

SIEMENS

SIMATIC

S7 S7-200 SMART

System Manual

Preface

<u>Product overview</u>	1
<u>Getting started</u>	2
<u>Installation</u>	3
<u>PLC concepts</u>	4
<u>Programming concepts</u>	5
<u>PLC device configuration</u>	6
<u>Program instructions</u>	7
<u>Communication</u>	8
<u>Libraries</u>	9
<u>Debugging and troubleshooting</u>	10
<u>PID loops and tuning</u>	11
<u>Open loop motion control</u>	12
<u>Technical specifications</u>	A
<u>Calculating a power budget</u>	B
<u>Error codes</u>	C
<u>Special memory (SM) and system symbol names</u>	D
<u>References</u>	E
<u>Order numbers</u>	F

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Preface

Purpose of the manual

The S7-200 SMART series is a line of micro-programmable logic controllers (Micro PLCs) that can control a variety of automation applications. Compact design, low cost, and a powerful instruction set make the S7-200 SMART a perfect solution for controlling small applications. The wide variety of S7-200 SMART models and the Windows-based programming tool give you the flexibility you need to solve your automation problems.

This manual provides information about installing and programming the S7-200 SMART CPUs and is designed for engineers, programmers, installers, and electricians who have a general knowledge of programmable logic controllers.

Required basic knowledge

To understand this manual, it is necessary to have a general knowledge of automation and programmable logic controllers.

Scope of the manual

This manual describes the following products:

- STEP 7-Micro/WIN SMART V2.0
- S7-200 SMART CPU firmware release V2.0

For a complete list of the S7-200 SMART products and order numbers described in this manual, see Technical Specifications (Page 509).

Certification, CE label and other standards

Refer to the technical specifications for more information.

Service and support

In addition to our documentation, Siemens offers technical expertise on the Internet and on the customer support web site (<http://www.siemens.com/automation/>).

Contact your Siemens distributor or sales office for assistance in answering any technical questions, for training, or for ordering S7 products. Because your sales representatives are technically trained and have the most specific knowledge about your operations, process and industry, as well as about the individual Siemens products that you are using, they can provide the fastest and most efficient answers to any problems you might encounter.

Security information

Siemens provides automation and drive products with industrial security functions that support the secure operation of plants or machines. They are an important component in a holistic industrial security concept. With this in mind, our products undergo continuous development. We therefore recommend that you keep yourself informed with respect to our product updates. Please find further information and newsletters on this subject at: (<http://support.automation.siemens.com>)

To ensure the secure operation of a plant or machine it is also necessary to take suitable preventive action (e.g. cell protection concept) and to integrate the automation and drive components into a state-of-the-art holistic industrial security concept for the entire plant or machine. Any third-party products that may be in use must also be taken into account. Please find further information at: (<http://www.siemens.com/industrialsecurity>)

Table of contents

Preface	3
1 Product overview	15
1.1 S7-200 SMART CPU	15
1.2 S7-200 SMART expansion modules.....	18
1.3 HMI devices for S7-200 SMART	19
1.4 Communications options.....	20
1.5 Programming software	21
1.6 New features	22
2 Getting started	23
2.1 Connecting to the CPU	23
2.1.1 Configuring the CPU for communication	24
2.1.1.1 Overview	24
2.1.1.2 Establishing the hardware communication connection.....	25
2.1.1.3 Setting up communication with the CPU	25
2.2 Creating the sample program	27
2.2.1 Network 1: Starting the timer	29
2.2.2 Network 2: Turning the output on.....	30
2.2.3 Network 3: Resetting the timer.....	31
2.2.4 Setting the CPU type and version for your project.....	31
2.2.5 Saving the sample project.....	32
2.3 Downloading the sample program	33
2.4 Changing the operating mode of the CPU	33
3 Installation	35
3.1 Guidelines for installing S7-200 SMART devices	35
3.2 Power budget	37
3.3 Installation and removal procedures	38
3.3.1 Mounting dimensions for the S7-200 SMART devices	38
3.3.2 Installing and removing the CPU	39
3.3.3 Installing and removing an expansion module.....	42
3.3.4 Installing and removing a signal board or battery board.....	43
3.3.5 Removing and reinstalling the terminal block connector	45
3.4 Wiring guidelines.....	46
4 PLC concepts	51
4.1 Execution of the control logic	51
4.1.1 Reading the inputs and writing to the outputs.....	52
4.1.2 Immediately reading or writing the I/O	53
4.1.3 Executing the user program.....	53

4.2	Accessing data	55
4.2.1	Accessing memory areas	56
4.2.2	Format for Real numbers	62
4.2.3	Format for strings	63
4.2.4	Assigning a constant value for instructions	63
4.2.5	Addressing the local and expansion I/O	64
4.2.6	Using pointers for indirect addressing	64
4.2.7	Pointer examples	67
4.3	Saving and restoring data	68
4.3.1	Downloading project components	68
4.3.2	Uploading project components	71
4.3.3	Types of storage	72
4.3.4	Using a memory card	73
4.3.5	Inserting a memory card in the CPU	75
4.3.6	Transferring your program with a memory card	75
4.3.7	Restoring data after power on	78
4.4	Changing the operating mode of the CPU	78
5	Programming concepts	79
5.1	Guidelines for designing a PLC system	79
5.2	Elements of the user program	81
5.3	Creating your user program	83
5.3.1	Earlier versions of STEP 7-Micro/WIN projects	83
5.3.2	Using STEP 7-Micro/WIN SMART user interface	85
5.3.3	Using STEP 7-Micro/WIN SMART to create your programs	86
5.3.4	Using wizards to help you create your control program	87
5.3.5	Features of the LAD editor	88
5.3.6	Features of the FBD editor	88
5.3.7	Features of the STL editor	89
5.4	Data block (DB) editor	90
5.5	Symbol table	92
5.6	Variable table	95
5.7	PLC error reaction	101
5.7.1	Non-fatal errors and I/O errors	101
5.7.2	Fatal errors	103
5.8	Program edit in RUN mode	103
5.9	Features for debugging your program	106
6	PLC device configuration	107
6.1	Configuring the operation of the PLC system	107
6.1.1	System block	107
6.1.2	Configuring communication	109
6.1.3	Configuring the digital inputs	111
6.1.4	Configuring the digital outputs	113
6.1.5	Configuring the retentive ranges	114
6.1.6	Configuring system security	115
6.1.7	Configuring the startup options	119

6.1.8	Configuring the analog inputs	120
6.1.9	Configuring the analog outputs	123
6.1.10	Configuring the RTD analog inputs.....	125
6.1.11	Configuring the TC analog inputs	129
6.1.12	Configuring the RS485/RS232 CM01 communications signal board	133
6.1.13	Configuring the BT01 battery signal board	134
6.1.14	Clearing PLC memory.....	135
6.1.15	Creating a reset-to-factory-defaults memory card	137
6.2	High-speed I/O	137
7	Program instructions	139
7.1	Bit logic.....	139
7.1.1	Standard inputs	139
7.1.2	Immediate inputs.....	141
7.1.3	Logic stack overview.....	142
7.1.4	STL logic stack instructions	144
7.1.5	NOT	146
7.1.6	Positive and negative transition detectors	147
7.1.7	Coils: output and output immediate instructions	148
7.1.8	Set, reset, set immediate, and reset immediate functions	149
7.1.9	Set and reset dominant bistable	150
7.1.10	NOP (No operation) instruction	151
7.1.11	Bit logic input examples	152
7.1.12	Bit logic output examples	153
7.2	Clock	155
7.2.1	Read and set real-time clock	155
7.2.2	Read and set real-time clock extended	157
7.3	Communication	160
7.3.1	GET and PUT (Ethernet)	160
7.3.2	Transmit and receive (Freeport on RS485/RS232)	167
7.3.3	Get port address and set port address (PPI protocol on RS485/RS232)	180
7.3.4	Get IP address and set IP address (Ethernet).....	181
7.4	Compare	182
7.4.1	Compare number values.....	182
7.4.2	Compare character strings.....	186
7.5	Convert.....	188
7.5.1	Standard conversion instructions.....	188
7.5.2	ASCII character array conversion	191
7.5.3	Number value to ASCII string conversion.....	197
7.5.4	ASCII sub-string to number value conversion	202
7.5.5	Encode and decode	205
7.6	Counters.....	206
7.6.1	Counter instructions	206
7.6.2	High-speed counter instructions	210
7.6.3	Noise reduction for high-speed inputs	214
7.6.4	High-speed counter programming	216
7.6.5	Example initialization sequences for high-speed counters	228
7.7	Pulse output	236
7.7.1	Pulse output instruction (PLS)	236

7.7.2	Pulse width modulation (PWM).....	238
7.7.3	Using SM locations to configure and control the PWM operation.....	238
7.8	Math	240
7.8.1	Add, subtract, multiply, and divide	240
7.8.2	Multiply integer to double integer and divide integer with remainder.....	244
7.8.3	Trigonometry, natural logarithm/exponential, and square root	246
7.8.4	Increment and decrement	248
7.9	PID	250
7.9.1	Using the PID wizard.....	251
7.9.2	PID algorithm	256
7.9.3	Converting and normalizing the loop inputs.....	259
7.9.4	Converting the loop output to a scaled integer value.....	260
7.9.5	Forward- or reverse-acting loops	261
7.10	Interrupt.....	263
7.10.1	Interrupt instructions.....	263
7.10.2	Interrupt routine overview and CPU model event support.....	265
7.10.3	Interrupt programming guidelines	267
7.10.4	Types of interrupt events that the S7-200 SMART CPU supports.....	268
7.10.5	Interrupt priority, queuing, and example program.....	269
7.11	Logical operations	275
7.11.1	Invert	275
7.11.2	AND, OR, and exclusive OR.....	276
7.12	Move.....	278
7.12.1	Move byte, word, double word, or real.....	278
7.12.2	Block move.....	279
7.12.3	Swap bytes.....	280
7.12.4	Move byte immediate (read and write).....	281
7.13	Program control.....	282
7.13.1	FOR-NEXT loop	282
7.13.2	JMP (jump to label)	284
7.13.3	SCR (sequence control relay)	285
7.13.4	END, STOP, and WDR (watchdog timer reset)	294
7.13.5	GET_ERROR (Get non-fatal error code)	296
7.14	Shift and rotate	297
7.14.1	Shift and rotate	297
7.14.2	Shift register bit	300
7.15	String	302
7.15.1	String (Get length, copy, and concatenate)	302
7.15.2	Copy substring from string	305
7.15.3	Find string and first character within string	306
7.16	Table	309
7.16.1	Add to table	309
7.16.2	First-in-first-out and last-in-first-out	310
7.16.3	Memory fill	313
7.16.4	Table find	314
7.17	Timer	318
7.17.1	Timer instructions	318

7.17.2	Timer programming tips and examples	321
7.17.3	Interval timers.....	327
7.18	Subroutine.....	328
7.18.1	CALL (subroutine) and RET (conditional return)	328
8	Communication.....	335
8.1	CPU communication connections	335
8.2	CPU communication ports	336
8.3	HMs and communication drivers.....	337
8.4	Ethernet.....	338
8.4.1	Overview	338
8.4.2	TCP/IP protocol.....	339
8.4.3	Local/partner connection.....	339
8.4.4	Sample Ethernet network configurations	340
8.4.5	Assigning Internet Protocol (IP) addresses	341
8.4.5.1	Assigning IP addresses to programming and network devices	341
8.4.5.2	Configuring or changing an IP address for a CPU or device in your project	343
8.4.5.3	Searching for CPUs and devices on your Ethernet network	350
8.4.6	Locating the Ethernet (MAC) address on the CPU.....	351
8.4.7	HMI-to-CPU communication	353
8.5	RS485	354
8.5.1	PPI protocol.....	354
8.5.2	Baud rate and network address	355
8.5.2.1	Definition of baud rate and network address	355
8.5.2.2	Setting the baud rate and network address for the S7-200 SMART CPU.....	356
8.5.3	Sample RS485 network configurations.....	358
8.5.3.1	Single-master PPI networks.....	358
8.5.3.2	Multi-master and multi-slave PPI networks.....	358
8.5.4	Building your network.....	359
8.5.4.1	General guidelines	359
8.5.4.2	Determining the distances, transmission rates, and cable lengths for your network.....	360
8.5.4.3	Repeaters on the network	360
8.5.4.4	Selection of the network cable	361
8.5.4.5	Connector pin assignments	361
8.5.4.6	Biassing and terminating the network cable	362
8.5.4.7	Biassing and terminating the CM01 signal board	363
8.5.4.8	Using HMI devices on your RS485 network	364
8.5.5	Freeport mode.....	365
8.5.5.1	Creating user-defined protocols with Freeport mode.....	365
8.5.5.2	Using the RS232/PPI Multi-Master cable and Freeport mode with RS232 devices	366
8.6	RS232	368
9	Libraries.....	369
9.1	Creating a user-defined library of instructions	369
9.2	USS library instructions.....	370
9.2.1	USS communication overview	370
9.2.1.1	USS protocol overview	370
9.2.1.2	Requirements for using the USS protocol	371
9.2.1.3	Calculating the time required for communicating with the drive	372

9.2.2	USS program instructions	373
9.2.2.1	Using the USS protocol instructions	373
9.2.2.2	USS_INIT instruction	374
9.2.2.3	USS_CTRL instruction	376
9.2.2.4	USS_RPM_x instruction	379
9.2.2.5	USS_WPM_x instruction	381
9.2.2.6	USS protocol execution error codes	383
9.2.2.7	USS protocol example program	384
9.3	Modbus library instructions	386
9.3.1	Modbus communication overview	386
9.3.1.1	Modbus library features	386
9.3.1.2	Initialization and execution time for Modbus protocol	389
9.3.1.3	Modbus addressing	389
9.3.2	Modbus RTU master	391
9.3.2.1	Using the Modbus master instructions	391
9.3.2.2	MBUS_CTRL instruction (initialize master)	392
9.3.2.3	MBUS_MSG instruction	394
9.3.2.4	Modbus master execution error codes	397
9.3.3	Modbus RTU slave	398
9.3.3.1	Using the Modbus slave instructions	398
9.3.3.2	MBUS_INIT instruction (initialize slave)	400
9.3.3.3	MBUS_SLAVE instruction	401
9.3.3.4	Modbus slave execution error codes	402
9.3.4	Modbus master example program	403
9.3.5	Modbus advanced user information	405
10	Debugging and troubleshooting.....	409
10.1	Debugging your program	409
10.1.1	Bookmark functions	409
10.1.2	Cross reference table	410
10.2	Displaying program status	412
10.2.1	Displaying status in the program editor	412
10.2.2	Configuring the STL status options	415
10.3	Using a status chart to monitor your program	416
10.4	Forcing specific values	418
10.5	Writing and forcing outputs in STOP mode	419
10.6	How to execute a limited number of scans	420
10.7	Hardware troubleshooting guide	421
11	PID loops and tuning.....	423
11.1	PID loop definition table	424
11.2	Prerequisites	427
11.3	Auto-hysteresis and auto-deviation	428
11.4	Auto-tune sequence	428
11.5	Exception conditions	430
11.6	Notes concerning PV out-of-range (result code 3)	431

11.7	PID Tune control panel	431
12	Open loop motion control	437
12.1	Using the PWM output	437
12.1.1	Configuring the PWM output.....	438
12.1.2	PWMx_RUN subroutine	439
12.2	Using motion control	440
12.2.1	Maximum and start/stop speeds	440
12.2.2	Entering the acceleration and deceleration times.....	441
12.2.3	Configuring the motion profiles	442
12.3	Features of motion control	445
12.4	Programming an Axis of Motion.....	446
12.5	Configuring an Axis of Motion	448
12.6	Subroutines created by the Motion wizard for the Axis of Motion.....	460
12.6.1	Guidelines for using the Motion subroutines.....	461
12.6.2	AXISx_CTRL subroutine	462
12.6.3	AXISx_MAN subroutine	463
12.6.4	AXISx_GOTO subroutine	465
12.6.5	AXISx_RUN subroutine.....	466
12.6.6	AXISx_RSEEK subroutine	467
12.6.7	AXISx_LDOFF subroutine.....	468
12.6.8	AXISx_LDPOS subroutine	469
12.6.9	AXISx_SRATE subroutine	470
12.6.10	AXISx_DIS subroutine	471
12.6.11	AXISx_CFG subroutine	471
12.6.12	AXISx_CACHE subroutine	472
12.6.13	AXISx_RDPOS subroutine.....	473
12.6.14	AXISx_ABSPOS subroutine.....	474
12.7	Sample programs for the Axis of Motion.....	475
12.8	Monitoring the Axis of Motion.....	481
12.8.1	Displaying and controlling the operation of the Axis of Motion	483
12.8.2	Displaying and modifying the configuration of the Axis of Motion	488
12.8.3	Displaying the profile configuration for the Axis of Motion.....	488
12.8.4	Error codes for the Axis of Motion (WORD at SMW620, SMW670, or SMW720).....	489
12.8.5	Error codes for the Motion instruction (seven LS bits of SMB634, SMB684, or SMB734).....	491
12.9	Advanced topics.....	492
12.9.1	Understanding the configuration/profile table for the Axis of Motion	492
12.9.2	Special memory (SM) locations for the Axis of Motion	500
12.10	Understanding the RP Seek modes of the Axis of Motion.....	503
12.10.1	Selecting the work zone location to eliminate backlash	508
A	Technical specifications	509
A.1	General specifications.....	509
A.1.1	General technical specifications	509
A.2	S7-200 SMART CPUs	513
A.2.1	CPU ST20 and CPU SR20	513
A.2.1.1	General specifications and features.....	513

A.2.1.2	Digital inputs and outputs.....	517
A.2.1.3	CPU ST20 and CPU SR20 wiring diagrams	519
A.2.2	CPU ST30 and CPU SR30	521
A.2.2.1	General specifications and features.....	521
A.2.2.2	Digital inputs and outputs.....	524
A.2.2.3	CPU ST30 and CPU SR30 wiring diagrams	526
A.2.3	CPU ST40, CPU SR40, and CPU CR40	528
A.2.3.1	General specifications and features.....	528
A.2.3.2	Digital inputs and outputs.....	531
A.2.3.3	CPU ST40, SR40 and CR40 wiring diagrams	534
A.2.4	CPU ST60, CPU SR60, and CPU CR60	537
A.2.4.1	General specifications and features.....	537
A.2.4.2	Digital inputs and outputs.....	540
A.2.4.3	CPU ST60, SR60 and CR60 wiring diagrams	543
A.2.5	Wiring diagrams for sink and source input, and relay output.....	546
A.3	Digital inputs and outputs expansion modules (EMs).....	547
A.3.1	EM DI08 digital input specifications	547
A.3.2	EM DT08 and EM DR08 digital output specifications	548
A.3.3	EM DT16, EM DR16, EM DT32, and EM DR32 digital input/output specifications	551
A.4	Analog inputs and outputs expansion modules (EMs).....	557
A.4.1	EM AE04 analog input specifications	557
A.4.2	EM AQ02 analog output module specifications	559
A.4.3	EM AM06 analog input/output module specifications	561
A.5	Thermocouple and RTD expansion modules (EMs).....	564
A.5.1	Thermocouple expansion modules (EMs)	564
A.5.1.1	EM AT04 thermocouple specifications.....	564
A.5.2	RTD expansion modules (EMs)	569
A.6	Digital signal boards.....	574
A.6.1	SB DT04 digital input/output specifications.....	574
A.7	Analog signal boards.....	577
A.7.1	SB AQ01 analog output specifications.....	577
A.8	RS485/RS232 signal boards.....	579
A.8.1	SB RS485/RS232 specifications.....	579
A.9	Battery board signal boards (SBs)	581
A.9.1	SB BA01 Battery board	581
B	Calculating a power budget.....	583
B.1	Power budget	583
B.2	Calculating a sample power requirement.....	584
B.3	Calculating your power requirement	586
C	Error codes	587
C.1	PLC non-fatal error codes	587
C.2	PLC non-fatal error SM flags.....	590
C.3	PLC fatal error codes	590
C.4	Timestamp mismatch	592

D	Special memory (SM) and system symbol names.....	593
D.1	SM (Special Memory) overview	593
D.2	SMB0: System status	595
D.3	SMB1: Instruction execution status.....	596
D.4	SMB2: Freeport receive character	597
D.5	SMB3: Freeport character error	597
D.6	SMB4: Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and value forced.....	598
D.7	SMB5: I/O error status	598
D.8	SMB6-SMB7: CPU ID, error status, and digital I/O points.....	599
D.9	SMB8-SMB19: I/O module ID and errors	599
D.10	SMW22-SMW26: Scan times	600
D.11	SMB28-SMB29: Signal board ID and errors.....	601
D.12	SMB30: (port 0) and SMB130: (port 1)	601
D.13	SMB34-SMB35: Time intervals for timed interrupts.....	602
D.14	SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3): high-speed counters.....	603
D.15	SMB67-SMB71, SMB77-SMB81, and SMB567-SMB571: PWM0, PWM1, and PWM2 high-speed outputs	606
D.16	SMB86-SMB94 and SMB186-SMB194: Receive message control.....	608
D.17	SMW98: I/O expansion bus communication errors	610
D.18	SMW100-SMW114 System alarms	611
D.19	SMB130: Freeport control for port 1 (See SMB30).....	612
D.20	SMB136-SMB145: HSC3 high-speed counter.....	612
D.21	SMB186-SMB194: Receive message control (See SMB86-SMB94).....	612
D.22	SMB480-SMB515: Data log status	612
D.23	SMB567-SMB571: PWM2 high-speed output (See SMB67-SMB71).....	613
D.24	SMB600-SMB749: Axis (0, 1, and 2) open loop motion control	613
D.25	SMB650-SMB699: Axis 1 open loop motion control (See SMB600-SMB740)	615
D.26	SMB700-SMB749: Axis 2 open loop motion control (See SMB600-SMB740)	615
D.27	SMB1000-SMB1049: CPU hardware/firmware ID	615
D.28	SMB1050-SMB1099: SB (signal board) hardware/firmware ID.....	615
D.29	SMB1100-SMB1399: EM (expansion module) hardware/firmware ID	616
E	References	619
E.1	Often-used special memory bits	619
E.2	Interrupt events in priority order	620

Table of contents

E.3	High-speed counter summary	621
E.4	Instructions	621
E.5	Memory ranges and features	628
F	Order numbers.....	631
F.1	Order numbers	631
Index	635

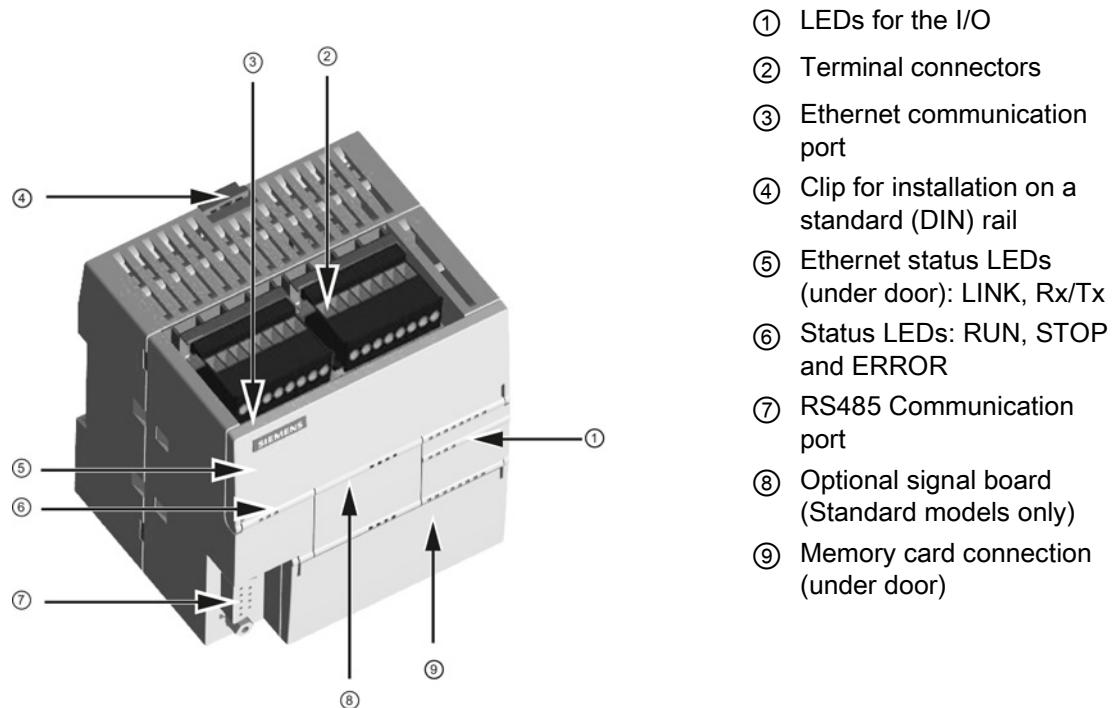
Product overview

The S7-200 SMART series of micro-programmable logic controllers (Micro PLCs) can control a wide variety of devices to support your automation needs.

The CPU monitors inputs and changes outputs as controlled by the user program, which can include Boolean logic, counting, timing, complex math operations, and communications with other intelligent devices. The compact design, flexible configuration, and powerful instruction set combine to make the S7-200 SMART a perfect solution for controlling a wide variety of applications.

1.1 S7-200 SMART CPU

The CPU combines a microprocessor, an integrated power supply, input circuits, and output circuits in a compact housing to create a powerful Micro PLC. After you have downloaded your program, the CPU contains the logic required to monitor and control the input and output devices in your application.



The CPU provides different models with a diversity of features and capabilities that help you create effective solutions for your varied applications. The different models of CPUs are shown below. For detailed information about a specific CPU, see the technical specifications (Page 513).

Product overview

1.1 S7-200 SMART CPU

Table 1- 1 S7-200 SMART CPUs

	CR40	CR60	SR20	ST20	SR30	ST30	SR40	ST40	SR60	ST60
Compact, non-expandable	X	X								
Standard, expandable			X	X	X	X	X	X	X	X
Relay output	X	X	X		X		X		X	
Transistor output (DC)				X		X		X		X
I/O points (built-in)	40	60	20	20	30	30	40	40	60	60

Table 1- 2 Compact non-expandable CPUs

Features	CPU CR40		CPU CR60
Dimensions: W x H x D (mm)	125 x 100 x 81		175 x 100 x 81
User memory	Program	12 Kbytes	12 Kbytes
	User data	8 Kbytes	8 Kbytes
	Retentive	10 Kbytes max. ¹	10 Kbytes max. ¹
On-board digital I/O	<ul style="list-style-type: none"> • Inputs • 24 DI • Outputs • 16 DQ Relay 		<ul style="list-style-type: none"> 36 DI 24 DQ Relay
Expansion modules	None		None
Signal board	None		None
High-speed counters	4 at 100 K Hz single phase or 2 at 50 K Hz A/B phase		4 at 100 K Hz single phase or 2 at 50 K Hz A/B phase
PID loops	8		8
Real-time clock with 7-day back-up	No		No

¹ You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive timers) to be retentive, up to the specified maximum amount.

Table 1- 3 Standard expandable CPUs

Features	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU SR60, CPU ST60															
Dimensions: W x H x D (mm)	90 x 100 x 81	110 x 100 x 81	125 x 100 x 81	175 x 100 x 81															
User memory	<table> <tr> <td>Program</td><td>12 Kbytes</td><td>18 Kbytes</td><td>24 Kbytes</td><td>30 Kbytes</td></tr> <tr> <td>User data</td><td>8 Kbytes</td><td>12 Kbytes</td><td>16 Kbytes</td><td>20 Kbytes</td></tr> <tr> <td>Retentive</td><td>10 Kbytes max.¹</td><td>10 Kbytes max.¹</td><td>10 Kbytes max.¹</td><td>10 Kbytes max.¹</td></tr> </table>	Program	12 Kbytes	18 Kbytes	24 Kbytes	30 Kbytes	User data	8 Kbytes	12 Kbytes	16 Kbytes	20 Kbytes	Retentive	10 Kbytes max. ¹						
Program	12 Kbytes	18 Kbytes	24 Kbytes	30 Kbytes															
User data	8 Kbytes	12 Kbytes	16 Kbytes	20 Kbytes															
Retentive	10 Kbytes max. ¹	10 Kbytes max. ¹	10 Kbytes max. ¹	10 Kbytes max. ¹															
On-board digital I/O	<ul style="list-style-type: none"> Inputs • 12 DI • 8 DQ 	<ul style="list-style-type: none"> • 18 DI • 12 DQ 	<ul style="list-style-type: none"> • 24 DI • 16 DQ 	<ul style="list-style-type: none"> • 36 DI • 24 DQ 															
Expansion modules	6 max.	6 max.	6 max.	6 max.															
Signal board	1	1	1	1															
High-speed counters	<table> <tr> <td>4 at 200 K Hz single phase</td><td>4 at 200 K Hz single phase</td><td>4 at 200 K Hz single phase</td><td>4 at 200 K Hz single phase</td></tr> <tr> <td>or</td><td>or</td><td>or</td><td>or</td></tr> <tr> <td>2 at 100 K Hz A/B phase</td><td>2 at 100 K Hz A/B phase</td><td>2 at 100 K Hz A/B phase</td><td>2 at 100 K Hz A/B phase</td></tr> </table>	4 at 200 K Hz single phase	4 at 200 K Hz single phase	4 at 200 K Hz single phase	4 at 200 K Hz single phase	or	or	or	or	2 at 100 K Hz A/B phase									
4 at 200 K Hz single phase	4 at 200 K Hz single phase	4 at 200 K Hz single phase	4 at 200 K Hz single phase																
or	or	or	or																
2 at 100 K Hz A/B phase	2 at 100 K Hz A/B phase	2 at 100 K Hz A/B phase	2 at 100 K Hz A/B phase																
Pulse outputs ²	2 at 100 KHz	3 at 100 KHz	3 at 100 KHz	3 at 100 KHz															
PID loops	8	8	8	8															
Real-time clock with 7-day back-up	Yes	Yes	Yes	Yes															

¹ You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive timers) to be retentive, up to the specified maximum amount.

² The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Refer to the technical specifications (Page 509) for the power requirements of the CPU and the expansion modules. Use the worksheets in Appendix B, Calculating a power budget (Page 586) to calculate your power budget.

1.2 S7-200 SMART expansion modules

To better solve your application requirements, the S7-200 SMART family includes a wide variety of expansion modules, and signal boards. You can use these expansion modules with the standard CPU models (SR20, ST20, SR30, ST30, SR40, ST40, SR60 or ST60) to add additional functionality to the CPU. The following table provides a list of the expansion modules that are currently available. For detailed information about a specific module, see the technical specifications (Page 509).

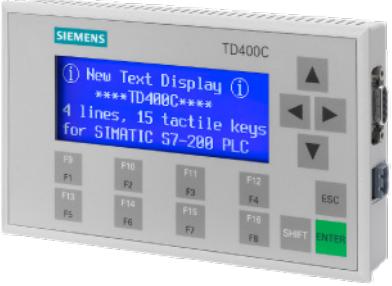
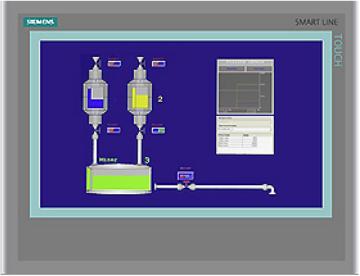
Table 1- 4 Signal modules and signal boards

Type	Input only	Output only	Combination In/Out	Other
Digital signal module	<ul style="list-style-type: none">• 8 x DC In	<ul style="list-style-type: none">• 8 x DC Out• 8 x Relay Out	<ul style="list-style-type: none">• 8 x DC In / 8 x DC Out• 8 x DC In / 8 x Relay Out• 16 x DC In / 16 x DC Out• 16 x DC In / 16 x Relay Out	
Analog signal modules	<ul style="list-style-type: none">• 4 x Analog In• 2 x RTD In• 4 x TC In	<ul style="list-style-type: none">• 2 x Analog Out	<ul style="list-style-type: none">• 4 x Analog In / 2 x Analog Out	
Signal boards	<ul style="list-style-type: none">• 1 x Analog Out		<ul style="list-style-type: none">• 2 x DC In x 2 x DC Out	<ul style="list-style-type: none">• RS485/RS232• Battery Board

1.3 HMI devices for S7-200 SMART

The S7-200 SMART supports Comfort HMIs, SMART HMIs, Basic HMIs and Micro HMIs. The TD400C and the SMART LINE Touch Panel are shown below. Refer to Appendix C, Human Machine Interface (Page 631) for a complete list of supported devices and order numbers.

Table 1- 5 HMI devices

 	<p>Text Display unit: The TD400C is a display device that can be connected to the CPU. Using the Text Display wizard, you can easily program your CPU to display text messages and other data pertaining to your application. The TD400C device provides a low cost interface to your application by allowing you to view, monitor, and change the process variables pertaining to your application.</p>
	<p>Basic HMIs: The SMART LINE Touch Panel provides operating and monitoring functions for small-scale machines and plants. Short configuration and commissioning times, their configuration in WinCC flexible/WinCC Basic/STEP 7 Basic, and a PROFINET interface form the highlights of these HMIs.</p>

The Text Display wizard in STEP 7-Micro/WIN SMART helps you configure Text Display messages quickly and easily for the TD400C. To start the Text Display wizard, select the "Text Display" command from the "Tools" menu.

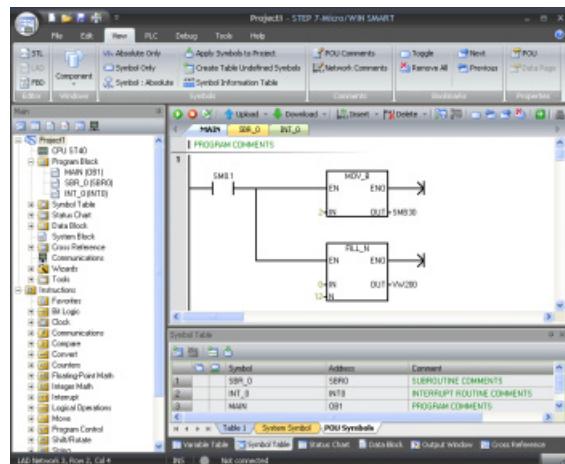
The SIMATIC Text Display (TD) User Manual can be downloaded from the Siemens customer support web site (Page 3).

1.4 Communications options

The S7-200 SMART offers several types of communication between CPUs, programming devices, and HMIs:

- Ethernet:
 - Exchange of data from the programming device to the CPU
 - Exchange of data between HMIs and the CPU
 - S7 peer-to-peer communication with other S7-200 SMART CPUs
- RS485:
 - Supports a total of 126 addressable devices (32 devices per network segment)
 - Supports PPI (point-to-point interface) protocol
 - Exchange of data between HMIs and the CPU
 - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)
- RS232:
 - Supports a point-to-point connection to one device
 - Supports PPI protocol
 - Exchange of data between HMIs and the CPU
 - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)

1.5 Programming software



STEP7-Micro/WIN SMART provides a user-friendly environment to develop, edit, and monitor the logic needed to control your application.

At the top is a quick access toolbar for frequent tasks, followed by menus for all common functions. At the left is the project tree and navigation bar for easy access to components and instructions. The program editor and other components that you open occupy the remainder of the user interface.

STEP7-Micro/WIN SMART provides three program editors (LAD, FBD, and STL) for convenience and efficiency in developing the control program for your application.

To help you find the information you need, STEP7-Micro/WIN SMART provides an extensive online help system.

Computer requirements

STEP 7-Micro/WIN SMART runs on a personal computer. Your computer should meet the following minimum requirements:

- Operating system: Windows XP SP3 (32 bit only), Windows 7 (both 32 bit and 64 bits supported)
- At least 350M bytes of free hard disk space
- Mouse (recommended)

Installing STEP 7-Micro/WIN SMART

Insert the STEP 7-Micro/WIN SMART CD into the CD-ROM drive of your computer or contact your Siemens distributor or sales office to download STEP7-Micro/WIN SMART from the customer support web site (Page 3). Installation starts automatically and prompts you through the installation process. Refer to the Readme file for more information about installing STEP 7-Micro/WIN SMART.

Note

To install STEP 7-Micro/WIN SMART on a Windows XP or Windows 7 operating system, you must log in with Administrator privileges.

1.6 New features

STEP 7-Micro/WIN SMART V2.0 and the S7-200 SMART V2.0 CPUs introduce the following new features:

- New CPU models (Page 513):
 - CPU ST20 (DC/DC/DC)
 - CPU ST30 (DC/DC/DC)
 - CPU SR30 (AC/DC/Relay)
 - CPU CR60 (AC/DC/Relay)
- New battery signal board (Page 581)(SB BA01) extends preservation of the CPUs time and date information when powered down
- New four-channel thermocouple input expansion module (Page 564) (EM AT04)
- CPUs can now have up to six expansion modules connected to a CPU
- Download in RUN mode (Page 103): You can edit a program and download it to the CPU while the CPU is in RUN mode. Some restrictions apply.
- Program editor improvements including zooming, sub-branching, connecting elements through connection points, autocompletion when entering symbol names, and automatic generation of new symbols in the symbol table
- Stronger security features for password protection for CPUs and the ability to hide portions of the user program from unauthorized users
- Ability to temporarily authorize access to a password-protected POU
- An I/O symbol mapping table in STEP 7-Micro/WIN SMART for the configured CPU that shows the I/O points available on the module and provides a convenient place to give names and descriptions for the I/O.
- Ability to launch the SINAMICS V90 drives configuration tool.
- New GET/PUT instructions that provide a means for reading and writing data from one CPU to another CPU on the same Ethernet network
- Additional motion (Page 437) capabilities:
 - Jerk compensation
 - Read absolute position allows you to read the SINAMIC V90 drive current position value
- Enhanced communication dialog (Page 343)that allows selection of devices from manually entered IP address
- New wizards
 - Get/Put (Page 87): provides the ability to select the type of operation (GET or PUT), the size of the transfer, and the remote CPU and its address in order to create the program blocks for GET/PUT operations.
 - Data Log (Page 87): provides the ability to configure up to four permanent data log files and assign the PLC process data to log in each file. The wizard creates the data log write subroutines which your program can call to log data according to the requirements of your application.

Getting started

STEP 7-Micro/WIN SMART makes it easy for you to program your CPU. In just a few short steps using a simple example, you can learn how to create a user program that you can download and run on your CPU.

All you need for this example is an Ethernet cable, a CPU, and a programming device running the STEP 7-Micro/WIN SMART programming software.

2.1 Connecting to the CPU

Connecting your CPU is easy. For this example, you only need to connect power to your CPU and then connect the Ethernet communication cable between your programming device and the CPU.

Connecting power to the CPU



WARNING

Ensure power is off prior to installing, wiring or removing devices

Before you install or remove any electrical device, ensure that the power to that equipment has been turned off.

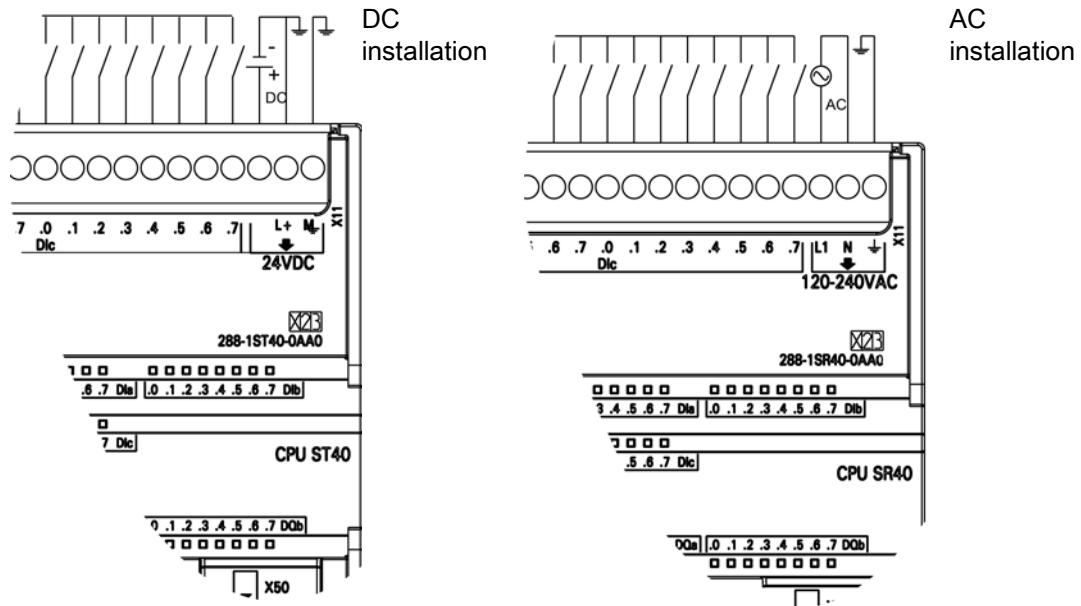
Attempts to install or connect the wiring for the CPU or related equipment with power applied could cause electric shock or faulty operation of equipment. Failure to disable all power to the CPU and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

Always follow appropriate safety precautions and ensure that power to the CPU is disabled before attempting to install or remove the CPU or related equipment.

Getting started

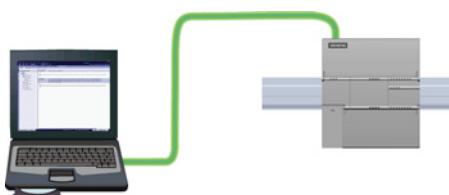
2.1 Connecting to the CPU

Connect the CPU to a power source. The following figure shows the wiring connections for either a DC or an AC model of the CPU.



2.1.1 Configuring the CPU for communication

2.1.1.1 Overview



A CPU can communicate with a STEP 7-Micro/WIN SMART programming device on an Ethernet network.

Consider the following when setting up communications between a CPU and a programming device:

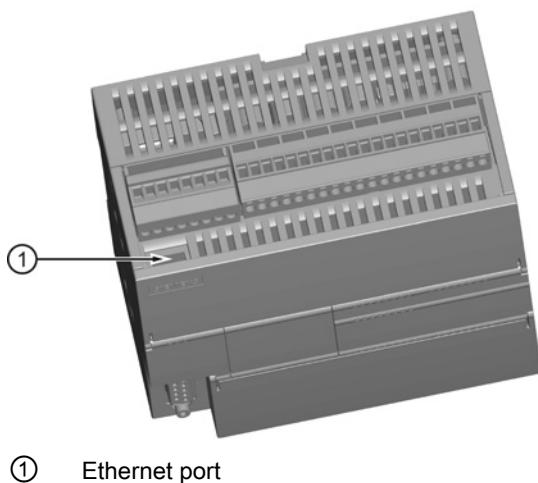
- Configuration/Setup: No hardware configuration is required for a single CPU. If you want multiple CPU's on the same network, then you must change the default IP addresses to new, unique IP addresses.
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

2.1.1.2 Establishing the hardware communication connection

The Ethernet interfaces establish the physical connections between a programming device and a CPU. Since Auto-Cross-Over functionality is built into the CPU, either a standard or crossover Ethernet cable can be used for the interface. An Ethernet switch is not required to connect a programming device directly to a CPU.

Follow the steps below to create the hardware connection between a programming device and a CPU:

1. Install the CPU.
2. Remove the RJ45 connection cover from the Ethernet port. Retain the cover for reuse.
3. Plug the Ethernet cable into the Ethernet port on top of the CPU as shown below.
4. Connect the Ethernet cable to the programming device.



(1) Ethernet port

2.1.1.3 Setting up communication with the CPU

From STEP 7-Micro/WIN SMART, use one of the following methods to display the "Communications" dialog for configuring communication to the CPU.

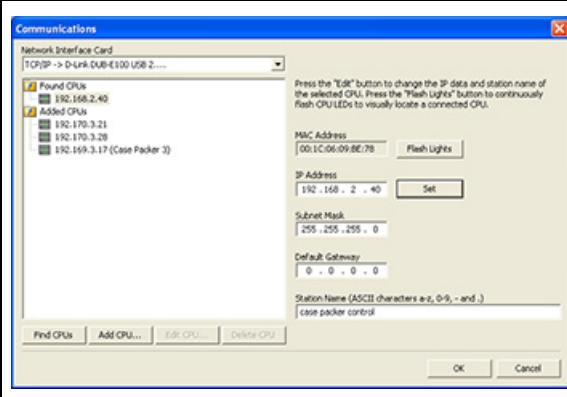
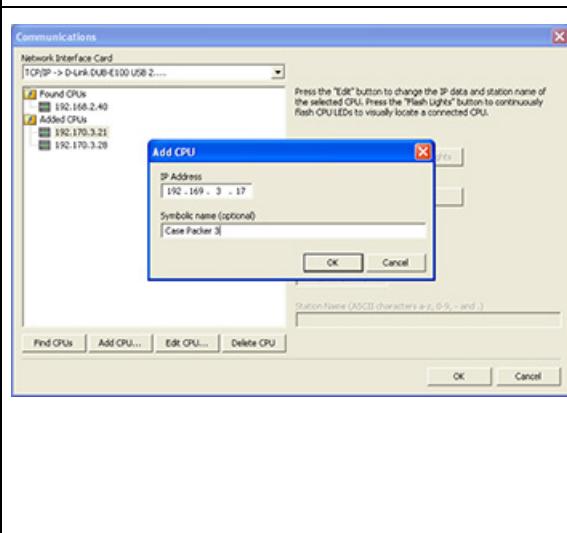
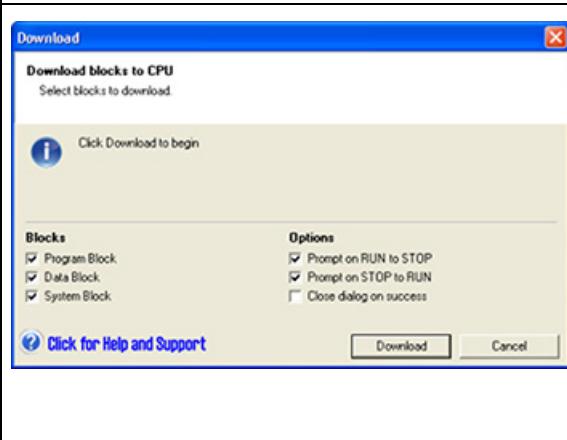
- From the project tree, double-click the "Communications" node.
- Click the "Communications" button  from the navigation bar.
- Select "Communications" from the Component drop-down list in the Windows area of the View menu ribbon strip.

The "Communication" dialog provides two methods of selecting the CPU to be accessed:

- Click the "Find CPUs" button to have STEP 7-Micro/WIN SMART search your local network for CPUs. The IP address of each CPU found on the network is listed under "Found CPUs".
- Click the "Add CPU ..." button to manually enter the access information (IP address and so forth) for a CPU that you wish to access. The IP address for each CPU, manually added with this method, is listed under "Added CPUs" and is retained.

Getting started

2.1 Connecting to the CPU

	<p>For "Found CPUs" (CPUs located on your local network), use the "Communications dialog" to connect with your CPU:</p> <ul style="list-style-type: none">• Select TCP/IP for your network interface card.• Click the "Find CPUs" button to display all operational CPUs ("Found CPUs") on the local Ethernet network. All CPUs have a default IP address. See the Note below.• Highlight a CPU, and then click "OK".
	<p>For "Added CPUs" (CPUs on the local or remote networks), use the "Communications dialog" to connect with your CPU:</p> <ul style="list-style-type: none">• Select TCP/IP for your network interface card.• Click the "Add CPU" button to do one of the following:<ul style="list-style-type: none">– Enter the IP address of a CPU that is accessible from the programming device, but is not on the local network.– Enter the IP address of a CPU directly that is on the local network. <p>All CPUs have a default IP address. See the Note below.</p> <ul style="list-style-type: none">• Highlight a CPU, and then click "OK".
	<p>After you have established communication with the CPU, you are ready to create and download the example program.</p> <p>To download all project components, click the Download button from the Transfer area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination CTRL+D.</p>  <p>If STEP 7-Micro/WIN SMART does not find your CPU, check the settings for the communications parameters and repeat these steps.</p>

Note

The CPU list will show all of the CPUs regardless of Ethernet network class and subnet.

To make a connection to your CPU, your network interface card (NIC) and the CPU must be on the same class of network and on the same subnet. You can either set up your network interface card to match the default IP address of the CPU, or you can change the IP address of the CPU to match the network class and subnet of your network interface card.

See the "Configuring or changing an IP address for a CPU or device in your project" for information about how to accomplish this.

2.2

Creating the sample program

Entering this example of a control program will help you understand how easy it is to use STEP 7-Micro/WIN SMART. This program uses six instructions in three networks to create a very simple, self-starting timer that resets itself.

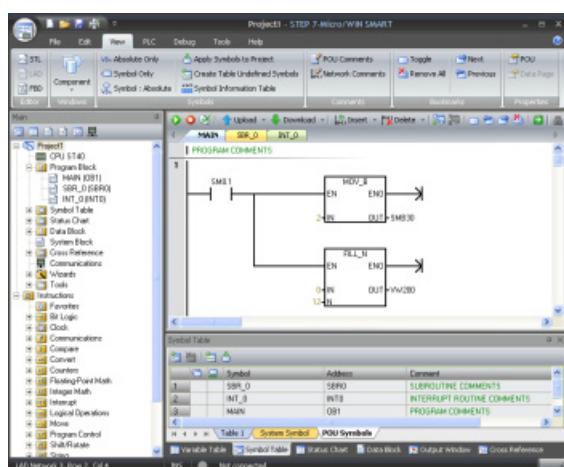
For this example, you use the Ladder (LAD) editor to enter the instructions for the program. The following example shows the complete program in both LAD and Statement List (STL). The description column explains the logic for each network. The timing diagram shows the operation of the program. There are no network comments in the STL program.

Getting started

2.2 Creating the sample program

Table 2- 1 Sample program for getting started with STEP 7-Micro/WIN SMART

LAD/FBD	STL	Description
	Network 1 LDN M0.0 TON T33, +100	10 ms timer T33 times out after (100 x 10 ms = 1 s) M0.0 pulse is too fast to monitor with Status view.
	Network 2 LDW>= T33, +40 = M10.0	Comparison becomes true at a rate that is visible with Status view. Turn on M10.0 after (40 x 10 ms = 0.4 s) for a 40% OFF / 60% ON waveform.
	Network 3 LD T33 = M0.0	T33 (bit) pulse is too fast to monitor with Status view. Reset the timer through M0.0 after the (100 x 10 ms = 1 s) period.
<p>The timing diagram illustrates the logic levels for the sample program. It shows five signals over time:</p> <ul style="list-style-type: none"> ① T33 (current): A sawtooth wave starting at 100 and decreasing to 40. ② current = 100: A constant high level. ③ current = 40: A constant low level. ④ T33 (bit) M0.0: A square wave that toggles between 0 and 1 every 0.6 seconds. ⑤ M10.0: A square wave that toggles between 0 and 1 every 0.4 seconds. <p>The diagram indicates a 0.4s period for the M10.0 waveform, which corresponds to the 40% duty cycle mentioned in the description.</p>		Timing diagram: <ul style="list-style-type: none"> ① T33 (current) ② Current = 100 ③ Current = 40 ④ T33 (bit) and M0.0 ⑤ M10.0



Notice the project tree and the program editor. You use the project tree to insert instructions into the networks of the program editor by dragging and dropping the instructions from the "Instructions" portion of the Project tree to the networks.

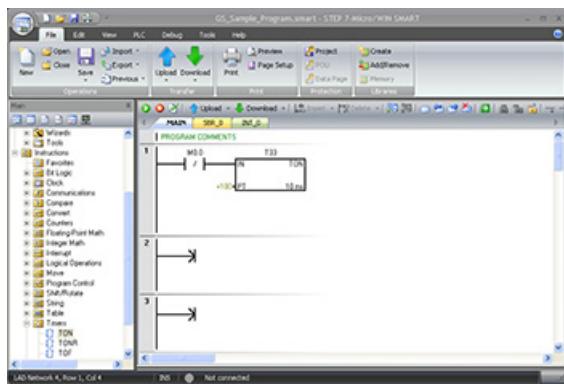
The Program Block folder in the project tree contains all of the blocks of your program.

The program editor toolbar icons provide shortcuts to PLC commands and programming operation.

After you enter and save the program, you can download the program to the CPU.

2.2.1 Network 1: Starting the timer

Network 1: Starting the timer



When M0.0 is off (0), this contact turns on and provides power flow to start the timer.

To enter the contact for M0.0:

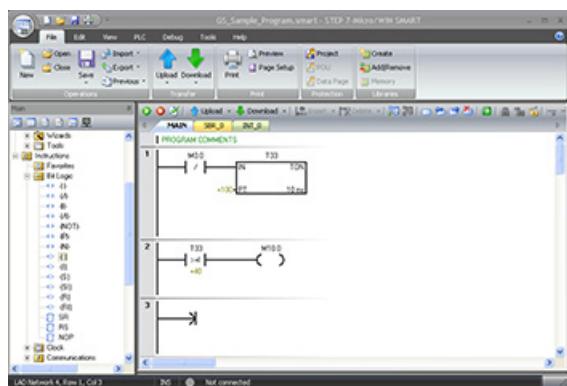
1. Either double-click the "Bit Logic" icon or click the plus sign (+) to display the bit logic instructions.
2. Select the "Normally Closed" contact.
3. Hold down the left mouse button and drag the contact onto the first network.
4. Enter the following address for the contact: M0.0
5. Press the Return key to enter the address for the contact.

To enter the timer instruction for T33:

1. Double-click the "Timers" icon to display the timer instructions.
2. Select the "TON" (on-delay timer) instruction.
3. Hold down the left mouse button and drag the timer onto the first network.
4. Enter the following timer number for the timer: T33
5. Press the Return key to enter the timer number and to move the focus to the preset time (PT) parameter.
6. Enter the following value for the preset time: +100.
7. Press the Return key to enter the value.

2.2.2 Network 2: Turning the output on

Network 2: Turning the output on



When the timer value for T33 is greater than or equal to 40 (40 times 10 milliseconds, or 0.4 seconds), the contact provides power flow to turn on output M10.0 of the CPU.

To enter the Compare instruction:

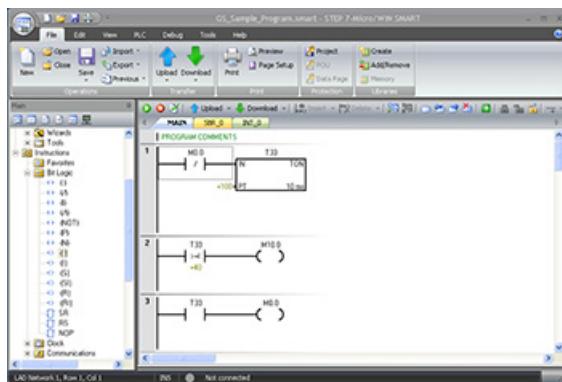
1. Double-click the Compare icon to display the compare instructions. Select the ">=I" instruction (greater-than-or-equal-to-integer).
2. Hold down the left mouse button and drag the compare instruction onto the second network.
3. Click "????" above the contact and enter the address for the timer value: T33
4. Press the Return key to enter the timer number and to move the focus to the other value to be compared with the timer value.
5. Enter the following value to be compared with the timer value: +40
6. Press the Return key to enter the value.

To enter the instruction for turning on output M10.0:

1. Double-click the Bit Logic icon to display the bit logic instructions and select the output coil.
2. Hold down the left mouse button and drag the coil onto the second network.
3. Click "????" above the coil and enter the following address: M10.0
4. Press the Return key to enter the address for the coil.

2.2.3 Network 3: Resetting the timer

Network 3: Resetting the timer



When the timer reaches the preset value (100) and turns the timer bit on, the contact for T33 turns on. Power flow from this contact turns on the M0.0 memory location. Because the timer is enabled by a Normally Closed contact for M0.0, changing the state of M0.0 from off (0) to on (1) resets the timer.

To enter the contact for the timer bit of T33:

1. Select the "Normally Open" contact from the bit logic instructions.
 2. Hold down the left mouse button and drag the contact onto the third network.
 3. Click "???" above the contact and enter the address of the timer bit: T33
 4. Press the Return key to enter the address for the contact.

To enter the coil for turning on M0.0:

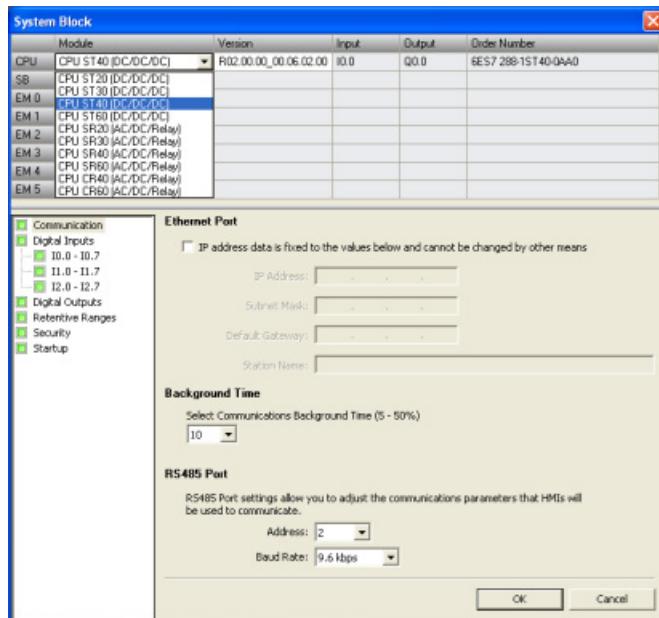
1. Select the output coil from the bit logic instructions.
 2. Hold down the left mouse button and drag the output coil onto the third network.
 3. Click "????" above the coil and enter the following address: M0.0
 4. Press the Return key to enter the address for the coil

2.2.4 Setting the CPU type and version for your project

Configure your project for the CPU and version matching your physical CPU. If the project is not configured for the correct CPU and CPU version, then the download could fail or the program may not run.

To select your CPU, click the "CPU" field under the "Module" column to display the dropdown list button, and select your CPU from the dropdown list. Using the same procedure, select your CPU version in the "Version" column.

2.2 Creating the sample program



2.2.5 Saving the sample project

Saving the sample project

After entering the three networks of instructions, you have finished entering the program. When you save the program, you create a project that includes the CPU type and other parameters. To save the project in a file name and location that you specify:

1. Click the down arrow under the Save button from the Operations area of the File menu ribbon strip to display the Save As button.



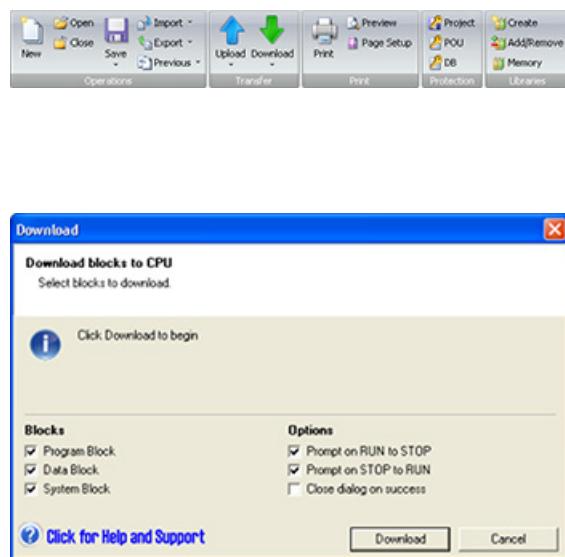
2. Click the Save As button and provide a filename for saving your project.



3. Enter a name for the project in the "Save As" dialog.
4. Browse to a location where you want to save your project.
5. Click "Save" to save the project.

After saving the project, you can download the program to the CPU.

2.3 Downloading the sample program



To download all project components, click the "Download" button from the "Transfer" area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination "CTRL+D".

Click the Download dialog "Download" button.

STEP 7-Micro/WIN SMART copies the complete program or program components that you selected to the CPU.

If your CPU is in RUN mode, a dialog prompts you to place the CPU in STOP mode. Clicking "Yes" sets the CPU to STOP mode.

Note

Each project is associated with a CPU type. If the project type does not match the CPU to which you are connected, STEP 7-Micro/WIN SMART indicates a mismatch and prompts you to take an action.

2.4 Changing the operating mode of the CPU

The CPU has two modes of operation: STOP mode and RUN mode. The status LEDs on the front of the CPU indicates the current mode of operation. In STOP mode, the CPU is not executing the program, and you can download program blocks. In RUN mode, the CPU is executing the program; however, you can download program blocks.

Placing the CPU in RUN mode

1. Click the "RUN" button on either the PLC menu ribbon strip or on the program editor toolbar:
2. When prompted, click "OK" to change the operating mode of the CPU.

You can monitor the program in STEP 7-Micro/WIN SMART by clicking the "Program Status" button from the "Debug" menu ribbon strip, or from the program editor toolbar. STEP 7-Micro/WIN SMART displays the values for the instructions.

Placing the CPU in STOP mode

To stop the program, click the "STOP" button  and acknowledge the prompt to place the CPU in STOP mode. You can also place a STOP instruction in your program logic to put the CPU in STOP mode.

Installation

3.1 Guidelines for installing S7-200 SMART devices

The S7-200 SMART equipment is designed to be easy to install. You can install the S7-200 SMART either on a panel or on a standard DIN rail, and you can orient the S7-200 SMART either horizontally or vertically. The small size of the S7-200 SMART allows you to make efficient use of space.



WARNING

Safety requirements for installing S7-200 SMART PLCs

S7-200 SMART PLCs are Open Type Controllers. You must install the PLC in a housing, cabinet, or electric control room. Limit entry to the housing, cabinet, or electric control room to authorized personnel.

Failure to follow these installation requirements could result in death or serious injury to personnel, and/or damage to equipment.

Always follow these requirements when installing the PLC.

Separate the devices from heat, high voltage, and electrical noise

As a general rule for laying out the devices of your system, always separate the devices that generate high voltage and high electrical noise from the low-voltage, logic-type devices such as the PLC.

When configuring the layout of the PLC inside your panel, consider the heat-generating devices and locate the electronic-type devices in the cooler areas of your cabinet. Reducing the exposure to a high-temperature environment will extend the operating life of any electronic device.

Consider also the routing of the wiring for the devices in the panel. Avoid placing low-voltage signal wires and communications cables in the same tray with AC power wiring and high-energy, rapidly-switched DC wiring.

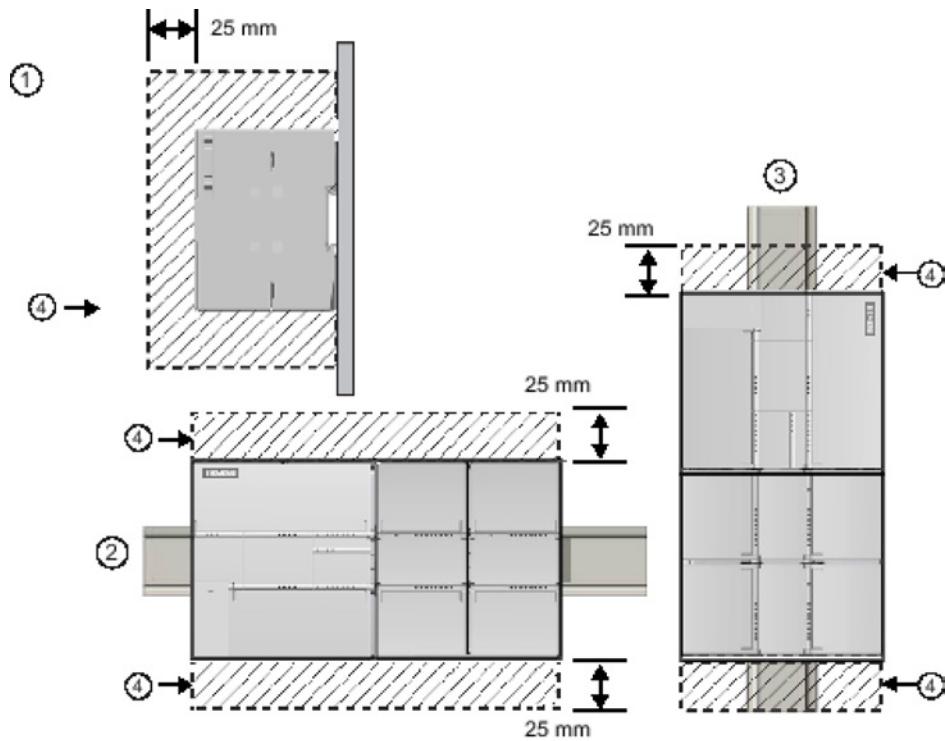
*3.1 Guidelines for installing S7-200 SMART devices***Provide adequate clearance for cooling and wiring**

S7-200 SMART devices are designed for natural convection cooling. For proper cooling, you must provide a clearance of at least 25 mm above and below the devices. Also, allow at least 25 mm of depth between the front of the modules and the inside of the enclosure.

**Temperature considerations**

Vertical mounting reduces the maximum allowable ambient temperature by 10 degrees C. Operating outside the maximum temperature range could result in erratic process operation and could result in minor personal injury.

Mount the CPU below any expansion modules, as shown in the following figure. Follow the prescribed guidelines for mounting modules to ensure proper cooling.



① Side view

③ Vertical installation

② Horizontal installation

④ Clearance area

When planning your layout for the PLC, allow enough clearance for the wiring and communications cable connections.

3.2 Power budget

Your CPU has an internal power supply that provides power for the CPU, the expansion modules, signal boards, and other 24 VDC user power requirements. Use the following information as a guide for determining how much power (or current) the CPU can provide for your configuration.

Refer to the technical specifications for your particular CPU to determine the 24 VDC sensor supply power budget, the 5 VDC logic budget supplied by your CPU and the 5 VDC power requirements of the expansion modules and signal boards. Refer to the Calculating a power budget (Page 583) to determine how much power (or current) the CPU can provide for your configuration.

The CPU provides the 5 VDC logic power needed for any expansion in your system. Pay careful attention to your system configuration to ensure that the CPU can supply the 5 VDC power required by your selected expansion modules. If your configuration requires more power than the CPU can supply, you must remove a module.

Note

If the CPU power budget is exceeded, you may not be able to connect the maximum number of modules allowed for your CPU.

The CPU also provides a 24 VDC sensor supply that can supply 24 VDC for input points, for relay coil power on the expansion modules, or for other requirements. If your power requirements exceed the budget of the sensor supply, then you must add an external 24 VDC power supply to your system. You must manually connect the 24 VDC supply to the input points or relay coils.

If you require an external 24 VDC power supply, ensure that the power supply is not connected in parallel with the sensor supply of the CPU. For improved electrical noise protection, it is recommended that the commons (M) of the different power supplies be connected.

 WARNING
Connecting power supplies safely
Connecting an external 24 VDC power supply in parallel with the 24 VDC sensor supply of the CPU can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level.
The result of this conflict can be shortened lifetime or immediate failure of one or both power supplies, with consequent unpredictable operation of the PLC system. Unpredictable operation could result in death or serious injury to personnel, and/or damage to equipment.
The DC sensor supply of the CPU and any external power supply should provide power to different points. A single connection of the commons is allowed.

3.3 Installation and removal procedures

Some of the 24 VDC power input ports in the S7-200 SMART system are interconnected, with a common logic circuit connecting multiple M terminals. For example, the following circuits are interconnected when designated as "not isolated" in the data sheets: the 24 VDC power supply of the CPU, the power input for the relay coil of an EM, or the power supply for a non-isolated analog input. All non-isolated M terminals must connect to the same external reference potential.



Avoiding unwanted current flow

Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and any connected equipment.

Failure to comply with these guidelines could cause damage or unpredictable operation which could result in death or severe personal injury and/or property damage.

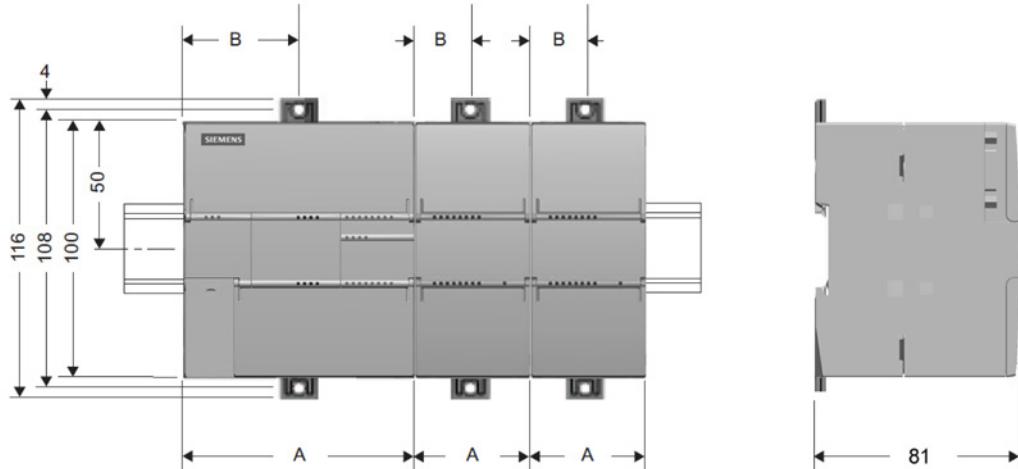
Always ensure that all non-isolated M terminals in an S7-200 SMART system are connected to the same reference potential.

Refer to the technical specifications for your particular CPU to determine the 24 VDC sensor supply power budget, the 5 VDC logic budget supplied by your CPU and the 5 VDC power requirements of the expansion modules and signal boards.

3.3 Installation and removal procedures

3.3.1 Mounting dimensions for the S7-200 SMART devices

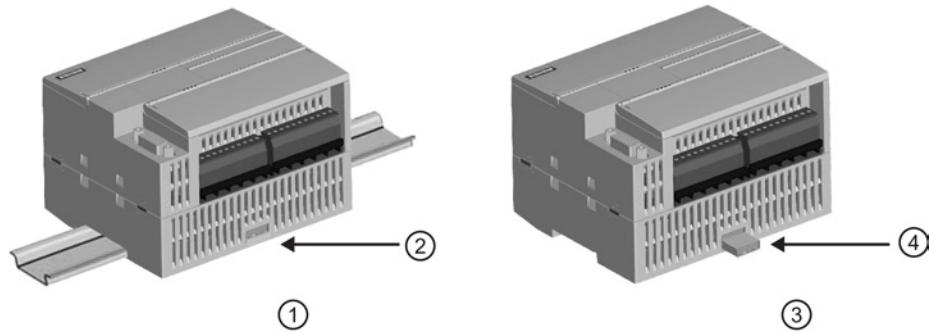
The CPU and expansion modules include mounting holes to facilitate installation on panels.



S7-200 SMART module		Width A (mm)	Width B (mm)
CPU SR20 and CPU ST20		90	45
CPU SR30 and CPU ST30		110	55
CPU CR40, CPU SR40, and CPU ST40		125	62.5
CPU CR60, CPU SR60 and CPU ST60		175	87.5
Expansion modules:	EM 4AI, EM 2AQ, EM 8DI, EM 8DQ, and EM 8DQRLY	45	22.5
	EM 8DI/8DQ and EM 8DI/8DQRLY	45	22.5
	EM 16DI/16DQ and EM 16DI/16DQRLY	70	35
	EM 4AI/2AQ	45	22.5
	EM 2RTD	45	22.5
	EM 4TC	45	22.5

3.3.2 Installing and removing the CPU

The CPU can be easily installed on a standard DIN rail or on a panel. DIN rail clips are provided to secure the device on the DIN rail. The clips also snap into an extended position to provide a screw mounting position for panel-mounting the unit.



- | | | | |
|---|-----------------------------------|---|---------------------------|
| ① | DIN rail installation | ③ | Panel installation |
| ② | DIN rail clip in latched position | ④ | Clip in extended position |

Figure 3-1 Installation on a DIN rail or on a panel

3.3 Installation and removal procedures

Before you install or remove any electrical device, ensure that the power to that equipment has been turned off. Also, ensure that the power to any related equipment has been turned off.

**WARNING****Remove power to PLC before installing or removing equipment**

Attempts to install or remove the PLC or related equipment with the power applied could cause electric shock or faulty operation of equipment.

Failure to disable all power to the PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

Always follow appropriate safety precautions and ensure that power to the PLC is disabled before attempting to install or remove the CPU or related equipment.

Always ensure that whenever you replace or install a device, you use the correct module or equivalent device.

**WARNING****Module replacement**

If you install an incorrect module, the program in the CPU could function unpredictably.

Failure to replace a device with the same model, orientation, or order could result in death or serious injury to personnel, and/or damage to equipment.

Replace the device with the same model, and be sure to orient and position it correctly.

Note

Install expansion modules separately after the CPU has been installed.

Consider the following when installing the units on the DIN rail or on a panel:

- For DIN rail mounting, make sure the upper DIN rail clip is in the latched (inner) position and that the lower DIN rail clip is in the extended position for the CPU.
- After installing the devices on the DIN rail, move the lower DIN rail clips to the latched position to lock the devices on the DIN rail.
- For panel mounting, make sure the DIN rail clips are pushed to the extended position.

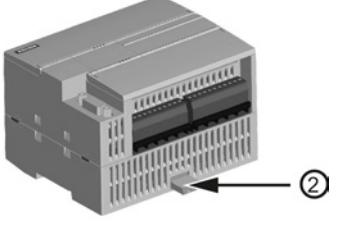
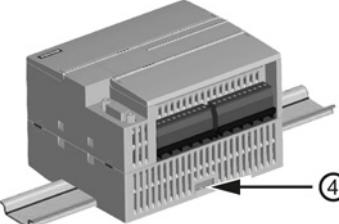
To install the CPU on a panel, follow these steps:

1. Locate, drill, and tap the mounting holes (M4 or American Standard number 8), using the dimensions in the table, Mounting dimensions (mm) (Page 38).
2. Ensure that the CPU and S7-200 SMART equipment are disconnected from electrical power.
3. Secure the module(s) to the panel, using a Pan Head M4 screw with spring and flat washer. Do not use a flat head screw.
4. If you are using an expansion module, put it next to the CPU and slide together until the connectors join securely.

Note

The type of screw will be determined by the material upon which it is mounted. You should apply appropriate torque until the spring washer becomes flat. Avoid applying excessive torque to the mounting screws. Do not use a flat head screw.

Table 3- 1 Installing a CPU on a DIN rail

Task	Procedure
 	<p>Follow the steps below to install a CPU on a DIN rail.</p> <ol style="list-style-type: none"> 1. Secure the rail to the mounting panel every 75 mm. 2. Snap open the DIN clip (located on the bottom of the module) and hook the back of the module onto the DIN rail. 3. Rotate the module down to the DIN rail and snap the clip closed. Carefully check that the clip has fastened the module securely onto the rail. To avoid damage to the module, press on the tab of the mounting hole instead of pressing directly on the front of the module.

Note

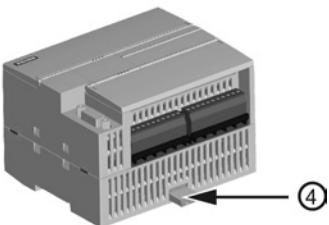
Using DIN rail stops could be helpful if your CPU is in an environment with high vibration potential or if the CPU has been installed vertically. Use an end bracket (8WA1 808 or 8WA1 805) on the DIN rail to ensure that the modules remain connected.

If your system is in a high-vibration environment, then panel-mounting the CPU will provide a greater level of vibration protection.

Installation

3.3 Installation and removal procedures

Table 3- 2 Removing a CPU from a DIN rail

Task	Procedure
	<p>Follow the steps below to remove a CPU from a DIN rail.</p> <ol style="list-style-type: none"> 1. Remove power from the CPU and any attached I/O modules. 2. Disconnect all the wiring and cabling that is attached to the CPU. The CPU and most expansion modules have removable connectors to make this job easier. 3. Unscrew the mounting screws or snap open the DIN clip. 4. If you have expansion modules connected, slide the CPU to the left to disengage it from the expansion module connector. Note: unscrewing or unsnapping the DIN clips of the expansion modules can make it easier to disengage the CPU. 5. Remove the CPU.

3.3.3 Installing and removing an expansion module

Table 3- 3 Installing an expansion module

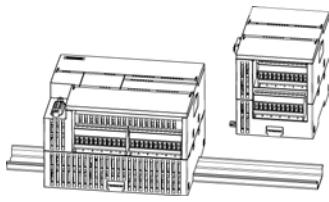
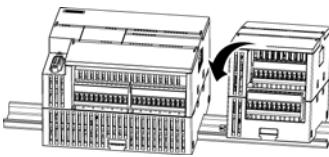
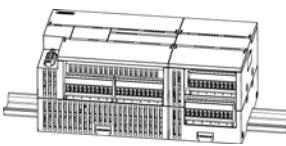
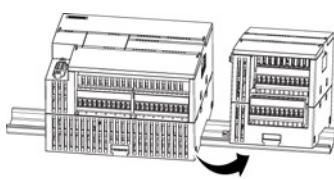
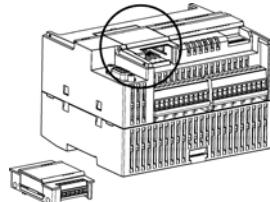
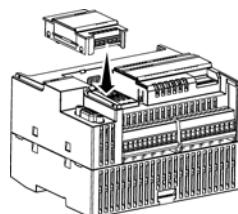
Task	Procedure
	<p>Follow the steps below to install an expansion module:</p> <ol style="list-style-type: none"> 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the cover for the I/O bus connector from the right side of the CPU. 3. Insert a screwdriver into the slot above the cover. 4. Gently pry the cover out at its top and remove the cover. Retain the cover for reuse.
	<p>Connect the expansion module to the CPU.</p> <ol style="list-style-type: none"> 1. Pull out the bottom DIN rail clip to allow the expansion module to fit over the rail. 2. Position the expansion module to the right of the CPU. 3. Hook the expansion module over the top of the DIN rail. 4. Slide the expansion module to the left until the I/O connector fully engages the connector on the right of the CPU and push the bottom clip in to latch the expansion module onto the rail.

Table 3- 4 Removing an expansion module

Task	Procedure
 	<p>Follow the steps below to remove an expansion module:</p> <ol style="list-style-type: none"> 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the I/O connectors and wiring from the expansion module. 3. Loosen the DIN rail clips of all the S7-200 SMART devices. 4. Physically slide the expansion module to the right.

3.3.4 Installing and removing a signal board or battery board

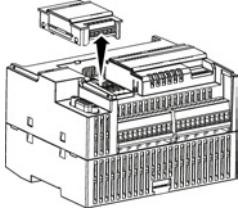
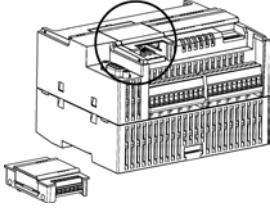
Table 3- 5 Installing a signal board

Task	Procedure
 	<p>Follow the steps below to install a signal board or battery board</p> <ol style="list-style-type: none"> 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Remove the top and bottom terminal block covers from the CPU. 3. Place a screwdriver into the slot on top of the CPU at the rear of the cover. 4. Gently pry the cover up and remove it from the CPU. 5. Place the signal board or battery board straight down into its mounting position in the top of the CPU. 6. Firmly press the module into position until it snaps into place. 7. Replace the terminal block covers.

Installation

3.3 Installation and removal procedures

Table 3- 6 Removing a signal board or battery board

Task	Procedure
	<p>Follow the steps below to remove a signal board or battery board</p> <ol style="list-style-type: none">1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power.2. Remove the top and bottom terminal block covers from the CPU.3. Place a screwdriver into the slot on top of the module.4. Gently pry the module up to disengage it from the CPU.5. Remove the module straight up from its mounting position in the top of the CPU.6. Replace the cover onto the CPU.7. Replace the terminal block covers.
	

Installing or replacing the battery in the SB BA01 battery board

The SB BA01 battery board requires battery type CR1025. The battery is not included with the SB BA01 and must be purchased.

To install the battery, follow these steps:

1. In the SB BA01, install the new battery with the positive side of the battery on top, and the negative side next to the printed wiring board.
2. The SB BA01 is now ready to be installed in the CPU. Follow the installation directions above.

To replace the battery, follow these steps:

1. Remove the SB BA01 from the CPU following the removal directions above.
2. Carefully remove the old battery using a small screwdriver. Push the battery out from under the clip.
3. Install a new CR1025 replacement battery with the positive side of the battery on top and the negative side next to the printed wiring board.
4. Re-install the SB BA01 battery board following the installation directions above.

3.3.5 Removing and reinstalling the terminal block connector

The S7-200 SMART modules have removable connectors to make connecting the wiring easy.

Table 3- 7 Removing the connector

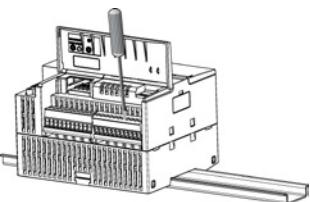
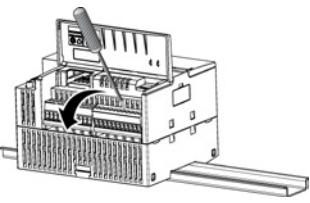
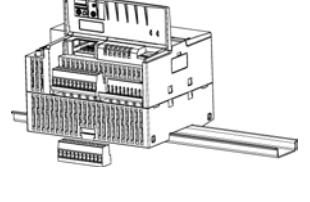
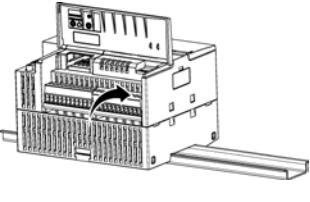
Task	Procedure
	<p>Prepare the system for terminal block removal by removing the power from the CPU and opening the cover above the connector.</p> <ol style="list-style-type: none"> 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Inspect the top of the connector and locate the slot for the tip of the screwdriver. 3. Insert a small screwdriver into the slot. 4. Gently pry the top of the connector away from the CPU. The connector will release with a snap. 5. Grasp the connector and remove it from the CPU.
	

Table 3- 8 Installing the connector

Task	Procedure
	<p>Prepare the components for terminal block installation by removing power from the CPU and opening the cover above the connector.</p> <ol style="list-style-type: none"> 1. Ensure that the CPU and all S7-200 SMART equipment are disconnected from electrical power. 2. Align the connector with the pins on the unit. 3. Align the wiring edge of the connector inside the rim of the connector base. 4. Press firmly down and rotate the connector until it snaps into place. <p>Check carefully to ensure that the connector is properly aligned and fully engaged.</p>
	

3.4 Wiring guidelines

Proper grounding and wiring of all electrical equipment is important to help ensure the optimum operation of your system and to provide additional electrical noise protection for your application and the PLC. Refer to the technical specifications (Page 509) for the wiring diagrams.

Prerequisites

Before you ground or install wiring to any electrical device, ensure that the power to that equipment has been turned off. Also, ensure that the power to any related equipment has been turned off.

Ensure that you follow all applicable electrical codes when wiring the PLC and related equipment. Install and operate all equipment according to all applicable national and local standards. Contact your local authorities to determine which codes and standards apply to your specific case.

WARNING

Attempts to install or wire the PLC or related equipment with power applied could cause electric shock or faulty operation of equipment. Failure to disable all power to the PLC and related equipment during installation or removal procedures could result in death or serious injury to personnel, and/or damage to equipment.

Always follow appropriate safety precautions and ensure that power to the PLC is disabled before attempting to install or remove the PLC or related equipment.

Always take safety into consideration as you design the grounding and wiring of your PLC system. Electronic control devices, such as the PLC, can fail and can cause unexpected operation of the equipment that is being controlled or monitored. For this reason, you should implement safeguards that are independent of the PLC to protect against possible personal injury or equipment damage.

WARNING

Control devices can fail in an unsafe condition, resulting in unexpected operation of controlled equipment. Such unexpected operations could result in death or serious injury to personnel, and/or damage to equipment.

Use an emergency stop function, electromechanical overrides, or other redundant safeguards that are independent of the PLC.

Isolation guidelines

The AC power supply boundaries and I/O boundaries to AC circuits have been designed and approved to provide safe separation between AC line voltages and low voltage circuits.

These boundaries include double or reinforced insulation, or basic plus supplementary insulation, according to various standards. Components which cross these boundaries such as optical couplers, capacitors, transformers, and relays have been approved as providing safe separation. Isolation boundaries which meet these requirements have been identified in the product data sheets as having 1500 VAC or greater isolation. This designation is based on a routine factory test of $(2U_e + 1000)$ VAC or equivalent according to approved methods. Safe separation boundaries have been type tested to 4242 VDC.

The sensor supply output, communications circuits, and internal logic circuits of the CPU with included AC power supply are sourced as SELV (safety extra-low voltage) according to EN 61131-2. These circuits become PELV (protective extra-low voltage) if the sensor supply M, or any other non-isolated M connection to the CPU is connected to ground. Other M connections which may ground reference the low voltage are designated as not isolated to logic on specific product data sheets. Examples are RS485 communications port M, analog I/O M, and relay coil power M.

To maintain the SELV / PELV character of the low voltage circuits, external connections to communications ports, analog circuits, and all 24 V nominal power supply and I/O circuits must be powered from approved sources that meet the requirements of SELV, PELV, Class 2, Limited Voltage, or Limited Power according to various standards.



WARNING

Safe use of power converters

Use of non-isolated or single insulation supplies to supply low voltage circuits from an AC line can result in hazardous voltages appearing on circuits that are expected to be touch safe, such as communications circuits and low voltage sensor wiring.

Such unexpected high voltages could result in death or serious injury to personnel, and/or damage to equipment.

Use only high-voltage-to-low-voltage power converters that are approved as sources of touch-safe, limited-voltage circuits.

Grounding guidelines

The best way to ground your application is to ensure that all the common and ground connections of your PLC and related equipment are grounded to a single point. This single point should be connected directly to the earth ground for your system.

All ground wires should be as short as possible and should use a large wire size, such as 2 mm² (14 AWG).

When locating grounds, remember to consider safety grounding requirements and the proper operation of protective interrupting devices.

Wiring guidelines

When designing the wiring for your PLC, adhere to the following guidelines:

- Provide a single disconnect switch that simultaneously removes power from the CPU power supply, from all input circuits, and from all output circuits.
- Provide overcurrent protection, such as a fuse or circuit breaker, to limit fault currents on supply wiring. Consider providing additional protection by placing a fuse or other current limit in each output circuit.
- Install appropriate surge suppression devices for any wiring that could be subject to lightning surges.
- Avoid placing low-voltage signal wires and communications cables in the same wire tray with AC wires and high-energy, rapidly switched DC wires.
- Always route wires in pairs, with the neutral or common wire paired with the hot or signal-carrying wire.
- Use the shortest wire possible and ensure that the wire is sized properly to carry the required current. The connector accepts wire sizes from 2 mm² to 0.3 mm² (14 AWG to 22 AWG). Use shielded wires for optimum protection against electrical noise. Typically, grounding the shield at the CPU gives the best results.
- When wiring input circuits that are powered by an external power supply, include an overcurrent protection device in that circuit. External protection is not necessary for circuits that are powered by the 24 VDC sensor supply from the CPU because the sensor supply is already current-limited.
- The CPU and expansion modules have removable connectors for user wiring. To prevent loose connections, ensure that the connector is seated securely and that the wire is installed securely into the connector. To avoid damaging the connector, take care to not over-tighten the screws. The maximum torque for the CPU and EM connector screw is 0.56 N·m (5 inch-pounds). The maximum torque for the SB connector screw is 0.33 N·m (3 inch-pounds)

Isolation boundaries

To help prevent unwanted current flows in your installation, the CPU provides isolation boundaries at certain points. When you plan the wiring for your system, you should consider these isolation boundaries. Refer to the technical specifications (Page 509) for the amount of isolation provided and the location of the isolation boundaries. Isolation boundaries rated less than 1500 VAC must not be depended on as safety boundaries.

Guidelines for lamp loads

Lamp loads are damaging to relay contacts because of the high turn-on surge current. This surge current will nominally be 10 to 15 times the steady state current for a tungsten lamp. A replaceable interposing relay or surge limiter is recommended for lamp loads that will be switched a large number of times during the lifetime of the application.

Guidelines for inductive loads

You should equip inductive loads with suppression circuits to limit voltage rise when the control output turns off. Suppression circuits protect your outputs from premature failure due to the high voltages associated with turning off inductive loads. In addition, suppression circuits limit the electrical noise generated when switching inductive loads. Placing an external suppression circuit so that it is electrically across the load, and physically located near the load is most effective in reducing electrical noise.

The DC (transistor) outputs include internal suppression circuits that are adequate for the inductive loads in most applications. Since the relay output contacts can be used to switch either a DC or an AC load, internal protection is not provided.

Note

The effectiveness of a given suppression circuit depends on the application, and you must verify it for your particular use. Always ensure that all components used in your suppression circuit are rated for use in the application.

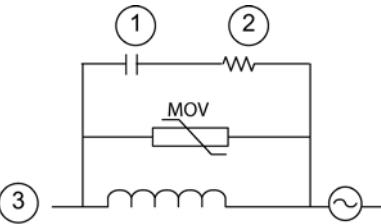
Table 3- 9 Typical suppressor circuit for DC or relay outputs that switch DC inductive loads

DC inductive load	Diagram		
In most applications, the addition of a diode ① across a DC inductive load is suitable, but if your application requires faster turn-off times, then the addition of a Zener diode ② is recommended. Be sure to size your Zener diode properly for the amount of current in your output circuit.		①	1N4001 diode or equivalent
		②	8.2 V Zener (DC outputs), 36 V Zener (Relay outputs)
		③	Output point

Installation

3.4 Wiring guidelines

Table 3- 10 Typical suppressor circuit for relay outputs that switch AC inductive loads

AC inductive load	Diagram		
When you use a relay output to switch 115 V/230 VAC loads, place the appropriately rated resistor-capacitor-metal oxide varistor (MOV) circuit across the AC load. Ensure that the working voltage of the MOV is at least 20% greater than the nominal line voltage.		①	0.1 μ F



WARNING

Correct placement of external resistor/capacitor noise suppression circuit

When relay expansion modules are used to switch AC inductive loads, you must place the external resistor/capacitor noise suppression circuit across the AC load to prevent unexpected machine or process operation. Unexpected machine or process operation could result in death or severe personal injury.

Always be sure to follow these guidelines in placing the external resistor/capacitor noise suppression circuit.

4

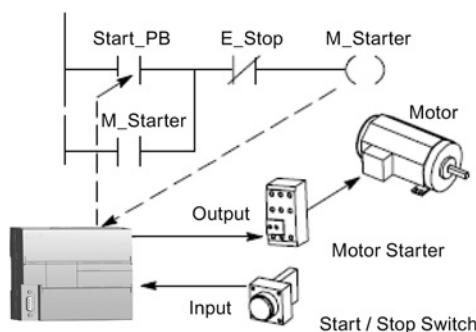
PLC concepts

The basic function of the CPU is to monitor field inputs and, based on your control logic, turn on or off field output devices. This chapter explains the concepts used to execute your program, the various types of memory used, and how that memory is retained.

4.1 Execution of the control logic

The CPU continuously cycles through the control logic in your program, reading and writing data. The basic operation is very simple:

- The CPU reads the status of the inputs.
- The program that is stored in the CPU uses these inputs to evaluate the control logic.
- As the program runs, the CPU updates the data.
- The CPU writes the data to the outputs.



The figure shows a simple diagram of how an electrical relay diagram relates to the CPU. In this example, the state of the switch for starting the motor is combined with the states of other inputs. The calculations of these states then determine the state for the output that goes to the actuator which starts the motor.

Tasks in a scan cycle

The CPU executes a series of tasks repetitively. This cyclical execution of tasks is called the scan cycle. The execution of the user program is dependent upon whether the CPU is in STOP mode or in RUN mode. In RUN mode, your program is executed; in STOP mode, your program is not executed.

Table 4- 1 Tasks performed by the CPU in a scan cycle

Scan cycle	Description
<p>The diagram illustrates the PLC scan cycle. It shows a central vertical stack of five boxes representing the CPU's internal tasks. From top to bottom: 1. 'Writes to the outputs' (an arrow points from the stack to a motor icon), 2. 'Performs the CPU diagnostics', 3. 'Processes any communications requests', 4. 'Executes the program' (an arrow points from the stack to two sensor icons), and 5. 'Reads the inputs' (an arrow points from the stack to the sensors). To the right of the stack is a circular arrow labeled 'Scan cycle'. To the left is a detailed view of the 'Reads the inputs' task, showing a row of digital input pins and a row of analog input pins.</p>	<ol style="list-style-type: none"> ① Reading the inputs: The CPU copies the state of the physical inputs to the process-image input register. ② Executing the control logic in the program: The CPU executes the instructions of the program and stores the values in the various memory areas. ③ Processing any communications requests: The CPU performs any tasks required for communications. ④ Executing the CPU self-test diagnostics: The CPU ensures that the firmware, the program memory, and any expansion modules are working properly. ⑤ Writing to the outputs: The values stored in the process-image output register are written to the physical outputs.

4.1.1 Reading the inputs and writing to the outputs

Reading the inputs

Digital inputs: Each scan cycle begins by reading the current value of the digital inputs and then writing these values to the process-image input register.

Analog inputs: The CPU does not read the analog input values as part of the normal scan cycle. Instead, an analog value is read immediately from the device when your program accesses the analog input.

Writing to the outputs

Digital outputs: At the end of every scan cycle, the CPU writes the values stored in the process-image output register to the digital outputs.

Analog outputs: The CPU does not write analog output values as part of the normal scan cycle. Instead, the analog outputs are written immediately when your program accesses the analog output.

4.1.2 Immediately reading or writing the I/O

The CPU instruction set provides instructions that immediately read from or write to the physical I/O. These immediate I/O instructions allow direct access to the actual input or output point, even though the image registers are normally used as either the source or the destination for I/O accesses. The corresponding process-image input register location is not modified when you use an immediate instruction to access an input point. The corresponding process-image output register location is updated simultaneously when you use an immediate instruction to access an output point.

Note

When you read an analog input, the value is read immediately. When you write a value to an analog output, the output is updated immediately.

