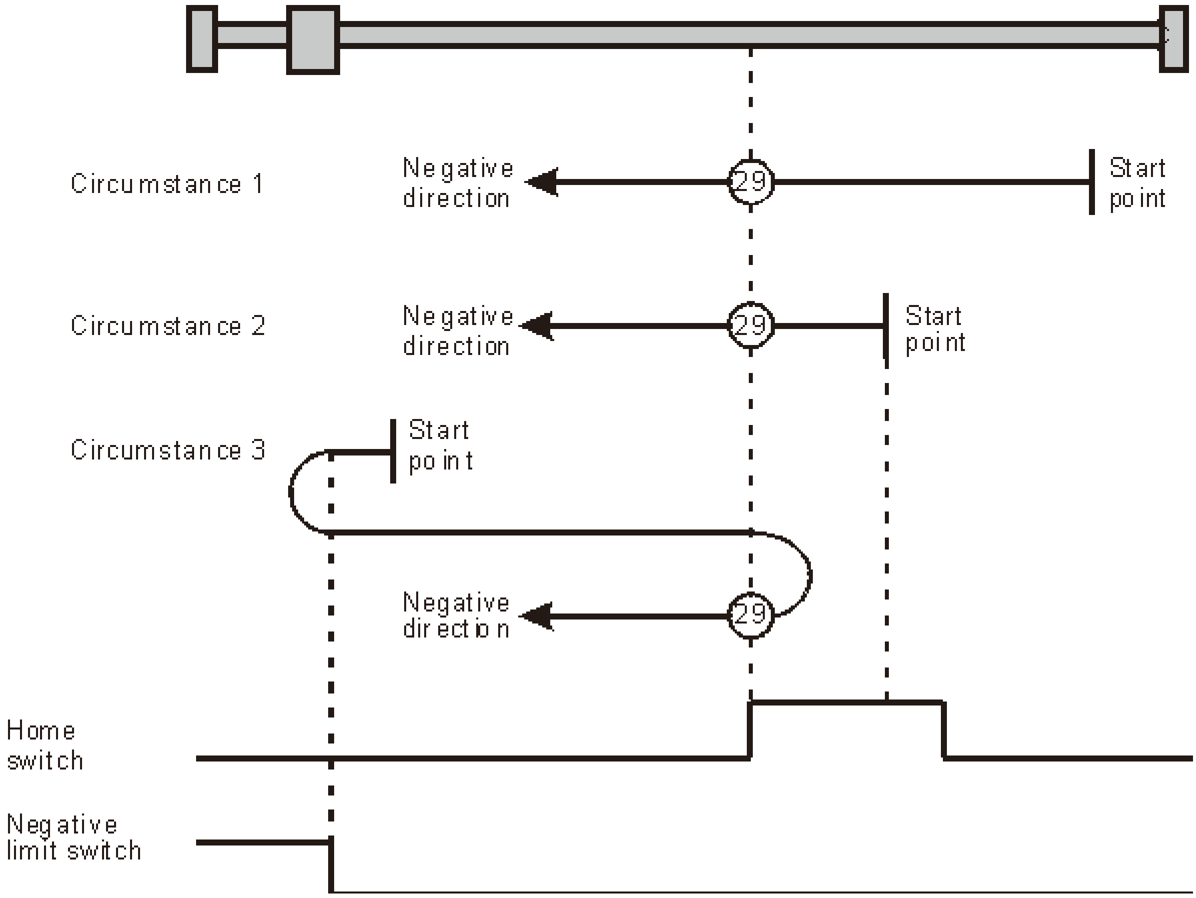
- Mode 29

Circumstance 1： MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the negative direction at the first-phase speed. When the home switch is ON, the axis starts to move at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 2： MC_Home instruction is executed while the home switch is ON and the axis moves in the negative direction at the second-phase speed. The motion direction changes and the axis moves at the second-phase speed when the home switch is OFF. Where the axis stands is the home position when the home switch is ON.

Circumstance 3： MC_Home instruction is executed while the home switch is OFF. The axis moves in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the negative limit switch is ON. Where the axis stands is the home position when the home switch is ON.