It is usually advantageous to use the process-image register rather than to directly access inputs or outputs during the execution of your program. There are three reasons for using the image registers:

- The sampling of all inputs at the start of the scan synchronizes and freezes the values of the inputs for the program execution phase of the scan cycle. The outputs are updated from the image register after the execution of the program is complete. This provides a stabilizing effect on the system.
- Your program can access the image register much more quickly than it can access I/O points, allowing faster execution of the program.
- I/O points are bit entities and must be accessed as bits or bytes, but you can access the image register as bits, bytes, words, or double words. Thus, the image registers provide additional flexibility.

4.1.3 Executing the user program

During the execution phase of the scan cycle, the CPU executes your main program, starting with the first instruction and proceeding to the last instruction. The immediate I/O instructions give you immediate access to inputs and outputs during the execution of either the main program or an interrupt routine.

If you use subroutines in your program, the subroutines are stored as part of the program. The subroutines are executed when they are called by the main program, by another subroutine, or by an interrupt routine. Subroutine nesting depth is 8 levels deep from the main and 4 levels deep from an interrupt routine.

If you use interrupts in your program, the interrupt routines that are associated with the interrupt events are stored as part of the program. The interrupt routines are not executed as part of the normal scan cycle, but are executed when the interrupt event occurs (which could be at any point in the scan cycle).

4.1 Execution of the control logic

Local memory is reserved for each of 14 entities: the main program, eight subroutine nesting levels when initiated from the main program, one interrupt routine, and four subroutine nesting levels when initiated from an interrupt routine. Local memory has a local scope in that it is available only within its associated program entity, and cannot be accessed by the other program entities. For more information about Local memory, refer to Local Memory Area: L in this chapter.

The following figure depicts the flow of a typical scan including the Local memory usage and two interrupt events, one during the program-execution phase and another during the communications phase of the scan cycle. Subroutines are called by the next higher level, and are executed when called. Interrupt routines are not called; they are a result of an occurrence of the associated interrupt event.

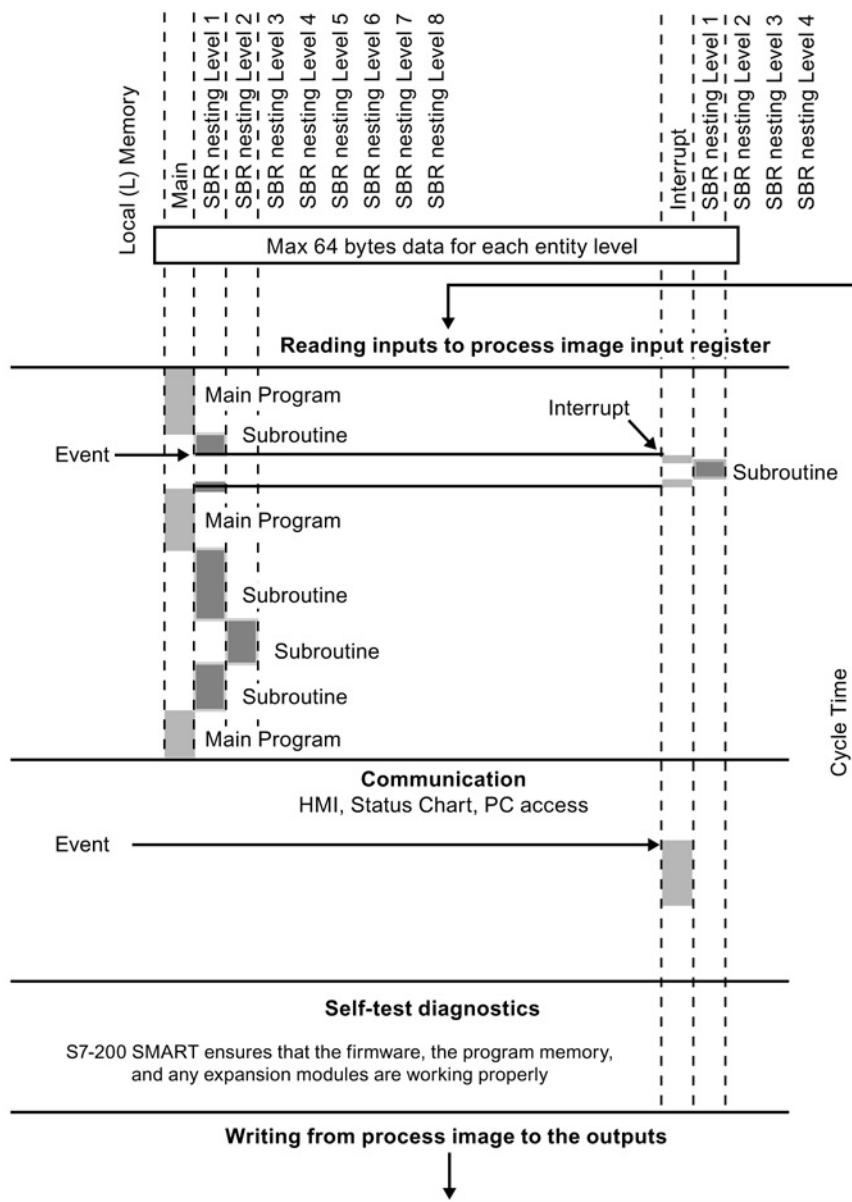
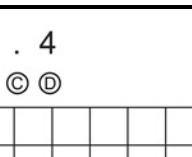


Figure 4-1 Typical scan flow

4.2 Accessing data

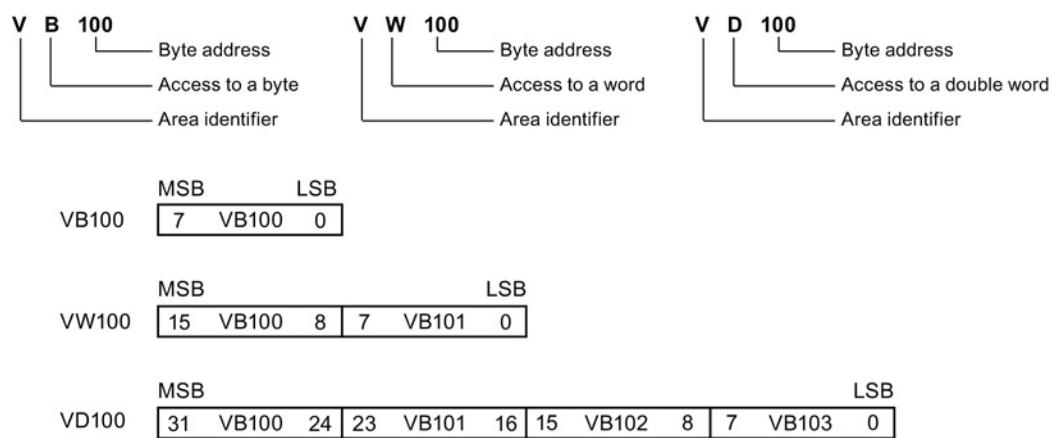
The CPU stores information in different memory locations that have unique addresses. You can explicitly identify the memory address that you want to access. This allows your program to have direct access to the information. To access a bit in a memory area, you specify the address, which includes the memory area identifier, the byte address, and the bit number (which is also called "byte.bit" addressing).

Table 4- 2 Bit addressing

Elements of a bit address		Description	
 M 3 . 4 (A) (B) (C) (D)	0 1 2 3 (E) 4 5 7 6 5 4 3 2 1 0 (F)	A	Memory area identifier
		B	Byte address: byte 3
		C	Separator ("byte.bit")
		D	Bit location of the byte (bit 4 of 8, bits numbered 7 to 0)
		E	Bytes of the memory area
		F	Bits of the selected byte

In this example, the memory area and byte address ("M3") designates byte 3 of M memory, with a period (".") to separate the bit address (bit 4).

You can access data in most memory areas (V, I, Q, M, S, L, and SM) as bytes, words, or double words by using the byte-address format. To access a byte, word, or double word of data in the memory, you must specify the address in a way similar to specifying the address for a bit. This includes an area identifier, data size designation, and the starting byte address of the byte, word, or double-word value, as shown in the following figure.



The following table shows the range of integer values that can be represented by the different sizes of data.

Table 4- 3 Decimal and hexadecimal ranges for the different sizes of data

Representation	Byte (B)	Word (W)	Double Word (D)
Unsigned Integer	0 to 255 16#00 to 16#FF	0 to 65,535 16#0000 to 16#FFFF	0 to 4,294,967,295 16#00000000 to 16#FFFFFF
Signed Integer	-128 to +127 16#80 to 16#7F	-32,768 to +32,767 16#8000 to 16#7FFF	-2,147,483,648 to +2,147,483,647 16#8000 0000 to 16#7FFF FFFF
Real (IEEE 32-bit Floating Point)	<i>Not applicable</i>	<i>Not applicable</i>	+1.175495E-38 to +3.402823E+38 (positive) -1.175495E-38 to -3.402823E+38 (negative)

Data in other memory areas (such as T, C, HC, and the accumulators) are accessed by using an address format that includes an area identifier and a device number.

4.2.1 Accessing memory areas

I (process-image input)

The CPU samples the physical input points at the beginning of each scan cycle and writes these values to the process-image input register. You can access the process-image input register in bits, bytes, words, or double words:

Table 4- 4 Absolute addressing for I memory

Bit:	I[byte address].[bit address]	I0.1
Byte, Word, or Double Word:	I[size][starting byte address]	IB4, IW7, ID20

Q (process-image output)

At the end of the scan cycle, the CPU copies the values stored in the process-image output register to the physical output points. You can access the process-image output register in bits, bytes, words, or double words:

Table 4- 5 Absolute addressing for Q memory

Bit:	Q[byte address].[bit address]	Q1.1
Byte, Word, or Double Word:	Q[size][starting byte address]	QB5, QW14, QD28

V (variable memory)

You can use V memory to store intermediate results of operations being performed by the control logic in your program. You can also use V memory to store other data pertaining to your process or task. You can access the V memory area in bits, bytes, words, or double words:

Table 4- 6 Absolute addressing for V memory

Bit:	<i>V[byte address].[bit address]</i>	V10.2
Byte, Word, or Double Word:	<i>V[size][starting byte address]</i>	VB16, VW100, VD2136

M (flag memory)

You can use the flag memory area (M memory) as internal control relays to store the intermediate status of an operation or other control information. You can access the flag memory area in bits, bytes, words, or double words:

Table 4- 7 Absolute addressing for M memory

Bit:	<i>M[byte address].[bit address]</i>	M26.7
Byte, Word, or Double Word:	<i>M[size][starting byte address]</i>	MB0, MW11, MD20

T (timer memory)

The CPU provides timers that count increments of time in resolutions (time-base increments) of 1 ms, 10 ms, or 100 ms. Two variables are associated with a timer:

- Current value: this 16-bit signed integer stores the amount of time counted by the timer.
- Timer bit: this bit is set or cleared as a result of comparing the current and the preset value. The preset value is entered as part of the timer instruction.

You access both of these variables by using the timer address (T + timer number). Access to either the timer bit or the current value is dependent on the instruction used: instructions with bit operands access the timer bit, while instructions with word operands access the current value. As shown in the following figure, the Normally Open Contact instruction accesses the timer bit, while the Move Word instruction accesses the current value of the timer.

Table 4- 8 Absolute addressing for T memory

Timer:	<i>T[timer number]</i>	T24
--------	------------------------	-----

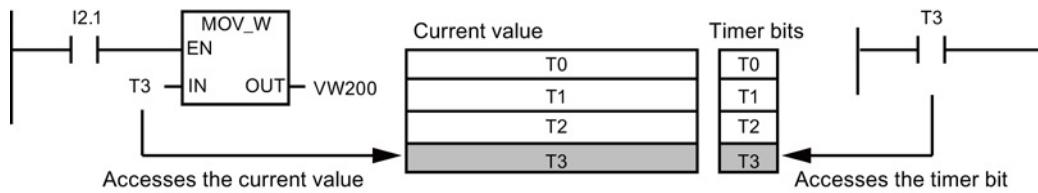


Figure 4-2 Accessing the timer bit or the current value of a timer

C (counter memory)

The CPU provides three types of counters that count each low-to-high transition event on the counter input(s): one type counts up only, one type counts down only, and one type counts both up and down. Two variables are associated with a counter:

- Current value: this 16-bit signed integer stores the accumulated count.
- Counter bit: this bit is set or cleared as a result of comparing the current and the preset value. The preset value is entered as part of the counter instruction.

You access both of these variables by using the counter address (C + counter number). Access to either the counter bit or the current value is dependent on the instruction used: instructions with bit operands access the counter bit, while instructions with word operands access the current value. As shown in the following figure, the Normally Open Contact instruction accesses the counter bit, while the Move Word instruction accesses the current value of the counter.

Table 4- 9 Absolute addressing of C memory

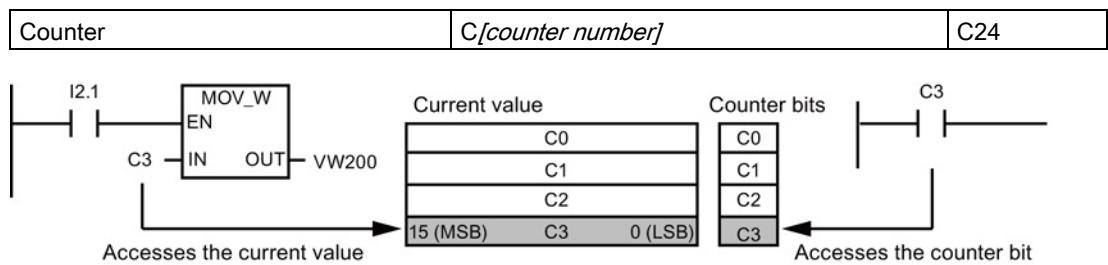


Figure 4-3 Accessing the counter bit or the current value of a counter

HC (high-speed counter)

The high-speed counters count high-speed events independent of the CPU scan. High-speed counters have a signed, 32-bit integer counting value (or current value). To access the count value for the high-speed counter, you specify the address of the high-speed counter, using the memory type (HC) and the counter number. The current value of the high-speed counter is a read-only value and can be addressed only as a double word (32 bits).

Table 4- 10 Absolute addressing of HC memory

High-speed counter	HC/[high-speed counter number]	HC1
--------------------	--------------------------------	-----

AC (accumulators)

The accumulators are read/write devices that can be used like memory. For example, you can use accumulators to pass parameters to and from subroutines and to store intermediate values used in a calculation. The CPU provides four 32-bit accumulators (AC0, AC1, AC2, and AC3). You can access the data in the accumulators as bytes, words, or double words.

The size of the data being accessed is determined by the instruction that is used to access the accumulator. As shown in the following figure, you use the least significant 8 or 16 bits of the value that is stored in the accumulator to access the accumulator as bytes or words. To access the accumulator as a double word, you use all 32 bits.

For information about how to use the accumulators within interrupt subroutines, refer to the Interrupt instructions (Page 263).

Table 4- 11 Absolute addressing of AC memory

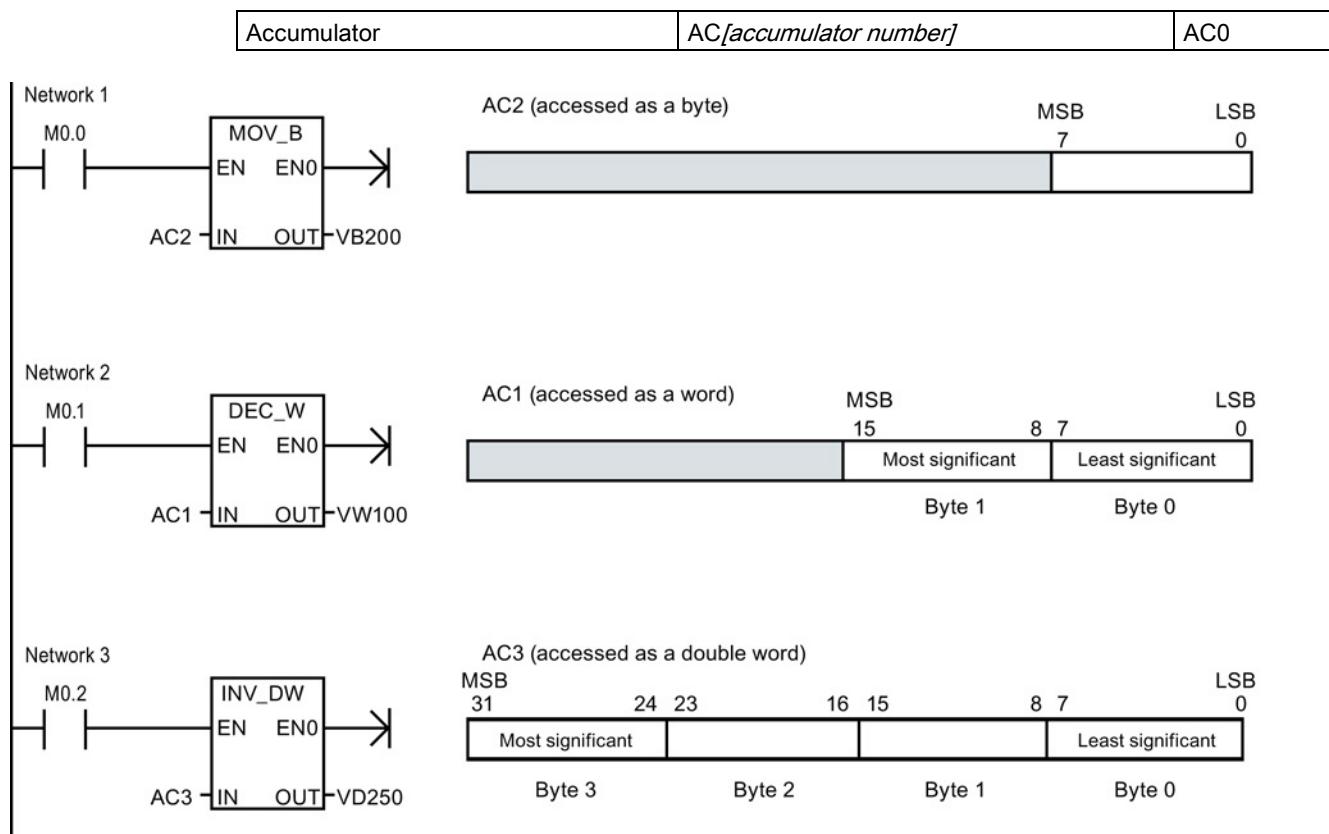


Figure 4-4 Accessing the accumulators

SM (special memory)

The SM bits provide a means for communicating information between the CPU and your user program. You can use these bits to select and control some of the special functions of the CPU, such as: a bit that turns on for the first scan cycle, a bit that toggles at a fixed rate, or a bit that shows the status of math or operational instructions. You can access the SM bits as bits, bytes, words, or double words:

Table 4- 12 Absolute addressing of SM memory

Bit:	<i>SM[/byte address].[bit address]</i>	SM0.1
Byte, Word, or Double Word:	<i>SM[size][starting byte address]</i>	SMB86, SMW300, SMD1000

For more information, see the descriptions of the SM bits (Page 593).

L (local memory area)

The CPU provides 64 L memory bytes for each POU (program organizational unit) in a local memory stack. A POU's associated L memory addresses are accessible only by the currently executing POU (main, subroutine, or interrupt routine). When you use interrupt routines and subroutines, the L memory stack is used to preserve L memory values of a POU that temporarily suspends execution, so another POU can execute. The suspended POU can then resume execution with the L memory values that existed prior to giving execution control to another POU.

L memory stack maximum nesting limits:

- Eight subroutine nesting levels when initiated from the main program
- Four subroutine nesting levels when initiated from an interrupt routine

The nesting limits allow a 14 level execution stack in your program. For example, the main program (level 1) has eight nested subroutines (levels 2 to 9). During execution of the 9th level subroutine, an interrupt occurs (level 10). The interrupt routine contains four nested subroutines (levels 11 to 14).

L memory rules:

- You can use L memory for local scratchpad "TEMP" variables in all POU types (main, subroutine, and interrupt routines)
- Only subroutines can use L memory for "IN" IN_OUT", and "OUT" variable types that are passed to or from subroutines.
- If you are programming a subroutine in either LAD or FBD, only 60 bytes are allowed for TEMP, IN, IN_OUT, and OUT variables. STEP 7-Micro/WIN SMART uses the last four bytes of local memory

Local memory symbols, variable types, and data types are assigned in the Variable table that is available when the associated POU is opened in the program editor. Absolute L memory addresses are automatically assigned when a POU is successfully compiled.

In most cases, use L memory symbol name references in your program logic, because you cannot know all the absolute L memory addresses until after the complete POU is successfully compiled. However, you can use absolute L memory addresses as shown in the following table.

Table 4- 13 Absolute addressing of L memory

Bit:	<i>L[/byte address].[bit address]</i>	L0.0
Byte, Word, or Double Word:	<i>L[/size] [starting byte address]</i>	LB33, LW5, LD20

Local memory and to global V memory use a similar address syntax, but V memory has a global scope while L memory has a local scope. Global scope means that the same memory address can be accessed from any POU. Local scope means that the L memory allocation is associated with a particular POU and cannot be accessed by another program unit.

The local scope of L memory also affects symbol usage, when a global symbol and a local symbol use the same name. If your program logic references that symbol name, the CPU ignores the global symbol and processes the address assigned to the local memory symbol.

Note

Local memory value assignments are not always preserved for successive executions of a POU

L memory addresses are reused for the next execution sequence, after the current nested sequence is completed. Depending on a POU's level in the execution stack and L memory assignments made since a POU's last execution, a POU's L memory assignments made in a previous execution may be overwritten with unexpected values.

Remember to reassign the correct values to L memory variables, in your program logic. Reinitialize all TEMP values before processing them and ensure that any output values (OUT and IN_OUT) are correct.

AI (analog input)

The CPU converts an analog value (such as temperature or voltage) into a word-length (16-bit) digital value. You access these values by the area identifier (AI), size of the data (W), and the starting byte address. Since analog inputs are words and always start on even-number bytes (such as 0, 2, or 4), you access them with even-number byte addresses (such as AIW0, AIW2, or AIW4). Analog input values are read-only values.

Table 4- 14 Absolute addressing of AI memory

Analog input	<i>AIW[starting byte address]</i>	AIW4
--------------	-----------------------------------	------

AQ (analog output)

The CPU converts a word-length (16-bit) digital value into a current or voltage, proportional to the digital value (such as for a current or voltage). You write these values by the area identifier (AQ), size of the data (W), and the starting byte address. Since analog outputs are words and always start on even-number bytes (such as 0, 2, or 4), you write them with even-number byte addresses (such as AQW0, AQW2, or AQW4). Analog output values are write-only values.

Table 4- 15 Absolute addressing of AQ memory

Analog output	AQW/[starting byte address]	AQW4
---------------	-----------------------------	------

S (sequence control relay)

S bits are associated with SCRs, which you can use to organize machine or steps into equivalent program segments. SCRs allow logical segmentation of the control program. You can access the S memory as bits, bytes, words, or double words.

Table 4- 16 Absolute addressing of S memory

Bit:	S/[byte address].[bit address]	S3.1
Byte, Word, or Double Word:	S/[size]/[starting byte address]	SB4, SW7, SD14

4.2.2**Format for Real numbers**

Real (or floating-point) numbers are represented as 32-bit, single-precision numbers, whose format is described in the ANSI/IEEE 754-1985 standard. Real numbers are accessed in double-word lengths.

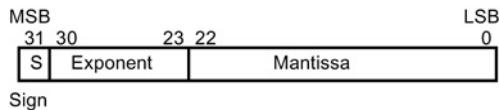


Figure 4-5 Format of a Real number

Note

Floating-point numbers are accurate up to 6 decimal places. Therefore, you can specify a maximum of 6 decimal places when entering a floating-point constant.

Calculations that involve a long series of values including very large and very small numbers can produce inaccurate results. This can occur if the numbers differ by 10 to the power of x , where $x > 6$. For example: $100\ 000\ 000 + 1 = 100\ 000\ 000$

4.2.3 Format for strings

A string is a sequence of characters, with each character being stored as a byte. The first byte of the string defines the length of the string, which is the number of characters. The following figure shows the format for a string. A string can have a length of 0 to 254 characters, plus the length byte, so the maximum length for a string is 255 bytes. A string constant is limited to 126 bytes.

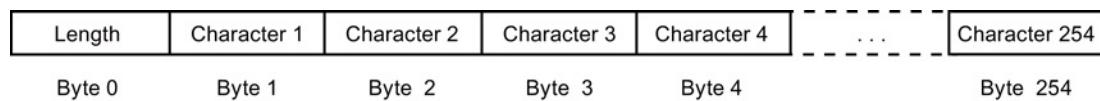


Figure 4-6 Format for strings

4.2.4 Assigning a constant value for instructions

You can use a constant value in many of the programming instructions. Constants can be bytes, words, or double words. The CPU stores all constants as binary numbers, which can then be represented in decimal, hexadecimal, ASCII, or real number (floating point) formats.

Table 4- 17 Representation of constant values

Representation	Format	Sample
Decimal	[decimal value]	20047
Hexadecimal	16#[hexadecimal value]	16#4E4F
Binary	2#[binary number]	2#1010_0101_1010_0101
ASCII	'[ASCII text]'	'ABCD'
Real	ANSI/IEEE 754-1985	+1.175495E-38 (positive) -1.175495E-38 (negative)
String	"[stringtext]"	"ABCDE"

Note

The CPU does not support "data typing" or data checking (such as specifying that the constant is stored as an integer, a signed integer, or a double integer). For example, an Add instruction can use the value in VW100 as a signed integer value, while an Exclusive Or instruction can use the same value in VW100 as an unsigned binary value.

4.2.5 Addressing the local and expansion I/O

The local I/O provided by the CPU provides a fixed set of I/O addresses. You can add I/O points by connecting expansion I/O modules to the right side of the CPU or by installing a signal board. The addresses of the points of the module are determined by the type of I/O and the position of the module in the chain. For example, an output module does not affect the addresses of the points on an input module, and vice versa. Likewise, analog modules do not affect the addressing of digital modules, and vice versa.

Note

Process-image register space for digital I/O is always reserved in increments of eight bits (one byte). If a module does not provide a physical point for each bit of each reserved byte, these unused bits cannot be assigned to subsequent modules in the I/O chain. For input modules, the unused bits are set to zero with each input update cycle.

Analog I/O points are always allocated in increments of two points. If a module does not provide physical I/O for each of these points, these I/O points are lost and are not available for assignment to subsequent modules in the I/O chain.

The following table provides an example of the fixed mapping convention (established by STEP 7 Micro/WIN SMART and downloaded as part of the I/O configuration, in the system block).

Table 4- 18 CPU mapping convention

	CPU	Signal board	Signal module 0	Signal module 1	Signal module 2	Signal module 3	Signal module 4	Signal module 5
Starting address	I0.0 Q0.0	I7.0 Q7.0 No AI SB AQ12	I8.0 Q8.0 AI16 AQ16	I12.0 Q12.0 AI32 AQ32	I16.0 Q16.0 AI48 AQ48	I20.0 Q20.0 AI64 AQ64	I24.0 Q24.0 AI80 AQ80	I28.0 Q28.0 AI96 AQ96

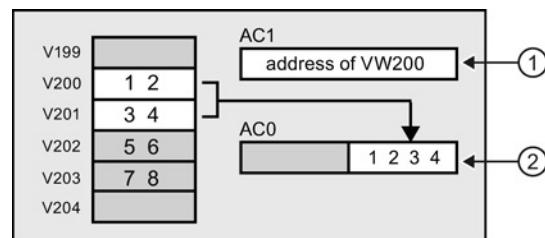
4.2.6 Using pointers for indirect addressing

Indirect addressing uses a pointer to access data in memory. Pointers are double word memory locations that contain the address of another memory location. You can only use V memory locations, L memory locations, or accumulator registers (AC1, AC2, AC3) as pointers. To create a pointer, you must use the Move Double Word instruction to move the address of the indirectly addressed memory location to the pointer location. Pointers can also be passed to a subroutine as a parameter.

An S7-200 SMART CPU allows pointers to access the following memory areas: I, Q, V, M, S, AI, AQ, SM, T (current value only), and C (current value only). You cannot use indirect addressing to access an individual bit or to access HC, L or accumulator memory areas.

To indirectly access the data in a memory address, you create a pointer to that location by entering an ampersand character (&) and the first byte of the memory location to be addressed. The input operand of the instruction must be preceded with an ampersand (&) to signify that the address of a memory location, instead of its contents, is to be moved into the location identified in the output operand of the instruction (the pointer).

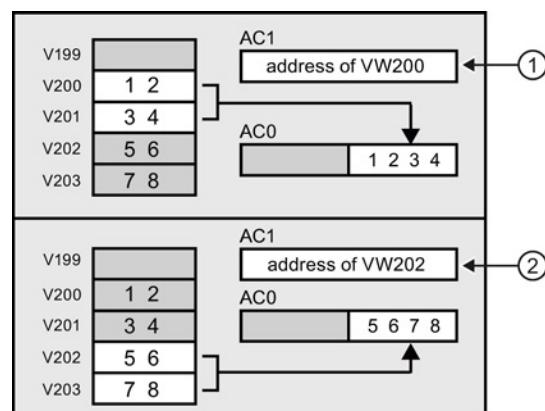
Entering an asterisk (*) in front of an operand for an instruction specifies that the operand is a pointer. As shown in the following figure, entering *AC1 means that AC1 stores a pointer to the word-length value being referenced by the Move Word (MOVW) instruction. In this example, the values stored in both VB200 and VB201 are moved to accumulator AC0.



- ① MOVD &VB200, AC1
Creates the pointer by moving the address of VB200 (initial byte of VW200) to AC1
- ② MOVW *AC1, AC0
Moves the word value referenced by the pointer in AC1

Figure 4-7 Creating and using a pointer

As shown in the following figure, you can change the value of a pointer. Since pointers are 32-bit values, use double-word instructions to modify pointer values. Simple mathematical operations, such as adding or incrementing, can be used to modify pointer values.



- ① MOVD &VB200, AC1
Creates the pointer by moving the address of VB200 (initial byte of VW200) to AC1
MOVW *AC1, AC0
Moves the word value referenced by the pointer in AC1
- ② +D +2, AC1
Adds 2 to the accumulator to point to the next word location
MOVW *AC1, AC0
Moves the word value referenced by the pointer in AC1

Figure 4-8 Modifying a pointer

Note

When modifying the value of a pointer, remember to adjust for the size of the data that you are accessing: to access a byte, increment the pointer value by 1; to access a word or a current value for a timer or counter, add or increment the pointer value by 2; and to access a double word, add or increment the pointer value by 4.

4.2.7 Pointer examples

Using a pointer to access data in a table

This example uses LD14 as a pointer to a recipe stored in a table of recipes that begins at VB100. In this example, VW1008 stores the index to a specific recipe in the table. If each recipe in the table is 50 bytes long, you multiply the index by 50 to obtain the offset for the starting address of a specific recipe. By adding the offset to the pointer, you can access the individual recipe from the table. In this example, the recipe is copied to the 50 bytes that start at VB1500.

Table 4- 19 Example: Using a pointer to access data in a table

LAD	STL
<pre> LD SM0.0 AND &VB100 MOVD &VB100, LD14 ITD VW1008, LD18 *MUL DI LD LD18 LD +50 LD LD18 ADD DI LD LD14 BLKMOV B LD *LD14 LD 50 LD VB1500 </pre>	<p>To transfer a recipe from a table of recipes:</p> <ul style="list-style-type: none"> • Each recipe is 50 bytes long. • The index parameter (VW1008) identifies the recipe to be loaded. <p>Create a pointer to the starting address of the recipe table.</p> <p>Convert the index of the recipe to a double-word value.</p> <p>Multiply the offset to accommodate the size of each recipe.</p> <p>Add the adjusted offset to the pointer.</p> <p>Transfer the selected recipe to VB1500 through VB1549</p>

Using an offset to access data

This example uses LD10 as a pointer to the address VB0. You then increment the pointer by an offset stored in VD1004. LD10 then points to another address in V memory (VB0 + offset). The value stored in the V memory address pointed to by LD10 is then copied to VB1900. By changing the value in VD1004, you can access any V memory location.

Table 4- 20 Example: Using an offset to read the value of any V memory location

LAD	STL
<pre> LD SM0.0 MOV_DW &VB0 IN LD10 EN OUT ADD_DI VD1004 IN1 LD10 IN2 EN OUT MOV_B *LD10 IN VB1900 EN OUT </pre>	<p>Load the starting address of the V memory to a pointer.</p> <p>Add the offset value to the pointer.</p> <p>Copy the value from the V memory location (offset) to VB1900</p> <pre> Network 1 LD SM0.0 MOVD &VB0, LD10 +D VD1004, LD10 MOVB *LD10, VB1900 </pre>

4.3 Saving and restoring data

4.3.1 Downloading project components

Note

Downloading a program block, data block, or system block to the CPU completely overwrites any pre-existing contents of that block in the CPU. Be sure that you want to overwrite the block before performing a download.

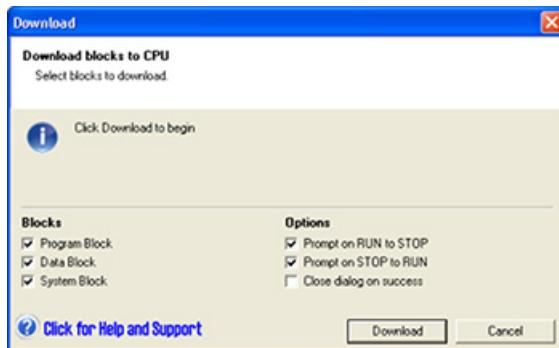
To download project components from STEP 7-Micro/WIN SMART to the CPU, follow these steps:

1. Ensure that your network hardware and PLC connector cable are working (Page 25), and that PLC communication is operating properly (Page 421).
2. Place the CPU in STOP mode (Page 33).

3. To download all project components, click the Download button from the Transfer area of the File or PLC menu ribbon strip, or alternatively press the shortcut key combination **CTRL+D**.



4. To download selected project components, click the down arrow under the Download button, and then select the specific project component you want to download (Program Block, Data Block, or System Block) from the drop-down list.
5. After clicking the Download button, if you see a Communications dialog, select the Network Interface Card and the IP address for the PLC to which you want to download.
6. From the Download dialog, set the download options for the blocks, and whether you want to be prompted on CPU transitions from RUN to STOP mode (Page 33) and STOP to RUN mode (Page 33).
7. Optionally click the "Close dialog on success" check box if you want the dialog to automatically close after a successful download.
8. Click the Download button.



STEP 7-Micro/WIN SMART copies the complete program or program components that you selected to the CPU. The status icon indicates informational messages, or whether potential problems or errors occurred with the download. The status message provides specific results of the operation.

Note

Project components originally created for use in an S7-200 SMART CPU with firmware version V1.x can be downloaded to a CPU with firmware version V2.0 or later. However, project components originally created for use in a CPU firmware version V2.0 or later may not successfully download into a CPU with firmware version V1.x, especially if the project components use functionality not supported in firmware version V1.x.

What happens when you download

STEP 7-Micro/WIN SMART and the CPU perform the following tasks in sequence on your project components when you download:

Step	Action	Related topics, additional description
1.	Project components in the program editors serve as input for the download operation, based on the download objects you selected. The program editors can include new program data that you've entered, a saved and opened .smart project, or an uploaded ASCII import file.	File open Range checking Project file I/O errors Program editor errors
2.	STEP 7-Micro/WIN SMART compile A compile or download command starts the compiler. If the compile passes, control passes to the next step; if not, the compile or download operation exits.	All STEP 7-Micro/WIN SMART compiler errors are listed in the Output Window. Double-click the error and the editor scrolls to the error location. A successful compile shows the resulting block size of the program and data block.
3.	Send blocks to CPU across communication network for PLC compile.	Communication Errors To download (Editor to PLC) or upload (PLC to Editor), PLC communication must be operating properly. Make sure your network hardware and PLC connector cable are working.
4.	PLC compile If PLC compile succeeds, control passes to the next step; if not, download exits with error(s).	The PLC Compiler verifies that the PLC hardware supports all program instructions, ranges, and structure. Click the PLC button from the Information area of the PLC menu to view the first compile error found.
5.	Program is in CPU permanent memory and ready to be executed in RUN mode.	Fatal Errors (Page 590) and non-fatal run-time errors (Page 587) are accessible from the Information area of the PLC menu.

If the download attempt produces compiler errors or download errors, correct the errors and reattempt the download.

See also

Uploading project components (Page 71)

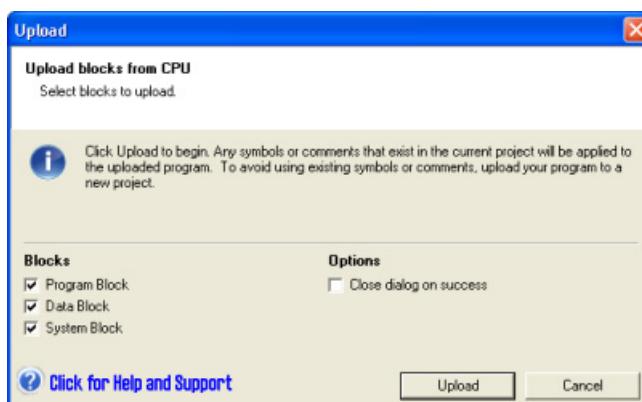
4.3.2 Uploading project components

To upload project components from the PLC to a STEP 7-Micro/WIN SMART program editor, follow these steps:

1. Ensure that your network hardware and PLC connector cable are working (Page 25), and that PLC communication is operating properly (Page 421).
2. To upload all project components, click the Upload button from the Transfer section of the File or PLC menu ribbon strip, or press the shortcut key combination **CTRL+U**.



3. To upload selected project components, click the down arrow under the Upload button, and then select the specific project component you want to upload (Program Block, Data Block, or System Block).
4. If you see a Communications dialog, select the Network Interface Card and the IP address of the PLC from which you want to upload.
5. From the Upload dialog, you can change your selection for which blocks to upload if you choose.
6. Optionally click the "Close dialog on success" check box if you want the dialog to automatically close after a successful upload.
7. Click the "Upload" button to start the upload.



STEP 7-Micro/WIN SMART copies the complete program or program components that you selected for uploading from the PLC to the currently open project. The status icon indicates informational messages, or whether potential problems or errors occurred with the upload. The status message provides specific results of the operation.

4.3 Saving and restoring data

If the upload is successful, you can save the uploaded program, or make further changes. The PLC does not contain symbol or status chart information; hence, you cannot upload a symbol table or status chart.

Note

Uploading into a new project is a risk-free way to capture the program block, system block, and/or data block information. Since the project is empty, you cannot inadvertently destroy data. If you want to make use of material from a status chart or symbol table that are in another project, you can always open a second instance of STEP 7-Micro/WIN SMART and copy that information in from the other project file (Page 86).

Uploading into an existing project is useful if you want to overwrite all modifications that have been made to the program since it was downloaded (Page 68) to the PLC. Uploading into an existing project does, however, overwrite any additions or modifications you have made to the project. Use this option, only if you want to completely overwrite your STEP 7-Micro/WIN SMART project with the project stored in the PLC.

STEP 7-Micro/WIN SMART does not upload comments, but if you currently have a program with comments open in the program editor, the comments are retained. Take care if uploading over an existing project and use this method only if the projects are similar.

4.3.3 Types of storage

The CPU provides a variety of features to ensure that your user program and data are properly retained.

- Retentive memory: selectable areas of memory that remain unchanged over a power cycle. Retentive memory can be configured in the system data block. V, M, and current values to timers and counters are the only memory areas that can be configured to be retentive.
- Permanent memory: memory used to store the program block, data block, system block, forced values, as well as values configured to be retentive.
- Memory card: removable microSDHC card used to store the projects blocks as a program transfer card (Page 75), to completely erase the PLC as a restore-to-factory-defaults card (Page 137), or to update the PLC and expansion module firmware as a firmware update card (Page 73).

4.3.4 Using a memory card

Using a memory card

The S7-200 SMART CPUs support the use of a microSDHC card for:

- User program transfer (Page 75)
- Reset CPU to factory default condition (Page 137)
- Firmware update of the CPU and attached expansion modules as supported

You can use any standard, commercial microSDHC card with a capacity in the range 4GB to 16GB.



The following CPU behaviors are common, regardless of the memory card usage:

1. Inserting a memory card into a CPU in RUN mode causes the CPU to automatically transition to STOP mode.
2. A CPU cannot advance to RUN mode if a memory card is inserted.
3. Memory card evaluation is performed only after a CPU power-up or warm restart. Therefore, program transfer and firmware update can only occur after a CPU power-up or warm restart.
4. The memory card can be used to store files and folders not related to program transfer and firmware update usage as long as their names do not conflict with the file and folder names used for program transfer and firmware update usage.

WARNING

Verify that the CPU is not actively running a process before installing the memory card.

Installing the memory card will cause the CPU to go to STOP mode, which could affect the operation of an online process or machine. Unexpected operation of a process or machine could result in death or injury to personnel and/or property damage.

Before inserting the memory card, always ensure that the CPU is offline and in a safe state.

Program transfer card

A memory card can be used to transfer user program content into the CPU permanent memory, completely or partially replacing content already in the load memory.

To be used for program transfer purposes, the memory card is organized as follows:

Table 4- 21 Memory card used for program transfer card

At the root level of the card	
File: S7_JOB.S7S	A text file containing the word TO_ILM
Folder: SIMATIC.S7S	A folder containing user program files to be transferred to the CPU

Reset to factory defaults card

A memory card can be used to erase all retained data, putting the CPU back into a factory default condition.

To be used for reset to factory default purposes, the memory card is organized as follows:

At the root of the card	
File: S7_JOB.S7S	A text file containing the word RESET_TO_FACTORY

Firmware update card

A memory card can be used to update the firmware in a CPU and any connected expansion modules.

To be used for firmware update purposes, the memory card is organized as follows:

Table 4- 22 Memory card used for firmware update purposes

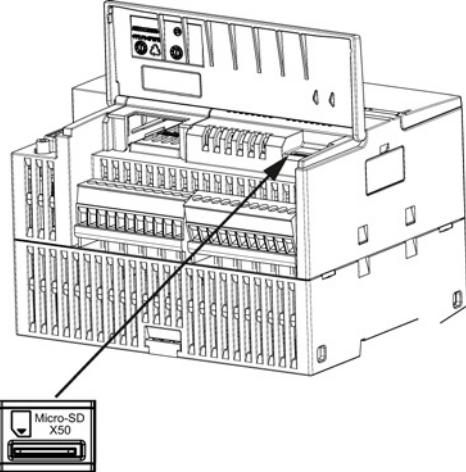
At the root level of the card	
File: S7_JOB.S7S	A text file containing the word FWUPDATE
Folder: FWUPDATE.S7S	A folder containing update files (.upd) for each device to be updated

After power-up, if the CPU detects the presence of a memory card, then it locates and opens the S7_JOB.SYS file on the card. If the FWUPDATE string is discovered in that file, then the CPU enters a firmware update sequence.

The CPU examines each update file (.upd) in the FWUPDATE.S7S folder and if the order ID contained in the update file name matches the order ID (MLFB) of a connected device (CPU, expansion module or signal board), then the firmware of that device is replaced with the firmware content contained within the update file.

4.3.5 Inserting a memory card in the CPU

Table 4- 23 Inserting and removing a memory card in the CPU

Task	Procedure
	<p>Follow the steps below to insert the microSDHC memory card into the CPU.</p> <ol style="list-style-type: none"> 1. Open the bottom terminal block connector cover. 2. Insert the microSDHC memory card in the memory card slot (marked X50) located above the terminal block connectors. 3. Replace the terminal block connector cover after inserting the card to ensure that the card is secure. <p>Follow the steps below to remove the microSDHC memory card from the CPU.</p> <ol style="list-style-type: none"> 1. Open the bottom terminal block connector cover. 2. Grasp the microSDHC memory card from the CPU and pull it out of the card slot (marked Micro-SD X50). 3. Replace the bottom terminal block cover.

4.3.6 Transferring your program with a memory card

S7-200 SMART CPUs support standard, commercial microSDHC cards with a capacity ranging from 4 GB to 16 GB using the FAT32 file system format. You can use a microSDHC card as a program transfer card for portable storage for your program and project data.

WARNING Verify that the CPU is not running a process before inserting the memory card. Inserting a memory card into a CPU in RUN mode causes the CPU to automatically transition to STOP mode. Inserting a memory card into a running CPU can cause disruption to process operation, possibly resulting in death or severe personal injury. Always ensure that the CPU is in STOP mode (Page 33) prior to inserting a memory card.
--

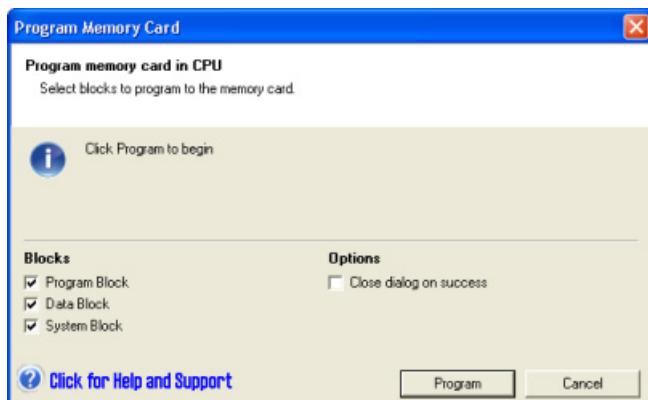
Creating a program transfer memory card

To program the memory card as a program transfer card, follow these steps:

1. Ensure that your network hardware and PLC connector cable are working, the CPU is powered on and in STOP mode, and that PLC communication is operating properly (Page 25).
2. If not already inserted, insert a microSDHC memory card in the CPU. You can insert or remove the memory card while the CPU is powered on.
3. Download (Page 33) the program to the PLC, if not already downloaded.
4. Click the Program button from the Memory Card area of the PLC menu ribbon strip.



5. Select which of the following blocks (or all) to store on the memory card:
 - Program block
 - Data block
 - System Block (PLC configuration)
6. Click the Program button.



7. Enter the password (Page 115) if one is required for programming the memory card.

Note

STEP 7-Micro/WIN SMART first erases any SIMATIC content on the card prior to transferring the program to the card. Any other data on the card that you've stored using a card reader and Windows Explorer is left undisturbed.

Note also that you cannot change the CPU to RUN mode if a memory card is inserted.

Restoring the program from a program transfer memory card

To copy the contents of the program transfer card to the PLC, you must cycle the power to the CPU with the program transfer card inserted. The CPU then performs the following tasks:

1. Clears RAM
2. Copies the user program, the system block (PLC configuration), and the data block from the memory card to CPU permanent memory.

While the copy operation is in progress, the STOP and RUN LEDs on the S7-200 SMART CPU alternately flash. When the S7-200 SMART CPU completes the copy operation, the LEDs stop flashing.

Note

Program transfer card compatibility

Restoring a program transfer card that you created on a different CPU model might fail due to the differences in model types. During the restore process, the CPU validates the following characteristics of the program content stored on the memory card:

- Size of program block
 - Size of V memory specified in the data block
 - Quantity of onboard digital I/O configured in the system block (Page 107)
 - Each retentive range that is configured in the system block
 - Signal module and signal board configurations in the system block
 - Axis of Motion configurations in the system block
 - Forced memory locations
-

Note

In addition to using a memory card as a program transfer card, you can also create a reset-to-factory-defaults memory card.

See also

[Creating a reset-to-factory-defaults memory card \(Page 137\)](#)

4.3.7 Restoring data after power on

When CPU power is applied:

- The CPU restores the program block and the system block from permanent memory.
- Up to 10 Kbytes of retentive memory assignments are restored.
- The non-retentive portions of V memory are restored from the contents of the data block in permanent memory.
- The non-retentive portions of other memory areas are cleared.

4.4 Changing the operating mode of the CPU

The CPU has two modes of operation: STOP mode and RUN mode. The status LEDs on the front of the CPU indicates the current mode of operation. In STOP mode, the CPU is not executing the program, and you can download program blocks. In RUN mode, the CPU is executing the program; however, you can download program blocks.

Placing the CPU in RUN mode

1. Click the "RUN" button on either the PLC menu ribbon strip or on the program editor toolbar: 
2. When prompted, click "OK" to change the operating mode of the CPU.

You can monitor the program in STEP 7-Micro/WIN SMART by clicking the "Program Status" button from the "Debug" menu ribbon strip, or from the program editor toolbar.

STEP 7-Micro/WIN SMART displays the values for the instructions.

Placing the CPU in STOP mode

To stop the program, click the "STOP" button  and acknowledge the prompt to place the CPU in STOP mode. You can also place a STOP instruction (Page 294) in your program logic to put the CPU in STOP mode.

Programming concepts

5.1 Guidelines for designing a PLC system

There are many methods for designing a PLC system. The following general guidelines can apply to many design projects. Of course, you must follow the directives of your own company's procedures and the accepted practices of your own training and location.

Partition your process or machine

Divide your process or machine into sections that have a level of independence from each other. These partitions determine the boundaries between controllers and influence the functional description specifications and the assignment of resources.

Create the functional specifications

Write the descriptions of operation for each section of the process or machine. Include the following topics: I/O points, functional description of the operation, states that must be achieved before allowing action for each actuator (such as solenoids, motors, and drives), description of the operator interface, and any interfaces with other sections of the process or machine.

Design the safety circuits

Identify equipment requiring hard-wired logic for safety. Control devices can fail in an unsafe manner, producing unexpected startup or change in the operation of machinery. Where unexpected or incorrect operation of the machinery could result in physical injury to people or significant property damage, consideration should be given to the use of electro-mechanical overrides which operate independently of the CPU to prevent unsafe operations.

The following tasks should be included in the design of safety circuits:

- Identify improper or unexpected operation of actuators that could be hazardous.
- Identify the conditions that would assure the operation is not hazardous, and determine how to detect these conditions independently of the CPU.
- Identify how the CPU and I/O affect the process when power is applied and removed, and when errors are detected. This information should only be used for designing for the normal and expected abnormal operation, and should not be relied on for safety purposes.
- Design manual or electro-mechanical safety overrides that block the hazardous operation independent of the CPU.
- Provide appropriate status information from the independent circuits to the CPU so that the program and any operator interfaces have necessary information.
- Identify any other safety-related requirements for safe operation of the process.

Specify the operator stations

Based on the requirements of the functional specifications, create drawings of the operator stations. Include the following items:

- Overview showing the location of each operator station in relation to the process or machine
- Mechanical layout of the devices, such as display, switches, and lights, for the operator station
- Electrical drawings with the associated I/O of the CPU or expansion module

Create the configuration drawings

Based on the requirements of the functional specification, create configuration drawings of the control equipment. Include the following items:

- Overview showing the location of each CPU in relation to the process or machine
- Mechanical layout of the CPU and expansion I/O modules (including cabinets and other equipment)
- Electrical drawings for each CPU and expansion I/O module (including the device model numbers, communications addresses, and I/O addresses)

Create a list of symbolic names (optional)

If you choose to use symbolic names for addressing, create a list of symbolic names for the absolute addresses. Include not only the physical I/O signals, but also the other elements to be used in your program.

5.2 Elements of the user program

A program organizational unit (POU) is composed of executable code and comments. The executable code consists of a main program and any subroutines or interrupt routines. The code is compiled and downloaded to the CPU. You can use the program organizational units (main program, subroutines, and interrupt routines) to structure your user program.

- The main body of the user program contains the instructions that control your application. The CPU executes these instructions sequentially, once per scan cycle.
- Subroutines are optional elements of your program which are executed only when called: by the main program, by an interrupt routine, or by another subroutine. Subroutines are useful in cases where you want to execute a function repeatedly. Rather than rewriting the logic for each place in the main program where you want the function to occur, you can write the logic once in a subroutine and call the subroutine as many times as needed during the main program. Subroutines provide several benefits:
 - Using subroutines reduces the overall size of your program.
 - Using subroutines decreases your scan time because you have moved the code out of the main program. The CPU evaluates the code in the main program every scan cycle, whether the code is executed or not, but the CPU evaluates the code in the subroutine only when you call the subroutine, and does not evaluate the code during the scans in which the subroutine is not called.
 - Using subroutines creates code that is portable. You can isolate the code for a function in a subroutine, and then copy that subroutine into other programs with little or no rework.

Note

Using V memory addresses can limit the portability of your subroutine, because it is possible for V memory address assignment from one program to conflict with an assignment in another program. Subroutines that use the local variable table (L memory) for all address assignments, by contrast, are highly portable because there is no concern about address conflicts between the subroutine and another part of the program when using local variables.

- Interrupt routines are optional elements of your program that react to specific interrupt events. You design an interrupt routine to handle a pre-defined interrupt event. Whenever the specified event occurs, the CPU executes the interrupt routine.

The interrupt routines are not called by your main program. You associate an interrupt routine with an interrupt event, and the CPU executes the instructions in the interrupt routine only on each occurrence of the interrupt event.

Note

Because it is not possible to predict when the CPU might generate an interrupt, it is desirable to limit the number of variables that are used both by the interrupt routine and elsewhere in the program.

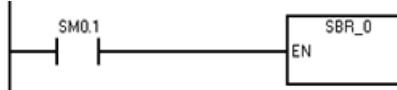
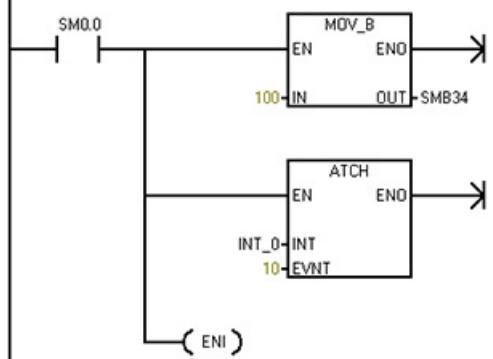
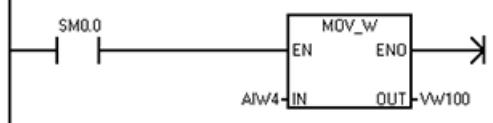
Use the local variable table of the interrupt routine to ensure that your interrupt routine uses only the temporary memory and does not overwrite data used somewhere else in your program.

There are a number of programming techniques you can use to ensure that data is correctly shared between your main program and the interrupt routines. Refer to the descriptions of the Interrupt instructions (Page 263).

- Other blocks contain information for the CPU. You can choose to download these blocks when you download your program:
 - System Block: The system block allows you to configure different hardware options for the CPU.
 - Data Block: The DB stores the initial values for different variables (V memory) used by your program.

The following example shows a program that includes a subroutine and an interrupt routine. This sample program uses a timed interrupt for reading the value of an analog input every 100 ms.

Table 5- 1 Sample program with a subroutine and an interrupt routine

Main		Network 1 LD SM0.1 CALL SBR_0	On first scan, call subroutine 0.
SBR 0		Network 1 LD SM0.0 MOV_B 100, SMB34 ATCH INT_0, 10 ENI	Set the interval to 100 ms for the timed interrupt. Enable interrupt 0.
INT 0		Network 1 LD SM0.0 MOVW AIW4, VW100	Sample the value of analog input AI4.

5.3 Creating your user program

The STEP 7-Micro/WIN SMART user interface provides a convenient working space for creating your user project program. (STEP 7-Micro/WIN SMART projects are files with a .smart file name extension.) To open the user interface, double-click the STEP 7-Micro/WIN SMART icon, or select "STEP 7-MicroWIN SMART" from the "SIMATIC" element on the "Start" menu.

5.3.1 Earlier versions of STEP 7-Micro/WIN projects

To work with a project that was created in STEP 7-Micro/WIN Version 4.0 or greater, follow these steps:

- Click the Open button from the Operations area of the File menu ribbon strip and select the desired project.



- Correct program as needed.

You cannot open projects created with a version earlier than STEP 7-Micro/WIN Version 4.0. If you attempt to open such a project, STEP 7-Micro/WIN SMART informs you that you cannot open it.

Note

Opening projects created with an older version

- Projects created by earlier versions of STEP 7-Micro/WIN (.mwp files) might contain one or more logical constructs that STEP 7-Micro/WIN SMART (.smart files) does not support. If older projects contain instructions that are not supported by STEP 7-Micro/WIN SMART, these instructions are omitted from the project when you open them in STEP 7-Micro/WIN SMART. You must take care to examine your project and redesign sections where logic was omitted.
- STEP 7-Micro/WIN SMART ignores the system block of the old project and uses a default system block for the opened project.
- STEP 7-Micro/WIN SMART omits all wizard-generated program blocks of the old project.
- If an earlier version of STEP 7-Micro/WIN (.mwp file) uses symbolic SM addressing in the OB, and the System Symbols table is generated, the symbols will map properly to the new addresses. However, if the .mwp file uses absolute SM addressing in the OB, those absolute SM addresses will not map to the new SM addresses. Refer to Symbol table (Page 92) or Special memory (Page 593) for further information.
- You cannot use Open to open a project that resides on a PLC; the project file must reside on your personal computer/ programming device.
- You can only open one project for each instance of STEP 7-Micro/WIN SMART. You must run two instances of STEP 7-Micro/WIN SMART to have two projects open at the same time. When two instances are open, you can copy and paste LAD/ FBD program elements and STL text from one project to the other.
- You can define a default path to a specific file directory where you want new STEP 7-Micro/WIN SMART projects to be opened and saved. Click the Options button from the Settings area of the Tools menu ribbon strip; click the General options, and from the Defaults tab enter the default file location.



WARNING

Risks with STEP 7-Micro/WIN Version 4.0 or greater (.mwp files) with absolute special memory (SM) addressing

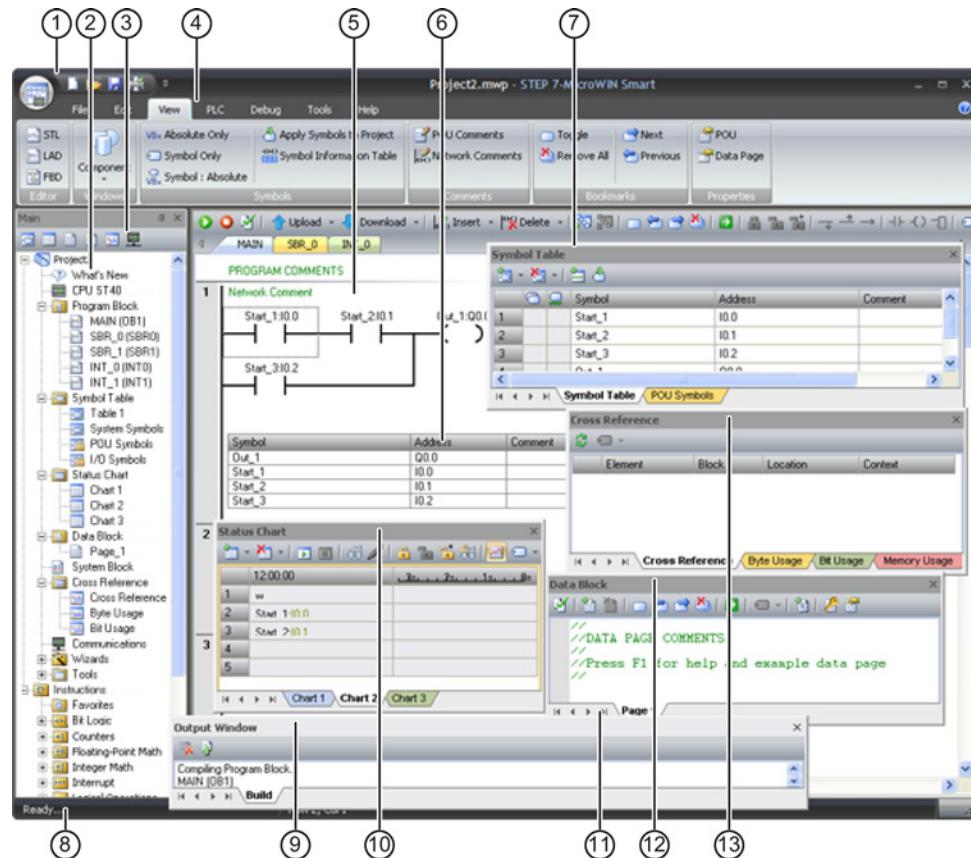
If an earlier version of STEP 7-Micro/WIN (.mwp file) uses symbolic SM addressing in the OB, and the System Symbols table is generated, the symbols will map properly to the new addresses. However, if the .mwp file uses absolute SM addressing in the OB, those absolute SM addresses will not map to the new SM addresses.

This incorrect mapping of SM addresses can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

Delete the "S7-200 Symbols" table and generate a SMART "System Symbols" table. The symbols in the OB will map to the new SM address scheme in the SMART System Symbols table.

5.3.2 Using STEP 7-Micro/WIN SMART user interface

The STEP 7-Micro/WIN SMART user interface appears below. Note that each editing window can be docked or floated and arranged on the screen as you choose. You can display each window separately, as shown below, or you can combine windows such that each one is accessible from a separate tab:



- ① Quick access toolbar (Page 86)
- ② Project tree (Page 86)
- ③ Navigation bar (Page 86)
- ④ Menus (Page 86)
- ⑤ Program editor (Page 86)
- ⑥ Symbol information table (Page 92)
- ⑦ Symbol table (Page 92)
- ⑧ Status bar (Page 86)
- ⑨ Output window (Page 86)
- ⑩ Status chart (Page 416)
- ⑪ Variable table (Page 95)
- ⑫ Data block (Page 90)
- ⑬ Cross reference (Page 410)

5.3.3 Using STEP 7-Micro/WIN SMART to create your programs

Quick access toolbar

The quick access toolbar appears just above the menu tabs. The quick access file button provides quick and easy access to most of the functions of the File menu, and to recent documents. The other buttons on the quick access toolbar are for the file functions New, Open, Save, and Print.

Project tree

The project tree displays all of the project objects and the instructions for creating your control program. You can drag and drop individual instructions from the tree into your program, or you can double-click an instruction to insert it at the current location of the cursor in the program editor.

The Project tree organizes your project:

- Right-click the Program Block folder to insert new subroutines and interrupt routines.
- Open the Program Block folder and right-click a POU to open the POU, edit its properties, password-protect it, or rename it.
- Right-click a Status Chart or Symbol Table folder to insert new charts or tables.
- Open the Status Chart or Symbol Table folder and right-click the icon in the instruction tree or double-click the appropriate POU tab to open it, rename it, or delete it.

Navigation bar

The Navigation bar appears at the top of the project tree and provides quick access to objects on the project tree. Clicking a navigation bar button is equivalent to expanding the project tree and clicking the same selection. The navigation bar presents groups of icons for accessing different programming features of STEP 7-Micro/WIN SMART.

Menu ribbon strips

STEP 7-Micro/WIN SMART displays a menu ribbon strip for each menu. You can minimize the menu ribbon strip to save space by right-clicking in the menu ribbon strip area and selecting "Minimize the Ribbon".

Program editor

The program editor contains the program logic and a variable table where you can assign symbolic names for temporary program variables. Subroutines and interrupt routines appear as tabs at the top of the program editor window. Click the tabs to move between the subroutines, interrupts, and the main program.

STEP 7-Micro/WIN SMART provides three editors for creating your program:

- Ladder logic (LAD)
- Statement list (STL)
- Function block diagram (FBD)

With some restrictions, programs written in any of these program editors can be viewed and edited with the other program editors.

You can change the editor to LAD, FBD, or STL from the Editor section of the View menu ribbon strip. You can configure the default editor at startup from the Options button of the Settings area of the Tools menu ribbon strip.

Status bar

The status bar, which is located at the bottom of the main window, provides information on the editing mode or online status operations that you perform in STEP 7-Micro/WIN SMART.

Output window

The Output Window keeps a list of the most recently compiled POU (Page 843) and any errors that occurred during the compilation. If you have the Program Editor window open as well as the Output Window, you can double-click an error message in the Output Window to scroll your program automatically to the network where the error is located.

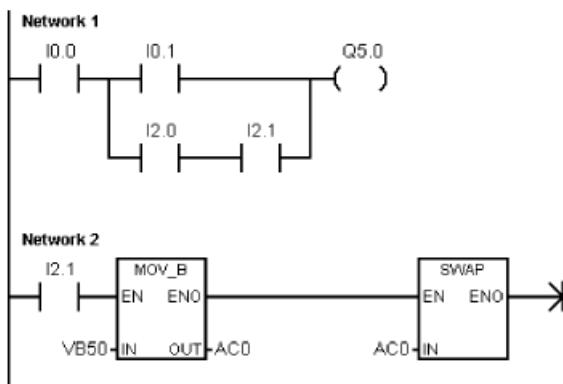
5.3.4 Using wizards to help you create your control program

STEP 7-Micro/WIN SMART provides the following wizards to make aspects of your programming easier and more automatic:

- High speed counter
- Motion
- PID
- PWM (Pulse Width Modulation)
- Text Display
- Get/Put
- Data Log

To start a wizard, select that wizard from the STEP 7-Micro/WIN SMART Tools menu ribbon strip or from the Wizards node in the project tree. You can press F1 when a wizard is displayed and get wizard details from the online Help system.

5.3.5 Features of the LAD editor



The LAD editor displays the program as a graphical representation similar to electrical wiring diagrams.

The LAD program emulates the flow of electric current from a power source through a series of logical input conditions that in turn enable logical output conditions.

A LAD program includes a left power rail that is energized. Contacts that are closed allow energy to flow through them to the next element, and contacts that are open block that energy flow. The logic is separated into networks. The program is executed one network at a time, from left to right and then top to bottom as dictated by the program.

The various instructions are represented by graphic symbols and include three basic forms:

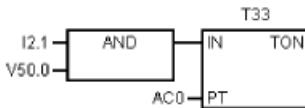
- Contacts represent logic input conditions such as switches, buttons, or internal conditions.
- Coils usually represent logic output results such as lamps, motor starters, interposing relays, or internal output conditions.
- Boxes represent additional instructions, such as timers, counters, or math instructions.

Consider these main points when you select the LAD editor:

- Ladder logic is easy for beginning programmers to use.
- Graphical representation is easy to understand and is popular around the world.
- You can always use the STL editor to display a program created with the SIMATIC LAD editor.

5.3.6 Features of the FBD editor

The FBD editor displays the program as a graphical representation that resembles common logic gate diagrams. There are no contacts and coils as found in the LAD editor, but there are equivalent instructions that appear as box instructions.



FBD does not use the concept of left and right power rails; therefore, the term "logic flow" is used to express the analogous concept of control flow through the FBD logic blocks.

The logic "1" path through FBD elements is called logic flow. The origin of a logic flow input and the destination of a logic flow output can be assigned directly to an operand.

The program logic is derived from the connections between these box instructions. That is, the output from one instruction (such as an AND box) can be used to enable another instruction (such as a timer) to create the necessary control logic. This connection concept allows you to solve a wide variety of logic problems.

Consider these main points when you select the FBD editor:

- The graphical logic gate style of representation is good for following program flow.
- You can always use the STL editor to display a program created with the SIMATIC FBD editor.

5.3.7 Features of the STL editor

The STL editor displays the program as a text-based language. The STL editor allows you to create control programs by entering the instruction mnemonics. The STL editor also allows you to create programs that you could not otherwise create with the LAD or FBD editors. This is because you are programming in the native language of the CPU, rather than in a graphical editor where some restrictions must be applied in order to draw the diagrams correctly. As shown in the following example, this text-based concept is very similar to assembly language programming.

Table 5- 2 Sample STL user program

LD	I0.0	// Read one input (I0.0).
A	I0.1	// AND with another input (Q1.0).
=	Q1.0	// Write the value to an output 1.

The CPU executes each instruction in the order dictated by the program, from top to bottom, and then restarts at the top.

STL uses a logic stack to resolve the control logic. You insert the STL instructions for handling the stack operations.

Consider these main points when you select the STL editor:

- STL is most appropriate for experienced programmers.
- STL sometimes allows you to solve problems that you cannot solve very easily with the LAD or FBD editor.
- While you can always use the STL editor to view or edit a program that was created with the LAD or FBD editors, the reverse is not always true. You cannot always use the LAD or FBD editors to display a program that was written with the STL editor.

5.4 Data block (DB) editor

The data block allows you to assign constants (Page 63) (either numeric values or character strings) to specific locations of V memory. You can make value assignments to byte (V or VB), word (VW), or double word (VD) addresses. You can also enter optional comments, preceded by // double forward slashes.

- The first line of the data block must have an explicit address assignment. You can use a memory address (absolute address) or a symbol name from the symbol table (Page 92) that you have previously assigned to an address (symbolic address).
- Subsequent lines can have explicit or implicit address assignments. An implicit address assignment is made by the editor when you type multiple data values after a single address assignment, or type a line that contains only data values. The editor assigns an appropriate amount of V memory based on your previous address allocations and the size (byte, word, or double word) of the data value(s).
- The data block editor is a free-form text editor; however, it expects an address or symbol name in the first position. If you are continuing an implicit data value entry, enter at least one space in the address position before entering the implicit data value assignment. After you finish typing a line and press the ENTER key, the data block editor formats the line (aligns columns of addresses, data, and comments; capitalizes V memory addresses) and redisplays it. The data block editor accepts uppercase or lowercase letters and allows commas, tabs, or spaces to serve as separators between addresses and data values.
- Pressing CTRL-ENTER after completing an assignment line auto-increments the address to the next available address.

Example: Data block page

```

Data Block
VB1    249, 250, 251      // Multiple data values on a single line
                                // Implicit address assignments
                                // VB2 holds data value of 250.
                                // VB3 holds data value of 251.

VB4    252                  // Cannot use previously-assigned addresses (VB0 - VB3)

                    253, 254, 255      // Line with no explicit address assignment results in
                                // implicit assignment of data values to VB5, VB6, and VB7

VW8    256, 257      // New data type: Word
                                // Implicit assignment of 257 to VW10

// Data value assignments must be valid for the data type addressed (Byte, Word, or Double Word)
X      65536      // Error indicates that data value 65536 is too large
                                // Implicit VW12 address is a Word data type

// You can omit the B, W, or D size designation from an address.
// The data block editor assigns the value to VB, VW, or VD memory according to size of value.

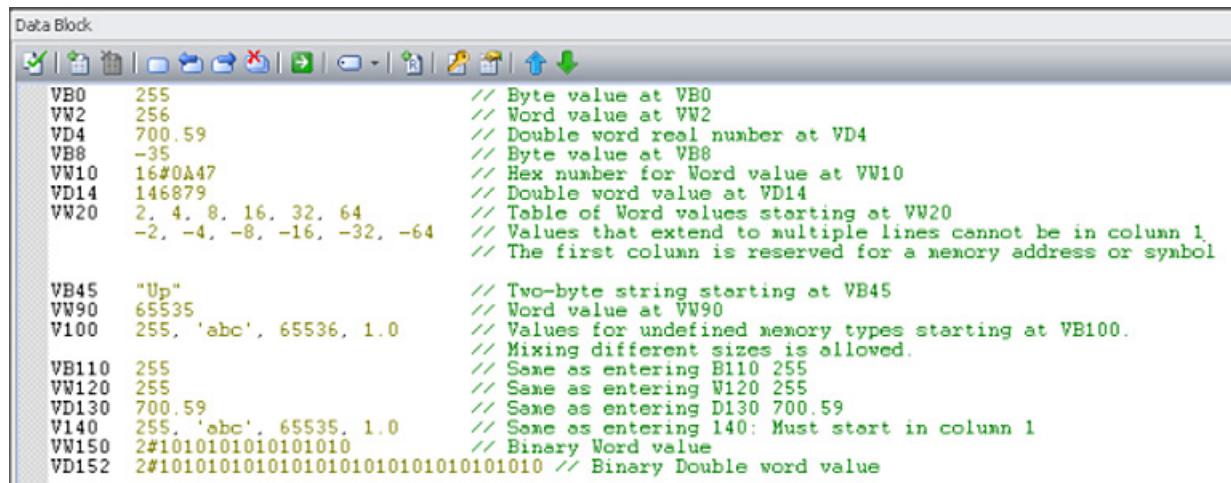
V12    258      // Word value explicitly assigned to VW12
                65537      // Double word value implicitly assigned to VD14

```

Note: Enter a space before the data values on the lines where you enter no explicit address.

Note: Enter a space before the data values on the lines where you enter no explicit address.

Example: Direct address and number values



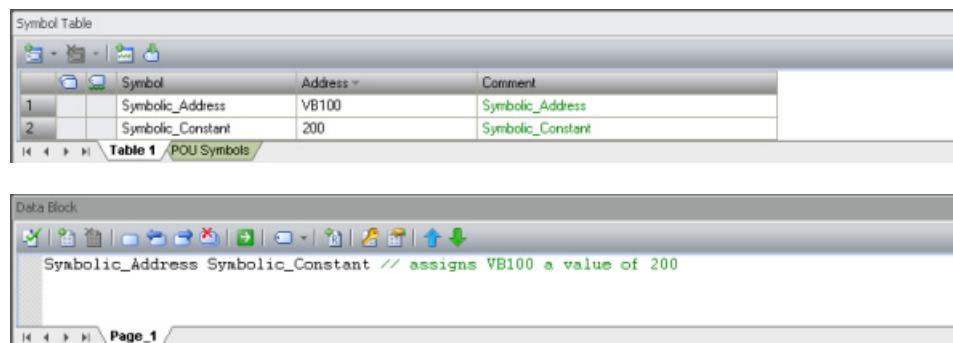
```

Data Block
VB0 255 // Byte value at VB0
VV2 256 // Word value at VV2
VD4 700.59 // Double word real number at VD4
VB8 -35 // Byte value at VB8
VV10 16#0A47 // Hex number for Word value at VV10
VD14 146879 // Double word value at VD14
VV20 2, 4, 8, 16, 32, 64 // Table of Word values starting at VV20
-2, -4, -8, -16, -32, -64 // Values that extend to multiple lines cannot be in column 1.
// The first column is reserved for a memory address or symbol

VB45 "Up" // Two-byte string starting at VB45
VV90 65535 // Word value at VV90
V100 255, 'abc', 65536, 1.0 // Values for undefined memory types starting at VB100.
// Mixing different sizes is allowed.
VB110 255 // Same as entering B110 255
VV120 255 // Same as entering W120 255
VD130 700.59 // Same as entering D130 700.59
V140 255, 'abc', 65535, 1.0 // Same as entering 140: Must start in column 1
VV150 2#1010101010101010 // Binary Word value
VD152 2#10101010101010101010101010101010 // Binary Double word value

```

Example: Symbolic address and symbolic number assignment



Symbol Table

	Symbol	Address	Comment
1	Symbolic_Address	VB100	Symbolic_Address
2	Symbolic_Constant	200	Symbolic_Constant

Data Block

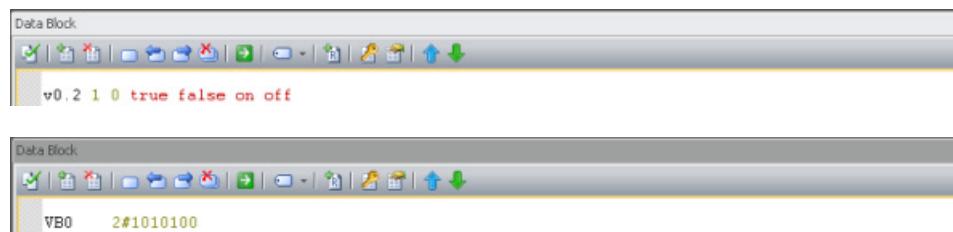
```

Symbolic_Address Symbolic_Constant // assigns VB100 a value of 200

```

Example: Alternate binary entry methods and resultant binary assignment

You can enter values of 1 or 0 for binary assignments, or "true", "false", "on" or "off" (in either lower, upper, or mixed case). The data block editor interprets your entry and shows the resultant binary assignment.



Data Block

```

v0.2 1 0 true false on off

```

Data Block

```

VB0 2#10101010

```

5.5 Symbol table

A symbol is a symbolic name you assign to a memory address or a constant. You can create symbol names for the following memory types: I, Q, M, SM, AI, AQ, V, S, C, T, HC. Symbols defined in the symbol table are global in scope. You can use your defined symbols in all of the Program Organizational Units (Page 81) (POUs) of your program. If you make a variable name assignment in a variable table (Page 95), the variable is local in scope. It only applies to the POU where you defined it. This type of symbol is referred to as a "local variable" to differentiate it from symbols that are global in scope. You can define symbols either before or after you create your program logic.

WARNING

Risks with STEP 7-Micro/WIN Version 4.0 or greater (.mwp files) with absolute special memory (SM) addressing

If an earlier version of STEP 7-Micro/WIN (.mwp file) uses symbolic SM addressing in the OB, and the System Symbols table is generated, the symbols will map properly to the new addresses. However, if the .mwp file uses absolute SM addressing in the OB, those absolute SM addresses will not map to the new SM addresses.

This incorrect mapping of SM addresses can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

Delete the "S7-200 Symbols" table and generate a SMART "System Symbols" table. The symbols in the OB will map to the new SM address scheme in the SMART System Symbols table.

Opening a symbol table

To open a symbol table, use one of the following methods:

- Click the Symbol Table  button on the navigation bar (Page 21).
- Select "Symbol Table" from the component drop down list in the Windows area of the View menu.
- Open the Symbol Table Folder in the project tree (Page 27), select a table name, then press Enter or double-click the table name.

You can also use symbols from the System symbol table in your project. The pre-defined table of system symbols provides access to commonly used PLC system functions. PLC system symbols associate a function name with the PLC special memory locations used to invoke that function.

Assigning symbols in the symbol table

To assign a symbol to an address or constant value, follow the procedure below:

1. Open the symbol table.
2. Type the symbol name (for example, Input1) in the Symbol column. The maximum number of characters in a symbol name is 23 single-byte characters.

Note

Until you assign an address or constant value to the symbol, it appears as an undefined symbol (green wavy underline). After you complete the Address column assignment, STEP 7-Micro/WIN SMART removes the green wavy underline.

If you have selected to view both symbolic and absolute view of operands for your project, lengthy symbol names are truncated with a tilde (~) in the program editor. You can place your mouse pointer over the truncated name to see the entire name displayed in a tooltip.

3. Type the address or constant value (for example, VB0 or 123) in the Address column. Note that to assign a string constant to a symbol, you enclose the string constant in double quotation marks.

4. Optionally, type in a comment up to a maximum of 79 characters.

You can resize the width of the columns in the symbol table editor as needed.

Note

You can create multiple symbol tables; however, you cannot use the same symbol name more than once as a global symbol assignment.

By contrast, you can reuse symbol names in variable tables.

Syntax rules and indication of errors

STEP 7-Micro/WIN SMART indicates erroneous or incomplete symbol assignments with color and wavy underlining:

Symbol	Address
2name	I0.0
	VBB0
Begin	I0.2

Red text indicates invalid syntax.

A symbol cannot begin with a numeral.

VBB0 is an invalid address.

Begin is a reserved word and invalid as a symbol name.

Symbol	Address
Pump1	I0.0
Pump1	I0.0
SymConstant	1234
SymConstant	5678

A red wavy underline indicates invalid use.

Pump1 and SymConstant are duplicate symbol names.

I0.0 is a duplicate address.

Symbol	Address
Pump1	

A green wavy underline indicates an undefined symbol.

Pump1 has no address.

Observe the following syntax rules when defining symbols:

- Symbolic names can contain alphanumeric characters, underscores, and extended characters from ASCII 128 to ASCII 255. The first character cannot be a numeral.
- Use double quotation marks to enclose an ASCII constant string that you assign to a symbol name.
- Use single quotation marks to enclose an ASCII character constant in byte, word, or double word memory.
- Do not use keywords as symbol names.
- The maximum length for a symbol name is 23 characters.

Note

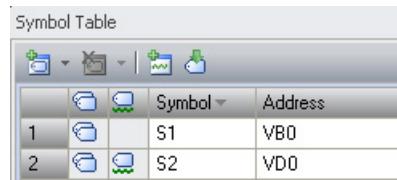
When you correct an erroneous symbol name or address, press the TAB key, ENTER key, or an ARROW key to complete the edited correction.

Indirect addressing

When referencing a symbol in the program editor, you can use indirect notation (& and *) with symbol names as with direct addresses. For more information about indirect addressing, see the topic on direct and indirect addressing.

Viewing overlapped and unused symbols

STEP 7-Micro/WIN SMART indicates overlapped symbols with the  icon and unused symbols with the  icon. In the symbol table below, symbols S1 and S2 overlap the VB0 memory address. Also, symbol S1 is not used in the project.



Symbol Table			
	Symbol	Address	
1	S1	VB0	
2	S2	VD0	

Inserting additional rows

Use one of the following methods to insert additional rows in the symbol table:

- Right-click a cell of the symbol table and select Insert>Row from the context menu. STEP 7-Micro/WIN SMART inserts a new row above the current location.
- From the Insert area of the Edit menu ribbon strip, select "Row". STEP 7-Micro/WIN SMART inserts a new row above the current location of the cursor in the symbol table.
- To insert a new row at the bottom of the Symbol Table, place the cursor in any cell of the last row and press the DOWN ARROW key.

Sorting a symbol table

You can sort a symbol table by the Symbol or by the Address column in either ascending or descending alphabetical order. In the Address column, numeric constants sort above string constants, which sort above addresses.

To sort a column, click either the Symbol or Address column header to sort by that value. To reverse the order of the sort, click the column again. STEP 7-Micro/WIN SMART displays an up or down arrow next to the column that is sorted to indicate the sort selection.

Note

You can print symbol tables from the Print area of the File menu ribbon strip.

You can view symbols on a network-by-network basis by displaying the symbol information table.

5.6 Variable table

A variable table allows you to define variables that are local to a specific POU. The following situations define when to use a local variable:

- You want to create portable subroutines that do not make references to absolute addresses or global symbols.
- You want to use interim variables (local variables declared as TEMP) to perform calculations in order to free up PLC memory.
- You want to define inputs and outputs for your subroutines.

If these descriptions do not fit your situation, you do not need to use local variables; you can make all of your symbolic values global by defining them in the symbol table (Page 92).

Understanding local variables

You can use the variable table of the program editor to assign variables that are unique to an individual subroutine or interrupt routine.

Local variables can be used as parameters that are passed in to a subroutine and can be used to increase the portability or reuse of a subroutine.

Each POU (Page 81) in your program has its own variable table, with 64 bytes of L memory (60 bytes if you are programming in LAD or FBD). These local variable tables allow you to define variables that are restricted in scope: a local variable is only valid inside the POU where it was created. By contrast, global symbols, which are valid in every POU, can only be defined in the symbol table. In cases where you use the same symbolic name (e.g., INPUT1) for a global symbol and a local variable, the local definition takes precedence inside the POU where the local variable has been defined and the global definition is used in the other POUs.

You assign a declaration type (TEMP, IN, IN_OUT, or OUT) and a data type, but not a memory address, when you make assignments in a Local Variable Table; the Program Editor automatically assigns memory locations in L memory for all local variables.

A variable table symbolic address assignment associates a symbol name with an L memory address, where the data value of concern is stored. The Local Variable Table does not support symbolic constants that assign a value directly to a symbol name (this is allowed in the Symbol\Global Variable tables).

Note

Local data values are not initialized to zero by the PLC. You must initialize the local variables that you use, in your program logic.

Declaration types for local variables

The type of local variable assignment you can make depends on the POU where you are making the assignment. The main program (OB1), interrupt routines, and subroutines can use temporary (TEMP) variables. Temporary variables are only available while the block is being executed and are then free to be overwritten when the block is completed.

Data values can be passed as parameters in and out of a subroutine as follows:

- If you want to pass a data value into a subroutine, then create a variable in the subroutine's variable table and specify its declaration type as IN.
- If you want to pass a data value established in the subroutine back to the calling routine, then create a variable in the subroutine's variable table and specify its declaration type as OUT.
- If you want to pass an initial data value into a subroutine, perform an operation that may modify the data value, and pass the modified result back to the calling routine, then create a variable in the subroutine's variable table and specify its declaration type as IN_OUT.

Declaration type	Description
IN	Input parameter provided by the calling POU.
OUT	Output parameter returned to the calling POU.
IN_OUT	Parameter whose value is supplied by the calling POU, modified by the subroutine, and then returned to the calling POU.
TEMP	Temporary variable that is saved temporarily in the local data stack. Once the POU has been executed completely, the value of the temporary variable is no longer available. Temporary variables do not keep their value between POU executions.

Data type checking for local variables

When you pass local variables as parameters for a subroutine, the data type that you have assigned in the Local Variable Table of that subroutine must match the data type of the value in the calling POU.

Example

You call SBR0 from OB1, using a global symbol called INPUT1 as an input parameter of the subroutine.

Inside the Local Variable Table of SBR0, you have defined a local variable called FIRST as an input parameter.

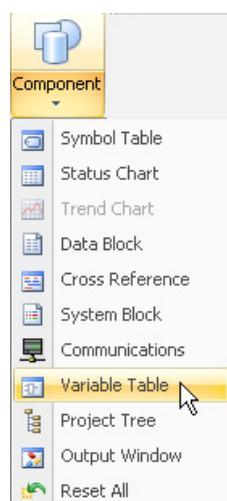
When OB1 calls SBR0, the value of INPUT1 is passed to FIRST.

The data types of INPUT1 and FIRST must match.

If INPUT1 is a REAL and FIRST is a REAL, the data types match. If INPUT1 is a REAL but FIRST is an INT, the data types do not match and the program cannot be compiled until this error is corrected.

Viewing the variable table

To view the variable table for the POU selected in the program editor, select "Variable table" from the Component drop-down list in the Windows area of the View menu.



Note

You can put the variable table on the quick access toolbar (Page 81) for easy access.

Making assignments in the variable table

Note

Make the assignments in the variable table **before** using local variables in your program. When you use symbolic names in your program, the program editor checks first the Local Variable Table of the appropriate POU, and then the symbol table. If the symbolic name is undefined in both places, the program editor treats it as an undefined global symbol which is indicated by a green wavy underline. The program editor does not automatically re-read the variable table and make corrections to your program logic. If you later make a data type assignment that defines that symbolic name (in the local variable table), you must manually insert a pound symbol (#) in front of the name, like this: #UndefinedLocalVar (in the program logic). For this reason, declaring the variables prior to usage minimizes the programming effort.

The maximum limit of input/ output parameters for each subroutine call is 16. If you attempt to download a program that exceeds this limit, STEP 7-Micro/WIN SMART returns an error.

To make an assignment in a variable table, follow the procedure below.

1. Ensure that the correct POU is displayed in the Program Editor window by clicking, if necessary, on the tab of the desired POU. (Since every POU has its own variable table, you need to make sure that you are making assignments to the correct POU.)
2. Display the variable table if it is not already visible by selecting "Variable Table" from the Component drop-down list in the Windows area of the view menu.
3. Choose a row that has the right variable type for the kind of variable that you want to define, and type a name for the variable in the Symbol field. If you are making an assignment in OB1 or an interrupt routine, the variable table contains only TEMP variables. If you are making an assignment in a subroutine, the variable table contains IN, IN_OUT, OUT, and TEMP variables. Do not preface the name with a pound symbol in the variable table. Pound symbols are only used to precede local variables in the program code.

Note

Local variable names are permitted to contain a maximum of 23 alphanumeric characters and underscores. They are also permitted to contain extended characters (ASCII 128 to ASCII 255). The first character is restricted to alpha and extended characters only. It is illegal to use keywords as symbolic names, or to use names that begin with a number or contain characters that are not alphanumeric or in the extended character set.

Local variable names are downloaded and stored in CPU memory. The use of longer variable names may reduce the memory available to store your program.

4. Click the mouse pointer in the Data Type field and use the list box to select an appropriate data type for the local variable.

Note

When you assign local variables as parameters for subroutines, you must ensure that the data type that you assign to the local variable does not conflict with the operand being used in the subroutine call.

5. Optionally provide a comment describing your local variable.

After you supply a value for the Symbol and Data Type fields, the Program Editor automatically assigns an L memory address to the local variable.

Entering additional variables

The variable table displays a fixed number of rows for local variables. To add more rows to the table, select a row in the table of the variable type that you want to add and click the Insert button  in the variable table window. A new row is automatically generated above the row you selected, and is for the same variable type that you selected.

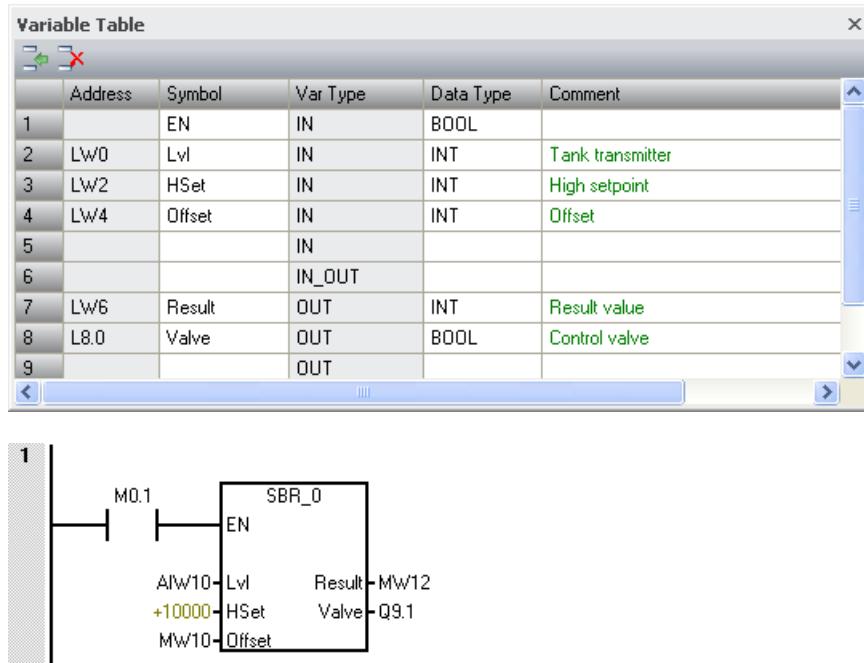
You can also add rows by right-clicking an existing row and selecting **Insert > Row** or **Insert > Row Below** from the context menu.

Deleting variables

To delete a local variable, select it in the variable table and click the Delete button . Alternatively you can delete a row by right-clicking it and selecting **Delete > Row** from the context menu.

Variable table example

The following example shows a typical variable table for SBR_0, and then a call to SBR_0 from another program block.



See also

Programming software (Page 21)

5.7 PLC error reaction



When the "Devices" entry of the tree is selected, then summary error status for the CPU and any expansion modules is displayed. To show detailed error information for each device, select the device name in the tree.

Error and status information:

- The "Last Fatal" field shows the previous fatal error code generated by the CPU. This value is retained over power cycles. This location is cleared whenever all memory of the CPU is cleared.
- Selecting the Event log entry in the tree displays the CPU's stored event history, including events such as power up, power down, errors, and mode transitions. The time of events is also listed.
- The PLC also provides SM bits for programmed reactions to errors. Refer to the listing of the SM bits (Page 593).
- The GET_ERROR (Get non-fatal error code) program instruction returns the PLC's current non-fatal error code and clears the non-fatal error information latched in the PLC. Refer to the GET_ERROR instruction (Page 296) for details.

5.7.1 Non-fatal errors and I/O errors

The CPU does not change to STOP mode when it detects a non-fatal error. It only logs the event in SM memory and continues with the execution of your program. However, you can design your program to force the CPU to STOP mode when a non-fatal error is detected.

The following sample program shows a network of a program that is monitoring two of the global non-fatal error bits and changes the CPU to STOP whenever either of these bits = 1.

Table 5- 3 Example logic for detecting a non-fatal error condition

LAD	STL
	Network 1 LD SM5.0 O SM4.3 STOP

Non-fatal errors are those indicating problems with the construction of the user program or with certain instruction execution problems in the user program. I/O errors are those indicating problems with the I/O of the CPU, signal board, and expansion modules. You can use STEP 7-Micro/WIN SMART to view the error codes that were generated by the non-fatal and I/O errors.

Click the PLC button from the Information section of the PLC menu ribbon strip, to see the current error status of a PLC connected to STEP 7-Micro/WIN SMART.

Table 5- 4 Non-fatal error types

	Description
Program-compile errors in the CPU	The CPU compiles the program as it downloads. If the CPU detects that the program violates a compilation rule, the download is aborted and an error code is generated. (A program that was already downloaded to the CPU would still exist in the permanent memory and would not be lost.) After you correct your program, you can download it again.
I/O device errors	After power-up and after a system block download, the CPU verifies that the I/O configuration stored in the system block matches the CPU, signal board, and expansion modules that are actually present. Any mismatch results in the generation of a configuration error for the device. During runtime, other I/O problems (such as missing user power or input value exceeding limits) that are detected by a device can generate an I/O error. The module status information is stored in special memory (SM) bits. Your program can monitor and evaluate these bits. SM5.0 is the global I/O error bit and remains set while any I/O error condition exists.
Program execution errors	Your program can create error conditions while being executed. These errors can result from improper use of an instruction or from the processing of invalid data by an instruction. For example, an indirect-address pointer that was valid when the program compiled could be modified during the execution of the program to point to an out-of-range address. This is an example of a run-time programming problem. SM4.3 is set upon the occurrence of a run-time programming problem and remains set while the CPU is in RUN mode. You can get any non-fatal error code and reset SM4.3 to OFF by executing the GET_ERROR instruction.

Refer to the non-fatal error code list (Page 587) for a description of compile rule violations and run-time programming problems.

Refer to the description of the SM bits (Page 593) for more information about the SM bits used for reporting I/O and program execution errors.

5.7.2 Fatal errors

Fatal errors cause the PLC to stop the execution of your program. Depending upon the severity of the fatal error, it can render the PLC incapable of performing any or all functions. The objective for handling fatal errors is to bring the PLC to a safe state from which the PLC can respond to interrogations about the existing error conditions.

When a fatal error is detected, the PLC changes to STOP mode, turns on the STOP and the ERROR LED, overrides the output table, and turns off the outputs. The PLC remains in this condition until the fatal error condition is corrected.

Once you have made the changes to correct the fatal error condition, use one of the following methods to restart the PLC:

- Turn the PLC power off and then on.
- Using STEP 7-Micro/WIN SMART, click the "Warm Start" button in the modify area of the PLC ribbon strip. This forces the PLC to restart and clear any fatal errors.

Restarting the PLC clears the fatal error condition and performs power-up diagnostic testing to verify that the fatal error has been corrected. If another fatal error condition is found, the PLC again sets the ERROR LED, indicating that an error still exists. Otherwise, the PLC begins normal operation.

Some error conditions can render the PLC incapable of communication. In these cases, you cannot view the error code from the PLC. These types of errors indicate hardware failures that require the PLC to be repaired; they cannot be fixed by changes to the program or clearing the memory of the PLC.

Refer to the fatal error code list (Page 590) for details.

5.8 Program edit in RUN mode



WARNING

Risks when downloading a program in RUN mode

When you download program changes to a PLC in RUN mode, your changes immediately affect process operation. You have no margin for error; mistakes in your programming edits can cause death or serious injury to personnel, and/or damage to equipment. Only qualified personnel should perform a program edit in RUN mode.

Overview

The "program edit in RUN mode" feature allows you to make changes to a program and download them to your PLC without switching to STOP mode.

- You can make minor changes to your current process without having to shut down.
Example: Change a parameter value.
- You can debug a program more quickly with this feature.
Example: Invert the logic for a normally open or normally closed switch.

If you download changes to a real process (as opposed to a simulated process, which you might do in the course of debugging a program), be sure to think through the possible safety consequences to machines and machine operators before you download.

You can download only the program block (OB1, subroutines, and interrupts) during a program edit in RUN mode. You cannot download the system block or the data block during a program edit in RUN mode.

Prerequisites for editing in RUN mode

You cannot download your program edits to a PLC that is in RUN mode unless you have met these prerequisites:

- Your program must compile successfully.
- You must have successfully established communications between the computer where you are running STEP 7-Micro/WIN SMART and the PLC.
- The firmware of the target PLC must support the program edit in RUN mode feature. Only S7-200 SMART CPUs with version V2.0 or later firmware support the program edit in RUN mode feature.
- You must provide a password for a protected POU to open the block (for normal editing, edit in RUN mode, and program status operations).

If you change the PLC to STOP mode while a program edit in RUN mode is in progress, the PLC aborts the editing session.

Possible complications

To help you decide whether to download your program modifications to the PLC in RUN mode or STOP mode, consider the following effects from various types of program modifications made during a RUN mode edit:

- If you delete the control logic for an output, the output maintains its last state until the next power cycle or transition to STOP mode.
- If you delete HSC or Motion/PWM functions that were running at the time of the edit in RUN mode, then these functions continue to run until the next power cycle or transition to STOP mode.
- If you delete ATCH or DTCH instructions in a RUN mode edit but do not delete the associated interrupt routine, then the interrupt routine continues to execute whenever the controlling event occurs until the next power cycle or transition to STOP mode.
- If you add ATCH instructions that are conditional on the first scan flag, the CPU does not enable these events until the next power cycle or STOP-to-RUN mode transition.
- If you delete an ENI or DISI instruction, activated interrupt routines events still continue to operate until the next power cycle or transition from RUN to STOP mode.

- If you modify the table address of a RCV instruction and the RCV instruction is active at the time of the edit in RUN mode, then the PLC writes the received data to the old table address. The PLC does not use the new address until the current receive request (to the old address) completes. Because you have edited your program, if the program looks for the data in the new address, the data will not be there. GET and PUT instructions function similarly.
- The PLC does not execute logic that is conditional on the first scan flag until after a power cycle or a transition from STOP to RUN mode. The startup of the modified program after a RUN mode edit does not set the first scan flag.

Handling positive or negative transitions

To minimize the process impact of changes that involve the relocation of positive transition (EU) and negative transition (ED) instructions in your program during a RUN mode edit, STEP 7-Micro/WIN SMART temporarily allocates a number to each transition instruction included in your program. For each transition instruction that you add in your program during a RUN mode edit, you must assign it a unique identification number. To assist you in selecting an unused number, STEP 7-Micro/WIN SMART provides an edge usage tab on the cross reference window, available when you activate the Program Edit in RUN Mode feature. This table lists all EU/ ED instructions that are currently in use in your program, so you can use this list to guide you in making changes to your program.

Performing a program edit and download in RUN mode

To initiate a program edit in RUN mode, follow these steps:

1. From the Settings area of the Debug menu ribbon strip, click the Edit In Run button.



Note

If you have not saved your current program in the program editor, STEP 7-Micro/WIN SMART prompts you to save your project. You can use the same name or you can change the name.

2. From the warning dialog, click the "Continue" button to confirm that you want to proceed with editing your program in RUN mode. STEP 7-Micro/WIN SMART uploads the program that is currently stored in the CPU and displays it in the program editor, where you can make the changes you need.

After you make the desired editing changes, you must download them before they can take effect in the CPU. Once you initiate a download, you cannot perform other tasks in STEP 7-Micro/WIN SMART until the download completes.

Examine the output window to see whether any compile errors exist (for instance, duplicate EU or ED numbers). You can double-click the error message to edit the offending network in the program editor.

Specifying CPU allocation (background time)

During a program edit in RUN mode, the CPU requires time to compile the modified program in the background while it continues to execute the currently loaded program. In the system block (Page 109), you can configure the amount of background time that is available for the compilation. Note that you can only download the system block when the CPU is in STOP mode.

5.9 Features for debugging your program

STEP 7-Micro/WIN SMART provides the following features to help you debug your program:

- Adding bookmarks in your program to make it easy to move back and forth between lines of a long program
- Tracking references in your program with the cross reference table (Page 410)
- Using a status chart (Page 416) to display PLC data values and status
- Displaying status in the program editor (Page 412)

For more information about debugging your program, refer to the chapter on diagnostics and troubleshooting (Page 409).

PLC device configuration

6.1 Configuring the operation of the PLC system

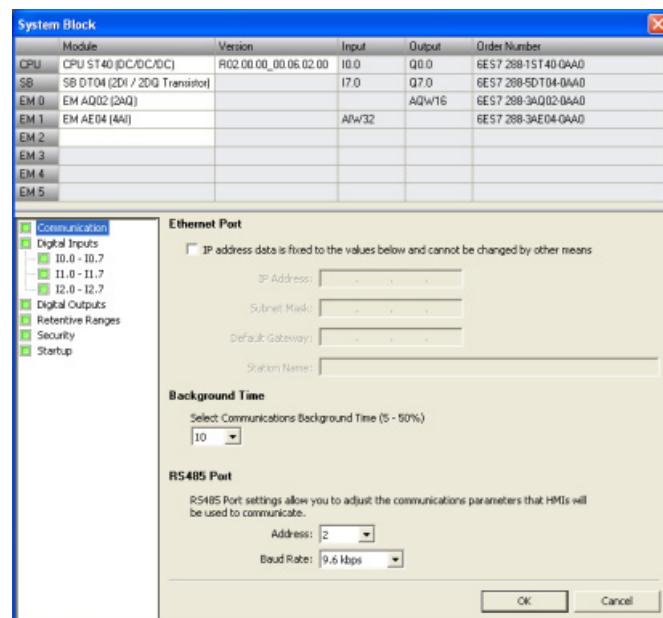
6.1.1 System block

The system block provides configuration of the S7-200 SMART CPU, signal boards, and expansion modules.

Use one of the following methods to view and edit the system block to set up CPU options:

- Click the "System Block"  button on the navigation bar (Page 21).
- Select "System Block" from the Component drop-down list (Page 21) in the Windows area of the View menu ribbon strip.
- Select the "System Block" node, then press Enter; or double-click the "System Block" node in the project tree (Page 21).

STEP 7-Micro/WIN SMART opens the system block, and displays the configuration options that are applicable for your CPU type.



6.1 Configuring the operation of the PLC system

Hardware configuration

The top part of the System Block dialog displays the modules that you have configured and allows you to add or delete modules. Use the drop-down lists to change, add, or delete the CPU model, signal board, and expansion modules. As you add modules, the input and output columns display the assigned input and output addresses.

Note

Optimally, select the CPU model and firmware version (V1 or V2) in the system block to be the model and firmware version of the actual CPU you plan to use. When downloading your project, if the CPU model and firmware version in the project does not match the model and firmware version of the connected CPU, STEP 7-Micro/WIN SMART issues a warning message. You can continue with the download, but if the connected CPU does not support the resources and capabilities that the project requires, a download error occurs.

Module options

The bottom part of the system block dialog displays options for the module that you select in the top part. Click any node in the configuration options tree to modify the project configuration for the selected module.

The system block includes the following configuration options for CPU modules:

- Communication (Page 109)
- Digital inputs and pulse catch bits (Page 111)
- Digital outputs (Page 113)
- Retentive Ranges (Page 114)
- Security (Page 115)
- Startup (Page 119)

Configuration options specific to other devices such as analog inputs (Page 120), analog outputs (Page 123), RTD analog inputs (Page 125), Thermocouple (TC) analog inputs (Page 129), RS485/RS232 CM01 communications signal board (Page 133), Battery BT01 signal board (Page 134), and additional digital inputs and outputs are accessible from the system block when you add those modules.

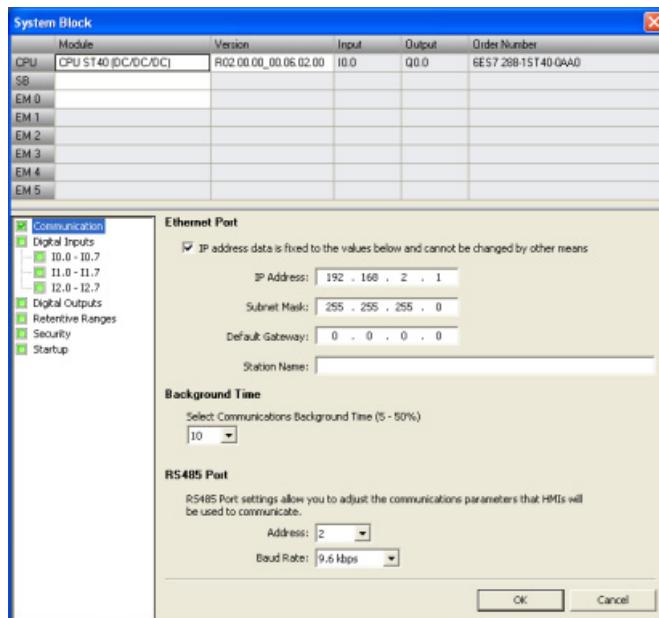
You must establish communications between STEP 7-Micro/WIN SMART and your CPU before you can download or upload a system block.

You can then download a modified system block in order to provide the CPU with a new system configuration. New properties that you enter take effect when you download (Page 33) the modifications to the CPU.

Alternatively, you can upload an existing system block from the CPU in order to make your STEP 7-Micro/WIN SMART project configuration match that of the CPU.

6.1.2 Configuring communication

Click the Communication node of the system block (Page 107) dialog to configure the Ethernet port, background time, and RS485 port.



Ethernet port

If you want your CPU to obtain its Ethernet network port information from the project, click the "IP address data is fixed to the values below and cannot be changed by other means" checkbox. You can then enter the following Ethernet network information:

- IP address: Each device must have an Internet Protocol (IP) address. The device uses this address to deliver data on a more complex, routed network.
- Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet are usually located in close physical proximity to each other on a Local Area Network (LAN). The subnet mask defines the boundaries of an IP subnet. A subnet mask of 255.255.255.0 is generally suitable for a local network.
- Default gateway: Gateways (or IP routers) are the link between LANs. Using a gateway, a computer in a LAN can send messages to other networks, which might have other LANs behind them. If the destination of the data is not within the LAN, the gateway forwards the data to another network or group of networks where it can be delivered to its destination. Gateways rely on IP addresses to deliver and receive data packets.

6.1 Configuring the operation of the PLC system

- Station name: The station name is the name by which this CPU is identified on the network. Use a name that helps you identify the CPU on the Communications dialog.

Note

The station name follows the standard DNS (Domain Name System) naming conventions. The S7-200 SMART CPUs limit the station name to a maximum of 63 characters, which can consist of the lower case letters a through z, the digits 0 through 9, the hyphen character (minus sign) and the period character.

The CPU prohibits certain names:

- The station name must not have the format n.n.n.n where n is a value of 0 through 999.
- You cannot begin the station name with the string port-nnn or the string port-nnn-nnnn where n is a digit 0 through 9. For example, port-123 and port-123-45678 are illegal station names. A station name cannot start or end with a hyphen or period.

Background time

You can configure the percentage of the scan cycle time that is dedicated to processing the communication requests. As you increase the percentage of time that is dedicated to processing communication requests, you are increasing scan time, which makes your control process run more slowly.

The default percentage of the scan time dedicated to processing communication requests is set to 10%. This setting provides a reasonable compromise for processing compilation/status operations, while minimizing the impact to your control process. You can adjust this value by 5% increments up to a maximum of 50%.

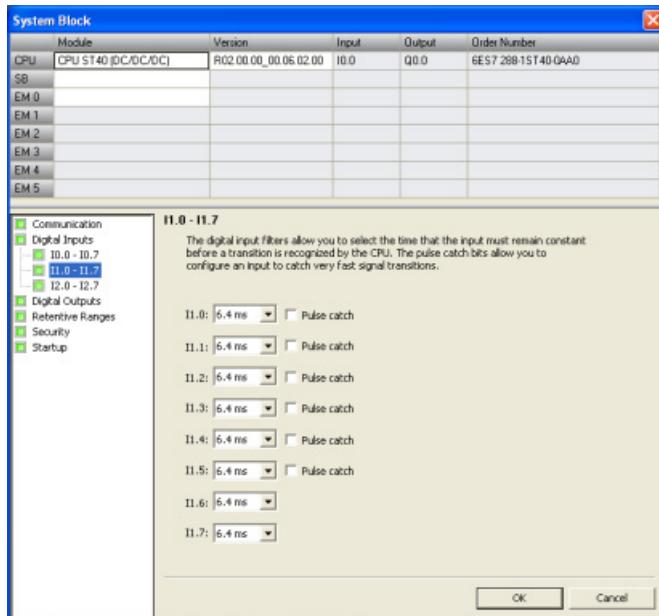
RS485 port

Use these settings to adjust the communication parameters for system protocols for the onboard RS485 port. The system protocols are used when connecting to HMI devices:

- RS485 port address: Click the scroll buttons to enter the desired CPU address (1-126). The default port address is 2.
- Baud Rate: Choose the desired data baud rate from the dropdown list (9.6 kbps, 19.2 kbps, or 187.5 kbps).

6.1.3 Configuring the digital inputs

Click the Digital Inputs node of the system block (Page 107) dialog to configure digital input filters and pulse catch bits.



Digital input filters

You can filter digital input signals by setting an input delay time. This delay helps to filter noise on the input wiring that could cause inadvertent changes to the states of the inputs. When an input state change occurs, the input must remain at the new state for the duration of the delay time in order to be accepted as valid. The filter rejects noise impulses and forces input lines to stabilize before the data is accepted.

The S7-200 SMART CPU allows you to select an input delay time for all of its digital input points. The quantity of input points available is dependent upon your CPU model (Page 15).

The first fourteen input points (I0.0 through I0.7 and I1.0 through I1.5) support an expanded set of delay time choices (selectable to one of seven settings in the range of 0.2 ms to 12.8 ms or one of seven settings in the range of 0.2 µs to 12.8 µs). The remaining input points (I1.6 and greater) support only a limited set of input delay choices (6.4 ms, 12.8 ms, or no filtering).

For example, all twelve input points of a CPU SR20 support the expanded list of input delay settings. In a CPU ST40, the expanded list of input delay choices is available for the first fourteen input points, while only the limited list of input delay choices is available for the remaining ten input points.

The default filter time for all input points is 6.4 ms.

6.1 Configuring the operation of the PLC system

To set an input delay, follow these steps:

1. Select the time of the delay from the drop-down list beside one or more inputs.
2. Click the OK button to enter the selections.

⚠️ WARNING

Risks with changes to filter time for digital input channel

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 12.8 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 12.8 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, a power cycle of the CPU must be applied.

Pulse catch bits

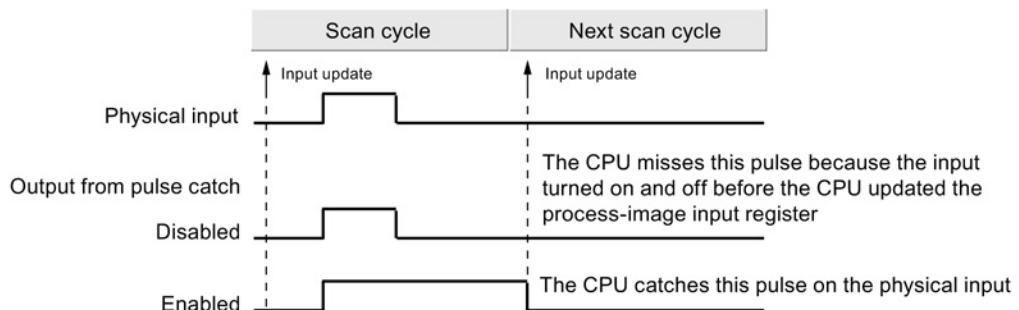
The S7-200 SMART CPU provides a pulse catch feature for digital input points. The pulse catch feature allows you to capture high-going pulses or low-going pulses that are of such a short duration that they would not always be seen when the CPU reads the digital inputs at the beginning of the scan cycle.

When pulse catch is enabled for an input, a change in state of the input is latched and held until the next input cycle update. This ensures that a pulse which lasts for a short period of time will be caught and held until the S7-200 SMART CPU reads the inputs.

You can enable individual pulse catch operation for the first fourteen digital input points (I0.0 through I0.7 and I1.0 through I1.5), dependent upon the CPU model (Page 15).

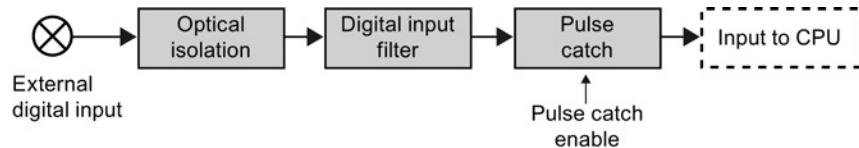
If your configuration includes an SB DT04, you can enable the two additional digital input points available on this signal board for pulse catch operation.

The figure below shows the basic operation of the S7-200 SMART CPU with and without pulse catch enabled:

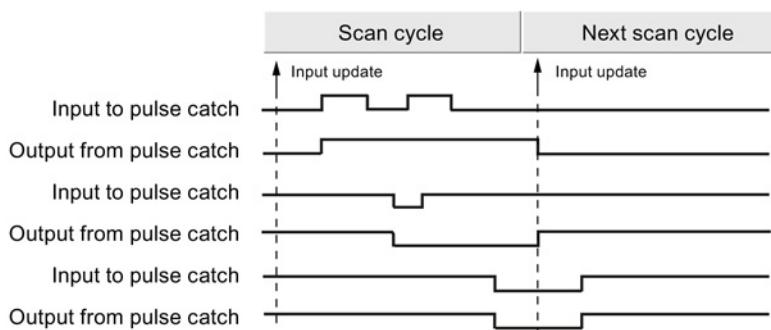


6.1 Configuring the operation of the PLC system

Because the pulse catch function operates on the input after it passes through the input filter, you must adjust the input filter time so that the pulse is not removed by the filter. The figure below shows a block diagram of the digital input circuit:

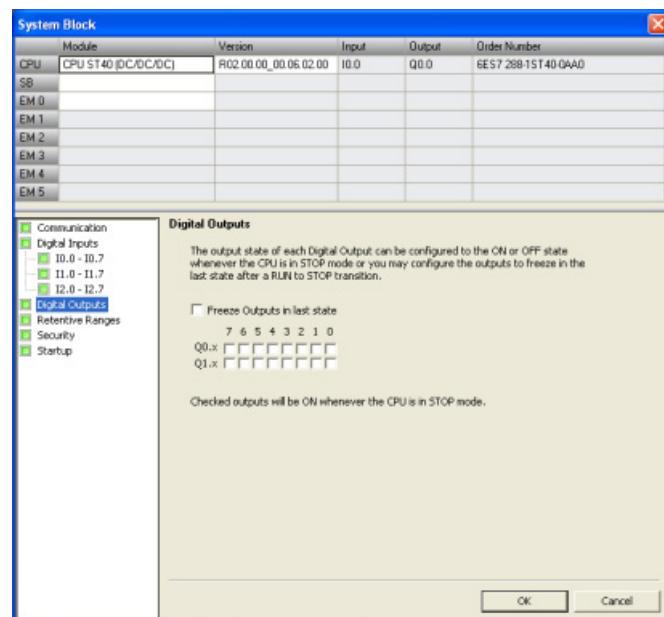


The figure below shows the response of an enabled pulse catch function to various input conditions. If you have more than one pulse in a given scan, only the first pulse is read. If you have multiple pulses in a given scan, you should use the rising/falling edge interrupt events:



6.1.4 Configuring the digital outputs

Click the Digital Outputs node of the system block (Page 107) to configure options for the digital outputs of the selected module.



6.1 Configuring the operation of the PLC system

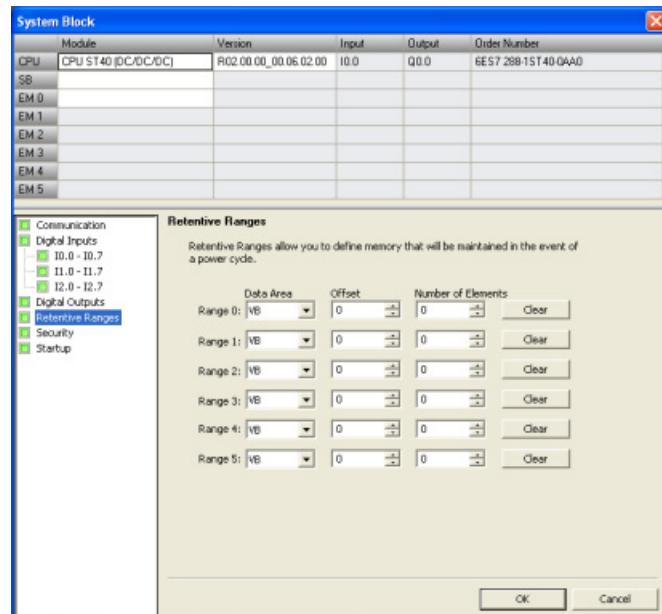
You can set digital output points to a specific value when the CPU is in STOP mode, or preserve the output states that existed before the transition to STOP mode.

You have two ways to set the digital output behavior in STOP mode:

- **Freeze Outputs in last state:** Click this checkbox to have all digital outputs frozen in their last states at the time of a RUN-to-STOP transition.
- **Substitute value:** If the Freeze Outputs in last state checkbox is not checked, this table allows you to select the desired state of each output whenever the CPU is in STOP mode. Click the checkbox for each output you want set to ON (1). The default substitute value for digital outputs is OFF (0).

6.1.5 Configuring the retentive ranges

Click the Retentive Ranges node of the system block (Page 107) dialog to configure ranges of memory that will be retained following a power cycle.



Configure the areas of memory you want to retain through power cycles. Enter new values for V, M, T, or C memory.

You can define ranges of addresses in the following memory areas to be retentive: V, M, T, and C. For timers, only the retentive timers (TONR) can be retained, and, for both timers and counters, only the current value can be retained (timer and counter bits are cleared on each power-up).

By default, no retentive areas are defined in the CPU, but you can configure the retentive ranges so that up to 10 Kbytes of memory are retentive.

Data retention after CPU power interruption

The CPU performs the following actions regarding retentive memory at power down and power up:

- **At power down:**

The CPU saves the memory ranges designated as retentive to permanent memory.

- **At power up:**

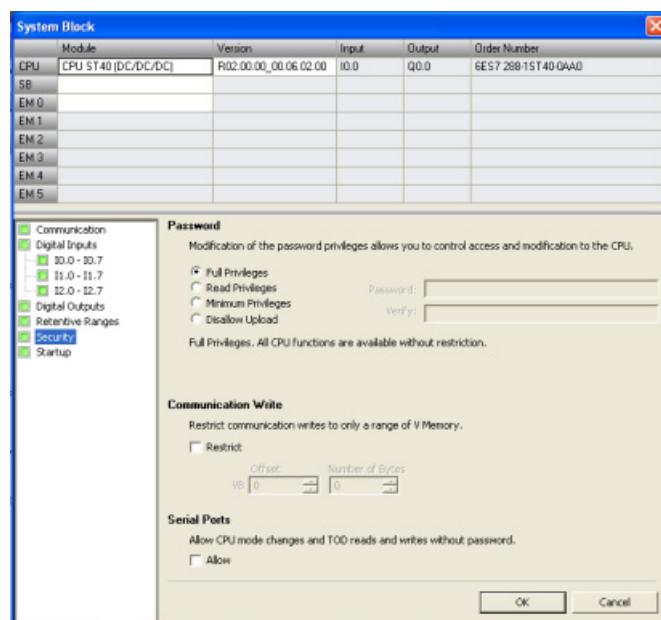
The CPU first clears V, M, C, and T memory, copies any initial values from the data block to V memory, and then copies the saved retentive values from permanent memory to RAM.

S7-200 SMART CPU memory addresses for retentive ranges

Data type	Desc.	CPU CR40	CPU CR60	CPU SR20 CPU ST20	CPU SR30 CPU ST30	CPU SR40 CPU ST40	CPU SR60 CPU ST60
V	Data Memory	VB0-VB8191	VB0-VB8191	VB0-VB8191	VB0-VB12281	VB0-VB16383	VB0-VB20479
T	Timers	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95	T0-T31, T64-T95
C	Counters	C0-C255	C0-C255	C0-C255	C0-C255	C0-C255	C0-C255
M	Flag bits	MB0-MB31	MB0-MB31	MB0-MB31	MB0-MB31	MB0-MB31	MB0-MB31

6.1.6 Configuring system security

Click the Security node of the system block (Page 107) dialog to configure a password and security settings for the CPU.



6.1 Configuring the operation of the PLC system

The password can be any combination of letters, numbers, and symbols and is case-sensitive.

Password-protected privilege levels

The CPU offers four levels of password protection, with "Full Privileges" (Level 1) providing unrestricted access and "Disallow Upload" (Level 4) providing the most restricted access. The default condition for the S7-200 SMART CPU is "Full Privileges" (Level 1).

A CPU password authorizes access to CPU functions and memory. With no CPU password downloaded ("Full privileges" (Level 1)), the S7-200 SMART CPU allows unrestricted access. If you have configured higher than "Full Privileges" (Level 1) access and downloaded a CPU password, the S7-200 SMART CPU requires password entry for access to CPU operations as defined in the table below.

The "Disallow Upload" (Level 4) password restriction protects the user program (your intellectual property) even if the password becomes known. You can never upload in Level 4 and can only change the privilege level if there is no user program present in the CPU. As a result, you can always protect your user program, even if someone discovers your password.

Table 6- 1 S7-200 SMART CPU password-protected privilege levels

Description of operation	Full privileges (Level 1)	Read privileges (Level 2)	Minimum privileges (Level 3)	Disallow upload (Level 4)
Read and write user data	Permitted	Permitted	Permitted	Permitted
Start, stop, and power-up reset of the CPU	Permitted	Restricted	Restricted	Restricted
Read the time-of-day clock	Permitted	Permitted	Permitted	Permitted
Write the time-of-day clock	Permitted	Restricted	Restricted	Restricted
Upload the user program, data, and the CPU configuration	Permitted	Permitted	Restricted	Never Permitted
Download of program block, data block, or system block	Permitted	Restricted	Restricted	Restricted Note: Never permitted for the system block if the user program block is present.
Reset to factory defaults	Permitted	Restricted	Restricted	Restricted
Delete of program block, data block, or system block	Permitted	Restricted	Restricted	Restricted Note: Never permitted for the system block if the user program block is present.

Description of operation	Full privileges (Level 1)	Read privileges (Level 2)	Minimum privileges (Level 3)	Disallow upload (Level 4)
Copy of program block, data block, or system data block to a memory card	Permitted	Restricted	Restricted	Restricted
Forcing of data in status chart	Permitted	Restricted	Restricted	Restricted
Execute single or multiple scan operations.	Permitted	Restricted	Restricted	Restricted
Writing of output in STOP mode.	Permitted	Restricted	Restricted	Restricted
Reset of scan rates in PLC information	Permitted	Restricted	Restricted	Restricted
Program status	Permitted	Restricted	Restricted	Never Permitted
Project compare	Permitted	Restricted	Restricted	Never Permitted

Communication write restrictions

You can restrict communication writes to a specific range of V memory and disallow communication writes to other memory areas (I, Q, AQ, and M). To restrict communication writes to a specific range of V memory, select the "Restrict" checkbox, and configure the range in bytes of V memory.

This area can be as small as no bytes to as large as all V memory.

With this functionality, the user program can validate the data written into this subset of memory before using it in your application for even better security. Note that these restrictions pertain only to communication writes (for example, writes from HMIs, STEP 7-Micro/WIN SMART, or PC Access), not writes from the user program.

Note

If you restrict write access to a specific range of V memory, be sure that Text Display modules or HMIs only write within the writable range of V memory. Also, if you use the PID wizard, PID control panel, motion wizard, or motion control panel be sure that the V memory that the wizards or panels use are within the writable range of V memory.

With this restriction disabled, you can write to the full ranges of memory areas, including I, Q, M, V, and AQ.

6.1 Configuring the operation of the PLC system

Serial ports mode changes and Time-of-Day (TOD) writes

You can allow, without a password, CPU mode changes (go-to-RUN, go-to-STOP) and TOD writes through the serial ports (both the RS485 built in and RS485/RS232 signal board). To do so, select the "Allow" checkbox in the "Serial Ports" section.

This checkbox provides backward compatibility with older HMIs that do not prompt for a password for these functions. The following selections are available:

- If this box is checked and the CPU is password protected, then you can change operating modes and make TOD writes with these older HMIs.
- If this box is unchecked and the CPU is password protected, you cannot change operating modes or make TOD writes with these older HMIs.
- If the CPU is not password protected, you can change operating modes and make TOD writes with these older HMIs, regardless of the setting of the checkbox.

Accessing a password-protected CPU

Note

When you enter the password for a password-protected CPU, the authorization level for that password remains effective for up to one minute after the programming device has been disconnected from the S7-200 SMART CPU. Always exit STEP 7-Micro/WIN SMART before disconnecting the cable to prevent another user from unauthorized access.

Entering the password over a network does not compromise the password protection for the S7-200 SMART CPU. If one authorized user is accessing restricted functions across a network, that does not authorize other users to access those functions. Only one user is allowed unrestricted access to the S7-200 SMART CPU at a time.

Disabling a password

You can disable the password by changing the privilege level 4, 3, or 2 to "Full privileges" (Level 1), since Level 1 allows all unrestricted CPU access.

Note

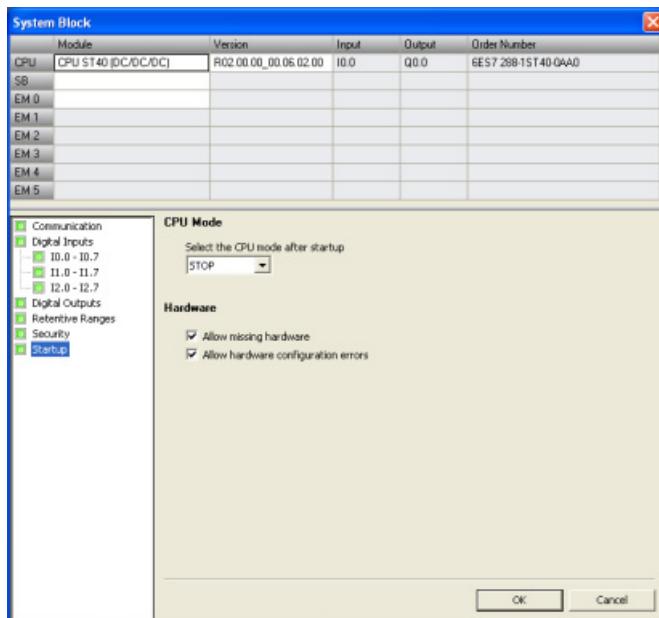
If the privilege level is at "Disallow Upload" (Level 4), you cannot download a new system block with a new password level if a valid user program exists. You must delete the user program first, and then you can download an updated system block.

What to do if you forget the password

If you forget your password, you have one option: Use the "Reset-to-factory-defaults memory card". (Refer to clear PLC memory (Page 135) for more information.)

6.1.7 Configuring the startup options

Click the Startup node of the system block (Page 107) dialog to configure startup options for the PLC.



CPU mode

From this dialog you can select the mode for the CPU after a startup. You have one of the following three choices:

- STOP

The CPU shall always enter STOP mode after a power up or restart (default selection).

- RUN

The CPU shall always enter RUN mode after a power up or restart. For most applications, especially those where the CPU operates independently without a connection to STEP 7-Micro/WIN SMART, the RUN startup mode selection is the correct choice.

- LAST

The CPU shall enter the operating mode that existed prior to the last power up or restart. This selection can be useful during program development or commissioning. Be aware that a running CPU can enter STOP mode for a variety of reasons, such as the failure of a signal module, occurrence of a scan watchdog timeout, the insertion of a memory card, or an erratic power up event. Once the CPU enters STOP mode, it will continue to enter STOP mode each time the CPU powers up. You must restore the CPU back to RUN mode (Page 33) from STEP 7-Micro/WIN SMART.

6.1 Configuring the operation of the PLC system

Hardware options

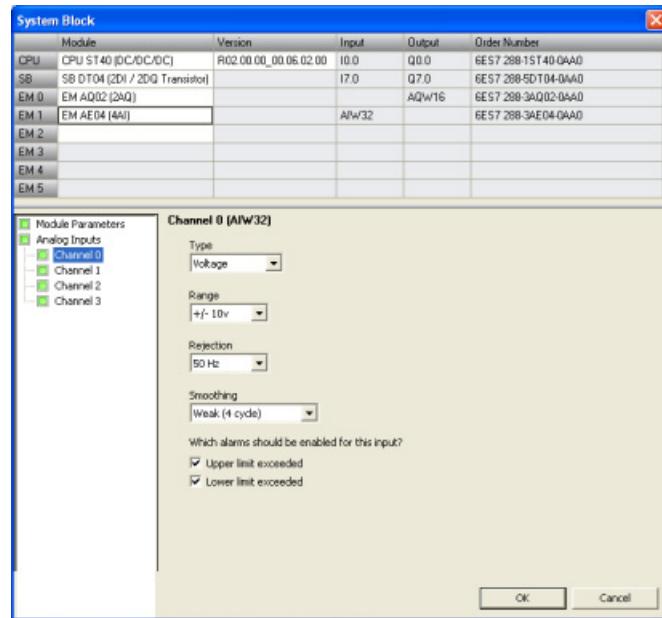
You can also configure the CPU to allow RUN mode operation under the following hardware conditions:

- One or more devices specified in the hardware configuration stored in the CPU are missing.
- A difference exists between the hardware configuration stored in the CPU and the devices actually present, resulting in configuration errors (for example, discrete input module in place of a configured discrete output module).

If you deselect one or both of the selections, the CPU is prohibited from entering RUN mode if any of the disallowed conditions are true.

6.1.8 Configuring the analog inputs

Click the Analog Inputs node of the system block (Page 107) dialog to configure options for an analog input module that you have selected in the top section.



Analog type configuration

For each analog input channel, you configure the type to be either voltage or current. The type selected for the even-numbered channels also applies to the odd-numbered channels: the type selection for Channel 0 also applies to Channel 1, and the type selection for Channel 2 also applies to Channel 3.

Range

You then configure either the voltage range or the current range for the channel. You can choose one of the following value ranges:

- +/- 2.5v
- +/- 5v
- +/- 10v
- 0 - 20ma

Rejection

Fluctuations in analog input values can also be caused by the response time of the sensor, or the length and condition of the wires carrying the analog signal to the module. In such cases, the fluctuating values could be changing too rapidly for the program logic to respond effectively. You can configure the module to reject signals to eliminate or minimize noise at the following frequencies:

- 10 Hz
- 50 Hz
- 60 Hz
- 400 Hz

Smoothing

You can also configure the module to smooth the analog input signal over a configured number of cycles, thus presenting an averaged value to the program logic. You have four choices for the smoothing algorithm:

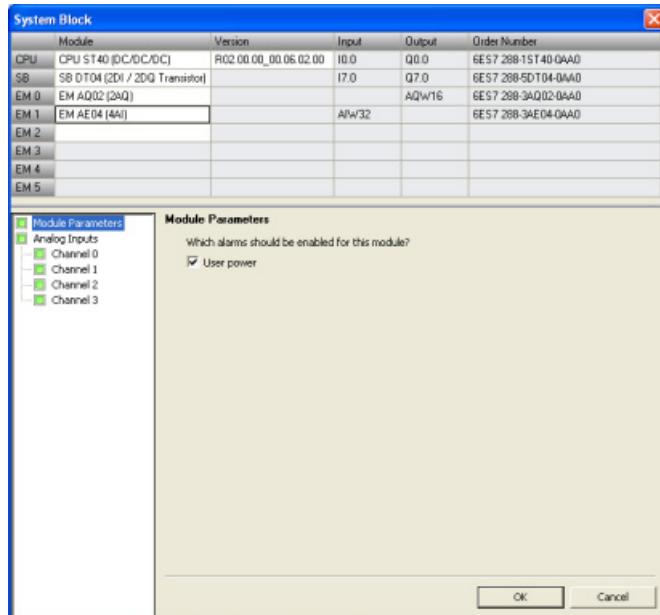
- None (no smoothing)
- Weak
- Medium
- Strong

6.1 Configuring the operation of the PLC system

Alarm configuration

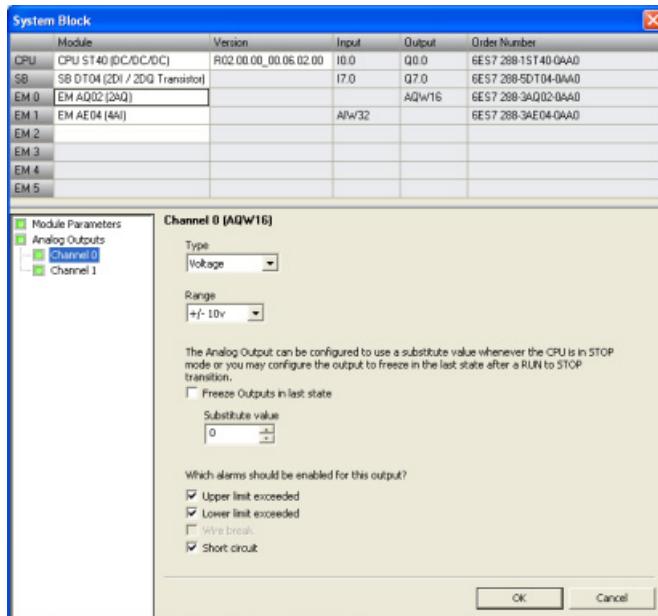
You select whether to enable or disable the following alarms for the selected channel of the selected module:

- Upper limit exceeded
- Lower limit exceeded
- User power (Configured in the system block "Module Parameters" node; see the figure below.)



6.1.9 Configuring the analog outputs

Click the Analog Outputs node of the system block (Page 107) dialog to configure options for an analog output module that you have selected in the top section.



Analog type configuration

For each analog output channel you configure the type to be either voltage or current.

Range

You then configure either the voltage range or the current range for the channel. You can choose one of the following value ranges:

- +/‐ 10v
- 0 - 20ma

Output behavior in STOP mode

You can set analog output points to a specific value when the CPU is in STOP mode or preserve the output states that existed before the transition to STOP mode.

You have two ways to set the analog output behavior in STOP mode:

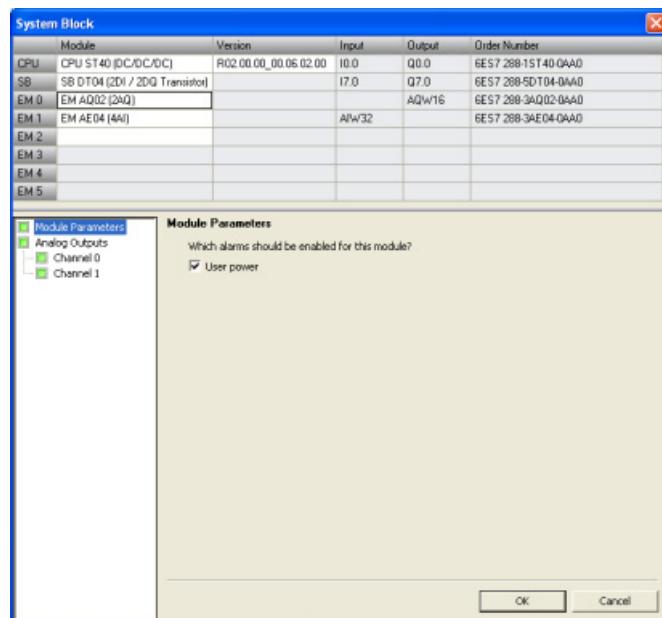
- Freeze outputs in last state: Click this checkbox to have all analog outputs frozen to their last values on a RUN-to-STOP transition.
- Substitute value: If the "Freeze outputs in last state" checkbox is not checked, you can enter a value (-32512 to 32511) that is applied to the output whenever the CPU is in STOP mode. The default substitute value is 0.

6.1 Configuring the operation of the PLC system

Alarm configuration

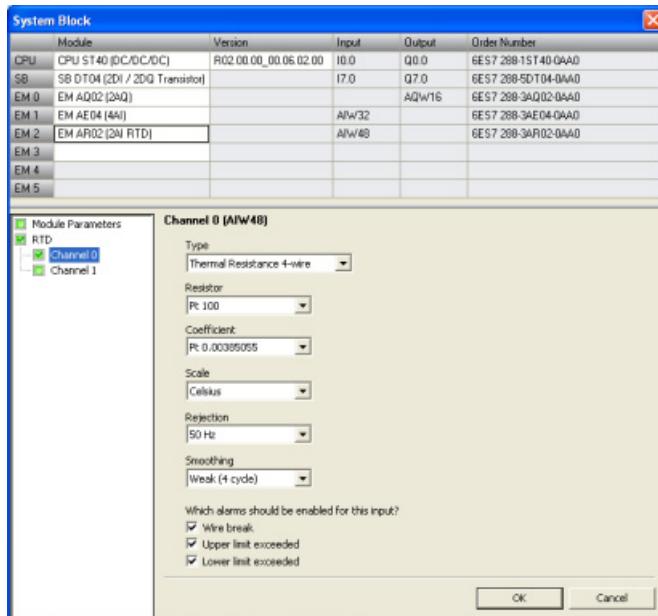
You select whether to enable or disable the following alarms for the selected channel of the selected module:

- Upper limit exceeded
- Lower limit exceeded
- Wire break (for current channels only)
- Short circuit (for voltage channels only)
- User power (Configured in the system block "Module Parameters" node; see the figure below.)



6.1.10 Configuring the RTD analog inputs

Click the RTD analog Input node of the system block (Page 107) dialog to configure options for an RTD analog input module that you have selected in the top section.



The RTD analog input module provides a current at terminals I+ and I- for resistance measurements. The current is fed to the resistance for measuring its voltage potential. The current cables must be wired directly to the resistance thermometer/resistor.

Measurements programmed for 4-or 3-wire connections compensate for line resistance and return considerably higher accuracy compared to 2-wire connections.

RTD type configuration

For each RTD input channel, you configure the type, choosing one of the following options:

- Resistance 4-wire
- Resistance 3-wire
- Resistance 2-wire
- Thermal Resistance 4-wire
- Thermal Resistance 3-wire
- Thermal Resistance 2-wire

6.1 Configuring the operation of the PLC system**Resistor**

Depending upon the RTD type that you select, you can configure the following RTD resistors for the channel:

Table 6- 2 RTD types and available resistors

RTD types	RTD resistors	
<ul style="list-style-type: none"> • Resistance 4-wire • Resistance 3-wire • Resistance 2-wire <p>Note: For these RTD types and resistors, you cannot configure temperature coefficients or temperature scales.</p>	<ul style="list-style-type: none"> • 48 ohms • 150 ohms • 300 ohms • 600 ohms • 3000 ohms 	
<ul style="list-style-type: none"> • Thermal Resistance 4-wire • Thermal Resistance 3-wire • Thermal Resistance 2-wire 	<ul style="list-style-type: none"> • Pt 10 • Pt 50 • Pt 100 • Pt 200 • Pt 500 • Pt 1000 • LG-Ni 1000 	<ul style="list-style-type: none"> • Ni 100 • Ni 120 • Ni 200 • Ni 500 • Ni 1000 • Cu 10 • Cu 50 • Cu 100

Coefficient

Depending upon the RTD resistor that you select, you can configure the following RTD temperature coefficients for the channel:

RTD resistors	RTD temperature coefficients
<ul style="list-style-type: none"> • 48 ohms • 150 ohms • 300 ohms • 600 ohms • 3000 ohms 	<p>Note: For these RTD resistors, you cannot configure temperature coefficients or temperature scales.</p>
<ul style="list-style-type: none"> • Pt 10 • Pt 50 	<ul style="list-style-type: none"> • Pt 0.00385055 • Pt 0.003910
<ul style="list-style-type: none"> • Pt 100 • Pt 500 	<ul style="list-style-type: none"> • Pt 0.00385055 • Pt 0.003916 • Pt 0.003902 • Pt 0.003920 • Pt 0.003910

RTD resistors	RTD temperature coefficients
<ul style="list-style-type: none"> Pt 200 Pt 1000 	<ul style="list-style-type: none"> Pt 0.00385055 Pt 0.003916 Pt 0.003902 Pt 0.003920
<ul style="list-style-type: none"> Ni 100 	<ul style="list-style-type: none"> Ni 0.006170 Ni 0.006180 Ni 0.006720
<ul style="list-style-type: none"> Ni 120 Ni 200 Ni 500 Ni 1000 	<ul style="list-style-type: none"> Ni 0.006180 Ni 0.006720
<ul style="list-style-type: none"> Cu 10 	<ul style="list-style-type: none"> Cu 0.00426 Cu 0.00428 Cu 0.00427
<ul style="list-style-type: none"> Cu 50 Cu 100 	<ul style="list-style-type: none"> Cu 0.00426 Cu 0.00428
<ul style="list-style-type: none"> LG-Ni 1000 	<ul style="list-style-type: none"> LG-Ni 0.005000

Scale

You configure a temperature scale for the channel, choosing one of the following options:

- Celsius
- Fahrenheit

Note

For the "Resistance 4-wire", "Resistance 3-wire", and "Resistance 2-wire" RTD types and associated resistors, you cannot configure temperature coefficients or temperature scales.

6.1 Configuring the operation of the PLC system

Rejection

Fluctuations in RTD analog input values can also be caused by the response time of the sensor, or the length and condition of the wires carrying the RTD analog signal to the module. In such cases, the fluctuating values could be changing too rapidly for the program logic to respond effectively. You can configure the module to reject signals to eliminate or minimize noise at the following frequencies:

- 10 Hz
- 50 Hz
- 60 Hz
- 400 Hz

Smoothing

You can also configure the module to smooth the RTD analog input signal over a configured number of cycles, thus presenting an averaged value to the program logic. You have four choices for the smoothing algorithm:

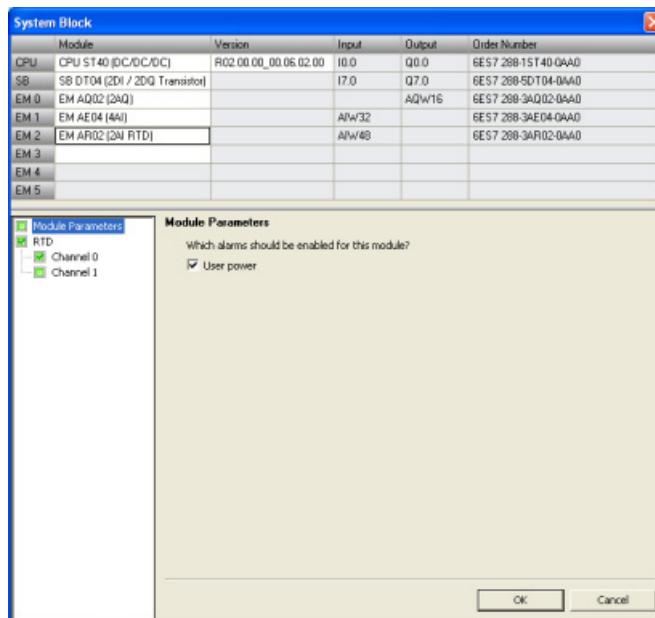
- None
- Weak
- Medium
- Strong

Alarm configuration

You select whether to enable or disable the following alarms for the selected channel of the selected RTD module:

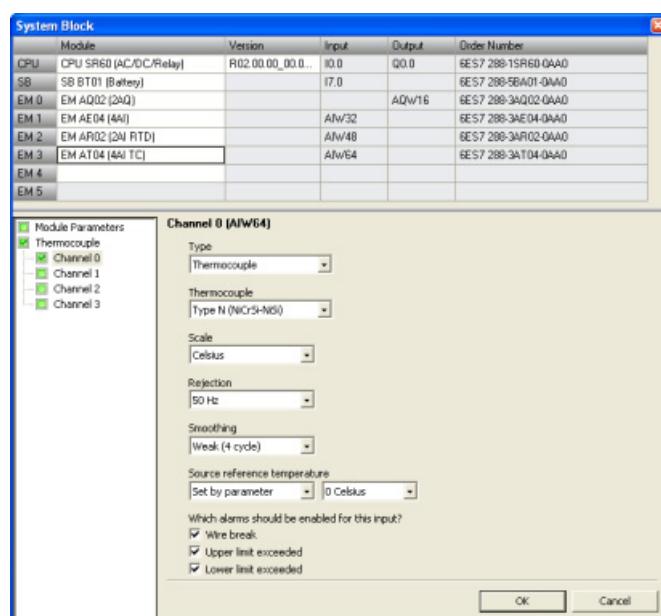
- Wire break
- Upper limit exceeded
- Lower limit exceeded

- User power (Configured in the system block "Module Parameters" node; see the figure below.)



6.1.11 Configuring the TC analog inputs

Click the TC (Thermocouple) analog input node of the system block (Page 107) dialog to configure options for a TC analog input module that you have selected in the top section.



The TC analog signal module measures the value of voltage connected to the module inputs.

6.1 Configuring the operation of the PLC system

Thermocouple type configuration

For each TC analog input module channel, you configure the type, choosing one of the following options:

- Thermocouple
- Voltage

Thermocouple

Depending upon the thermocouple type that you select, you can configure the following thermocouples for the channel:

- Type B (PtRh-PtRh)
- Type N (NiCrSi-NiSi)
- Type E (NiCr-CuNi)
- Type R (PtRh-Pt)
- Type S (PtRh-Pt)
- Type J (Fe-CuNi)
- Type T (Cu-CuNi)
- Type K (NiCr-Ni)
- Type C (W5Re-W26Re)
- TXK/XK (TXK/XK(L))

Scale

You configure a temperature scale for the channel, choosing one of the following options:

- Celsius
- Fahrenheit

Rejection

Fluctuations in thermocouple analog input values can also be caused by the response time of the sensor, or the length and condition of the wires carrying the thermocouple analog signal to the module. In such cases, the fluctuating values could be changing too rapidly for the program logic to respond effectively. You can configure the TC analog input module to reject signals to eliminate or minimize noise at the following frequencies:

- 10 Hz
- 50 Hz
- 60 Hz
- 400 Hz

Smoothing

You can also configure the module to smooth the thermocouple analog input signal over a configured number of cycles, thus presenting an averaged value to the program logic. You have four choices for the smoothing algorithm:

- None
- Weak
- Medium
- Strong

Source reference temperature

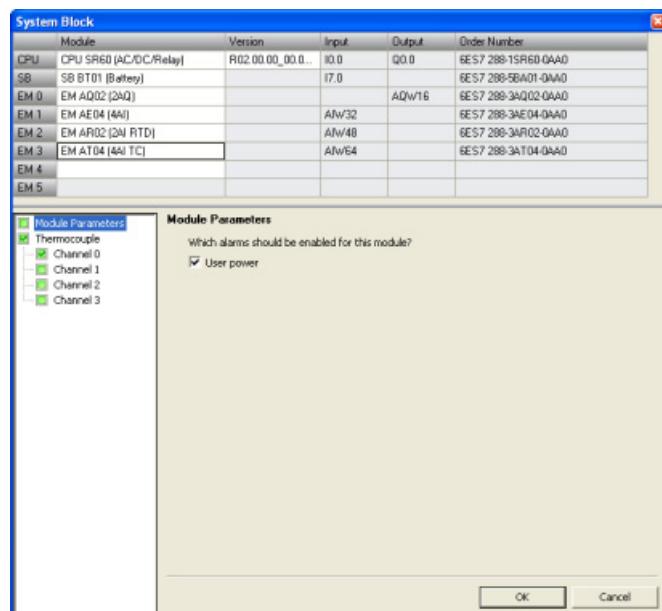
You configure a source reference temperature for each TC analog input module channel, choosing one of the following options:

- Set by parameter
- Internal reference

Alarm configuration

You select whether to enable or disable the following alarms for the selected channel of the selected TC analog input module:

- Wire break
- Upper limit exceeded
- Lower limit exceeded
- User power (Configured in the system block "Module Parameters" node; see the figure below.)



6.1 Configuring the operation of the PLC system

Basic operation of a thermocouple

Thermocouples are formed whenever two dissimilar metals are electrically bonded to each other. A voltage is generated that is proportional to the junction temperature. This voltage is small; one microvolt could represent many degrees. Measuring the voltage from a thermocouple, compensating for extra junctions, and then linearizing the result forms the basis of temperature measurement using thermocouples.

When you connect a thermocouple to the TC analog input module, the two dissimilar metal wires are attached to the module at the module signal connector. The place where the two dissimilar wires are attached to each other forms the sensor thermocouple.

Two more thermocouples are formed where the two dissimilar wires are attached to the signal connector. The connector temperature causes a voltage that adds to the voltage from the sensor thermocouple. If this voltage is not corrected, then the temperature reported will deviate from the sensor temperature.

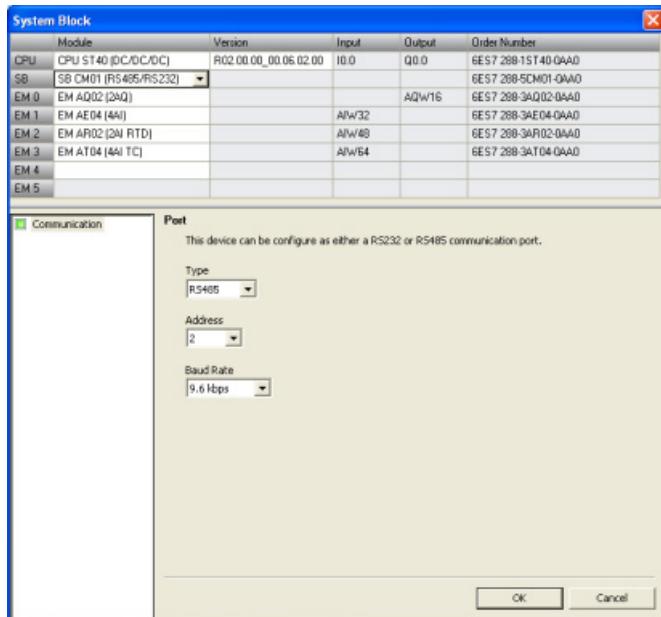
Cold junction compensation is used to compensate for the connector thermocouple. Thermocouple tables are based on a reference junction temperature, usually zero degrees Celsius. The cold junction compensation compensates the connector to zero degrees Celsius. The cold junction compensation restores the voltage added by the connector thermocouples. The temperature of the module is measured internally, and then converted to a value to be added to the sensor conversion. The corrected sensor conversion is then linearized using the thermocouple tables.

For optimum operation of the cold junction compensation, the thermocouple module must be located in a thermally stable environment. Slow variation (less than 0.1° C/minute) in ambient module temperature is correctly compensated within the module specifications. Air movement across the module will also cause cold junction compensation errors.

If better cold junction error compensation is needed, an external iso-thermal terminal block may be used. The thermocouple module provides for use of a 0° C referenced or 50° C referenced terminal block.

6.1.12 Configuring the RS485/RS232 CM01 communications signal board

Click the CM01 communications signal board node of the system block (Page 107) dialog to configure options for an RS485/RS232 CM01 communications signal board that you have selected in the top section.



CM01 signal board type configuration

You configure the CM01 signal board type from the dropdown list, choosing one of the following options:

- RS485
- RS232

Address

Click the scroll buttons to enter the desired port address (1-126).for the RS485 or RS232 port: The default port address is 2.

Baud rate

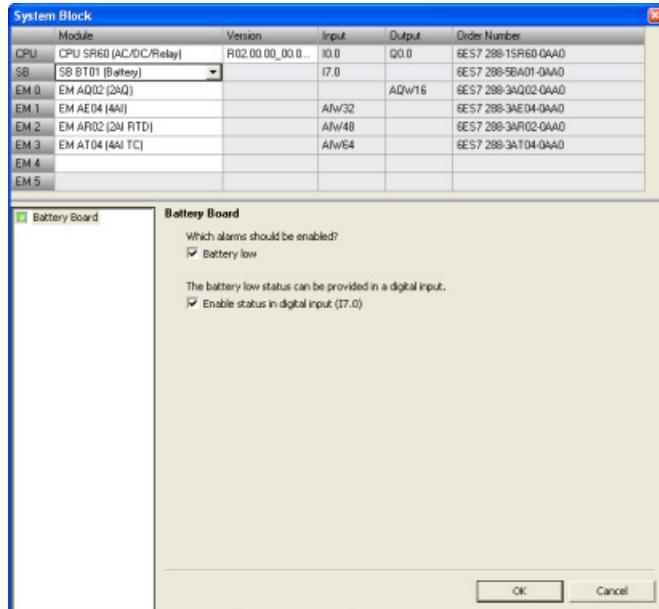
Choose the desired data baud rate from the dropdown list:

- 9.6 kbps
- 19.2 kbps
- 187.5 kbps

6.1 Configuring the operation of the PLC system

6.1.13 Configuring the BT01 battery signal board

Click the BT01 battery signal board node of the system block (Page 107) dialog to configure options for a BT01 battery signal board that you have selected in the top section.



Enable bad diagnostic alarm

Click the "Enable bad diagnostic alarm" checkbox to trigger an alarm when the battery fails.

Enable status in digital input

Click the "Enable status in digital input" to enable a digital input to monitor the status of the signal board.

Operation of the Battery (BT01) Signal Board

The battery signal board contains a red LED that provides the customer a visual indication of the battery health. An illuminated LED indicates a battery low condition.

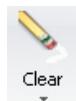
The CPU automatically performs the RTC swap-over, battery test, and battery health LED operation, whether or not the System block contains a configuration for the signal board.

The battery signal board System block configuration contains selections that allow the customer to report the battery low state as a diagnostic alarm and/or to report the battery state (1=battery low, 0 = battery OK) in the LSBit of the configured image register input byte for the device (for example, I7.0). The customer must select the battery signal board in the System block configuration in order to gain access to the additional battery health reporting options.

6.1.14 Clearing PLC memory

To clear designated areas of PLC memory, follow these steps:

1. Ensure that the PLC is in STOP mode.
2. Click the Clear button from the Modify area of the PLC menu ribbon strip.



⚠ WARNING

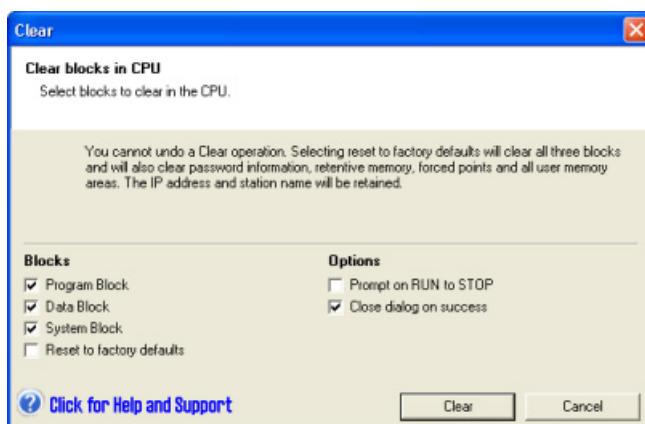
Effect of clearing PLC memory on outputs

Clearing the PLC memory affects the state of digital and analog outputs. The default is for digital and analog outputs to use a substitute value of 0. If you have defined substitute values other than 0 or chosen "Freeze" for your digital or analog outputs, you need to be aware that when you delete the system block, you are deleting the substitute and freeze information and, as a result, your outputs shall return to the default value of 0. Furthermore, if you perform a selective clear such that you keep your system block but delete your program block, then your analog outputs are frozen at their current value. Until you download a new program block, the only way to make changes to the state of the analog outputs is by means of the status chart.

If the S7-200 SMART PLC is connected to equipment when you clear the PLC memory, changes to the state of the digital outputs can be transmitted to the equipment. If you clear PLC memory without planning for the consequences to your digital and analog outputs, your equipment could operate in an unpredictable fashion, which could result in death or serious injury to personnel and/or damage to equipment.

Always follow appropriate safety precautions and ensure that your process is in a safe state before clearing the PLC memory.

3. Select what to clear - Program Block, Data Block, System Block, or all blocks, or select "Reset to factory defaults".
4. Click the Clear button.



Clearing the PLC memory requires the PLC to be in STOP mode and then deletes the selected blocks or resets the PLC to the factory-set defaults, depending on your selection. A clear operation does not clear the IP address, station name, or reset the time-of-day clock.

6.1 Configuring the operation of the PLC system

When executed, the "Reset to factory defaults" setting deletes all blocks, resets all user memory to the initial powerup state, and resets all Special Memory to initial values.

What to do if you forget the PLC password

If you forget the PLC password (Page 115), you must clear the PLC memory from a reset-to-factory-defaults memory card (Page 137) that you have made for this purpose.



Inserting a memory card into a CPU

Inserting a memory card into a CPU in RUN mode causes the CPU to automatically transition to STOP mode. You cannot change the CPU to RUN mode if a memory card is inserted.

Inserting a memory card into a running CPU can cause disruption to process operation, possibly resulting in death or severe personal injury.

Always ensure that the CPU is in STOP mode prior to inserting a memory card.

To clear the PLC using this card follow this steps:

1. Insert the reset-to-factory-defaults memory card. The CPU goes to STOP mode and flashes the STOP LED.
2. Power cycle the CPU. The CPU flashes the RUN/STOP LEDs until the reset is complete (about one second), and then flashes the STOP LED indicating that the reset is finished.
3. Remove the memory card.
4. Power cycle the CPU. The CPU is reset to the factory defaults. The former IP address and baud rate settings are cleared, but the time-of-day clock is unaffected.

After the CPU is reset, you can assign a new password and begin programming, or load a program from another program transfer memory card (Page 75) or from your hard disk.

Note

If you load a password-protected program from a memory card or file on your hard disk, you must enter the password to access the protected areas. You cannot access a password-protected program component without a password, nor can you clear an assigned password without entry of the password.

6.1.15 Creating a reset-to-factory-defaults memory card

You can create a memory card that will return the CPU to a factory default state. You can use this reset-to-factory-defaults memory card if you ever want to clear the contents of your CPU. To create a reset-to-factory-defaults memory card, follow these steps:

1. Using a card reader and Windows explorer, delete all contents from a microSDHC card.
2. Create a simple text file with an editor such as Notepad that contains one line with the string "RESET_TO_FACTORY". (Do not enter the quotation marks.)
3. Save this file to the microSDHC card root level under the file name "S7_JOB.S7S".
4. Label the card and store it in a safe place for future use.

6.2 High-speed I/O

High-speed counters

The CPU provides integrated high-speed counter functions that count high speed external events without degrading the performance of the CPU. Refer to the "Product overview" (Page 15) chapter for the rates supported by your CPU. Dedicated inputs exist for clocks, direction control, and reset, where these functions are supported. You can select single phase, dual phase, or AB quadrature phase for varying the counting rate. For more information, refer to the description of the high-speed counter instructions (Page 210).

Pulse Width Modulated (PWM) outputs

The standard CPU models support a pulse width modulated (PWM) output capability on certain outputs. Refer to the "Product overview" (Page 15) chapter for the quantity and rates supported by your CPU.

The PWM function provides a fixed cycle time with a variable duty cycle output, with the cycle time and the pulse width specified in either microsecond or millisecond increments. When the pulse width is equal to the cycle time, the duty cycle is 100 percent and the output is turned on continuously. When the pulse width is zero, the duty cycle is 0 percent and the output is turned off.

Refer to the high-speed pulse output instruction (Page 236) for more information. The chapter on open-loop motion control provides additional information using PWM (Page 437).

Open-loop motion control

The standard CPU models support an open-loop motion control capability. Motion profiles can be constructed and executed, interactive movement can be performed under user program control, and a number of built-in reference point seek sequences are available.

Depending upon configuration, open-loop motion support in the CPU requires the use of certain CPU resources, such as high-speed outputs, high-speed counters, and edge interrupts.

Refer to the "Product overview" (Page 15) chapter for the quantity of motion axes and pulse rates supported by your CPU.

Refer to the chapter on open-loop motion control (Page 437) for a full description of the motion capabilities in your CPU.

Program instructions

7.1 Bit logic

7.1.1 Standard inputs

LAD	FBD	STL	Description
		LD bit A bit O bit	<p>LAD: Normally open and normally closed switches are represented by a contact symbol. If power flow is present on the left-side and the contact is closed, then power flows through the contact to the right-side connector and to the next connected element.</p> <ul style="list-style-type: none"> The Normally Open (N.O.) LAD contact is closed (ON) when the bit is equal to 1. The Normally Closed (N.C.) LAD contact is closed (ON) when the bit is equal to 0.
		LDN bit AN bit ON bit	<p>FBD: Normally open instructions are represented by AND/OR boxes. Box instructions can be used to evaluate Boolean signals in the same manner as ladder contact networks. Normally closed instructions are also represented by boxes. A normally closed instruction is created by placing the negation circle on a binary input signal connector. The number of inputs for the AND/OR boxes can be expanded to a maximum of 31 inputs.</p> <p>STL: The normally open contact is represented by the LD (load), A (AND), and O (OR) instructions. These instructions load, AND, or OR the value of the addressed bit with the top bit of the logic stack. The normally closed contact is represented by the LDN (Load NOT), AN (AND NOT), and ON (OR NOT) instructions. These instructions load, AND, or OR the logical NOT of the addressed bit value with the top bit of the logic stack.</p>

Input / output	Data type	Operand
bit (LAD, STL)	BOOL	I, Q, V, M, SM, S, T, C, L,
Input (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
Output (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

FBD AND/OR input assignment

The editor feature described in the following table is active only if an input stub is selected and colored red, inside the FBD box cursor.

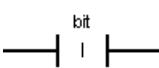
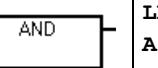
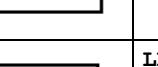
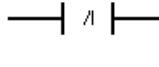
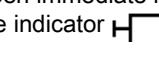
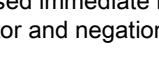
Input option	Place cursor	Tool button	Shortcut key
Add input	On box		+
Remove input	On box and bottom input		-

See also

[Bit logic input examples \(Page 152\)](#)

[Logic stack overview \(Page 142\)](#)

7.1.2 Immediate inputs

LAD	FBD	STL	Description
	 	LDI bit AI bit OI bit	<p>The immediate instruction obtains the physical input value when the instruction is executed, but the process-image register is not updated. An immediate contact does not wait on the PLC scan cycle to update; it updates immediately.</p> <p>The Normally Open Immediate contact is closed (ON) when the physical input point (bit) state is 1.</p>
	 	LDNI bit ANI bit ONI bit	<p>The Normally Closed Immediate contact is closed (ON) when the physical input point (bit) state is 0.</p> <p>LAD: Normally open and normally closed immediate instructions are represented by contacts.</p> <p>FBD: Normally open immediate instructions are represented by the vertical immediate indicator  in front of an input connection.</p> <p>The Normally closed immediate instruction is represented by the immediate indicator and negation circle  in front of an input connection.</p> <p>The immediate indicator cannot be used when a logic flow connection is used instead of a physical input (I) bit address.</p> <p>FBD box instructions can be used to evaluate physical signals in the same manner as ladder contacts. The number of inputs for the AND/OR boxes can be expanded to a maximum of 31 inputs.</p> <p>STL: The Normally Open Immediate contact is represented by the LDI (Load Immediate), AI (AND Immediate), and OI (OR Immediate) instructions. These instructions load, AND, or OR the physical input value with the top of the logic stack.</p> <p>A normally Closed Immediate contact is represented by the LDNI (Load NOT Immediate), ANI (AND NOT Immediate), and ONI (OR NOT Immediate) instructions. These instructions immediately load, AND, or OR the logical NOT of the value of the physical input value with the top of the logic stack.</p>

Input / output	Data type	Operand
bit (LAD, STL)	BOOL	I
Input (FBD)	BOOL	I

FBD editor input assignment

The editor feature described in the following table is active only if an input stub is selected and colored red, inside the FBD box cursor.

Input option	Place cursor	Tool button	Shortcut key
Add input	On box		+
Remove input	On box and bottom input		-
Toggle negate input	On box and input		F11
Toggle immediate input	On box and input		CTRL F11

See also

[Bit logic input examples \(Page 152\)](#)

[Logic stack overview \(Page 142\)](#)

7.1.3 Logic stack overview

The STEP 7-Micro/WIN SMART program compiler uses the logic stack to transform the graphical I/O networks of LAD and FBD programs into STL (statement list) programs. The resultant STL program is logically the same as the original LAD or FBD graphical network and can be executed as a program list. All successfully compiled LAD and FBD programs have generated the underlying STL program and can be viewed as LAD, FBD, or STL.

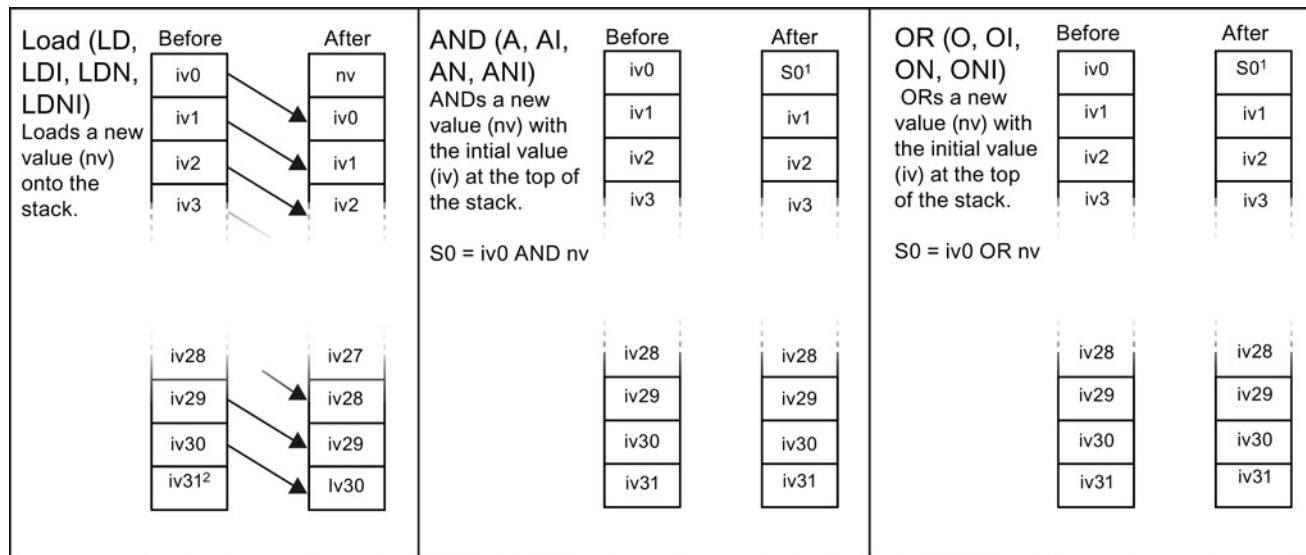
For LAD and FBD editing, the STL logic stack instructions are automatically generated and the programmer does not need to use the logic stack instructions.

You can also create STL programs directly with the STL editor. An STL programmer uses the logic stack instructions directly. Combination logic can be created in the STL editor that is too complex to be viewed in the LAD or FBD editor, but may be necessary for special applications.

All successfully compiled LAD and FBD programs can be viewed in STL, but not all successfully compiled STL programs can be viewed in LAD or FBD.

Input networks and the logic stack

As shown in the following figure, the CPU uses a logic stack to combine the logic states of STL inputs. In these examples, "iv0" to "iv31" identify the initial values of the logic stack levels, "nv" identifies a new value provided by the instruction, and "S0" identifies the calculated value that is stored in the logic stack.



¹ S0 identifies the calculated value that is stored in the logic stack.

² After the execution of a Load, the value iv31 is lost.

Output networks and the logic stack

ENO is a binary output for boxes in LAD and FBD. If a LAD box has power flow at the EN input and is executed without error, the ENO output passes power flow to the next LAD element. You can use the ENO as an enable bit that indicates the successful completion of an instruction. The ENO bit is used with the top of stack to affect power flow for execution of subsequent instructions. STL instructions do not have an EN input. The top of the stack must have a value of logic 1 for conditional instructions to be executed. In STL there is no ENO output. However, the STL instructions that correspond to LAD and FBD instructions with ENO outputs set a special ENO bit. This bit is accessible with the AND ENO (AENO) instruction.

STL	Description
AENO	AENO is used in the STL representation of LAD/FBD box ENO bit. AENO performs a logical AND of the ENO bit with the top of stack for the same effect as the ENO bit of a LAD/FBD box. The result of the AND operation is the new top of stack value.

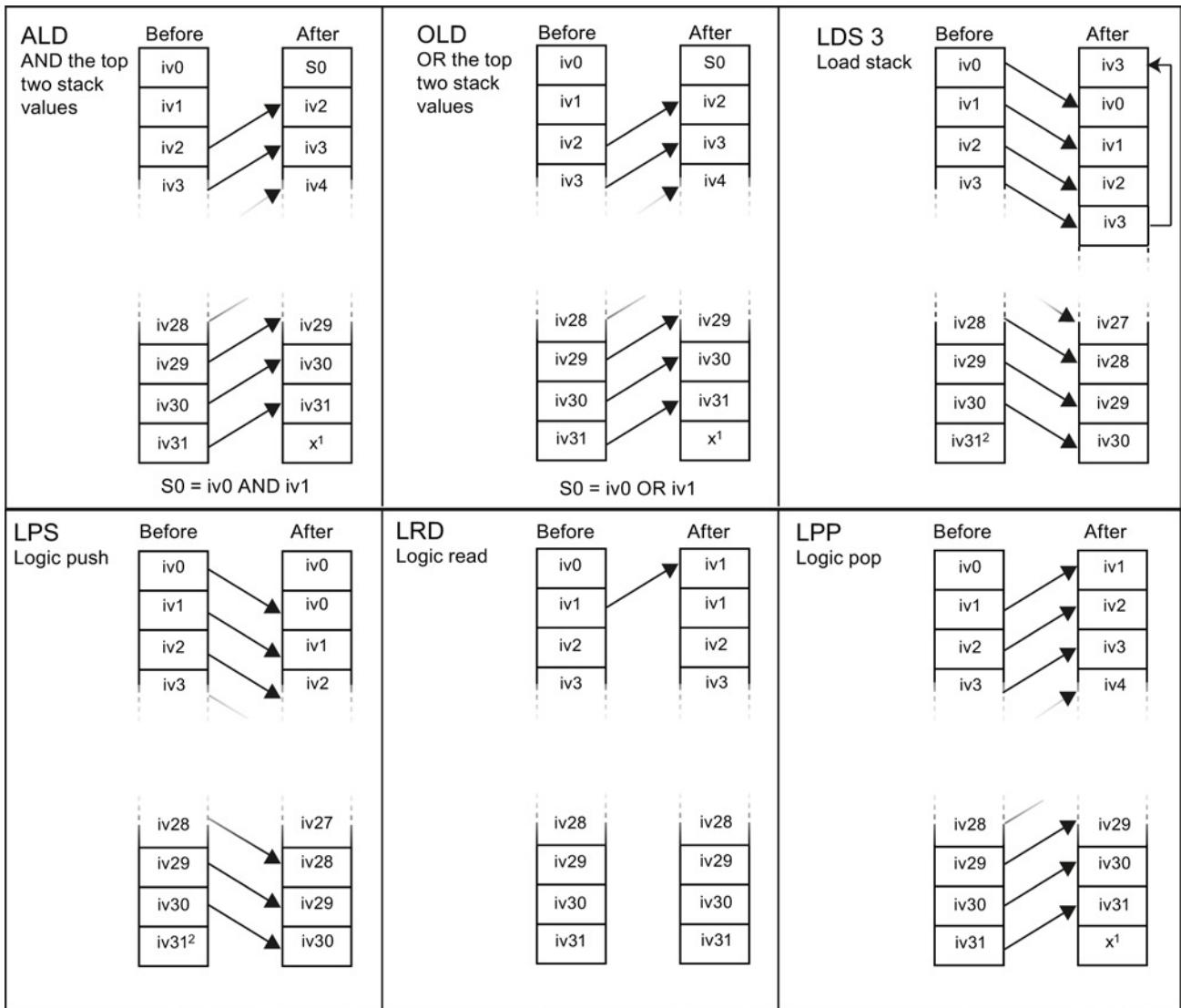
7.1.4 STL logic stack instructions

STL ¹	Description
ALD	The AND Load instruction (ALD) combines the values in the first and second levels of the stack using a logical AND operation. The result is loaded in the top of stack. After the ALD is executed, the stack depth is decreased by one.
OLD	The OR Load instruction (OLD) combines the values in the first and second levels of the stack, using a logical OR operation. The result is loaded in the top of the stack. After the OLD is executed, the stack depth is decreased by one.
LPS	The Logic Push instruction (LPS) duplicates the top value on the stack and pushes this value onto the stack. The bottom of the stack is pushed off and lost.
LRD	The Logic Read instruction (LRD) copies the second stack value to the top of stack. The stack is not pushed or popped, but the old top-of-stack value is destroyed by the copy.
LPP	The Logic Pop instruction (LPP) pops one value off of the stack. The second stack value becomes the new top of stack value.
LDS N	The Load Stack instruction (LDS) duplicates the stack bit (N) on the stack and places this value on top of the stack. The bottom of the stack is pushed off and lost.
AENO	AENO is used in the STL representation of the LAD/FBD box ENO bit. AENO performs a logical AND of the ENO bit with the top of stack for the same effect as the ENO bit of a LAD/FBD box. The result of the AND operation is the new top of stack.

¹ Not applicable for LAD or FBD

LDS (Load Stack) Input	Data type	Operands
N	BYTE	Constant (0 to 31)

As shown in the following figure, the CPU uses a logic stack to resolve the control logic. In these examples, "iv0" to "iv31" identify the initial values of the logic stack, "nv" identifies a new value provided by the instruction, and "S0" identifies the calculated value that is stored in the logic stack.



¹ The value is unknown (it could be either a 0 or a 1).

² After the execution of a Logic push or a Load stack instruction, value iv31 is lost.

Program instructions

7.1 Bit logic

Logic Stack example: LAD networks transformed into STL code

LAD	STL
	Network 1 LD I0.0 LD I0.1 LD I2.0 A I2.1 OLD ALD = Q5.0
	Network 2 LD I0.0 LPS LD I0.5 O I0.6 ALD = Q7.0 LRD LD I2.1 O I1.3 ALD = Q6.0 LPP A I1.0 = Q3.0

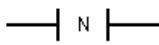
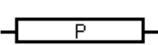
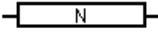
7.1.5 NOT

LAD	FBD	STL	Description
		NOT	<p>The Not instruction (NOT) inverts the state of the power flow input.</p> <p>LAD: The NOT contact changes the state of power flow input. When power flow reaches the NOT contact, it stops. When power flow does not reach the NOT contact, it supplies power flow.</p> <p>FBD: The NOT instruction is represented as a graphical negation (bubble) symbol on Boolean box input connectors and functions as a logic state inverter.</p> <p>STL: The NOT instruction changes the value on the top of the stack from 0 to 1, or from 1 to 0.</p>

See also

Bit logic input examples (Page 152)

7.1.6 Positive and negative transition detectors

LAD	FBD	STL	Description
 	 	EU ED	<p>The positive transition contact instruction (Edge Up) allows power to flow for one scan for each OFF-to-ON transition.</p> <p>The negative transition contact instruction (Edge Down) allows power to flow for one scan for each ON-to-OFF transition.</p> <p>S7-200 SMART CPUs support a combined total (positive and negative) of 1024 edge detector instructions in your program.</p> <p>LAD: Positive and negative transition instructions are represented by contacts.</p> <p>FBD: The transition instructions are represented by the P and N boxes.</p> <p>STL: The positive transition is detected by the EU (Edge Up) instruction. Upon detection of a 0-to-1 transition in the value on the top of the stack, the top of the stack value is set to 1; otherwise, it is set to 0.</p> <p>The negative transition is detected by the ED (Edge Down) instruction. Upon detection of a 1-to-0 transition in the value on the top of the stack, the top of the stack value is set to 1; otherwise, it is set to 0.</p>

Input / output	Data type	Operand
IN (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
OUT (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

Note

Because the Positive Transition and Negative Transition instructions require an on-to-off or an off-to-on transition, you cannot detect an edge-up or edge-down transition on the first scan. During the first scan, the CPU saves the initial input state in a memory bit. On subsequent scans, these instructions compare the current state and the state of the memory bit, to detect a transition.

See also

Bit logic input examples (Page 152)

7.1.7 Coils: output and output immediate instructions

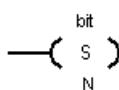
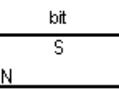
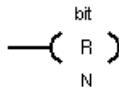
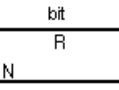
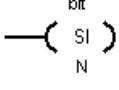
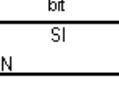
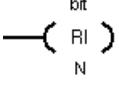
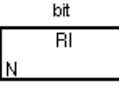
LAD	FBD	STL	Description
		= bit	<p>The Output instruction writes the new value for the output bit to the process-image register.</p> <p>LAD and FBD: When the output instruction is executed, the S7-200 turns the output bit in the process-image register ON or OFF. The assigned bit is set equal to power flow state.</p> <p>STL: The value on the top of the stack is copied to the assigned bit.</p>
		=I bit	<p>The Output Immediate instruction writes the new value to both the physical output and the corresponding process-image register location when the instruction is executed.</p> <p>LAD and FBD: When the output immediate instruction is executed, the physical output point (bit) is immediately set equal to power flow. The "I" indicates an immediate reference; the new value is written to both the physical output point and the corresponding process-image register address. This differs from the non-immediate references, which only write the new value to the process-image register.</p> <p>STL: The instruction immediately copies the value on the top of the stack to the assigned physical output bit and process-image address.</p>

Input / output	Data type	Operand
Bit	BOOL	I, Q, V, M, SM, S, T, C, L
Bit (immediate)	BOOL	Q
Input (LAD)	BOOL	Power flow
Input (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

See also

Bit logic output examples (Page 153)

7.1.8 Set, reset, set immediate, and reset immediate functions

LAD	FBD	STL	Description
		S bit, N	The Set (S) and Reset (R) instructions set (ON) or reset (OFF) the number of bits (N), starting at the address (bit). You can set or reset from 1 to 255 bits. If the Reset instruction specifies either a timer bit (T address) or counter bit (C address), the instruction resets the timer or counter bit and clears the current value of the timer or counter.
		R bit, N	
		SI bit, N	The Set Immediate and Reset Immediate instructions immediately set (ON) or immediately reset (OFF) the number of points (N), starting at address (bit). You can set or reset from 1 to 255 points immediately. The "I" indicates an immediate reference; when the instruction is executed, the new value is written to both the physical output point and the corresponding process-image register location. This differs from the non-immediate references, which write the new value to the process-image register only.
		RI bit, N	

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • N = 0 (zero) • 0006H Indirect address • 0091H Operand out of range 	None

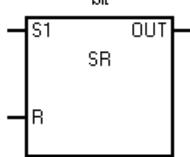
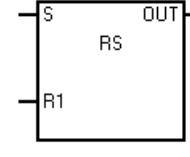
Input / output	Data type	Operand
Bit	BOOL	I, Q, V, M, SM, S, T, C, L
Bit (immediate)	BOOL	Q
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, Constant, *VD, *AC, *LD
Input (LAD)	BOOL	Power flow
Input (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

See also

[Bit logic input examples \(Page 152\)](#)

[Bit logic output examples \(Page 153\)](#)

7.1.9 Set and reset dominant bistable

LAD/FBD ¹	Description
	The bit parameter assigns the Boolean address that is set or reset. The optional OUT connection reflects the signal state of the Bit parameter. SR (Set dominant bistable) is a latch where the set dominates. If the set (S1) and reset (R) signals are both true, the output (OUT) is true.
	RS (Reset dominant bistable) is a latch where the reset dominates. If the set (S) and reset (R1) signals are both true, the output (OUT) is false.

¹ Not applicable for STL

Input / outputs	Data type	Operand
bit	BOOL	I, Q, V, M, S
S1, R (LAD SR)	BOOL	Power flow
S, R1 (LAD RS)	BOOL	Power flow
OUT (LAD)	BOOL	Power flow
S1, R (FBD SR)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
S, R1 (FBD RS)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
OUT (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow

SR truth table

S1	R	Out (bit)
0	0	Previous state
0	1	0
1	0	1
1	1	1

RS truth table

S	R1	Out (bit)
0	0	Previous state
0	1	0
1	0	1
1	1	0

Example SR and RS

LAD	STL
	NETWORK 1 LD I0.0 LD I0.1 NOT A Q0.0 OLD = Q0.0
	NETWORK 2 LD I0.0 LD I0.1 NOT LPS A Q0.1 = Q0.1 LPP ALD O Q0.1 = Q0.1

7.1.10 NOP (No operation) instruction

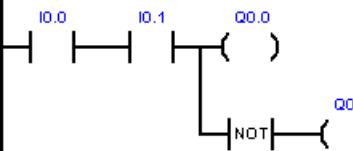
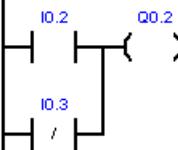
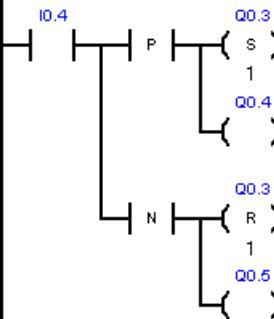
LAD	STL	Description
	NOP N	The No Operation (NOP) instruction has no effect on the user program execution. This instruction is not available in FBD mode. The operand N is a number from 0 to 255.

Inputs / Output	Data type	Operand
N (LAD, STL)	BYTE	N: Constant (0 to 255)

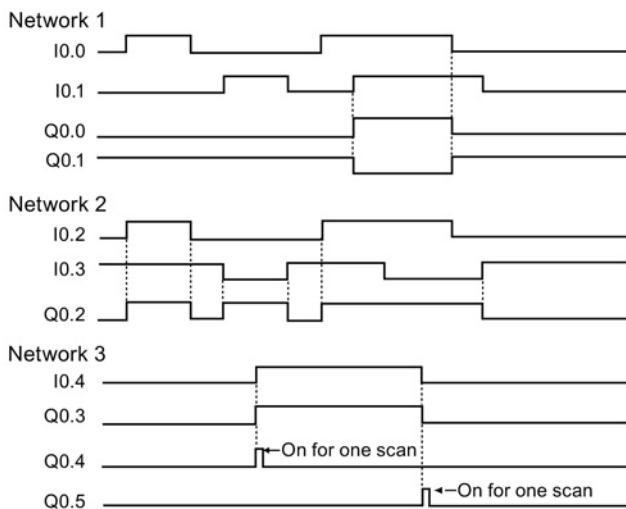
Program instructions

7.1 Bit logic

7.1.11 Bit logic input examples

LAD		STL
	Normally-open contacts I0.0 AND I0.1 must be ON (closed) to activate Q0.0. The NOT instruction acts as an inverter. In RUN mode, Q0.0 and Q0.1 have opposite logic states.	Network 1 LD I0.0 A I0.1 = Q0.0 NOT = Q0.1
	(Normally-open contact I0.2 must be ON) or (Normally-closed contact I0.3 must be OFF), to activate Q0.2. One or more parallel LAD branches (OR logic) must be true to make the output active.	Network 2 LD I0.2 ON I0.3 = Q0.2
	A positive Edge Up input on a P contact or a negative Edge Down input on an N contact outputs a pulse with a 1 scan cycle duration. In RUN mode, the pulsed state changes of Q0.4 and Q0.5 are too fast to be visible in program status view. The Set and Reset outputs latch the pulse state into Q0.3 and make the state change visible in program status view.	Network 3 LD I0.4 LPS EU S Q0.3, 1 = Q0.4 LPP ED R Q0.3, 1 = Q0.5

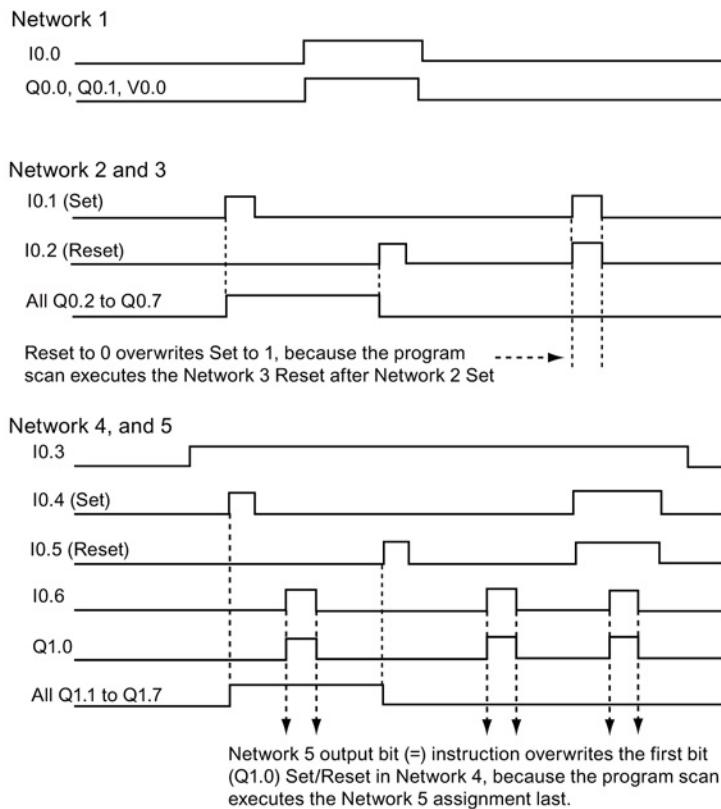
Run-mode timing for input example



7.1.12 Bit logic output examples

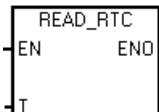
LAD	STL
	Network 1 LD I0.0 = Q0.0 = Q0.1 = V0.0
	Network 2 LD I0.1 S Q0.2, 6
	Network 3 LD I0.2 R Q0.2, 6
	Network 4 LD I0.3 LPS A I0.4 S Q1.0, 8 LPP A I0.5 R Q1.0, 8
	Network 5 LD I0.6 = Q1.0

Run-mode timing for output examples



7.2 Clock

7.2.1 Read and set real-time clock

LAD / FBD	STL	Description
	TODR T	The Read real-time clock instruction reads the current time and date from the CPU and loads it in an 8 byte Time buffer starting at byte address T.
	TODW T	The Set real-time clock instruction writes a new time and date to the CPU using the 8 byte Time buffer data that is assigned by T.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0007H T data error 	None

Input	Data type	Operand
T	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

Note

READ_RTC, SET_RTC programming tips

Invalid dates are not accepted. If you enter February 30, for example, a time-of-day non-fatal error will occur (0007H).

Do not use the READ_RTC / SET_RTC instructions in both the main program and in an interrupt routine. A READ_RTC / SET_RTC instruction in an interrupt routine cannot execute, while another READ_RTC / SET_RTC instruction is executing. In this case, system flag bit SM4.3 is set, indicating that two simultaneous accesses to the clock were attempted resulting in a T data error (non-fatal error 0007H).

The time-of-day clock in the CPU uses only the least significant two digits for the year, so for the year 2000, the year is represented as 00. However, user programs that use the year's value must take into account the two-digit representation.

Leap year is correctly handled through year 2099.

Format of 8 byte time buffer, beginning at byte address T

You must assign all date and time values in BCD format (for example, 16#12 for the year 2012). The BCD value range of 00 to 99 can assign years, in 2000 to 2099 range.

T byte	Description	Data value
0	Year	00 to 99 (BCD value) Year 20xx: where xx is the two digit BCD value in T-byte 0
1	Month	01 to 12 (BCD value)
2	Day	01 to 31 (BCD value)
3	Hour	00 to 23 (BCD value)
4	Minute	00 to 59 (BCD value)
5	Second	00 to 59 (BCD value)
6	reserved	Always set to 00
7	Day of week	Value ignored when written with the SET_RTC / TODW instruction. Value reports correct day of week when read with the READ_RTC / TODR instruction based upon current Year/Month/Day values. 1 to 7, 1 = Sunday, 7 = Saturday (BCD value)

Extended power outage effect on the CPU clock

See the S7-200 SMART system manual appendix A CPU specifications, for how long the real-time clock can maintain the correct time during power outages.

A CPU initializes with the time values shown in the following table, after an extended power outage.

Date	Time	Day of week
01-Jan-2000	00:00:00	Saturday

Note

Compact S7-200 SMART CPU models CR40 and CR60 do not have a RTC (Real Time Clock) or super cap

The READ_RTC and SET_RTC instructions can be used to set the year, date, and time values in CPU models CR40 and CR60, but the values will be lost on the next CPU power off-on cycle. On power-up, the date and time will initialize to January 1, 2000.