D



Homing depending on the home switch and negative limit switch (29 : mode 29)

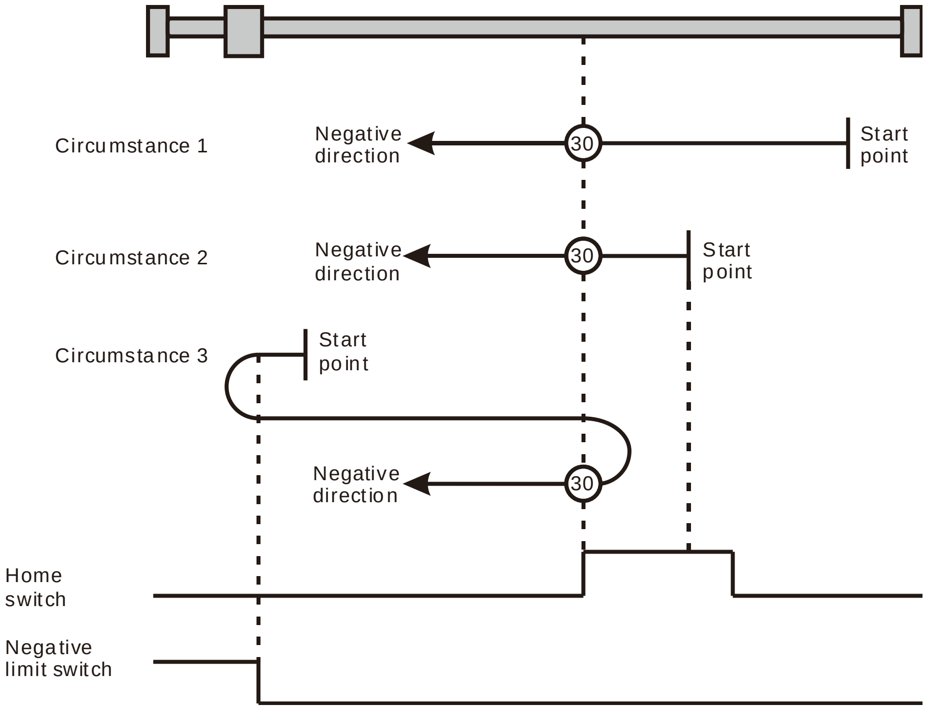
➢ Mode 30

Circumstance 1 : MC_Home instruction is executed while the home switch is OFF and the axis starts to move in the negative direction at the first-phase speed. When the home switch is ON, the axis starts to move at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.

Circumstance 2 : MC_Home instruction is executed while the home switch is ON and the axis moves in the negative direction at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.

Circumstance 3 : MC_Home instruction is executed while the home switch is OFF. The axis moves in the negative direction at the first-phase speed. The motion direction changes and the axis moves at the first-phase speed when the home switch is OFF and the negative limit switch is ON. When the home switch is ON, the motion direction changes again and the axis moves at the second-phase speed. Where the axis stands is the home position when the home switch is OFF.

D



Homing depending on the home switch and negative limit switch (30: mode 30)

Mode 31 and mode 32 Reserved for future development.

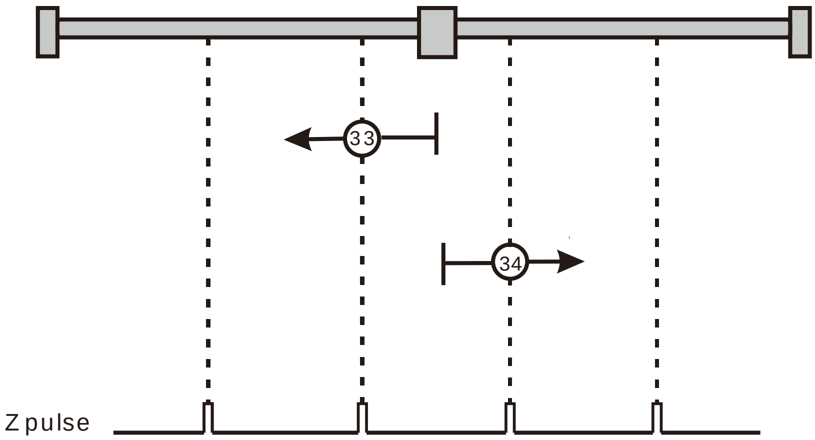
Mode 33 ~ mode 34 Homing which depends on Z pulse

- Mode 33

In mode 33, MC_Home instruction is executed and the axis moves at the second-phase speed in the negative direction. And the place where the axis stands is the home position once the first Z pulse is met.