7.2.2 Read and set real-time clock extended

LAD / FBD	STL	Description
	TODRX T	The Read real-time clock extended instruction reads the current time, date, and daylight saving configuration from the PLC and loads it in a 19-byte buffer beginning at the address assigned by T.
	TODWX T	The Set real-time clock instruction writes a new time, date, and daylight saving configuration to the PLC using the 19-byte buffer data that is assigned by byte address T.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0007H T data error • 0091H Operand out of range 	None

Input	Data type	Operand
T	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

Note

READ_RTCX, SET_RTCX programming tips

Invalid dates are not accepted. If you enter February 30, for example, a time-of-day non-fatal error will occur (0007H).

Do not use the READ_RTCX / SET_RTCX instructions in both the main program and in an interrupt routine. A READ_RTCX / SET_RTCX instruction in an interrupt routine cannot execute, while another READ_RTCX / SET_RTCX instruction is executing. In this case, system flag bit SM4.3 is set, indicating that two simultaneous accesses to the clock were attempted resulting in a T data error (non-fatal error 0007H).

The time-of-day clock in the CPU uses only the least significant two digits for the year, so for the year 2000, the year is represented as 00. However, user programs that use the year's value must take into account the two-digit representation.

Leap year is correctly handled through year 2099.

Format of 19 byte time buffer, beginning at byte address T

Note

T bytes (9 to 18) or (9 to 20) are used only when a time correction mode is assigned in byte 8. Otherwise, the last values written to bytes (9 to 18) or (9 to 20) by STEP 7-Micro/WIN SMART or the SET_RTCX instruction are returned.

You must assign all date and time values in BCD format (for example, 16#12 for the year 2012). The BCD value range of 00 to 99 can assign the year in the range of 2000 to 2099.

T byte	Description	Data value
0	Year	00 to 99 (BCD value) Year 20xx: where xx is the two digit BCD value in T-byte 0
1	Month	01 to 12 (BCD value)
2	Day	01 to 31 (BCD value)
3	Hour	00 to 23 (BCD value)
4	Minute	00 to 59 (BCD value)
5	Second	00 to 59 (BCD value)
6	reserved	Always set to 00
7	Day of week	<p>Value ignored when written with the SET_RTCX / TODWX instruction.</p> <p>Value reports correct day of week when read with the READ_RTCX / TODRX instruction based upon current Year/Month/Day values.</p> <p>1 to 7, 1 = Sunday, 7 = Saturday (BCD value)</p>
8	Correction mode: For Daylight saving time (DST)	<p>00H = correction disabled</p> <p>01H = EU (time zone offset from UTC = 0 hrs)¹</p> <p>02H = EU (time zone offset from UTC = +1 hrs)¹</p> <p>03H = EU (time zone offset from UTC = +2 hrs)¹</p> <p>04H-07H = reserved</p> <p>08H = EU (time zone offset from UTC = -1 hrs)¹</p> <p>09H-0FH = reserved</p> <p>10H = US²</p> <p>11H = Australia³</p> <p>12H = reserved</p> <p>13H = New Zealand⁴</p> <p>14H-EDH = reserved</p> <p>EEH = user defined (day of week) (using values in bytes 9-20)</p> <p>EFH-FDH reserved</p> <p>FEH = reserved</p> <p>FFH = user defined (day of month) (using values in bytes 9-18)</p>
<i>The following bytes 9-18 are used only for correction mode = FFH (legacy user assigned)</i>		
9	DST correction hours	0 to 23 (BCD value)
10	DST correction minutes	0 to 59 (BCD value)
11	DST beginning month	1 to 12 (BCD value)
12	DST beginning day	1 to 31 (BCD value)
13	DST beginning hour	0 to 23 (BCD value)
14	DST beginning minute	0 to 59 (BCD value)

T byte	Description	Data value
15	DST ending month	1 to 12 (BCD value)
16	DST ending day	1 to 31 (BCD value)
17	DST ending hour	0 to 23 (BCD value)
18	DST ending minute	0 to 59 (BCD value)
<i>The following bytes 9-20 are used only for correction mode = EEH (extended user assigned)</i>		
9	DST correction hours	0 to 23 (BCD value)
10	DST correction minutes	0 to 59 (BCD value)
11	DST beginning month	1 to 12 (BCD value)
12	DST beginning week	1 to 5 (BCD value) ⁵
13	DST beginning weekday	1 to 7 (BCD value)
14	DST beginning hour	0 to 23 (BCD value)
15	DST beginning minute	0 to 59 (BCD value)
16	DST ending month	1 to 12 (BCD value)
17	DST ending week	1 to 5 (BCD value) ⁵
18	DST ending weekday	1 to 7 (BCD value)
19	DST ending hour	0 to 23 (BCD value)
20	DST ending minute	0 to 59 (BCD value)

- ¹ EU convention: Adjust time ahead one hour on last Sunday in March at 1:00 a.m. UTC. Adjust time back one hour on last Sunday in October at 2:00 a.m. UTC. (The local time when the correction is made depends upon the time zone offset from UTC).
- ² US convention: 2007 standard - Adjust time ahead one hour on second Sunday in March at 2:00 a.m. local time. Adjust time back one hour on first Sunday in November at 2:00 a.m. local time.
- ³ Australia convention: 2007 standard - Adjust time ahead one hour on first Sunday in October at 2:00 a.m. local time. Adjust time back one hour on first Sunday in April at 2:00 a.m. local time (also for Australia - Tasmania).
- ⁴ New Zealand convention: 2007 standard - Adjust time ahead one hour on last Sunday in September at 2:00 a.m. local time. Adjust time back one hour on first Sunday in April at 2:00 a.m. local time.
- ⁵ To assign the last occurrence of the weekday in the month (for example, the last Monday in April), set the week = 5.

Extended power outage effect on the CPU clock

See the S7-200 SMART system manual appendix A CPU specifications, for how long the real-time clock can maintain the correct time during power outages.

A CPU initializes with the time values shown in the following table, after an extended power outage.

Date	Time	Day of week
01-Jan-2000	00:00:00	Saturday

Note

Compact S7-200 SMART CPU models CR40 and CR60 do not have a RTC (Real Time Clock) or super cap

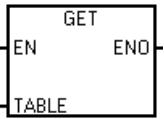
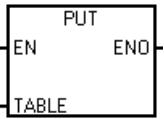
The READ_RTC and SET_RTC instructions can be used to set the year, date, and time values in CPU models CR40 and CR60, but the values will be lost on the next CPU power off-on cycle. On power-up, the date and time will initialize to January 1, 2000.

7.3 Communication

7.3.1 GET and PUT (Ethernet)

You can use the GET and PUT instructions for communication between S7-200 SMART CPUs through the Ethernet connection:

Table 7- 1 GET and PUT instructions

LAD/FBD	STL	Description
	<code>GET</code>	<p>The GET instruction initiates a communications operation on the Ethernet port to gather data from a remote device, as defined in the description table (TABLE).</p> <p>The GET instruction can read up to 222 bytes of information from a remote station.</p>
	<code>PUT</code>	<p>The PUT instruction initiates a communications operation on the Ethernet port to write data to a remote device, as defined in the description table (TABLE).</p> <p>The PUT instruction can write up to 212 bytes of information to a remote station.</p>

You can have any number of GET and PUT instructions in the program, but only a maximum of eight GET and PUT instructions can be activated at any one time. For example, you can have four GET and four PUT instructions, or two GET and six PUT instructions, active at the same time in a given CPU.

When you execute a GET or PUT instruction, the CPU makes an Ethernet connection to the remote IP address in the GET or PUT table. The CPU maintains a maximum of eight connections at a time. Once a connection is established, that connection is maintained until the CPU goes to STOP mode.

The CPU utilizes a single connection for all GET/PUT instructions that are directed to the same IP address. For example, if there are three GET instructions enabled at the same time when the remote IP address is 192.168.2.10, then the GET instructions execute sequentially on one Ethernet connection to IP address 192.168.2.10.

If you try to create a ninth connection (a ninth IP address), the CPU searches through the connections to find the connection that has been inactive for the longest period of time. The CPU disconnects this connection and then creates a connection to the new IP address.

Table 7- 2 Valid operands for the GET and PUT instructions

Inputs/Outputs	Data Type	Operands
TABLE	BYTE	IB, QB, VB, MB, SMB, SB, *VD, *LD, *AC

Error conditions that set ENO = 0:

- 0006 (indirect address)
- If the function returns an error and sets the E bit of table status byte (see the figure below)

The following figure describes the table that is referenced by the TABLE parameter, and the following table lists the error codes.

Table 7- 3 Definition of GET and PUT instructions TABLE parameter

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	D ¹	A ²	E ³	0	Error code			
1					Remote station			
2					IP			
3					Address ⁴			
4								
5					Reserved = 0 (Must be set to zero)			
6					Reserved = 0 (Must be set to zero)			
7								
8					Pointer to the data area in the remote station			
9								
10					(I, Q, M, V, or DB1) ⁵			

Byte offset	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
11	Data length ⁶							
12								
13								
14								
15	Pointer to the data area in the local station (this CPU) (I, Q, M, V, or DB1) ⁷							

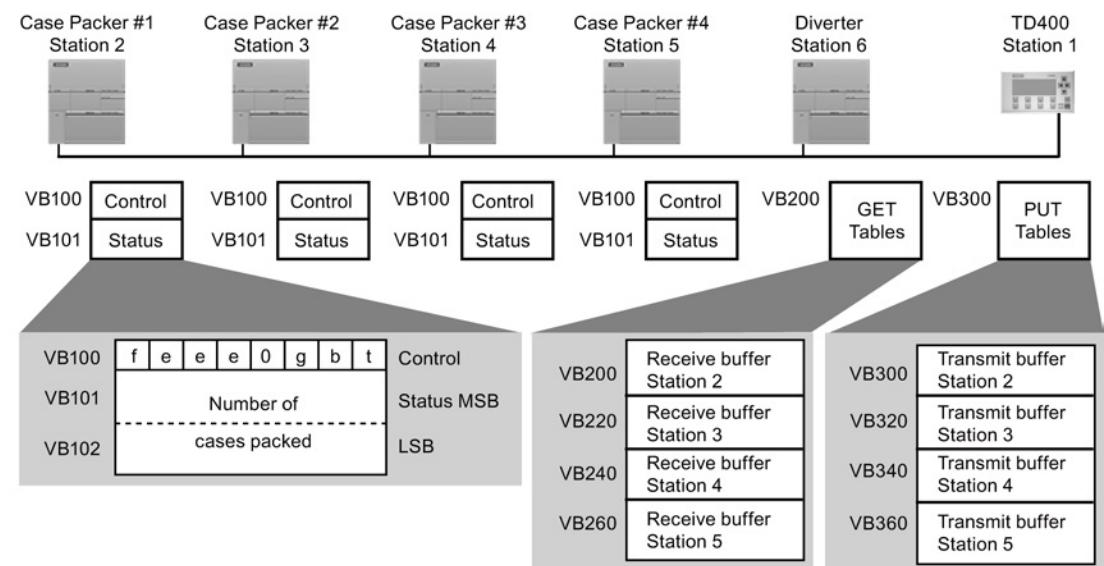
- ¹ D - Done (function has been completed)
- ² A - Active (function has been queued)
- ³ E - Error (function returned an error)
- ⁴ Remote station IP address: The address of the CPU whose data is to be accessed.
- ⁵ Pointer to the data area in the remote station: An indirect pointer to the data that is to be accessed in the remote station.
- ⁶ Data length: The number of bytes of data that are to be accessed in the remote station (1 to 212 bytes for PUT and 1 to 222 bytes for GET).
- ⁷ Pointer to the data area in the local station: An indirect pointer to the data that is to be accessed in the local station (this CPU).

Table 7- 4 Error codes for the GET and PUT instructions TABLE parameter

Code	Definition
0	No error
1	Illegal parameter in the PUT/GET table: <ul style="list-style-type: none"> • Local area is not I, Q, M, or V • Local area is not large enough for the data length requested • Data length is zero or greater than 222 bytes for a GET or greater than 212 bytes for a PUT • Remote area is not I, Q, M, or V • Remote IP address is illegal (0.0.0.0) • Remote IP address is a broadcast address or a multicast address • Remote IP address is the same as the Local IP address • Remote IP address is on a different subnet
2	Too many PUT/GET instructions are currently active (only 16 allowed)
3	No connection available. All connections are currently active with outstanding requests
4	Error returned from remote CPU: <ul style="list-style-type: none"> • Too much data was requested or sent • Writing to Q memory is not allowed in STOP mode • Memory area is write-protected (see SDB configuration)

Code	Definition
5	No connection available to the remote CPU: <ul style="list-style-type: none"> • Remote CPU does not have an available server connection • Connection to remote CPU was lost (CPU powered off, physical disconnect)
6 to 9, A to F	Not used (Reserved for future use)

The following figure shows an example to illustrate the utility of the GET and PUT instructions. For this example, consider a production line where tubs of butter are being filled and sent to one of four boxing machines (case packers). The case packer packs eight tubs of butter into a single cardboard box. A diverter machine controls the flow of butter tubs to each of the case packers. Four CPUs control the case packers, and a CPU with a TD 400 operator interface controls the diverter.



- t Out of butter tubs to pack; t=1, out of butter tubs
- b Box supply is low; b=1, must add boxes in the next 30 minutes
- g Glue supply is low; g=1, must add glue in the next 30 minutes
- eee Error code identifying the type of fault experienced
- f Fault indicator; f=1, the case packer has detected an error

The following figure shows the GET table (VB200) and PUT table (VB300) for accessing the data in station 2. The Diverter CPU uses a GET instruction to read the control and status information on a continuous basis from each of the case packers. Each time a case packer has packed 100 cases, the diverter notes this and sends a message to clear the status word using a PUT instruction.

Table 7- 5 GET and PUT instructions buffer for reading from and clearing the count of Case Packer
1

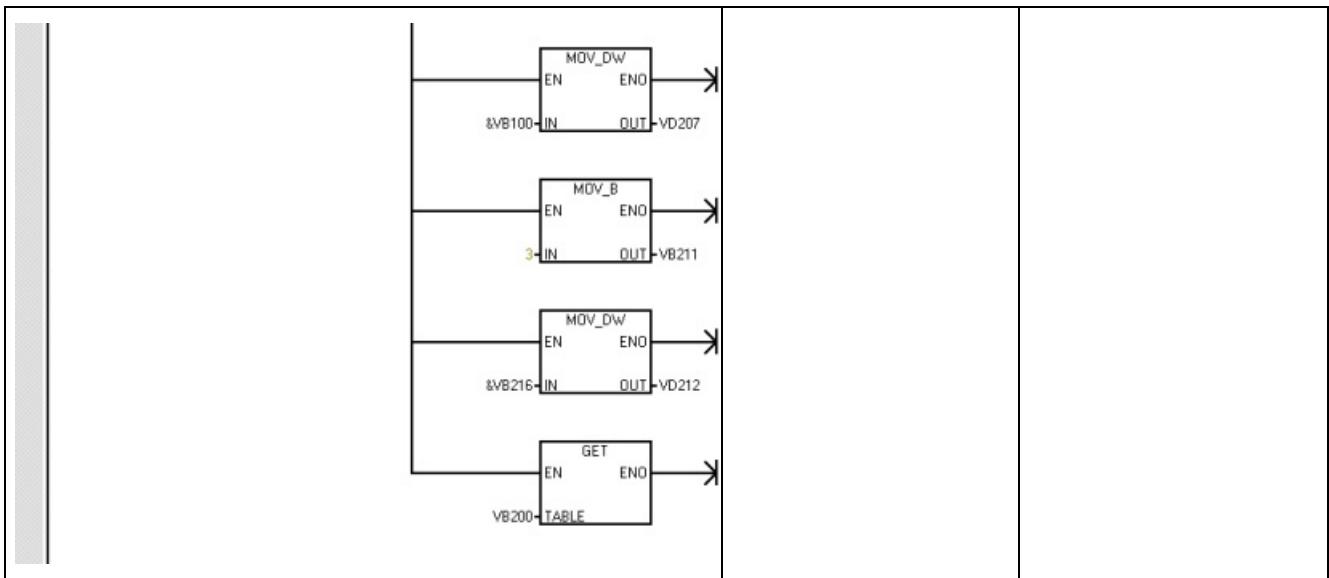
GET_TABLE buffer	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	PUT_TABLE buffer	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
VB200	D	A	E	0	Error code				VB300	D	A	E	0	Error code			
VB201	Remote station IP address = 192.								VB301	Remote station IP address = 192.							
VB202	168.								VB302	168.							
VB203	50.								VB303	50.							
VB204	2								VB304	2							
VB205	Reserved = 0 (Must be set to zero)								VB305	Reserved = 0 (Must be set to zero)							
VB206	Reserved = 0 (Must be set to zero)								VB306	Reserved = 0 (Must be set to zero)							
VB207	Pointer to the data								VB307	Pointer to the data							
VB208	area in the								VB308	area in the							
VB209	remote station =								VB309	remote station =							
VB210	(&VB100)								VB310	(&VB101)							
VB211	Data length = 3 bytes								VB311	Data length = 2 bytes							
VB212	Pointer to the data								VB312	Pointer to the data							
VB213	area in the								VB313	area in the							
VB214	local station (this CPU) =								VB314	local station (this CPU) =							
VB215	(&VB216)								VB315	(&VB316)							
VB216	Control								VB316	0							
VB217	Status MSB								VB317	0							
VB218	Status LSB																

In this example, the data immediately follows the PUT and GET tables. This data can be placed anywhere in the CPU memory since it is pointed to by the local station pointer in a table (for example, VB212 - VB215).

Table 7- 6 Example: GET and PUT instructions

	Network 1 LD SM0.1 FILL +0, VW200, 40 FILL +0, VW300, 40	On the first scan, clear all receive and transmit buffers.
	Network 2 LD V200.7 AW= VW217, +100 MOVB 192, VB301 MOVB 168, VB302 MOVB 50, VB303 MOVB 2, VB304 MOVW 0, VB305 MOVD &VB101, VD307 MOVB 2, VB311 MOVD &VB316, VD312 MOVW 0, VW316 PUT VB300	When the GET "Done" bit (V200.7) is set and 100 cases have been packed: <ol style="list-style-type: none"> 1. Load the station address of case packer 1. 2. Load a pointer to the data in the remote station. 3. Load the length of data to be transmitted. 4. Load the data to transmit. Reset the number of cases packed by case packer 1

3		Network 3 LD V200.7 MOV B VB216, VB400	When the GET "Done" bit is set, save the control data from case packer 1.
4		Network 4 LDN SM0.1 AN V200.6 AN V200.5 MOV B 192, VB201 MOV B 168, VB202 MOV B 50, VB203 MOV B 2, VB204 MOV W 0, VB205 MOVD &VB100, VD207 MOV B 3, VB211 MOVD &VB216, VD212 GET VB200	If not the first scan and there are no errors: 1. Load the station address of case packer 1. 2. Load a pointer to the data in the remote station. 3. Load the length of data to be received. 4. Read the control and status data in case packer 1.



7.3.2 Transmit and receive (Freeport on RS485/RS232)

You can use the Transmit (XMT) and Receive (RCV) instructions for communication between a S7-200 SMART CPU and other devices through the CPU serial port(s). Each S7-200 SMART CPU provides an integrated RS485 port (Port 0). The standard CPUs additionally support an optional CM01 Signal Board (SB) RS232/RS485 port (Port 1). The communication protocol must be implemented in the user program.

LAD / FBD	STL	Description
	XMT TBL, PORT	The Transmit instruction (XMT) is used in Freeport mode to transmit data by means of the communications port(s).
	RCV TBL, PORT	The Receive instruction (RCV) initiates or terminates the receive message function. You must specify a start and an end condition for the Receive box to operate. Messages received through the specified port (PORT) are stored in the data buffer (TBL). The first entry in the data buffer specifies the number of bytes received.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0009H Simultaneous Transmit/Receive on port 0 • 000BH Simultaneous Transmit/Receive on port 1 • 0090H Port number is invalid • Receive parameter error sets SM86.6 or SM186.6 • CPU is not in Freeport mode 	<ul style="list-style-type: none"> • SM 86.6 Receive message terminated on port 0 • SM 186.6 Receive message terminated on port 1

Input / output	Data type	Operand
TBL	BYTE	IB, QB, VB, MB, SMB, SB, *VD, *LD, *AC
PORt	BYTE	<p>Constant: 0 or 1</p> <p>Note: The two available ports are as follows:</p> <ul style="list-style-type: none"> • Integrated RS485 port (Port 0), • CM01 Signal Board (SB) RS232/RS485 port (Port 1)

Using Freeport mode to control the serial communications port

You can select the Freeport mode to control the serial communications port of the CPU by means of the user program. When you select Freeport mode, your program controls the operation of the communications port through the use of the receive interrupts, the transmit interrupts, the Transmit instruction, and the Receive instruction. The communications protocol is entirely controlled by the user program while in Freeport mode. SMB30 and SMB130 are used to select the baud rate and parity.

Two special memory bytes are assigned to the two physical ports:

- SMB30 to the integrated RS485 port (Port 0)
- SMB130 to the CM01 RS232/RS485 Signal Board (SB) port (Port 1)

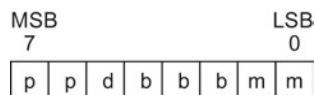
The Freeport mode is disabled and normal communications are re-established (for example, HMI device access) when the CPU is in STOP mode.

In the simplest case, you can send a message to a printer or a display using only the Transmit (XMT) instruction. Other examples include a connection to a bar code reader, a weigh scale, and a welder. In each case, you must write your program to support the protocol that is used by the device with which the CPU communicates while in Freeport mode.

Freeport communications are possible only when the CPU is in RUN mode. Enable the Freeport mode by setting a value of 01 in the protocol select field of SMB30 (Port 0) or SMB130 (Port 1). While in Freeport mode, communication with an HMI on the same port is not possible.

Changing PPI communications to Freeport mode

SMB30 and SMB130 configure the communications ports, 0 and 1 respectively, for Freeport operation and provide selection of baud rate, parity, and number of data bits. The following figure describes the Freeport control byte. One stop bit is generated for all configurations.



pp	Parity select		d	Data bits per character	
	00 =	No parity		0 =	8 bits per character
	01 =	Even parity		1 =	7 bits per character
bbb	Freeport baud rate		mm	Protocol selection	
	000 =	38400		00 =	PPI slave mode
	001 =	19200		01 =	Freeport mode
	010 =	9600		10 =	Reserved (defaults to PPI slave mode)
	011 =	4800		11 =	Reserved (defaults to PPI slave mode)
	100 =	2400			
	101 =	1200			
	110 =	115200			
	111 =	57600			

Transmit data

The Transmit instruction lets you send a buffer of one or more characters, up to a maximum of 255. The following figure shows the format of the Transmit buffer.



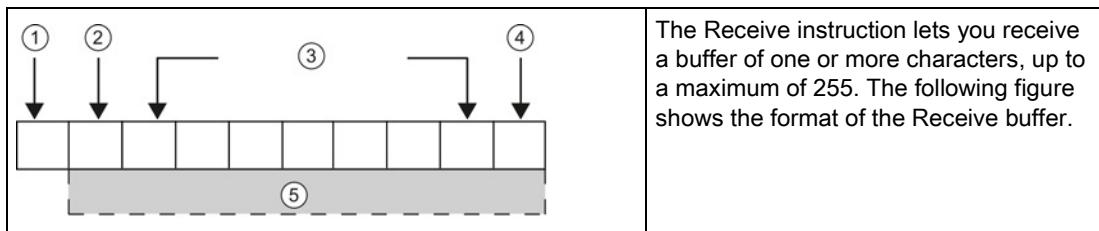
- ① Number of bytes to transmit
- ② Characters of the message

If an interrupt routine is attached to the transmit complete event, the CPU generates an interrupt (interrupt event 9 for port 0 and interrupt event 26 for port 1) after the last character of the buffer is sent.

You can transmit without using interrupts (for example, sending a message to a printer) by monitoring SM4.5 (port 0) or SM4.6 (port 1) to signal when transmission is complete.

You can use the Transmit instruction to generate a BREAK condition by setting the number of characters to zero and then executing the Transmit instruction. This generates a BREAK condition on the line for 16-bit times at the current baud rate. Transmitting a BREAK is handled in the same manner as transmitting any other message, in that a Transmit interrupt is generated when the BREAK is complete and SM4.5 or SM4.6 signals the current status of the Transmit operation.

Receive data



The Receive instruction lets you receive a buffer of one or more characters, up to a maximum of 255. The following figure shows the format of the Receive buffer.

- ① Number of bytes received (byte field)
- ② Start character
- ③ Message
- ④ End character
- ⑤ Characters of the message

If an interrupt routine is attached to the receive message complete event, the CPU generates an interrupt (interrupt event 23 for port 0 and interrupt event 24 for port 1) after the last character of the buffer is received.

You can receive messages without using interrupts by monitoring SMB86 (port 0) or SMB186 (port 1). This byte is non-zero when the Receive instruction is inactive or has been terminated. It is zero when a receive is in progress.

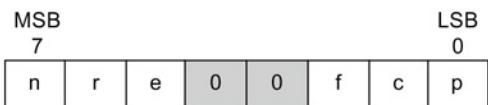
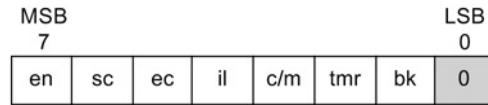
As shown in the following table, the Receive instruction allows you to select the message start and message end conditions, using SMB86 through SMB94 for port 0 and SMB186 through SMB194 for port 1.

Note

The receive message function is automatically terminated in case of a framing, parity, overrun, or break error.

You must define a start condition and an end condition (maximum character count) for the receive message function to operate.

Receive buffer format (SMB86 to SMB94, and SMB186 to SMB194)

Port 0	Port 1	Description
SMB86	SMB186	<p>Receive message status byte</p>  <p>n: 1 = Receive message function terminated; user issued disable command. r: 1 = Receive message function terminated; error in input parameters or missing start or end condition. e: 1 = End character received. t: 1 = Receive message function terminated; timer expired. c: 1 = Receive message function terminated; maximum character count achieved. p: 1 = Receive message function terminated; a parity error.</p>
SMB87	SMB187	<p>Receive message control byte</p>  <p>en: 0 = Receive message function is disabled. 1 = Receive message function is enabled. The enable/disable receive message bit is checked each time the RCV instruction is executed.</p> <p>sc: 0 = Ignore SMB88 or SMB188. 1 = Use the value of SMB88 or SMB188 to detect start of message.</p> <p>ec: 0 = Ignore SMB89 or SMB189. 1 = Use the value of SMB89 or SMB189 to detect end of message.</p> <p>il: 0 = Ignore SMB90 or SMB190. 1 = Use the value of SMB90 or SMB190 to detect start of message.</p> <p>c/m: 0 = Timer is an inter-character timer. 1 = Timer is a message timer.</p> <p>tmr: 0 = Ignore SMW92 or SMW192. 1 = Terminate receive if the time period in SMW92 or SMW192 is exceeded.</p> <p>bk: 0 = Ignore break conditions. 1 = Use break condition as start of message detection.</p>
SMB88	SMB188	Start of message character.
SMB89	SMB189	End of message character.
SMW90	SMW190	Idle line time period given in milliseconds. The first character received after idle line time has expired is the start of a new message.

Port 0	Port 1	Description
SMW92	SMW192	Inter-character/message timer time-out value given in milliseconds. If the time period is exceeded, the receive message function is terminated.
SMB94	SMB194	Maximum number of characters to be received (1 to 255 bytes). This range must be set to the expected maximum buffer size, even if the character count message termination is not used.

Start and End conditions for the Receive instruction

The Receive instruction uses the bits of the receive message control byte (SMB87 or SMB187) to define the message start and end conditions.

Note

If there is traffic present on the communications port from other devices when the Receive instruction is executed, the receive message function could begin receiving a character in the middle of that character, resulting in a possible parity or framing error and termination of the receive message function. If parity is not enabled the received message could contain incorrect characters. This situation can occur when the start condition is specified to be a specific start character or any character, as described in item 2 and item 6 below.

The Receive instruction supports several message start conditions. Specifying a start condition involving a break or an idle line detection avoids the problem of starting a message in the middle of a character by forcing the receive message function to synchronize the start of the message with the start of a character before placing characters into the message buffer.

The Receive instruction supports several start conditions:

1. *Idle line detection*: The idle line condition is defined as a quiet or idle time on the transmission line. A receive is started when the communications line has been quiet or idle for the number of milliseconds specified in SMW90 or SMW190. When the Receive instruction in your program is executed, the receive message function initiates a search for an idle line condition. If any characters are received before the idle line time expires, the receive message function ignores those characters and restarts the idle line timer with the time from SMW90 or SMW190. See the following figure. After the idle line time expires, the receive message function stores all subsequent characters received in the message buffer.

The idle line time should always be greater than the time to transmit one character (start bit, data bits, parity and stop bits) at the specified baud rate. A typical value for the idle line time is three character times at the specified baud rate.

You use idle line detection as a start condition for binary protocols, protocols where there is not a particular start character, or when the protocol specifies a minimum time between messages.

Setup: il = 1, sc = 0, bk = 0, SMW90/SMW190 = idle line timeout in milliseconds



- ① Receive instruction is executed: Starts the idle time
② Restarts the idle time
③ Idle time is detected: Starts the Receive Message function
④ First character is placed in the message buffer
2. *Start character detection*: The start character is any character which is used as the first character of a message. A message is started when the start character specified in SMB88 or SMB188 is received. The receive message function stores the start character in the receive buffer as the first character of the message. The receive message function ignores any characters that are received before the start character. The start character and all characters received after the start character are stored in the message buffer.

Typically, you use start character detection for ASCII protocols in which all messages start with the same character.

Setup: il = 0, sc = 1, bk = 0, SMW90/SMW190 = don't care, SMB88/SMB188 = start character

3. *Idle line and start character:* The Receive instruction can start a message with the combination of an idle line and a start character. When the Receive instruction is executed, the receive message function searches for an idle line condition. After finding the idle line condition, the receive message function looks for the specified start character. If any character but the start character is received, the receive message function restarts the search for an idle line condition. All characters received before the idle line condition has been satisfied and before the start character has been received are ignored. The start character is placed in the message buffer along with all subsequent characters.

The idle line time should always be greater than the time to transmit one character (start bit, data bits, parity and stop bits) at the specified baud rate. A typical value for the idle line time is three character times at the specified baud rate.

Typically, you use this type of start condition when there is a protocol that specifies a minimum time between messages, and the first character of the message is an address or something which specifies a particular device. This is most useful when implementing a protocol where there are multiple devices on the communications link. In this case the Receive instruction triggers an interrupt only when a message is received for the specific address or devices specified by the start character.

Setup: il = 1, sc = 1, bk = 0, SMW90/SMW190 > 0, SMB88/SMB188 = start character

4. *Break detection:* A break is indicated when the received data is held to a zero value for a time greater than a full character transmission time. A full character transmission time is defined as the total time of the start, data, parity and stop bits. If the Receive instruction is configured to start a message on receiving a break condition, any characters received after the break condition are placed in the message buffer. Any characters received before the break condition are ignored.

Typically, you use break detection as a start condition only when a protocol requires it.

Setup: il = 0, sc = 0, bk = 1, SMW90/SMW190 = don't care, SMB88/SMB188 = don't care

5. *Break and a start character:* The Receive instruction can be configured to start receiving characters after receiving a break condition, and then a specific start character, in that sequence. After the break condition, the receive message function looks for the specified start character. If any character but the start character is received, the receive message function restarts the search for a break condition. All characters received before the break condition has been satisfied and before the start character has been received are ignored. The start character is placed in the message buffer along with all subsequent characters.

Setup: il = 0, sc = 1, bk = 1, SMW90/SMW190 = don't care, SMB88/SMB188 = start character

6. *Any character:* The Receive instruction can be configured to immediately start receiving any and all characters and placing them in the message buffer. This is a special case of the idle line detection. In this case the idle line time (SMW90 or SMW190) is set to zero. This forces the Receive instruction to begin receiving characters immediately upon execution.

Setup: il = 1, sc = 0, bk = 0, SMW90/SMW190 = 0, SMB88/SMB188 = don't care

Starting a message on any character allows the message timer to be used to time out the receiving of a message. This is useful in cases where Freeport is used to implement the master or host portion of a protocol and there is a need to time out if no response is received from a slave device within a specified amount of time. The message timer starts when the Receive instruction executes because the idle line time was set to zero. The message timer times out and terminates the receive message function if no other end condition is satisfied.

Setup: il = 1, sc = 0, bk = 0, SMW90/SMW190 = 0, SMB88/SMB188 = don't care, c/m = 1, tmr = 1, SMW92 = message timeout in milliseconds

The Receive instruction supports several ways to terminate a message. The message can be terminated on one or a combination of the following:

1. *End character detection*: The end character is any character which is used to denote the end of the message. After finding the start condition, the Receive instruction checks each character received to see if it matches the end character. When the end character is received, it is placed in the message buffer and the receive is terminated.

Typically, you use end character detection with ASCII protocols where every message ends with a specific character. You can use end character detection in combination with the intercharacter timer, the message timer or the maximum character count to terminate a message.

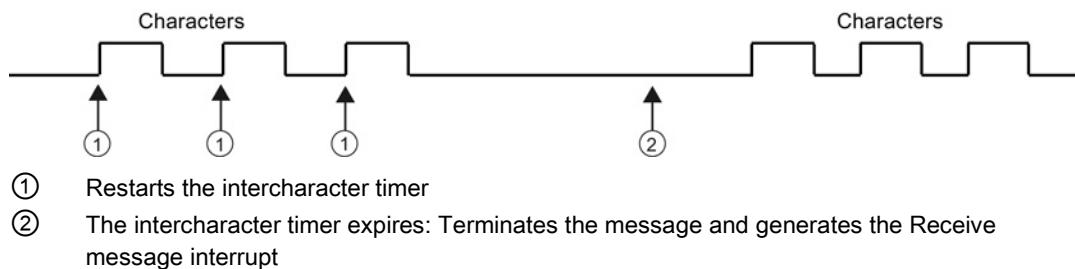
Setup: ec = 1, SMB89/SMB189 = end character

2. *Intercharacter timer*: The intercharacter time is the time measured from the end of one character (the stop bit) to the end of the next character (the stop bit). If the time between characters (including the second character) exceeds the number of milliseconds specified in SMW92 or SMW192, the receive message function is terminated. The intercharacter timer is restarted on each character received. See the following figure.

You can use the intercharacter timer to terminate a message for protocols which do not have a specific end-of-message character. This timer must be set to a value greater than one character time at the selected baud rate since this timer always includes the time to receive one entire character (start bit, data bits, parity and stop bits).

You can use the intercharacter timer in combination with the end character detection and the maximum character count to terminate a message.

Setup: c/m = 0, tmr = 1, SMW92/SMW192 = timeout in milliseconds

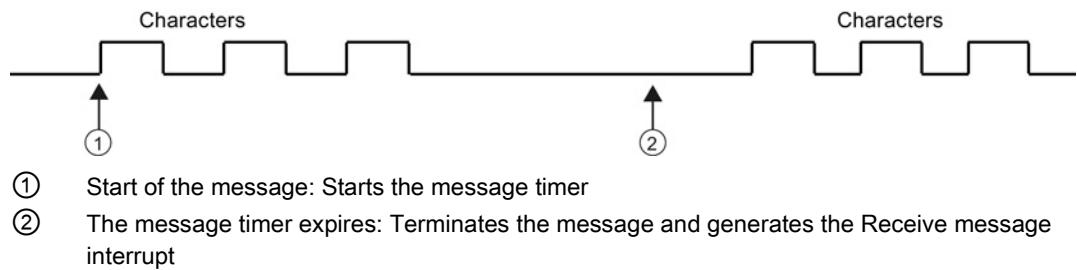


3. *Message timer:* The message timer terminates a message at a specified time after the start of the message. The message timer starts as soon as the start condition(s) for the receive message function have been met. The message timer expires when the number of milliseconds specified in SMW92 or SMW192 have passed. See the following figure.

Typically, you use a message timer when the communications devices cannot guarantee that there will not be time gaps between characters or when operating over modems. For modems, you can use a message timer to specify a maximum time allowed to receive the message after the message has started. A typical value for a message timer would be about 1.5 times the time required to receive the longest possible message at the selected baud rate.

You can use the message timer in combination with the end character detection and the maximum character count to terminate a message.

Setup: c/m = 1, tmr = 1, SMW92/SMW192 = timeout in milliseconds



4. *Maximum character count:* The Receive instruction must be told the maximum number of characters to receive (SMB94 or SMB194). When this value is met or exceeded, the receive message function is terminated. The Receive instruction requires that the user specify a maximum character count even if this is not specifically used as a terminating condition. This is because the Receive instruction needs to know the maximum size of the receive message so that user data placed after the message buffer is not overwritten.

The maximum character count can be used to terminate messages for protocols where the message length is known and always the same. The maximum character count is always used in combination with the end character detection, intercharacter timer, or message timer.

5. *Parity errors:* The Receive instruction automatically terminates when the hardware signals a parity, framing, or overrun error; or if a break condition is detected after the start of a message. Parity errors occur only if parity is enabled in SMB30 or SMB130. Framing errors occur if the stop bit is not correct. Overrun errors occur if characters come in too quickly for the hardware to handle. A break condition terminates a message because it resembles a parity or framing error to the hardware. There is no way to disable this function.
6. *User termination:* The user program can terminate a receive message function by executing another Receive instruction with the enable bit (EN) in SMB87 or SMB187 set to zero. This immediately terminates the receive message function.

Using character interrupt control to receive data

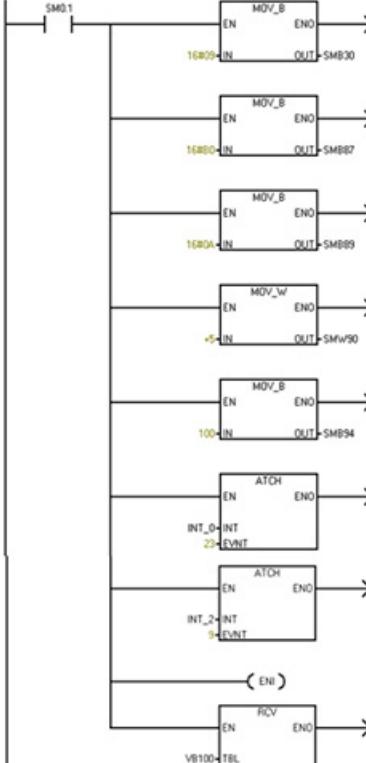
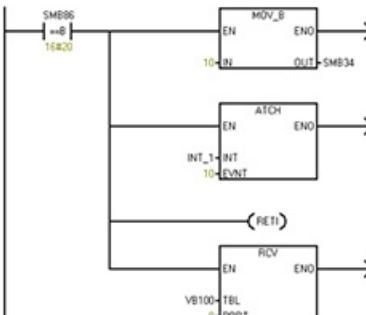
To allow complete flexibility in protocol support, you can also receive data using character interrupt control. Each character received generates an interrupt. The received character is placed in SMB2, and the parity status (if enabled) is placed in SM3.0 just prior to execution of the interrupt routine attached to the receive character event. SMB2 is the Freeport receive character buffer. Each character received while in Freeport mode is placed in this location for easy access from the user program. SMB3 is used for Freeport mode and contains a parity error bit that is turned on when a parity, framing, overrun, or break error is detected on a received character. All other bits of the byte are reserved. Use the parity bit either to discard the message or to generate a negative acknowledgement to the message.

When the character interrupt is used at high baud rates (38.4 kbaud to 115.2 kbaud), the time between interrupts is very short. For example, the character interrupt for 38.4 kbaud is 260 microseconds, for 57.6 kbaud is 173 microseconds, and for 115.2 kbaud is 86 microseconds. Ensure that you keep the interrupt routines very short to avoid missing characters, or else use the Receive instruction.

Note

SMB2 and SMB3 are shared between Port 0 and Port 1. When the reception of a character on Port 0 results in the execution of the interrupt routine attached to that event (interrupt event 8), SMB2 contains the character received on Port 0, and SMB3 contains the parity status of that character. When the reception of a character on Port 1 results in the execution of the interrupt routine attached to that event (interrupt event 25), SMB2 contains the character received on Port 1 and SMB3 contains the parity status of that character.

Example: Transmit and Receive instructions

MAIN	Network 1	Network 1 //This program receives a string of characters until a line feed character is received. The message is then transmitted back to the sender.	
		LD SM0.1 MOVB 16#09, SMB30	On the first scan: 1. Initialize Freeport: - Select 9600 baud. - Select 8 data bits. - Select no parity.
		MOVB 16#B0, SMB87	2. Initialize RCV message control byte: - RCV enabled. - Detect end of message character. - Detect idle line condition as the message start condition.
		MOVB 16#0A, SMB89	3. Set end of message character to hex 0A (line feed).
		MOVW +5, SMW90	4. Set idle line timeout to 5 ms.
		MOVB 100, SMB94	5. Set maximum number of characters to 100.
		ATCH INT_0, 23	6. Attach interrupt 0 to the Receive Complete event.
		ATCH INT_2, 9	7. Attach interrupt 2 to the Transmit Complete event.
		ENI	8. Enable user interrupts.
INT 0	Network 1	RCV VB100, 0	9. Enable receive box with buffer at VB100.
		LDB= SMB86, 16#20 MOVB 10, SMB34 ATCH INT_1, 10 CRETI NOT RCV VB100, 0	Receive complete interrupt routine: 1. If receive status shows receive of end character, then attach a 10 ms timer to trigger a transmit and return. 2. If the receive completed for any other reason, then start a new receive.
INT 1	Network 1	LD SM0.0 DTCH 10 XMT VB100, 0	10-ms Timer interrupt: 1. Detach timer interrupt. 2. Transmit message back to user on port.

Program instructions

7.3 Communication

INT 2	Network 1 	Network 1 LD SM0.0 RCV VB100, 0	Transmit Complete interrupt: Enable another receive.
-------	----------------------	--	--

7.3.3 Get port address and set port address (PPI protocol on RS485/RS232)

LAD / FBD	STL	Description
	GPA ADDR, PORT	The GET_ADDR instruction reads the station address of the CPU port specified in PORT and places the value in the address specified in ADDR.
	SPA ADDR, PORT	The SET_ADDR instruction sets the port station address (PORT) to the value specified in ADDR. The new address is not saved permanently. After a power cycle, the affected port returns to the last address (the one that was downloaded with the system block).

Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 006H Indirect address • 0004H Attempted to perform a SET_ADDR instruction in an interrupt routine • 0090H Port number is invalid • 0091H Port address is invalid 	None

Input / output	Data type	Operand
ADDR	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant (A constant value is valid only for the Set Port Address instruction.)
PORT	BYTE	Constant: 0 or 1 Note: The two available ports are as follows: <ul style="list-style-type: none"> • Integrated RS485 port (Port 0), • CM01 Signal Board (SB) RS232/RS485 port (Port 1)

7.3.4 Get IP address and set IP address (Ethernet)

LAD / FBD	STL	Description
	GIP ADDR, MASK, GATE	The GIP_ADDR instruction copies the CPU's IP address into ADDR, the CPU's subnet mask into MASK, and the CPU's gateway into GATE.
	SIP ADDR, MASK, GATE	The SIP_ADDR instruction sets the CPU's IP address to the value found in ADDR, the CPU's subnet mask to the value found in MASK, and the CPU's gateway to the value found in GATE.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 006H Indirect address • 0004H Attempted to execute a SIP_ADDR instruction in an interrupt routine • IP address cannot be changed (see following note) • IP address is invalid for the current subnet 	None

Input / output	Data type	Operand
ADDR	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
MASK	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
GATE	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Note

- In order to use the SIP_ADDR instruction, the "IP address data is fixed to the values below and cannot be changed by other means" checkbox must be unchecked. This configuration checkbox is located in the "System Block", "Communication" node in the "Ethernet" field.
- The IP address, subnet mask, and gateway values are written to persistent memory.

7.4 Compare

7.4.1 Compare number values

The compare instructions can compare two number values with the same data type. You can compare bytes, integers, double integers, and real numbers.

For **LAD** and **FBD**: When the comparison is TRUE, the compare instruction sets ON a contact (LAD network power flow), or output (FBD logic flow).

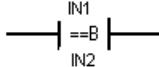
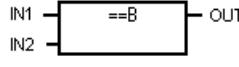
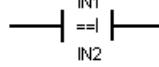
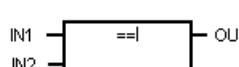
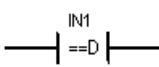
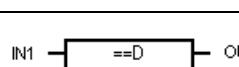
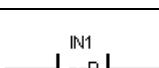
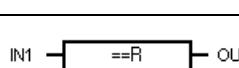
For **STL**: When the comparison is TRUE, the compare instructions can load, AND, or OR a 1 with the value on the top of the logic stack.

Six comparison types are available

Comparison type	The output is TRUE only if
<code>== (LAD/FBD)</code> <code>= (STL)</code>	IN1 is equal to IN2
<code><></code>	IN1 is not equal to IN2
<code>>=</code>	IN1 is greater than or equal to IN2
<code><=</code>	IN1 is less than or equal to IN2
<code>></code>	IN1 is greater than IN2
<code><</code>	IN1 is less than IN2

Data type selection determines the required data type for the IN1 and IN2 comparison

Data type identifier	Required IN1, IN2 data type
B	Unsigned byte
W	Signed word integer
D	Signed double word integer
R	Signed real

LAD contacts, FBD boxes	STL	Comparison result
	LDB= IN1, IN2 OB= IN1, IN2 AB= IN1, IN2	Compare two unsigned byte values: The result is TRUE, if IN1 = IN2
		
	LDW= IN1, IN2 OW= IN1, IN2 AW= IN1, IN2	Compare two signed integer values: The result is TRUE, if IN1 = IN2
		
	LDD= IN1, IN2 OD= IN1, IN2 AD= IN1, IN2	Compare two signed double integer values: The result is TRUE, if IN1 = IN2
		
	LDR= IN1, IN2 OR= IN1, IN2 AR= IN1, IN2	Compare two signed real values: The result is TRUE, if IN1 = IN2
		

Note

The following conditions cause a non-fatal error, set power flow to OFF (ENO bit = 0), and use value 0 as the result of the comparison

- Illegal indirect address is encountered (any compare instruction)
- Illegal real number (for example, NaN) is encountered for compare real instruction

To prevent these conditions from occurring, ensure that you properly initialize pointers and values that contain real numbers before executing compare instructions that use these values.

Compare instructions are executed regardless of the state of power flow.

Program instructions

7.4 Compare

Input / output	Data type	Operand
IN1, IN2	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	BOOL	LAD: Power flow FBD: I, Q, V, M, SM, S, T, C, L, Logic Flow

Example compare values

LAD	STL
<p>I0.1</p> <p>Activate I0.1 to load V memory addresses with low values that make the comparisons FALSE and that set the status indicators OFF.</p>	Network 1 LD I0.1 MOVW -30000, VW0 MOVD -200000000, VD2 MOVR 1.012E-006, VD6
<p>I0.2</p> <p>Activate I0.2 to load V memory addresses with high values that make the comparisons TRUE and that set the status indicators ON.</p>	Network 2 LD I0.2 MOVW +30000, VW0 MOVD -100000000, VD2 MOVR 3.141593, VD6
<p>I0.3</p> <p>Activate I0.3 to perform comparisons. The Integer Word comparison tests to find if VW0 > +10000 is TRUE. You can also compare two values stored in variable memory like VW0 > VW100.</p>	Network 3 LD I0.3 LPS AW> VW0, +10000 = Q0.2 LRD AD< -150000000, VD2 = Q0.3 LPP AR> VD6, 5.001E-006 = Q0.4

See also

Constants (Page 63)

7.4.2 Compare character strings

The compare string instructions can compare two ASCII character strings.

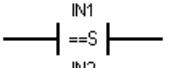
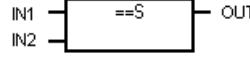
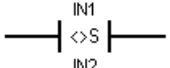
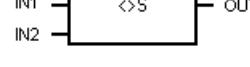
For **LAD** and **FBD**: When the comparison is TRUE, the compare instruction turns ON the contact (LAD), or output (FBD).

For **STL**: When the comparison is TRUE, the compare instruction loads, ANDs, or ORs a 1 with the value on the top of the logic stack.

Comparisons can be made between two variables, or between a constant and a variable. If a constant is used in a comparison, then it must be the top parameter (LAD contact / FBD box) or the first parameter (STL).

In the program editor, a constant string parameter assignment must begin and end with a double quote character. The maximum length of a constant string entry is 126 characters (bytes).

In contrast, a variable string is referenced by the byte address of the initial length byte with the character bytes stored the next byte addresses. A variable string has a maximum length of 254 characters (bytes) and can be initialized in the data block editor (with beginning and ending double quote character).

LAD contact FBD box	STL	Description
	LDS= IN1, IN2 OS= IN1, IN2 AS= IN1, IN2	Compare two character strings of STRING data type: The result is TRUE, if string IN1 equals string IN2.
		
	LDS<> IN1, IN2 OS<> IN1, IN2 AS<> IN1, IN2	Compare two character strings of STRING data type: The result is TRUE, if string IN1 does not equal string IN2.
		

Note

The following conditions cause a non-fatal error, set power flow to OFF (ENO bit = 0), and use value 0 as the result of the comparison:

- Illegal indirect address is encountered (any compare instruction)
- A variable string with a length greater than 254 characters is encountered (Compare String instruction)
- A variable string whose starting address and length are such that it will not fit in the specified memory area (Compare String instruction)

To prevent these conditions from occurring, ensure that you properly initialize pointers and memory locations that are intended to hold ASCII strings prior to executing compare instructions that use these values. Ensure that the buffer reserved for an ASCII string can reside completely within the specified memory area.

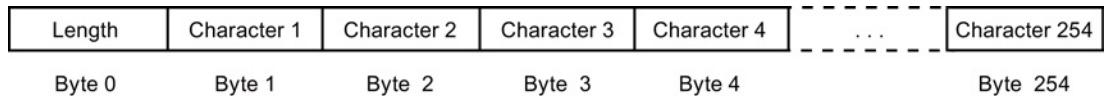
Compare instructions are executed regardless of the state of power flow.

Input / output	Data type	Operand
IN1	STRING	VB, LB, *VD, *LD, *AC, Constant string
IN2	STRING	VB, LB, *VD, *LD, *AC
OUT	BOOL	LAD: Power flow FBD: I, Q, V, M, SM, S, T, C, L, Logic Flow

Format of the STRING data type

A string variable is a sequence of characters, with each character stored as a byte. The first byte of the STRING data type defines the length of the string, which is the number of character bytes.

The diagram below shows the STRING data type stored as a variable in memory. The string can have a length of 0 to 254 characters. The maximum storage requirement for a variable string is 255 bytes (the length byte plus 254 characters).



If a constant string parameter is entered directly in the program editor (126 characters maximum) or a variable string is initialized in the data block editor (254 characters maximum), the string assignment must begin and end with double quote characters.

See also Constants (Page 63)

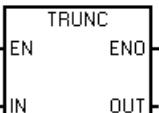
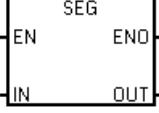
7.5 Convert

7.5.1 Standard conversion instructions

These instructions convert an input value IN to the assigned format and store the output value in the memory location assigned by OUT. For example, you can convert a double integer value to a real number. You can also convert between integer and BCD formats.

Standard conversions

LAD / FBD	STL	Description
	BTI IN, OUT	Byte to integer: Convert the byte value IN to an integer value and place the result at the address assigned to OUT. The byte is unsigned, therefore there is no sign extension.
	ITB IN, OUT	Integer to byte: Convert the word value IN to a byte value and place the result at the address assigned to OUT. Values 0 to 255 are converted. All other values result in overflow and the output is not affected. Note: To change an integer to a real number, execute the Integer to Double Integer instruction and then the Double Integer to Real instruction.
	ITD IN, OUT	Integer to double integer: Convert the integer value IN to a double integer value and place the result at the address assigned to OUT. The sign is extended.
	DTI IN, OUT	Double Integer to integer: Convert the double integer value IN to an integer value and place the result at the address assigned to OUT. If the value that you convert is too large to be represented in the output, then the overflow bit is set and the output is not affected.
	DTR IN, OUT	Double integer to real: Convert a 32-bit, signed integer IN into a 32-bit real number and place the result at the address assigned to OUT.
	BCDI OUT	BCD to Integer: Convert the binary-coded decimal WORD data type value IN to an integer WORD data type value and load the result in the address assigned to OUT. The valid range for IN is 0 to 9999 BCD.
	IBCD OUT	Integer to BCD: Convert the input integer WORD data type value IN to a binary-coded decimal WORD data type and load the result at the address assigned to OUT. The valid range for IN is 0 to 9999 integer. For STL, the IN and OUT parameters use the same address.

LAD / FBD	STL	Description
 	ROUND IN, OUT TRUNC IN, OUT	<p>Round: Convert the 32-bit real-number value IN to a double integer value and place the rounded result at the address assigned to OUT. If the fraction portion is 0.5 or greater, the number is rounded up.</p> <p>Truncate: Convert the 32-bit real-number value IN into a double integer value and place the result at the address assigned to OUT. Only the whole number portion of the real number is converted, and the fraction is discarded.</p> <p>Note: If the value that you are converting is not a valid real number or is too large to be represented in the output, then the overflow bit is set and the output is not affected.</p>
	SEG IN, OUT	<p>SEG: To illuminate the segments of a seven-segment display, the Segment instruction converts the character byte specified by IN to generate a bit pattern byte at the address assigned to OUT.</p> <p>The illuminated segments represent the character in the least significant digit of the input byte.</p>

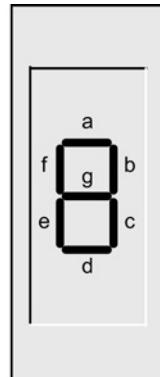
Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow • SM1.6 Invalid BCD 	<ul style="list-style-type: none"> • SM1.1 Overflow • SM1.6 Invalid BCD

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD (BCD_I, I_BCD), INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AIW, AC, *VD, *LD, *AC, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, HC, AC, *VD, *LD, *AC, Constant
	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD (BCD_I, I_BCD)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	INT (B_I, DI_I)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC
	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: SEG

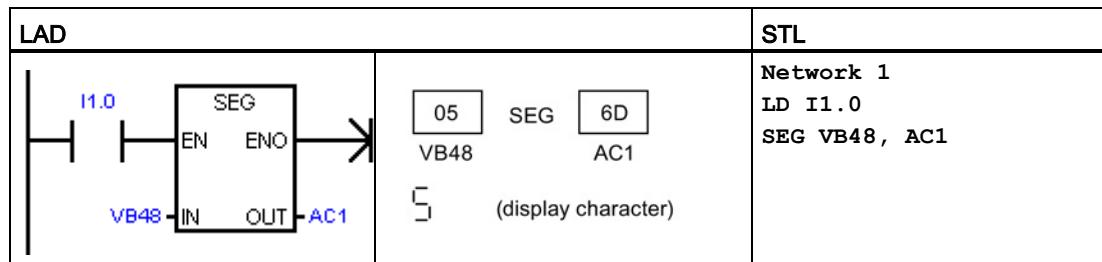
Coding for a seven-segment display

(IN) LSD	Segment Display	(OUT) - gfe dcba
0	0	0011 1111
1	1	0000 0110
2	2	0101 1011
3	3	0100 1111
4	4	0110 0110
5	5	0110 1101
6	6	0111 1101
7	7	0000 0111



(IN) LSD	Segment Display	(OUT) - gfe dcba
8	8	0111 1111
9	9	0110 0111
A	A	0111 0111
B	b	0111 1100
C	c	0011 1001
D	d	0101 1110
E	e	0111 1001
F	f	0111 0001

Display number character 5 on a seven-segment display



Examples: I_DI, DI_R, and BCD_I

LAD	STL
	Network 1 LD I0.0 ITD C10, AC1 DTR AC1, VD0 MOVR VD0, VD8 *R VD4, VD8 ROUND VD8, VD12
	Network 2 LD I3.0 BCDI AC0

See also

[Assigning a constant value for instructions](#)

See also

[Assigning a constant value for instructions \(Page 63\)](#)

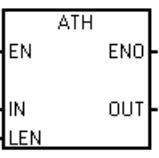
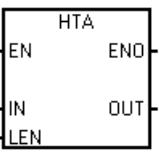
7.5.2 ASCII character array conversion

Converting from or to ASCII character byte arrays

The ASCII character array instructions use the BYTE data type for character input or output. An array of ASCII characters is referenced a sequence of byte addresses.

This is not the STRING data type, as no length byte is used. Use the ASCII string instructions to work with the variables of the STRING data type.

ASCII to Hex and Hex to ASCII

LAD / FBD	STL	Description
 	ATH IN, OUT, LEN HTA IN, OUT, LEN	<p>ATH converts a number LEN of ASCII characters, starting at IN, to hexadecimal digits starting at OUT. The maximum number of ASCII characters that can be converted is 255.</p> <p>HTA converts the hexadecimal digits, starting with the input byte IN, to ASCII characters starting at OUT. The number of hexadecimal digits to be converted is assigned by length LEN. The maximum number of ASCII characters or hexadecimal digits that can be converted is 255.</p> <p>Valid ASCII input characters are the alphanumeric characters 0 to 9 with a hexadecimal code value of 30 to 39, and uppercase characters A to F with a hex code value of 41 to 46.</p>

Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> 0006H Indirect address 0091H Operand out of range SM1.7 ATH: Illegal ASCII value 	<ul style="list-style-type: none"> SM1.7 ATH: Illegal ASCII value

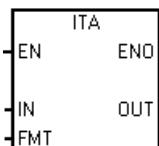
Input / output	Data type	Operand
IN, OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC
LEN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Converting number values to the ASCII character representation (ITA, DTA, and RTA)

ASCII character output number format:

- Positive values are written to the output buffer without a sign.
- Negative values are written to the output buffer with a leading minus sign (-).
- Leading zeros to the left of the decimal point (except the digit adjacent to the decimal point) are suppressed.
- Values are right-justified in the output buffer.
- Real numbers: Values to the right of the decimal point are rounded to fit in the assigned number of digits to the right of the decimal point.
- Real numbers: The size of the output buffer must be a minimum of three bytes more than the number of digits to the right of the decimal point.

Integer to ASCII

LAD / FBD	STL	Description
	ITA IN, OUT, FMT	The Integer to ASCII instruction converts the integer value IN to an array of ASCII characters. The format parameter FMT assigns the conversion precision to the right of the decimal, and whether the decimal point is to be shown as a comma or a period. The resulting conversion is placed in 8 consecutive bytes beginning with the address assigned by OUT.

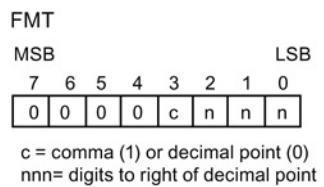
Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • FMT bit is not zero for 4 most significant bits of the FMT byte • nnn > 5 	None

Input / output	Data type	Operand
IN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

The size of the output buffer is always 8 bytes. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted with no decimal point. For values of nnn greater than 5, the output buffer is filled with ASCII space characters. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction. The most significant 4 bits must always be zero.

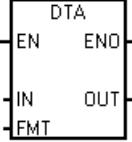
The following figure shows examples of values that are formatted using a decimal point (c=0) with three digits to the right of the decimal point (nnn=011).

FMT operand for the integer to ASCII (ITA) instruction



	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6	Out +7
in = 12				0	.	0	1	2
in = -123			-	0	.	1	2	3
in = 1234				1	.	2	3	4
in = -12345		-	1	2	.	3	4	5

Double integer to ASCII

LAD / FBD	STL	Description
	DTA IN, OUT, FMT	The Double Integer to ASCII instruction converts a double word IN to an array of ASCII characters. The format parameter FMT specifies the conversion precision to the right of the decimal. The resulting conversion is placed in 12 consecutive bytes beginning with OUT.

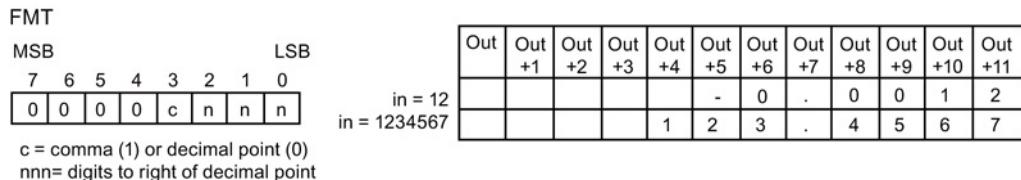
Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Invalid indirect address • 0091H Operand out of range • FMT bit is not zero for 4 most significant bits, of the FMT byte • nnn > 5 	<ul style="list-style-type: none"> • None

Input / output	Data type	Operand
IN	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

The size of the output buffer is always 12 bytes. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted with no decimal point. For values of nnn bigger than 5, the output buffer is filled with ASCII spaces. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction. The most significant 4 bits must always be zero.

The following figure shows examples of values that are formatted using a decimal point (c=0) with four digits to the right of the decimal point (nnn=100).

FMT operand for the double integer to ASCII (DTA) instruction



Real to ASCII

LAD / FBD	STL	Description
	RTA IN, OUT, FMT	The Real to ASCII instruction converts a real-number value IN to ASCII characters. The format parameter FMT specifies the conversion precision to the right of the decimal, whether the decimal point is shown as a comma or a period, and the output buffer size. The resulting conversion is placed in an output buffer beginning with OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Invalid indirect address • 0091H Operand out of range • nnn > 5 • ssss < 3 • ssss < number of characters in OUT 	None

Input / output	Data type	Operand
IN	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC

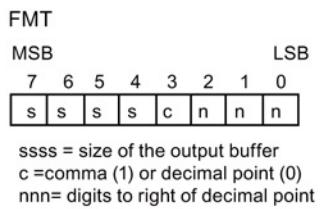
The number (or length) of the resulting ASCII characters is the size of the output buffer and can be assigned from 3 to 15 bytes or characters.

The real-number format supports a maximum of 7 significant digits. Attempting to display more than 7 significant digits produces a rounding error.

The following figure describes the format operand (FMT) for the RTA instruction. The size of the output buffer is assigned by the ssss field. A size of 0, 1, or 2 bytes is not valid. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted without a decimal point. The output buffer is filled with ASCII spaces for values of nnn greater than 5 or when the assigned output buffer is too small to store the converted value. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction.

The following figure also shows examples of values that are formatted using a decimal point (c=0) with one digit to the right of the decimal point (nnn=001) and a buffer size of six bytes (ssss=0110).

FMT Operand for the Real to ASCII (RTA) instruction



	Out	Out +1	Out +2	Out +3	Out +4	Out +5
in = 1234.5	1	2	3	4	.	5
in = -0.0004				0	.	0
in = -3.67526			-	3	.	7
in = 1.95				2	.	0

Example: ASCII to Hexadecimal

LAD	STL
<p>'3' 33 45 41 ATH [3E] Ax VB30 VB40</p>	Network 1 LD I3.2 ATH VB30, VB40, 3

- ¹ The "x" indicates that the "nibble" (half of a byte) is unchanged.

Example: Integer to ASCII

LAD	STL																				
<p>Convert the integer value at VW2 to 8 ASCII characters starting at VB10, using a format of 16#0B (a comma for the decimal point, followed by 3 digits).</p> <table border="1"> <tr> <td>12345</td> <td>ITA</td> <td>20</td> <td>20</td> <td>31</td> <td>32</td> <td>2C</td> <td>33</td> <td>34</td> <td>35</td> </tr> <tr> <td>VW2</td> <td></td> <td>VB10</td> <td>VB11</td> <td>...</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	12345	ITA	20	20	31	32	2C	33	34	35	VW2		VB10	VB11	...						Network 1 LD I2.3 ITA VW2, VB10, 16#0B
12345	ITA	20	20	31	32	2C	33	34	35												
VW2		VB10	VB11	...																	

Example: Real to ASCII

LAD	STL																								
<p>Convert the real value at VD2 to 10 ASCII characters starting at VB10, using a format of 16#A3 (a period for the decimal point, followed by 3 digits).</p> <table border="1"> <tr> <td>123.45</td> <td>RTA</td> <td>20</td> <td>20</td> <td>20</td> <td>31</td> <td>32</td> <td>33</td> <td>2E</td> <td>34</td> <td>35</td> <td>30</td> </tr> <tr> <td>VD2</td> <td></td> <td>VB10</td> <td>VB11</td> <td>...</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	123.45	RTA	20	20	20	31	32	33	2E	34	35	30	VD2		VB10	VB11	...								Network 1 LD I2.3 RTA VD2, VB10, 16#A3
123.45	RTA	20	20	20	31	32	33	2E	34	35	30														
VD2		VB10	VB11	...																					

See also

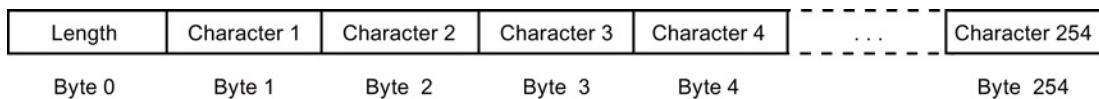
Assigning a constant value for instructions (Page 63)

7.5.3 Number value to ASCII string conversion

Format of the STRING data type

A string variable is a sequence of characters, with each character stored as a byte. The first byte of the STRING data type defines the length of the string, which is the number of character bytes.

The diagram below shows the STRING data type stored as a variable in memory. The string can have a length of 0 to 254 characters. The maximum storage requirement for a variable string is 255 bytes (the length byte plus 254 characters).



If a constant string parameter is entered directly in the program editor (126 characters maximum) or a variable string is initialized in the data block editor (254 characters maximum), the string assignment must begin and end with double quote characters.

ASCII output number format

- Positive values are written to the output buffer without a sign.
- Negative values are written to the output buffer with a leading minus sign (-).
- Leading zeros to the left of the decimal point (except the digit adjacent to the decimal point) are suppressed.
- Values are right-justified in the output string.
- Real numbers: Values to the right of the decimal point are rounded to fit in the specified number of digits to the right of the decimal point.
- Real numbers: The size of the output string must be a minimum of three bytes more than the number of digits to the right of the decimal point.

Integer to string conversion

LAD / FBD	STL	Description
	<code>ITS IN, OUT, FMT</code>	The Integer to String instruction converts an integer word IN to an ASCII string with a length of 8 characters. The format (FMT) assigns the conversion precision to the right of the decimal, and whether the decimal point is to be shown as a comma or a period. The resulting string is written to 9 consecutive bytes starting at OUT.

Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> 0006H indirect address 0091H operand out of range Illegal format (nnn > 5) FMT bit is not zero for the four most significant bits of the FMT byte 	None

Input / output	Data type	Operand
IN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	STRING	VB, LB, *VD, *LD, *AC

The length of the output string is always 8 characters. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is converted without a decimal point. For values of nnn greater than 5, the output is a string of 8 ASCII space characters. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between whole number and fraction. The most significant 4 bits of the format must be zero.

The following figure also shows examples of values that are formatted using a decimal point (c= 0) with three digits to the right of the decimal point (nnn = 011). The value at OUT is the length of the string stored in the next byte addresses.

FMT parameter for the integer to string instruction

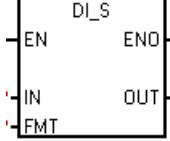
FMT

MSB	LSB							
7 6 5 4 3 2 1 0								
0	0	0	0	c	n	n	n	

c = comma (1) or decimal point (0)
nnn= digits to right of decimal point

	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6	Out +7	Out +8
in = 12	8				0	.	0	1	2
in = -123	8			-	0	.	1	2	3
in = 1234	8				1	.	2	3	4
in = -12345	8		-	1	2	.	3	4	5

Double integer to string conversion

LAD / FBD	STL	Description
	<code>DTS IN, OUT, FMT</code>	The Double Integer to String instruction converts a double integer IN to an ASCII string with a length of 12 characters. The format (FMT) assigns the conversion precision to the right of the decimal, and whether the decimal point is to be shown as a comma or a period. The resulting string is written to 13 consecutive bytes starting at OUT.

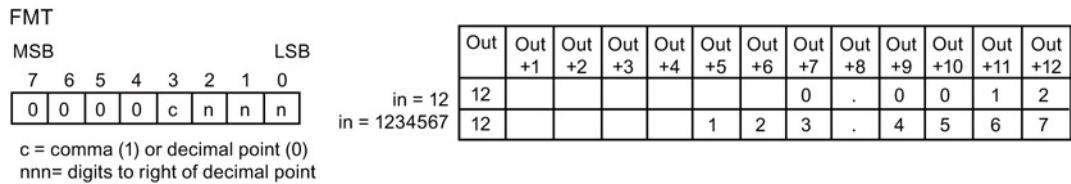
Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H indirect address • 0091H operand out of range • Illegal format (nnn > 5) • FMT bit is not zero for the four most significant bits of the FMT byte 	None

Input / output	Data type	Operand
IN	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	STRING	VB, LB, *VD, *LD, *AC

The length of the output string is always 12 characters. The number of digits to the right of the decimal point in the output buffer is specified by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point causes, then the value is displayed without a decimal point. For values of nnn greater than 5, the output is a string of 12 ASCII space characters. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between the whole number and the fraction. The upper 4 bits of the format must be zero.

The following figure also shows examples of values that are formatted using a decimal point (c= 0) with four digits to the right of the decimal point (nnn = 100). The value at OUT is the length of the string stored in the next byte addresses.

FMT operand for the double integer to string instruction



Real to string conversion

LAD / FBD		Description
	RTS IN, OUT, FMT	The Real to String instruction converts a real value IN to an ASCII string. The format (FMT) assigns the conversion precision to the right of the decimal, whether the decimal point is to be shown as a comma or a period and the length of the output string. The resulting conversion is placed in a string beginning with OUT. The length of the resulting string is specified in the format and can be 3 to 15 characters.

Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H indirect address • 0091H operand out of range • Illegal format <ul style="list-style-type: none"> – (nnn > 5) – ssss < 3 – ssss < number of characters required 	None

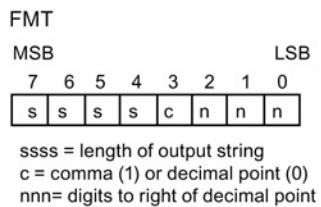
Input / output	Data type	Operand
IN	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
FMT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	STRING	VB, LB, *VD, *LD, *AC

The real-number format used by the CPU supports a maximum of 7 significant digits. An attempt to display more than the 7 significant digits produces a rounding error.

The length of the output string is specified by the ssss field. A size of 0, 1, or 2 bytes is not valid. The number of digits to the right of the decimal point in the output buffer is assigned by the nnn field. The valid range of the nnn field is 0 to 5. If you assign 0 digits to the right of the decimal point, then the value is displayed without a decimal point. The output string is filled with ASCII space characters when nnn is greater than 5 or when the assigned length of the output string is too small to store the converted value. The c bit specifies the use of either a comma (c=1) or a decimal point (c=0) as the separator between the whole number and the fraction.

The following figure also shows examples of values that are formatted using a decimal point (c= 0) with one digit to the right of the decimal point (nnn = 001) and an output string length of 6 characters (ssss = 0110). The value at OUT is the length of the string stored in the next byte addresses.

FMT operand for the real to string instruction

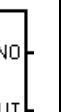
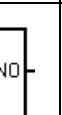
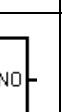


	Out	Out +1	Out +2	Out +3	Out +4	Out +5	Out +6
in = 1234.5	6	1	2	3	4	.	5
in = -0.0004	6				0	.	0
in = -3.67526	6			-	3	.	7
in = 1.95	6				2	.	0

See Also

Assigning a constant value for instructions (Page 63)

7.5.4 ASCII sub-string to number value conversion

LAD / FBD	STL	Description
	STI IN, INDX, OUT	ASCII sub-string to integer value conversion
	STD IN, INDX, OUT	ASCII sub-string to double integer value conversion
	STR IN, INDX, OUT	ASCII sub-string to real value conversion

Non-fatal error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • 009BH Index = 0 • SM1.1 Overflow or illegal value 	<ul style="list-style-type: none"> • SM1.1 Overflow or illegal value

Input / output	Data type	Operand
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
INDX	BYTE	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	INT	VW, IW, QW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC
	DINT, REAL	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

String input format for S_I (integer number) and S_DI (double integer number)

[spaces] [+ or -] [digits 0 - 9]

String input format for S_R (real number)

[spaces] [+ or -] [digits 0 - 9] [. or ,] [digits 0 - 9]

INDX parameter

The INDX value is normally set to 1, which starts the conversion with the first character of the string. The INDX value can be set to other values to start the conversion at different points within the string. This can be used when the input string contains text that is not part of the number to be converted. For example, if the input string is "Temperature: 77.8", you set INDX to a value of 13 to skip over the word "Temperature: " at the start of the string.

The Substring to Real instruction does not convert strings using scientific notation or exponential forms of real numbers. The instruction does not produce an overflow error (SM1.1) but converts the string to a real number up to the exponential and then terminates the conversion. For example, the string '1.234E6' converts without errors to a real value of 1.234.

The conversion is terminated when the end of the string is reached or when the first invalid character is found. An invalid character is any character that is not a digit (0 - 9), or one of the following characters: plus (+), minus (-), comma (,), or period (.).

The overflow error (SM1.1) is set whenever the conversion produces an integer value that is too large for the output value. For example, the Substring to Integer instruction sets the overflow error if the input string produces a value greater than 32767 or less than -32768.

The overflow error (SM1.1) is also set if no conversion is possible when the input string does not contain a valid value. For example, if the input string contains 'A123', the conversion instruction sets SM1.1 (overflow) and the output value remains unchanged.

Examples of valid and invalid input stringsValid Input Strings
for Integer and Double Integer

Input String	Output Integer
'123'	123
'-00456'	-456
'123.45'	123
'+2345'	2345
'000000123ABCD'	123

Valid Input Strings
for Real Numbers

Input String	Output Real
'123'	123.0
'-00456'	-456.0
'123.45'	123.45
'+2345'	2345.0
'00.000000123'	0.000000123

Invalid Input Strings

Input String
'A123'
' '
'++123'
'+-123'
'+ 123'

Example string conversion: Substring to integer, double integer, and real

LAD	STL												
<p>S_I converts the numeric string to an integer value.</p> <p>S_DI converts the numeric string to a double integer value.</p> <p>S_R converts the numeric string to a real value.</p>	Network 1 LD I0.0 STI VB0, 7, VW100 STD VB0, 7, VD200 STR VB0, 7, VD300												
<p>VB0</p> <table border="1"> <tr> <td>11</td> <td>'T'</td> <td>'e'</td> <td>m'</td> <td>'p'</td> <td>''</td> <td>''</td> <td>'9'</td> <td>'8'</td> <td>'.'</td> <td>'6'</td> <td>'F'</td> </tr> </table> <p>VB11</p>	11	'T'	'e'	m'	'p'	''	''	'9'	'8'	'.'	'6'	'F'	
11	'T'	'e'	m'	'p'	''	''	'9'	'8'	'.'	'6'	'F'		

After executing the network:

VW100 (integer) = 98
VD200 (double integer) = 98
VD300 (real) = 98.6

See also

Assigning a constant value for instructions (Page 63)

7.5.5 Encode and decode

LAD / FBD	STL	Description
	ENCO IN, OUT	Encode writes the bit number of the least significant bit set in the input word IN, to the least significant "nibble" (4 bits) of the output byte OUT.
	DECO IN, OUT	Decode sets the bit in the output word OUT that corresponds to the bit number represented by the least significant "nibble" (4 bits) of the input byte IN. All other bits of the output word are set to 0.

Non-fatal error conditions with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
IN	WORD (ENCO)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	BYTE (DECO)	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	BYTE (ENCO)	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD (DECO)	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC

Program instructions

7.6 Counters

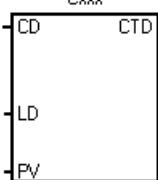
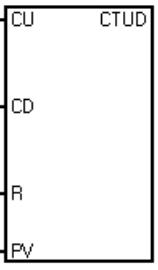
Example: Encode and decode

LAD	STL					
<p>If AC2 contains error bits:</p> <ol style="list-style-type: none"> 1. The DECO instruction sets the bit in VW40 that corresponds to this error code 2. The ENCO instruction converts the least significant bit set to an error code that is stored in VB50. 	Network 1 LD I3.1 DECO AC2, VW40 ENCO AC3, VB50					
<p>AC2 <table border="1"><tr><td>3</td></tr></table></p> <p>VW40 <table border="1"><tr><td>15 0000 0000 0000 1000</td></tr></table></p>	3	15 0000 0000 0000 1000	<p>AC3 <table border="1"><tr><td>15 9 0</td><td>1000 0010 0000 0000</td></tr></table></p> <p>ENCO</p> <p>VB50 <table border="1"><tr><td>9</td></tr></table></p>	15 9 0	1000 0010 0000 0000	9
3						
15 0000 0000 0000 1000						
15 9 0	1000 0010 0000 0000					
9						

7.6 Counters

7.6.1 Counter instructions

LAD / FBD	STL	Description
<p>Cxxx</p>	CTU Cxxx, PV	<p>LAD/FBD: The CTU count up instruction counts up from the current value each time the count up CU input makes the transition from OFF to ON. When the current value Cxxx is greater than or equal to the preset value PV, the counter bit Cxxx is set ON. The current count value is reset when the reset input R is set ON, or when the reset instruction is executed for the Cxxx address. The counter stops counting when it reaches the maximum value 32,767.</p> <p>STL: R reset input is the top of stack value. CU count up input is loaded in the second stack level</p>

LAD / FBD	STL	Description
	CTD Cxxx, PV	<p>LAD/FBD: The CTD count down instruction counts down from the current value of that counter each time the CD count down input makes the transition from OFF to ON. When the current value Cxxx is equal to 0, the counter bit Cxxx turns ON. The counter resets the counter bit Cxxx and loads the current value with the preset value PV when the LD load input is set ON. The counter stops upon reaching zero, and the counter bit Cxxx is set ON.</p> <p>STL: LD load input is the top of stack value. CD count down input value is loaded in the second stack level</p>
	CTUD Cxxx, PV	<p>LAD/FBD: The CTUD count up/down instruction counts up each time the CU count up input makes the transition from OFF to ON, and counts down each time the CD count down input makes the transition from OFF to ON. The current value Cxxx of the counter maintains the current count. The PV preset value is compared to the current value each time the counter instruction is executed.</p> <p>Upon reaching maximum value 32,767, the next rising edge at the count up input causes the current count to wrap around to the minimum value -32,768. On reaching the minimum value -32,768, the next rising edge at the count down input causes the current count to wrap around to the maximum value 32,767.</p> <p>When the current value Cxxx is greater than or equal to the PV preset value, the counter bit Cxxx is set ON. Otherwise, the counter bit is OFF. The counter is reset when the R reset input is set ON, or when the Reset instruction is executed for the Cxxx address.</p> <p>STL: R reset input is the top of stack value. CD count down input value is loaded in the second stack level. CU count Up input value is loaded in the third stack level</p>

Input / output	Data type	Operand
Cxxx	WORD	Constant (C0 to C255)
CU, CD (LAD)	BOOL	Power flow
CU, CD (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
R (LAD)	BOOL	Power Flow
R (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
LD (LAD)	BOOL	Power Flow
LD (FBD)	BOOL	I, Q, V, M, SM, S, T, C, L, Logic flow
PV	INT	IW, QW, VW, MW, SMW, SW, LW, T, C, AC, AIW, *VD, *LD, *AC, Constant

Note

Since there is one current value for each counter, do not assign the same counter number to more than one counter. (Up Counters, Up/Down Counters, and Down counters with the same number access the same current value.)

When you reset a counter using the Reset instruction, the counter bit is reset and the counter current value is set to zero. Use the counter number to reference both the current value and the counter bit of that counter.

See also Configuring the retentive ranges - system block configuration

Counter operation

Type	Operation	Counter bit	Power cycle / first scan
CTU	<ul style="list-style-type: none"> CU increments the current value. Current value continues to increment until it reaches 32,767. 	The counter bit is set ON when: Current value \geq Preset	<ul style="list-style-type: none"> Counter bit is OFF. Current value can be retained ¹
CTD	<ul style="list-style-type: none"> CD decrements the current value until the current value reaches 0. 	The counter bit is set ON when: Current value = 0	<ul style="list-style-type: none"> Counter bit is OFF. Current value can be retained ¹
CTUD	<ul style="list-style-type: none"> CU increments the current value. CD decrements the current value. Current value continues to increment or decrement until the counter is reset. 	The counter bit is set ON when: Current value \geq Preset	<ul style="list-style-type: none"> Counter bit is OFF. Current value can be retained ¹

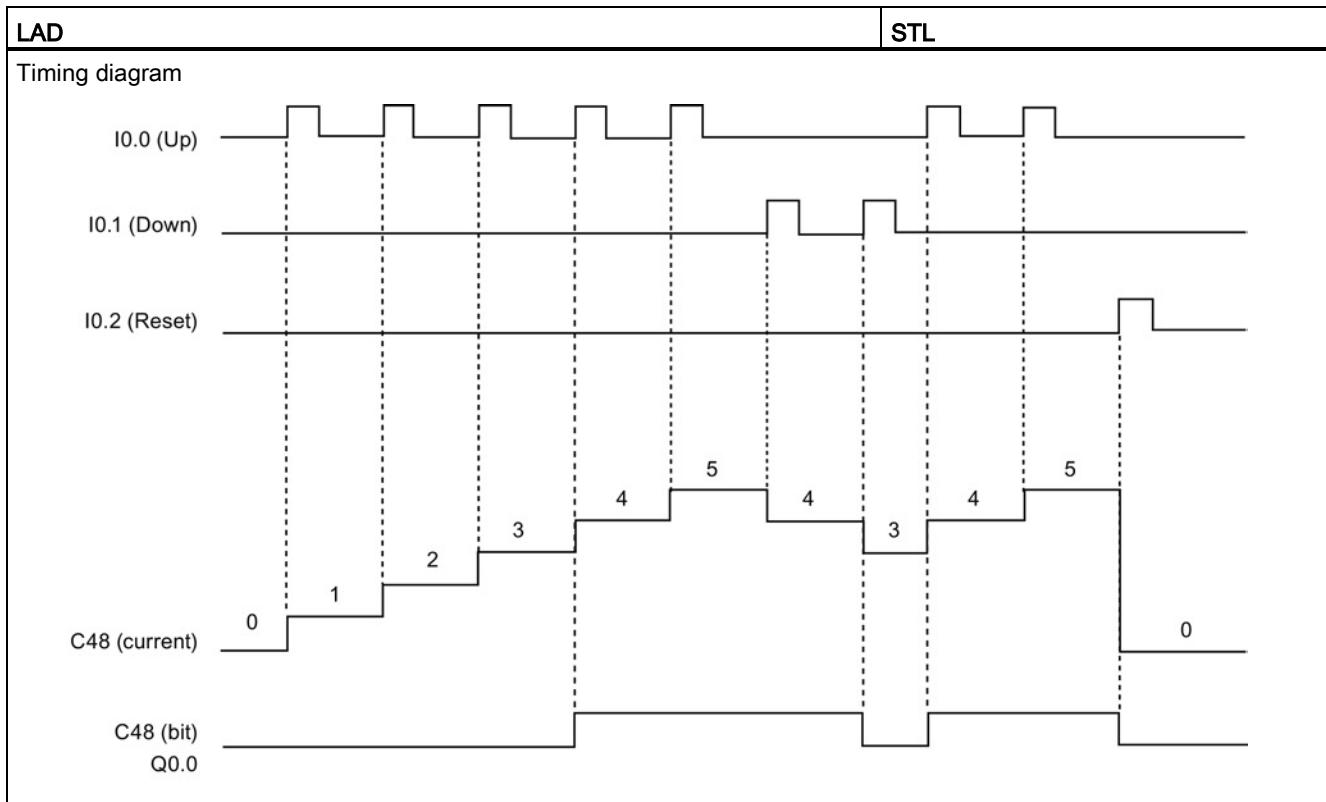
¹ You can select the current value for the counter to be retentive, but not the counter bit value.

Example CTD count down

LAD	STL
	<p>Count down counter C1 current value counts from 3 to 0 With I0.1 OFF, I0.0 OFF-ON decrements C1 current value I0.1 ON loads countdown preset value 3</p> <p>Network 1 LD I0.0 LD I0.1 CTD C1, +3</p>
	<p>C1 bit is ON when counter C1 current value = 0</p> <p>Network 2 LD C1 = Q0.0</p>
<p>Timing diagram</p>	

Example CTUD count up/down

LAD	STL
	<p>I0.0 counts up I0.1 counts down I0.2 resets current value to 0</p> <p>Network 1 LD I0.0 LD I0.1 LD I0.2 CTUD C48, +4</p>
	<p>Count Up/Down counter C48 turns on C48 bit when current value >= 4</p> <p>Network 2 LD C48 = Q0.0</p>



7.6.2 High-speed counter instructions

High-speed counters can count high-speed events that cannot be controlled by standard counters. Standard counters operate at a slower rate that is limited by the PLC scan time. You can use the HDEF and HSC instructions and create your own HSC routines, or you can simplify the programming tasks by using the High Speed Counter wizard.

LAD / FBD	STL	Description
	HDEF HSC, MODE	The High-Speed Counter Definition instruction (HDEF) selects the operating mode of a specific high-speed counter (HSC0-3). The mode selection defines the clock, direction, and reset functions of the high-speed counter. You must use one High-Speed Counter Definition instruction for each of up to four active high-speed counters.
	HSC N	The High-Speed Counter (HSC) instruction configures and controls the high-speed counter, based on the state of the HSC special memory bits. The parameter N specifies the high-speed counter number. The high-speed counters can be configured for up to eight different modes of operation. Each counter has dedicated inputs for clocks, direction control, and reset where these functions are supported. In AB quadrature phase, you can select one times (1x) or four times (4x) the maximum counting rate. All counters run at maximum rates without interfering with one another.

Error conditions with ENO = 0	SM bits affected
HDEF: • 0003H Input point conflict • 0004H Illegal instruction in interrupt • 000AH HSC redefinition • 0016H Attempted to use HSC or Edge Interrupt on Input that is allocated for use by Motion Functionality • 0090H Invalid HSC number	HSC: • 0001H HSC before HDEF • 0005H Simultaneous HSC/PLS • 0090H Invalid HSC number

Input / output	Data type	Operand
HSC	BYTE	HSC number constant (0, 1, 2, or 3)
MODE	BYTE	Mode number constant: Eight possible modes (0, 1, 3, 4, 6, 7, 9, or 10)
N	WORD	HSC number constant (0, 1, 2, or 3)

HSC operation

A high-speed counter can be used as the drive for a drum timer, where a shaft rotating at a constant speed is fitted with an incremental shaft encoder. The shaft encoder provides a specified number of counts per revolution and a reset pulse that occurs once per revolution. The clock(s) and the reset pulse from the shaft encoder provide the inputs to the high-speed counter.

The high-speed counter is loaded with the first of several presets, and the desired outputs are activated for the time period where the current count is less than the current preset. The counter is set up to provide an interrupt when the current count is equal to preset and also when reset occurs.

As each current-count-value-equals-preset-value interrupt event occurs, a new preset is loaded and the next state for the outputs is set. When the reset interrupt event occurs, the first preset and the first output states are set, and the cycle is repeated.

Since the program interrupts occur at a much lower rate than the counting rates of the high-speed counters, precise control of high-speed operations can be implemented with relatively minor impact to the overall PLC scan cycle time. The method of interrupt attachment allows each load of a new preset to be performed in a separate interrupt routine for easy state control. (Alternatively, all interrupt events can be processed in a single interrupt routine.)

HSC input assignments and capabilities

All high-speed counters function the same way for the same mode of operation, but every mode is not supported for every HSC number. The HSC input connections (clock, direction, and reset) must use the CPU's integrated input channels as shown in the following table. Input channels located on a signal board or an expansion module cannot be used for high-speed counters.

Note

You must ensure that high-speed counter inputs are correctly filtered and wired, for counting high frequency signals.

In an S7-200 SMART CPU, all high-speed counter inputs are connected to internal input filter circuits. The S7-200 SMART default input filter setting is 6.4 ms, which limits the maximum counting rate to 78 Hz. You must change the filter settings to count higher frequencies.

Refer to "Noise reduction for high-speed inputs (Page 214)" for details about system block filter options, maximum counting frequencies, shielding requirements, and external pull-down circuits.

	Clock A	Dir / Clock B	Reset	Single phase max. clock / input rate	Dual phase / AB quadrature phase max. clock / input rate
HSC0	I0.0	I0.1	I0.4	200 kHz (S model CPUs) ¹ 100 kHz (C model CPUs) ²	S model CPUs: 100 kHz = Maximum 1x count rate 400 kHz = Maximum 4x count rate C model CPUs: 50 kHz = Maximum 1x count rate 200 kHz = Maximum 4x count rate
HSC1	I0.1			200 kHz (S model CPUs) 100 kHz (C model CPUs)	
HSC2	I0.2	I0.3	I0.5	200 kHz (S model CPUs) 100 kHz (C model CPUs)	S model CPUs: 100 kHz = Maximum 1x count rate 400 kHz = Maximum 4x count rate C model CPUs: 50 kHz = Maximum 1x count rate 200 kHz = Maximum 4x count rate
HSC3	I0.3			200 kHz (S model CPUs) 100 kHz (C model CPUs)	

¹ S model CPUs: SR20, ST20, SR30, ST30, SR40, ST40, SR60, ST60

² C model CPUs: CR40, CR60

HSC counting mode support

- A total of four HSC devices can be used (HSC0, HSC1, HSC2, and HSC3)
- HSC0 and HSC2 support eight counter modes (mode 0, 1, 3, 4, 6, 7, 9, and 10)
- HSC1 and HSC3 can only support one counter mode (mode 0)

Available HSC counter types

- Single-phase clock counter with internal direction control
 - Mode 0:
 - Mode 1: with external reset
- Single-phase clock counter with external direction control
 - Mode 3:
 - Mode 4: with external reset
- Two-phase clock counter with 2 clock inputs (clock-up and clock-down)
 - Mode 6:
 - Mode 7: with external reset
- AB quadrature phase counter
 - Mode 9:
 - Mode 10: with external reset

HSC operating rules

- Before you use a high-speed counter, you must execute the HDEF instruction (High-Speed Counter Definition) to select a counter mode. Use the first scan memory bit, SM0.1 (this bit is ON for the first scan and OFF for subsequent scans) to execute HDEF directly, or call a subroutine that contains the HDEF instruction.
- You can use all counter types with or without a reset input.
- When you activate the reset input, it clears the current value and holds it clear until you deactivate the reset input.

See also

[High-speed counter programming examples \(Page 216\)](#)

[High-speed counter initialization sequence examples \(Page 228\)](#)

[Noise reduction for high-speed inputs \(Page 214\)](#)

7.6.3 Noise reduction for high-speed inputs

Counting high-speed pulses with HSC inputs

Note

High-speed input wiring must use shielded cables

Use shielded cable with a maximum length of 50 m, when connecting HSC input channels I0.0, I0.1, I0.2, and I0.3.

One or both of the following actions may be necessary to correctly operate a high-speed counter.

- Adjust the System Block digital input filter time of the input channels used by the HSC channel. In an S7-200 SMART CPU, Input filtering is applied before the counting of pulses by the HSC channel. This means that if an HSC input pulse occurs at a rate that is filtered out by the input filtering, then the HSC does not detect any pulses on the input. You must make sure that you configure the filter time of each input of the HSC to a value that will allow counting at the rate your application requires. This includes direction and reset inputs. The following table shows the maximum input frequency that can be detected for each input filter configuration.

Input filter time	Maximum detectable frequency
0.2 µs	200 kHz for S model CPUs ¹ 100 kHz for C model CPUs ²
0.4 µs	200 kHz for S model CPUs 100 kHz for C model CPUs
0.8 µs	200 kHz for S model CPUs 100 kHz for C model CPUs
1.6 µs	200 kHz for S model CPUs 100 kHz for C model CPUs
3.2 µs	156 kHz for S model CPUs 100 kHz for C model CPUs
6.4 µs	78 kHz
12.8 µs	39 kHz
0.2 ms	2.5 kHz
0.4 ms	1.25 kHz
0.8 ms	625 Hz
1.6 ms	312 Hz
3.2 ms	156 Hz
6.4 ms	78 Hz
12.8 ms	39 Hz

¹ S model CPUs: SR20, ST20, SR30, ST30, SR40, ST40, SR60, ST60

² C model CPUs: CR40, CR60

- If the device generating the HSC input signals does not drive the input signals both high and low, then signal distortion can occur at high speeds. This can occur if the output of the device is an open-collector transistor. When the transistor turns off, there is nothing driving the signal to a low state. The signal will transition to a low state, but the time to do so will be dependant on the input resistance and capacitance of the circuitry. This condition can result in missed pulses. This condition can be prevented by wiring a pull-down resistor to the input signals as seen in the following figure. Since the input voltage of the CPU is 24V, the resistor would have to be rated for a high wattage. A 100 ohm 5 Watt resistor is a suitable choice.