- Mode 34

In mode 34, MC_Home instruction is executed and the axis moves at the second-phase speed in the positive direction. And the place where the axis stands is the home position once the first Z pulse is met.



Homing depending on Z pulse (33: mode 33, 34: mode 34)

- Mode 35 Homing which depends on the current position

In mode 35, MC_Home instruction is executed, the axis does not move and its current position is regarded as the home position.

D

Appendix E List of Accessories

Table of Contents

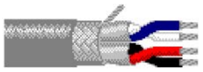
E.1 Accessories for CANopen Communication E-2

E.2 Accessories for PROFIBUS DP Communication E-4

E.3 Accessories for DeviceNet Communication E-4

E.1 Accessories for CANopen Communication

- **Cables**

Figure	Model	Length	Diameter（AWG）
	UC-DN01Z-01A	305M	2#15，2#18 SHLD PVC（Thick cable）
	UC-DN01Z-02A	305M	2#22，2#24 SHLD PVC（Thin cable）
	UC-CMC003-01A	0.3M	4#26，1#24　PVC（Thin cable）
	UC-CMC005-01A	0.5M	4#26，1#24　PVC（Thin cable）
	UC-CMC010-01A	1.0M	4#26，1#24　PVC（Thin cable）
	UC-CMC015-01A	1.5M	4#26，1#24　PVC（Thin cable）
	UC-CMC020-01A	2.0M	4#26，1#24　PVC（Thin cable）
	UC-CMC030-01A	3.0M	4#26，1#24　PVC（Thin cable）
	UC-CMC050-01A	5.0M	4#26，1#24　PVC（Thin cable）
	UC-CMC100-01A	10.0M	4#26，1#24　PVC（Thin cable）
	UC-CMC200-01A	20.0M	4#26，1#24　PVC（Thin cable）

Notes:

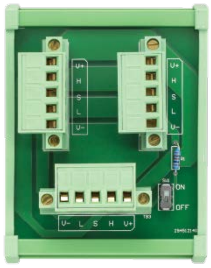
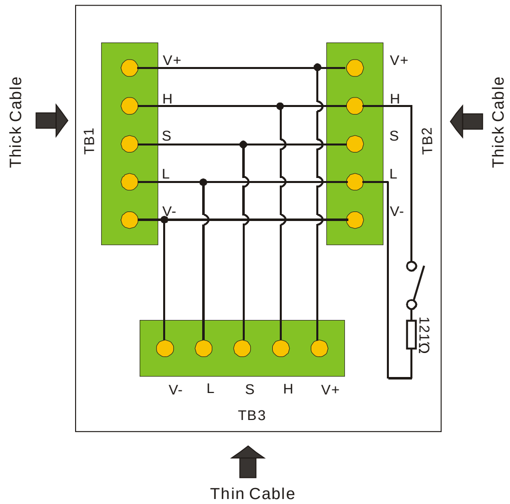
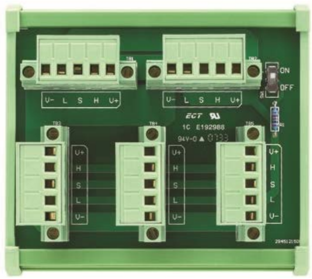
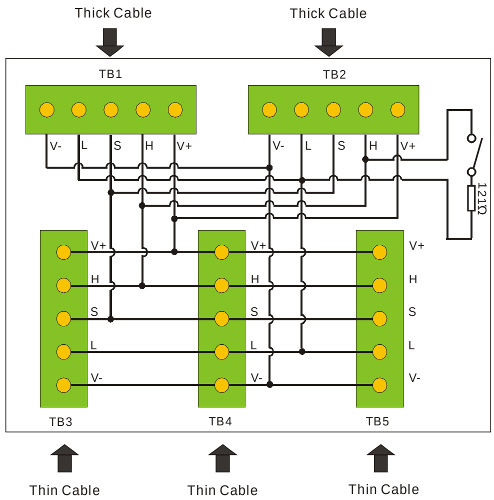
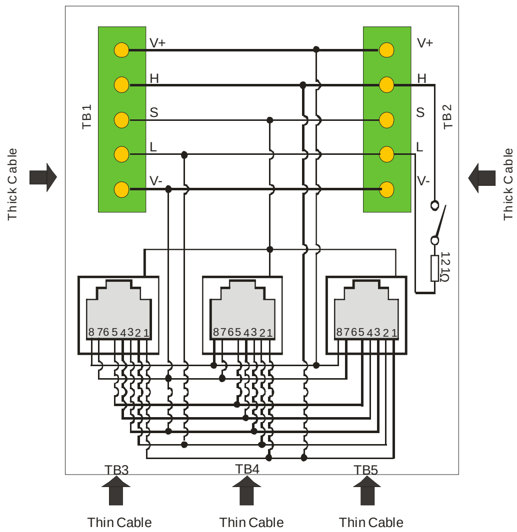
1. The maximum cable length for purchase is 305M per reel and mimimum length is 1M with metre as the unit.
2. UC-DN01Z-01A and UC-DN01Z-02A can be used as the main-line cable as well as the branch-line cable. The maximum communication distances that they support are different.
   The maximum communication distances the two cables support at different CANopen transmission speed are displayed as follows.