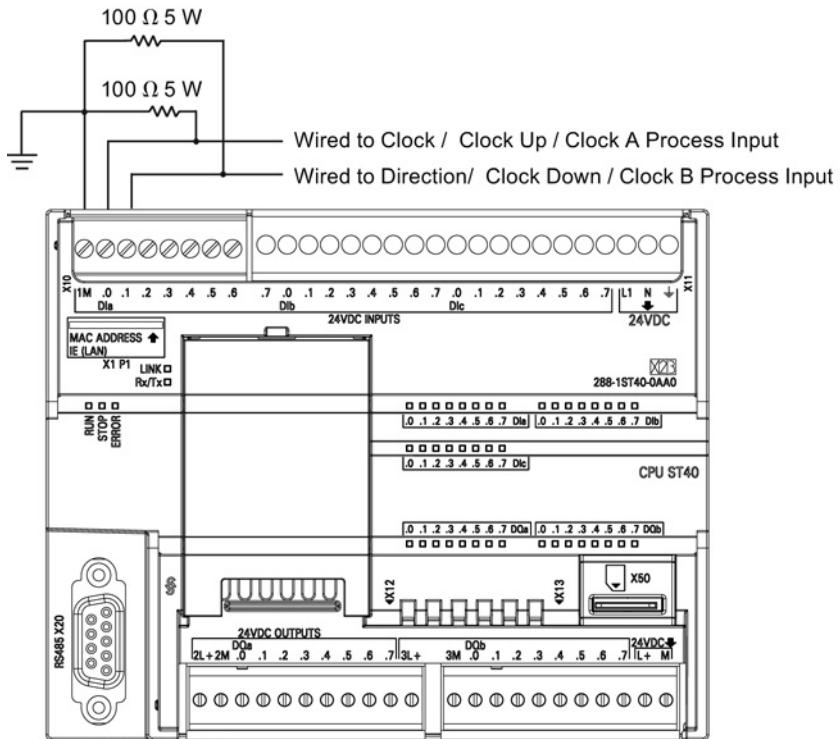


Figure 7-1 Pull-down resistor wiring for open-collector HSC input drivers

7.6.4 High-speed counter programming

You can use the high-speed counter wizard to simplify HSC programming tasks. The wizard helps you select the counter type mode, preset/current values, counter options, and will generate the necessary special memory assignments, subroutines, and interrupt routines.

Note

You must ensure that high-speed counter inputs are correctly filtered and wired, for counting high frequency signals.

In an S7-200 SMART CPU, all high-speed counter inputs are connected to internal input filter circuits. The S7-200 SMART default input filter setting is 6.4 ms, which limits the maximum counting rate to 78 Hz. You must change the filter settings to count higher frequencies.

Refer to "Noise reduction for high-speed inputs (Page 214)" for details about system block filter options, maximum counting frequency, shielding requirements, and external pull-down circuits.

Configuring high-speed counters



Use one of the following actions to configure the high-speed counter wizard:

- Open the wizard: Select "High-Speed Counter" in the wizards area of the Tools menu ribbon strip.
- Open the wizard: Double-click "High-Speed Counter" node in the wizards folder, from the project tree.

With the wizard open, assign the HSC setup values. You can navigate through the wizard setup pages, modify parameters, and then generate new wizard program code.

Your program must perform the following basic tasks to use a high-speed counter:

- Define the counter and mode (execute the HDEF instruction exactly once for each counter).
- Set the control byte in SM memory.
- Set the current value (starting value) in SM memory.
- Set the preset value (target value) in SM memory.
- Assign and enable appropriate interrupt routines.
- Activate the high-speed counter (execute the HSC instruction).

HDEF instruction sets the counting mode

The HDEF instruction assigns HSC counter mode. The following table shows the physical inputs assigned for clock, direction control, and reset functions. The same input cannot be used for two different functions, but any input not being used by the present mode of its high-speed counter can be used for another purpose. For example, if HSC0 is used in mode 1, which uses I0.0 and I0.4; then I0.1, I0.2, and I0.3 can be used for edge interrupts, HSC3, or motion control inputs.

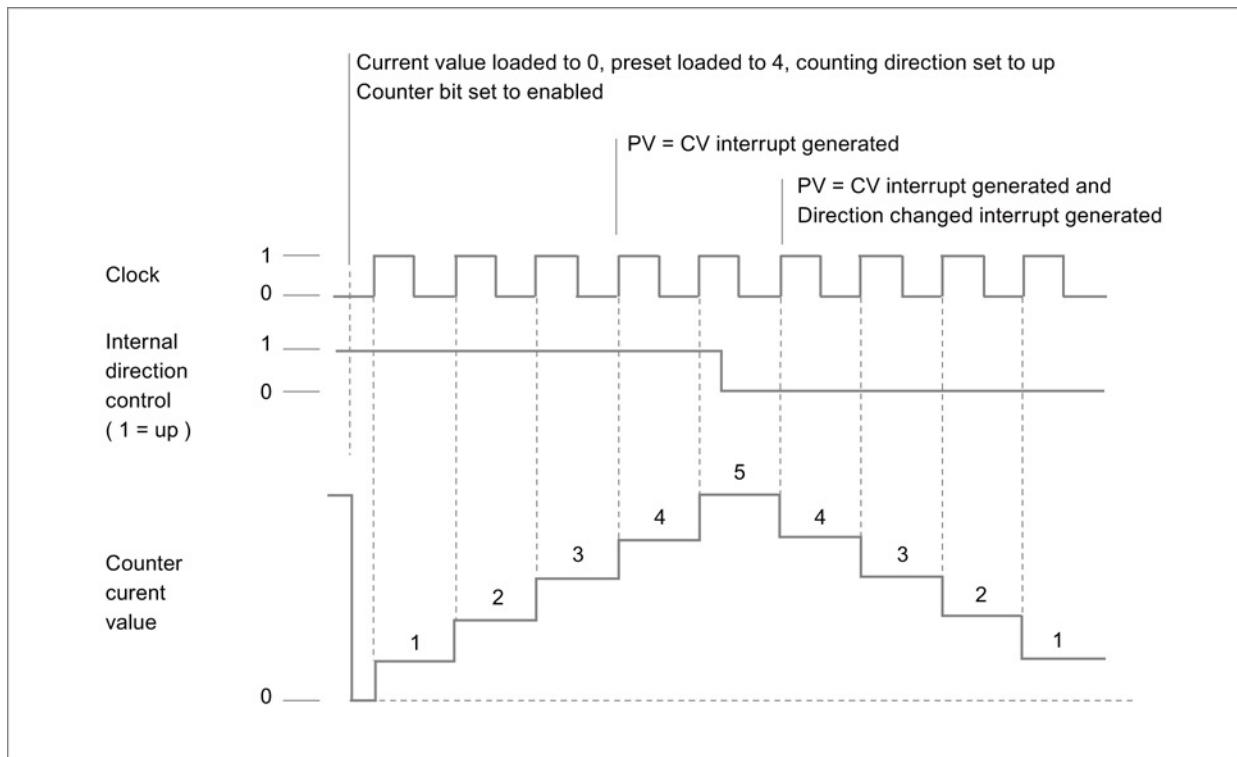
Note

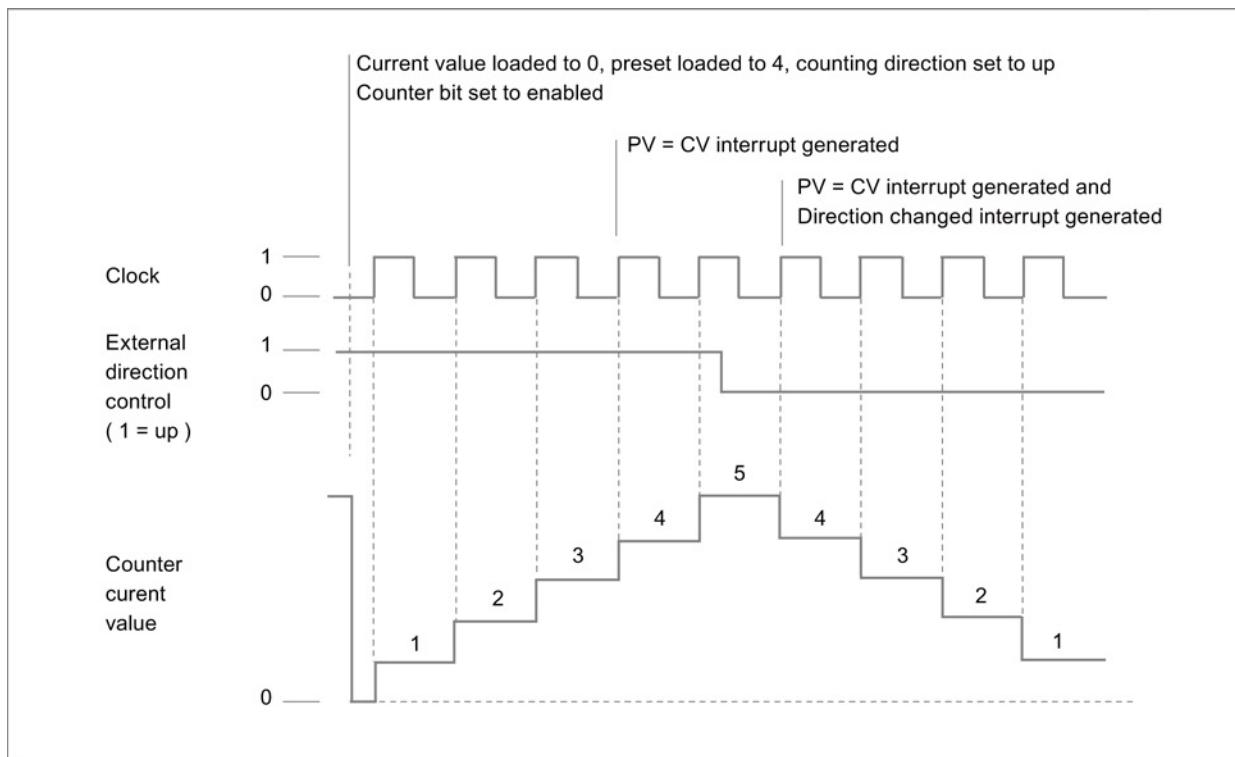
All counting modes of HSC0 always use I0.0 and all modes of HSC2 always use I0.2, so these inputs are never available for other uses when these counters are in use.

Mode	Description	Input assignment		
	HSC0	I0.0	I0.1	I0.4
	HSC1	I0.1		
	HSC2	I0.2	I0.3	I0.5
	HSC3	I0.3		
0	Single-phase counter with internal direction control	Clock		
1		Clock		Reset
3	Single-phase counter with external direction control	Clock	Direction	
4		Clock	Direction	Reset
6	Two-phase counter with 2 clock inputs	Clock Up	Clock Down	
7		Clock Up	Clock Down	Reset
9	AB quadrature phase counter	Clock A	Clock B	
10		Clock A	Clock B	Reset

How mode selection affects counter operation

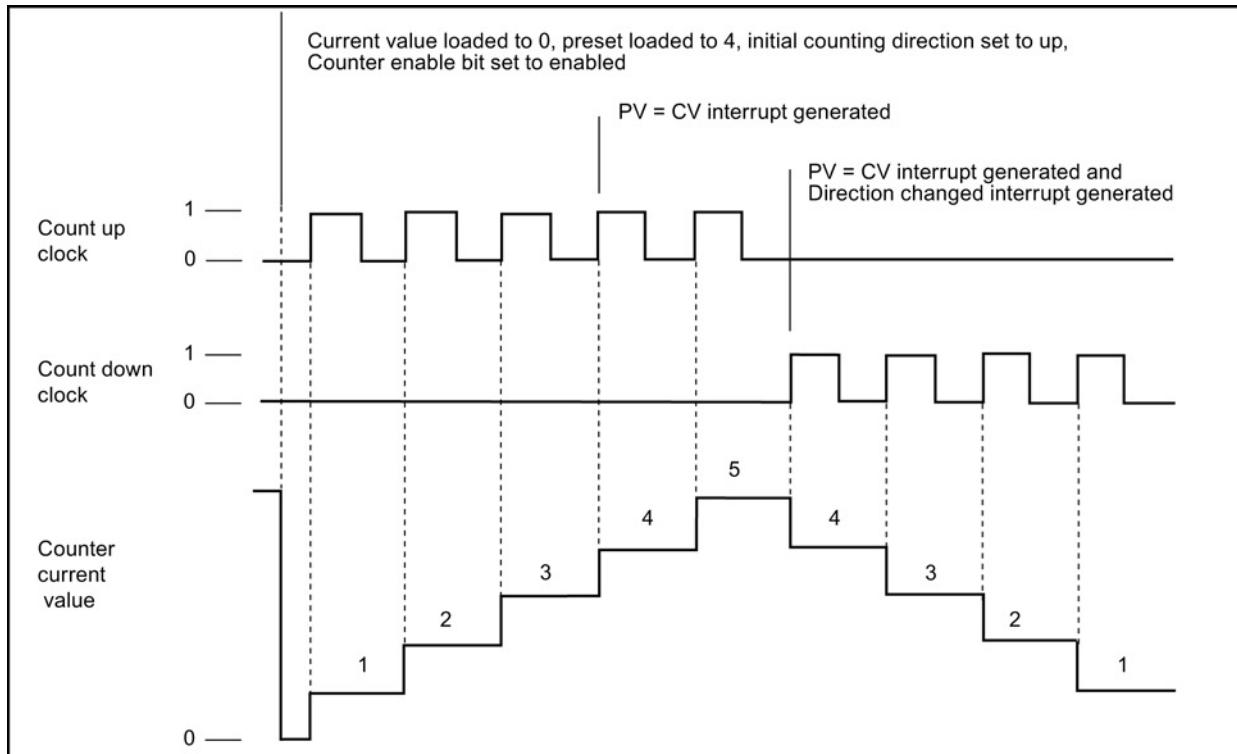
HSC modes 0 and 1



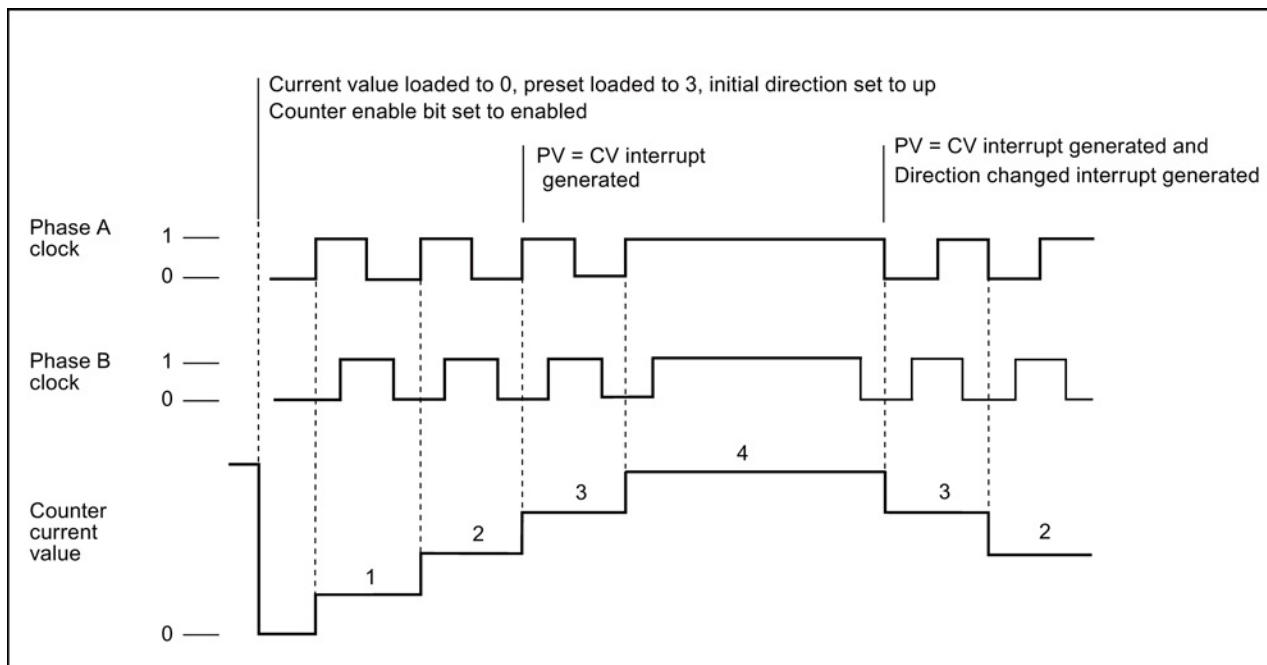
HSC modes 3 and 4

HSC modes 6 and 7

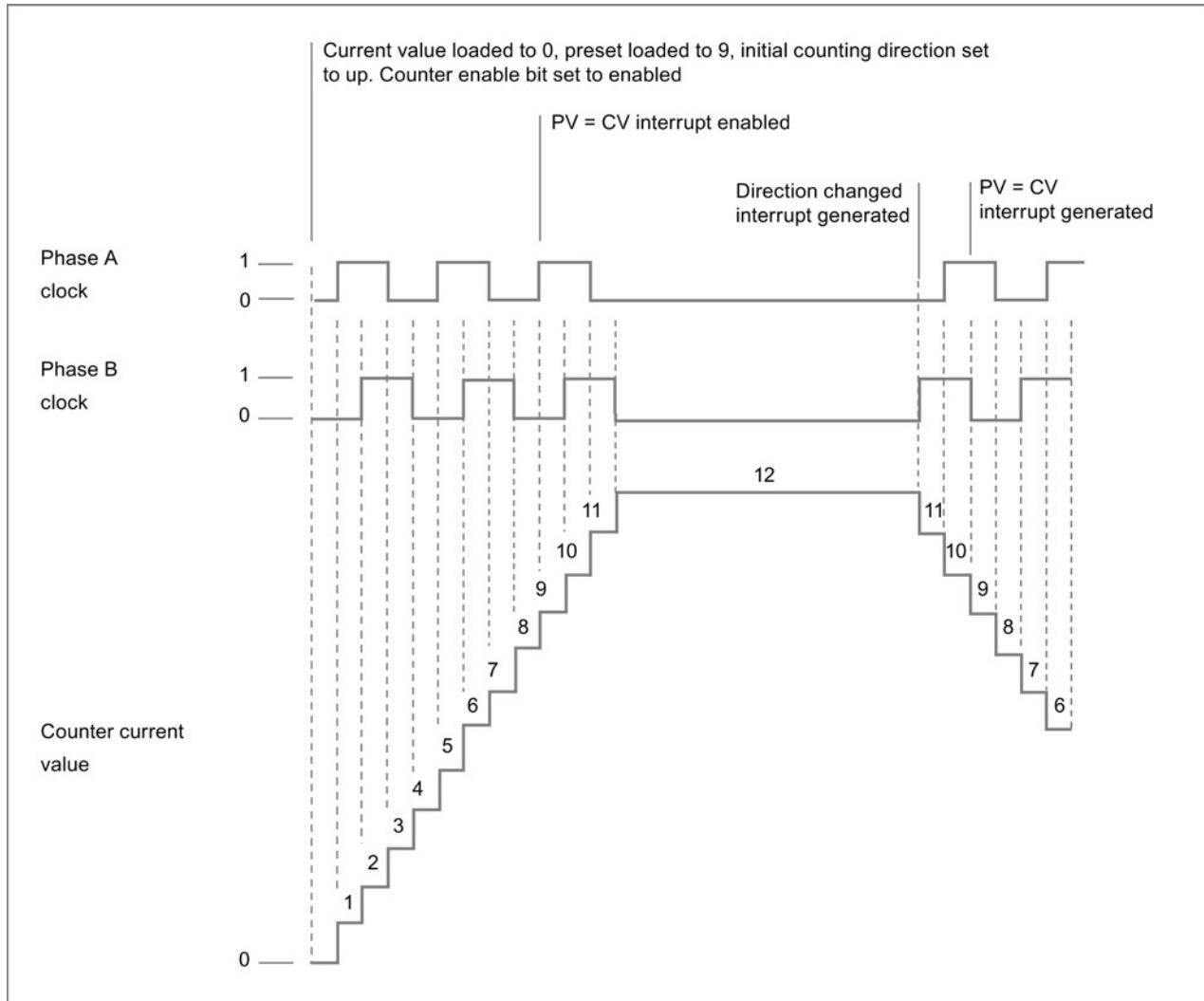
When you use counting modes 6 or 7, and rising edges on both the up clock and down clock inputs occur within 0.3 microseconds of each other, the high-speed counter could see these events as happening simultaneously. If this happens, the current value is unchanged and no change in counting direction is indicated. As long as the separation between rising edges of the up and down clock inputs is greater than this time period, the high-speed counter captures each event separately. In either case, no error is generated and the counter maintains the correct count value.



HSC modes 9 and 10 (AB quadrature phase 1x)



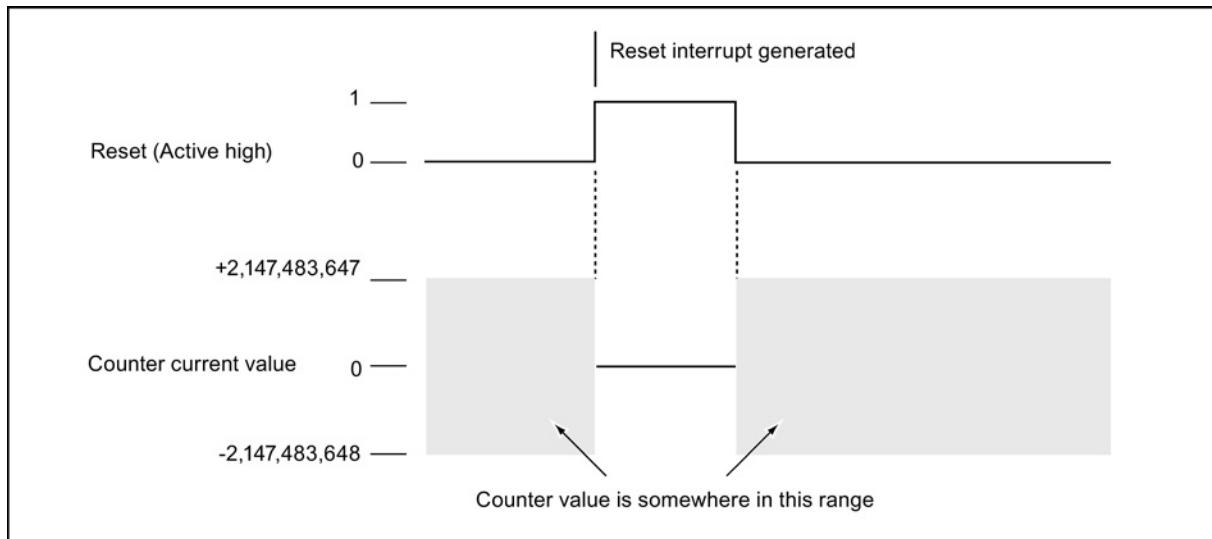
HSC modes 9 and 10 (AB quadrature phase 4x)



Reset operation

The operation of reset shown in the following figure applies to all modes that use the reset input. In the figure below, the reset operation is shown with the active state assigned as the high level.

HSC reset



HDEF instruction sets the reset active level and counting rate

HSC0 and HSC2 counters have two control bits that are used to configure the active state of the reset and to select 1x or 4x counting modes (AB quadrature phase counters only). These bits are located in the HSC control byte for the respective counter and are only used when the HDEF instruction is executed. These bits are defined in the following table.

Note

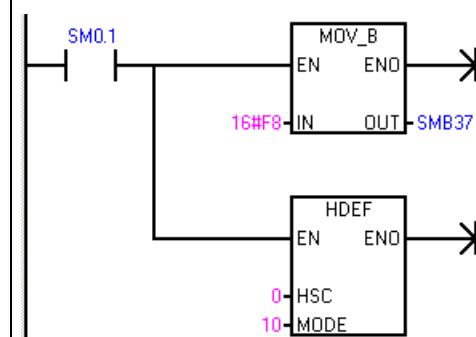
You must set these two control bits to the desired state before the HDEF instruction is executed. Otherwise, the counter takes on the default configuration for the counter mode selected.

Once the HDEF instruction has been executed, you cannot change the counter setup unless you first place the CPU in STOP mode.

HSC0	HSC1	HSC2	HSC3	Description (used only when HDEF is executed)
SM37.0	Not supported	SM57.0	Not supported	Active level control bit for Reset¹: <ul style="list-style-type: none"> • 0 = Reset is active high • 1 = Reset is active low
SM37.2	Not supported	SM57.2	Not supported	Counting rate selection for AB quadrature phase counters¹: <ul style="list-style-type: none"> • 0 = 4X counting rate • 1 = 1X counting rate

¹ The default setting of the reset input is active high, and the AB quadrature phase counting rate is 4x (or four times the input clock frequency).

Example: High-speed counter definition

LAD	STL
	<p>On the first scan:</p> <ol style="list-style-type: none"> 1. Select the reset input to be active high and select 4x mode. 2. Configure HSC0 for AB quadrature phase with reset input (mode 10). <pre> Network 1 LD SM0.1 MOVB 16#F8, SMB37 HDEF 0, 10 </pre>

HSC instruction enables counters, sets counting direction, and loads preset/current count values

The HSC instruction uses the control byte during execution. After you assign the counter and the counter mode, you can program the dynamic parameters of the counter. Each high-speed counter has a control byte in SM memory that allows the following actions:

- Enabling or disabling the counter
- Controlling the direction (modes 0 and 1 only), or the initial counting direction for all other modes
- Loading the current value
- Loading the preset value

HSC Control bytes

HSC0	HSC1	HSC2	HSC3	Description
SM37.3	SM47.3	SM57.3	SM137.3	Counting direction control bit: <ul style="list-style-type: none"> • 0 = Count down • 1 =Count up
SM37.4	SM47.4	SM57.4	SM137.4	Write the counting direction to the HSC: <ul style="list-style-type: none"> • 0 = No update • 1 =Update direction
SM37.5	SM47.5	SM57.5	SM137.5	Write the new preset value to the HSC: <ul style="list-style-type: none"> • 0 = No update • 1 = Update preset
SM37.6	SM47.6	SM57.6	SM137.6	Write the new current value to the HSC: <ul style="list-style-type: none"> • 0 = No update • 1 =Update current value
SM37.7	SM47.7	SM57.7	SM137.7	Enable the HSC: <ul style="list-style-type: none"> • 0 = Disable the HSC • 1 =Enable the HSC

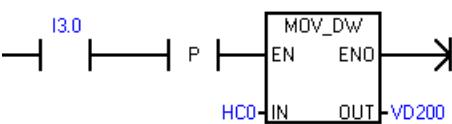
Read the HSC current value with your program

The current value of each high-speed counter can only be read using the data type HC (High-Speed-Counter Current) followed by the counter identifier number (0, 1, 2, or 3) as shown in the following table. Use the HC data type whenever you wish to read the current count, either in a status chart or in the user program. The HC data type is read-only double word value; you cannot write a new current count to the high speed counter using the HC data type.

Table 7- 7 Current values of HSC0, HSC1, HSC2, and HSC3

Value to be read	HSC0 address	HSC1 address	HSC2 address	HSC3 address
CV (counter current value)	HC0	HC1	HC2	HC3

Example: Reading and saving the current count value

LAD	STL
MAIN 	Network 1 LD I3.0 EU MOVD HCO, VD200

Set current values and preset values with your program

Each high-speed counter has a 32-bit current value (CV) and a 32-bit preset value (PV) stored internally. The current value is the actual count value of the counter, while the preset value is a comparison value optionally used to trigger an interrupt when the current value reaches the preset value. You can read the current value using the HC data type as described in the previous section. You cannot read the preset value directly. To load a new current or preset value into the high-speed counter, you must set up the control byte and the special memory double-word(s) that hold the desired new current and/or new preset values, and also execute the HSC instruction to cause the new values to be transferred to the high-speed counter. The table below lists the special memory double words used to hold the desired new current and preset values.

Use the following steps to write a new current value and/or new preset value to the high-speed counter (steps 1 and 2 can be done in either order):

1. Load the value to be written into the appropriate SM new-current value and/or new preset value (see the table below). Loading these new values does not affect the high-speed counter yet.
2. Set or clear the appropriate bits in the appropriate control byte to indicate whether to update the current and/or preset values (bit x.5 for preset and x.6 for current). Manipulating these bits does not affect the high-speed counter yet.
3. Execute the HSC instruction referencing the appropriate high-speed counter number. Executing this instruction causes the control byte to be examined. If the control byte specifies an update for the current, the preset, or both, then the appropriate values are copied from the SM new current value and/or new preset value locations into the high-speed counter internal registers.

Value to be loaded	HSC0	HSC1	HSC2	HSC3
New current value (new CV)	SMD38	SMD48	SMD58	SMD138
New preset value (new PV)	SMD42	SMD52	SMD62	SMD142

Note

Changes to the control byte and the SM locations for new current value and new preset value will not affect the high-speed counter until the corresponding HSC instruction is executed.

Example: Updating the current and preset values

LAD	STL
<p>MAIN program network</p>	<p>Update the current count to 1000 and the preset value to 2000 for HSC0 when I2.0 transitions from OFF to ON.</p> <pre> Network 1 LD I2.0 EU MOVD 1000, SMD38 MOVD 2000, SMD42 = SM37.5 = SM37.6 HSC 0 </pre>

Attaching HSC interrupt routines in your program

All counter modes support an interrupt event when the current value of the HSC is equal to the loaded preset value. Counter modes that use an external reset input support an interrupt on activation of the external reset. All counter modes except modes 0 and 1 support an interrupt on a change in counting direction. Each of these interrupt conditions can be enabled or disabled separately. For a complete discussion on the use of interrupts, see the section about Interrupt instructions (Page 263).

HSC status byte

A status byte for each high-speed counter provides status memory bits that indicate the current counting direction and whether the current value is greater or equal to the preset value. The following table defines these status bits for each high-speed counter.

Note

Status bits are valid only while the high-speed counter interrupt routine is being executed. The purpose of monitoring the state of the high-speed counter is to enable interrupts for the events that are of consequence to the operation being performed.

Table 7- 8 Status bits for HSC0, HSC1, HSC2, and HSC3

HSC0	HSC1	HSC2	HSC3	Description
SM36.5	SM46.5	SM56.5	SM136.5	Current counting direction status bit: • 0 = Counting down • 1 = Counting up
SM36.6	SM46.6	SM56.6	SM136.6	Current value equals preset value status bit: • 0 = Not equal • 1 = Equal
SM36.7	SM46.7	SM56.7	SM136.7	Current value greater than preset value status bit: • 0 = Less than or equal • 1 = Greater than

See Also

[High-speed counter instructions \(Page 210\)](#)

[Example initialization sequences for the high-speed counters \(Page 228\)](#)

7.6.5**Example initialization sequences for high-speed counters**

HSC0 is used as the counter in the following descriptions of the initialization and operation sequences.

- HSC0 and HSC2 support counting modes (0, 1), (3, 4), (6, 7), and (9, 10).
- HSC1 and HSC3 only support counting mode 0.

The initialization descriptions assume that the CPU has just been placed in RUN mode, and for that reason, the first scan memory bit is true. If this is not the case, remember that the HDEF instruction can be executed only one time for each high-speed counter, after entering RUN mode. Executing HDEF for a high-speed counter a second time generates a run-time error and does not change the counter setup from the way it was set up on the first execution of HDEF for that counter.

Note

Although the following sequences show how to change direction, current value, and preset value individually, you can change all or any combination of them in the same sequence by setting the value of SMB37 appropriately and then executing the HSC instruction.

Initialization of modes 0 and 1

The following steps describe how to initialize HSC0 for single-phase up/down counter with internal direction (modes 0 and 1).

1. Use the first scan memory bit to call a subroutine in which the initialization operation is performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

For example: SMB37 = 16#F8

Produces the following results:

- Enables the counter
 - Writes a new current value
 - Writes a new preset value
 - Sets the direction to count up
 - Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE input set to one of the following:
 - Mode 0 for no external reset
 - Mode 1 for external reset
 4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
 5. Load SMD42 (double-word-sized value) with the desired preset value.
 6. In order to capture the current value equal to preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section that discusses the Interrupt instructions for complete details on interrupt processing.
 7. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
 8. Execute the global interrupt enable instruction (ENI) to enable interrupts.
 9. Execute the HSC instruction to cause the CPU to program HSC0.
 10. Exit the subroutine.

Initialization of modes 3 and 4

The following steps describe how to initialize HSC0 for single-phase up/down counter with external direction control (modes 3 and 4):

1. Use the first scan memory bit to call a subroutine in which the initialization operation is performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

For example: SMB37 = 16#F8

Produces the following results:

- Enables the counter
 - Writes a new current value
 - Writes a new preset value
 - Sets the initial direction of the HSC to count up
 - Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE input set to one of the following:
 - Mode 3 for no external reset
 - Mode 4 for external reset
 4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
 5. Load SMD42 (double-word-sized value) with the desired preset value.
 6. In order to capture the current-value-equal-to-preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section that discusses the Interrupt instructions for complete details on interrupt processing.
 7. In order to capture direction changes, program an interrupt by attaching the direction changed interrupt event (event 27) to an interrupt routine.
 8. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
 9. Execute the global interrupt enable instruction (ENI) to enable interrupts.
 10. Execute the HSC instruction to cause the CPU to program HSC0.
 11. Exit the subroutine.

Initialization of modes 6 and 7

The following steps describe how to initialize HSC0 for two-phase up/down counter with up/down clocks (modes 6 and 7):

1. Use the first scan memory bit to call a subroutine in which the initialization operations are performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

For example: SMB37 = 16#F8

Produces the following results:

- Enables the counter
 - Writes a new current value
 - Writes a new preset value
 - Sets the initial direction of the HSC to count up
 - Sets the reset input to be active high
3. Execute the HDEF instruction with the HSC input set to 0 and the MODE set to one of the following:
 - Mode 6 for no external reset
 - Mode 7 for external reset
 4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
 5. Load SMD42 (double-word-sized value) with the desired preset value.
 6. In order to capture the current-value-equal-to-preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section on interrupts.
 7. In order to capture direction changes, program an interrupt by attaching the direction changed interrupt event (event 27) to an interrupt routine.
 8. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
 9. Execute the global interrupt enable instruction (ENI) to enable interrupts.
 10. Execute the HSC instruction to cause the CPU to program HSC0.
 11. Exit the subroutine.

Initialization of modes 9 and 10

The following steps describe how to initialize HSC0 as an AB quadrature phase counter (for modes 9 and 10):

1. Use the first scan memory bit to call a subroutine in which the initialization operations are performed. Since you use a subroutine call, subsequent scans do not make the call to the subroutine, which reduces scan time execution and provides a more structured program.
2. In the initialization subroutine, load SMB37 according to the desired control operation.

Example (1x counting mode): SMB37 = 16#FC

Produces the following results:

- Enables the counter
- Writes a new current value
- Writes a new preset value
- Sets the initial direction of the HSC to count up
- Sets the reset input to be active high

Example (4x counting mode): SMB37 = 16#F8

Produces the following results:

- Enables the counter
- Writes a new current value
- Writes a new preset value
- Sets the initial direction of the HSC to count up
- Sets the reset input to be active high

3. Execute the HDEF instruction with the HSC input set to 0 and the MODE input set to one of the following:
 - Mode 9 for no external reset
 - Mode 10 for external reset
4. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).
5. Load SMD42 (double-word-sized value) with the desired preset value.
6. In order to capture the current-value-equal-to-preset event, program an interrupt by attaching the CV = PV interrupt event (event 12) to an interrupt routine. See the section on enabling interrupts (ENI) for complete details on interrupt processing.
7. In order to capture direction changes, program an interrupt by attaching the direction changed interrupt event (event 27) to an interrupt routine.
8. In order to capture an external reset event, program an interrupt by attaching the external reset interrupt event (event 28) to an interrupt routine.
9. Execute the global interrupt enable instruction (ENI) to enable interrupts.
10. Execute the HSC instruction to cause the CPU to program HSC0.
11. Exit the subroutine.

Change direction in modes 0 and 1

The following steps describe how to configure HSC0 for change direction for single-phase counter with internal direction (modes 0 and 1):

1. Load SMB37 to write the desired direction:

SMB37 = 16#90

- Enables the counter
 - Sets the direction of the HSC to count down
- SMB37 = 16#98
- Enables the counter
 - Sets the direction of the HSC to count up

2. Execute the HSC instruction to cause the CPU to program HSC0.

Loading a new current value (any mode)

The following steps describe how to change the counter current value of HSC0 (any mode):

1. Load SMB37 to write the desired current value:

SMB37 = 16#C0

- Enables the counter
- Writes the new current value

2. Load SMD38 (double-word-sized value) with the desired current value (load with 0 to clear it).

3. Execute the HSC instruction to cause the CPU to program HSC0.

Loading a new preset value (any mode)

The following steps describe how to change the preset value of HSC0 (any mode):

1. Load SMB37 to write the desired preset value:

SMB37 = 16#A0

- Enables the counter
- Writes the new preset value

2. Load SMD42 (double-word-sized value) with the desired preset value.

3. Execute the HSC instruction to cause the CPU to program HSC0.

Disabling a high-speed counter (any mode)

The following steps describe how to disable the HSC0 high-speed counter (any mode):

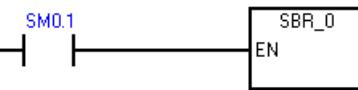
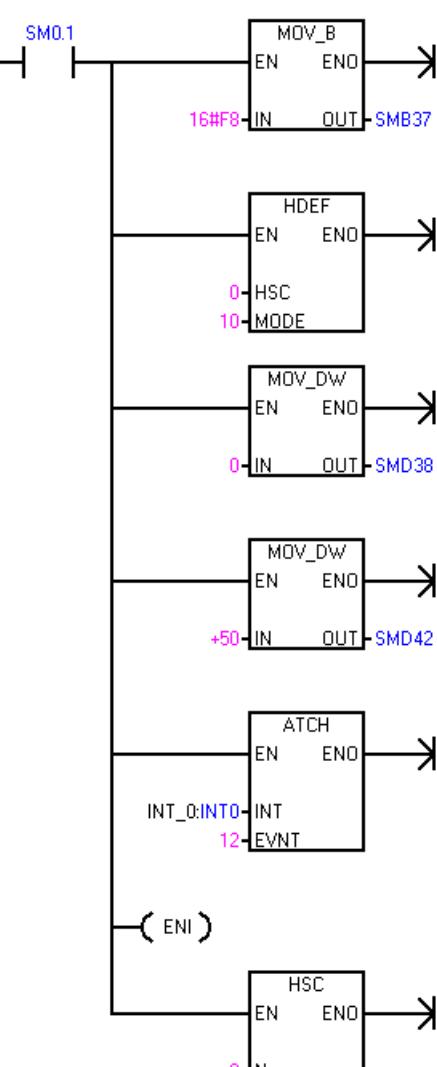
1. Load SMB37 to disable the counter:

SMB37 = 16#00

– Disables the counter

2. Execute the HSC instruction to disable the counter.

Example: high-speed counter instruction

LAD	STL
MAIN 	Network 1 LD SM0.1 CALL SBR_0
SBR0 	On the first scan, configure HSC0: 1. Enable the counter <ul style="list-style-type: none"> - Write a new current value. - Write a new preset value. - Set the initial direction to count up. - Select the reset input to be active high. - Select 4x mode. 2. Configure HSC0 for AB quadrature phase with reset input. 3. Clear the current value of HSC0. 4. Set the HSC0 preset value to 50. 5. Attach event 12 to interrupt routine INT_0. The interrupt is executed when HSC0 current value = preset value. 6. Global interrupt enable 7. Configure HSC0. Network 1 LD SM0.1 MOV B 16#F8, SMB37 HDEF 0, 10 MOVD +0, SMD38 MOVD +50, SMD42 ATCH INT_0, 12 ENI HSC 0

LAD	STL
	<p>Program HSC0:</p> <ol style="list-style-type: none"> 1. Clear the current value of HSC0. 2. Select to write only a new current and leave HSC0 enabled. 3. Configure HSC0. <pre> Network 1 LD SM0.0 MOVD +0, SMD38 MOVB 16#C0, SMB37 HSC 0 </pre>

See also

[High-speed counter instructions \(Page 210\)](#)

[High-speed counter programming \(Page 216\)](#)

[Interrupt instructions \(Page 263\)](#)

7.7 Pulse output

7.7.1 Pulse output instruction (PLS)

The Pulse Output (PLS) instruction controls the Pulse Width Modulation (PWM) functions available on the high-speed outputs (Q0.0, Q0.1, and Q0.3).

An optional wizard is available when using the PLS instruction to create PWM instructions.

LAD / FBD	STL	Description
	PLS N	<p>You can use the PLS instruction to create PWM operations. PWM provides three channels that allow user control of a fixed cycle time output with a variable duty cycle.</p>

Error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0005H Simultaneous HSC/PLS • 0017H Attempt to assign resource for PWM that is already assigned to motion control • 001BH Attempt to change timebase on enabled PWM • 0090H N is not 0, 1, or 2. 	None

Input / output	Data type	Operand
N (channel)	WORD	Constant: 0 (= Q0.0), 1 (= Q0.1), or 2 (= Q0.3)

The CPU has three PWM generators that create a pulse width modulated waveform. One generator is assigned to digital output point Q0.0, one generator is assigned to digital output point Q0.1, and one generator is assigned to digital output point Q0.3. A designated special memory (SM) location stores the following data for each generator: a control byte (8-bit value), a cycle time (an unsigned 16-bit value), and a pulse width value (an unsigned 16-bit value).

The PWM generators and the process-image register share the use of Q0.0, Q0.1, and Q0.3. When a PWM function is active on Q0.0, Q0.1, or Q0.3, the PWM generator has control of the output, and normal use of the output point is inhibited. The output waveform is not affected by the state of the process-image register, the forced value of the point, or the execution of immediate output instructions. When the PWM generator is inactive, control of the output reverts to the process-image register. The process-image register determines the initial and final state of the output waveform, causing the waveform to start and end at a high or low level.

Note

PWM through the PLS instruction is not possible if the selected output point is already configured for use with motion control through use of the Motion wizard.

Before enabling PWM operation, set the value of the process-image register for Q0.0, Q0.1, and Q0.3 to 0.

Default values for all control bits, cycle time, and pulse width values are 0.

The PWM outputs must have a minimum load of at least 10% of rated load to provide crisp transitions from off to on, and from on to off.

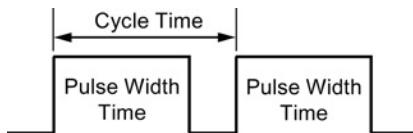
Note

The Pulse Output (PLS) instruction can only be used with the following S7-200 SMART CPUs:

- SR20 / ST20 (Two channels, Q0.0 and Q0.1)
 - SR30 / ST30, SR40 / ST40, and SR60 / ST60 (Three channels, Q0.0, Q0.1, and Q0.3)
-

7.7.2 Pulse width modulation (PWM)

PWM provides three channels that allow a fixed cycle time output with a variable duty cycle. (See the following figure.) You can specify the cycle time and the pulse width in either microsecond or millisecond increments:



- Cycle time: 10 µs to 65,535 µs or 2 ms to 65,535 ms
- Pulse width time: 0 µs to 65,535 µs or 0 ms to 65,535 ms

As shown in the following table, setting the pulse width equal to the cycle time (which makes the duty cycle 100 percent) turns the output on continuously. Setting the pulse width to 0 (which makes the duty cycle 0 percent) turns the output off.

Pulse width time and cycle time and reactions in the PWM function

Pulse width time / cycle time	Reaction
Pulse width time >= Cycle time value	The duty cycle is 100%: the output is turned on continuously.
Pulse width time = 0	The duty cycle is 0%: the output is turned off continuously.
Cycle time < 2 time units	The cycle time defaults to two time units.

Changing the characteristics of a PWM waveform

You can only use synchronous updates to change the characteristics of a PWM waveform. With a synchronous update, the change in the waveform characteristics occurs on a cycle boundary, providing a smooth transition.

7.7.3 Using SM locations to configure and control the PWM operation

The PLS instruction reads the data stored in the specified SM memory locations and programs the PWM generator accordingly. SMB67 controls PWM 0, SMB77 controls PWM 1, and SMB567 controls PWM 2. The first table below describes the registers used to control the PWM operation. You can use the second table below as a quick reference to determine the value to place in the PWM control register to invoke the desired operation.

You can change the characteristics of a PWM waveform by modifying the locations in the SM area (including the control byte) and then executing the PLS instruction. You can disable the generation of a PWM waveform at any time by writing 0 to the PWM enable bit of the control byte (SM67.7, SM77.7, or SM567.7) and then executing the PLS instruction.

Note

- When you load a pulse width (SMW70 or SMW80), or cycle time (SMW68 or SMW78), also set the appropriate update bits in the control register before you execute the PLS instruction.
- After aborting a PWM operation, one cycle time should elapse before re-enabling the PWM channel for operation. If the PWM channel is re-enabled before this time expires, pulse distortion can occur in the initial pulses caused by completion of the original PWM command.
- If you attempt to change the time base of a PWM output while the PWM is executing, the request will be ignored and a non-fatal error (0x001B - ILLEGAL PWM TIMEBASE CHG) is created.

SM locations for the PWM control registers

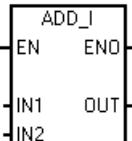
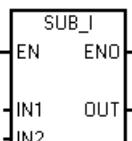
PWM control addresses			PWM control functions
Q0.0	Q0.1	Q0.3	PWM output channel identifier
SM67.0	SM77.0	SM567.0	PWM update the cycle time: <ul style="list-style-type: none"> • 0 = no update • 1 = update cycle time
SM67.1	SM77.1	SM567.1	PWM update the pulse width time: <ul style="list-style-type: none"> • 0 = no update • 1 = update pulse width
SM67.2	SM77.2	SM567.2	Reserved
SM67.3	SM77.3	SM567.3	PWM time base: <ul style="list-style-type: none"> • 0 = 1µs/tick • 1 = 1 ms/tick
SM67.4	SM77.4	SM567.4	Reserved
SM67.5	SM77.5	SM567.5	Reserved
SM67.6	SM77.6	SM567.6	Reserved
SM67.7	SM77.7	SM567.7	PWM enable: <ul style="list-style-type: none"> • 0 = disable • 1 = enable
Q0.0 Q0.1 Q0.3			Other PWM registers
SMW68	SMW78	SMW568	PWM cycle time value <ul style="list-style-type: none"> • range: 2 to 65,535
SMW70	SMW80	SMW570	PWM pulse width value <ul style="list-style-type: none"> • range: 0 to 65,535

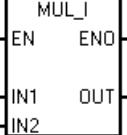
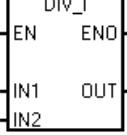
PWM control byte format

Control register (Hex value)	Enable	Time base	Pulse width	Cycle time
16#80	Yes	1 µs/cycle		
16#81	Yes	1 µs/cycle		Update
16#82	Yes	1 µs/cycle	Update	
16#83	Yes	1 µs/cycle	Update	Update
16#88	Yes	1 ms/cycle		
16#89	Yes	1 ms/cycle		Update
16#8A	Yes	1 ms/cycle	Update	
16#8B	Yes	1 ms/cycle	Update	Update

7.8 Math

7.8.1 Add, subtract, multiply, and divide

LAD / FBD	STL	Description
 ADD_I ADD_DI ADD_R	$+I \text{ IN1, OUT}$ $+D \text{ IN1, OUT}$ $+R \text{ IN1, OUT}$	<p>The Add Integer instruction adds two 16-bit integers to produce a 16-bit result. The Add Double Integer instruction adds two 32-bit integers to produce a 32-bit result. The Add Real (+R) instruction adds two 32-bit real numbers to produce a 32-bit real number result.</p> <ul style="list-style-type: none"> • LAD and FBD: $\text{IN1} + \text{IN2} = \text{OUT}$ • STL: $\text{IN1} + \text{OUT} = \text{OUT}$
 SUB_I SUB_DI SUB_R	$-I \text{ IN1, OUT}$ $-D \text{ IN1, OUT}$ $-R \text{ IN1, OUT}$	<p>The Subtract Integer instruction subtracts two 16-bit integers to produce a 16-bit result. The Subtract Double Integer (-D) instruction subtracts two 32-bit integers to produce a 32-bit result. The Subtract Real (-R) instruction subtracts two 32-bit real numbers to produce a 32-bit real number result.</p> <ul style="list-style-type: none"> • LAD and FBD: $\text{IN1} - \text{IN2} = \text{OUT}$ • STL: $\text{OUT} - \text{IN1} = \text{OUT}$

LAD / FBD	STL	Description
 MUL_I MUL_DI MUL_R	*I IN1, OUT *D IN1, OUT *R IN1, OUT	<p>The Multiply Integer instruction multiplies two 16-bit integers to produce a 16-bit result. The Multiply Double Integer instruction multiplies two 32-bit integers to produce a 32-bit result. The Multiply Real instruction multiplies two 32-bit real numbers to produce a 32-bit real number result.</p> <ul style="list-style-type: none"> • LAD and FBD: $IN1 * IN2 = OUT$ • STL: $IN1 * OUT = OUT$
 DIV_I DIV_DI DIV_R	/I IN1, OUT /D IN1, OUT /R IN1, OUT	<p>The Divide Integer instruction divides two 16-bit integers to produce a 16-bit result. (No remainder is kept.) Divide Double Integer instruction divides two 32-bit integers to produce a 32-bit result. (No remainder is kept.) The Divide Real (/R) instruction divides two 32-bit real numbers to produce a 32-bit real number result.</p> <ul style="list-style-type: none"> • LAD and FBD: $IN1 / IN2 = OUT$ • STL: $OUT / IN1 = OUT$

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow • SM1.3 Divide by zero 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result • SM1.3 Divide by zero

SM1.1 indicates overflow errors and illegal values. If SM1.1 is set, then the status of SM1.0 and SM1.2 is not valid and the original input operands are not altered. If SM1.1 and SM1.3 are not set, then the math operation has completed with a valid result and SM1.0 and SM1.2 contain valid status. If SM1.3 is set during a divide operation, then the other math status bits are left unchanged.

Input / output	Data Type	Operand
IN1, IN2	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *AC, *LD, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	INT	IW, QW, VW, MW, SMW, SW, LW, T, C, AC, *VD, *AC, *LD
	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

¹ Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

Example: Integer math instructions

LAD	STL
<pre> Network LD I0.0 +I AC1, AC0 *I AC1, VW100 /I VW10, VW200 </pre>	

Integer operations from the LAD example

	IN1		IN2		OUT
Add data	40	+	60	=	100
Data address	AC1		AC0		AC0
Multiply data	40	*	20	=	800
Data address	AC1		VW100		VW100
Divide data	4000	/	40	=	100
Data address	W200		VW10		VW200

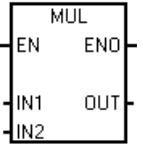
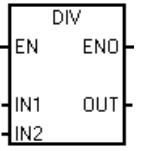
Example: Real math instructions

LAD	STL
	Network 1 LD I0.0 +R AC1, AC0 *R AC1, VD100 /R VD10, VD200

Real number operations from the LAD example

	IN1		IN2		OUT
Add data	4000.0	+	6000.0	=	10000.0
Data address	AC1		AC0		AC0
Multiply data	400.0	*	200.0	=	80000.0
Data address	AC1		VD100		VD100
Divide data	4000.0	/	41.0	=	97.5609
Data address	VD200		VD10		VD200

7.8.2 Multiply integer to double integer and divide integer with remainder

LAD / FBD	STL	Description
	MUL IN1, OUT	The multiply integer to double Integer instruction multiplies two 16-bit integers and produces a 32-bit product. In STL, the least-significant word (16 bits) of the 32-bit OUT is used as one of the factors. <ul style="list-style-type: none"> • LAD and FBD: $IN1 * IN2 = OUT$ • STL: $IN1 * OUT = OUT$
	DIV IN1, OUT	The divide integer with remainder instruction divides two 16-bit integers and produces a 32-bit result consisting of a 16-bit remainder (the most-significant word) and a 16-bit quotient (the least-significant word). In STL, the least-significant word (16 bits) of the 32-bit OUT is used as the dividend. <ul style="list-style-type: none"> • LAD and FBD: $IN1 / IN2 = OUT$ • STL: $OUT / IN1 = OUT$

Non-fatal errors with ENO=0	SM bits affected ¹
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow • SM1.3 Divide by zero 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result • SM1.3 Divide by zero

¹ For both of these instructions, SM bits indicate errors and illegal values. If SM1.3 (divide by zero) is set during a divide operation, then the other math status bits are left unchanged. Otherwise, all supported math status bits contain valid status upon completion of the math operation.

Input / output	Data type	Operand
IN1, IN2	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
OUT	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: MUL and DIV instructions

LAD	STL
<pre> LD I0.0 MUL AC1, VD100 DIV VW10, VD200 </pre>	

¹ VD100 contains: VW100 and VW102, and VD200 contains: VW200 and VW202.

Real number operations from the LAD example

	IN1	IN2	=	OUT
MUL data	400	*	200	80000
Data address	AC1		VW102	VD100
DIV data	4000	/	41	remainder quotient
Data address	VW202		VW10	VW200 VW202
				VD200

7.8.3 Trigonometry, natural logarithm/exponential, and square root

Sine (SIN), Cosine (COS), and Tangent (TAN) instructions

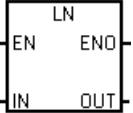
LAD / FBD	STL	Description
	SIN IN, OUT	The sine (SIN), cosine (COS), and tangent (TAN) instructions evaluate the trigonometric function of the angle value IN and place the result in OUT. The input angle value is measured in radians. <ul style="list-style-type: none">• SIN (IN) = OUT• COS (IN) = OUT• TAN (IN) = OUT
	COS IN, OUT	To convert an angle from degrees to radians: Use the MUL_R (*R) instruction to multiply the angle in degrees by 1.745329E-2 (approximately by $\pi/180$).
	TAN IN, OUT	For the numeric functions instructions, SM1.1 is used to indicate overflow errors and illegal values. If SM1.1 is set, then the status of SM1.0 and SM1.2 is not valid and the original input operands are not altered. If SM1.1 is not set, then the math operation has completed with a valid result and SM1.0 and SM1.2 contain valid status.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / outputs	Data type	Operand
IN	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

¹ Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

Natural logarithm (LN) and natural exponential (EXP) instructions

LAD / FBD	STL	Description
	LN IN, OUT	<p>The Natural Logarithm instruction (LN) performs the natural logarithm of the value in IN and places the result in OUT.</p> <p>The Natural Exponential instruction (EXP) performs the exponential operation of e raised to the power of the value in IN and places the result in OUT.</p> <ul style="list-style-type: none"> • LN (IN) = OUT • EXP (IN)= OUT
	EXP IN, OUT	<p>To obtain the base 10 logarithm from the natural logarithm: Divide the natural logarithm by 2.302585 (approximately the natural logarithm of 10).</p> <p>To raise any real number to the power of another real number, including fractional exponents: Combine the Natural Exponential instruction with the Natural Logarithm instruction. For example, to raise X to the Y power, use EXP (Y * LN (X)).</p>

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / outputs	Data type	Operand
IN	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

¹ Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

Square root (SQRT) instruction

LAD / FBD	STL	Description
	SQRT IN, OUT	<p>The Square Root instruction (SQRT) takes the square root of a real number (IN) and produces a real number result OUT.</p> <ul style="list-style-type: none"> • SQRT (IN)= OUT <p>To obtain other roots:</p> <ul style="list-style-type: none"> • 5 cubed = $5^3 = \text{EXP}(3*\text{LN}(5)) = 125$ • The cube root of 125 = $125^{(1/3)} = \text{EXP}((1/3)*\text{LN}(125))= 5$ • The square root of 5 cubed = $5^{(3/2)} = \text{EXP}(3/2*\text{LN}(5)) = 11.18034$

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / outputs	Data type	Operand
IN	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	REAL ¹	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

¹ Real (or floating-point) numbers are represented in the format described in the ANSI/IEEE 754-1985 standard (single-precision). Refer to that standard for more information.

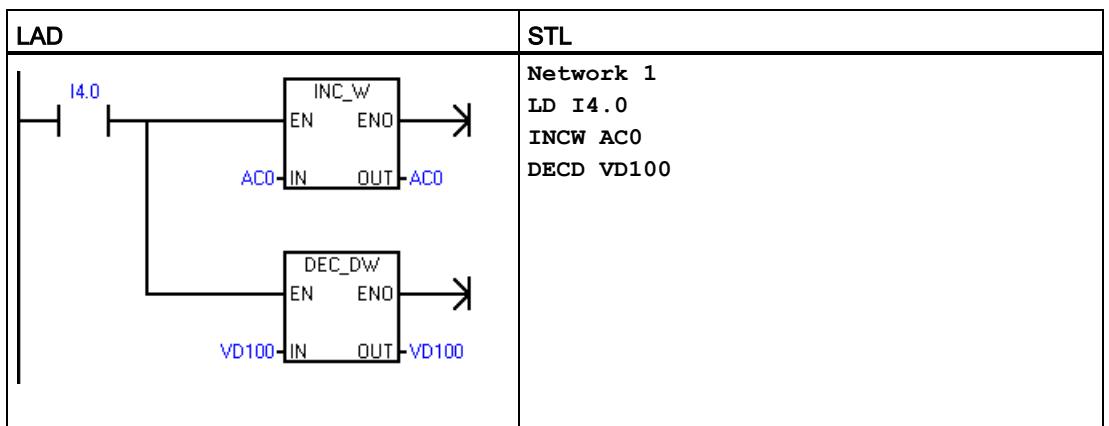
7.8.4 Increment and decrement

LAD / FBD	STL	Description
 INC_W INC_DW	INC_B OUT INCW OUT INCD OUT	<p>The increment instructions add 1 to the input value IN and place the result into the location at OUT.</p> <ul style="list-style-type: none"> • LAD and FBD: $IN + 1 = OUT$ • STL: $OUT + 1 = OUT$ <p>Increment byte (INC_B) operations are unsigned. Increment word (INC_W) operations are signed. Increment double word (INC_DW) operations are signed.</p>
 DEC_W DEC_DW	DECB OUT DECW OUT DEC_D OUT	<p>The decrement instructions subtract 1 from the input value IN and place the result into the location at OUT.</p> <ul style="list-style-type: none"> • LAD and FBD: $IN - 1 = OUT$ • STL: $OUT - 1 = OUT$ <p>Decrement byte (DEC_B) operations are unsigned. Decrement Word (DEC_W) operations are signed. Decrement double Word (DEC_D) operations are signed.</p>

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • SM1.1 Overflow 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow, illegal value generated during the operation, or illegal input • SM1.2 Negative result

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: Increment and decrement



Increment/decrement operations from the LAD example

	IN		OUT
Increment word	125	+	1 = 126
Data address	AC0		AC0
Decrement double word	128000	-	1 = 127999
Data address	VD100		VD100

7.9 PID

LAD / FBD	STL	Description
	PID TBL, LOOP	The PID loop instruction (PID) executes a PID loop calculation on the referenced LOOP based upon the input and configuration information in Table (TBL).

Non-fatal errors with ENO = 0	SM bits affected
• 0013H Illegal PID loop table	• SM1.1 Overflow

Input / output	Data type	Operand
TBL	BYTE	VB
LOOP	BYTE	Constant (0 to 7)

The PID loop instruction (Proportional, Integral, Derivative Loop) is provided to perform the PID calculation. The top of the logic stack (TOS) must be ON (power flow) to enable the PID calculation. The instruction has two operands: a TABLE address which is the starting address of the loop table and a LOOP number which is a constant from 0 to 7.

Eight PID instructions can be used in a program. If two or more PID instructions are used with the same loop number (even if they have different table addresses), the PID calculations will interfere with one another and the output will be unpredictable.

The loop table stores nine parameters used for controlling and monitoring the loop operation and includes the current and previous value of the process variable, the setpoint, output, gain, sample time, integral time (reset), derivative time (rate), and the integral sum (bias).

To perform the PID calculation at the desired sample rate, the PID instruction must be executed either from within a timed interrupt routine or from within the main program at a rate controlled by a timer. The sample time must be supplied as an input to the PID instruction via the loop table.

Auto-Tune capability has been incorporated into the PID instruction. Refer to "PID loops and tuning" (Page 423) for a detailed description of auto-tuning. The "PID Tune control panel" (Page 431) only works with PID loops created by the PID wizard.

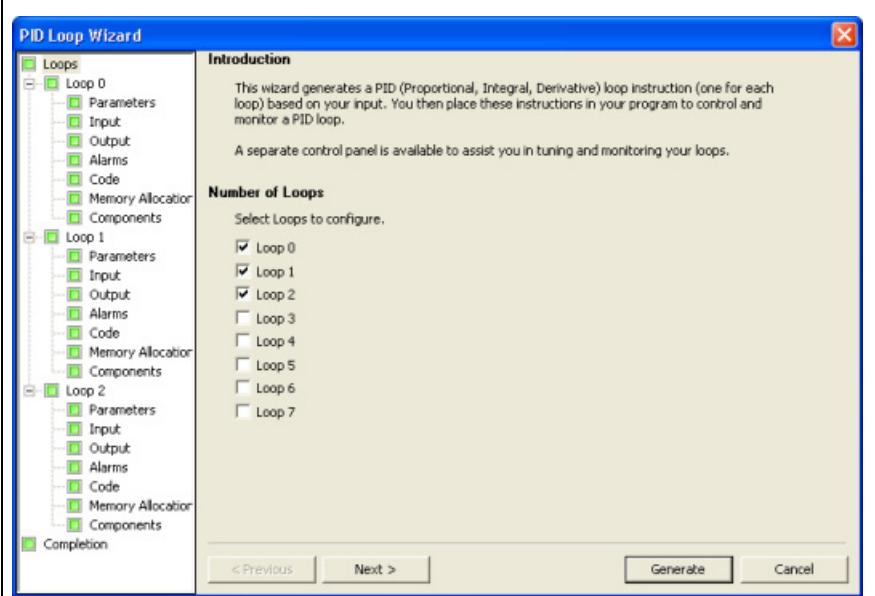
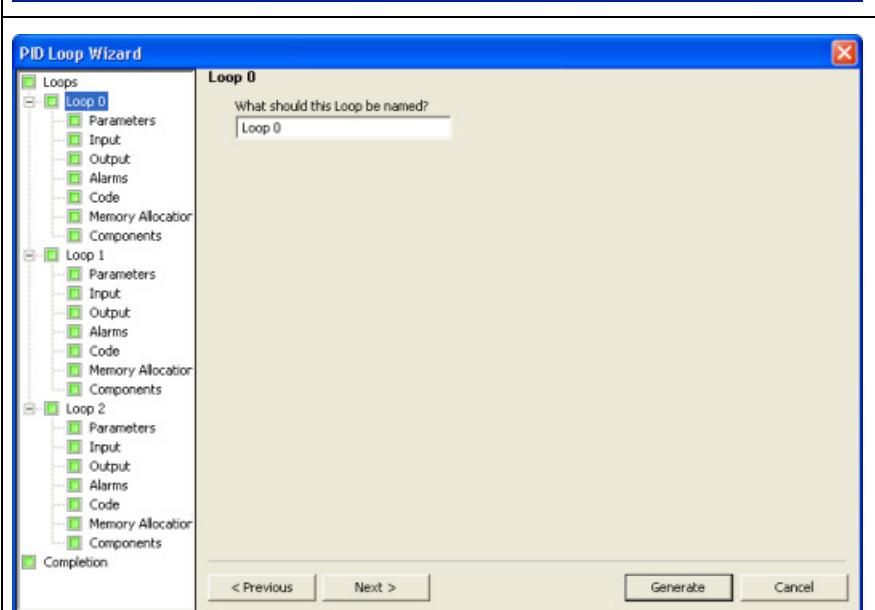
STEP 7-Micro/WIN SMART offers the PID wizard to guide you in defining a PID algorithm for a closed-loop control process. Select the "Instruction wizard" command from the "Tools" menu and then select "PID" from the "Instruction wizard" window.

Note

The setpoint of the low range and the setpoint of the high range should correspond to the process variable low range and high range.

7.9.1 Using the PID wizard

Use the PID wizard to configure your PID loop

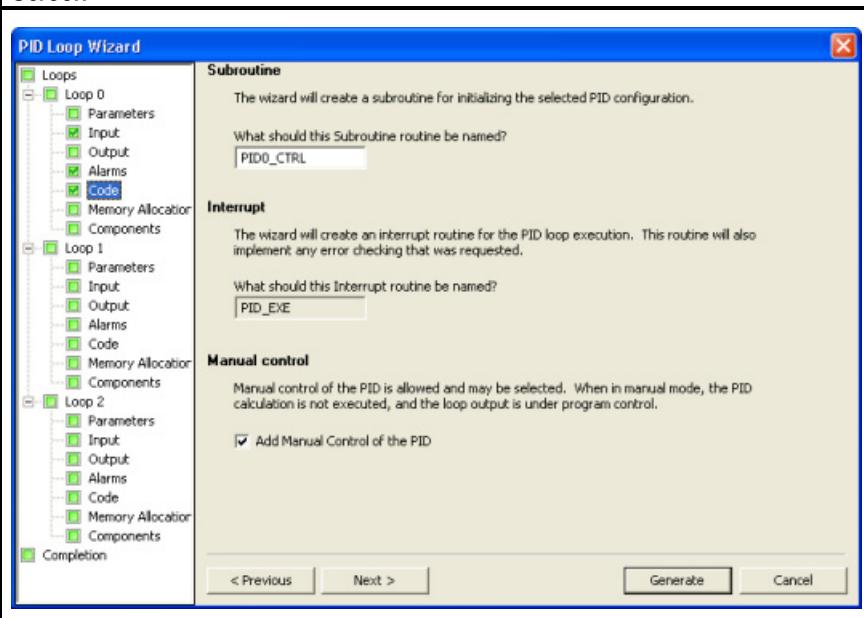
Screen	Description
	<p>In this dialog, you select which loops to configure. You can configure a maximum of eight loops.</p> <p>When you select a loop on this dialog, the tree view on the left side of the PID wizard updates with all nodes necessary for configuring that loop.</p>
	<p>You can configure a custom name for your loop. The default name of this screen is "Loop x", where "x" is equal to the loop number.</p>

Program instructions

7.9 PID

Screen	Description						
<p>PID Loop Wizard</p> <p>Loops</p> <ul style="list-style-type: none"> Loop 0 <ul style="list-style-type: none"> Parameters Input Output Alarms Code Memory Allocation Components Loop 1 <ul style="list-style-type: none"> Parameters Input Output Alarms Code Memory Allocation Components Loop 2 <ul style="list-style-type: none"> Parameters Input Output Alarms Code Memory Allocation Components <p>Completion</p> <p>< Previous Next > Generate Cancel</p>	<p>Set the following loop parameters:</p> <ul style="list-style-type: none"> Gain (Default value = 1.00) Sample Time (Default value = 1.00) Integral Time (Default value = 10.00) Derivative Time (Default value = 0.00) 						
<p>PID Loop Wizard</p> <p>Specify how the loop Process Variable (PV) should be scaled. The Loop PV is a parameter you specify for the subroutine generated by the wizard.</p> <p>Type</p> <p>Process Variable Scaling</p> <p>Unipolar 20% offset</p> <p>Scaling</p> <table border="1"> <tr> <td>Process Variable</td> <td>Loop Setpoint</td> </tr> <tr> <td>Low: 5530</td> <td>0.0</td> </tr> <tr> <td>High: 27648</td> <td>100.0</td> </tr> </table> <p>< Previous Next > Generate Cancel</p>	Process Variable	Loop Setpoint	Low: 5530	0.0	High: 27648	100.0	<p>You assign how the loop Process Variable (PV) is to be scaled. You can choose from the following options:</p> <ul style="list-style-type: none"> Unipolar (default: 0 to 27648; can edit) Bipolar (default: -27648 to 27648; can edit) Unipolar 20% offset (range: 5530 to 27648; set and unchangeable) Temperature x 10 °C Temperature x 10 °F <p>In the Scaling parameter, you assign how the loop setpoint (SP) is to be scaled. Default is a real number between 0.0 and 100.0.</p>
Process Variable	Loop Setpoint						
Low: 5530	0.0						
High: 27648	100.0						

Screen	Description
<p>The PID Loop Wizard - Loop Output screen. The left sidebar shows a tree structure with Loops, Parameters, Input, Output, Alarms, Code, Memory Allocation, Components, and Completion. Under Loop 0, Output is selected. The main panel displays the configuration for Loop 0 Output. The Type is set to Analog, and the Scaling is set to Unipolar. The Range is defined by Low: 0 and High: 27648. Navigation buttons < Previous, Next >, Generate, and Cancel are at the bottom.</p>	<p>You enter the loop output options:</p> <ul style="list-style-type: none"> How the loop output is to be scaled: <ul style="list-style-type: none"> Analog Digital Analog scaling parameter: <ul style="list-style-type: none"> Unipolar (default: 0 to 27648; can edit) Bipolar (default: -27648 to 24678; can edit) Unipolar 20% offset (range: 5530 to 27648; is set and unchangeable) Analog range parameter: Assign the loop output range. The possible range is -27648 to +27648, depending on your scaling selection.
<p>The PID Loop Wizard - Alarms screen. The left sidebar shows a tree structure with Loops, Parameters, Input, Output, Alarms, Code, Memory Allocation, Components, and Completion. Under Loop 0, Alarms is selected. The main panel displays the configuration for Loop 0 Alarms. It includes sections for Low (Enable low alarm (PV) at 0.10), High (Enable high alarm (PV) at 0.90), and Module (Enable analog input error at EM 0). Navigation buttons < Previous, Next >, Generate, and Cancel are at the bottom.</p>	<p>You can assign what conditions to recognize with alarm inputs. Use the checkboxes to enable the alarms as required:</p> <ul style="list-style-type: none"> Low Alarm (PV): Set normalized low alarm limit from 0.0 to high alarm limit; default is 0.10. High Alarm (PV): Set normalized high alarm limit from low alarm limit to 1.00; default is 0.90. Analog Input Error: Assign where the input module is attached to the PLC.

Screen	Description
 <p>PID Loop Wizard</p> <p>Subroutine The wizard will create a subroutine for initializing the selected PID configuration.</p> <p>What should this Subroutine routine be named? <input type="text" value="PID0_CTRL"/></p> <p>Interrupt The wizard will create an interrupt routine for the PID loop execution. This routine will also implement any error checking that was requested.</p> <p>What should this Interrupt routine be named? <input type="text" value="PID_EXE"/></p> <p>Manual control Manual control of the PID is allowed and may be selected. When in manual mode, the PID calculation is not executed, and the loop output is under program control.</p> <p><input checked="" type="checkbox"/> Add Manual Control of the PID</p> <p>< Previous Next > Generate Cancel</p>	<p>You can make the following code selections:</p> <ul style="list-style-type: none"> • Subroutine: The PID wizard creates a subroutine for initializing the selected PID configuration. • Interrupt: The PID wizard creates an interrupt routine for the PID loop execution. <p>Note: The wizard assigns a default name for the subroutine and the interrupt routine; you can edit the default names.</p> <ul style="list-style-type: none"> • Manual control: Use the "Add Manual Control of the PID" checkbox to allow manual control of your PID loops.

Screen	Description										
<p>The screenshot shows the 'Memory Allocation' step of the PID Loop Wizard. On the left is a tree view of loop components: Loops, Loop 0 (Parameters, Input, Output, Alarms, Code, Memory Allocation, Components), Loop 1, Loop 2, and Completion. The 'Memory Allocation' node under Loop 0 is selected. The main area is titled 'Memory Allocation' with the instruction: 'Please specify a starting address where the configuration will be placed in the Data Block. The wizard can also suggest an address that represents an unused block of V-memory of the correct size.' A 'Suggest' button is present. Below it is an input field showing 'VB 120 - VB239 (120 bytes)'. At the bottom are navigation buttons: '< Previous', 'Next >', 'Generate', and 'Cancel'.</p>	<p>You can assign the starting address of the V memory byte where the configuration is placed in the data block. The wizard can suggest an address that represents an unused block of V memory of the correct size.</p>										
<p>The screenshot shows the 'Components' step of the PID Loop Wizard. The left sidebar shows the same tree view as the previous screen. The main area is titled 'Components' with the instruction: 'The Subroutines and Interrupt Routines listed below will become part of the project. To enable this configuration within the program, place a call in the MAIN program block to the initialization subroutine "PID0_CTRL".' Below this is a table titled 'The requested configuration consists of the following project components:'.</p> <table border="1" data-bbox="377 1102 949 1237"> <thead> <tr> <th data-bbox="377 1102 600 1136">Component</th> <th data-bbox="600 1102 949 1136">Description</th> </tr> </thead> <tbody> <tr> <td data-bbox="377 1136 600 1170">0 PID0_CTRL</td> <td data-bbox="600 1136 949 1170">Subroutine used to initialize PID</td> </tr> <tr> <td data-bbox="377 1170 600 1203">1 PID_EXE</td> <td data-bbox="600 1170 949 1203">Interrupt used to cyclically execute PID function</td> </tr> <tr> <td data-bbox="377 1203 600 1237">2 PID0_DATA</td> <td data-bbox="600 1203 949 1237">Data page with configuration placed at (VB120 - VB239)</td> </tr> <tr> <td data-bbox="377 1237 600 1271">3 PID0_SYM</td> <td data-bbox="600 1237 949 1271">Symbol table created for this configuration</td> </tr> </tbody> </table> <p>At the bottom are navigation buttons: '< Previous', 'Next >', 'Generate', and 'Cancel'.</p>	Component	Description	0 PID0_CTRL	Subroutine used to initialize PID	1 PID_EXE	Interrupt used to cyclically execute PID function	2 PID0_DATA	Data page with configuration placed at (VB120 - VB239)	3 PID0_SYM	Symbol table created for this configuration	<p>This screen shows a list of the subroutines and interrupt routines generated by the PID wizard and gives a brief description of how these should be integrated into your program.</p>
Component	Description										
0 PID0_CTRL	Subroutine used to initialize PID										
1 PID_EXE	Interrupt used to cyclically execute PID function										
2 PID0_DATA	Data page with configuration placed at (VB120 - VB239)										
3 PID0_SYM	Symbol table created for this configuration										

STEP 7-Micro/WIN SMART includes a PID tune control panel (Page 431) that allows you to graphically monitor the behavior of your PID loops. In addition, the control panel allows you to initiate the auto-tune sequence, abort the sequence, and apply the suggested tuning values or your own tuning values.

7.9.2 PID algorithm

In steady state operation, a PID controller regulates the value of the output so as to drive the error (e) to zero. A measure of the error is given by the difference between the setpoint (SP) (the desired operating point) and the process variable (PV) (the actual operating point). The principle of PID control is based upon the following equation that expresses the output, $M(t)$, as a function of a proportional term, an integral term, and a differential term:

Output = Proportional term + Integral term + Differential term

$$M(t) = K_c * e + K_c \int_0^t e dt + M_{initial} + K_c * de/dt$$

where:

M(t)	Loop output as a function of time
K_c	Loop gain
e	Loop error (the difference between setpoint and process variable)
M_{initial}	Initial value of the loop output

In order to implement this control function in a digital computer, the continuous function must be quantized into periodic samples of the error value with subsequent calculation of the output. The corresponding equation that is the basis for the digital computer solution is:

Output = Proportional term + Integral term + Differential term

$$M_n = K_c * e_n + K_i * \sum_{i=1}^n e_i + M_{initial} + K_d * (e_n - e_{n-1})$$

where:

M_n	Calculated value of the loop output at sample time n
K_c	Loop gain
e_n	Value of the loop error at sample time n
e_{n-1}	Previous value of the loop error (at sample time n - 1)
K_i	Proportional constant of the integral term
M_{initial}	Initial value of the loop output
K_d	Proportional constant of the differential term

From this equation, the integral term is shown to be a function of all the error terms from the first sample to the current sample. The differential term is a function of the current sample and the previous sample, while the proportional term is only a function of the current sample. In a digital computer, it is not practical to store all samples of the error term, nor is it necessary.

Since the digital computer must calculate the output value each time the error is sampled beginning with the first sample, it is only necessary to store the previous value of the error and the previous value of the integral term. As a result of the repetitive nature of the digital computer solution, a simplification in the equation that must be solved at any sample time can be made. The simplified equation is:

Output = Proportional term + Integral term + Differential term

$$M_n = K_c * e_n + K_i * e_n + M_{initial} + K_d * (e_n - e_{n-1})$$

where:

M_n	Calculated value of the loop output at sample time n
K_c	Loop gain
e_n	Value of the loop error at sample time n
e_{n-1}	Previous value of the loop error (at sample time n - 1)
K_i	Proportional constant of the integral term
MX	Previous value of the integral term (at sample time n - 1)
K_D	Proportional constant of the differential term

The CPU uses a modified form of the above simplified equation when calculating the loop output value. This modified equation is:

$$\text{Output} = \text{Proportional term} + \text{Integral term} + \text{Differential term}$$

$$M_n = MP_n + MI_n + MD_n$$

where:

M_n	Calculated value of the loop output at sample time n
MP_n	Value of the proportional term of the loop output at sample time n
MI_n	Value of the integral term of the loop output at sample time n
MD_n	Value of the differential term of the loop output at sample time n

Understanding the elements of the PID equation

Proportional term of the PID equation: The proportional term MP is the product of the gain (K_c), which controls the sensitivity of the output calculation, and the error (e), which is the difference between the setpoint (SP) and the process variable (PV) at a given sample time. The equation for the proportional term as solved by the CPU is:

$$MP_n = K_c * (SP_n - PV_n)$$

where:

MP_n	Value of the proportional term of the loop output at sample time n
K_c	Loop gain
SP_n	Value of the setpoint at sample time n
PV_n	Value of the process variable at sample time n

Integral term of the PID equation: The integral term MI is proportional to the sum of the error (e) over time. The equation for the integral term as solved by the CPU is:

$$MI_n = K_i e_n + MX = K_c * (T_S / T_I) * (SP_n - PV_n) + MX$$

where:

MI_n	Value of the integral term of the loop output at sample time n
K_c	Loop gain
T_s	Loop sample time
T_i	Integral time (also called the integral time or reset)
SP_n	Value of the setpoint at sample time n
PV_n	Value of the process variable at sample time n
MX	Value of the integral term at sample time n-1 (also called the integral sum or the bias)

The integral sum or bias (MX) is the running sum of all previous values of the integral term. After each calculation of MI_n , the bias is updated with the value of MI_n which might be adjusted or clamped (see the section "Variables and Ranges" for details). The initial value of the bias is typically set to the output value ($MI_{initial}$) just prior to the first loop output calculation. Several constants are also part of the integral term, the gain (K_c), the sample time (T_s), which is the cycle time at which the PID loop recalculates the output value, and the integral time or reset (T_i), which is a time used to control the influence of the integral term in the output calculation.

Differential term of the PID equation: The differential term MD is proportional to the change in the error. The CPU uses the following equation for the differential term:

$$MD_n = K_c * (T_d / T_s) * ((SP_n - PV_n) - (SP_{n-1} - PV_{n-1}))$$

To avoid step changes or bumps in the output due to derivative action on setpoint changes, this equation is modified to assume that the setpoint is a constant ($SP_n = SP_{n-1}$). This results in the calculation of the change in the process variable instead of the change in the error as shown:

$$MD_n = K_c * (T_d / T_s) * ((PV_{n-1} - PV_n))$$

or just:

$$MD_n = K_c * (T_d / T_s) * (PV_{n-1} - PV_n)$$

where:

MD_n	Value of the differential term of the loop output at sample time n
K_c	Loop gain
T_s	Loop sample time
T_d	Differentiation period of the loop (also called the derivative time or rate)
SP_n	Value of the setpoint at sample time n
SP_{n-1}	Value of the setpoint at sample time n - 1
PV_n	Value of the process variable at sample time n - 1
PV_{n-1}	Value of the process variable at sample time n - 1

The process variable rather than the error must be saved for use in the next calculation of the differential term. At the time of the first sample, the value of PV_{n-1} is initialized to be equal to PV_n .

Selecting the type of loop control

In many control systems, it might be necessary to employ only one or two methods of loop control. For example, only proportional control or proportional and integral control might be required. The selection of the type of loop control desired is made by setting the value of the constant parameters.

If you do not want integral action (no "I" in the PID calculation), then a value of infinity "INF", should be specified for the integral time (reset). Even with no integral action, the value of the integral term might not be zero, due to the initial value of the integral sum MX.

If you do not want derivative action (no "D" in the PID calculation), then a value of 0.0 should be assigned for the derivative time (rate).

If you do not want proportional action (no "P" in the PID calculation) and you want I or ID control, then a value of 0.0 should be specified for the gain. Since the loop gain is a factor in the equations for calculating the integral and differential terms, setting a value of 0.0 for the loop gain will result in a value of 1.0 being used for the loop gain in the calculation of the integral and differential terms.

7.9.3 Converting and normalizing the loop inputs

A loop has two input variables, the setpoint and the process variable. The setpoint is generally a fixed value such as the speed setting on the cruise control in your automobile. The process variable is a value that is related to loop output and therefore measures the effect that the loop output has on the controlled system. In the example of the cruise control, the process variable would be a tachometer input that measures the rotational speed of the tires.

Both the setpoint and the process variable are real world values whose magnitude, range, and engineering units could be different. Before these real world values can be operated upon by the PID instruction, the values must be converted to normalized, floating-point representations.

The first step is to convert the real world value from a 16-bit integer value to a floating-point or real number value. The following instruction sequence is provided to show how to convert from an integer value to a real number.

```
ITD    AIW0, AC0    //Convert an input value to a double word
DTR    AC0, AC0    //Convert the 32-bit integer to a real number
```

The next step is to convert the real number value representation of the real world value to a normalized value between 0.0 and 1.0. The following equation is used to normalize either the setpoint or process variable value:

$$R_{Norm} = ((R_{Raw} / Span) + Offset)$$

where:

R_{Norm} is the normalized, real number value representation of the real world value

R_{Raw} is the un-normalized or raw, real number value representation of the real world value

Offset	is 0.0 for unipolar values is 0.5 for bipolar values
Span	is the maximum possible value minus the minimum possible value: = 27,648 for unipolar values (typical) = 55,296 for bipolar values (typical)

The following instruction sequence shows how to normalize the bipolar value in AC0 (whose span is 55,296) as a continuation of the previous instruction sequence:

```
/R    55296.0, AC0 //Normalize the value in the accumulator
+R    0.5, AC0      //Offset the value to the range from 0.0 to 1.0
MOVR AC0, VD100 //Store the normalized value in the loop TABLE
```

7.9.4 Converting the loop output to a scaled integer value

The loop output is the control variable, such as the throttle setting of the cruise control on an automobile. The loop output is a normalized, real number value between 0.0 and 1.0. Before the loop output can be used to drive an analog output, the loop output must be converted to a 16-bit, scaled integer value. This process is the reverse of converting the PV and SP to a normalized value. The first step is to convert the loop output to a scaled, real number value using the formula given below:

$$R_{Scal} = (M_n - \text{Offset}) * \text{Span}$$

R_{Scal}	is the scaled, real number value of the loop output
M_n	is the normalized, real number value of the loop output
Offset	is 0.0 for unipolar values is 0.5 for bipolar values
Span	is the maximum possible value minus the minimum possible value = 27,648 for unipolar values (typical) = 55,296 for bipolar values (typical)

The following instruction sequence shows how to scale the loop output:

```
MOVR VD108, AC0 //Moves the loop output to the accumulator
-R    0.5, AC0      //Include this statement only if the value is bipolar
*R    55296.0, AC0 //Scales the value in the accumulator
```

Next, the scaled, real number value representing the loop output must be converted to a 16-bit integer. The following instruction sequence shows how to do this conversion:

```
ROUND AC0, AC0 //Converts the real number to a 32-bit integer
DTI    AC0, LW0 //Converts the value to a 16-bit integer
MOVW  LW0, AQW0 //Writes the value to the analog output
```

7.9.5 Forward- or reverse-acting loops

The loop is forward-acting if the gain is positive and reverse-acting if the gain is negative. (For I or ID control, where the gain value is 0.0, specifying positive values for integral and derivative time will result in a forward-acting loop, and specifying negative values will result in a reverse-acting loop.)

Variables and ranges

The process variable and setpoint are inputs to the PID calculation. Therefore the loop table fields for these variables are read but not altered by the PID instruction.

The output value is generated by the PID calculation, so the output value field in the loop table is updated at the completion of each PID calculation. The output value is clamped between 0.0 and 1.0. The output value field can be used as an input by the user to specify an initial output value when making the transition from manual control to PID instruction (auto) control of the output. (See the discussion in the "Modes" section below.)

If integral control is being used, then the bias value is updated by the PID calculation and the updated value is used as an input in the next PID calculation. When the calculated output value goes out of range (output would be less than 0.0 or greater than 1.0), the bias is adjusted according to the following formulas:

- when the calculated output $M_n > 1.0$

$$MX = 1.0 - (MP_n + MD_n)$$

- when the calculated output $M_n < 0$

$$MX = - (MP_n + MD_n)$$

MX is the value of the adjusted bias

MP_n is the value of the proportional term of the loop output at sample time n

MD_n is the value of the differential term of the loop output at sample time n

M_n is the value of the loop output at sample time n

By adjusting the bias as described, an improvement in system responsiveness is achieved once the calculated output comes back into the proper range. The calculated bias is also clamped between 0.0 and 1.0 and then is written to the bias field of the loop table at the completion of each PID calculation. The value stored in the loop table is used in the next PID calculation.

The bias value in the loop table can be modified by the user prior to execution of the PID instruction in order to address bias value problems in certain application situations. Care must be taken when manually adjusting the bias, and any bias value written into the loop table must be a real number between 0.0 and 1.0.

A comparison value of the process variable is maintained in the loop table for use in the derivative action part of the PID calculation. You should not modify this value.

Modes

There is no built-in mode control for PID loops. The PID calculation is performed only when power flows to the PID box. Therefore, "automatic" or "auto" mode exists when the PID calculation is performed cyclically. "Manual" mode exists when the PID calculation is not performed.

The PID instruction has a power-flow history bit, similar to a counter instruction. The instruction uses this history bit to detect a 0-to-1 power-flow transition. When the power-flow transition is detected, it will cause the instruction to perform a series of actions to provide a bumpless change from manual control to auto control. In order for change to auto mode control to be bumpless, the value of the output as set by the manual control must be supplied as an input to the PID instruction (written to the loop table entry for M_n) before switching to auto control. The PID instruction performs the following actions to values in the loop table to ensure a bumpless change from manual to auto control when a 0-to-1 power-flow transition is detected:

- Sets setpoint (SP_n) = process variable (PV_n)
- Sets old process variable (PV_{n-1}) = process variable (PV_n)
- Sets bias (MX) = output value (M_n)

The default state of the PID history bits is "set" and that state is established at startup and on every STOP-to-RUN mode transition of the controller. If power flows to the PID box the first time that it is executed after entering RUN mode, then no power-flow transition is detected and the bumpless mode change actions are not performed.

Alarm checking and special operations

The PID instruction is a simple but powerful instruction that performs the PID calculation. If other processing is required such as alarm checking or special calculations on loop variables, these must be implemented using the basic instructions supported by the CPU.

Error conditions

When it is time to compile, the CPU will generate a compile error (range error) and the compilation will fail if the loop table start address or PID loop number operands specified in the instruction are out of range.

Certain loop table input values are not range checked by the PID instruction. You must take care to ensure that the process variable and setpoint (as well as the bias and previous process variable if used as inputs) are real numbers between 0.0 and 1.0.

If any error is encountered while performing the mathematical operations of the PID calculation, then SM1.1 (overflow or illegal value) is set and execution of the PID instruction is terminated. (Update of the output values in the loop table could be incomplete, so you should disregard these values and correct the input value causing the mathematical error before the next execution of the loop's PID instruction.)

Loop table

The loop table is 80 bytes long and has the format shown in the following table.

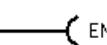
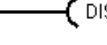
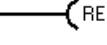
Offset	Field	Format	Type	Description
0	Process variable (PV_n)	REAL	In	Contains the process variable, which must be scaled between 0.0 and 1.0.
4	Setpoint (SP_n)	REAL	In	Contains the setpoint, which must be scaled between 0.0 and 1.0.
8	Output (M_n)	REAL	In/Out	Contains the calculated output, scaled between 0.0 and 1.0.
12	Gain (K_C)	REAL	In	Contains the gain, which is a proportional constant. Can be a positive or negative number.
16	Sample time (T_S)	REAL	In	Contains the sample time, in seconds. Must be a positive number.
20	Integral time or reset (T_I)	REAL	In	Contains the integral time or reset, in minutes. Must be a positive number.
24	Derivative time or rate (T_D)	REAL	In	Contains the derivative time or rate, in minutes. Must be a positive number.
28	Bias (MX)	REAL	In/Out	Contains the bias or integral sum value between 0.0 and 1.0.
32	Previous process variable (PV_{n-1})	REAL	In/Out	Contains the value of the process variable stored from the last execution of the PID instruction.
36 to 79	Reserved for auto-tuning variables. Refer to PID loop definition table (Page 424) for details.			

7.10 Interrupt

7.10.1 Interrupt instructions

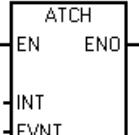
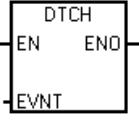
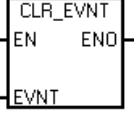
When you make the transition to RUN mode, interrupts are initially disabled. In RUN mode, you can enable interrupt processing by executing the ENI (enable Interrupt) instruction. Executing the DISI (disable interrupt) instruction inhibits the processing of interrupts; however, active interrupt events will continue to be queued.

ENI, DISI, and RETI

LAD	FBD	STL	Description
		ENI	The enable interrupt instruction globally enables processing of all attached interrupt events.
		DISI	The disable interrupt instruction globally disables processing of all interrupt events.
		CRETI	The conditional return from interrupt instruction can be used to return from an interrupt, based upon the condition of the preceding program logic.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0004H Attempted ENI or DISI execution disallowed in an interrupt routine 	None

ATCH, DTCH, and CEVENT

LAD / FBD	STL	Description
	ATCH INT, EVNT	The attach interrupt instruction associates an interrupt event EVNT with an interrupt routine number INT and enables the interrupt event.
	DTCH EVNT	The detach interrupt instruction disassociates an interrupt event EVNT from all interrupt routines and disables the interrupt event.
	CEVENT EVNT	The clear interrupt event instruction removes all interrupt events of type EVNT from the interrupt queue. Use this instruction to clear the interrupt queue of unwanted interrupt events. If this instruction is being used to clear out spurious interrupt events, then you should detach the event before clearing the events from the queue. Otherwise new events will be added to the queue after the clear event instruction has been executed.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0002H Conflicting assignment of inputs to an HSC • 0016H Attempt to use HSC or edge interrupt on input channel that is already allocated to a motion control function • 0019H Attempt to use a signal board function without a signal board installed and configured • 0090H Invalid operand (illegal event number) 	None

Input / output	Data type	Operand
INT	BYTE	Constant: interrupt routine number (0 to 127)
EVNT	BYTE	Constant: interrupt event number CPU CR40, CR60: 0-13, 16-18, 21-23, 27, 28, and 32 CPU SR20/ST20, SR30/ST30, SR40/ST40, SR60/ST60: 0-13, 16-18, 21-28, 32, and 35-38

7.10.2 Interrupt routine overview and CPU model event support

Before an interrupt routine can be invoked, an association must be assigned between the interrupt event and the program segment that you want to execute when the event occurs. Use the attach interrupt instruction to associate an interrupt event (specified by the interrupt event number) and the program segment (specified by an interrupt routine number). You can attach multiple interrupt events to one interrupt routine, but one event cannot be concurrently attached to multiple interrupt routines.

When you attach an event and interrupt routine, new occurrences of this event will cause execution of the attached interrupt routine only if the global ENI (enable interrupts) instruction was executed and interrupt event processing is active. Otherwise, the event is added to the interrupt event queue. If you disable all interrupts using the global DISI (disable interrupts) instruction, each occurrence of the interrupt event is queued until interrupts are re-enabled, using the global ENI (enable interrupt) instruction, or the interrupt queue overflows.

You can disable individual interrupt events by breaking the association between the interrupt event and the interrupt routine with the detach Interrupt instruction. The detach interrupt instruction returns the interrupt to an inactive or ignored state. The following table lists the different types of interrupt events.

Event	Description	CR40/CR60	SR20/ST20 SR30/ST30 SR40/ST40 SR60/ST60
0	Rising edge I0.0	Y	Y
1	Falling edge I0.0	Y	Y
2	Rising edge I0.1	Y	Y
3	Falling edge I0.1	Y	Y
4	Rising edge I0.2	Y	Y
5	Falling edge I0.2	Y	Y
6	Rising edge I0.3	Y	Y
7	Falling edge I0.3	Y	Y
8	Port 0 Receive character	Y	Y
9	Port 0 Transmit complete	Y	Y
10	Timed interrupt 0 (SMB34 controls the time interval)	Y	Y
11	Timed interrupt 1 (SMB35 controls the time interval)	Y	Y
12	HSC0 CV=PV (current value = preset value)	Y	Y
13	HSC1 CV=PV (current value = preset value)	Y	Y
14-15	Reserved	N	N
16	HSC2 CV=PV (current value = preset value)	Y	Y
17	HSC2 Direction changed	Y	Y
18	HSC2 External reset	Y	Y
19-20	Reserved	N	N
21	Timer T32 CT=PT (current time = preset time)	Y	Y
22	Timer T96 CT=PT (current time = preset time)	Y	Y
23	Port 0 Receive message complete	Y	Y
24	Port 1 Receive message complete	N	Y
25	Port 1 Receive character	N	Y
26	Port 1 Transmit complete	N	Y
27	HSC0 Direction changed	Y	Y
28	HSC0 External reset	Y	Y
29-31	Reserved	N	N
32	HSC3 CV=PV (current value = preset value)	Y	Y
33-34	Reserved	N	N
35	Rising edge, signal board input 0	N	Y
36	Falling edge, signal board input 0	N	Y
37	Rising edge, signal board Input 1	N	Y
38	Falling edge, signal board input 1	N	Y

7.10.3 Interrupt programming guidelines

Interrupt routine execution

An interrupt routine executes in response to an associated internal or external event. Once the last instruction of an interrupt routine has executed, control returns to the point in the scan cycle at the point of the interruption. You can exit the routine by executing a conditional return from interrupt instruction (CRETI).

Interrupt processing provides quick reaction to special internal or external events. Optimize your interrupt routines to perform a specific task, and then return control to the scan cycle.