CANopen transmission speed（bit/s）	125K	250K	500K	1M
Max. communication distance for UC-DN01Z-01A（m）	500	250	100	40
Max. communication distance for UC-DN01Z-02A（m）	100	100	100	40

3. The maximum communication distance at a transmission speed is regulated in the CANopen protocol. The relationships between maximum communication distances and transmission speeds are shown in the following table.

Transmission speed（bit/s）	10K	20K	50K	125K	250K	500K	800K	1M
Max. communication distance（m）	5000	2500	1000	500	250	100	50	40

- **Distribution box**

Model		Circuit figure
TAP-CN01		
TAP-CN02		
TAP-CN03		
Connector	Removable terminals (5.08mm)	
Terminal resistor	120Ω	

- **Terminal resistor**

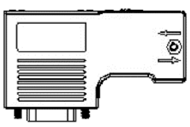
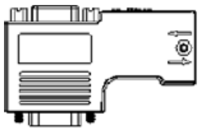
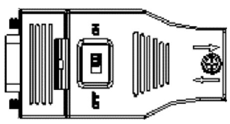
  As suggested in the CANopen protocol, the two ends of the CANopen communication cable should connect a terminal resistor of 120Ω (1/4W) respectively in order to match the impedance of the communication signal and reduce the signal reflection interference in normal signal transmission.

  - The terminal resistor connected to the start of the cable:
    The terminal resistor on the distribution box can be used just by setting the terminal resistor switch to ON.
  - The terminal resistor connected to the terminal end of the cable:
    A terminal resistor TAP-TR01 is needed for connecting to the other end of the cable.
  - The model of a terminal resistor: TAP-TR01, resistance value: 120Ω (1/4W) as shown below

TAP-TR01

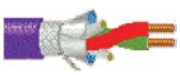

E.2 Accessories for PROFIBUS DP Communication

- **Connector**

	①	②	③
Model	 UN-03PF-01A	 UN-03PF-02A	 UN-03PF-03A
Connector	Male DB9 connector	Male DB9 connector	Male DB9 connector
Program planning connector	--	Female DB9 connector	--
Terminal resistor*[1]	120Ω	120Ω	120Ω

*[1.] : Please set the switches of the connectors to ON when the connectors are placed at two ends of the PROFIBUS network. Set the switches of the connectors to OFF if they are not placed at two ends of the PROFIBUS network.

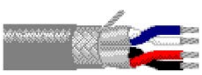
- **Cable**

	Model	Length	Diameter
	UC-PF01Z-01A	305M	1PR #22 AWG FRFPE FRPE

Note: The maximum cable length for purchase is 305M per reel and mimimum length is 1M with Metre as the unit.