Note

- You cannot use the disable interrupt (DISI), enable interrupt (ENI), high-speed counter definition (HDEF), and end (END) instructions in an interrupt routine.
 - Keep interrupt routine program logic short and to the point, so execution is quick and other processes are not deferred for long periods of time. If this is not done, unexpected conditions can cause abnormal operation of equipment controlled by the main program.
-

System support for interrupts

Because interrupts can affect contact, coil, and accumulator logic, the system saves and reloads the logic stack, accumulator registers, and the special memory bits (SM) that indicate the status of accumulator and instruction operations. This avoids disruption to the main user program caused by branching to and from an interrupt routine.

Calling subroutines from interrupt routines

You can call four nesting levels of subroutines from an interrupt routine. The accumulators and the logic stack are shared between an interrupt routine and the four nesting levels of subroutines called from the interrupt routine

Sharing data between the main program and the interrupt routines

You can share data between the main program and one or more interrupt routines. Because it is not possible to predict when the CPU might generate an interrupt, it is desirable to limit the number of variables that are used by both the interrupt routine and elsewhere in the program. Problems with the consistency of shared data can result due to the actions of interrupt routines when the execution of instructions in your main program is interrupted by interrupt events. Use the interrupt block "variable table" (block call interface table) to ensure that your interrupt routine uses only the temporary memory and does not overwrite data used somewhere else in your program.

Ensuring access for a single shared variable

- For an STL program that is sharing a single variable: If the shared data is a single byte, word, or double word variable and your program is written in STL, then correct shared access can be ensured by storing the intermediate values from operations on shared data only in non-shared memory locations or accumulators.
- For a LAD program that is sharing a single variable: If the shared data is a single byte, word, or double word variable and your program is written in LAD, then correct shared access can be ensured by establishing the convention that access to shared memory locations be made using only Move instructions (MOVB, MOVW, MOVD, MOVR). While many LAD instructions are composed of interruptible sequences of STL instructions, these Move instructions are composed of a single STL instruction whose execution cannot be affected by interrupt events.

Ensuring access for multiple shared variables

For an STL or LAD program that is sharing multiple variables: If the shared data is composed of a number of related bytes, words, or double words, then the interrupt disable/enable instructions (DISI and ENI) can be used to control interrupt routine execution. At the point in your main program where operations on shared memory locations are to begin, disable the interrupts. Once all actions affecting the shared locations are complete, re-enable the interrupts. During the time that interrupts are disabled, interrupt routines cannot be executed and therefore cannot access shared memory locations; however, this approach can result in delayed response to interrupt events.

7.10.4 Types of interrupt events that the S7-200 SMART CPU supports

Communication port interrupts

The serial communications port of the CPU can be controlled by your program. This mode of operating the communications port is called Freeport mode. In Freeport mode, your program defines the baud rate, bits per character, parity, and protocol. The Receive and Transmit interrupts are available to facilitate your program-controlled communications. Refer to the Transmit and Receive instructions for more information.

I/O interrupts

I/O interrupts include rising/falling edge interrupts and high-speed counter interrupts. The CPU can generate an interrupt on rising and/or falling edges of an input for input channels I0.0, I0.1, I0.2, and I0.3 (and for I7.0 and I7.1 for standard CPUs with an optional digital input signal board). The rising edge and the falling edge events can be captured for each of these input points. These rising/falling edge events can be used to signify a condition that must receive immediate attention when the event happens.

The high-speed counter interrupts allow you to respond to conditions such as the current value reaching the preset value, a change in counting direction that might correspond to a reversal in the direction in which a shaft is turning, or an external reset of the counter. Each of these high-speed counter events allows action to be taken in real time in response to high-speed events that cannot be controlled at programmable logic controller scan speeds.

You enable each of the above interrupts by attaching an interrupt routine to the related I/O event.

Time-based interrupts

Time-based interrupts include timed interrupts and the timer T32/T96 interrupts. You can specify actions to be taken on a cyclic basis using a timed interrupt. The cycle time is set in 1-ms increments from 1 ms to 255 ms. You must write the cycle time in SMB34 for timed interrupt 0, and in SMB35 for timed interrupt 1.

The timed interrupt event transfers control to the appropriate interrupt routine each time the timer expires. Typically, you use timed interrupts to control the sampling of analog inputs or to execute a PID loop at regular intervals.

A timed interrupt is enabled and timing begins when you attach an interrupt routine to a timed interrupt event. During the attachment, the system captures the cycle time value, so subsequent changes to SMB34 and SMB35 do not affect the cycle time. To change the cycle time, you must modify the cycle time value, and then re-attach the interrupt routine to the timed interrupt event. When the re-attachment occurs, the timed interrupt function clears any accumulated time from the previous attachment and begins timing with the new value.

After being enabled, the timed interrupt runs continuously and executes the attached interrupt routine, at the end of each successive time interval. If you exit RUN mode or detach the timed interrupt, the timed interrupt is disabled. If the global DISI (disable interrupt) instruction is executed, timed interrupts continue to occur, but the attached interrupt routine is not processed yet. Each occurrence of the timed interrupt is queued (until either interrupts are enabled or the queue is full).

The timer T32/T96 interrupts allow timely response to the completion of a specified time interval. These interrupts are only supported for the 1-ms resolution on-delay (TON) and off-delay (TOF) timers T32 and T96. The T32 and T96 timers otherwise behave normally. Once the interrupt is enabled, the attached interrupt routine is executed when the active timer's current value becomes equal to the preset time value during the normal 1-ms timer update performed in the CPU. You enable these interrupts by attaching an interrupt routine to the T32 (event 21) and T96 (event 22) interrupt events.

7.10.5 Interrupt priority, queuing, and example program

Interrupt service

Interrupts are serviced by the CPU on a first-come-first-served basis within their respective priority group. Only one user-interrupt routine is ever being executed at any point in time. Once the execution of an interrupt routine begins, the routine is executed to completion. It cannot be pre-empted by another interrupt routine, even by a higher priority routine. Interrupts that occur while another interrupt is being processed are queued for later processing. The following table shows the three interrupt queues and the maximum number of interrupts they can store.

It is possible that more interrupts can occur than a queue can hold. Therefore, queue overflow memory bits (identifying the type of interrupt events that have been lost) are maintained by the system. The following table shows the interrupt queue overflow bits. You should use these bits only in an interrupt routine because they are reset when the queue is emptied, and control is returned to the scan cycle.

If multiple interrupt events occur at the same time, the priority (group and within a group) determines which interrupt event is processed first. Once the highest priority has been handled, the queue is examined to find the current highest priority event that remains in the queue and the interrupt routine attached to that event is executed. This continues until the queue is empty and control is returned to the scan cycle.

The following table shows all interrupt events, with their priority and assigned event number.

Maximum number of entries per interrupt queue

Queue	Queue depth for all S7-200 SMART CPU models
Communications queue	4
I/O interrupt queue	16
Timed interrupt queue	8

Interrupt queue overflow bits

Description (0 = No Overflow, 1 = Overflow)	SM Bit
Communications queue	SM4.0
I/O Interrupt queue	SM4.1
Timed Interrupt queue	SM4.2

Priority order for interrupt events

Priority group	Event	Description
Communications <i>Highest Priority</i>	8	Port 0 Receive character
	9	Port 0 Transmit complete
	23	Port 0 Receive message complete
	24	Port 1 Receive message complete
	25	Port 1 Receive character
	26	Port 1 Transmit complete
Discrete <i>Medium Priority</i>	0	I0.0 Rising edge
	2	I0.1 Rising edge
	4	I0.2 Rising edge
	6	I0.3 Rising edge
	35	I7.0 Rising edge (signal board)
	37	I7.1 Rising edge (signal board)
	1	I0.0 Falling edge
	3	I0.1 Falling edge

Priority group	Event	Description
<i>Timed Lowest Priority</i>	5	I0.2 Falling edge
	7	I0.3 Falling edge
	36	I7.0 Falling edge (signal board)
	38	I7.1 Falling edge (signal board)
	12	HSC0 CV=PV (current value = preset value)
	27	HSC0 Direction changed
	28	HSC0 External reset
	13	HSC1 CV=PV (current value = preset value)
	16	HSC2 CV=PV (current value = preset value)
	17	HSC2 Direction changed
	18	HSC2 External reset
	32	HSC3 CV=PV (current value = preset value)
<i>Timed Lowest Priority</i>	10	Timed interrupt 0 SMB34
	11	Timed interrupt 1 SMB35
	21	Timer T32 CT=PT interrupt
	22	Timer T96 CT=PT interrupt

Example 1: Input signal edge detector interrupt

LAD	STL
MAIN Network 1	<p>On the first scan:</p> <ol style="list-style-type: none"> Define interrupt routine INT_0 to be a falling-edge interrupt for I0.0. Globally enable interrupts.
Network 2	<p>If an I/O error is detected, disable the falling-edge interrupt for I0.0. (This network is optional.)</p>
Network 3	<p>When M5.0 is on, disable all interrupts. When disabled, attached interrupt events will be queued, but the corresponding interrupt routines will not be executed until interrupts are re-enabled with the ENI instruction.</p>
INT 0 Network 1	<p>I0.0 falling-edge interrupt routine: Conditional return based on an I/O error.</p>

Example 2: Timed interrupt for reading the value of an analog input

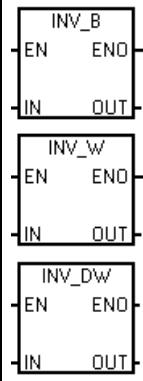
LAD	STL
MAIN Network 1	Network 1 LD SM0.1 CALL SBR_0
SBR 0 Network 1	Set the interval for the timed interrupt 0 to 100 ms. Attach timed interrupt 0 (Event 10) to INT_0. Global interrupt enable.
INT 0 Network 1	Read the value of AIW16 every 100 ms.

Example 3: Clear interrupt event instruction

LAD	STL
<p>SBR 1 Network 1</p> <p>HSC instruction wizard: Set control bits, write preset. PV = 6 Attach interrupt HSC1_STEP1: CV = PV for HC1 Configure HSC 1.</p>	Network 1 LD SM0.0 MOVB 16#A0, SMB47 MOVD +6, SMD52 ATCH HSC1_STEP1, 13
<p>SBR 1 Network 2</p> <p>Clear unwanted interrupts caused by machine vibration.</p>	Network 2 LD SM0.0 CEVNT 13

7.11 Logical operations

7.11.1 Invert

LAD / FBD	STL	Description
	INV_B OUT INVW OUT INVD OUT	The Invert Byte, Invert Word, and Invert Double Word instructions form the one's complement of the input IN and load the result into the memory location OUT

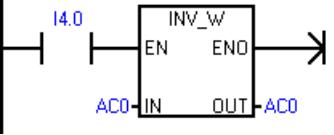
Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	• SM1.0 Result of operation = zero

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

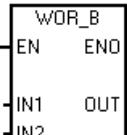
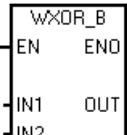
Program instructions

7.11 Logical operations

Example: Invert instruction

LAD	STL		
	<p>Invert word value in AC0. Result is put in AC0.</p> <p>Invert Word input AC0 <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1101 0111 1001 01</td></tr></table></p> <p>Execute one's complement inv</p> <p>Invert Word output AC0 <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0010 1000 0110 10</td></tr></table></p>	1101 0111 1001 01	0010 1000 0110 10
1101 0111 1001 01			
0010 1000 0110 10			

7.11.2 AND, OR, and exclusive OR

LAD / FBD	STL	Description
 WAND_W WAND_DW	ANDB IN1, OUT ANDW IN1, OUT ANDD IN1, OUT	<p>The AND Byte, AND Word, and AND Double Word instructions logically AND the corresponding bits of two input values IN1 and IN2 and load the result in a memory location assigned to OUT.</p> <ul style="list-style-type: none"> • LAD and FBD: IN1 AND IN2 = OUT • STL: IN1 AND OUT = OUT
 WOR_W WOR_DW	ORB IN1, OUT ORW IN1, OUT ORD IN1, OUT	<p>The OR Byte, OR Word, and OR Double Word instructions logically OR the corresponding bits of two input values IN1 and IN2 and load the result in a memory location assigned to OUT.</p> <ul style="list-style-type: none"> • LAD and FBD: IN1 OR IN2 = OUT • STL: IN1 OR OUT = OUT
 WXOR_W WXOR_DW	XORB IN1, OUT XORW IN1, OUT XORD IN1, OUT	<p>The Exclusive OR Byte, Exclusive OR Word, and Exclusive OR Double Word instructions logically XOR the corresponding bits of two input values IN1 and IN2 and load the result in a memory location OUT.</p> <ul style="list-style-type: none"> • LAD and FBD: IN1 XOR IN2 = OUT • STL: IN1 XOR OUT = OUT

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	• SM1.0 Result of operation = zero

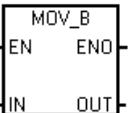
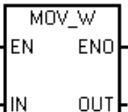
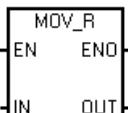
Input / output	Data type	Operand
IN1, IN2	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *AC, *LD

Example: AND, OR, and Exclusive OR instructions

LAD	STL																		
<pre> LD I4.0 WAND_W EN I4.0 ENO OUT1 AC1 IN1 AC0 IN2 WOR_W EN OUT2 ENO VW100 AC1 IN1 AC0 IN2 WXOR_W EN OUT3 ENO AC0 AC1 IN1 AC0 IN2 </pre>	<p>Network 1</p> <pre> LD I4.0 ANDW AC1, AC0 ORW AC1, VW100 XORW AC1, AC0 </pre> <p>equals</p> <table border="1"> <tr><td>AC1</td><td>0001 1111 0110 1101</td></tr> <tr><td>AC0</td><td>1101 0011 1110 0110</td></tr> <tr><td>AC0</td><td>0001 0011 0110 0100</td></tr> </table> <table border="1"> <tr><td>AC1</td><td>0001 1111 0110 1101</td></tr> <tr><td>VW100</td><td>1101 0011 1010 0000</td></tr> <tr><td>VW100</td><td>1101 1111 1110 1101</td></tr> </table> <table border="1"> <tr><td>AC1</td><td>0001 1111 0110 1101</td></tr> <tr><td>AC0</td><td>0001 0011 0110 0100</td></tr> <tr><td>AC0</td><td>0000 1100 0000 1001</td></tr> </table>	AC1	0001 1111 0110 1101	AC0	1101 0011 1110 0110	AC0	0001 0011 0110 0100	AC1	0001 1111 0110 1101	VW100	1101 0011 1010 0000	VW100	1101 1111 1110 1101	AC1	0001 1111 0110 1101	AC0	0001 0011 0110 0100	AC0	0000 1100 0000 1001
AC1	0001 1111 0110 1101																		
AC0	1101 0011 1110 0110																		
AC0	0001 0011 0110 0100																		
AC1	0001 1111 0110 1101																		
VW100	1101 0011 1010 0000																		
VW100	1101 1111 1110 1101																		
AC1	0001 1111 0110 1101																		
AC0	0001 0011 0110 0100																		
AC0	0000 1100 0000 1001																		

7.12 Move

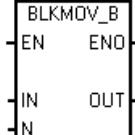
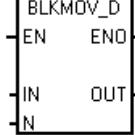
7.12.1 Move byte, word, double word, or real

LAD / FBD	STL	Description
	MOV_B IN, OUT	The Move Byte, Move Word, Move Double Word, and Move Real instructions move a data value from a source (constant or memory location) IN to a new memory location OUT, without changing the value stored in a source memory location.
	MOV_W IN, OUT	
	MOV_{DW} IN, OUT	
	MOV_R IN, OUT	

Non-fatal error with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *AC, *LD, Constant
	DWORD, DINT	ID, QD, VD, MD, SMD, SD, LD, HC, &VB, &QB, &MB, &SB, &T, &C, &SMB, &AIW, &AQW, AC, *VD, *LD, *AC, Constant
	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC
	DWORD, DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

7.12.2 Block move

LAD / FBD	STL	Description
	BMB IN, OUT, N	The Block Move Byte, Block Move Word, and Block Move Double Word instructions move an assigned block of data values from a source memory location (starting address IN and successive addresses) to a new memory location (starting address OUT and successive addresses). Parameter N assigns the number of bytes, words, or double words to move. The block of data values stored in the source location are not changed.
	BMW IN, OUT, N	N has a range of 1 to 255.
	BMD IN, OUT, N	

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range 	None

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AIW, *VD, *LD, *AC
	DWORD, DINT	ID, QD, VD, MD, SMD, SD, LD, *VD, *LD, *AC
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, *VD, *LD, *AC
	WORD, INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AQW, *VD, *LD, *AC
	DWORD, DINT	ID, QD, VD, MD, SMD, SD, LD, *VD, *LD, *AC
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, Constant, *VD, *LD, *AC

Example: Block Move instruction

LAD	STL			
	Network 1 LD I2.1 BMB VB20, VB100, 4			
Source data values	30 31 32 33			
Source data addresses	VB20 VB21 VB22 VB23			
If I2.1 = 1, then execute BLKMOV_B to move source data values to destination addresses				
Destination data values	30 31 32 33			
Destination data addresses	VB100 VB101 VB102 VB103			

7.12.3 Swap bytes

LAD / FBD	STL	Description
	SWAP IN	The Swap Bytes instruction exchanges the most significant byte with the least significant byte of the word IN.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
IN	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC

Example: Swap instructions

LAD	STL
	Network 1 LD I2.1 SWAP VW50

Hexadecimal data values	D6	C3
Data addresses	VB50	VB51
If I2.1 = 1, then execute SWAP to exchange the byte data in a data word		
Hexadecimal data values	C3	D6
Data addresses	VB50	VB51

7.12.4 Move byte immediate (read and write)

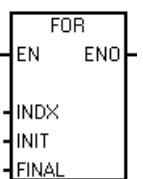
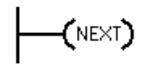
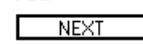
LAD / FBD	STL	Description
	BIR IN, OUT	The Move Byte Immediate Read instruction reads the state of physical input IN and writes the result to the memory address OUT, but the process-image register is not updated.
	BIW IN, OUT	The Move Byte Immediate Write instruction reads the data from the memory address IN and writes to physical output OUT, and the corresponding process image location.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> 0006H Indirect address Unable to access expansion module 	None

Input / output	Data type	Operand
IN (BIR)	BYTE	IB, *VD, *LD, *AC
IN (BIW)	BYTE	B, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT (BIR)	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
OUT (BIW)	BYTE	QB, *VD, *LD, *AC

7.13 Program control

7.13.1 FOR-NEXT loop

LAD / FBD	STL	Description
	FOR INDX, INIT, FINAL	The FOR instruction executes the instructions between the FOR and the NEXT instructions. You assign the index value or current loop count INDX, the starting loop count INIT, and the ending loop count FINAL.
 	NEXT	The NEXT instruction marks the end of the FOR loop program segment.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
INDX	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
INIT, FINAL	INT	VW, IW, QW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant

Use the FOR and NEXT instructions to execute a program segment in a loop that is repeated for the assigned count. Each FOR instruction requires one NEXT instruction. You place a FOR-NEXT loop within a FOR-NEXT loop to a maximum nesting depth of eight.

If you enable a FOR-NEXT loop, the execution loop continues until it finishes the iterations, unless you change the FINAL value from within the loop itself. You can change the values while the FOR-NEXT loop is in the looping process. When the loop is enabled again, it copies the INIT value to the INDX value (current loop number).

For example, given an INIT value of 1 and a FINAL value of 10, the instructions between the FOR instruction and the NEXT instruction are executed 10 times with the INDX value being incremented: 1, 2, 3, ... 10.

If the INIT value is greater than the FINAL value, the loop is not executed. After each execution of the instructions between the FOR instruction and the NEXT instruction, the INDX value is incremented and the result is compared to the final value. If the INDX is greater than the final value, the execution loop is terminated.

For STL, if the top of the logic stack value is 1 when your program enters the FOR-NEXT loop, then the top of the logic stack will be 1 when your program exits the FOR-NEXT loop.

Example: FOR-NEXT loop

LAD		STL
1		When I2.0 is ON, the outside loop (Network 1 - 4) is executed 100 times. Network 1 LD I2.0 FOR VW100, +1, +100
2		The inside loop (Network 2 - 3) is executed twice for each execution of the outside loop when I2.1 is on. Network 2 LD I2.1 FOR VW225, +1, +2
3		End of inside loop Network 3 NEXT
4		End of outside loop Network 4 NEXT

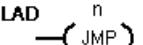
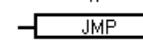
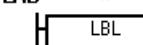
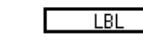
7.13.2 JMP (jump to label)

You can use the JMP (Jump) instruction in the main program, in subroutines, or in interrupt routines. The JMP and its corresponding LBL (Label) instruction must be located within the same program segment either in the main program, a subroutine, or an interrupt routine.

Note

You cannot jump from the main program to a label in either a subroutine or an interrupt routine. Likewise, you cannot jump from a subroutine or interrupt routine to a label outside that subroutine or interrupt routine.

You can use a Jump instruction within an SCR program segment, but the corresponding Label instruction must be located within the same SCR program segment.

LAD / FBD	STL	Description
 	JMP N	The JMP (Jump) instruction performs a branch to the label N within the program.
 	LBL N	The LBL (Label) instruction marks the location of the jump destination n.

Input / output	Data type	Operand
n	WORD	Constant (0 to 255)

Example: Jump to Label

LAD	STL
	Network 1 LDN SM0.2 JMP 4
	Network 2 LBL 4

7.13.3 SCR (sequence control relay)

SCR (Sequence Control Relay) instructions provide a simple yet powerful state control programming technique for a LAD, FBD, or STL program. Whenever an application consists of a sequence of operations that must be performed repetitively, you can use SCRs to structure the program so that it corresponds directly to your application. As a result, you can program and debug the application quickly and easily.



S bit usage in POU

Do not use the same S bit in more than one POU. For example, if you use S0.1 in the main program, do not use it in a subroutine.

Multiple POUs accessing the same S bit could result in unexpected process operation, possibly resulting in death or severe personal injury.

Check your program to ensure that multiple POU's do not access the same S bit.

Note

SCR programming restrictions

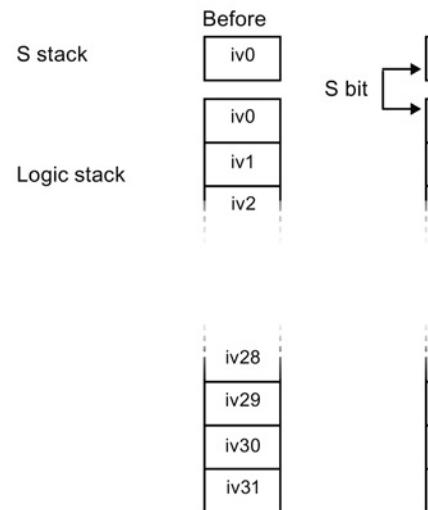
- You cannot jump into or out of an SCR segment; however, you can use Jump and Label instructions to jump around SCR segments or to jump within an SCR segment.
 - You cannot use the END instruction in an SCR segment.

LAD	FBD	STL	Description
		LSCR S_bit	<p>The SCR instruction loads the SCR and logic stacks with the value of the S_bit referenced by the instruction.</p> <p>The resulting value of the SCR stack either energizes or de-energizes the SCR stack. The value of the SCR stack is copied to the top of the logic stack so that LAD boxes or output coils can be tied directly to the left power rail and no preceding contact instruction is required.</p>
		SCRT S_bit	
		CSCRE	<p>The SCRT instruction identifies the SCR bit to be enabled (the next S_bit to be set). When power flows to the coil or FBD box, the CPU turns on the referenced S_bit and turns off the S_bit of the LSCR instruction (that enabled this SCR segment).</p>
		SCRE	<p>The CSCRE (conditional SCR end) instruction, for STL and FBD, terminates execution of the SCR segment when enabled. For LAD, a conditional contact placed before a SCRE coil performs the conditional SCR end function.</p> <p>The SCRE (unconditional SCR end) instruction, for STL and FBD, terminates execution of the SCR segment. For LAD, an SCRE coil connected directly to the power rail performs the unconditional SCR end function.</p>

Input / output	Data type	Operand
S_bit	BOOL	S

S stack and logic stack interaction

Load the value of Sx.y onto the SCR and load the value of Sx.y onto the logic stack. The figure shows the S stack and the logic stack and the effect of executing the Load SCR instruction.



Controlling program flow with SCR segments

The main program consists of instructions that execute sequentially once per scan of the PLC. For many applications, it may be appropriate to logically divide the main program into a series of operational steps that mirror steps within a controlled process (for example, a series of machine operations).

One way to logically divide a program into multiple steps is to use SCR segments. SCR segments can divide your program into a single stream of sequential steps, or into multiple streams that can be active simultaneously. It is possible to have a single stream conditionally diverge into multiple streams, and to have multiple streams conditionally re-converge into a single stream.

SCR operations

- SCR (Load SCR) marks the beginning of an SCR segment, and the SCRE (End SCR) marks the end of an SCR program segment. All logic between the SCR and the SCRE instructions is dependent upon the value of the S stack for its execution. Logic between SCRE and the next SCR instruction is not dependent on the value of the S stack.
- SCRT (SCR Transition) transfers control from an active SCR segment to another SCR segment.

Execution of the SCR transition instruction, when it has power flow, will reset the S bit of the currently active SCR segment and will set the S bit of the referenced segment.

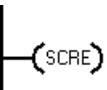
Resetting the S bit of the active segment does not affect the S stack at the time the SCR Transition instruction executes. Consequently, the SCR segment remains energized until it is exited.

- The STL only instruction CSRE (Conditional SCR End) exits an active SCR segment without executing the instructions between the CSRE and the SCRE (SCR End) instructions. The Conditional SCR End instruction does not affect any S bit nor does it affect the S stack.

Example: SCR sequential control flow

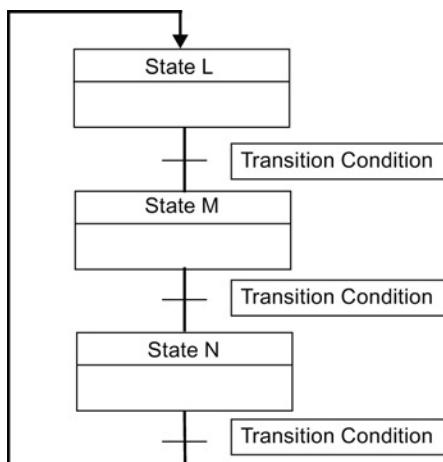
In the following sample program, the first scan bit SM0.1, is used to set S0.1, which will be the active State 1 on the first scan. After a 2-second delay, T37 causes a transition to State 2. This transition deactivates the State 1 SCR (S0.1) segment and activates the State 2 SCR (S0.2) segment.

LAD	STL
	Network 1 LD SM0.1 S S0.1, 1
	Network 2 LSCR S0.1
	Network 3 LD SM0.0 S Q0.4, 1 R Q0.5, 2 TON T37, +20
	Network 4 LD T37 SCRT S0.2
	Network 5 SCRE
	Network 6 LSCR S0.2
	Network 7 LD SM0.0 S Q0.2, 1 TON T38, +250
	Network 8 LD T38 SCRT S0.3

LAD	STL
	Network 9 SCRE

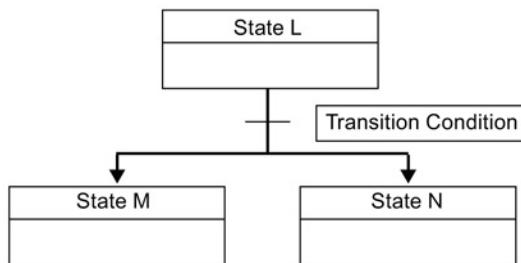
Sequential control flow

A process with a well-defined sequence of steps is easy to model with SCR segments. For example, consider a cyclical process, with 3 steps, that should return to the first step when the third has completed.



Divergent control flow

In many applications, a single stream of sequential states must be split into two or more different streams. When a control stream diverges into multiple streams, all outgoing streams must be activated simultaneously.



The divergence of control streams can be implemented in an SCR program by using multiple SCRT instructions enabled by the same transition condition, as shown in the following example.

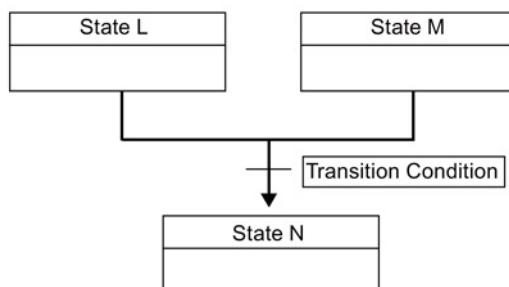
Example: SCR divergent flow control

LAD		STL
	Beginning of state L control region	Network 1 LSCR S3.4
	S3.5: Transition to state M S6.5: Transition to state N	Network 2 LD M2.3 A I2.1 SCRT S3.5 SCRT S6.5
	End of the state region for state L	Network 3 SCRE

Convergent flow control

When streams converge, all incoming streams must be complete before the next state is executed.

The convergence of control streams can be implemented in an SCR program by making the transition from state L to state L' and by making the transition from state M to state M'. When both SCR bits representing L' and M' are true, state N is enabled as shown in the following example.

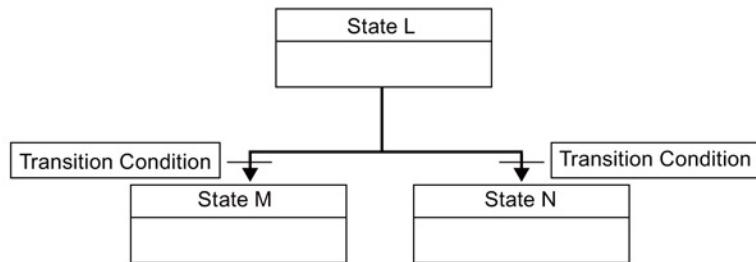


Example: SCR convergent flow control

LAD	STL
	Network 1 LSCR S3.4
	Network 2 LD V100.5 SCRT S3.5
	Network 3 SCRE
	Network 4 LSCR S6.4
	Network 5 LD C50 SCRT S6.5
	Network 6 SCRE
	Network 7 LD S3.5 A S6.5 S S5.0, 1 R S3.5, 1 R S6.5, 1

Divergence of a control stream, depending on transition conditions

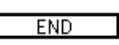
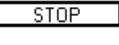
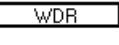
In other situations, a control stream might be directed into one of several possible control streams, depending upon which transition condition becomes true first.



Example: SCR divergent flow control, depending of transition conditions

LAD	STL
 S3.4 SCR	Network 1 LSCR S3.4
 M2.3 S3.5 (SCRT)	Network 2 LD M2.3 SCRT S3.5
 I3.3 S6.5 (SCRT)	Network 3 LD I3.3 SCRT S6.5
 (SCRE)	Network 4 SCRE

7.13.4 END, STOP, and WDR (watchdog timer reset)

LAD	FBD	STL	Description
		END	The conditional END instruction terminates the current scan based upon the condition of the preceding logic. You can use the conditional END instruction in the main program, but you cannot use it in either subroutines or interrupt routines.
		STOP	The conditional STOP instruction terminates the execution of your program by causing a transition of the CPU from RUN to STOP mode. If the STOP instruction is executed in an interrupt routine, the interrupt routine is terminated immediately and all pending interrupts are ignored. Remaining actions in the current scan cycle are completed, including execution of the main user program. The transition from RUN to STOP mode is made at the end of the current scan.
		WDR	The watchdog reset instruction retriggers the system watchdog timer and adds 500 milliseconds to the time allowed for the scan to complete before a watchdog timeout error occurs.

Watchdog timer operation

When the CPU is in RUN mode, the duration of the main scan is limited to 500 milliseconds, by default. If the duration of the main scan exceeds 500 milliseconds, then the CPU automatically transitions to STOP mode and the non-fatal error 001AH (Scan watchdog timeout) is issued.

You can extend the duration of the Main Scan by executing the Watchdog Reset (WDR) instruction in your program. The scan watchdog timeout period is reset to 500 milliseconds each time the WDR instruction is executed.

However, there is an absolute maximum main scan duration of 5 seconds. The CPU will unconditionally transition to STOP mode if the current scan duration reaches 5 seconds.

Example: STOP, END, and WDR (Watchdog reset) instructions

LAD	STL
	Network 1 LD SM5.0 STOP
	Network 1 LD SM5.6 WDR
	Network 1 LD I.0 END

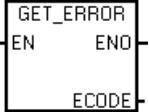
Note

If you expect your scan time to exceed 500 ms, or if you expect a burst of interrupt activity that prevents returning to the main scan for more than 500 ms, you should use the watchdog reset instruction to retrigger the watchdog timer.

Use the watchdog reset instruction carefully. If program execution loops prevent scan completion or excessively delay the completion of the scan, then the following processes are inhibited until the scan cycle is completed.

- Communications (except Freeport mode)
- I/O updating (except Immediate I/O)
- Forced values updating
- SM bit updating (SM0, SM5 to SM29 are not updated)
- RUN-time diagnostics
- STOP instruction, when used in an interrupt routine

7.13.5 GET_ERROR (Get non-fatal error code)

LAD / FBD	STL	Description
	GERR ECODE	The get non-fatal error code instruction stores the CPU's current non-fatal error code in the location assigned to ECODE. After the error code is stored, the non-fatal error code is cleared in the CPU.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
ECODE	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC

Non-fatal run-time errors also affect certain special memory error flag addresses that can be evaluated along with the GET_ERROR instruction to determine the cause of a run-time fault. In the event that the generic error flag SM4.3 = 1 (Run-time programming problem) is active, a GET_ERROR execution can be used to identify the specific error.

Non-fatal error code 0000H indicates that no actual error currently exists. In the case of a temporary run-time non-fatal error, a GET_ERROR (ECODE output) produces a non-zero error value and then the next program scan can produce a zero ECODE value.

You should use compare logic to save the ECODE value in another memory location. Your program can then test the saved error code value and begin a programmatic reaction.

Note

The error codes for the ECODE output are listed in the PLC non-fatal error codes table (see reference below). The error code values are in hexadecimal (16#xxxx).

See Also

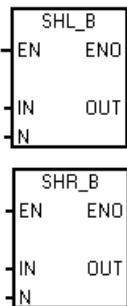
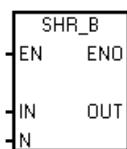
[PLC non-fatal error codes \(Page 587\)](#)

[PLC non-fatal error SM flags \(Page 590\)](#)

7.14 Shift and rotate

7.14.1 Shift and rotate

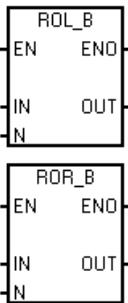
Shift instructions (only the byte size LAD box is illustrated, the others are similar)

LAD / FBD	STL	Shift type	Description
 	SLB OUT , N SRB OUT , N	Shift left byte Shift right byte	<p>The shift instructions shift the bit values of input value IN right or left by the bit position shift count N and load the result in the memory location assigned to OUT.</p> <p>The shift instructions fill empty bit positions with zero as each bit is shifted out. If the shift count N is greater than or equal to the maximum allowed (8 for byte operations, 16 for word operations, and 32 for double word operations), the value is shifted the maximum number of times for the operation. If the shift count is greater than 0, the overflow memory bit SM1.1 is set to the value of the last bit shifted out. The SM1.0 zero memory bit is set if the result of the shift operation is zero.</p> <p>Byte operations are unsigned. For word and double word operations, the sign bit is shifted when you use signed data values.</p>
SHL_W SHR_W	SLW OUT , N SRW OUT , N	Shift left word Shift right word	
SHL_DW SHR_DW	SLD OUT , N SRD OUT , N	Shift left double word Shift right double word	

Non-fatal errors with ENO=0	SM bits affected
• 0006H Indirect address	<ul style="list-style-type: none"> SM1.0 Result of operation = zero SM1.1 Overflow (last bit shifted out)

Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Rotate instructions (only the byte size LAD box is illustrated, the others are similar)

LAD / FBD	STL	Rotate type	Description
	RLB OUT, N RRB OUT, N	Rotate left byte Rotate right byte	<p>The rotate instructions rotate the bit values of input value IN right or left by the bit position rotate count N and load the result in the memory location assigned to OUT. The rotate operation is circular.</p> <p>If the rotate count is greater than or equal to the maximum for the operation (8 for a byte operation, 16 for a word operation, or 32 for a double-word operation), the CPU performs a modulo operation on the rotate count to obtain a valid shift count before the rotation is executed. This result is a shift count of 0 to 7 for byte operations, 0 to 15 for word operations, and 0 to 31 for double-word operations.</p> <p>If the rotate count is 0, a rotate operation is not performed. If the rotate operation is performed, the overflow bit SM1.1 is set to the value of the last bit rotated out.</p>
ROL_W ROR_W	RLW OUT, N RRW OUT, N	Rotate left word Rotate right word	<p>If the rotate count is not an integer multiple of 8 (for byte operations), 16 (for word operations), or 32 (for double-word operations), the last bit value rotated out is copied to the overflow memory bit SM1.1. The zero memory bit SM1.0 is set when the value to be rotated is zero.</p>
ROL_DW ROR_DW	RLD OUT, N RRD OUT, N	Rotate left double word Rotate right double word	<p>Byte operations are unsigned. For word and double word operations, the sign bit is rotated when you use signed data types.</p>

Non-fatal errors with ENO=0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address 	<ul style="list-style-type: none"> • SM1.0 Result of operation = zero • SM1.1 Overflow (last bit shifted out)

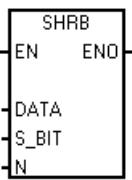
Input / output	Data type	Operand
IN	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC, Constant
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC
	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
	DWORD	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Example: Shift and Rotate instructions

LAD	STL																									
<pre> LD I4.0 ROR_W AC0, 2 AC0 OUT AC0 IN N 2 SHL_W VW200, 3 VW200 OUT VW200 IN N 3 </pre>	Network 1 LD I4.0 RRW AC0, 2 SLW VW200, 3																									
<p>Rotate right</p> <table border="0"> <tr> <td style="vertical-align: top;"> Before Rotate AC0 <table border="1"><tr><td>0100 0000 0000 0001</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> After first rotate AC0 <table border="1"><tr><td>1010 0000 0000 0000</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> After second rotate AC0 <table border="1"><tr><td>0101 0000 0000 0000</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> Zero Memory Bit (SM1.0) Overflow Memory Bit (SM1.1) </td> <td style="vertical-align: top; padding-left: 20px;"> = 0 = 0 </td> </tr> </table> <p>Shift left</p> <table border="0"> <tr> <td style="vertical-align: top;"> Before Shift VW200 <table border="1"><tr><td>1110 0010 1010 1101</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> After first shift VW200 <table border="1"><tr><td>1100 0101 0101 1010</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> After second shift VW200 <table border="1"><tr><td>1000 1010 1011 0100</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> After third shift VW200 <table border="1"><tr><td>0001 0101 0110 1000</td></tr></table> </td> <td style="vertical-align: top; padding-left: 20px;"> Overflow <input checked="checked" type="checkbox"/> </td> </tr> <tr> <td style="vertical-align: top;"> Zero Memory Bit (SM1.0) Overflow Memory Bit (SM1.1) </td> <td style="vertical-align: top; padding-left: 20px;"> = 0 = 1 </td> </tr> </table>	Before Rotate AC0 <table border="1"><tr><td>0100 0000 0000 0001</td></tr></table>	0100 0000 0000 0001	Overflow <input checked="checked" type="checkbox"/>	After first rotate AC0 <table border="1"><tr><td>1010 0000 0000 0000</td></tr></table>	1010 0000 0000 0000	Overflow <input checked="checked" type="checkbox"/>	After second rotate AC0 <table border="1"><tr><td>0101 0000 0000 0000</td></tr></table>	0101 0000 0000 0000	Overflow <input checked="checked" type="checkbox"/>	Zero Memory Bit (SM1.0) Overflow Memory Bit (SM1.1)	= 0 = 0	Before Shift VW200 <table border="1"><tr><td>1110 0010 1010 1101</td></tr></table>	1110 0010 1010 1101	Overflow <input checked="checked" type="checkbox"/>	After first shift VW200 <table border="1"><tr><td>1100 0101 0101 1010</td></tr></table>	1100 0101 0101 1010	Overflow <input checked="checked" type="checkbox"/>	After second shift VW200 <table border="1"><tr><td>1000 1010 1011 0100</td></tr></table>	1000 1010 1011 0100	Overflow <input checked="checked" type="checkbox"/>	After third shift VW200 <table border="1"><tr><td>0001 0101 0110 1000</td></tr></table>	0001 0101 0110 1000	Overflow <input checked="checked" type="checkbox"/>	Zero Memory Bit (SM1.0) Overflow Memory Bit (SM1.1)	= 0 = 1	
Before Rotate AC0 <table border="1"><tr><td>0100 0000 0000 0001</td></tr></table>	0100 0000 0000 0001	Overflow <input checked="checked" type="checkbox"/>																								
0100 0000 0000 0001																										
After first rotate AC0 <table border="1"><tr><td>1010 0000 0000 0000</td></tr></table>	1010 0000 0000 0000	Overflow <input checked="checked" type="checkbox"/>																								
1010 0000 0000 0000																										
After second rotate AC0 <table border="1"><tr><td>0101 0000 0000 0000</td></tr></table>	0101 0000 0000 0000	Overflow <input checked="checked" type="checkbox"/>																								
0101 0000 0000 0000																										
Zero Memory Bit (SM1.0) Overflow Memory Bit (SM1.1)	= 0 = 0																									
Before Shift VW200 <table border="1"><tr><td>1110 0010 1010 1101</td></tr></table>	1110 0010 1010 1101	Overflow <input checked="checked" type="checkbox"/>																								
1110 0010 1010 1101																										
After first shift VW200 <table border="1"><tr><td>1100 0101 0101 1010</td></tr></table>	1100 0101 0101 1010	Overflow <input checked="checked" type="checkbox"/>																								
1100 0101 0101 1010																										
After second shift VW200 <table border="1"><tr><td>1000 1010 1011 0100</td></tr></table>	1000 1010 1011 0100	Overflow <input checked="checked" type="checkbox"/>																								
1000 1010 1011 0100																										
After third shift VW200 <table border="1"><tr><td>0001 0101 0110 1000</td></tr></table>	0001 0101 0110 1000	Overflow <input checked="checked" type="checkbox"/>																								
0001 0101 0110 1000																										
Zero Memory Bit (SM1.0) Overflow Memory Bit (SM1.1)	= 0 = 1																									

7.14.2 Shift register bit

The Shift Register Bit instruction shifts a bit value into the Shift Register. This instruction provides an easy method for sequencing and controlling product flow or data. Use this instruction to shift the entire register one bit, once per scan.

LAD / FBD	STL	Description
	SHRB DATA, S_bit, N	<p>The shift register bit instruction shifts the bit value of DATA into the Shift Register. S_BIT assigns the location of the least significant bit of the shift register. N assigns the length of the Shift Register and the direction of the shift (Shift Plus = N, Shift Minus = -N).</p> <p>Each bit value shifted out by the SHRB instruction is copied to the overflow memory bit SM1.1.</p> <p>The shift register bits are defined by both the least significant bit S_BIT location and the number of bits assigned by the length N.</p>

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • 0092H Error in count field 	<ul style="list-style-type: none"> • SM1.1 Overflow (last bit shifted out)

Input / output	Data type	Operand
DATA, S_bit	BOOL	I, Q, V, M, SM, S, T, C, L
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

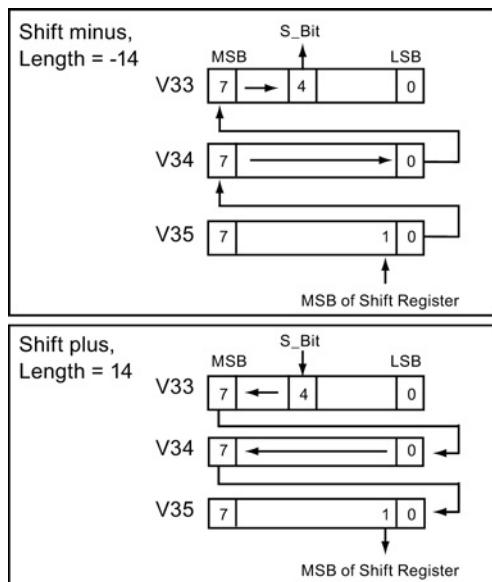
Use the following equation to compute the address of the most significant bit of the Shift Register (MSB.b):

$$\text{MSB.b} = \lceil (\text{Byte of S_BIT}) + ([N] - 1 + (\text{bit of S_BIT})) / 8 \rceil. [\text{remainder of the division by 8}]$$

For example: if S_BIT is V33.4 and N is 14, the following calculation shows that the MSB.b is V35.1.

$$\begin{aligned}
 \text{MSB.b} &= V33 + ([14] - 1 + 4) / 8 \\
 &= V33 + 17 / 8 \\
 &= V33 + 2 \text{ with a remainder of } 1 \\
 &= V35.1
 \end{aligned}$$

The following figure shows bit shifting for negative and positive values of N.



A Shift Minus operation is indicated by a negative value of length N. The input value of DATA shifts into the most significant bit of the shift register, and shifts out of the least significant bit location assigned by S_BIT. The data shifted out is then placed in the overflow memory bit SM1.1.

A Shift Plus operation is indicated by a positive value of length N. The input value of DATA shifts into the least significant bit location assigned by S_BIT and out of the most significant bit of the Shift Register. The bit value shifted out is then placed in the overflow memory bit SM1.1.

The maximum length of the shift register assigned by N is 64 bits (positive or negative).

Example: SHRB instruction

LAD	STL																																																								
<pre> LD I0.2 SHRB I0.3, V100.0, +4 </pre>	Network 1 LD I0.2 EU SHRB I0.3, V100.0, +4																																																								
<p>Timing Diagram</p> <table border="1"> <thead> <tr> <th></th> <th>7 MSB</th> <th>0 LSB</th> <th>S_BIT</th> </tr> </thead> <tbody> <tr> <td>Before first shift</td> <td>0 1 0 1</td> <td>X</td> <td>I0.3</td> </tr> <tr> <td>Overflow (SM1.1)</td> <td>1 0 1 1</td> <td>0</td> <td>I0.3</td> </tr> <tr> <td>After first shift</td> <td>1 0 1 1</td> <td>0</td> <td>I0.3</td> </tr> <tr> <td>Overflow (SM1.1)</td> <td>0 1 1 0</td> <td>1</td> <td>I0.3</td> </tr> <tr> <td>After second shift</td> <td>0 1 1 0</td> <td>1</td> <td>I0.3</td> </tr> <tr> <td>Overflow (SM1.1)</td> <td>1 0 1 1</td> <td>0</td> <td>I0.3</td> </tr> </tbody> </table>		7 MSB	0 LSB	S_BIT	Before first shift	0 1 0 1	X	I0.3	Overflow (SM1.1)	1 0 1 1	0	I0.3	After first shift	1 0 1 1	0	I0.3	Overflow (SM1.1)	0 1 1 0	1	I0.3	After second shift	0 1 1 0	1	I0.3	Overflow (SM1.1)	1 0 1 1	0	I0.3	<p>Timing Diagram</p> <table border="1"> <thead> <tr> <th></th> <th>7 MSB</th> <th>0 LSB</th> <th>S_BIT</th> </tr> </thead> <tbody> <tr> <td>Before first shift</td> <td>0 1 0 1</td> <td>X</td> <td>I0.3</td> </tr> <tr> <td>Overflow (SM1.1)</td> <td>1 0 1 1</td> <td>0</td> <td>I0.3</td> </tr> <tr> <td>After first shift</td> <td>1 0 1 1</td> <td>0</td> <td>I0.3</td> </tr> <tr> <td>Overflow (SM1.1)</td> <td>0 1 1 0</td> <td>1</td> <td>I0.3</td> </tr> <tr> <td>After second shift</td> <td>0 1 1 0</td> <td>1</td> <td>I0.3</td> </tr> <tr> <td>Overflow (SM1.1)</td> <td>1 0 1 1</td> <td>0</td> <td>I0.3</td> </tr> </tbody> </table>		7 MSB	0 LSB	S_BIT	Before first shift	0 1 0 1	X	I0.3	Overflow (SM1.1)	1 0 1 1	0	I0.3	After first shift	1 0 1 1	0	I0.3	Overflow (SM1.1)	0 1 1 0	1	I0.3	After second shift	0 1 1 0	1	I0.3	Overflow (SM1.1)	1 0 1 1	0	I0.3
	7 MSB	0 LSB	S_BIT																																																						
Before first shift	0 1 0 1	X	I0.3																																																						
Overflow (SM1.1)	1 0 1 1	0	I0.3																																																						
After first shift	1 0 1 1	0	I0.3																																																						
Overflow (SM1.1)	0 1 1 0	1	I0.3																																																						
After second shift	0 1 1 0	1	I0.3																																																						
Overflow (SM1.1)	1 0 1 1	0	I0.3																																																						
	7 MSB	0 LSB	S_BIT																																																						
Before first shift	0 1 0 1	X	I0.3																																																						
Overflow (SM1.1)	1 0 1 1	0	I0.3																																																						
After first shift	1 0 1 1	0	I0.3																																																						
Overflow (SM1.1)	0 1 1 0	1	I0.3																																																						
After second shift	0 1 1 0	1	I0.3																																																						
Overflow (SM1.1)	1 0 1 1	0	I0.3																																																						

7.15 String

7.15.1 String (Get length, copy, and concatenate)

SLEN (String length)

LAD / FBD	STL	Description
<pre> SLEN IN, OUT </pre>		<p>The string length instruction returns the length in bytes of the string specified by IN.</p> <p>Note: Because Chinese characters are not represented by a single byte, the STR_LEN function does not return the number of characters in a string containing Chinese characters.</p>

Error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> 0006H Indirect address 0091H Operand out of range 	None

Input / output	Data type	Operand
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC

SCPY and SCAT (String copy and string concatenate)

LAD / FBD	STL	Description
	<code>SCPY IN, OUT</code>	The copy string instruction copies the string assigned by IN to the string assigned by OUT.
	<code>SCAT IN, OUT</code>	The concatenate string instruction appends the string assigned by IN to the end of the string assigned by OUT.

Note: The STR_CPY and STR_CAT instructions operate on bytes and not characters. Because Chinese characters are not represented by a single byte, unexpected results can occur with the STR_CPY and STR_CAT instructions with strings containing Chinese characters. If you know the number of bytes that a character string occupies, you can use the STR_CPY and STR_CAT instructions with the correct number of bytes.

Error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range 	None

Inputs/Outputs	Data types	Operands
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
OUT	STRING	VB, LB, *VD, *LD, *AC

Example: Concatenate string, copy string, and string length Instructions

LAD	STL																																									
<pre> LD I0.0 SCAT "WORLD", VB0 SCPY VB0, VB100 SLEN VB100, AC0 </pre>	<ol style="list-style-type: none"> Append the string "WORLD" to the string at VB0. Copy the string at VB0 to a new string at VB100. Get the length of the string that starts at VB100. 																																									
<p>Before executing the program</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">VB0</td> <td style="text-align: center;">VB6</td> </tr> <tr> <td>6</td> <td>'H'</td> <td>'E'</td> <td>'L'</td> <td>'L'</td> <td>'O'</td> <td>' '</td> </tr> </table> <p>After executing the program</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">VB0</td> <td style="text-align: center;">VB11</td> </tr> <tr> <td>11</td> <td>'H'</td> <td>'E'</td> <td>'L'</td> <td>'L'</td> <td>'O'</td> <td>' '</td> <td>'W'</td> <td>'O'</td> <td>'R'</td> <td>'L'</td> <td>'D'</td> </tr> <tr> <td style="text-align: center;">VB100</td> <td style="text-align: center;">VB111</td> </tr> <tr> <td>11</td> <td>'H'</td> <td>'E'</td> <td>'L'</td> <td>'L'</td> <td>'O'</td> <td>' '</td> <td>'W'</td> <td>'O'</td> <td>'R'</td> <td>'L'</td> <td>'D'</td> </tr> <tr> <td style="text-align: center;">AC0</td> <td></td> </tr> <tr> <td>11</td> <td></td> </tr> </table>		VB0	VB6	6	'H'	'E'	'L'	'L'	'O'	' '	VB0	VB11	11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'	VB100	VB111	11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'	AC0		11	
VB0	VB6																																									
6	'H'	'E'	'L'	'L'	'O'	' '																																				
VB0	VB11																																									
11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'																															
VB100	VB111																																									
11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'																															
AC0																																										
11																																										

7.15.2 Copy substring from string

LAD / FBD	STL	Description
<pre> SSTR_CPY EN ENO IN OUT INDX N </pre>	SSCPY IN, INDX, N, OUT	<p>The copy substring from string instruction copies the assigned number of characters N from the string specified by IN, starting at the index INDX, to a new string assigned by OUT.</p> <p>Note: The SSTR_CPY instruction operates on bytes and not characters. Because Chinese characters are not represented by a single byte, unexpected results can occur with the SSTR_CPY instruction with strings containing Chinese characters. If you know the number of bytes that a character string occupies, you can use the SSTR_CPY instruction with the correct number of bytes.</p>

Non-fatal errors with ENO=0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • 009BH Index = 0 	None

Input / output	Data type	Operand
IN	STRING	VB, LB, *VD, *LD, *AC, Constant string
OUT	STRING	VB, LB, *VD, *LD, *AC
INDX, N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant

Example: Copy substring instruction

AD	STL
<p>Starting at the seventh byte after the byte count in the string at VB0, copy 5 bytes to a new string at VB20.</p>	Network 1 LD I0.0 SSCPY VB0, 7, 5, VB20

Before executing the program

VB0	11	'H'	'E'	'L'	'L'	'O'	' '	'W'	'O'	'R'	'L'	'D'
-----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

After executing the program

VB20	5	'W'	'O'	'R'	'L'	'D'
VB25						

7.15.3 Find string and first character within string

LAD / FBD	STL	Description
	SFND IN1, IN2, OUT	STR_FIND searches for the first occurrence of the string IN2 within the string IN1. The search begins at the starting position assigned by the initial value of OUT (which must be in the range of 1 to the IN1 string length before STR_FIND execution). If a sequence of characters is found that matches exactly the string IN2, the position of the first character in the sequence within the IN1 string is written to OUT. If the string IN2 was not found in the string IN1, then OUT is set to 0.
	CFND IN1, IN2, OUT	CHR_FIND searches the string IN1 for the first occurrence of any character from the character set in the string IN2. The search begins at starting position assigned by the initial value of OUT (which must be in the range of 1 to the IN1 string length before CHR_FIND execution). If a matching character is found, then the position of the character is written to OUT. If no matching character is found, OUT is set to 0.

Note: Because Chinese characters are not represented by a single byte, and the string instructions operate on bytes and not characters, unexpected results can occur with the STR_FIND and CHR_FIND instructions with strings containing Chinese characters.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • 009BH Index = 0 	None

Input / output	Data type	Operand
IN1, IN2	STRING	VB, LB, *VD, *LD, *AC, Constant String
OUT	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC

Example: Find string within string instruction

A string stored at VB0 is used as a command for turning a pump on or off. A string "On" is stored at VB20 and a string "Off" is stored at VB30. The result of the find string within string instruction is stored in AC0 (the OUT parameter). If the result is not 0, then the string 'On' was found in the command string (VB12).

LAD	STL																																										
	<p>1. Set AC0 to 1. (AC0 is used as the OUT parameter.)</p> <p>2. Search the string at VB0 for the string at VB20 ('On'), starting at the first position (AC0=1).</p> <pre> Network 1 LD I0.0 MOVB 1, AC0 SFND VB0, VB20, AC0 </pre>																																										
<table border="1"> <tr> <td style="text-align: center;">VB0</td> <td colspan="16" style="text-align: center;">VB12</td> </tr> <tr> <td>12</td> <td>'T'</td> <td>'u'</td> <td>'r'</td> <td>'n'</td> <td>' '</td> <td>'P'</td> <td>'u'</td> <td>'m'</td> <td>'p'</td> <td>' '</td> <td>'O'</td> <td>'n'</td> <td></td> <td></td> <td></td> <td></td> </tr> </table> <table border="1"> <tr> <td style="text-align: center;">VB20</td> <td style="text-align: center;">VB22</td> <td style="text-align: center;">VB30</td> <td style="text-align: center;">VB33</td> </tr> <tr> <td>2</td> <td>'O'</td> <td>'n'</td> <td></td> </tr> </table> <p>If the string in VB20 is found: AC0 <input type="text" value="11"/></p> <p>If the string in VB20 is not found: AC0 <input type="text" value="0"/></p>	VB0	VB12																12	'T'	'u'	'r'	'n'	' '	'P'	'u'	'm'	'p'	' '	'O'	'n'					VB20	VB22	VB30	VB33	2	'O'	'n'		
VB0	VB12																																										
12	'T'	'u'	'r'	'n'	' '	'P'	'u'	'm'	'p'	' '	'O'	'n'																															
VB20	VB22	VB30	VB33																																								
2	'O'	'n'																																									

Example: Find character within string instruction

A string stored at VB0 contains the temperature. The string constant at IN1 provides all the numeric characters (0-9, +, and -) that can identify a temperature number in a string.

CHR_FIND execution finds the starting position of the character "9" in the VB0 string and then S_R execution converts the real number characters into a real number value. VD200 is used to store the real-number value of the temperature.

LAD	STL								
<pre> LD I0.0 MOVB 1, AC0 CFND VB0, VB20, AC0 STR VB0, AC0, VD200 </pre>	<ol style="list-style-type: none"> Set AC0 to 1. (AC0 is used as the OUT parameter and points to the first character position in the string.) Find the first numeric character in the string stored at VB0. Convert the string to a real number value. 								
<table border="1"> <tr> <td style="text-align: center;">VB0</td> <td style="text-align: center;">VB11</td> </tr> <tr> <td>11 'T' 'e' 'm' 'p' '9' '8' '.' '6' 'F'</td> <td></td> </tr> </table> <table border="1"> <tr> <td style="text-align: center;">VB20</td> <td style="text-align: center;">VB32</td> </tr> <tr> <td>12 '1' '2' '3' '4' '5' '6' '7' '8' '9' '0' '+' '-'</td> <td></td> </tr> </table> <p>Starting position of the temperature stored in VB0: <input type="text" value="7"/> Real-number value of the temperature: <input type="text" value="98.6"/></p>	VB0	VB11	11 'T' 'e' 'm' 'p' '9' '8' '.' '6' 'F'		VB20	VB32	12 '1' '2' '3' '4' '5' '6' '7' '8' '9' '0' '+' '-'		
VB0	VB11								
11 'T' 'e' 'm' 'p' '9' '8' '.' '6' 'F'									
VB20	VB32								
12 '1' '2' '3' '4' '5' '6' '7' '8' '9' '0' '+' '-'									

7.16 Table

7.16.1 Add to table

LAD / FBD	STL	Description
	ATT DATA, TBL	<p>The add to table instruction adds word values DATA to a table TBL. The first value in the table is the maximum table length TL. The second value is the entry count EC, which stores the number of entries in the table and is updated automatically. New data are added to the table after the last entry. Each time new data are added to the table, the entry count is incremented.</p> <p>A table can have up to 100 data entries.</p>

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • SM1.4 Table overflow 	<ul style="list-style-type: none"> • SM1.4 Set to 1 if you try to overflow the table

Input / output	Data type	Operand
DATA	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
TBL	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, *VD, *LD, *AC

Note

To create a table, first make an entry for the maximum number of table entries. If you do not do this, then you cannot make any entries in the table.

Edge trigger instructions must activate all table read and table write instructions.

Program instructions

7.16 Table

Example: Add to Table instruction

LAD	STL																																		
	Network 1 LD SM0.1 MOVW +6, VW200																																		
	Network 2 LD IO.0 ATT VW100, VW200																																		
<p style="text-align: center;">Before execution of ATT</p> <table border="1"> <tr><td>VW100</td><td>1234</td></tr> <tr><td>VW200</td><td>0006</td></tr> <tr><td>VW202</td><td>0002</td></tr> <tr><td>VW204</td><td>5431</td></tr> <tr><td>VW206</td><td>8942</td></tr> <tr><td>VW208</td><td>xxxx</td></tr> <tr><td>VW210</td><td>xxxx</td></tr> <tr><td>VW212</td><td>xxxx</td></tr> <tr><td>VW214</td><td>xxxx</td></tr> </table> <p style="text-align: center;">TL (max. no. of entries) EC (entry count) d0 (data 0) d1 (data 1)</p>	VW100	1234	VW200	0006	VW202	0002	VW204	5431	VW206	8942	VW208	xxxx	VW210	xxxx	VW212	xxxx	VW214	xxxx	<p style="text-align: center;">After execution of ATT</p> <table border="1"> <tr><td>VW200</td><td>0006</td></tr> <tr><td>VW202</td><td>0003</td></tr> <tr><td>VW204</td><td>5431</td></tr> <tr><td>VW206</td><td>8942</td></tr> <tr><td>VW208</td><td>1234</td></tr> <tr><td>VW210</td><td>xxxx</td></tr> <tr><td>VW212</td><td>xxxx</td></tr> <tr><td>VW214</td><td>xxxx</td></tr> </table> <p style="text-align: center;">TL (max. no. of entries) EC (entry count) d0 (data 0) d1 (data 1) d2 (data 2)</p>	VW200	0006	VW202	0003	VW204	5431	VW206	8942	VW208	1234	VW210	xxxx	VW212	xxxx	VW214	xxxx
VW100	1234																																		
VW200	0006																																		
VW202	0002																																		
VW204	5431																																		
VW206	8942																																		
VW208	xxxx																																		
VW210	xxxx																																		
VW212	xxxx																																		
VW214	xxxx																																		
VW200	0006																																		
VW202	0003																																		
VW204	5431																																		
VW206	8942																																		
VW208	1234																																		
VW210	xxxx																																		
VW212	xxxx																																		
VW214	xxxx																																		

7.16.2 First-in-first-out and last-in-first-out

Table 7- 9 FIFO and LIFO instructions

LAD / FBD	STL	Description
	FIFO TBL, DATA	The first-in-first-out instruction moves the oldest (or first) entry in a table to an output memory address, by removing the first entry in the assigned table (TBL) and moving the value to the location assigned by DATA. All other entries of the table are shifted up one location. The entry count in the table is decremented for each FIFO execution.
	LIFO TBL, DATA	The last-in-first-out instruction moves the newest (or last) entry in the table to an output memory address, by removing the last entry in the table (TBL) and moving the value to the location assigned by DATA. The entry count in the table is decremented for each LIFO execution.

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range • SM1.5: Attempt to remove entry from empty table 	<ul style="list-style-type: none"> • SM1.5: Attempt to remove entry from empty table

Input / output	Data type	Operand
TBL	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, *VD, *LD, *AC
DATA	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AQW, *VD, *LD, *AC

Note

All table read and table write instructions must be activated by edge trigger instructions.

To create a table, you must first make an entry for the maximum number of table entries before any entries can be put in the table.

Example: FIFO instruction

LAD	STL																																
<pre>I4.1 +-- FIFO EN ENO VW200 TBL DATA VW400</pre>	Network 1 LD I4.1 FIFO VW200, VW400																																
<p>Before execution of FIFO</p> <table border="1"> <tr><td>VW200</td><td>0006</td></tr> <tr><td>VW202</td><td>0003</td></tr> <tr><td>VW204</td><td>5431</td></tr> <tr><td>VW206</td><td>8942</td></tr> <tr><td>VW208</td><td>1234</td></tr> <tr><td>VW210</td><td>xxxx</td></tr> <tr><td>VW212</td><td>xxxx</td></tr> <tr><td>VW214</td><td>xxxx</td></tr> </table> <p>TL (max. no. of entries) EC (entry count) d0 (data 0) —————→ VW400 d1 (data 1) d2 (data 2)</p>	VW200	0006	VW202	0003	VW204	5431	VW206	8942	VW208	1234	VW210	xxxx	VW212	xxxx	VW214	xxxx	<p>After execution of FIFO</p> <table border="1"> <tr><td>VW200</td><td>0006</td></tr> <tr><td>VW202</td><td>0002</td></tr> <tr><td>VW204</td><td>8942</td></tr> <tr><td>VW206</td><td>1234</td></tr> <tr><td>VW208</td><td>xxxx</td></tr> <tr><td>VW210</td><td>xxxx</td></tr> <tr><td>VW212</td><td>xxxx</td></tr> <tr><td>VW214</td><td>xxxx</td></tr> </table> <p>TL (max. no. of entries) EC (entry count) d0 (data 0) d1 (data 1)</p>	VW200	0006	VW202	0002	VW204	8942	VW206	1234	VW208	xxxx	VW210	xxxx	VW212	xxxx	VW214	xxxx
VW200	0006																																
VW202	0003																																
VW204	5431																																
VW206	8942																																
VW208	1234																																
VW210	xxxx																																
VW212	xxxx																																
VW214	xxxx																																
VW200	0006																																
VW202	0002																																
VW204	8942																																
VW206	1234																																
VW208	xxxx																																
VW210	xxxx																																
VW212	xxxx																																
VW214	xxxx																																

Example: LIFO instruction

LAD	STL																																
<pre>I0.1 +-- LIFO EN ENO VW200 TBL DATA VW300</pre>	Network 1 LD I0.1 LIFO VW200, VW300																																
<p>Before execution of LIFO</p> <table border="1"> <tr><td>VW200</td><td>0006</td></tr> <tr><td>VW202</td><td>0003</td></tr> <tr><td>VW204</td><td>5431</td></tr> <tr><td>VW206</td><td>8942</td></tr> <tr><td>VW208</td><td>1234</td></tr> <tr><td>VW210</td><td>xxxx</td></tr> <tr><td>VW212</td><td>xxxx</td></tr> <tr><td>VW214</td><td>xxxx</td></tr> </table> <p>TL (max. no. of entries) EC (entry count) d0 (data 0) —————→ VW300 d1 (data 1) d2 (data 2)</p>	VW200	0006	VW202	0003	VW204	5431	VW206	8942	VW208	1234	VW210	xxxx	VW212	xxxx	VW214	xxxx	<p>After execution of LIFO</p> <table border="1"> <tr><td>VW200</td><td>0006</td></tr> <tr><td>VW202</td><td>0002</td></tr> <tr><td>VW204</td><td>5431</td></tr> <tr><td>VW206</td><td>8942</td></tr> <tr><td>VW208</td><td>xxxx</td></tr> <tr><td>VW210</td><td>xxxx</td></tr> <tr><td>VW212</td><td>xxxx</td></tr> <tr><td>VW214</td><td>xxxx</td></tr> </table> <p>TL (max. no. of entries) EC (entry count) d0 (data 0) d1 (data 1)</p>	VW200	0006	VW202	0002	VW204	5431	VW206	8942	VW208	xxxx	VW210	xxxx	VW212	xxxx	VW214	xxxx
VW200	0006																																
VW202	0003																																
VW204	5431																																
VW206	8942																																
VW208	1234																																
VW210	xxxx																																
VW212	xxxx																																
VW214	xxxx																																
VW200	0006																																
VW202	0002																																
VW204	5431																																
VW206	8942																																
VW208	xxxx																																
VW210	xxxx																																
VW212	xxxx																																
VW214	xxxx																																

7.16.3 Memory fill

LAD / FBD	STL	Description
	FILL IN, OUT, N	<p>The memory fill instruction writes N consecutive words, beginning at address OUT, with the word value contained in address IN.</p> <p>N has a range of 1 to 255.</p>

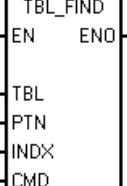
Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range 	None

Input / output	Data type	Operand
IN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
N	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *LD, *AC, Constant
OUT	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AQW, *VD, *LD, *AC

Example: Memory fill instruction

LAD	STL												
	Network 1 LD I2.1 FILL +0, VW200, 10												
<table style="width: 100%; text-align: center;"> <tr> <td style="width: 25%;">IN</td> <td style="width: 25%;">VW200</td> <td style="width: 25%;">VW202</td> <td style="width: 25%;">VW218</td> </tr> <tr> <td>0</td> <td>FILL</td> <td>0</td> <td>0</td> </tr> <tr> <td></td> <td></td> <td>-----</td> <td>0</td> </tr> </table>		IN	VW200	VW202	VW218	0	FILL	0	0			-----	0
IN	VW200	VW202	VW218										
0	FILL	0	0										
		-----	0										

7.16.4 Table find

LAD / FBD	STL	Description
	FND= TBL, PTN, INDX FND<> TBL, PTN, INDX FND< TBL, PTN, INDX FND> TBL, PTN, INDX	<p>The Table Find instruction searches a table for data that matches your search criteria. The Table Find instruction searches the table TBL, starting with the table entry INDX, for the data value or pattern PTN that matches the search criteria defined by CMD. The command parameter CMD is given a numeric value of 1 to 4 that corresponds to =, <>, <, and >, respectively.</p> <p>If a match is found, the INDX points to the matching entry in the table. To find the next matching entry, the INDX must be incremented before invoking the table find instruction again. If a match is not found, the INDX has a value equal to the entry count.</p> <p>A table can have up to 100 data entries. The data entries (area to be searched) are numbered from 0 to a maximum value of 99.</p>

Non-fatal errors with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 0091H Operand out of range 	None

Input / output	Data type	Operand
TBL	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, *VD, *LD, *AC
PTN	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant
INDX	WORD	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, *VD, *LD, *AC
CMD	BYTE	Constant: <ul style="list-style-type: none"> • 1 = Equal (=) • 2 = Not Equal (<>) • 3 = Less Than (<) • 4 = Greater Than (>)

Note

When you use the table find instruction with tables generated with the Add-to-table, Last-in-first-out, and First-in-first-out instructions, the entry count and the data entries correspond directly. The maximum-number-of-entries word required for the Add-to-table, Last-in-first-out, or First-in-first-out instructions is not required by the Table find instruction. See the following figure.

Consequently, you should set the TBL operand of a Find instruction to one-word address (two bytes) higher than the TBL operand of a corresponding the Add-to-table, Last-in-first-out, or First-in-first-out instruction.

Different table formats are used for the ATT, LIFO, and FIFO instructions compared to the Table Find instruction

Table format for ATT, LIFO, and FIFO

VW200	0006	TL (max. no. of entries)
VW202	0006	EC (entry count)
VW204	xxxx	d0 (data 0)
VW206	xxxx	d1 (data 1)
VW208	xxxx	d2 (data 2)
VW210	xxxx	d3 (data 3)
VW212	xxxx	d4 (data 4)
VW214	xxxx	d5 (data 5)

Table format for TBL_FIND

VW202	0006	EC (entry count)
VW204	xxxx	d0 (data 0)
VW206	xxxx	d1 (data 1)
VW208	xxxx	d2 (data 2)
VW210	xxxx	d3 (data 3)
VW212	xxxx	d4 (data 4)
VW214	xxxx	d5 (data 5)

Example: Table Find instruction

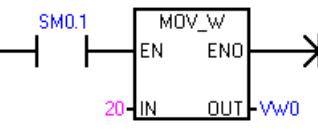
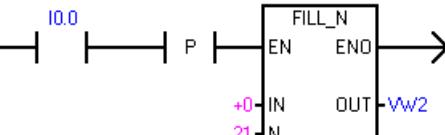
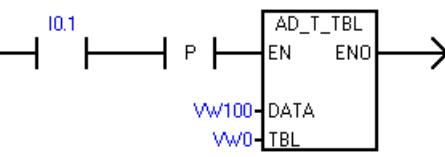
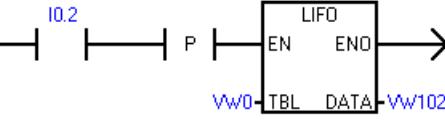
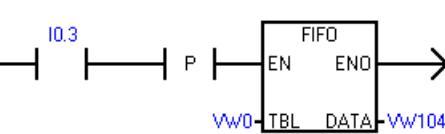
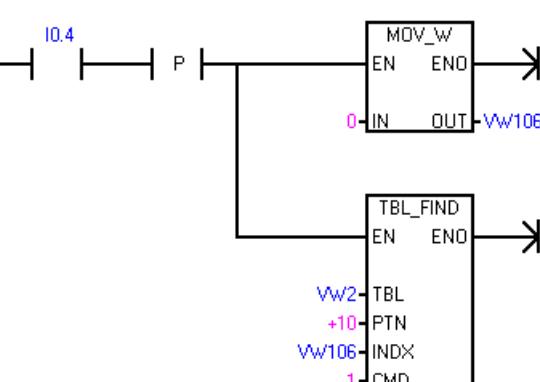
LAD	STL														
<pre> I2.1 VW202 16#3130 AC1 1 CMD </pre>	Network 1 LD I2.1 FND= VW202, 16#3130, AC1														
<p>When I2.1 is on, search the table for a value equal to 3130 HEX.</p> <table border="1"> <tr><td>VW202</td><td>0006</td></tr> <tr><td>VW204</td><td>3133</td></tr> <tr><td>VW206</td><td>4142</td></tr> <tr><td>VW208</td><td>3130</td></tr> <tr><td>VW210</td><td>3030</td></tr> <tr><td>VW212</td><td>3130</td></tr> <tr><td>VW214</td><td>4541</td></tr> </table> <p>If the table was created using ATT, LIFO, and FIFO instructions, VW200 contains the maximum number of allowed entries and is not required by the Find instructions.</p>	VW202	0006	VW204	3133	VW206	4142	VW208	3130	VW210	3030	VW212	3130	VW214	4541	<p>AC1 <input type="text" value="0"/> AC1 must be set to 0 to search from the top of table.</p> <p>Execute a table search AC1 <input type="text" value="2"/> AC1 contains the data entry number corresponding to the first match in the table (d2).</p> <p>AC1 <input type="text" value="3"/> Increment the INDEX by one, before searching the remaining entries of the table.</p> <p>Execute a table search AC1 <input type="text" value="4"/> AC1 contains the data entry number corresponding to the second match in the table (d4).</p> <p>AC1 <input type="text" value="5"/> Increment the INDEX by one, before searching the remaining entries of the table.</p> <p>Execute a table search AC1 <input type="text" value="6"/> AC1 contains a value equal to the entry count. The entire table has been searched without finding another match.</p> <p>AC1 <input type="text" value="0"/> Before the table can be searched again, the INDEX value must be reset to 0.</p>
VW202	0006														
VW204	3133														
VW206	4142														
VW208	3130														
VW210	3030														
VW212	3130														
VW214	4541														

Example: Table

The following program creates a table with 20 entries. The first memory location of the table contains the length of the table (in this case 20 entries). The second memory location shows the current number of table entries. The other locations contain the entries. A table can have up to 100 entries. It does not include the parameters defining the maximum length of the table or the actual number of entries (here VW0 and VW2). The actual number of entries in the table (here VW2) is automatically incremented or decremented by the CPU with every command.

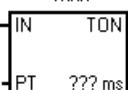
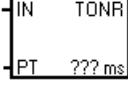
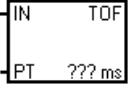
Before you work with a table, assign the maximum number of table entries. Otherwise, you cannot make entries in the table. Also, be sure that all read and write commands are activated with edge instructions.

To search the table, the index (VW106) must set to 0 before doing the find. If a match is found, the index will have the table entry number, but if no match is found, the index will match the current entry count for the table (VW2).

LAD		STL
	Create table with 20 entries starting with memory location 4. • On the first scan, define the maximum length of the table.	Network 1 LD SM0.1 MOVW +20, VW0
	Reset table with input I0.0. • On the rising edge of I0.0, fill memory locations from VW2 to VW2+20 with "+0".	Network 2 LD I0.0 EU FILL +0, VW2, 21
	Write value to table with input I0.1. • On the rising edge of I0.1, copy value of memory location VW100 to table.	Network 3 LD I0.1 EU ATT VW100, VW0
	Read last table value with input I0.2. • Move the last table value to location VW102. This reduces the number of entries. On the rising edge of I0.2, move last table value to VW102.	Network 4 LD I0.2 EU LIFO VW0, VW102
	Read first table value with input I0.3. • Move the first table value to location VW104. This reduces the number of entries. On the rising edge of I0.3, move first table value to VW104.	Network 5 LD I0.3 EU FIFO VW0, VW104
	Search table for the first location that has a value of 10. • On the rising edge of I0.4, reset index pointer. • Find a table entry that equals 10.	Network 6 LD I0.4 EU MOVW +0, VW106 FND= VW2, +10, VW106

7.17 Timer

7.17.1 Timer instructions

LAD / FBD	STL	Description
	TON Txxx, PT	Use TON On-delay timers for a timing a single time interval.
	TONR Txxx, PT	Use TONR On-delay retentive timers for accumulating the time value of many timed intervals.
	TOF Txxx, PT	Use the TOF Off-delay timer for extending a time interval past an OFF (or FALSE) condition, such as a delay time for cooling a motor.

Input / output	Data type	Operand
Txxx	WORD	Timer number (T0 to T255)
IN	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
PT	INT	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *LD, *AC, Constant

Timer resolution

TON, TONR, and TOF timers are available in three resolutions. The resolution is determined by the timer number, as shown below. Each unit of the current value is a multiple of the time base. For example, a count of 50 on a 10 ms timer represents 500 ms of elapsed time.

Your Txxx timer number assignment determines the resolution of the timer. When a valid timer number is assigned, the resolution is displayed in LAD or FBD timer boxes.

Timer number and resolution options

Timer type	Resolution	Maximum value	Timer number
TON, TOF	1 ms	32.767 s	T32, T96
	10 ms	327.67 s	T33-T36, T97-T100
	100 ms	3276.7 s	T37-T63, T101-T255
TONR	1 ms	32.767 s	T0, T64
	10 ms	327.67 s	T1-T4, T65-T68
	100 ms	3276.7 s	T5-T31, T69-T95

Note

Avoid timer number conflicts

You cannot use the same timer number for both a TON and TOF timer. For example, you cannot have both a TON T32 and a TOF T32.

Note

To guarantee a minimum time interval, increase the preset value (PV) by 1.

For example: To ensure a minimum timed interval of at least 2100 ms for a 100-ms timer, set the PV to 22.

TON and TONR timer operation

The TON and TONR instructions begin timing when the enabling input IN is ON. When the current value is equal to or greater than the preset time, the timer bit is set ON.

- The current value of a TON timer is cleared when the enabling input is OFF.
- The current value of the TONR timer is maintained when the enabling input is OFF. You can use the TONR timer to accumulate time when the input IN is ON. Use the Reset instruction (R) to clear the current value of the TONR.
- Both the TON and the TONR timers continue timing after the preset time is reached, and they stop timing at the maximum value of 32,767.

TOF timer operation

The TOF instruction is used to delay turning an output OFF for a fixed period of time after the input turns OFF. When the enabling input turns ON, the timer bit turns ON immediately, and the current value is set to 0. When the input turns OFF, timing begins and continues until the current time equals the preset time.

- When the preset is reached, the timer bit turns OFF and the current value stops incrementing; however, if the enabling input turns ON again before the TOF reaches the preset value, the timer bit remains ON.
- The enabling input must make an ON-OFF transition for a TOF timer to begin timing the OFF-delay time interval.
- If a TOF timer is inside an SCR region and the SCR region is inactive, then the current value is set to 0, the timer bit is turned OFF, and the current value does not increment.

Type	Current >= Preset	State of IN, the enabling input	Power cycle / first scan
TON	Timer bit ON Current value continues timing to 32,767	ON: Current value = timing value OFF: Timer bit OFF, current value = 0	Timer bit = OFF Current value = 0
TONR ¹	Timer bit ON Current value continues timing to 32,767	ON: Current value = timing value OFF: Timer bit and current value maintain last state and value	Timer bit = OFF Current value can be maintained ¹
TOF	Timer bit OFF Current = Preset, stops timing	ON: Timer bit ON, current value = 0 OFF: Timer begins timing after ON-to-OFF transition	Timer bit = OFF Current value = 0

¹ The retentive timer current value can be assigned for retention through a power cycle. See Configuring the retentive ranges for details (Page 114).

Note

Using the Reset instruction with timer instructions

The TONR timer can only be reset with the Reset (R) instruction.

The TON and TOF timers can be reset by the timer's enable input and also the Reset (R) instruction.

The Reset instruction performs the following actions:

- Timer bit = OFF
- Timer current value = 0
- After a reset, TOF timers require the enable input to make the transition from ON-to-OFF in order restart the OFF-delay timer.

7.17.2 Timer programming tips and examples

Timer types

You can use timers to implement time-based counting functions. The S7-200 instruction set provides three different types of timers.

- On-Delay Timer (TON) for timing a single interval
- Retentive On-Delay Timer (TONR) for accumulating a number of timed intervals
- Off-Delay Timer (TOF) for extending time past an off (or false condition), such as for cooling a motor after it is turned off

Addressing timer values

The meaning of the T number depends on the context in your program.

- "T37" assigned to a timer box identifies which timer is to use.
- "T37" assigned to normally open contacts addresses the Boolean T37 timer bit.
- "T37" assigned to integer operations addresses the T37 current time value, as a data word.

1-millisecond resolution

The 1-ms timers count the number of 1-ms timer intervals that have elapsed since the active 1-ms timer was enabled. The execution of the timer instruction starts the timing; however, the 1-ms timers are updated (timer bit and timer current) every millisecond asynchronous to the scan cycle. In other words, the timer bit and timer current are updated multiple times throughout any scan that is greater than 1 ms.

The timer instruction is used to turn the timer on, reset the timer, or, in the TONR timer, to turn the timer off.

Since the timer can be started anywhere within a millisecond, the preset must be set to one time interval greater than the minimum desired timer interval. For example, to guarantee a timed interval of at least 56 ms using a 1-ms timer, the preset time value should be set to 57.

10-millisecond resolution

The 10-ms timers count the number of 10-ms timer intervals that have elapsed since the active 10-ms timer was enabled. The execution of the timer instruction starts the timing; however the 10-ms timers are updated at the beginning of each scan cycle (in other words, the timer current and timer bit remain constant throughout the scan), by adding the accumulated number of 10-ms intervals (since the beginning of the previous scan) to the current value for the active timer.

Since the timer can be started anywhere within a 10-ms interval, the preset must be set to one time interval greater than the minimum desired timer interval. For example, to guarantee a timed interval of at least 140 ms using a 10-ms timer, the preset time value should be set to 15.

100-millisecond resolution

The 100-ms timers count the number of 100-ms timer intervals that have elapsed since the active 100-ms timer was last updated. These timers are updated by adding the accumulated number of 100-ms intervals (since the previous scan cycle) to the timer's current value when the timer instruction is executed.

The current value of a 100-ms timer is updated only if the timer instruction is executed. Consequently, if a 100-ms timer is enabled but the timer instruction is not executed each scan cycle, the current value for that timer is not updated and it loses time. Likewise, if the same 100-ms timer instruction is executed multiple times in a single scan cycle, the number of 100-ms intervals is added to the timer's current value multiple times, and it gains time. 100-ms timers should only be used where the timer instruction is executed exactly once per scan cycle.

Since the timer can be started anywhere within a 100-ms interval, the preset must be set to one time interval greater than the minimum desired timer interval. For example, to guarantee a timed interval of at least 2100 ms using a 100-ms timer, the preset time value should be set to 22.

TON On-delay timer example

LAD	STL
	Network 1 LD I0.0 TON T37, +10 <ul style="list-style-type: none"> • I0.0 ON = T37 enabled, • I0.0 OFF = disable and reset T37
	Network 2 LD T37 = Q0.0
Timing Diagram	

TON self-resetting On-delay timer example

LAD		STL
	10 ms timer T33 times out after 1 s (100 x 10 ms). M0.0 pulse is too fast to monitor with Status view.	Network 1 LDN M0.0 TON T33, +100
	The Compare contact becomes TRUE at a rate that is visible in Status view. Turn ON Q0.0 after (40 x 10 ms) for 40% OFF / 60% ON.	Network 2 LDW>= T33, +40 = Q0.0
	T33 (bit) pulse is too fast to monitor with Status view. Reset the timer with M0.0 after the (100 x 10 ms) period.	Network 3 LD T33 = M0.0
Timing Diagram		

TONR retentive On-delay timer example

LAD	STL
	<p>10 ms TONR timer T1 times out at PT = 1 s (100 x 10 ms).</p> <p>Network 1 LD I0.0 TONR T1, +100</p>
	<p>T1 bit is controlled by timer T1. Q0.0 is ON after the timer accumulates a total of 1 second.</p> <p>Network 2 LD T1 = Q0.0</p>
	<p>TONR timers must be reset by a Reset instruction with a T address. Reset timer T1 (current value and bit) when I0.1 is on.</p> <p>Network 3 LD I0.1 R T1, 1</p>
Timing Diagram	

TOF Off-delay timer example

LAD	STL
<p>I0.0 → IN → T33 +100 PT 10 ms</p>	<p>10-ms timer T33 times out after 1 s (100 x 10 ms).</p> <ul style="list-style-type: none"> • I0.0 ON-to-OFF = T33 enabled • I0.0 OFF-to-ON=disable and reset T33 <p>Network 1 LD I0.0 TOF T33, +100</p>
<p>T33 → Q0.0</p>	<p>Timer T33 controls Q0.0 through timer contact T33.</p> <p>Network 2 LD T33 = Q0.0</p>
<p>Timing Diagram</p> <p>I0.0</p> <p>current - 100</p> <p>T33 (current)</p> <p>T33 (bit) Q0.0</p> <p>1 s</p> <p>0.8 s</p>	

Timer resolution affects when timer bits and current time values are updated

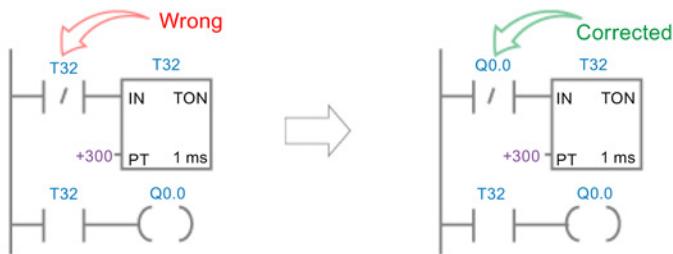
- **1 ms** timer: The timer bit and the current value are updated asynchronous to the scan cycle. For scans greater than 1 ms, the timer bit and the current value are updated multiple times throughout the scan.
- **10 ms** timer: The timer bit and the current value are updated at the beginning of each scan cycle. The timer bit and current value remain constant throughout the scan. Time intervals that accumulate during the scan are added to the current value at the start of each scan.
- **100 ms** timer: The timer bit and current value are updated when the instruction is executed; therefore, ensure that your program executes the instruction for a 100-ms timer only once per scan cycle in order for the timer to maintain the correct timing.

Example: automatically retriggered One-shot timers

The corrected examples use the normally closed contact Q0.0 instead of the timer bit as the timer enabling input. This ensures that output Q0.0 is turned ON for one scan cycle, each time a timer reaches the preset value.

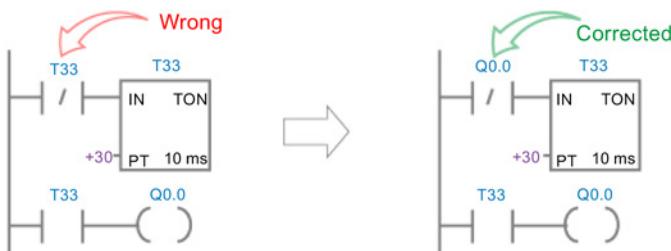
1 ms timer

Q0.0 is turned ON for one scan whenever the timer's current value is updated after the normally closed contact T32 is executed and before the normally open contact T32 is executed.



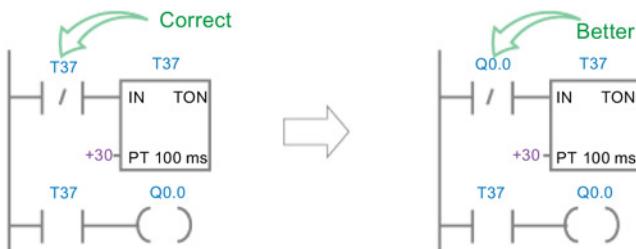
10 ms timer

Q0.0 is never turned ON, because the timer bit T33 is turned ON from the top of the scan to the point where the timer box is executed. Once the timer box has been executed, the timer's current value and its T-bit are set to zero. When the normally open contact T33 is executed, T33 is OFF and Q0.0 is turned OFF.



100 ms timer

Q0.0 is always turned ON for one scan whenever the timer's current value reaches the preset value.



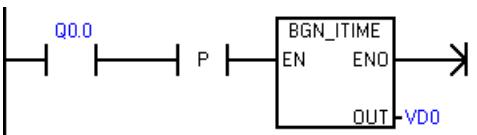
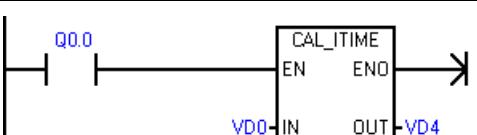
7.17.3 Interval timers

LAD / FBD	STL	Description
	BITIM OUT	The Begin interval time instruction reads the current value of the built-in 1 millisecond counter and stores the value in OUT. The maximum timed interval for a DWORD millisecond value is 2 raised to the 32 power or 49.7 days.
	CITIM IN, OUT	The Calculate interval time instruction calculates the time difference between the current time and the time provided at IN. The difference is stored in OUT. The maximum timed interval for a DWORD millisecond value is 2 raised to the 32 power or 49.7 days. CITIM automatically handles the one millisecond timer rollover that occurs within the maximum interval, depending on when the BITIM instruction was executed.

Non-fatal errors with ENO = 0	SM bits affected
• 0006H Indirect address	None

Input / output	Data type	Operand
IN	DWORD	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC
OUT	DWORD	VD, ID, QD, MD, SMD, SD, LD, AC, *VD, *LD, *AC

Example: Begin and Calculate interval timers

LAD	STL
 Ex1_Interval_time_net1	Network 1 LD Q0.0 EU BITIM VD0
	Network 2 LD Q0.0 CITIM VD0, VD4

7.18 Subroutine

7.18.1 CALL (subroutine) and RET (conditional return)

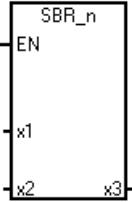
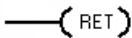
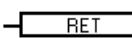
To add a new subroutine, select the **Edit** ribbon strip then **Insert Object** and **Subroutine** command. STEP 7-Micro/WIN SMART automatically adds an unconditional return from each subroutine. You can also add conditional return CRET instructions within the subroutine.

From the main program, you can nest subroutines (place a subroutine call within a subroutine) to a depth of eight.

From an interrupt routine, you can nest subroutines to a depth of four.

Note

Recursion (a subroutine that calls itself) is not prohibited, but you should use caution when using recursion with subroutines.

LAD / FBD	STL	Description
	CALL SBR_n, x1, x2, x3	<p>The Call subroutine instruction transfers control to subroutine SBR_n. You can use a Call subroutine instruction with or without parameters. After the subroutine completes its execution, control returns to the instruction that follows the Call subroutine.</p> <p>The call parameters x1 (IN), x2 (IN_OUT), and x3 (OUT) represent three call parameters passed in, in and out, or out of the subroutine. The call parameters are optional. You may use from 0 to 16 call parameters.</p> <p>When a subroutine is called, the entire logic stack is saved, the top of stack is set to one, all other stack locations are set to zero, and control is transferred to the called subroutine. When this subroutine is completed, the stack is restored with the values saved at the point of call, and control is returned to the calling routine.</p> <p>Accumulators are common to subroutines and the calling routine. No save or restore operation is performed on accumulators due to subroutine use.</p> <p>When a subroutine is called more than once in the same cycle, the edge up, edge down, timer and counter instructions should not be used.</p>
LAD:  FBD: 	CRET	<p>The Conditional Return from Subroutine instruction (CRET) terminates the subroutine based upon the preceding logic.</p>

Error conditions with ENO = 0	SM bits affected
<ul style="list-style-type: none"> • 0006H Indirect address • 008H Maximum subroutine nesting exceeded 	None

Input / output	Data type	Operand
SBR_n	WORD	Constant: 0-127
IN	BOOL	V, I, Q, M, SM, S, T, C, L, Power Flow (LAD), Logic flow (FBD)
	BYTE	VB, IB, QB, MB, SMB, SB, LB, AC, *VD, *LD, *AC ¹ , Constant
	WORD, INT	VW, T, C, IW, QW, MW, SMW, SW, LW, AC, AIW, *VD, *LD, *AC ¹ , Constant
	DWORD, DINT	VD, ID, QD, MD, SMD, SD, LD, AC, HC, *VD, *LD, *AC ¹ , &VB, &IB, &QB, &MB, &T, &C, &SB, &AI, &AQ, &SMB, Constant
	STRING	*VD, *LD, *AC ¹ , Constant
IN_OUT	BOOL	V, I, Q, M, SM ² , S, T, C, L
	BYTE	VB, IB, QB, MB, SMB ² , SB, LB, AC, *VD, *LD, *AC ¹
	WORD, INT	VW, T, C, IW, QW, MW, SMW ² , SW, LW, AC, *VD, *LD, *AC ¹
	DWORD, DINT	VD, ID, QD, MD, SMD ² , SD, LD, AC, *VD, *LD, *AC ¹
OUT	BOOL	V, I, Q, M, SM ² , S, T, C, L
	BYTE	VB, IB, QB, MB, SMB ² , SB, LB, AC, *VD, *LD, *AC ¹
	WORD, INT	VW, T, C, IW, QW, MW, SMW ² , SW, LW, AC, AQW, *VD, *LD, *AC ¹
	DWORD, DINT	VD, ID, QD, MD, SMD ² , SD, LD, AC, *VD, *LD, *AC ¹

¹ Only AC1, AC2 or AC3 (AC0 not allowed)

² Must be from byte offset 30 to byte offset 999 for read/write access

Calling a subroutine with call parameters

Subroutines have the option of using passed parameters. The parameters are defined in the variable table of the subroutine. Each parameter must be assigned a local symbol name (a maximum of 23 characters), a variable type, and a data type. A maximum of sixteen parameters can be passed to or from a subroutine. The VAR_Type type field in the variable table defines whether the variable is passed into the subroutine (IN), passed into and out of the subroutine (IN_OUT), or passed out of the subroutine (OUT).

To add a new parameter row, place the cursor on the Var_Type field of the type (IN, IN_OUT, OUT, or TEMP) that you want to add. Click the right mouse button to get a menu of options. Select the Insert option and then the Row Below option. Another parameter row of the selected type will appear below the current entry.

Temporary (TEMP) parameters can be assigned in the variable table to store data that is valid only within the scope of the subroutine execution. Local TEMP data is not passed as a call parameter. You can also assign TEMP parameters in the main routine and interrupt routines, but only subroutines can use IN, IN_OUT, and OUT call parameters.

Variable table parameter types for a subroutine

Parameter	Description
IN	Parameters are passed into the subroutine. If the parameter is a direct address (such as VB10), the value at the specified location is passed into the subroutine. If the parameter is an indirect address (such as *AC1), the value at the location pointed to is passed into the subroutine. If the parameter is a data constant (16#1234) or an address (&VB100), the constant or address value is passed into the subroutine.
IN_OUT	The value at the specified parameter location is passed into the subroutine, and the result value from the subroutine is returned to the same location. Constants (such as 16#1234) and addresses (such as &VB100) are not allowed for input/output parameters.
OUT	The result value from the subroutine is returned to the specified parameter location. Constants (such as 16#1234) and addresses (such as &VB100) are not allowed as output parameters. Since output parameters do not retain the value assigned by the last execution of the subroutine, you must assign values to outputs each time the subroutine is called.
TEMP	Any local memory that is not used for passed parameters can be used for temporary storage within the subroutine.

Data types allowed for call parameters

- Power Flow: Boolean power flow is allowed only for bit (Boolean) inputs. This declaration assigns an input parameter to the result of power flow based on a combination of bit logic instructions. Power flow inputs are similar to the EN input in that they connect to bit logic (for ex. LAD contacts) and not to a direct/indirect address assignment. Boolean power flow input(s) must be assigned in the top row(s) of the variable table before any non-BOOL data type assignment. Only input parameters are allowed to be used this way. The enable input (EN) and the IN1 inputs in the following example use power flow logic.
- BOOL: This data type is used for single bit inputs and outputs. IN3 in the following example is a Boolean input assigned to a direct address.
- BYTE, WORD, DWORD: These data types identify an unsigned input or output parameter of 1, 2, or 4 bytes, respectively.
- INT, DINT: These data types identify signed input or output parameters of 2 or 4 bytes, respectively.
- REAL: This data type identifies a single precision (4 byte) IEEE floating-point value.
- STRING: This data type is used as a four-byte pointer to a string.

Example variable table

Variable Table					
	Address	Symbol	Var Type	Data Type	Comment
1		EN	IN	BOOL	
2	LW0	Lvl	IN	INT	Tank transmitter
3	LW2	HSet	IN	INT	High setpoint
4	LW4	Offset	IN	INT	Offset
5			IN		
6			IN_OUT		
7	LW6	Result	OUT	INT	Result value
8	L8.0	Valve	OUT	BOOL	Control valve
9			OUT		

Example: Subroutine call with call parameters

LAD	STL
<pre> LD I0.0 CALL SBR_0, I0.1, VB10, I1.0, &VB100, *AC1, VD200 </pre> <p>To display correctly in LAD and FBD:</p> <pre> Network 1 LD I0.0 = L60.0 LD I0.1 = L63.7 LD L60.0 CALL SBR_0, L63.7, VB10, I1.0, &VB100, *AC1, VD200 </pre>	<p>STL only:</p> <p>Network 1</p> <pre> LD I0.0 CALL SBR_0, I0.1, VB10, I1.0, &VB100, *AC1, VD200 </pre> <p>To display correctly in LAD and FBD:</p> <p>Network 1</p> <pre> LD I0.0 = L60.0 LD I0.1 = L63.7 LD L60.0 CALL SBR_0, L63.7, VB10, I1.0, &VB100, *AC1, VD200 </pre>

Note

There are two STL examples provided above. STL programmers can use the first simplified STL instructions which can be only be displayed in the STL editor. This is because the BOOL parameters used as LAD/FBD power flow inputs are not saved to L memory.

The second set of compiler generated STL instructions can be displayed in the LAD, FBD, and STL editors, because L memory is used by the program compiler to save the state of the BOOL input parameters that are assigned as power flow inputs in LAD/FBD.

Address parameters such as IN4 (&VB100) are passed into a subroutine as a DWORD (unsigned double word) value. The type of a constant parameter must be specified for the parameter in the calling routine with a constant descriptor in front of the constant value. For example, to pass an unsigned double word constant with a value of 12,345 as a parameter, the constant parameter must be specified as DW#12345. If the constant descriptor is omitted from the parameter, the constant can be assumed to be a different type.

There are no automatic data type conversions performed on the input or output parameters. For example, if the variable table specifies that a parameter has the data type REAL, and in the calling routine a double word (DWORD) is specified for that parameter, the value in the subroutine will be a double word.

When values are passed to a subroutine, they are placed into the local memory of the subroutine. The left-most column of the variable table shows the local memory address for each passed parameter. Input parameter values are copied to the subroutine's local memory when the subroutine is called. Output parameter values are copied from the subroutine's local memory to the specified output parameter addresses when the subroutine execution is complete.

The data element size and type are represented in the coding of the parameters.

Assignment of parameter values to local memory in the subroutine is as follows:

- Parameter values are assigned to local memory in the order specified by the call subroutine instruction with parameters starting at L 0.0.
- One to eight consecutive bit parameter values are assigned to a single byte starting with Lx.0 and continuing to Lx.7.
- Byte, word, and double word values are assigned to local memory on byte boundaries (LBx, LWx, or LDx).

In the Call Subroutine instruction with parameters, parameters must be arranged in order with input parameters first, followed by input/output parameters, and then followed by output parameters.

If you are programming in STL, the format of the CALL instruction is:

CALL subroutine number, parameter 1, parameter 2, ... , parameter 16

Example: Subroutine and return from subroutine instructions

LAD	STL
MAIN SMO.1 SBR_0 EN	Network 1 LD SMO.1 CALL SBR_0
SBR0 M14.3 (RET)	You can use a conditional return to leave the subroutine before the last network. Network 1 LD M14.3 CRET
SBR0 SMO.0 MOV_B 10 IN EN OUT VBO	This network will be skipped if M14.3 is ON. Network 2 LD SMO.0 MOVE 10, VB0

Example: Subroutine call using string parameter

This example copies a different string literal to a unique address depending upon the given input. The unique address of this string is saved. The string address is then passed to the subroutine by using an indirect address. The data type of the subroutine input parameter is string. The subroutine then moves the string to a different location.

A string literal can also be passed to the subroutine. The string reference inside the subroutine is always the same.

LAD	STL
MAIN	Network 1 LD I0.0 SCPY "string1", VB100 AENO MOVD &VB100, VD0
MAIN	Network2 LD I0.1 SCPY "string2", VB200 AENO MOVD &VB200, VD0
MAIN	Network3 LD I0.2 CALL SBR_0, *VD0
MAIN	Network4 LD I0.3 CALL SBR_0, "string3"
SBR0	Network 1 LD SM0.0 SSCPY *LD0, VB300

Communication

The S7-200 SMART offers several types of communication between CPUs, programming devices, and HMIs:

- Ethernet:
 - Exchange of data from the programming device to the CPU
 - Exchange of data between HMIs and the CPU
 - S7 peer-to-peer communication with other S7-200 SMART CPUs
- RS485:
 - Supports a total of 126 addressable devices (32 devices per network segment)
 - Supports PPI (point-to-point interface) protocol
 - Exchange of data between HMIs and the CPU
 - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)
- RS232:
 - Supports a point-to-point connection to one device
 - Supports PPI protocol
 - Exchange of data between HMIs and the CPU
 - Exchange of data between devices and the CPU using Freeport (XMT/RCV instructions)

8.1 CPU communication connections

The CPU supports the following maximum number of simultaneous, asynchronous communication connections:

- Ethernet programming port: The CPU provides the following connections:
 - HMI/OPC connections: Eight dedicated HMI/OPC connections.
 - PG connections: One programming device (PG) connection.
 - Peer-to-peer (GET/PUT) connections: Eight connections to support S7-200 SMART CPUs or network devices.

Note

The S7-200 SMART CPU uses the GET and PUT instructions for CPU-to-CPU communication.

- Integrated RS485 port (Port 0): Four connections to support HMI devices.
 - CM01 Signal Board (SB) RS232/RS485 port (Port 1): Four connections to support HMI devices.
-

Note

STEP 7-Micro/WIN SMART can only connect to an S7-200 SMART CPU through the Ethernet port. Only one PG can monitor one CPU at a time.

The RS485 and RS232 ports are only for HMI access (Data read/write) and Freeport communications.

8.2

CPU communication ports

There are three communication interfaces on the S7-200 SMART CPU that provide the following communication types:

- Ethernet port:
 - STEP 7-Micro/WIN SMART programming
 - GET/PUT communication
 - HMIs - Ethernet type
- RS485 port (Port 0):
 - TDs/HMIs - RS485 type
 - Freeport (XMT/RCV) including Siemens-provided USS and Modbus libraries
- RS485/RS232 signal board (SB) (if present, Port 1):
 - TDs/HMIs- RS485 or RS232 type
 - Freeport (XMT/RCV) including Siemens-provided USS (RS485 only) and Modbus (RS485 or RS432) libraries

8.3 HMIs and communication drivers

HMIs

The S7-200 SMART supports the HMIs from the following Siemens HMI families:

- **COMFORT HMIs:**
 - SIMATIC HMI TP700 COMFORT
 - SIMATIC HMI TP900 COMFORT
 - SIMATIC HMI TP1200 COMFORT
 - SIMATIC HMI KP400 COMFORT
 - SIMATIC HMI KP700 COMFORT
 - SIMATIC HMI KP900 COMFORT
 - SIMATIC HMI KP1200 COMFORT
 - SIMATIC HMI KTP400 COMFORT
- **SMART HMIs:**
 - SMART 700 IE
 - SMART 1000 IE
- **BASIC HMIs:**
 - SIMATIC HMI KTP400 BASIC MONO PN
 - SIMATIC HMI KTP600 BASIC MONO PN
 - SIMATIC HMI KTP600 BASIC COLOR DP
 - SIMATIC HMI KTP600 BASIC COLOR PN
 - SIMATIC HMI KTP1000 BASIC COLOR DP
 - SIMATIC HMI KTP1000 BASIC COLOR PN
 - SIMATIC HMI TP1500 BASIC COLOR PN
 - SIMATIC HMI KP300 BASIC MONO PN
- **Micro HMIs:**
 - TD 400C TEXT DISPLAY, 4 LINES

Communication drivers

Communication drivers for your S7-200 SMART HMIs can be selected in two locations:

- WinCC Flexible software
- TIA portal

WinCC Flexible

In the WinCC Flexible software package, you can select the required "Communication driver" with the following menu selections:

- Communications
- Connections table

In the "Connections table", select the "SIMATIC S7 200 SMART" driver. If the SMART driver is not in the list, select the "SIMATIC S7 200" driver.

TIA portal

In the TIA portal, you can select the required "Communication driver" with the following menu selections:

- HMI tags
- Connections

In "Connections", select the "SIMATIC S7 200" driver.

Note

If the HMI panel is using a DP connection (RS485), then also set the "Network Profile" to PPI.

8.4 Ethernet

8.4.1 Overview

An Ethernet network is a differential (multi-point) network that can have up to 32 segments and 1,024 nodes. Ethernet allows for data transfer at a high speed (up to 100 Mbit/s) and long distances (Copper: Maximum approximately 1.5km; Optical: Maximum approximately 4.3km).

Possible Ethernet connections include connections for the following:

- Programming devices
- CPU-to-CPU GET/PUT communication
- HMI displays

8.4.2 TCP/IP protocol

TCP/IP Ethernet allows an S7-200 SMART CPU to be linked to an Industrial Ethernet network.

The Industrial Ethernet network includes the following features:

- Communication based upon TCP/IP communication standards
- Factory installed MAC address
- Auto sensory full duplex or half duplex communications, 10 Mbits and 100 Mbits
- Multiple connections:
 - Up to eight HMI connections
 - One programmer connection
 - Up to eight GET/PUT active connections
 - Up to eight GET/PUT passive connections

8.4.3 Local/partner connection

A Local / Partner (remote) connection defines a logical assignment of two communication partners to establish a communication connection. A connection is defined by the following:

- Communication partners involved (one active, one passive)
- Type of connection (programming device, HMI, CPU, or other device)
- Connection path (network, IP address, subnet mask, gateway)

The communication partners set up and establish the communication connection. The active device establishes the connection, and the passive device either accepts or rejects the connection request from the active device. After a connection is established, it is automatically maintained by the active device and monitored by both the devices.

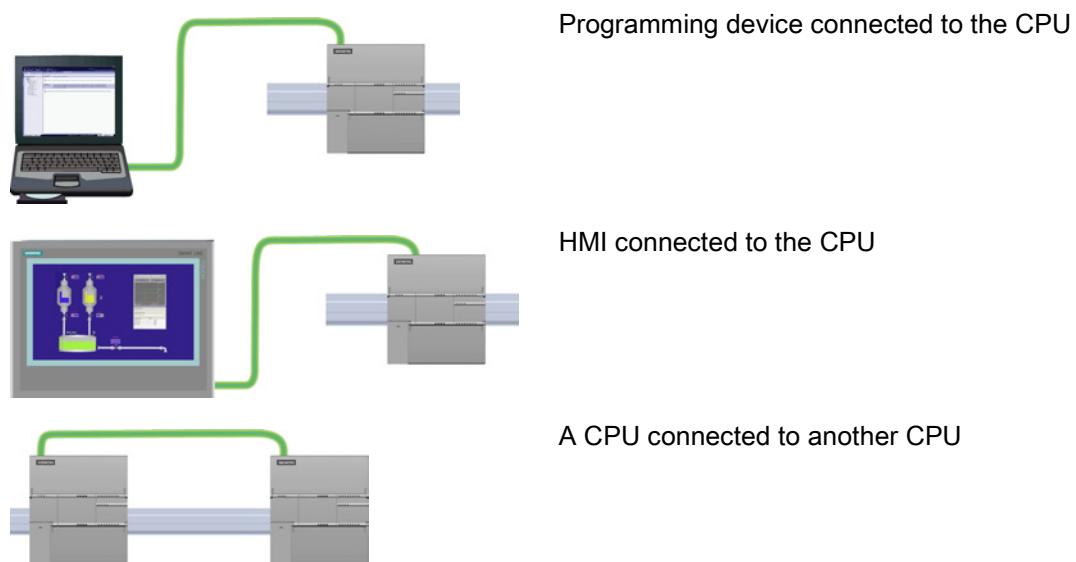
If the connection is terminated (for example, due to a line break or one of the partners breaks the connection), the active partner attempts to re-establish the connection. The passive device will also note the termination of the connection and take actions (for example, revoking the password privileges of the now disconnected active partner).

The S7-200 SMART CPUs are both active and passive devices. When an active device (for example, a computer running STEP 7-MicroWIN SMART or an HMI) establishes a connection, the CPU decides whether to accept or reject the connection request, based upon the type of the connection and how many connections of a given type are allowed.

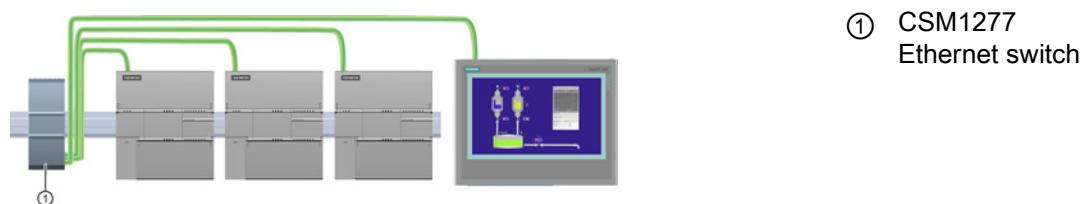
8.4.4 Sample Ethernet network configurations

You have three different types of communication options when using the S7-200 SMART CPU Ethernet network:

- Connecting a CPU to a programming device
- Connecting a CPU to an HMI
- Connecting a CPU to another S7-200 SMART CPU



The Ethernet port on the CPU does not contain an Ethernet switching device. A direct connection between a programming device or HMI and a CPU does not require an Ethernet switch. However, a network with more than two CPUs or HMI devices requires an Ethernet switch.



You can use the rack-mounted CSM1277 4-port Ethernet switch for connecting multiple CPUs and HMI devices.

8.4.5 Assigning Internet Protocol (IP) addresses

8.4.5.1 Assigning IP addresses to programming and network devices

If your programming device is using an on-board adapter card connected to your plant LAN (and possibly the world-wide web), both the programming device and the CPU must exist on the same subnet. The subnet is specified as a combination of the IP Address and subnet mask for the device. Please see your local network administrator for help.

The Network ID is the first part of the IP address (first three octets) (for example, **211.154.184.16**) that determines what IP network you are on. The subnet mask normally has a value of **255.255.255.0**; however, since your computer is on a plant LAN, the subnet mask may have various values (for example, **255.255.254.0**) in order to set up unique subnets. The subnet mask, when combined with the device IP address in a logical AND operation, defines the boundaries of an IP subnet.

Note

In a World Wide Web scenario, where your programming devices, network devices, and IP routers will communicate with the world, unique IP addresses must be assigned to avoid conflict with other network users. Contact your company IT department personnel, who are familiar with your plant networks, for assignment of your IP addresses.

Note

A secondary network adapter card is useful when you do not want your CPU on your company LAN. During initial testing or commissioning tests, this arrangement is particularly useful.

Assigning or checking the IP address of your programming device using "My Network Places" (on your desktop)

If you are using Windows 7, you can assign or check your programming device's IP address with the following menu selections:

- "Start"
- "Control Panel"
- "Network and Sharing Center"
- "Local Area Connection" for the network adapter connected to your CPU
- "Properties"

- In the "Local Area Connection Properties" dialog, in the "This connection uses the following items:" field:
 - Scroll down to "Internet Protocol Version 4 (TCP/IPv4)".
 - Click "Internet Protocol Version 4 (TCP/IPv4)".
 - Click the "Properties" button.
 - Select "Obtain an IP address automatically (DCP)" or "Use the following IP address" (to enter a static IP address).
- If you have the "Obtain an IP address automatically" selected you may want to change the selection to "Use the following IP address" to connect to the S7-200 SMART CPU:
 - Select an IP address on the same subnet as the CPU (**192.168.2.1**).
 - Set the IP address to an address with the same Network ID (for example, **192.168.2.200**).
 - Select a subnet mask of **255.255.255.0**.
 - Leave the default gateway blank.
 - This will allow you to connect to the CPU.

Note

The Network Interface Card and the CPU must be on the same subnet to allow STEP 7-MicroWIN SMART to find and communicate to the CPU.

Consult your IT personnel to help you set up a network configuration to allow you to connect to the S7-200 SMART CPU.

8.4.5.2 Configuring or changing an IP address for a CPU or device in your project

You must enter the following IP information for each S7-200 SMART CPU that is attached to your Ethernet network:

- IP address: Each CPU or device must have an Internet Protocol (IP) address. The CPU or device uses this address to deliver data on a more complex, routed network. Each IP address is divided into four 8-bit segments and is expressed in a dotted, decimal format (for example, 211.154.184.16). The first part of the IP address is used for the Network ID (What network are you on?), and the second part of the address is for the Host ID (unique for each device on the network). An IP address of 192.168.x.y is a standard designation recognized as part of a private network that is not routed on the Internet.

Note

All S7-200 SMART CPUs have a default IP address of: 192.168.2.1

Note

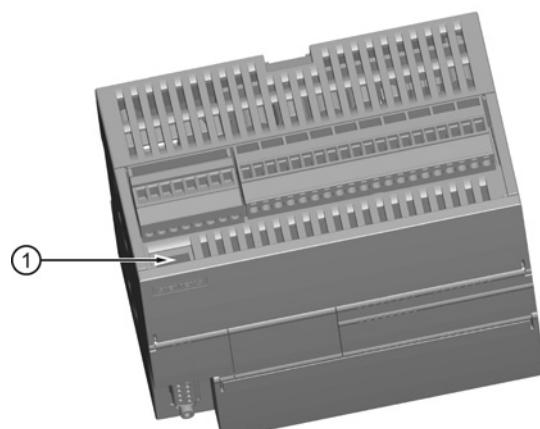
You must have a unique IP address for each device on your network.

- Subnet mask: A subnet is a logical grouping of connected network devices. Nodes on a subnet are usually located in close physical proximity to each other on a Local Area Network (LAN). The subnet mask defines the boundaries of an IP subnet.

Note

A subnet mask of 255.255.255.0 is generally suitable for a local network.

- Default gateway IP address: Gateways (or IP routers) are the link between LANs. Using a gateway, a computer in a LAN can send messages to other networks, which might have other LANs behind them. If the data destination is not within the LAN, the gateway forwards the data to another network or group of networks where it can be delivered to its destination. Gateways rely on IP addresses to deliver and receive data packets.



There are three ways to configure or change the IP information for the onboard Ethernet port of a CPU or device:

- Configuring the IP information in the "Communications" dialog (dynamic IP information)
- Configuring the IP information in the "System Block" dialog (static IP information)
- Configuring the IP information in the user program (dynamic IP information)

Note

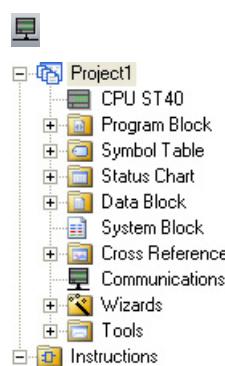
You can have static or dynamic IP information in the CPU:

- Static IP information: If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block is checked, then the Ethernet network information that you enter is static. Static IP information must be downloaded to the CPU before it is active in the CPU, and, if you want to change the IP information, this IP information can only be changed in the system block dialog and once again downloaded to the CPU.
 - Dynamic IP information: If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block is not checked, then you change the IP address of the CPU through other means and this IP address information is considered to be dynamic. You can change the IP address information in the Communications dialog or with the SIP_ADDR instruction in the user program.
 - For both static and dynamic IP, the information is stored in persistent memory.
-

Configuring the IP information in the Communications dialog (dynamic IP information)

IP information changes done through the Communications dialog are immediate and do not require a download of the project.

To access this dialog, perform one of the following:



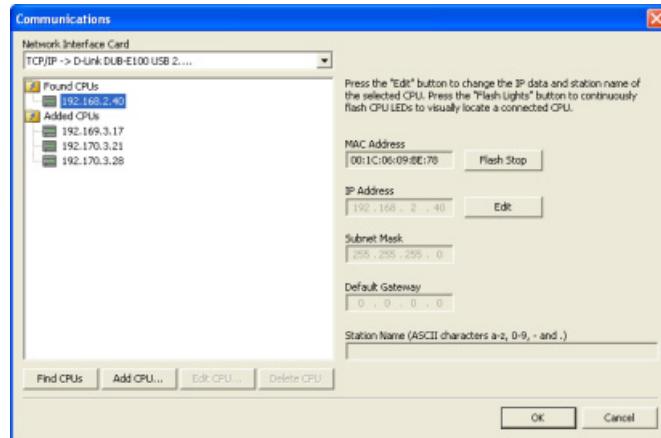
- In the Navigation bar, click the "Communications" button.
- In the Project tree, select the "Communications" node, then press Enter; or double-click the "Communications" node.

You can access CPUs in one of two ways:

- "Found CPUs": CPUs located on your local network
- "Added CPUs": CPUs on the local or remote networks (for example, CPUs accessed on another network through a router)

For "Found CPUs" (CPUs located on your local network), use the "Communications dialog" to connect with your CPU:

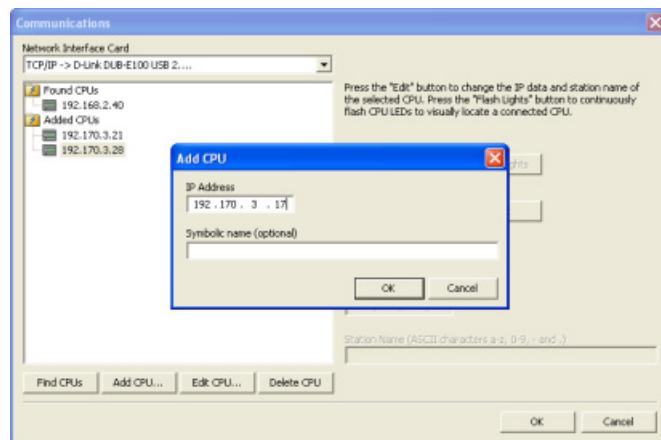
- Click the "Network Interface Card" dropdown list, and select the "TCP/IP" Network Interface Card (NIC) for your programming device.
- Click the "Find CPUs" button to display all operational CPUs ("Found CPUs") on the local Ethernet network.. All CPUs have a default IP address.
- Highlight a CPU, and then click "OK".



For "Added CPUs" (CPUs on the local or remote networks), use the "Communications dialog" to connect with your CPU:

- Click the "Network Interface Card" dropdown list, and select the "TCP/IP" Network Interface Card (NIC) for your programming device.
- Click the "Add CPU" button to do one of the following:
 - Enter the IP address of a CPU that is accessible from the programming device, but is not on the local network.
You can add these CPUs, select them as the communication partner in STEP 7-Micro/WIN SMART, and program and operate these CPUs in the same way you would a CPU on the local network. As long as there is a valid network path through routers, STEP 7-Micro/WIN SMART can communicate with any S7-200 SMART CPU.
 - Enter the IP address of a CPU directly that is on the local network.
You can add multiple CPUs, on the local network and/or remote network. As always, STEP 7-Micro/WIN SMART communicates with one CPU at a time. All CPUs have a default IP address.
- Highlight a CPU, and then click "OK".

8.4 Ethernet



To enter or change IP information, perform the following:

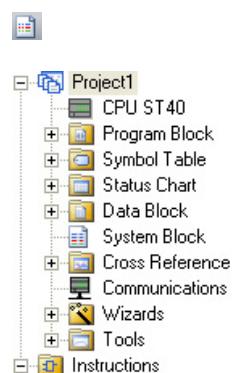
- Click the required CPU.
- If you need to identify which CPU to configure or change, click the "Flash Lights" button. This button flashes the STOP, RUN, and FAULT lights for the highlighted CPU in the list.
- Click the "Edit" button to make changes in the IP information.
- Change the following IP information:
 - IP address
 - Subnet mask
 - Default gateway
 - Station name
- Press the "Set" button. When the "Set" button is pressed, these values are updated within the CPU.
- When finished, click "OK".

When you configure IP information for the onboard Ethernet port in the Communications dialog, this information is "dynamic". If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block dialog is not checked, then you must enter IP information in the Communications dialog. You can enter new IP address information and update this information in the CPU by clicking the "Set" button.

Configuring the IP information in the System Block dialog (static IP information)

IP information configuration or changes done in the system block are part of the project and do not become active until you download your project to the CPU.

To access this dialog, perform one of the following:



- In the Navigation bar, click the "System Block" button.
- In the Project tree, select the "System Block" node, then press Enter; or double-click the "System Block" node.

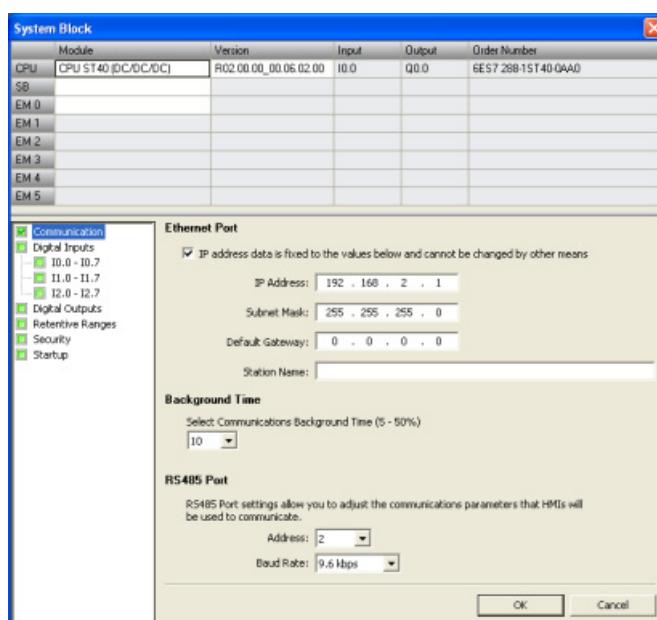
To enter or change IP information, perform the following:

- If not already checked, click the "IP address data is fixed to the values below and cannot be changed by other means" checkbox. The Ethernet port IP information fields are enabled.
- Enter or change the following IP information:
 - IP address
 - Subnet mask
 - Default gateway
 - Station name

Note

The Station Name follows the standard DNS (Domain Name System) naming conventions. The S7-200 SMART CPUs limit the Station Name to a maximum of 63 characters. The Station Name can consist of the lower case letters a through z, the digits 0 through 9, the hyphen character (minus sign), and the period character.

Certain names are not allowed and this can depend on the tool used to set the Station Name. The Station Name must not have the format "n.n.n.n" where n is a value of 0 through 999. The Station Name cannot begin with the string "port-nnn" or the string "port-nnn-nnnnn" where "n" is a digit 0 through 9 (for example, "port-123" and "port-123-45678" are illegal). The Station Name cannot start or end with the hyphen or period.



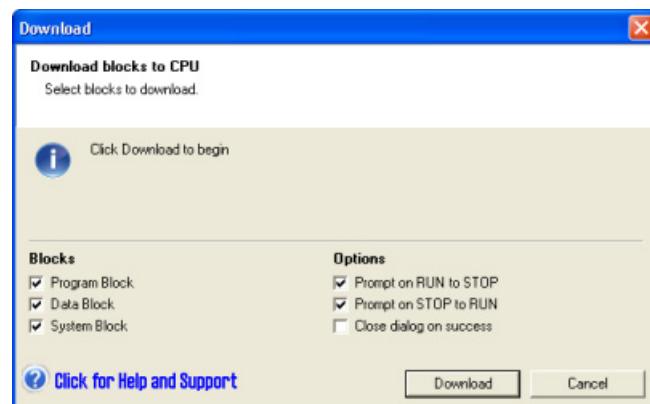
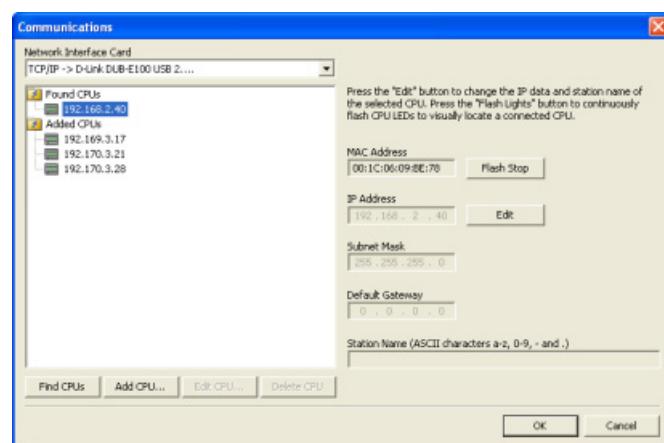
When you check the "IP address data is fixed to the values below and cannot be changed by other means" checkbox in the system block dialog, the IP information that you enter for the onboard Ethernet port is static. Static IP information must be downloaded to the CPU before it is active in the CPU. If you want to change the IP information, this IP information can only be changed in the system block dialog and once again downloaded to the CPU.

Note

If the "IP address data is fixed to the values below and cannot be changed by other means" checkbox is checked, then the IP information cannot be set in the Communications dialog.

In order to use the SIP_ADDR instruction, the "IP address data is fixed to the values below and cannot be changed by other means" checkbox must be unchecked.

After completing the IP information configuration, download the project to the CPU. All CPUs and devices that have valid IP addresses are displayed in the Communications dialog.



Configuring the IP information in the user program (dynamic IP information)

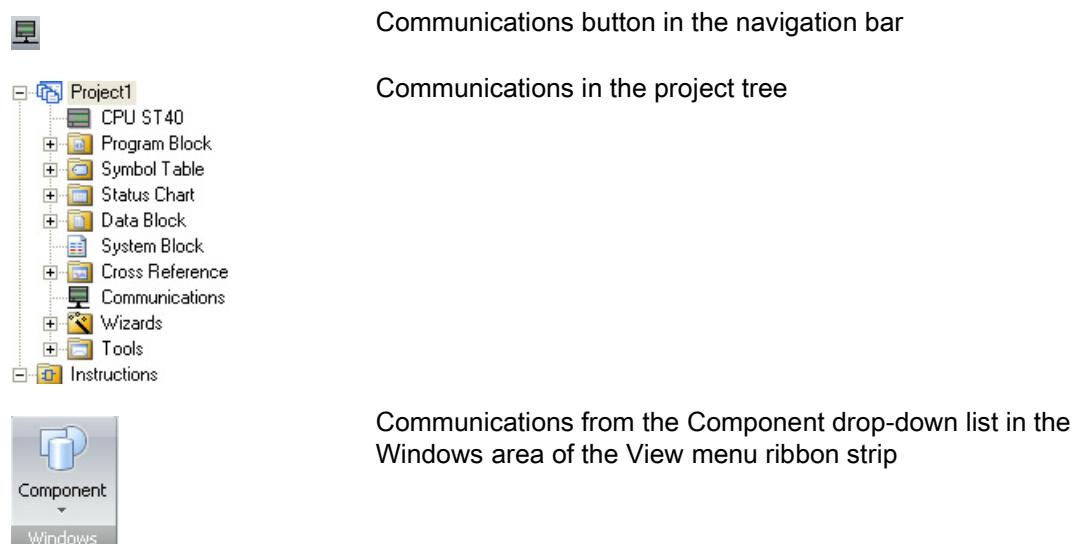
The SIP_ADDR instruction sets the CPU's IP address to the value found in its "ADDR" input, the CPU's subnet mask to the value found in its "MASK" input, and the CPU's gateway to the value found in its "GATE" input.

IP information or changes done through the SIP_ADDR instruction are immediate and do not require a download of the project. The IP address information set with the SIP_ADDR instruction is stored in permanent memory in the CPU.

Refer to the "Get IP address and set IP address (Ethernet)" (Page 181) instructions for more information.

8.4.5.3 Searching for CPUs and devices on your Ethernet network

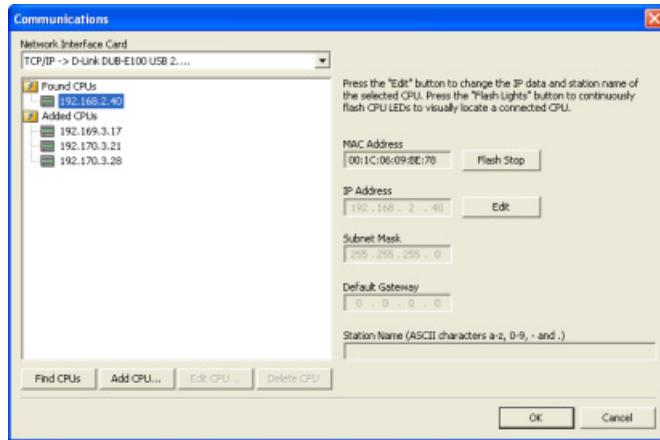
You can search for and identify the S7-200 SMART CPUs that are attached to your Ethernet network in the "Communications" dialog. To access this dialog, click one of the following:



The "Communications" dialog will autodetect all connected and available S7-200 SMART CPUs on a given Ethernet network by creating a lifelist. (See the figure below.) After selecting a CPU, the following detailed information about the CPU is listed:

- MAC address
- IP information
- Station name

The IP address of a CPU is not associated with a STEP 7-Micro/WIN SMART project. Opening a STEP 7-Micro/WIN SMART project does not automatically select an IP address or establish a connection to a CPU. Every time you create a new or open an existing STEP 7-Micro/WIN SMART project, you must go to the Communications dialog to establish a connection to a CPU. The Communications dialog will show the last selected CPU.



8.4.6 Locating the Ethernet (MAC) address on the CPU

In Ethernet networking, a Media Access Control address (MAC address) is an identifier assigned to the network interface by the manufacturer for identification. A MAC address usually encodes the manufacturer's registered identification number.

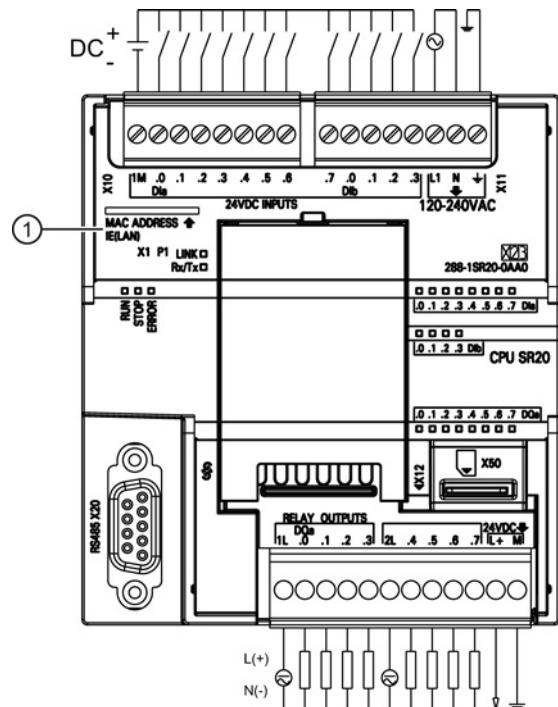
The standard (IEEE 802.3) format for printing MAC addresses in human-friendly form is six groups of two hexadecimal digits separated by hyphens (-) or colons (:), for example, 01-23-45-67-89-ab or 01:23:45:67:89:ab.

Note

Each CPU is loaded at the factory with a permanent, unique MAC address. You cannot change the MAC address of a CPU.

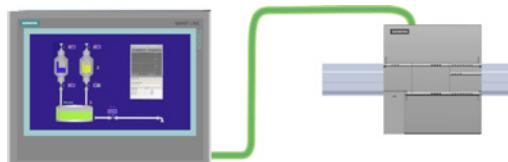
8.4 Ethernet

The MAC address is printed on the front, upper-left corner of the CPU. Note that you have to open the upper door to see the MAC address information.



① MAC address

8.4.7 HMI-to-CPU communication



The CPU supports Ethernet communication connections to HMIs. The following requirements must be considered when setting up communications between CPUs and HMIs:

Configuration/Setup:

- The CPU must be configured with an IP address.
- The HMI must be setup and configured to connect with the IP address of the CPU.
- No Ethernet switch is required for one-to-one communications; an Ethernet switch is required for more than two devices in a network.

Note

The rack-mounted CSM1277 4-port Ethernet switch can be used to connect your CPUs and HMI devices. The Ethernet port on the CPU does not contain an Ethernet switching device.

Supported functions:

- The HMI can read/write data to the CPU.
- Messages can be triggered, based upon information retrieved from the CPU.
- System diagnostics

To ensure that your CPU and HMI are communicating properly, follow the sequence of steps in the table below:

Table 8- 1 Required steps in configuring communications between an HMI and a CPU

Step	Task
1	<p>Establishing the hardware communications connection</p> <p>An Ethernet interface establishes the physical connection between an HMI and a CPU. Since Auto-Cross-Over functionality is built into the CPU, you can use either a standard or crossover Ethernet cable for the interface. An Ethernet switch is not required to connect an HMI and a CPU.</p> <p>Refer to "Establishing the hardware communications connection" (Page 25) for more information.</p>
2	If you have already created a project with a CPU, open your project in STEP 7-Micro/WIN SMART. If not, create a project and insert a CPU In the project.
3	<p>Configuring an IP address in your project</p> <p>Use the same configuration process; however, you must configure IP addresses for the HMI and the CPU. You must download the configuration for each CPU and HMI device.</p> <p>Refer to "Configuring or changing an IP address for a CPU or device in your project" (Page 343) for more information.</p>

Note

You can restrict communication writes to a specific range of V memory; this can affect HMI communications. See "Configuring system security" (Page 115) for more information.

8.5 RS485

An RS485 network is a differential (multi-point) network and can have up to 126 addressable nodes per network and up to 32 devices per segment. Repeaters are used to segment the network. Repeaters are not addressable nodes; therefore, they are not included in the count of addressable nodes, but are counted in the devices per segment.

RS485 allows for data transfer at a high speed (from 100 m at 12 Mbit/s to 1 km at 187.5 Kbit/s).

RS485 can operate with the PPI protocol and Freeport:

- PPI protocol: Can operate on RS485 or RS232 (half-duplex). Possible connections include:
 - PPI protocol devices
 - RS485 HMI displays
- Freeport: Can operate on RS485 or RS232 (half-duplex). Possible connections include:
 - RS485-compatible devices (for example, a bar code scanner)
 - Devices that have RS485 interfaces (for example, a control system)
 - Third-party devices using Freeport
 - Modems

8.5.1 PPI protocol

Definition

PPI is a master-slave protocol: the master devices send requests to the slave devices, and the slave devices respond. See the following figure. Slave devices do not initiate messages, but wait until a master sends them a request or polls them for a response.



Masters communicate to slaves by means of a shared connection which is managed by the PPI protocol. PPI does not limit the number of masters that can communicate with any one slave; however, you cannot install more than 32 masters on the network.

PPI protocol and S7-200 SMART CPUs

PPI Advanced allows network devices to establish a logical connection between the devices. With PPI Advanced, there are a limited number of connections supplied by each device. See the following table for the number of connections supported by the S7-200 SMART CPU.

All S7-200 SMART CPUs support both PPI and PPI Advanced protocols.

Table 8- 2 Number of connections for the S7-200 SMART CPU

Module	Baud rate	Connections
RS485 port	9.6 kbaud, 19.2 kbaud, or 187.5 kbaud	4
RS485/RS232 signal board	9.6 kbaud, 19.2 kbaud, or 187.5 kbaud	4

8.5.2 Baud rate and network address

8.5.2.1 Definition of baud rate and network address

Baud rate

The speed that data is transmitted across the network is the baud rate, which is typically measured in units of kilobaud (kbaud) or megabaud (Mbaud). The baud rate measures how much data can be transmitted within a given amount of time. For example, a baud rate of 19.2 kbaud describes a transmission rate of 19,200 bits per second.

Every device that communicates over a given network must be configured to transmit data at the same baud rate. Therefore, the fastest baud rate for the network is determined by the slowest device connected to the network.

The following table lists the baud rates supported by the S7-200 SMART CPU.

Table 8- 3 Baud rate supported by the S7-200 SMART CPU

Network	Baud rate
PPI protocol	9.6 kbaud, 19.2 kbaud, and 187.5 kbaud only
Freeport Mode	1200 baud to 115.2 kbaud

Network address

The network address is a unique number that you assign to each device on the network. The unique network address ensures that the data is transferred to or retrieved from the correct device. The S7-200 SMART CPU supports network addresses from 0 to 126. The following table lists the default (factory) settings for the S7-200 SMART devices.

Table 8- 4 Default addresses for S7-200 SMART devices

S7-200 SMART device	Default address
HMI	1
S7-200 SMART CPU	2

8.5.2.2 Setting the baud rate and network address for the S7-200 SMART CPU

Introduction

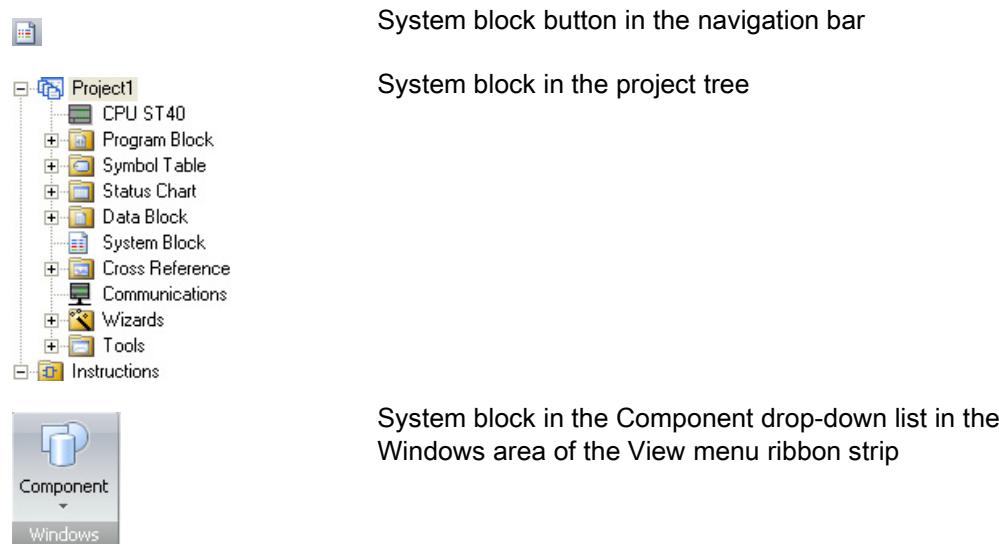
You must configure the RS485 port network address and baud rate for the S7-200 SMART CPU so that the CPU can communicate with SIMATIC HMIs over the RS485 network (for example, the TD400C).

The RS485 port network address must be unique from the network addresses of other devices on the RS485 network, and the RS485 port baud rate must be the same as the other devices on the RS485 network. The default RS485 port network address is 2, and the default RS485 port baud rate for each CPU port is 9.6 kbaud.

The system block of the CPU stores the RS485 port network address and baud rate. After you select the parameters for the CPU, you must download the system block to the S7-200 SMART CPU.

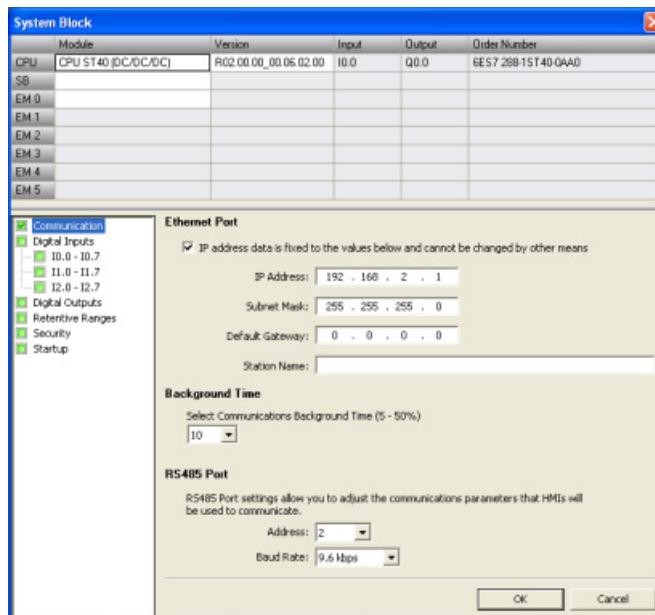
Procedure

To access the "System Block" dialog, click one of the following:



After you select the "System Block" dialog, you perform the following steps:

1. Select the network address and baud rate for the RS485 port.
2. Download the system block to the CPU.



Note

Freeport protocol baud rates are set using SM memory.

8.5.3 Sample RS485 network configurations

8.5.3.1 Single-master PPI networks

Introduction

The following network configurations are possible using only S7-200 SMART devices:

- Single-master PPI networks
- Multi-master and multi-slave PPI networks
- Complex PPI networks

Single-master PPI networks

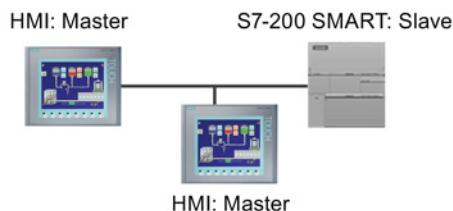
In the sample network in the figure below, a human-machine interface (HMI) device (for example, a TD400C, TP, or KP) is the network master:



In the sample network, the CPU is a slave that responds to requests from the master.

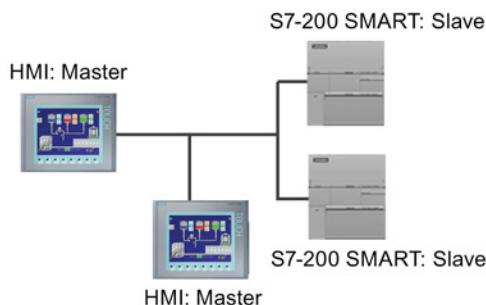
8.5.3.2 Multi-master and multi-slave PPI networks

The following figure shows a sample network of multiple masters with one slave. The HMI devices share the network.



The HMI devices are masters and must have separate network addresses. The S7-200 SMART CPU is a slave.

The following figure shows a PPI network with multiple masters communicating with multiple slaves. In this example, the HMI can request data from any CPU slave.



All devices (masters and slaves) have different network addresses. The S7-200 SMART CPUs are slaves.

8.5.4 Building your network

8.5.4.1 General guidelines

Always install appropriate surge suppression devices for any wiring that could be subject to lightning surges.

Avoid placing low-voltage signal wires and communication cables in the same wire tray with AC wires and high-energy, rapidly switched DC wires. Always route wires in pairs, with the neutral or common wire paired with the hot or signal-carrying wire.

The communication port of the S7-200 SMART CPU is not isolated. Consider using an RS485 repeater to provide isolation for your network.

NOTICE

Avoiding unwanted current flow

Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable.

These unwanted currents can cause communications errors or can damage equipment.

Be sure all equipment that you are about to connect with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

8.5.4.2 Determining the distances, transmission rates, and cable lengths for your network

As shown in the following table, the maximum length of a network segment is determined by two factors: isolation (using an RS485 repeater) and baud rate.

Isolation is required when you connect devices at different ground potentials. Different ground potentials can exist when grounds are physically separated by a long distance. Even over short distances, load currents of heavy machinery can cause a difference in ground potential.

Table 8- 5 Maximum length for a network cable

Baud rate	Non-isolated CPU port ¹	CPU port with repeater
9.6 kbaud to 187.5 kbaud	50 m	1,000 m
500 kbaud	Not supported	400 m
1 Mbaud to 1.5 Mbaud	Not supported	200 m
3 Mbaud to 12 Mbaud	Not supported	100 m

¹ The maximum distance allowed without using an isolator or repeater is 50 m. You measure this distance from the first node to the last node in the segment.

8.5.4.3 Repeaters on the network

An RS485 repeater provides bias and termination for the network segment. You can use a repeater for the following purposes:

- **To increase the length of a network**

Adding a repeater to your network allows you to extend the network another 50 m. If you connect two repeaters with no other nodes in between, (as shown in the figure below), you can extend the network to the maximum cable length for the baud rate. You can use up to 9 repeaters in series on a network, but the total length of the network must not exceed 9600 m.

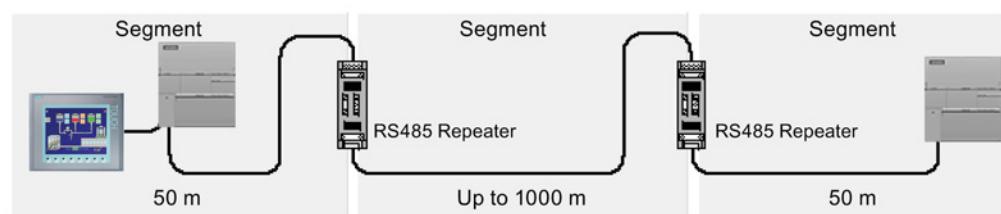
- **To add devices to a network**

Each segment can have a maximum of 32 devices connected up to 50 m at 9600 baud. Using a repeater allows you to add another segment (32 devices) to the network.

- **To electrically isolate different network segments**

Isolating the network improves the quality of the transmission by separating the network segments which may be at different ground potentials.

A repeater on your network counts as one of the nodes on a segment, even though it is not assigned a network address. The following is a sample network with repeaters.



8.5.4.4 Selection of the network cable

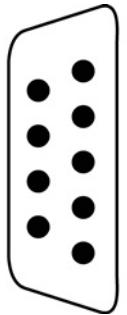
S7-200 SMART CPU networks use the RS485 standard on twisted pair cables. The following table lists the specifications for the network cable. You can connect up to 32 devices on a network segment.

Specifications	Description
Cable type	Shielded, twisted pair
Loop resistance	$\leq 115 \Omega/\text{km}$
Effective capacitance	30 pF/m
Nominal impedance	Approximately 135Ω to 160Ω (frequency = 3MHz to 20 MHz)
Attenuation	0.9 dB/100 m (frequency=200 kHz)
Cross-sectional core area	0.3 mm ² to 0.5 mm ²
Cable diameter	8 mm +0.5 mm

8.5.4.5 Connector pin assignments

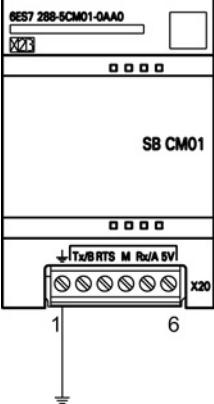
The RS485 communication port on the S7-200 SMART CPUs is RS485-compatible on a nine-pin subminiature D connector, in accordance with the PROFIBUS standard as defined in the European Standard EN 50170. The following table shows the connector that provides the physical connection for the communication port and describes the communication port pin assignments.

Table 8- 6 Pin assignments for the S7-200 SMART CPU integrated RS485 port (Port 0)

Pin number	Connector	Signal	Integrated RS485 port (Port 0)
1		Shield	Chassis ground
2		24 V Return	Logic common
3		RS485 Signal B	RS485 Signal B
4		Request-to-Send	RTS (TTL)
5		5 V Return	Logic common
6		+5 V	+5 V, 100 Ω series resistor
7		+24 V	+24 V
8		RS485 Signal A	RS485 Signal A
9		Not applicable	10-bit protocol select (input)
Connector shell	Shield	Chassis ground	

The CM01 signal board is RS485-compatible. The following table shows the connector that provides the physical connection for the signal board and describes the pin assignments.

Table 8- 7 Pin assignments for the S7-200 SMART CM01 Signal Board (SB) port (Port 1)

Pin number	Connector	Signal	CM01 Signal Board (SB) port (Port 1)
1		Ground	Chassis ground
2		Tx/B	RS232-Tx/RS485-B
3		Request-to-Send	RTS (TTL)
4		M-ground	Logic common
5		Rx/A	RS232-Rx/RS485-A
6		+5 V DC	+5 V, 100 Ω series resistor

8.5.4.6 Biasing and terminating the network cable

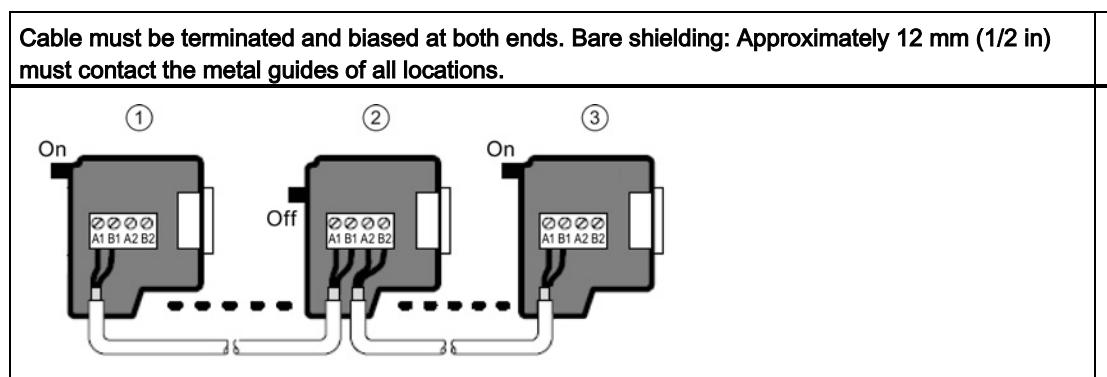
Siemens provides two types of network connectors that you can use to easily connect multiple devices to a network:

- Standard network connector
- Connector that includes a port which allows you to connect an HMI device to the network without disturbing any existing network connections

The programming port connector passes all signals (including the power pins) from the S7-200 SMART CPU through to the programming port, which is especially useful for connecting devices that draw power from the S7-200 SMART CPU (such as a TD 400C).

Both connectors have two sets of terminal screws to allow you to attach the incoming and outgoing network cables. Both connectors also have switches to bias and terminate the network selectively. The following shows typical biasing and termination for the cable connectors.

Table 8- 8 Biasing and termination for cable connectors

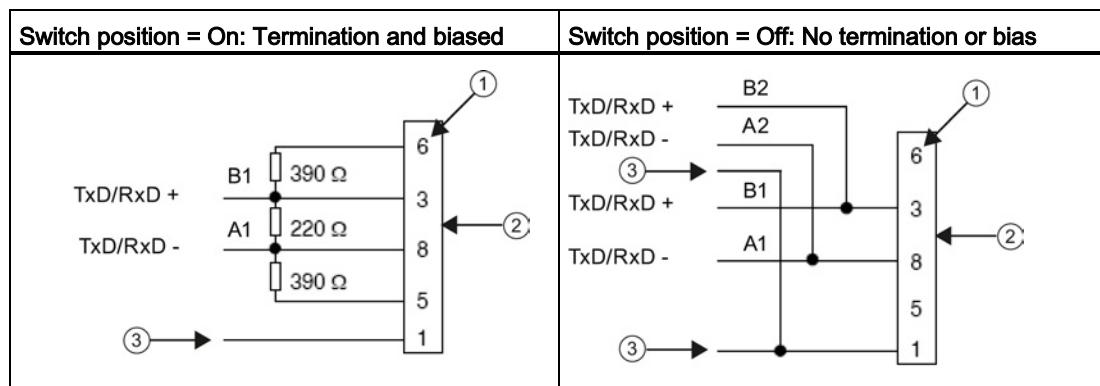


① Switch position = On: Terminated and biased

② Switch position = Off: No termination or bias

③ Switch position = On: Terminated and biased

Table 8- 9 Termination and bias switch positions



① Pin number

② Network connector

③ Cable shield

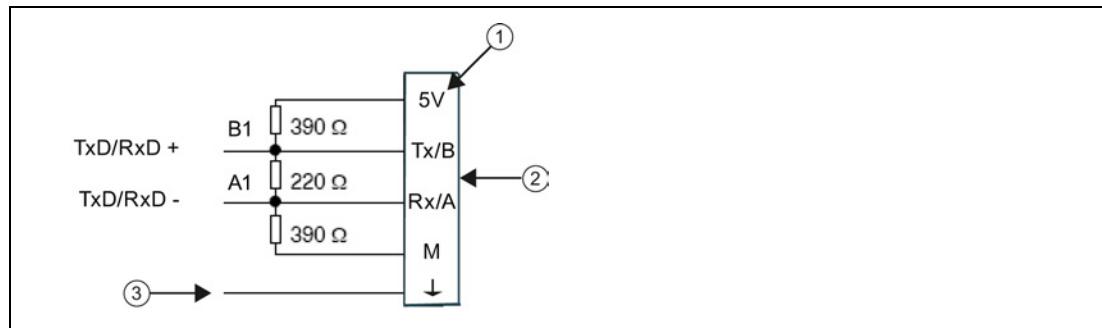
8.5.4.7

Biassing and terminating the CM01 signal board

You can use the CM01 signal board to easily connect multiple devices to a network.

The signal board passes all signals (including the power pins) from the S7-200 SMART CPU through to the programming port, which is especially useful for connecting devices that draw power from the S7-200 SMART CPU (such as a TD 400C).

Table 8- 10 Biasing and termination for the CM01 signal board



- ① Terminal name
- ② Terminal block
- ③ Cable shield

8.5.4.8 Using HMI devices on your RS485 network

Introduction

The S7-200 SMART CPU supports many types of RS485 HMI devices from Siemens and also from other manufacturers. While some of these HMI devices (such as the TD400C) do not allow you to select the communication protocol used by the device, other devices (such as the KP and TP product lines) allow you to select the communication protocol for that device.

Guidelines

If your HMI device allows you to select the communication protocol, consider the following guidelines:

- For an HMI device connected to the communication port of the CPU, with no other devices on the network, select the PPI protocol for the HMI device.
- For an HMI device connected to the communication port of the CPU which has been configured as a master, select the PPI protocol for the HMI device. Advanced PPI is optimal.

For more information about how to configure the HMI device, refer to the specific manual for your device (see the following table). These manuals are included in the STEP 7-Micro/WIN SMART documentation CD.

Table 8- 11 RS485 HMI devices supported by the S7-200 SMART CPU

HMI	Configuration software
TD400C	Text Display wizard (part of STEP 7-Micro/WIN SMART)
KTP600 DP	WinCC flexible
KTP1000 DP	WinCC flexible

8.5.5 Freeport mode

8.5.5.1 Creating user-defined protocols with Freeport mode

Introduction

Freeport mode allows your program to control the communication port of the S7-200 SMART CPU. You can use Freeport mode to implement user-defined communication protocols to communicate with many types of intelligent devices. Freeport mode supports both ASCII and binary protocols.

Using Freeport mode

To enable Freeport mode, you use special memory bytes SMB30 (for the Integrated RS485 port (Port 0)) and SMB130 (for the CM01 Signal Board (SB) port (Port 1)). Your program uses the following to control the operation of the communication port:

- **Transmit instruction (XMT) and the transmit interrupt:**

The Transmit instruction allows the S7-200 SMART CPU to transmit up to 255 characters from the COM port. The transmit interrupt notifies your program in the CPU when the transmission has been completed.

- **Receive character interrupt:**

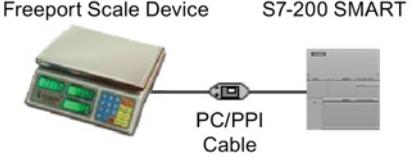
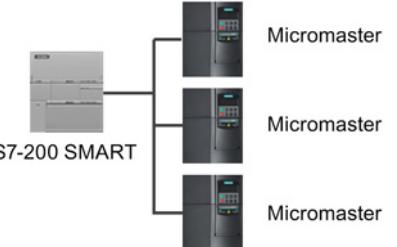
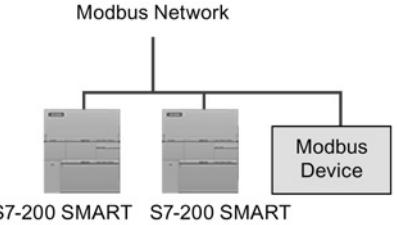
The receive character interrupt notifies the user program that a character has been received on the COM port. Your program can then act on that character, based on the protocol being implemented.

- **Receive instruction (RCV):**

The Receive instruction receives the entire message from the COM port and then generates an interrupt for your program when the message has been completely received. You use the SM memory of the CPU to configure the Receive instruction for starting and stopping the receiving of messages, based on defined conditions. The Receive instruction allows your program to start or stop a message based on specific characters or time intervals. Most protocols can be implemented with the Receive instruction, instead of using the more cumbersome receive-character interrupt method.

Freeport mode is active only when the CPU is in RUN mode. Setting the CPU to STOP mode halts all Freeport communication, and the communication port then reverts to the PPI protocol with the settings which were configured in the system block of the CPU.

Table 8- 12 Using Freeport mode

Network configuration	Description
Using Freeport over an RS232 connection	<p>Example: Using an S7-200 SMART CPU with an electronic scale that has an RS232 port.</p>  <ul style="list-style-type: none"> • Connect the two devices using one of the following methods: <ul style="list-style-type: none"> – RS232/PPI Multi-Master cable connects the RS232 port on the scale to the RS485 port on the CPU. (Set the cable to PPI/Freeport mode, switch 5 = 0.) – Using the CM01 signal board (SB) (S CPUs only) which supports RS232 and RS485, you can connect the RS232 device directly to the CPU SB RS232 without using the PC/PPI cable. • CPU uses Freeport to communicate with the scale. • Baud rate can be from 1200 baud to 115.2 kbaud. • User program defines the protocol.
Using USS protocol	 <p>Example: Using an S7-200 SMART CPU with SIMODRIVE MicroMaster drives.</p> <ul style="list-style-type: none"> • STEP 7-Micro/WIN SMART provides a USS library. • The CPU is a master, and the drives are slaves.
Creating a user program that emulates a slave device on another network	 <p>Example: Connecting S7-200 SMART CPUs to a Modbus network.</p> <ul style="list-style-type: none"> • User program in the CPU emulates a Modbus slave. • STEP 7-Micro/WIN SMART provides a Modbus library.

8.5.5.2 Using the RS232/PPI Multi-Master cable and Freeport mode with RS232 devices

Purpose

You can use the RS232/PPI Multi-Master cable and the Freeport communication functions to connect the S7-200 SMART CPU to many devices that are compatible with the RS232 standard. The cable must be set to PPI/Freeport mode (switch 5 = 0) for Freeport operation. Switch 6 selects either Local mode (DCE) (switch 6 = 0), or Remote mode (DTE) (switch 6 = 1).

The RS232/PPI Multi-Master cable is in Transmit mode when data is transmitted from the RS232 port to the RS485 port. The cable is in Receive mode when it is idle or is transmitting data from the RS485 port to the RS232 port. The cable changes from Receive to Transmit mode immediately when it detects characters on the RS232 transmit line.

The CM01 signal board (SB) (S CPUs only) supports both RS232 half-duplex and RS485. With the CM01 signal board, you can connect an RS232 device directly to the CPU SB RS232 port without using a PC/PPI cable.

Baud Rates and turnaround time

The RS232/PPI Multi-Master cable supports baud rates between 1200 baud and 115.2 kbaud. Use the DIP switches on the housing of the RS232/PPI Multi-Master cable to configure the cable for the correct baud rate. The following table shows the baud rates and switch positions.

Table 8- 13 Turnaround time and settings

Baud rate	Turnaround time	Settings (1 = Up)
115200	0.15 ms	110
57600	0.3 ms	111
38400	0.5 ms	000
19200	1.0 ms	001
9600	2.0 ms	010
4800	4.0 ms	011
2400	7.0 ms	100
1200	14.0 ms	101

The cable switches back to Receive mode when the RS232 transmit line is in the idle state for a period of time defined as the turnaround time of the cable. The baud rate selection of the cable determines the turnaround time, as shown in the table.

If you are using the RS232/PPI Multi-Master cable in a system where Freeport communications is used, the program in the S7-200 SMART CPU must comprehend the turnaround time for the following situations:

- The CPU responds to messages transmitted by the RS232 device.

After the CPU receives a request message from the RS232 device, the CPU must delay the transmission of a response message for a period of time greater than or equal to the turnaround time of the cable.

- The RS232 device responds to messages transmitted from the CPU.

After the CPU receives a response message from the RS232 device, the CPU must delay the transmission of the next request message for a period of time greater than or equal to the turnaround time of the cable.

In both situations, the delay allows the RS232/PPI Multi-Master cable sufficient time to switch from Transmit mode to Receive mode so that data can be transmitted from the RS485 port to the RS232 port.

8.6 RS232

An RS232 network is a point-to-point connection between two devices. RS232 allows for data transfer at relatively slow speeds (up to 115.2 kbaud) and short distances (up to 50 feet).

Possible RS232 connections include the following:

- Freeport
- Modems
- RS232-compatible devices (for example, a bar code scanner)
- Devices that have RS232 interfaces (for example, a control system)
- RS232 displays

Libraries

9.1 Creating a user-defined library of instructions

STEP 7-Micro/WIN SMART allows you either to create a custom library of instructions, or to use a library created by someone else.

Creating a library

To create a user-defined library of instructions, you create standard STEP 7-Micro/WIN SMART subroutines and group them together. By grouping these subroutines into a library, you can hide the code to prevent unwanted changes and to protect the technology (know-how) of the author.

To create a user-defined library, perform the following tasks:

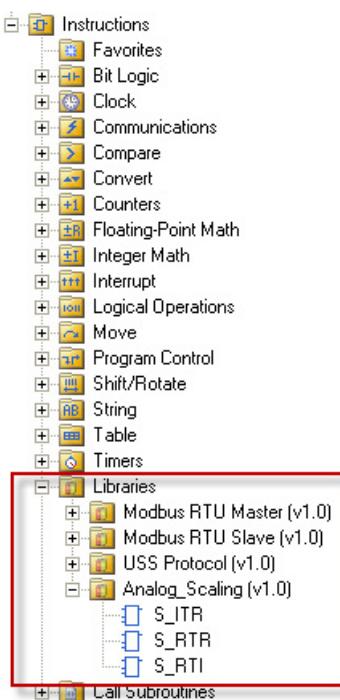
1. Write the program as a standard project and put the function to be included in the library into subroutines.
2. Ensure that you have assigned a symbolic name to all V memory locations in the subroutines or interrupt routines. To minimize the amount of V memory that the library requires, use sequential V memory locations.
3. Rename the subroutines to the names that you want to appear in the instruction library.
4. Click the Create button  from the Libraries area of the File menu ribbon strip to compile and create the new library from the subroutines you selected. If the subroutine references interrupts, STEP 7-Micro/WIN SMART also includes the interrupt routines in the library.

Adding a library to a project

Use the following procedure to add a library to a project and use the library instructions:

1. Click the Add/Remove button  from the Libraries area of the File menu ribbon strip to add a new library.
2. Navigate to and select the specific library you wish to add.
3. After you add the library to your project you can add instructions from the Libraries folder of the Instructions folder in the project tree into your program as you would any standard instruction.

If the library routine requires any V memory, STEP 7-Micro/WIN SMART prompts you when you compile the project to assign a block of memory. Click the Memory button  from the Libraries area of the File menu ribbon strip to assign a starting address for the V memory that the user-defined library requires. Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu.



Further information

See the STEP 7-Micro/WIN SMART online help library topics for library programming tips and a user-defined library example.

9.2 USS library instructions

9.2.1 USS communication overview

9.2.1.1 USS protocol overview

STEP 7-Micro/WIN SMART instruction libraries make controlling Siemens drives easier by including pre-configured subroutines and interrupt routines that are specifically designed for using the USS protocol to communicate with a motor drive. You can control the physical drive and the read/write drive parameters with the USS instructions.

You find these instructions in the "Libraries" folder of the STEP 7-Micro/WIN SMART instruction tree. When you select a USS instruction, one or more associated subroutines and interrupts are added automatically.

The USS protocol library overview discusses the following subjects:

- Requirements for using the USS protocol (Page 371)
- Calculating the time required for communicating with the drive (Page 372)

Refer to "Using the USS protocol instructions (Page 373)" for a listing of USS protocol instructions, error codes, and example programs.

9.2.1.2 Requirements for using the USS protocol

The STEP 7-Micro/WIN SMART instruction libraries provide subroutines, interrupt routines, and instructions to support the USS protocol. The USS instructions use the following resources in the S7-200 SMART CPU:

- The USS protocol is an interrupt driven application. In the worst case, the receive message interrupt routine requires up to 2.5 ms to execute. During this time, all other interrupt events are queued for service after the receive message interrupt routine has been executed. If your application cannot tolerate this worst case delay, then you may want to consider other solutions for controlling drives.
- Initializing the USS protocol dedicates an S7-200 SMART CPU port for USS communications.

You use the USS_INIT instruction to select either USS or PPI for port 0 or port 1. (USS refers to the USS protocol for Siemens drives.) When a port is set to use the USS protocol for communicating with drives, you cannot use the port for any other purpose, including communicating with an HMI. The second communications port allows STEP 7-Micro/WIN SMART to monitor the control program while USS protocol is running.
- The USS instructions affect all of the SM locations that are associated with Freeport communication on the assigned port.
- The USS subroutines and interrupt routines are stored in your program. The USS instructions increase the amount of memory required for your program by up to 3050 bytes. Depending on the specific USS instructions used, the support routines for these instructions can increase the overhead for the control program by at least 2150 bytes and up to 3050 bytes.
- The variables for the USS instructions require a 400-byte block of V memory. The starting address for this block is assigned by the user and is reserved for USS variables.
- Some of the USS instructions also require a 16-byte communications buffer. As a parameter for the instruction, you provide a starting address in V memory for this buffer. It is recommended that a unique buffer be assigned for each instance of USS instructions.
- When performing calculations, the USS instructions use accumulators AC0 to AC3. You can also use the accumulators in your program; however, the values in the accumulators will be changed by the USS instructions.
- The USS instructions cannot be used in an interrupt routine.

9.2.1.3 Calculating the time required for communicating with the drive

Communications with the drive are asynchronous to the S7-200 SMART CPU scan. The CPU typically completes several scans before one drive communications transaction is completed. The following factors help determine the amount of time required:

- Number of drives present
- Baud rate
- Scan time of the CPU

Some drives require longer delays when using the parameter access instructions. The amount of time required for a parameter access is dependent on the drive type and the parameter being accessed.

After a USS_INIT instruction assigns Port 0 to use the USS Protocol (or USS_INIT_P1 for port 1), the CPU regularly polls all active drives at the intervals shown in the following table. You must set the time-out parameter of each drive to allow for this task:

Table 9- 1 Communications times

Baud rate	Time between polls of active drives (with no parameter access instructions active)
1200	240 ms (maximum) times the number of drives
2400	130 ms (maximum) times the number of drives
4800	75 ms (maximum) times the number of drives
9600	50 ms (maximum) times the number of drives
19200	35 ms (maximum) times the number of drives
38400	30 ms (maximum) times the number of drives
57600	25 ms (maximum) times the number of drives
115200	25 ms (maximum) times the number of drives

9.2.2 USS program instructions

9.2.2.1 Using the USS protocol instructions

Procedure

To use the USS protocol instructions in your S7-200 SMART program, follow these steps:

1. Insert the USS_INIT instruction in your program and execute the USS_INIT instruction for one scan only. You can use the USS_INIT instruction either to initiate or to change the USS protocol communication parameters.

When you insert the USS_INIT instruction, several hidden subroutines and interrupt routines are automatically added to your program.

2. Place only one USS_CTRL instruction in your program for each active drive.

You can add as many USS_RPM_x and USS_WPM_x instructions as required, but only one of these can be active at a time.

3. Click the Memory button  from the Libraries area of the File menu ribbon strip to assign a starting address for the V Memory that the USS library requires. Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu.

4. Configure the drive parameters to match the baud rate and address used in the program.

5. Connect the communications cable between the S7-200 SMART CPU and the drives.

Ensure that all of the control equipment, such as the S7-200 SMART CPU, that is connected to the drive be connected by a short, thick cable to the same ground or star point as the drive.



Avoiding unwanted current flow

Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or damage equipment.

Ensure that all equipment that is connected with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

The shield must be tied to chassis ground or pin 1 on the 9-pin connector. It is recommended that you tie terminal 2-0V on the drive to chassis ground.

The USS protocol instructions consist of the following:

- USS_INIT (Page 374)
- USS_CTRL (Page 376)
- USS_RPM_X (Page 379)
- USS_WPM_x (Page 381)

USS protocol program examples (Page 384) and a listing of USS protocol error codes (Page 383) are also discussed in this section.

9.2.2.2 USS_INIT instruction

Table 9- 2 USS_INIT instruction

LAD / FBD	STL	Description
	<pre>CALL USS_INIT, Mode, Baud, Port, Active, Done, Error</pre>	<p>The USS_INIT instruction is used to enable and initialize, or to disable Siemens drive communications. Before any other USS instruction can be used, the USS_INIT instruction must be executed without errors. The instruction completes and the "Done" bit is set immediately, before continuing to the next instruction.</p>

The instruction is executed on each scan when the "EN" input is on.

Execute the USS_INIT instruction only once for each change in communications state. Use an edge detection instruction to pulse the "EN" input on. To change the initialization parameters, execute a new USS_INIT instruction.

Table 9- 3 Parameters for the USS_INIT instruction

Inputs/outputs	Data type	Operands
Mode, Port	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Baud, Active	DWORD	VD, ID, QD, MD, SD, SMD, LD, Constant, AC *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

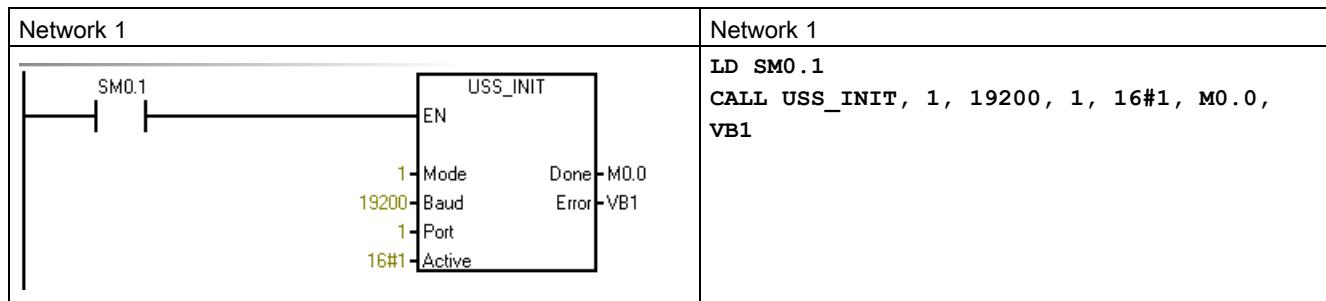
Table 9- 4 USS_INIT parameter descriptions

Parameter	Description
Mode	This value selects the communications protocol: <ul style="list-style-type: none"> An input value of 1 assigns the port to USS protocol and enables the protocol. An input value of 0 assigns the port to PPI and disables the USS protocol.
Baud	Sets the baud rate at 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200
Port	Sets the physical communication port (0 = RS485 integrated in CPU, 1 = RS485 or RS232 located on the optional CM01 signal board)
Active	Indicates which drives are active. Some drives only support addresses 0 through 30.
Done	Turned on when the USS_INIT instruction completes
Error	This output byte contains the result of executing the instruction. The USS protocol execution error codes (Page 383) define the error conditions that could result from executing the instruction.

Table 9- 5 Format for the Active drive parameter

Refer to the USS protocol execution error codes (Page 383) to compute the time between status polls and the error conditions that could result from executing the instruction.

Table 9- 6 USS_INIT example program



Refer to "Using the USS protocol instructions" (Page 373) for a listing of USS protocol instructions and error codes and example programs.

9.2.2.3 USS_CTRL instruction

Table 9- 7 USS_CTRL instruction

LAD / FBD	STL	Description
<pre> USS_CTRL EN RUN OFF2 OFF3 F_ACK DIR Drive Type Speed_SP Resp_R Error Status Speed Run_EN D_Dir Inhibit Fault </pre>	<pre> CALL USS_CTRL, RUN, OFF2, OFF3, F_ACK, DIR, Drive, Type, Speed_SP, Resp_R, Error, Status, Speed, Run_EN, D_Dir, Inhibit, Fault </pre>	<p>The USS_CTRL instruction is used to control an active Siemens drive. The USS_CTRL instruction places the selected commands in a communications buffer, which is then sent to the addressed drive ("Drive" parameter), if that drive has been selected in the "Active" parameter of the USS_INIT instruction.</p>

Only one USS_CTRL instruction should be assigned to each drive.

Some drives report speed only as a positive value. If the speed is negative, the drive reports the speed as positive, but reverses the "D_Dir" (direction) bit.

The "EN" bit must be on to enable the USS_CTRL instruction. This instruction should always be enabled.

Table 9- 8 Parameters of the USS_CTRL instruction

Inputs/outputs	Data types	Operands
RUN, OFF 2, OFF 3, F_ACK, DIR	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow
Resp_R, Run_EN, D_Dir, Inhibit, Fault	BOOL	I, Q, M, S, SM, T, C, V, L
Drive, Type	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD, Constant
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD
Status	WORD	VW, T, C, IW, QW, SW, MW, SMW, LW, AC, AQW, *VD, *AC, *LD
Speed_SP	REAL	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD, Constant
Speed	REAL	VD, ID, QD, MD, SD, SMD, LD, AC, *VD, *AC, *LD

RUN parameter

RUN (RUN/STOP) indicates whether the drive is on (1) or off (0). When the "RUN" bit is on, the drive receives a command to start running at the specified speed and direction. In order for the drive to run, the following must be true:

- Drive must be selected as "Active" in USS_INIT.
- "OFF2" and "OFF3" must be set to 0.
- "Fault" and "Inhibit" must be 0.

When "RUN" is off, a command is sent to the drive to ramp the speed down until the motor comes to a stop:

- The "OFF2" bit is used to allow the drive to coast to a stop.
- The "OFF3" bit is used to command the drive to stop quickly.

Resp_R parameter

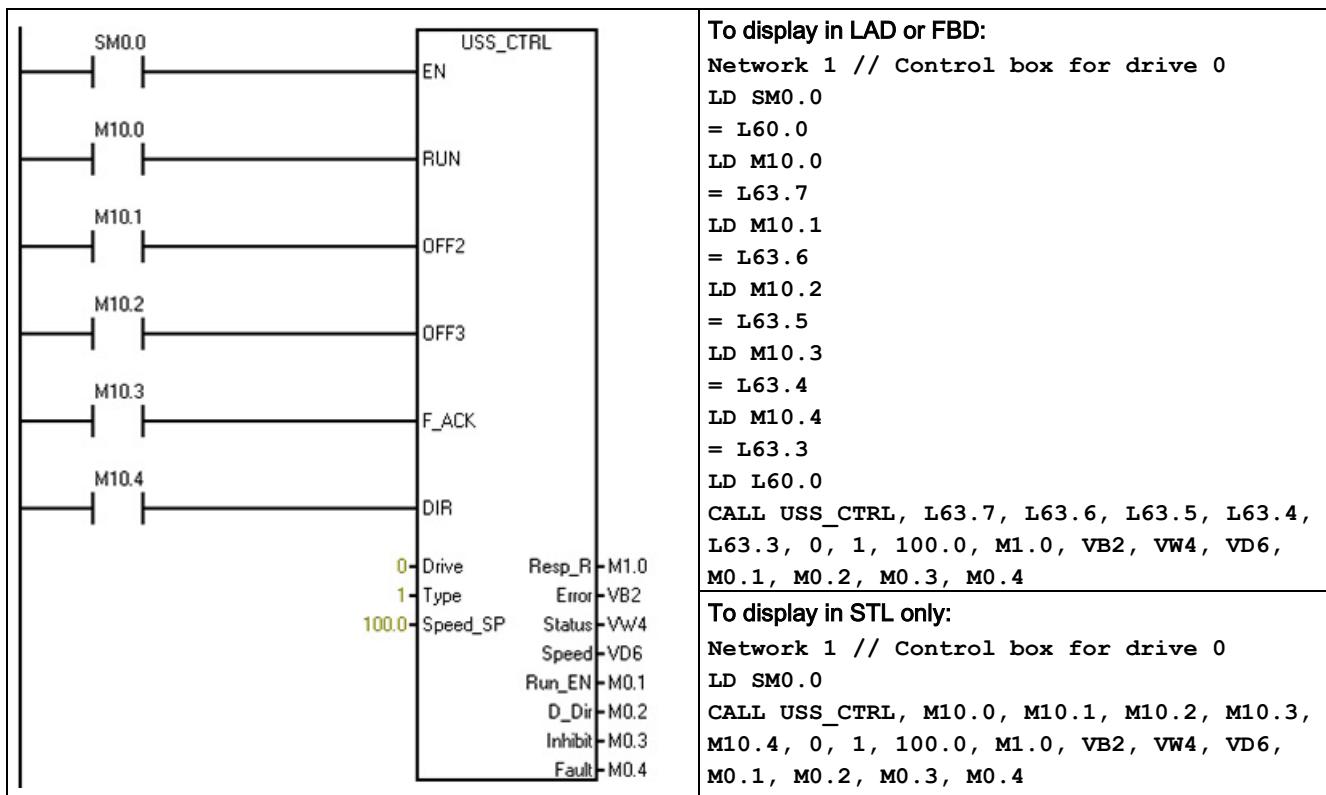
The "Resp_R" (response received) bit acknowledges a response from the drive. All the Active drives are polled for the latest drive status information. Each time the CPU receives a response from the drive, the "Resp_R" bit is turned on for one scan and all the following values are updated:

Parameter	Description
F_ACK (fault acknowledge)	Bit that acknowledges a fault in the drive. The drive clears the fault ("Fault" bit) when "F_ACK" goes from 0 to 1.
DIR (direction)	Bit that indicates in which direction the drive should move.
Drive (drive address)	Input for the address of the drive to which the USS_CTRL command is to be sent. Valid addresses: 0 to 31
Type (drive type)	Input that selects the type of drive
Speed_SP (speed setpoint)	Drive speed as a percentage of full speed: <ul style="list-style-type: none"> • Negative values of "Speed_SP" cause the drive to reverse its direction of rotation. • Range: -200.0% to 200.0%
Error	Byte that contains the result of the latest communications request to the drive. The USS protocol execution error codes (Page 383) define the error conditions that could result from executing the instruction.
Status	Raw value of the status word returned by the drive. The figures below show the status bits for standard status word and main feedback.
Speed	Drive speed as a percentage of full speed. Range: -200.0% to 200.0%
Run_EN (RUN enable)	Indicates the drive condition: <ul style="list-style-type: none"> • Running (1) • Stopped (0)
D_Dir	Indicates the drive's direction of rotation

9.2 USS library instructions

Parameter	Description
Inhibit	<p>Indicates the state of the "Inhibit" bit on the drive:</p> <ul style="list-style-type: none"> • 0: not inhibited • 1: inhibited <p>To clear the "Inhibit" bit, the following bits must be OFF:</p> <ul style="list-style-type: none"> • "Fault" • "RUN" • "OFF2" • "OFF3"
Fault	<p>Indicates the state of the "Fault" bit:</p> <ul style="list-style-type: none"> • 0: no fault • 1: fault <p>The drive displays the fault code. (Refer to the manual for your drive). To clear the "Fault" bit, correct the cause of the fault and turn on the "F_ACK" bit.</p>

Table 9- 9 USS_CTRL example program



Refer to "Using the USS protocol instructions" (Page 373) for a listing of USS protocol instructions and error codes and example programs.

9.2.2.4 USS_RPM_x instruction

Table 9- 10 USS_RPM_x instructions

LAD / FBD	STL	Description
<pre> USS_RPM_W EN XMT_REQ Drive Param Index DB_Ptr </pre>	<pre> CALL USS_RPM_W, XMIT_REQ, Drive, Param, Index, DB_Ptr, Done, Error, Value CALL USS_RPM_D, XMIT_REQ, Drive, Param, Index, DB_Ptr, Done, Error, Value CALL USS_RPM_R, XMIT_REQ, Drive, Param, Index, DB_Ptr, Done, Error, Value </pre>	<p>There are three read instructions for the USS protocol:</p> <ul style="list-style-type: none"> • USS_RPM_W instruction reads an unsigned word parameter. • USS_RPM_D instruction reads an unsigned double word parameter. • USS_RPM_R instruction reads a floating-point parameter. • Only one read (USS_RPM_x) or write (USS_WPM_x) instruction can be active at a time.

The USS_RPM_x transactions complete when the drive acknowledges receipt of the command or when an error condition is posted. The logic scan continues to execute while this process awaits a response.

Table 9- 11 Valid operands for the USS_RPM_x instructions

Inputs/outputs	Data type	Operands
XMT_REQ	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow conditioned by a rising edge detection element
Drive	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD, Constant
Param, Index	WORD	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AIW, *VD, *AC, *LD, Constant
DB_Ptr	DWORD	&VB
Value	WORD	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AQW, *VD, *AC, *LD
	DWORD, REAL	VD, ID, QD, MD, SD, SMD, LD, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC. *VD, *AC, *LD

The "EN" bit must be on to enable transmission of a request, and should remain on until the "Done" bit is set, signaling completion of the process. For example, a USS_RPM_x request is transmitted to the drive on each scan when the "XMT_REQ" input is on. Therefore, the "XMT_REQ" input should be pulsed on through an edge detection element which causes one request to be transmitted for each positive transition of the "EN" input.

Table 9- 12 USS_RPM_x parameter descriptions

Parameter	Description
XMT_REQ (transmit request)	When ON, a USS_RPM_x request is transmitted to the drive on every scan.
Drive	Address of the drive to which the USS_RPM_x command is to be sent. Valid addresses of individual drives are 0 to 31.
Param	Parameter number
Index	Index value of the parameter that is to be read

9.2 USS library instructions

Parameter	Description
DB_Ptr	The address of a 16-byte buffer must be supplied to the "DB_Ptr" input. This buffer is used by the USS_RPM_x instruction to store the results of the command issued to the drive.
Done	Turned on when the USS_RPM_x instruction completes
Error	This output byte contains the result of executing the instruction. The USS protocol execution error codes (Page 383) define the error conditions that could result from executing the instruction.
Value	Parameter value returned

When the USS_RPM_x instruction completes, the "Done" output is turned on and the "Error" output byte and the "Value" output contain the results of executing the instruction. The "Error" and "Value" outputs are not valid until the "Done" output turns on.

USS_RPM_x and USS_WPM_x example program

Table 9- 13 USS_RPM_x and USS_WPM_x example program

Network 1	Network 1
	<pre> LD M10.5 = L60.0 LD M10.5 EU = L63.7 LD L60.0 CALL USS_RPM_W, L63.7, 0, 5, 0, &VB20, M1.1, VB10, VW12 </pre>
Network 2	Network 2
	<pre> LD M10.6 = L60.0 LD M10.6 EU = L63.7 LDN SM0.0 = L63.6 LD L60.0 CALL USS_WPM_W, L63.7, L63.6, 0, 2000, 0, 50.0, &VB40, M1.2, VB14 </pre>

Refer to "Using the USS protocol instructions" (Page 373) for and a listing of USS protocol instructions and error codes and example programs.

9.2.2.5 USS_WPM_x instruction

Table 9- 14 USS_WPM_x instructions

LAD / FBD	STL	Description
<pre> USS_WPM_W EN XMT_REQ EEPROM Drive Param Index Value DB_Ptr </pre>	<pre> CALL USS_WPM_W, XMT_REQ, EEPROM, Drive, Param, Index, Value, DB_Ptr, Done, Error CALL USS_WPM_D, XMT_REQ, EEPROM, Drive, Param, Index, Value, DB_Ptr, Done, Error CALL USS_WPM_R, XMT_REQ, EEPROM, Drive, Param, Index, Value, DB_Ptr, Done, Error </pre>	<p>There are three write instructions for the USS protocol:</p> <ul style="list-style-type: none"> • USS_WPM_W instruction writes an unsigned word parameter. • USS_WPM_D instruction writes an unsigned double word parameter. • USS_WPM_R instruction writes a floating-point parameter. <p>Only one read (USS_RPM_x) or write (USS_WPM_x) instruction can be active at a time.</p>

The USS_WPM_x transactions complete when the drive acknowledges receipt of the command or when an error condition is posted. The logic scan continues to execute while this process awaits a response.

Table 9- 15 Valid operands for the USS_WPM_x instructions

Inputs/outputs	Data type	Operands
XMT_REQ	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow conditioned by a rising edge detection element
EEPROM	BOOL	I, Q, M, S, SM, T, C, V, L, Power Flow
Drive	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD, Constant
Param, Index	WORD	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AIW, *VD, *AC, *LD, Constant
DB_Ptr	DWORD	&VB
Value	WORD	VW, IW, QW, MW, SW, SMW, LW, T, C, AC, AQW, *VD, *AC, *LD
	DWORD, REAL	VD, ID, QD, MD, SD, SMD, LD, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC. *VD, *AC, *LD

The "EN" bit must be on to enable transmission of a request, and should remain on until the "Done" bit is set, signaling completion of the process. For example, a USS_WPM_x request is transmitted to the drive on each scan when "XMT_REQ" input is on. Therefore, the "XMT_REQ" input should be pulsed on through an edge detection element which causes one request to be transmitted for each positive transition of the "EN" input.

Table 9- 16 USS_WPM_x parameter descriptions

Parameter	Description
XMT_REQ (transmit request)	When ON, a USS_WPM_x request is transmitted to the drive on every scan.
EEPROM	This input enables writing to both RAM and EEPROM of the drive when it is on and only to the RAM when it is off.
Drive	Address of the drive to which the USS_WPM_x command is to be sent. Valid addresses of individual drives are 0 to 31.
Param	Parameter number
Index	Index value of the parameter that is to be written
Value	Parameter value to be written to the RAM in the drive.
DB_Ptr	The address of a 16-byte buffer must be supplied to the "DB_Ptr" input. This buffer is used by the USS_RPM_x instruction to store the results of the command issued to the drive.
Done	Turned on when the USS_RPM_x instruction completes
Error	This output byte contains the result of executing the instruction. The USS protocol execution error codes (Page 383) define the error conditions that could result from executing the instruction.

When the USS_WPM_x instruction completes, the "Done" output is turned on and the "Error" output byte contains the result of executing the instruction. The "Error" output is not valid until the "Done" output turns on.

EEPROM

When the "EEPROM" input is turned on, the instruction writes to both the RAM and the EEPROM of the drive. When the input is turned off, the instruction writes only to the RAM of the drive.

NOTICE

Do not exceed the maximum number of write cycles to the EEPROM

When you use an USS_WPM_x instruction to update the parameter set stored in drive EEPROM, you must ensure that the maximum number of write cycles (approximately 50,000) to the EEPROM is not exceeded.

Exceeding the maximum number of write cycles will result in corruption of the stored data and subsequent data loss, and possible property damage. The number of read cycles is unlimited.

Do not exceed the maximum number of write cycles to the EEPROM.

USS_RPM_x and USS_WPM_x example program

Table 9- 17 USS_RPM_x and USS_WPM_x example program

Network 1 	Network 1 <pre> LD M10.5 = L60.0 LD M10.5 EU = L63.7 LD L60.0 CALL USS_RPM_W, L63.7, 0, 5, 0, &VB20, M1.1, VB10, VW12 </pre>
Network 2 	Network 2 <pre> LD M10.6 = L60.0 LD M10.6 EU = L63.7 LDN SM0.0 = L63.6 LD L60.0 CALL USS_WPM_W, L63.7, L63.6, 0, 2000, 0, 50.0, &VB40, M1.2, VB14 </pre>

Refer to "Using the USS protocol instructions" (Page 373) for and a listing of USS protocol instructions and error codes and example programs.

9.2.2.6 USS protocol execution error codes

Table 9- 18 USS protocol execution error codes

Error code	Description
0	No error
1	Drive did not respond.
2	A checksum error in the response from the drive was detected.
3	A parity error in the response from the drive was detected.
4	An error was caused by interference from the user program.
5	An illegal command was attempted.
6	An illegal drive address was supplied.
7	The communications port was not set up for USS protocol.