E.3 Accessories for DeviceNet Communication

- **Cable**

Figure	Model	Length	Diameter（AWG）
	UC-DN01Z-01A	305M	2#15，2#18 SHLD PVC (Thick)
	UC-DN01Z-02A	305M	2#22，2#24 SHLD PVC (Thin)

Notes:

1. The maximum cable length for purchase is 305M per reel and mimimum length is 1M with metre as the unit.

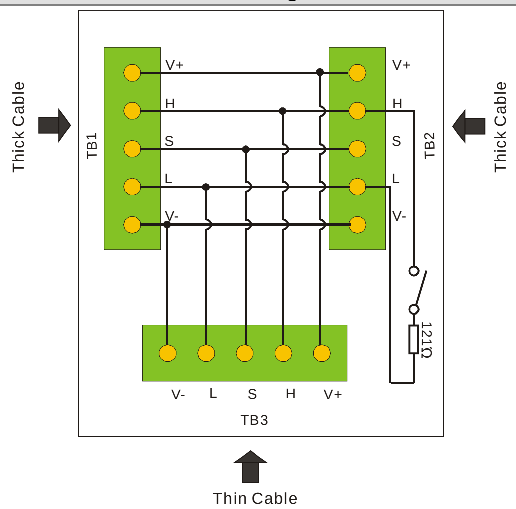
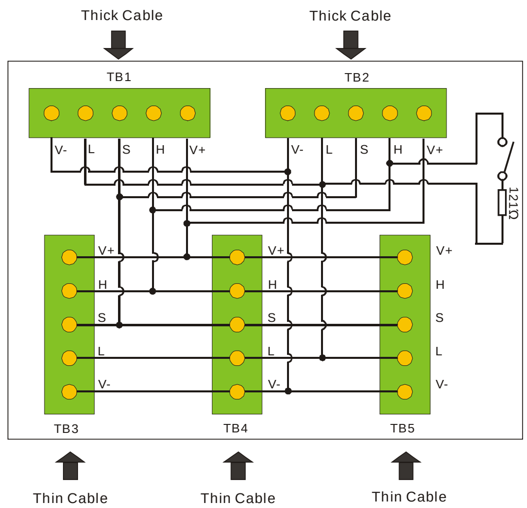
2. UC-DN01Z-01A and UC-DN01Z-02A can be used as the main-line cable as well as the branch-line cable. The maximum communication distances that they support are different.
   The maximum communication distances the two cables support at different DeviceNet transmission speed are displayed as follows.

DeviceNet transmission speed (bit/s)	125K	250K	500K
Max. communication distance for UC-DN01Z-01A (m)	500	250	100
Max. communication distance for UC-DN01Z-02A (m)	100	100	100

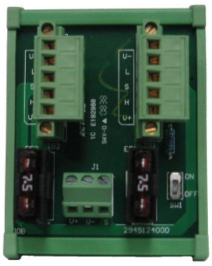
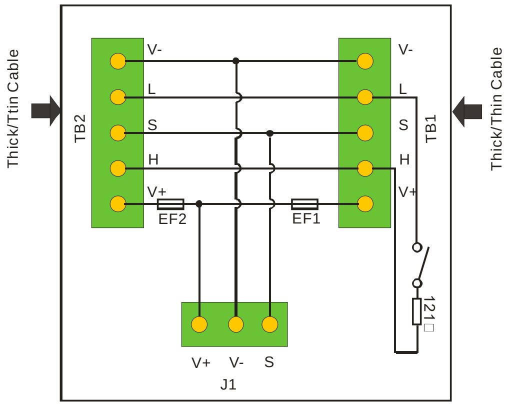
3. The maximum communication distance at a transmission speed is regulated in the DeviceNet protocol. The relationships between maximum communication distances and transmission speeds are shown in the following table.

Transmission speed (bit/s)	10K	20K	50K	125K	250K	500K
Max. communication distance (m)	5000	2500	1000	500	250	100

- **Distribution box**

Model		Circuit figure
TAP-CN01		
TAP-CN02		

E

Model		Circuit figure
TAP-CP01 (Power distribution box)		
Connector	Removable terminals (5.08mm)	
Terminal resistor	120Ω	

- **Terminal resistor**

As required in the DeviceNet protocol, the two ends of the DeviceNet communication cable should connect a terminal resistor of 120Ω (1/4W) respectively.

1. The terminal resistor connected to the start of the cable:
   The terminal resistor on the distribution box can be used by setting the terminal resistor switch to ON.
2. The terminal resistor connected to the terminal end of the cable:
   A terminal resistor of 120Ω (1/4W) is needed for connecting to the terminal end of the cable.