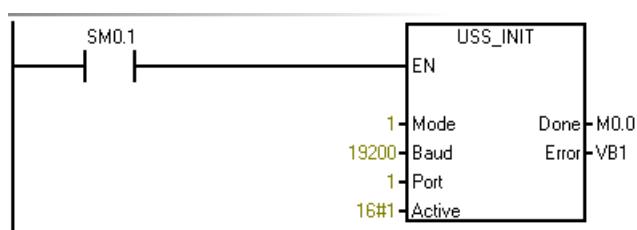
Error code	Description
8	The communications port is busy processing an instruction.
9	The drive speed input is out-of-range.
10	The length of the drive response is incorrect.
11	The first character of the drive response is incorrect.
12	The length character in the drive response is not supported by USS instructions.
13	The wrong drive responded.
14	The DB_Ptr address supplied is incorrect.
15	The parameter number supplied is incorrect.
16	An invalid protocol was selected.
17	USS is active; change is not allowed.
18	An illegal baud rate was specified.
19	No communications: the drive is not ACTIVE.
20	The parameter or value in the drive response is incorrect or contains an error code.
21	A double word value was returned instead of the word value requested.
22	A word value was returned instead of the double word value requested.
23	Invalid port number
24	Signal board (SB) port 1 is missing or not configured.

Refer to "Using the USS protocol instructions" (Page 373) for and a listing of USS protocol instructions and error codes and example programs.

9.2.2.7 USS protocol example program

Table 9- 19 Sample USS program

Network 1



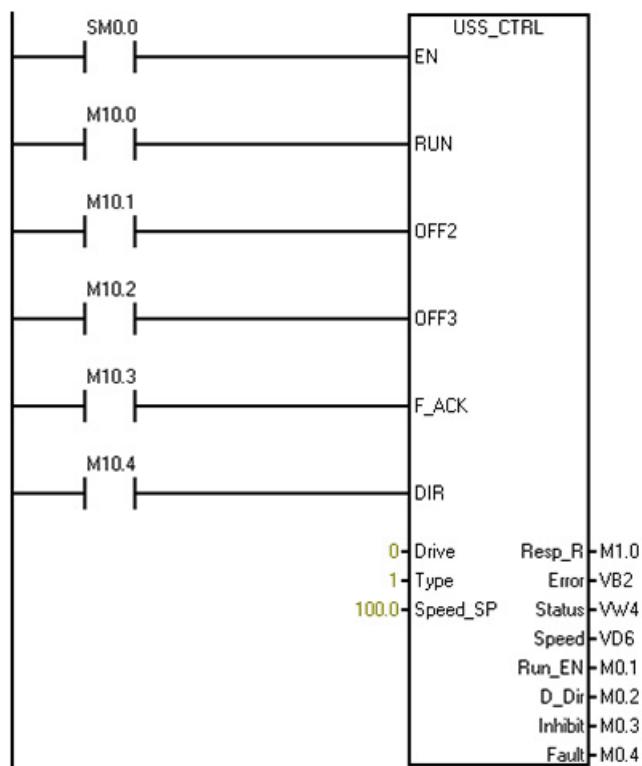
Network 1:

Initialize USS protocol: On the first scan, enable USS protocol for port 1 at 19200 with drive address "0" active.

```

LD SMO.1
CALL USS_INIT, 1, 19200, 16#00000001,
Q0.0, VB1
    
```

Network 2



Network 2:

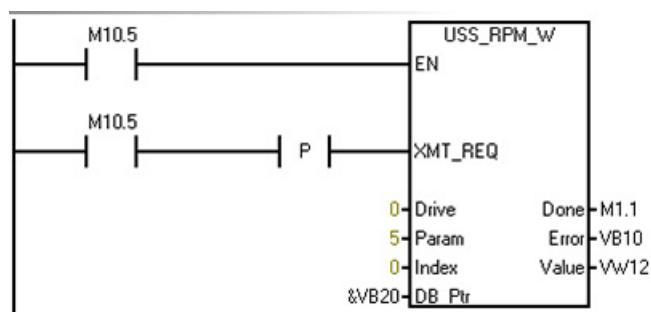
Control parameters for Drive 0

```

LD SM0.0
CALL USS_CTRL, M10.0, M10.1, M10.2,
M10.3, M10.4, 0, 1, 100.0, M1.0, VB2,
VW4, VD6, M0.1, M0.2, M0.3, M0.4

```

Network 3



Network 3:

Read a Word parameter from Drive 0.

Read parameter 5, index 0:

1. Save the state of M10.5 to a temporary location so that this network displays in LAD.
2. Save the rising edge pulse of I0.5 to a temporary L location so that it can be passed to the subroutine.

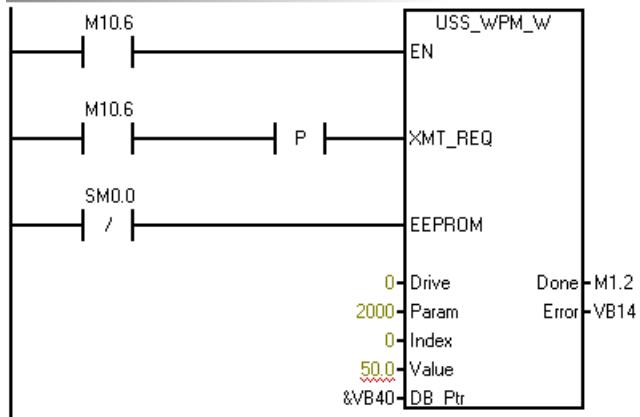
```

LD M10.5
= L60.0
LD M10.5
EU
= L63.7
LD L60.0
CALL USS_RPM_W, L63.7, 0, 5, 0, &VB20,
M1.1, VB10, VW12

```

9.3 Modbus library instructions

Network 4



Network 4:

Write a Word parameter to Drive 0. Write parameter 2000, index 0.

Note: This STL code does not compile to LAD or FBD.

```
LD M10.6
= L60.0
LD M10.6
EU
= L63.7
LDN SMO.0
= L63.6
LD L60.0
CALL USS_WPM_R, L63.7, L63.6, 0, 2000,
0, 50.0, &VB40, M1.2, VB14
```

Refer to "Using the USS protocol instructions" (Page 373) for and a listing of USS protocol instructions and error codes and example programs.

9.3 Modbus library instructions

9.3.1 Modbus communication overview

9.3.1.1 Modbus library features

When you install STEP 7-Micro/WIN SMART, the Siemens Modbus library is also installed. The Modbus library makes communicating to Modbus RTU master and slave devices easier by including pre-configured subroutines and interrupt routines that are designed for Modbus RTU communication.

Modbus communication over RS-485 (integrated port 0 and optional signal board port 1) and RS-232 (optional signal board port 1 only) is supported for both master and slave devices.

Modbus master instructions can configure the S7-200 SMART to act as a Modbus RTU master device and communicate to one or more Modbus RTU slave devices.

Modbus slave instructions can configure the S7-200 SMART to act as a Modbus RTU slave device and communicate with Modbus RTU master devices.

Open the libraries folder in the instruction branch of the project tree, for access to the Modbus instructions. When you place a Modbus instruction in your program, one or more associated POU's (subroutines and interrupt routines) are automatically added to your project.

Modbus RTU master protocol

Modbus master instructions use the following resources from the CPU:

- MBUS_CTRL execution initializes the Modbus master protocol and dedicates the assigned CPU port (0 or 1), for Modbus master communication.
When a CPU port is used for Modbus communications, it cannot be used for any other purpose, including communication with an HMI.
- Modbus master instructions affect all of the SM locations associated with Freeport communications on the port assigned by the MBUS_CTRL instruction.
- Modbus master instructions use interrupts for some functions. These interrupts must not be disabled by the user program.
- Modbus master instructions program size
 - 3 subroutines and 1 interrupt routine
 - 1942 bytes of program space for two master instructions and support routines
 - Variables for Modbus master instructions require a 286 byte block of V memory. The starting address for this block is assigned by the user and is reserved for Modbus variables.

Note

To change the CPU communication port from Modbus back to PPI so that you can communicate with an HMI device, set the mode parameter of the MBUS_CTRL instruction to a zero (0).

Modbus RTU slave protocol

Modbus slave protocol instructions use the following resources from the CPU:

- The MBUS_INIT instruction initializes the Modbus slave protocol and dedicates the assigned CPU port (0 or 1), for Modbus slave communication.

When a CPU port is used for Modbus communication, it cannot be used for any other purpose, including communications with an HMI.

- Modbus slave instructions affect all of the SM locations associated with a Freeport communications on the port assigned by the MBUS_INIT instruction.
- Modbus slave instructions program size
 - Modbus slave instructions use 3 subroutines and 2 interrupts.
 - Modbus slave instructions require 2113 bytes of program space for the two slave instructions and support routines.
 - The variables for the Modbus slave instructions require a 786 byte block of V memory. The starting address for this block is assigned by the user and is reserved for Modbus variables.

Note

To change the CPU communication port from Modbus back to PPI so that you can communicate with an HMI device, set the mode parameter of the MBUS_INIT instruction to a zero (0).

9.3.1.2 Initialization and execution time for Modbus protocol

- **Modbus RTU master protocol:** The master protocol requires a small amount of time every scan to execute the MBUS_CTRL instruction. The time will be about .2 milliseconds when the MBUS_CTRL is initializing the Modbus master (first scan), and about 0.1 milliseconds on subsequent scans.

The scan time is extended when the MBUS_MSG instruction executes a request. Most of the time is spent calculating the Modbus CRC for the request and response. The CRC (Cyclic Redundancy Check) insures the integrity of the communications message. The PLC scan time is extended by about 86 microseconds for each word in request and in the response. A maximum request/response (read or write of 120 words) extends the scan time to approximately 10.3 milliseconds. A read request extends the scan mainly when the response is received from a slave, and to a lesser extent when the request is sent. A write request extends the scan mainly when the data is sent to a slave, and to a lesser extent when the response is received.

- **Modbus RTU slave protocol:** Modbus communications uses a CRC (cyclic redundancy check) to insure the integrity of the communications messages. The Modbus slave protocol uses a table of pre-calculated values to decrease the time required to process a message. The initialization of this CRC table requires about 11.3 milliseconds. This initialization is done inside the MBUS_INIT instruction and is normally done in the first scan of the user program after entering RUN mode. You are responsible for resetting the watchdog timer, if the time required by the MBUS_INIT instruction and any other user initialization exceeds the 500 millisecond scan watchdog time. The output module watchdog timer is reset by writing to the outputs of the module.

The scan time is extended when the MBUS_SLAVE subroutine services a request. Since most of the time is spent calculating the Modbus CRC, the scan time is extended by about 40 microseconds for every byte in the request and in the response. A maximum request/response (read or write of 120 words) extends the scan time by approximately 4.8 milliseconds.

9.3.1.3 Modbus addressing

Modbus addresses are normally written as 5 character values containing the data type and the offset. The first character determines the data type and the last four characters contain the value.

Modbus master addressing

Modbus master instructions map the address to the correct functions to send to the slave device. The following Modbus addresses are supported by the Modbus master instructions:

- 00001 to 09999 are discrete outputs (coils)
- 10001 to 19999 are discrete inputs (contacts)
- 30001 to 39999 are input registers (generally analog inputs)
- (40001 to 49999) and (400001 to 465535) are holding registers

All Modbus addresses are one-based, meaning that the first data value starts at address one. The actual range of valid addresses will depend on the slave device. Different slave devices will support different data types and address ranges.

Modbus slave addressing

The Modbus master device maps the addresses to the correct functions. The following addresses are supported by the Modbus slave instructions:

- 00001 to 00256 are discrete outputs mapped to Q0.0 - Q31.7
- 10001 to 10256 are discrete inputs mapped to I0.0 - I31.7
- 30001 to 30056 are analog input registers mapped to AIW0 - AIW110
- 40001 to 49999 and 400001 to 465535 are holding registers mapped to V memory.

Mapping Modbus addresses to CPU addresses

All Modbus addresses are one-based.

Table 9- 20 Mapping Modbus addresses to CPU addresses

Modbus address	CPU address
00001	Q0.0
00002	Q0.1
00003	Q0.2
...	...
00255	Q31.6
00256	Q31.7
10001	I0.0
10002	I0.1
10003	I0.2
...	...
10255	I31.6
10256	I31.7
30001	AIW0
30002	AIW2
30003	AIW4
...	...
30056	AIW110
40001	Vx (Holding reg. start)
40002	Vx+2 =(Hold reg. start+2)
40003	Vx+4 =(Hold reg. start+4)
...	...
4yyyy	Vx+2(yyyy-1) or Vx+2(zzzzz-1)

MBUS_INIT parameters that limit slave accessibility

The Modbus slave protocol allows you to limit the number of inputs, outputs, analog inputs, and holding registers (V memory) that are accessible to a Modbus master.

- **MaxIQ** assigns the maximum number of discrete inputs or outputs (Is or Qs) a Modbus master is allowed to access.
- **MaxAI** assigns the maximum number of input registers (AIWs) a Modbus master is allowed to access.
- **MaxHold** assigns the maximum number of holding registers (V memory words) a Modbus master is allowed to access.

See the description of the MBUS_INIT instruction for more information on setting up the memory restrictions for the Modbus slave.

9.3.2 Modbus RTU master

9.3.2.1 Using the Modbus master instructions

Procedure

To use the Modbus RTU master instructions in your S7-200 SMART program, follow these steps:

1. Insert the MBUS_CTRL instruction in your program and execute the MBUS_CTRL on every scan. You can use the MBUS_CTRL instruction either to initiate or to change the Modbus communications parameters. When you insert the MBUS_CTRL instruction, several protected subroutines and interrupt routines are automatically added to your program.
2. Click the Memory button  from the Libraries area of the File menu ribbon strip to assign a starting address for the V-Memory that the Modbus library requires.
Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu.
3. Place one or more MBUS_MSG instructions in your program. You can add as many MBUS_MSG instructions to your program as you require, but only one of these instructions can be active at a time.
4. Connect a communications cable between the S7-200 SMART CPU port you assigned with the MBUS_CTRL port parameter and the Modbus slave devices.

NOTICE**Avoiding unwanted current flow**

Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or damage equipment.

Ensure that all equipment that is connected with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

The Modbus master instructions utilize the Modbus functions shown below to read or write a specific Modbus address. The Modbus slave device must support the Modbus function(s) required to read or write a particular Modbus address.

Table 9- 21 Required Modbus slave function support

Modbus address	Read or write	Modbus slave function required
00001 – 09999 discrete outputs	Read	Function 1
	Write	Function 5 for a single output point Function 15 for multiple output points
10001 – 19999 discrete inputs	Read	Function 2
	Write	not possible
30001 – 39999 input registers	Read	Function 4
	Write	not possible
40001 – 49999 holding registers 400001 - 465535	Read	Function 3
	Write	Function 6 for a single register Function 16 for multiple registers

9.3.2.2 MBUS_CTRL instruction (initialize master)

Table 9- 22 MBUS_CTRL instruction

LAD / FBD	STL	Description
<pre> EN +--> MBUS_CTRL +--> Mode +--> Baud +--> Parity +--> Port +--> Timeout +--> Done +--> Error </pre>	<pre> CALL MBUS_CTRL, Mode, Baud, Parity, Port, Timeout, Done, Error </pre>	<p>The MBUS_CTRL instruction is used to initialize, monitor, or to disable Modbus communications. Before the MBUS_MSG instruction can be used, the MBUS_CTRL instruction must be executed without errors. The instruction completes and the Done bit is set ON, before continuing to the next instruction. This instruction is executed on each scan when the EN input is on.</p>

The MBUS_CTRL instruction must be called every scan (including the first scan) to allow it to monitor the progress of any outstanding messages initiated with the MBUS_MSG instruction. The Modbus master protocol will not operate correctly unless MBUS_CTRL is called every scan.

Table 9- 23 Parameters for the MBUS_CTRL instruction

Parameter	Data type	Operands
Mode	BOOL	I, Q, M, S, SM, T, C, V, L
Baud	DWORD	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD
Parity, Port	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Timeout	WORD	VW, IW, QW, MW, SW, SMW, LW, AC, Constant, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

The value for the **Mode** input selects the communications protocol. An input value of 1 assigns the CPU port to Modbus protocol and enables the protocol. An input value of 0 assigns the CPU port to PPI system protocol and disables Modbus protocol.

Parameter **Parity** is set to match the parity of the Modbus slave device. All settings use one start bit and one stop bit. The allowed values are: 0 (no parity), 1 (odd parity), and 2 (even parity).

Parameter **Port** sets the physical communication port (0 = RS-485 integrated in CPU, 1 = RS-485 or RS-232 located on the optional CM01 signal board).

Parameter **Timeout** is set to the number of milliseconds to wait for the response from the slave. The Timeout value can be set anywhere in the range of 1 millisecond to 32767 milliseconds. A typical value would be 1000 milliseconds (1 second). The Timeout parameter should be set to a value large enough so that the slave device has time to respond at the selected baud rate.

The Timeout parameter is used to determine if the Modbus slave device is responding to a request. The Timeout value determines how long the Modbus Master will wait for the first character of the response after the last character of the request has been sent. The Modbus master will receive the entire response from the Modbus slave device if at least one character of the response is received within the Timeout time.

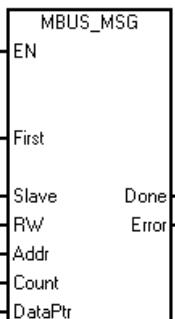
When the MBUS_CTRL instruction completes, the Done output is turned ON.

The Error output contains the result of executing the instruction.

See also Modbus master execution error codes (Page 397)

9.3.2.3 MBUS_MSG instruction

Table 9- 24 MBUS_MSG instruction

LAD / FBD	STL	Description
 <pre> MBUS_MSG EN First Slave RW Addr Count DataPtr Done Error </pre>	<pre> CALL MBUS_MSG, First, Slave, RW, Addr, Count, DataPtr, Done, Error </pre>	The MBUS_MSG instruction (or MBUS_MSG_P1 for port 1) is used to initiate a request to a Modbus slave and process the response.

The MBUS_MSG instruction initiates a master request to a Modbus slave when both the EN input and the First inputs are ON. Sending the request, waiting for the response, and processing the response usually requires several PLC scan times. The EN input must be ON to enable the send request, and should remain ON until the Done bit is set ON.

Only one MBUS_MSG instruction can be active at a time. If there is more than one MBUS_MSG instruction enabled, the first MBUS_MSG instruction executed will be processed and all subsequent MBUS_MSG instructions will abort with an error code 6.

Table 9- 25 Parameters for the MBUS_MSG instruction

Parameter	Data type	Operands
First	BOOL	I, Q, M, S, SM, T, C, V, L (Power flow conditioned by a positive edge detection element)
Slave	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
RW	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Addr	DWORD	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD
Count	INT	VW, IW, QW, MW, SW, SMW, LW, AC, Constant, *VD, *AC, *LD
DataPtr	DWORD	&VB
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

Parameter **First** should be ON for only one scan when there is a new request to send. The First input should be pulsed on through an edge detection element (for example, Positive Edge) which will cause the request to be transmitted one time. See the example program for details.

Parameter **Slave** is the address of the Modbus slave device. The allowed range is 0 through 247. Address 0 is the broadcast address and can only be used for write requests. There is no response to a broadcast request to address 0. Not all slave devices will support the broadcast address. The S7-200 SMART Modbus slave library does not support the broadcast address.

Parameter **RW** assigns if this message is to be a read or a write. The following two values are allowed for RW: 0 (Read) and 1 (Write).

Discrete outputs (coils) and holding registers support both read and write requests. Discrete inputs (contacts) and input registers only support read requests.

Parameter **Addr** is the starting Modbus address. The following ranges of values are allowed:

- 00001 to 09999 for discrete outputs (coils)
- 10001 to 19999 for discrete inputs (contacts)
- 30001 to 39999 for input registers
- 40001 to 49999 and 400001 to 465535 for holding registers

The actual range of values for Addr are based on the addresses that the Modbus slave device supports.

Parameter **Count** assigns the number of data elements to read or write in this request. The Count will be the number of bits for the bit data types, and the number of words for the word data types.

- Address 0xxxx Count is the number of bits to read or write
- Address 1xxxx Count is the number of bits to read
- Address 3xxxx Count is the number of input register words to read
- Address 4xxxx or 4yyyy Count is the number of holding register words to read or write

The MBUS_MSG instruction will read or write a maximum of 120 words or 1920 bits (240 bytes of data). The actual limit on the value of Count will depend upon the limits in the Modbus slave device.

The parameter **DataPtr** is an indirect address pointer which points to the V memory in the CPU for the data associated with the read or write request. For a read request, DataPtr should point to the first CPU memory location used to store the data read from the Modbus slave. For a write request, DataPtr should point to the first CPU memory location of the data to be sent to the Modbus slave.

The DataPtr value is passed into MBUS_MSG as an indirect address pointer. For example, if the data to be written to a Modbus slave device starts at address VW200 in the CPU, the value for the DataPtr would be &VB200 (address of VB200). Pointers must always be a type VB even if they point to word data.

Holding registers (address 4xxxx or 4yyyy) and input registers (address 3xxxx) are word values (2 bytes or 16 bits). CPU words are formatted the same as Modbus registers. The lower numbered V memory address is the most significant byte of the register. The higher numbered V memory address is the least significant byte of the register. The table below shows how the CPU byte and word addressing corresponds to the Modbus register format.

9.3 Modbus library instructions

Table 9- 26 Modbus Holding Register

CPU memory byte address		CPU memory word address		Modbus holding register address	
Address	Hex data	Address	Hex data	Address	Hex data
VB200	12	VW200	12 34	40001	12 34
VB201	34				
VB202	56	VW202	56 78	40002	56 78
VB203	78				
VB204	9A	VW204	9A BC	40003	9A BC
VB205	BC				

The bit data (addresses 0xxxx and 1xxxx) areas are read and written as packed bytes, that is, 8 bits are packed into each byte of data. The least significant bit of the first data byte is the addressed bit number (the parameter Addr). If only a single bit is written then the bit must be in the least significant bit of the byte pointed to by DataPtr.

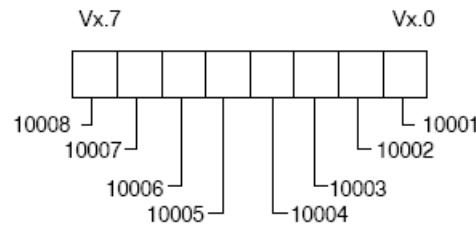
For bit data addresses that do not start on even byte boundaries, the bit corresponding to the starting address must be in the least significant bit of the byte. See the example of the packed byte format for 3 bits starting at Modbus address 10004.

When writing to the discrete output data type (coils), the user is responsible for placing the bits in the correct bit positions within the packed byte before the data is passed to the MBUS_MSG instruction via the DataPtr.

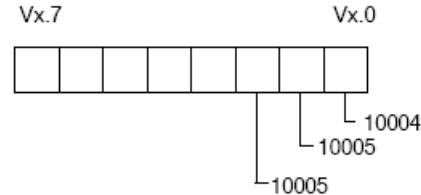
The Done output is OFF while a request is being sent and the response is being received. The Done output is ON when the response is complete or when the MBUS_MSG instruction was aborted because of an error.

The Error output is valid only when the Done output is ON.

See also Modbus master execution error codes (Page 397)



Format for Packed Bytes (Discrete Input Addresses)



Format for Packed Bytes (Discrete input starting at address 10004)

9.3.2.4 Modbus master execution error codes

The high numbered error codes (starting with 101) are errors that are returned by the Modbus slave device. These errors indicate that the slave does not support the requested function or that the requested address (either data type or range of addresses) is not supported by the Modbus slave device.

The low numbered error codes (1 through 12) are errors that are detected by the MBUS_MSG instruction. These error codes generally indicate a problem with the input parameters of the MBUS_MSG instruction, or a problem receiving the response from the slave. Parity and CRC errors indicate that there was a response but that the data was not received correctly. This is usually caused by an electrical problem such as a bad connection or electrical noise.

MBUS_CTRL error codes	Description
0	No error
1	Invalid parity type
2	Invalid baud rate
3	Invalid timeout
4	Invalid mode
9	Invalid port number
10	Signal board port 1 missing or not configured

MBUS_MSG error codes	Description
0	No error
1	Parity error in response: This is only possible if even or odd parity is used. The transmission was disturbed and possibly incorrect data was received. This error is usually caused by an electrical problem such as incorrect wiring or electrical noise affecting the communication.
2	Not used
3	Receive timeout: There was no response from the slave within the Timeout time. Some possible causes are bad electrical connections to the slave device, master and slave are set to a different baud rate / parity setting, and incorrect slave address.
4	Error in request parameter: One or more of the input parameters (Slave, RW, Addr, or Count) is set to an illegal value. Check the documentation for allowed values for the input parameters.
5	Modbus master not enabled: Call MBUS_CTRL on every scan prior to calling MBUS_MSG.
6	Modbus is busy with another request: Only one MBUS_MSG instruction can be active at a time.
7	Error in response: The response received does not correspond to the request. This indicates some problem in the slave device or that the wrong slave device answered the request.

MBUS_MSG error codes	Description
8	CRC error in response: The transmission was disturbed and possibly incorrect data was received. This error is usually caused by an electrical problem such as incorrect wiring or electrical noise affecting the communication.
11	Invalid port number
12	Signal board port 1 missing or not configured
101	Slave does not support the requested function at this address: See the required Modbus slave function support table in the "Using the Modbus master Instructions" help topic.
102	Slave does not support the data address: The requested address range of Addr plus Count is outside the allowed address range of the slave.
103	Slave does not support the data type: The Addr type is not supported by the slave device.
104	Slave device failure
105	Slave accepted the message but the response is delayed: This is an error for MBUS_MSG and the user program should resend the request at a later time.
106	Slave is busy and rejected the message: You can try the same request again to get a response.
107	Slave rejected the message for an unknown reason.
108	Slave memory parity error: There is an error in the slave device.

9.3.3 Modbus RTU slave

9.3.3.1 Using the Modbus slave instructions

Procedure

To use the Modbus slave instructions in your S7-200 SMART program, follow these steps:

1. Insert the MBUS_INIT instruction in your program and execute the MBUS_INIT instruction for one scan only. You can use the MBUS_INIT instruction either to initiate or to change the communications parameters. When you insert the MBUS_INIT instruction, several hidden subroutines and interrupt routines are automatically added to your program.
2. Click the Memory button  from the Libraries area of the File menu ribbon strip to assign a starting address for the V memory that the Modbus library requires. Alternatively, you can right-click the Program Block node in the project tree and select "Library Memory" from the context menu. In addition to this V memory block, you define another V memory block with the HoldStart and MaxHold parameters of MBUS_INIT. Be careful that your program assignments in V memory do not overlap. If there is any overlap of the memory areas, the MBUS_INIT instruction returns an error.

3. Place only one MBUS_SLAVE instruction in your program. This instruction should be called every scan to service any requests that have been received.
4. Connect a communications cable between the S7-200 SMART CPU port you assigned with the MBUS_INIT port parameter and the Modbus master device.

NOTICE

Avoiding unwanted current flow

Interconnecting equipment with different reference potentials can cause unwanted currents to flow through the interconnecting cable. These unwanted currents can cause communications errors or damage equipment.

Ensure that all equipment that is connected with a communications cable either shares a common circuit reference or is isolated to prevent unwanted current flows.

The accumulators (AC0, AC1, AC2, AC3) are used by the Modbus slave instructions and appear in the Cross Reference listing. Prior to execution, the values in the accumulators of a Modbus slave instruction are saved and restored to the accumulators before the Modbus slave instruction is complete, ensuring that all user data in the accumulators is preserved while executing a Modbus slave instruction.

The Modbus slave instructions support the Modbus RTU protocol. These instructions use the Freeport feature of the S7-200 SMART CPU to support the most common Modbus functions. The following Modbus functions are supported:

Function	Description
1	Read single/multiple coil (discrete output) status. Function 1 returns the on/off status of any number of output points (Qs).
2	Read single/multiple contact (discrete input) status. Function 2 returns the on/off status of any number of input points (Is).
3	Read single/multiple holding registers. Function 3 returns the contents of V memory. Holding registers are word values under Modbus and allow you to read up to 120 words in one request.
4	Read single/multiple input registers. Function 4 returns analog Input values.
5	Write single coil (discrete output). Function 5 sets a discrete output point to the specified value. The point is not forced and the program can overwrite the value written by the Modbus request.
6	Write single holding register. Function 6 writes a single holding register value to the V memory of the S7-200 SMART.
15	Write multiple coils (discrete outputs). Function 15 writes the discrete output values to the Q image register of the S7-200 SMART. The starting output point must begin on a byte boundary (for example, Q0.0 or Q2.0) and the number of outputs written must be a multiple of eight. This is a restriction for the Modbus slave protocol instructions. The points are not forced and the program can overwrite the values written by the Modbus request.
16	Write multiple holding registers. Function 16 writes multiple holding registers to the V memory of the S7-200 SMART. There can be up to 120 words written in one request.

9.3.3.2 MBUS_INIT instruction (initialize slave)

Table 9- 27 MBUS_INIT instruction

LAD/ FBD	STL	Description
	<pre>CALL MBUS_INIT, Mode, Addr, Baud, Parity, Port, Delay, MaxIQ, MaxAI, MaxHold, HoldStart, Done, Error</pre>	<p>The MBUS_INIT instruction enables, initializes, or disables Modbus communications. Before an MBUS_SLAVE instruction can be used, MBUS_INIT must be executed without errors. The instruction completes and the Done bit is set immediately, before continuing to the next instruction.</p> <p>The instruction is executed on each scan when the EN input is ON.</p>

The MBUS_INIT instruction should be executed exactly once for each change in communications state. Therefore, the EN input should be pulsed on through an edge detection element, or executed only on the first scan.

Table 9- 28 MBUS_INIT parameters

Inputs/outputs	Data type	Operands
Mode, Addr, Parity, Port	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, Constant, *VD, *AC, *LD
Baud, HoldStart	DWORD	VD, ID, QD, MD, SD, SMD, LD, AC, Constant, *VD, *AC, *LD
Delay, MaxIQ, MaxAI, MaxHold	WORD	VW, IW, QW, MW, SW, SMW, LW, AC, Constant, *VD, *AC, *LD
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

The value for the **Mode** input selects the communications protocol: an input value of 1 assigns Modbus protocol and enables the protocol, and an input value of 0 PPI protocol and disables Modbus protocol.

Parameter **Addr** sets the address at inclusive values between 1 and 247.

Parameter **Baud** sets the baud rate at 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

Parameter **Parity** is set to match the parity of the Modbus master. All settings use one stop bit. The accepted values are: 0 (no parity), 1 (odd parity), and 2 (even parity).

Parameter **Port** sets the physical communication port (0 = RS-485 integrated in CPU, 1 = RS-485 or RS-232 located on an optional signal board).

Parameter **Delay** extends the standard Modbus end-of-message timeout condition by adding the assigned number of milliseconds to the standard Modbus message timeout. The typical value for this parameter should be 0 when operating on a wired network. If you are using modems with error correction, set the delay to a value of 50 to 100 milliseconds. If you are using spread spectrum radios, set the delay to a value of 10 to 100 milliseconds. The Delay value can be 0 to 32767 milliseconds.

Parameter **MaxIQ** sets the number of I and Q points available to Modbus addresses 0xxxx and 1xxxx at values of 0 to 256. A value of 0 disables all reads and writes to the inputs and outputs. The suggested value for MaxIQ is 256.

Parameter **MaxAI** sets the number of word input (AI) registers available to Modbus address 3xxxx at values of 0 to 56. A value of 0 disables reads of the analog inputs. The suggested value for MaxAI to allow access to all of the CPU analog inputs, is as follows:

- 0 for CPU CR40
- 56 for all other CPU models

Parameter **MaxHold** sets the number of word holding registers in V memory available to Modbus address 4xxxx or 4yyyyy. For example, if you want to allow Modbus master access for 2000 bytes of V memory, set MaxHold to a value of 1000 words (holding registers).

Parameter **HoldStart** is the address of the start of the holding registers in V memory. This value is generally set to VB0, so the parameter HoldStart is set to &VB0 (address of VB0). Other V memory addresses can be specified as the starting address for the holding registers to allow VB0 to be used elsewhere in the project. The Modbus master has access to MaxHold number of words of V memory starting at HoldStart.

When the MBUS_INIT instruction completes, the Done output is turned ON.

The Error output byte contains the result of executing the instruction. This output is only valid if Done is ON. If Done is OFF, the error parameter is not changed.

See also Modbus slave execution error codes (Page 402)

9.3.3.3 MBUS_SLAVE instruction

Table 9- 29 MBUS_SLAVE instruction

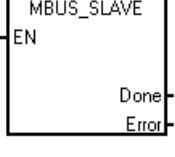
LAD / FBD	STL	Description
	CALL MBUS_SLAVE, Done, Error	<p>The MBUS_SLAVE instruction is used to service a request from the Modbus master and must be executed every scan to allow it to check for and respond to Modbus requests.</p> <p>The instruction is executed on each scan when the EN input is ON.</p> <p>The MBUS_SLAVE instruction has no input parameters.</p>

Table 9- 30 Parameters for the MBUS_SLAVE Instruction

Parameter	Data Type	Operands
Done	BOOL	I, Q, M, S, SM, T, C, V, L
Error	BYTE	VB, IB, QB, MB, SB, SMB, LB, AC, *VD, *AC, *LD

The Done output is ON when the MBUS_SLAVE instruction responds to a Modbus request. The Done output is OFF, if there was no request serviced.

The Error output contains the result of executing the instruction. This output is only valid if Done is ON. If Done is OFF, the error parameter is not changed.

9.3 Modbus library instructions

Table 9- 31 Example program of S7-200 SMART CPU operating as a Modbus slave

LAD	STL
	Network 1 LD SM0.1 CALL MBUS_INIT, 1, 1, 9600, 2, 0, 128, 32, 1000, &VBO, M0.1, MB1
	Network 2 LD SM0.0 CALL MBUS_SLAVE, M0.2, MB2

See also Modbus slave execution error codes (Page 402)

9.3.3.4 Modbus slave execution error codes

Table 9- 32

Error codes	Description
0	No error
1	Memory range error
2	Illegal baud rate or parity
3	Illegal slave address
4	Illegal value for Modbus parameter
5	Holding registers overlap Modbus Slave symbols
6	Receive parity error
7	Receive CRC error
8	Illegal function request/function not supported
9	Illegal memory address in request
10	Slave function not enabled
11	Invalid port number
12	Signal board port 1 missing or not configured

9.3.4 Modbus master example program

This example program shows how to use the Modbus Master instructions to write and read four holding registers to and from a Modbus slave each time input I0.0 is turned on.

The CPU writes four words starting at VW100 to the Modbus slave. The data is written to four holding registers in the slave starting at address 40001.

The CPU then reads four holding registers from the Modbus slave. The data comes from holding registers 40010 - 40013 and is placed into the V memory of the CPU starting at VW200.

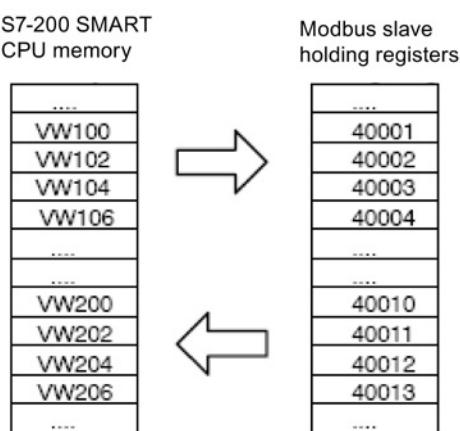


Figure 9-1 Example Program Data Transfers

The following program turns on outputs Q0.1 and Q0.2 if there is an error returned from the MBUS_MSG instruction.

Libraries

9.3 Modbus library instructions

Table 9- 33 Example Modbus master program

LAD	Description
<pre> S0.0 --- EN S0.0 --- Mode 9600 --- Baud 0 --- Parity 0 --- Port 1000 --- Timeout Done --- M0.0 Error --- MB1 </pre>	<p>Network 1:</p> <p>Initialize and monitor the Modbus master by calling MBUS_CTRL on every scan. The Modbus master is set for 9600 baud and no parity. The slave device is allowed 1000 milliseconds (1 second) to respond.</p>
<pre> SM0.1 --- M2.0 </pre>	<p>Network 2</p> <p>On the first scan, reset the enable flags (M2.0 and M2.1) used for the two MBUS_MSG instructions.</p>
<pre> I0.0 --- P --- M2.0 </pre>	<p>Network 3</p> <p>When I0.0 changes from OFF to ON, set the enable flag for the first MBUS_MSG instruction (M2.0).</p>
<pre> M2.0 --- EN M2.0 --- First 2 --- Slave 1 --- RW 40001 --- Addr 4 --- Count &VB100 --- DataPtr Done --- M0.1 Error --- MB1 </pre>	<p>Network 4</p> <p>Call the MBUS_MSG instruction when the first enable flag (M2.0) is ON. The First parameter must be set for only the first scan that the instruction is enabled.</p> <p>This instruction writes (RW = 1) 4 holding registers to slave 2. The write data is taken from VB100-VB107 (4 words) in the CPU and written to address 40001 - 40004 in the Modbus slave.</p>
<pre> M0.1 --- P --- MB1 MB1 --- Q0.1 Q0.1 --- S S --- M2.1 M2.1 --- M2.0 M2.0 --- R </pre>	<p>Network 5</p> <p>When the first MBUS_MSG instruction is complete (Done goes from 0 to 1), clear the enable for the first MBUS_MSG and set the enable for the second MBUS_MSG instruction.</p> <p>If Error (MB1) is not zero, then set Q0.1 to show the error.</p>

LAD	Description
<pre> M2.1 --- EN M2.1 --- First P +--- Slave=2 RW=0 Addr=40010 Count=4 DataPtr=&VB200 +--- Done=M0.2 +--- Error=MB1 </pre>	<p>Network 6</p> <p>Call the second MBUS_MSG instruction when the second enable flag (M2.1) is ON. The First parameter must be set for only the first scan that the instruction is enabled.</p> <p>This instruction reads (RW = 0) 4 holding registers from slave 2. The data is read from address 40010 - 40013 in the Modbus slave and copied to VB200 - VB207 (4 words) in the CPU.</p>
<pre> M0.2 --- P --- MBUS_MSG +--- Done--- Q0.2 +--- Error--- M2.1 </pre>	<p>Network 7</p> <p>When the second MBUS_MSG instruction is complete (Done goes from 0 to 1), clear the enable for the second MBUS_MSG instruction.</p> <p>If Error (MB1) is not zero, then set Q0.2 to show the error.</p>

9.3.5 Modbus advanced user information

Overview

This topic contains information for advanced users of the Modbus RTU master library. Most users should not need this information and should not modify the default operation of the Modbus RTU master library.

Retries

The Modbus master instructions automatically resends the request to the slave device if one of the following errors is detected:

- There is no response within the response timeout time (parameter Timeout on the MBUS_CTRL instruction (Error code 3).
- The time between characters of the response exceeded the allowed value (Error code 3).
- There is a parity error in the response from the slave (Error code 1).
- There is a CRC error in the response from the slave (Error code 8).
- The returned function did not match the request (Error code 7).

The Modbus Master resends the request two additional times before setting the Done and Error output parameters.

9.3 Modbus library instructions

You can change the number of retries by finding the symbol *mModbusRetries* in the Modbus master symbol table and changing this value after MBUS_CTRL has been executed. The *mModbusRetries* value is a BYTE with a range of 0 to 255 retries.

Inter-character timeout

Modbus master execution aborts a response from a slave device if the time between characters in the response exceeds a assigned time limit. The default time is set to 100 milliseconds which should allow the Modbus master instructions to work with most slave devices over wire or telephone modems. If this error is detected, the MBUS CTRL Error parameter is set to error code 3.

There may be cases where a longer time between characters is required, either because of the transmission medium (for example, telephone modem) or because the slave device itself requires more time. You can lengthen this timeout by finding the symbol *mModbusCharTimeout* in the Modbus master symbol table and changing this value after MBUS_CTRL has been executed. The *mModbusCharTimeout* value is an INT with a range of 1 to 30000 milliseconds.

Single vs. multiple bit / word write functions

Some Modbus slave devices do not support the Modbus functions to write a single discrete output bit (Modbus function 5) or to write a single holding register (Modbus function 6). These devices only support the multiple bit write (Modbus function 15) or multiple register write (Modbus function 16) instead. The MBUS_MSG instruction returns an error code 101 if the slave device does not support the single bit/word Modbus functions.

The Modbus master protocol allows you to force the MBUS_MSG instruction to use the multiple bit/word Modbus functions instead of the single bit/word Modbus functions. You can force the multiple bit/word instructions by finding the symbol *mModbusForceMulti* in the Modbus master symbol table and changing this value after MBUS_CTRL has been executed. The *mModbusForceMulti* value is an data type BOOL value and should be set to a "1" to force the use of the multiple bit/word functions when a single bit/register is written.

Accumulator usage

The accumulators (AC0, AC1, AC2, AC3) are used by the Modbus master instructions and appear in the Cross Reference listing. The values in the accumulators are saved and restored by the Modbus master instructions. All user data in the accumulators is preserved while executing the instructions.

Holding register addresses greater than 49999

Modbus holding addresses are within the range of 40001 to 49999. This range is adequate for most applications but there are some Modbus slave devices with data mapped into holding registers at a higher address range.

The MBUS_MSG instruction allows an additional range for the parameter **Addr** to support an extended range of holding register addresses at addresses 400001 to 465536.

For example: to access holding register 16768, the **Addr** parameter of MBUS_MSG should be set to 416768.

The extended addressing allows access to the full range of 65536 possible addresses supported by the Modbus protocol. This extended addressing is only supported for holding registers.

Debugging and troubleshooting

STEP 7-Micro/WIN SMART provides software tools to help you debug and test your program. These features include viewing the status of the program as it is executed by the CPU, selecting to run the CPU for a specified number of scans, and forcing values.

Use the hardware troubleshooting guide (Page 421) as a guide for determining the cause and possible solution when troubleshooting problems with the hardware.

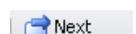
10.1 Debugging your program

10.1.1 Bookmark functions

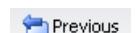
You can set bookmarks  in your program to make it easy to move back and forth between designated networks in a long program:



Toggle Bookmark: Click this button to set or remove a bookmark at the program network designated by the current cursor location.



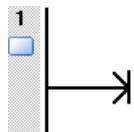
Next Bookmark: Click this button to move to the next bookmarked network of your program.



Previous Bookmark: Click this button to move to the previous bookmarked network of your program.



Remove All Bookmarks: Click this button to remove all bookmarks in your program.



10.1.2 Cross reference table

Note

You must compile your program in order to view the cross reference table.

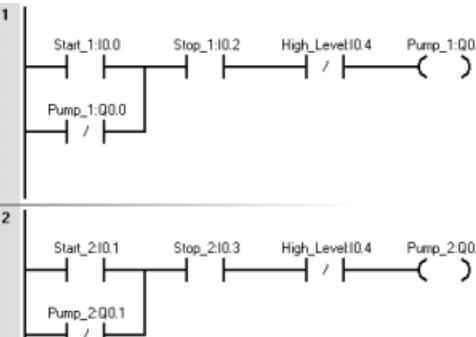
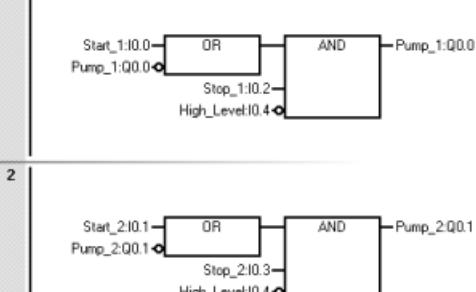
Use the cross reference table when you want to know whether a symbolic name or memory assignment is already in use in your program, and where it is used. The cross reference table identifies all operands used in the program, and identifies the POU, network or line location, and instruction context of the operand each time it is used. Double-clicking an element in the cross-reference table displays that part of your POU.

Element refers to the operands used in your program. You can use the toggle button  to toggle between symbolic and absolute addressing to change the representation of all operands.

- **Block** refers to the POU where the operand is used.
- **Location** refers to the line or network where the operand is used.
- **Context** refers to the program instruction where the operand is used.

Examples

The following examples show the cross-reference table for a simple program in all three languages: LAD, FBD, and STL.

Language	Program	Cross reference																																												
LAD	 <pre> 1 Start_1:I0.0 Stop_1:I0.2 High_Level10.4 Pump_1:Q0.0 2 Start_2:I0.1 Stop_2:I0.3 High_Level10.4 Pump_2:Q0.1 </pre>	<p>Cross Reference</p> <table border="1"> <thead> <tr> <th>Element</th> <th>Block</th> <th>Location</th> <th>Context</th> </tr> </thead> <tbody> <tr><td>1 Start_1:I0.0</td><td>MAIN (OB1)</td><td>Network 1</td><td>-I-</td></tr> <tr><td>2 Start_2:I0.1</td><td>MAIN (OB1)</td><td>Network 2</td><td>-I-</td></tr> <tr><td>3 Stop_1:I0.2</td><td>MAIN (OB1)</td><td>Network 1</td><td>-I-</td></tr> <tr><td>4 Stop_2:I0.3</td><td>MAIN (OB1)</td><td>Network 2</td><td>-I-</td></tr> <tr><td>5 High_Level10.4</td><td>MAIN (OB1)</td><td>Network 1</td><td>-I-</td></tr> <tr><td>6 High_Level10.4</td><td>MAIN (OB1)</td><td>Network 2</td><td>-I-</td></tr> <tr><td>7 Pump_1:Q0.0</td><td>MAIN (OB1)</td><td>Network 1</td><td>-()</td></tr> <tr><td>8 Pump_1:Q0.0</td><td>MAIN (OB1)</td><td>Network 1</td><td>-I-</td></tr> <tr><td>9 Pump_2:Q0.1</td><td>MAIN (OB1)</td><td>Network 2</td><td>-()</td></tr> <tr><td>10 Pump_2:Q0.1</td><td>MAIN (OB1)</td><td>Network 2</td><td>-I-</td></tr> </tbody> </table>	Element	Block	Location	Context	1 Start_1:I0.0	MAIN (OB1)	Network 1	-I-	2 Start_2:I0.1	MAIN (OB1)	Network 2	-I-	3 Stop_1:I0.2	MAIN (OB1)	Network 1	-I-	4 Stop_2:I0.3	MAIN (OB1)	Network 2	-I-	5 High_Level10.4	MAIN (OB1)	Network 1	-I-	6 High_Level10.4	MAIN (OB1)	Network 2	-I-	7 Pump_1:Q0.0	MAIN (OB1)	Network 1	-()	8 Pump_1:Q0.0	MAIN (OB1)	Network 1	-I-	9 Pump_2:Q0.1	MAIN (OB1)	Network 2	-()	10 Pump_2:Q0.1	MAIN (OB1)	Network 2	-I-
Element	Block	Location	Context																																											
1 Start_1:I0.0	MAIN (OB1)	Network 1	-I-																																											
2 Start_2:I0.1	MAIN (OB1)	Network 2	-I-																																											
3 Stop_1:I0.2	MAIN (OB1)	Network 1	-I-																																											
4 Stop_2:I0.3	MAIN (OB1)	Network 2	-I-																																											
5 High_Level10.4	MAIN (OB1)	Network 1	-I-																																											
6 High_Level10.4	MAIN (OB1)	Network 2	-I-																																											
7 Pump_1:Q0.0	MAIN (OB1)	Network 1	-()																																											
8 Pump_1:Q0.0	MAIN (OB1)	Network 1	-I-																																											
9 Pump_2:Q0.1	MAIN (OB1)	Network 2	-()																																											
10 Pump_2:Q0.1	MAIN (OB1)	Network 2	-I-																																											
FBD	 <pre> 1 Start_1:I0.0 Pump_1:Q0.0 Stop_1:I0.2 High_Level10.4 Pump_1:Q0.0 2 Start_2:I0.1 Pump_2:Q0.1 Stop_2:I0.3 High_Level10.4 Pump_2:Q0.1 </pre>	<p>Cross Reference</p> <table border="1"> <thead> <tr> <th>Element</th> <th>Block</th> <th>Location</th> <th>Context</th> </tr> </thead> <tbody> <tr><td>1 Start_1:I0.0</td><td>MAIN (OB1)</td><td>Network 1</td><td>OR</td></tr> <tr><td>2 Start_2:I0.1</td><td>MAIN (OB1)</td><td>Network 2</td><td>OR</td></tr> <tr><td>3 Stop_1:I0.2</td><td>MAIN (OB1)</td><td>Network 1</td><td>AND</td></tr> <tr><td>4 Stop_2:I0.3</td><td>MAIN (OB1)</td><td>Network 2</td><td>AND</td></tr> <tr><td>5 High_Level10.4</td><td>MAIN (OB1)</td><td>Network 1</td><td>AND</td></tr> <tr><td>6 High_Level10.4</td><td>MAIN (OB1)</td><td>Network 2</td><td>AND</td></tr> <tr><td>7 Pump_1:Q0.0</td><td>MAIN (OB1)</td><td>Network 1</td><td>AND</td></tr> <tr><td>8 Pump_1:Q0.0</td><td>MAIN (OB1)</td><td>Network 1</td><td>OR</td></tr> <tr><td>9 Pump_2:Q0.1</td><td>MAIN (OB1)</td><td>Network 2</td><td>AND</td></tr> <tr><td>10 Pump_2:Q0.1</td><td>MAIN (OB1)</td><td>Network 2</td><td>OR</td></tr> </tbody> </table>	Element	Block	Location	Context	1 Start_1:I0.0	MAIN (OB1)	Network 1	OR	2 Start_2:I0.1	MAIN (OB1)	Network 2	OR	3 Stop_1:I0.2	MAIN (OB1)	Network 1	AND	4 Stop_2:I0.3	MAIN (OB1)	Network 2	AND	5 High_Level10.4	MAIN (OB1)	Network 1	AND	6 High_Level10.4	MAIN (OB1)	Network 2	AND	7 Pump_1:Q0.0	MAIN (OB1)	Network 1	AND	8 Pump_1:Q0.0	MAIN (OB1)	Network 1	OR	9 Pump_2:Q0.1	MAIN (OB1)	Network 2	AND	10 Pump_2:Q0.1	MAIN (OB1)	Network 2	OR
Element	Block	Location	Context																																											
1 Start_1:I0.0	MAIN (OB1)	Network 1	OR																																											
2 Start_2:I0.1	MAIN (OB1)	Network 2	OR																																											
3 Stop_1:I0.2	MAIN (OB1)	Network 1	AND																																											
4 Stop_2:I0.3	MAIN (OB1)	Network 2	AND																																											
5 High_Level10.4	MAIN (OB1)	Network 1	AND																																											
6 High_Level10.4	MAIN (OB1)	Network 2	AND																																											
7 Pump_1:Q0.0	MAIN (OB1)	Network 1	AND																																											
8 Pump_1:Q0.0	MAIN (OB1)	Network 1	OR																																											
9 Pump_2:Q0.1	MAIN (OB1)	Network 2	AND																																											
10 Pump_2:Q0.1	MAIN (OB1)	Network 2	OR																																											
STL	<pre> 1 LD Start_1:I0.0 ON Pump_1:Q0.0 A Stop_1:I0.2 AN High_Level10.4 = Pump_1:Q0.0 2 LD Start_2:I0.1 ON Pump_2:Q0.1 A Stop_2:I0.3 AN High_Level10.4 = Pump_2:Q0.1 </pre>	<p>Cross Reference</p> <table border="1"> <thead> <tr> <th>Element</th> <th>Block</th> <th>Location</th> <th>Context</th> </tr> </thead> <tbody> <tr><td>1 Start_1:I0.0</td><td>MAIN (OB1)</td><td>Network 1, Line 1</td><td>LD</td></tr> <tr><td>2 Start_2:I0.1</td><td>MAIN (OB1)</td><td>Network 2, Line 1</td><td>LD</td></tr> <tr><td>3 Stop_1:I0.2</td><td>MAIN (OB1)</td><td>Network 1, Line 3</td><td>A</td></tr> <tr><td>4 Stop_2:I0.3</td><td>MAIN (OB1)</td><td>Network 2, Line 3</td><td>A</td></tr> <tr><td>5 High_Level10.4</td><td>MAIN (OB1)</td><td>Network 1, Line 4</td><td>AN</td></tr> <tr><td>6 High_Level10.4</td><td>MAIN (OB1)</td><td>Network 2, Line 4</td><td>AN</td></tr> <tr><td>7 Pump_1:Q0.0</td><td>MAIN (OB1)</td><td>Network 1, Line 2</td><td>ON</td></tr> <tr><td>8 Pump_1:Q0.0</td><td>MAIN (OB1)</td><td>Network 1, Line 5</td><td>=</td></tr> <tr><td>9 Pump_2:Q0.1</td><td>MAIN (OB1)</td><td>Network 2, Line 2</td><td>ON</td></tr> <tr><td>10 Pump_2:Q0.1</td><td>MAIN (OB1)</td><td>Network 2, Line 5</td><td>=</td></tr> </tbody> </table>	Element	Block	Location	Context	1 Start_1:I0.0	MAIN (OB1)	Network 1, Line 1	LD	2 Start_2:I0.1	MAIN (OB1)	Network 2, Line 1	LD	3 Stop_1:I0.2	MAIN (OB1)	Network 1, Line 3	A	4 Stop_2:I0.3	MAIN (OB1)	Network 2, Line 3	A	5 High_Level10.4	MAIN (OB1)	Network 1, Line 4	AN	6 High_Level10.4	MAIN (OB1)	Network 2, Line 4	AN	7 Pump_1:Q0.0	MAIN (OB1)	Network 1, Line 2	ON	8 Pump_1:Q0.0	MAIN (OB1)	Network 1, Line 5	=	9 Pump_2:Q0.1	MAIN (OB1)	Network 2, Line 2	ON	10 Pump_2:Q0.1	MAIN (OB1)	Network 2, Line 5	=
Element	Block	Location	Context																																											
1 Start_1:I0.0	MAIN (OB1)	Network 1, Line 1	LD																																											
2 Start_2:I0.1	MAIN (OB1)	Network 2, Line 1	LD																																											
3 Stop_1:I0.2	MAIN (OB1)	Network 1, Line 3	A																																											
4 Stop_2:I0.3	MAIN (OB1)	Network 2, Line 3	A																																											
5 High_Level10.4	MAIN (OB1)	Network 1, Line 4	AN																																											
6 High_Level10.4	MAIN (OB1)	Network 2, Line 4	AN																																											
7 Pump_1:Q0.0	MAIN (OB1)	Network 1, Line 2	ON																																											
8 Pump_1:Q0.0	MAIN (OB1)	Network 1, Line 5	=																																											
9 Pump_2:Q0.1	MAIN (OB1)	Network 2, Line 2	ON																																											
10 Pump_2:Q0.1	MAIN (OB1)	Network 2, Line 5	=																																											

10.2 Displaying program status

10.2.1 Displaying status in the program editor

To display current data values and I/O status in the program editor, click the Program Status ON/OFF button from either the program editor toolbar  or from the Debug menu ribbon strip  **Program Status**.

Status data collection begins and shows the results of all logic operations during the execution of the program. You can also pause and resume program status collection with the Pause Status ON/OFF button from either the program editor toolbar  or from the Debug menu ribbon strip  **Pause Status**.

A status chart (Page 416) displays values at the end of scan.

Execution status coloring

- The power rail (LAD) is colored when the program is being scanned.
- Power flow or logic flow in the diagrams is indicated by coloring.
- Contacts and coils (LAD) that have power or are logically true are colored blue.

You can assign your own color choice from the Options settings of the Tool menu ribbon strip and selecting the Colors tab.

- Box instructions – The box instructions are colored when the instruction has power and the instruction successfully executes without an error.
- Green timers and counters indicate that the Timer or Counter has valid data.
- Red indicates an instruction executed with an error.
- Jump and Label instructions display in the powerflow color when active. If not active, they display in gray.
- Gray color (default assignment) indicates no power flow - the instruction not scanned (jumped over or not called), or the PLC is in STOP mode.
- Blue color for Boolean Powerflow bits (FBD only).
- LAD, FBD, and STL program editors display the value of operands and indicate powerflow as each instruction executes during the execute program phase of a scan cycle. Execution status can display intermediate data values that may be overwritten by executing subsequent program instructions. All displayed PLC data values are collected from a single program scan cycle.

Example program status in an STL program

When you start program status in STL, the program editor window is divided into a code region and a status region. You can customize the status region according to the types of values that you want to monitor.

There are three categories of values that you can monitor in STL Status:

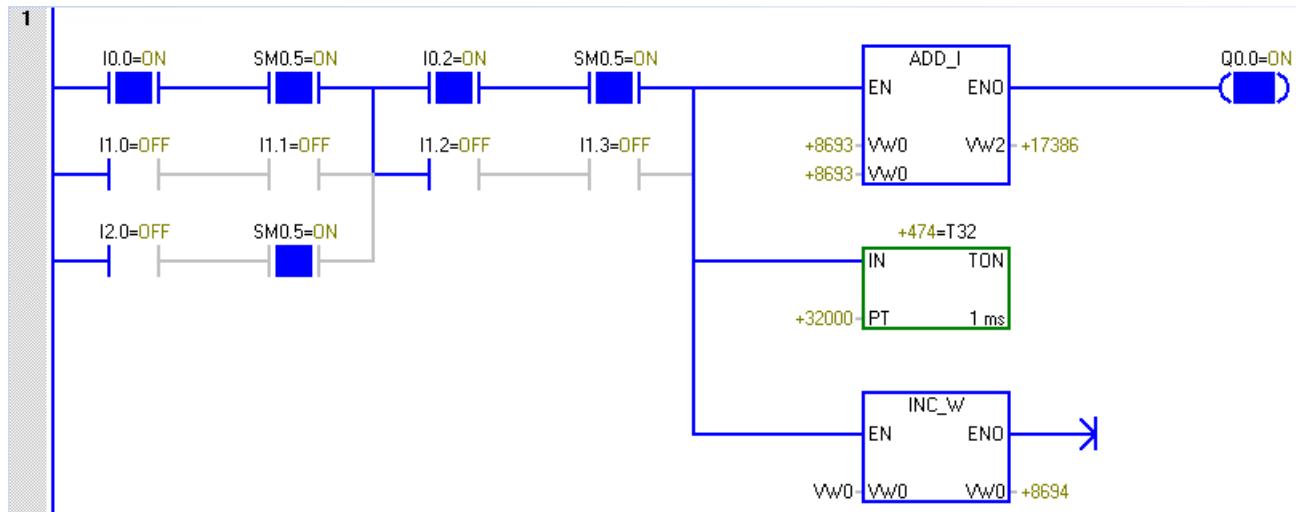
- | | |
|--------------------------------|---|
| Operands | You can monitor up to 17 operands per instruction. |
| Logic stack | You can monitor up to the four most recent values from the logic stack. |
| Instruction status bits | You can monitor up to eleven status bits. |

The STL Status tab of the Program Editor options (Page 415) in the Options settings of the Tools menu ribbon strip allows you to select or deselect any of these categories of values. If you deselect an item, it does not appear in the status display.

		Op 1	Op 2	Op 3	0123	⊕
LD	I0..0	ON			1000	1
A	SM0..5	ON			1000	1
LD	I1..0	OFF			0100	0
A	I1..1	OFF			0100	0
OLD					1000	1
LD	I2..0	OFF			0100	0
A	SM0..5	ON			0100	1
OLD					1000	1
LD	I0..2	ON			1100	1
A	SM0..5	ON			1100	1
LD	I1..2	OFF			0110	0
A	I1..3	OFF			0110	0
OLD					1100	1
ALD					1000	1
LPS					1100	1
MOWW	VW0..VW2	+8525	+8525		1100	1
AENO					1100	1
+I	VW0..VW2	+8525	+17050		1100	1
AENO					1100	1
=	Q0..0	ON			1100	1
IRD					1100	1
TON	T32..+32000	+294	+32000		1100	1
IRD					1100	1
INCW	VW0	+8526			1100	1
IRD					1100	1
INCW	VW1000	+8526			1100	1
					--	--

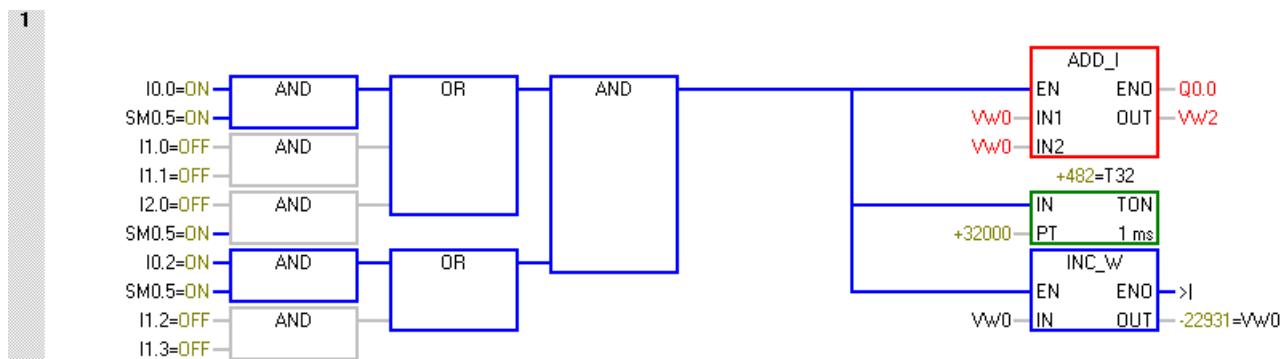
Example program status in the LAD program editor

The example below shows status in the LAD program editor. The program editor displays the values of operands and indicates powerflow as each instruction executes during the execute program phase of a scan cycle.



Example program status in the FBD program editor

The example below shows status in the FBD program editor. The red color of the ADD_I instruction box indicates errors in the operands.



10.2.2 Configuring the STL status options

To configure the STL program status display options, follow these steps:

1. Click the Options button from the Settings area of the Tools menu ribbon strip.



2. Under Options, click Program Editor > STL > Status.

3. Configure the following STL program status options:

Type	Select the font type for STL program status text.
Style	Select Regular, Italic, Bold, or Bold Italic for the text style.
Size	Select the point size for the font.
Watch Values	The check boxes and selection boxes allow you to include or remove operands, stack values, and instruction status bits (that is, flags) from the Program Status display.
Number of operands	If you chose to include operands in the Program Status display, you can edit the Operands list box to display more or fewer operands. The maximum number possible is 17.
Number of Stack Bits	If you chose to include logic stack values in the Program Status display, you can edit the Logic Stack list box to display more or fewer stack values. The maximum number possible is four.
Instruction Status Bits	If you chose to include instruction status bits in the program status display, select the Instruction Status Bits that you want to be shown and which (if any) should be omitted. A check mark indicates that you are choosing to watch a particular status bit in the program status display; if you deselect the checkbox, STEP 7-Micro/WIN SMART does not display that status bit in the Program Status.

See also

[How to display status in the program editor \(Page 412\)](#)

10.3 Using a status chart to monitor your program

In a status chart, you can enter addresses or defined symbol names to monitor or modify the status of program inputs, outputs, or variables by displaying the current values. The status chart also allows you to force or change the values of the process variables. You can create multiple status charts in order to view elements from different portions of your program.

You can display timer and counter values as either bits or words. If you display a timer or counter value as a bit, you see the output status of the instruction (0 or 1). If you display a timer or counter value as a word, you see the current value of the timer or counter.

Creating a new chart

To create a new status chart, make sure that Chart Status and Program Status is off and use one of these methods to create a new chart:

- From the project tree, right-click the Status Chart folder and select the context menu command **Insert > Chart**.
- From the Insert area of the Edit menu ribbon strip, click the down arrow beneath "Object" and select "Chart" from the drop-down menu.
- From a status chart tab in the status chart editor, or from any cell in an existing status chart, right-click and select the context menu command **Insert > Chart**.
- From the status chart toolbar, click the insert button and select "Chart".



After you successfully insert a new status chart, the new chart appears under "Status Chart" in the project tree and a new tab appears at the bottom of the Status Chart window.

Opening an existing chart

If the status chart editor is not open, you can open an existing status chart from the project tree, navigation bar, or from the Component drop-down list in the Windows area of the View menu. If the status chart editor is open, you can click a status chart tab in the editor to switch to that status chart.

Building a status chart

To build a status chart, follow these steps:

1. Enter the address (or symbol name) for each desired value in the Address field. Symbol names must be names that you have already defined in the symbol table.
2. If the element is a bit (I, Q, or M, for example), the format is set as bit in the Format column. If the element is a byte, word, or double word, select the drop-down list in the Format column and select the valid format from the available options.
3. To insert an additional row, use one of the following methods:

- Click the insert button on the status chart toolbar and select "Row".

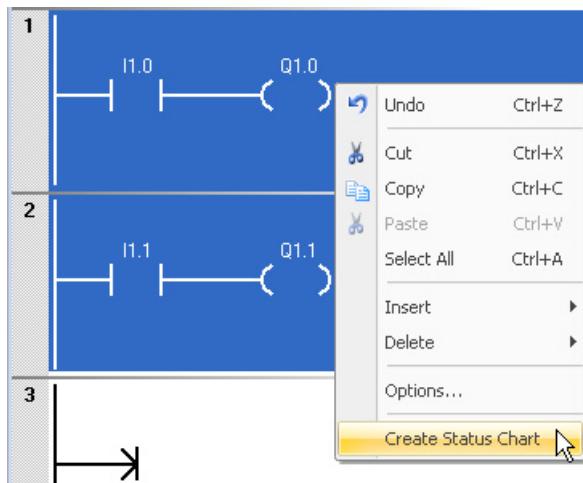


- From the Insert area of the Edit menu ribbon strip, click the "Row" button.
- Right-click a cell in the status chart to bring up a context menu, and select the menu command **Insert > Row**.

The new row is inserted above the current location of the cursor in the status chart. You can also place the cursor in the last cell of the last row and press the DOWN ARROW key to insert a row at the bottom of the status chart.

Building a status chart from a section of program code

Highlight a selection of networks in the program editor, right-click, and select "**Create Status Chart**" from the context menu. The new chart contains an entry for each unique operand in the selected region for which status can be gathered. STEP 7-Micro/WIN SMART places the entries in the order of their occurrence in the program, gives the chart a default name, and adds this chart after the last tab in the status chart editor.



When creating a chart from the program editor, note that only the first 150 addresses can be added each time you select "**Create Status Chart**". After STEP 7-Micro/WIN SMART creates the status chart, you can edit the chart entries.

You can also add an entry to a status chart by pressing the Ctrl key and dragging an operand from the LAD or FBD program editor to the status chart. From STL, you can select an address and drag it to a status chart.

Additionally, you can also copy and paste data from a Microsoft Excel spreadsheet.

Note

A project can store a maximum of 32 status charts.

Copying symbols from the symbol table to a status chart

You can copy addresses or symbol names from the symbol table and paste them into the status chart to build your chart more quickly.

10.4 Forcing specific values

The CPU allows you to force any or all of the I/O points (I and Q bits). In addition, you can also force up to 16 memory values (V or M) or analog I/O values (AI or AQ). V memory or M memory values can be forced in bytes, words, or double words. Analog values are forced as words only, on even-numbered byte boundaries, such as AIW6 or AQW14. All forced values are stored in the non-volatile memory of the CPU.

Because the forced data might be changed during the scan cycle (either by the program, by the I/O update cycle, or by the communication-processing cycle), the CPU reapplies the forced values at various times in the scan cycle.

- *Reading the inputs:* The CPU applies the forced values to the inputs as they are read.
- *Executing the control logic in the program:* The CPU applies the forced values to all immediate I/O accesses. Forced values are applied for up to 16 memory values after the program has been executed.
- *Processing any communications requests:* The CPU applies the forced values to all read/write communications accesses.
- *Writing to the outputs:* The CPU applies the forced values to the outputs as they are written.

Note

The Force function overrides a Read Immediate or Write Immediate instruction. The Force function also overrides the STOP mode values that you configured in the system block. If the CPU goes to STOP mode, the output reflects the forced value and not the STOP mode value that you configured for the output in the system block.

You can use the status chart to force values.

1. To force a new value, enter the value in the New Value column of the Status Chart, then click the Force button  on the status chart toolbar, or right-click in the New Value column and select "Force" from the context menu.
2. To force an existing value, select the value in the Current Value column and click the Force button  on the status chart toolbar, or right-click the value in the Current Value column and select "Force" from the context menu.

10.5 Writing and forcing outputs in STOP mode

To enable Write and Force functions while in STOP mode, click the "Force in Stop" button from the Settings area of the Debug menu ribbon strip.



The S7-200 SMART PLCs support writing and forcing outputs (both analog and digital) while the PLC is in STOP mode. However, as a safety precaution, you must specifically enable this functionality in STEP 7-Micro/WIN SMART with the "Force in Stop" setting.



Effect on process equipment of writing or forcing outputs

If the S7-200 SMART PLC is connected to equipment when you write or force an output, these changes can be transmitted to the equipment. This could result in unanticipated activity in the equipment, which could also cause death or serious injury to personnel, and/or property damage.

Only write or force outputs when your process equipment can safely accept those changes.

Each time you open STEP 7-Micro/WIN SMART, the menu option for "Force in Stop" defaults to unselected, and you are prevented from writing or forcing outputs while the PLC is in STOP mode. Selecting the menu option enables writing and forcing for the current editing session with the current project. When you open a different project, "Force in Stop" returns to its default state and you are prevented from either writing or forcing output addresses while the PLC is in STOP mode.

10.6 How to execute a limited number of scans

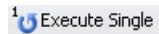
You can specify that the PLC execute your program for a limited number of scans (from 1 scan to 65,535 scans). By selecting the number of scans for the PLC to run, you can monitor the program as it changes the process variables.

On the first scan, the value of SM0.1 is one (ON).

Before executing a single scan or multiple scans, change the PLC to STOP mode (Page 33) if the PLC is not already in STOP mode.

Executing a single scan

To execute a single scan, click the "Execute Single" button from the Scan area of the Debug menu ribbon strip.



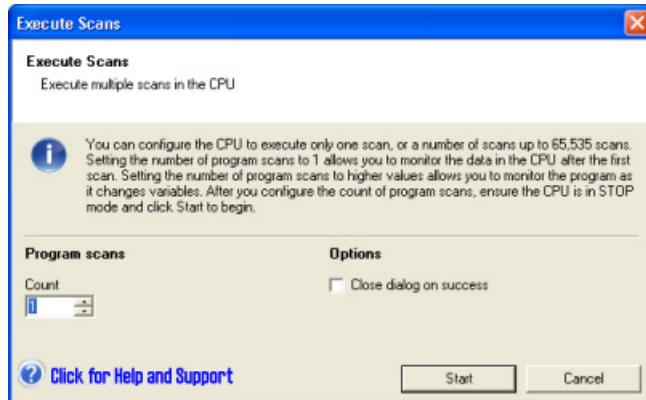
Executing multiple scans

To execute multiple scans, follow these steps:

1. Click the "Execute Multiple" button from the Scan area of the Debug menu ribbon strip.



The Execute Scans dialog box appears.



2. Enter a value for the desired number of scans, and click "Start" to execute the number of scans you entered.

Note

When you are ready to resume normal program operation, change the PLC back to RUN mode (Page 33).

See also

[Overview of debugging and monitoring features \(Page 409\)](#)

[How to display status in the editor windows \(Page 412\)](#)

[How to display status in a status chart \(Page 416\)](#)

How to download a program (Page 68)

Timestamp mismatch error (Page 592) (ensuring that project in programming device matches project in PLC)

Cross reference and element usage (Page 410) (ensuring that program edits do not cause duplicate assignments)

Forcing values (Page 418)

Forcing outputs in STOP mode (Page 419)

10.7 Hardware troubleshooting guide

Table 10- 1 Troubleshooting guide for the S7-200 SMART hardware

Symptom	Possible cause	Possible solution
Outputs stop working	The device being controlled has caused an electrical surge that damaged the output	When connecting to an inductive load (such as a motor or relay), a proper suppression circuit should be used. Refer to the wiring guidelines in Chapter 3.
	Wiring loose or incorrect	Check wiring and correct
	Excessive load	Check load against contact ratings
	Output point is forced	Check the CPU for forced I/O
ERROR light on the CPU turns on (Red)	Electrical noise	Refer to the wiring guidelines in Chapter 3. It is very important that the control panel is connected to a good ground and that high voltage wiring is not run in parallel with low voltage wiring. Connect the M terminal on the 24 VDC Sensor Power Supply to ground
	Component damage	Send in hardware for repair or replacement
None of the CPU LEDs turn on	Blown fuse	Use a line analyzer and monitor the input power to check the magnitude and duration of the over-voltage spikes. Based on this information, add the proper type surge arrestor device to your power wiring.
	Reversed 24 V power wires	Refer to the wiring guidelines in Chapter 3 for information about installing the field wiring.
	Incorrect voltage	
Intermittent operation associated with high energy devices	Improper grounding	Refer to the wiring guidelines in Chapter 3.

Symptom	Possible cause	Possible solution
	Routing of wiring within the control cabinet	It is very important that the control panel is connected to a good ground and that high voltage wiring is not run in parallel with low voltage wiring. Connect the M terminal on the 24 VDC Sensor Power Supply to ground.
	Time delay on input filters too short	Increase the input filter delay in the system data block.
Serial communications (RS-485 or RS-232) are damaged when connecting to an external device. Either the port on the external device or the port on the CPU is damaged.	The communications cable can provide a path for unwanted currents if all non-isolated devices, such as PLCs, computers, or other devices do not share the same network circuit common reference. The unwanted currents can cause communications errors or damage to electric circuits.	<ul style="list-style-type: none">Refer to the wiring guidelines in Chapter 3 and to the network guidelines in Chapter 8.Purchase network isolators or isolated RS485-to-RS485 repeaters when you connect devices that do not have a common electrical reference. <p>Refer to Appendix F for information about order numbers for S7-200 SMART equipment.</p>
Other communications problems (STEP 7-Micro/WIN SMART)		Refer to Chapter 8 for information about network communications.
Error handling		Refer to Appendix C for information about error codes.

PID loops and tuning

PID auto-tune capability is incorporated into the CPU, and STEP 7-Micro/WIN SMART adds a PID Tune control panel. Together, these two features greatly enhance the utility and ease-of-use of the PID function provided.

You can initiate auto-tune in the user program from an operator panel or by the PID Tune control panel. The PID Auto-Tuner computes suggested (near optimum) values for the gain, integral time (reset), and derivative time (rate) tuning values. You can select tuning for fast, medium, slow, or very slow response of your loop.

With the PID Tune control panel, you can initiate the auto-tuning process, abort the auto-tuning process, and monitor the results in a graphical form. The control panel displays any error conditions or warnings that might be generated. You can apply the gain, reset, and rate values computed by auto-tune.

The purpose of the PID Auto-Tuner is to determine a set of tuning parameters that provide a reasonable approximation to the optimum values for your loop. Starting with the suggested tuning values will allow you to make fine tuning adjustments and truly optimize your process. The auto-tuning algorithm used in the CPU is based upon a technique called relay feedback suggested by K. J. Åström and T. Hägglund in 1984. Over the past twenty years, relay feedback has been used across a wide variety of industries.

The relay feedback concept produces a small, but sustained oscillation in an otherwise stable process. Based upon the period of the oscillations and the amplitude changes observed in the process variable, the ultimate frequency and the ultimate gain of the process are determined. Then, using the ultimate gain and ultimate frequency values, the PID Auto-tuner suggests a value for the gain, reset, and rate tuning values.

The values suggested depend upon your selection for speed of response of the loop for your process. You can select fast, medium, slow, or very slow response. Depending upon your process, a fast response can overshoot and corresponds to an underdamped tuning condition. A medium speed response can be on the verge of overshoot and corresponds to a critically-damped tuning condition. A slow response cannot have any overshoot and corresponds to an overdamped tuning condition. A very slow response cannot have overshoot and corresponds to a heavily overdamped tuning condition.

In addition to suggesting tuning values, the PID Auto-tuner can automatically determine the values for hysteresis and peak PV deviation. You use these parameters to reduce the effect of the process noise, while limiting the amplitude of the sustained oscillations set up by the PID Auto-Tuner.

The PID Auto-Tuner can determine suggested tuning values for both direct-acting and reverse-acting P, PI, PD, and PID loops.

11.1 PID loop definition table

Eighty (80) bytes are allocated for the loop table from the starting address you enter for Table (TBL) in the PID instruction box. The PID instruction for the S7-200 SMART CPU references this loop table that contains the loop parameters.

If you use the PID Tune control panel, all interaction with the PID loop table is handled for you by the control panel. If you need to provide auto-tuning capability from an operator panel, your program must provide the interaction between the operator and the PID loop table to initiate and monitor the auto-tuning process, and then apply the suggested tuning values.

Table 11- 1 Loop table

Offset	Field	Format	Type	Description
0	Process variable (PV _n)	REAL	In	Contains the process variable, which must be scaled between 0.0 and 1.0.
4	Setpoint (SP _n)	REAL	In	Contains the setpoint, which must be scaled between 0.0 and 1.0.
8	Output (M _n)	REAL	In/Out	Contains the calculated output, scaled between 0.0 and 1.0.
12	Gain (K _c)	REAL	In	Contains the gain, which is a proportional constant. Can be a positive or negative number.
16	Sample time (T _s)	REAL	In	Contains the sample time, in seconds. Must be a positive number.
20	Integral time or reset (T _i)	REAL	In	Contains the integral time or reset, in minutes.
24	Derivative time or rate (T _D)	REAL	In	Contains the derivative time or rate, in minutes.
28	Bias (MX)	REAL	In/Out	Contains the bias or integral sum value between 0.0 and 1.0.
32	Previous process variable (PV _{n-1})	REAL	In/Out	Contains the value of the process variable stored from the last execution of the PID instruction.
36	PID Extended Table ID	ASCII	Constant	'PIDA' (PID Extended Table, Version A): ASCII constant
40	AT Control (ACNTL)	BYTE	In	See the following table
41	AT Status (ASTAT)	BYTE	Out	See the following table
42	AT Result (ARES)	BYTE	In/Out	See the following table
43	AT Config (ACNFG)	BYTE	In	See the following table
44	Deviation (DEV)	REAL	In	Normalized value of the maximum PV oscillation amplitude (range: 0.025 to 0.25).
48	Hysteresis (HYS)	REAL	In	Normalized value of the PV hysteresis used to determine zero crossings (range: 0.005 to 0.1). If the ratio of DEV to HYS is less than 4, a warning will be indicated during auto-tune.
52	Initial Output Step (STEP)	REAL	In	Normalized size of the step change in the output value used to induce oscillations in the PV (range: 0.05 to 0.4).
56	Watchdog Time (WDOG)	REAL	In	Maximum time allowed between zero crossings in seconds (range: 60 to 7200).
60	Suggested Gain (AT_Kc)	REAL	Out	Suggested loop gain as determined by the auto-tune process.

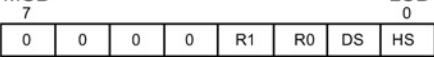
Offset	Field	Format	Type	Description
64	Suggested Integral Time (AT_T _I)	REAL	Out	Suggested integral time as determined by the auto-tune process.
68	Suggested Derivative Time (AT_T _D)	REAL	Out	Suggested derivative time as determined by the auto-tune process.
72	Actual Step size (ASTEP)	REAL	Out	Normalized output step size value as determined by the auto-tune process.
76	Actual Hysteresis (AHYS)	REAL	Out	Normalized PV hysteresis value as determined by the auto-tune process.

PID loops and tuning

11.1 PID loop definition table

Table 11-2 Expanded description of control and status fields

Field	Description	
AT Control (ACNTL) Input - Byte	MSB 7 	LSB 0
	EN	Set to 1 to start auto-tune; set to 0 to abort auto-tune
AT Status (ASTAT) Output - Byte	MSB 7 	LSB 0
	W0	Warning: The deviation setting is not four times greater than the hysteresis setting.
	W1	Warning: Inconsistent process deviations may result in incorrect adjustment of the output step value.
	W2	Warning: Actual average deviation is not four times greater than the hysteresis setting.
	AH	Auto-hysteresis calculation in progress: 0 - not in progress 1 - in progress
	IP	Auto-tune in progress: 0 - not in progress 1 - in progress
	Each time an auto-tune sequence is started the CPU clears the warning bits and sets the in progress bit. Upon completion of auto-tune, the CPU clears the in progress bit.	
AT Result (ARES) Input/Output - Byte	MSB 7 	LSB 0
	① Result code	
	D	Done bit: 0 - auto-tune not complete 1 - auto-tune complete Must be set to 0 before auto-tune can start
	Result Code	00 - completed normally (suggested tuning values available) 01 - aborted by the user 02 - aborted, watchdog timed out waiting for a zero crossing 03 - aborted, process (PV) out-of-range 04 - aborted, maximum hysteresis value exceeded 05 - aborted, illegal configuration value detected 06 - aborted, numeric error detected 07 - aborted, PID instruction executed without having power flow (loop in manual mode) 08 - aborted, auto-tuning allowed only for P, PI, PD, or PID loops 09 to 7F - reserved

Field	Description													
AT Config (ACNFG) Input - Byte	MSB 7  LSB 0													
R1	R0	Dynamic response												
0	0	Fast response												
0	1	Medium response												
1	0	Slow response												
1	1	Very slow response												
DS	Deviation setting: 0 - use deviation value from loop table 1 - determine deviation value automatically													
HS	Hysteresis setting: 0 - use hysteresis value from loop table 1 - determine hysteresis value automatically													

Note

The standard PID instruction (Page 250) is not used directly by projects with PID wizard configurations. If you use a PID wizard configuration, then your program must use "PIDx_CTRL", to activate the PID wizard subroutine.

To simplify the use of PID loop control in your application, STEP 7-Micro/WIN SMART provides a PID wizard (Page 251) to configure your PID loops.

11.2 Prerequisites

The loop that you want to auto-tune must be in automatic mode. The loop output must be controlled by the execution of the PID instruction. Auto-tune will fail if the loop is in manual mode.

Before initiating an auto-tune operation, your process must be brought to a stable state which means that the PV has reached setpoint (or for a P-type loop, a constant difference between PV and setpoint) and the output is not changing erratically.

Ideally, the loop output value needs to be near the center of the control range when auto-tuning is started. The auto-tune procedure sets up an oscillation in the process by making small step changes in the loop output. If the loop output is close to either extreme of its control range, the step changes introduced in the auto-tune procedure can cause the output value to attempt to exceed the minimum or the maximum range limit.

If this happens, the generation of an auto-tune error condition results, and the determination of less than near optimal suggested values certainly results.

11.3 Auto-hysteresis and auto-deviation

Hysteresis parameter

The hysteresis parameter specifies the excursion (plus or minus) from setpoint that the PV (process variable) is allowed to make without causing the relay controller to change the output. This value is used to minimize the effect of noise in the PV signal to more accurately determine the natural oscillation frequency of the process.

If you select to automatically determine the hysteresis value, the PID Auto-Tuner will enter a hysteresis determination sequence. This sequence involves sampling the process variable for a period of time and then performing a standard deviation calculation on the sample results.

In order to have a statistically meaningful sample, a set of at least 100 samples must be acquired. For a loop with a sample time of 200 msec, acquiring 100 samples takes 20 seconds. For loops with a longer sample time it will take longer. Even though 100 samples can be acquired in less than 20 seconds for loops with sample times less than 200 msec, the hysteresis determination sequence always acquires samples for at least 20 seconds.

Once all the samples have been acquired, the standard deviation for the sample set is calculated. The hysteresis value is defined to be two times the standard deviation. The calculated hysteresis value is written into the actual hysteresis field (AHYS) of the loop table.

Note

While the auto-hysteresis sequence is in progress, the normal PID calculation is not performed. Therefore, it is imperative that the process be in a stable state prior to initiating an auto-tune sequence. This will yield a better result for the hysteresis value and it will ensure that the process does not go out of control during the auto-hysteresis determination sequence.

Deviation parameter

The deviation parameter specifies the desired peak-to-peak swing of the PV around the setpoint. If you select to automatically determine this value, the desired deviation of the PV is computed by multiplying the hysteresis value by 4.5. The output will be driven proportionally to induce this magnitude of oscillation in the process during auto-tuning.

11.4 Auto-tune sequence

The auto-tuning sequence begins after the hysteresis and deviation values have been determined. The tuning process begins when the initial output step is applied to the loop output.

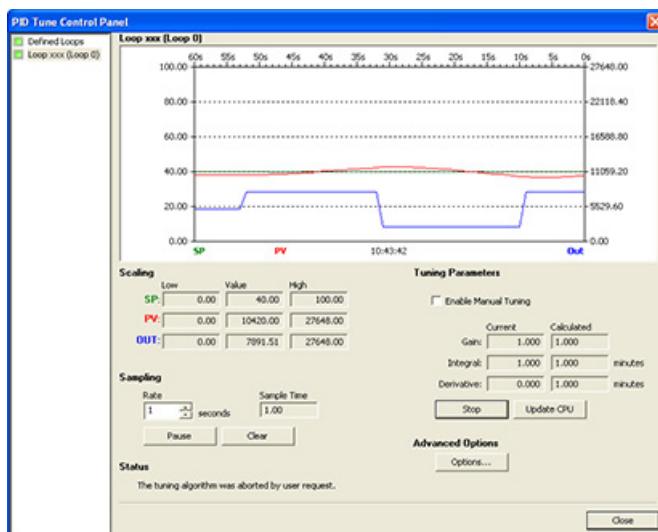
This change in output value causes a corresponding change in the value of the process variable. When the output change drives the PV away from setpoint far enough to exceed the hysteresis boundary, a zero-crossing event is detected by the auto-tuner. Upon each zero-crossing event, the auto-tuner drives the output in the opposite direction.

The tuner continues to sample the PV and waits for the next zero-crossing event. The tuner requires a total of twelve zero-crossings to complete the sequence. The magnitude of the observed peak-to-peak PV values (peak error) and the rate at which zero-crossings occur are directly related to the dynamics of the process.

Early in the auto-tuning process, the output step value is proportionally adjusted once to induce subsequent peak-to-peak swings of the PV to more closely match the desired deviation amount. Once the adjustment is made, the new output step amount is written into the Actual Step Size field (ASTEP) of the loop table.

If the time between zero-crossings exceeds the zero-crossing watchdog interval time, the auto-tuning sequence is terminated with an error. The default value for the zero-crossing watchdog interval time is two hours.

The following figure shows the output and process variable behaviors during an auto-tuning sequence on a direct acting loop. You use the PID Tune control panel to initiate and monitor the tuning sequence.



Notice how the auto-tuner switches the output to cause the process (as evidenced by the PV value) to undergo small oscillations. The frequency and the amplitude of the PV oscillations are indicative of the process gain and natural frequency.

Based upon the information collected about the frequency and gain of the process during the auto-tune process, the ultimate gain and ultimate frequency values are calculated. From these values, the suggested values for gain (loop gain), reset (integral time), and rate (derivative time) are calculated.

Note

Your loop type determines which tuning values are calculated by the auto-tuner. For example, for a PI loop, the auto-tuner will calculate gain and integral time values, but the suggested derivative time will be 0.0 (no derivative action).

Once the auto-tune sequence has completed, the output of the loop is returned to its initial value. The next time the loop is executed, the normal PID calculation will be performed.

11.5 Exception conditions

Warning conditions

Tuning execution can generate three warning conditions. Tuning execution reports these warnings in three bits of the ASTAT field of the loop table, and, once set, these bits remain set until the next auto-tune sequence is initiated:

- Warning 0: Generated if the deviation value is not at least 4X greater than the hysteresis value. This check is performed when the hysteresis value is actually known, which depends upon the auto-hysteresis setting.
- Warning 1: Generated if there is more than an 8X difference between the two peak error values gathered during the first 2.5 cycles of the auto-tune procedure
- Warning 2: Generated if the measured average peak error is not at least 4X greater than the hysteresis value

Error conditions

In addition to the warning conditions, several error conditions are possible. The following table lists the error conditions, along with a description of the cause of each error.

Table 11- 3 Error conditions during tuning execution

Result code (in ARES)		Condition
01	aborted by user	EN bit cleared while tuning is in progress
02	aborted due to a zero-crossing watchdog timeout	Half-cycle elapsed time exceeds zero-crossing watchdog interval
03	aborted due to the process out-of-range	PV goes out-of-range: <ul style="list-style-type: none"> • during the auto-hysteresis sequence, or • twice before the fourth zero-crossing, or • after the fourth zero crossing
04	aborted due to hysteresis value exceeding maximum	User-specified hysteresis value, or automatically determined hysteresis value > maximum
05	aborted due to illegal configuration value	The following range checking errors: <ul style="list-style-type: none"> • Initial loop output value is < 0.0 or > 1.0 • User-specified deviation value is <= hysteresis value, or is > maximum • Initial output step is <= 0.0 or is > maximum • Zero-crossing watchdog interval time is < minimum • Sample time value in loop table is negative
06	aborted due to a numeric error	Illegal floating point number or divide by zero encountered
07	PID instruction was executed with no power flow (manual mode)	PID instruction executed with no power flow while auto-tuning is in progress or is requested
08	auto-tuning allowed only for P, PI, PD, or PID loops	Loop type is not P, PI, PD, or PID

11.6 Notes concerning PV out-of-range (result code 3)

The process variable is considered to be in-range by the auto-tuner if its value is greater than 0.0 and less than 1.0.

If the PV is detected to be out-of-range during the auto-hysteresis sequence, then the tuning is immediately aborted with a process out-of-range error result.

If the PV is detected to be out-of-range between the starting point of the tuning sequence and the fourth zero-crossing, then the output step value is cut in half and the tuning sequence is restarted from the beginning. If a second PV out-of-range event is detected after the first zero-crossing following the restart, then the tuning is aborted with a process out-of-range error result.

Any PV out-of-range event occurring after the fourth zero-crossing results in an immediate abort of the tuning and a generation of a process out-of-range error result.

11.7 PID Tune control panel

STEP 7-Micro/WIN SMART includes a PID Tune control panel that allows you to graphically monitor the behavior of your PID loops. In addition, the control panel allows you to initiate the auto-tune sequence, abort the sequence, and apply the suggested tuning values or your own tuning values.

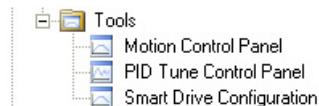
To use the control panel, you must be communicating with the CPU and a wizard-generated configuration for a PID loop must exist in the CPU. The CPU must be in RUN mode for the control panel to display the operation of a PID loop.

To open the PID control panel, use one of the following methods:

- Click the "PID Control Panel" button from the Tools area of the Tools menu ribbon strip.

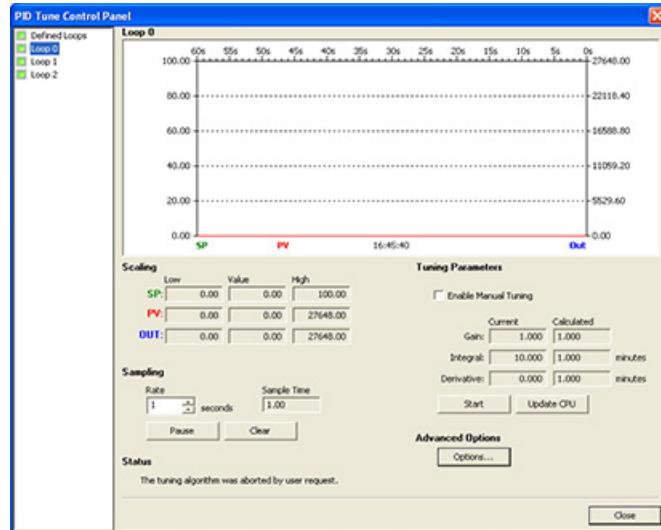


- Open the Tools folder in the project tree, select the "PID Tune Control Panel" node and press Enter; or double-click the "PID Tune Control Panel" node.



11.7 PID Tune control panel

STEP 7-Micro/WIN SMART opens the PID control panel if the connected CPU is in RUN mode:



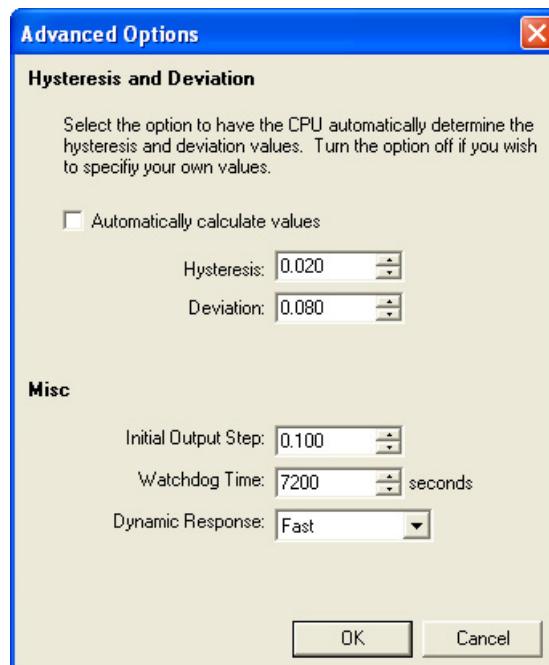
The PID control panel includes the following fields:

- Current values: The values of the SP (Setpoint), PV (Process Variable), OUT (Output), Sample Time, Gain, Integral time, and Derivative time are displayed. The SP, PV, OUT are shown in green, red, and blue, respectively; the same color legend is used to plot the PV, SP, and OUT values.
- Graphical display: The graphical display shows color-coded plots of the PV, SP, and Output as a function of time. The PV and SP share the same vertical scale which is located at the left hand side of the graph while the vertical scale for the output is located on the right hand side of the graph.
- Tuning Parameters: At the bottom left-hand side of the screen are the Tuning Parameters (Minutes). Here, the Gain, Integral Time, and Derivative Time values are displayed. You click in the "Calculated" column to modify any one of the three sources for these values.
- "Update CPU" button: You can use the "Update CPU" button to transfer the displayed Gain, Integral Time, and Derivative Time values to the CPU for the PID loop that is being monitored. You can use the "Start" button to initiate an auto-tuning sequence. Once an auto-tuning sequence has started, the "Start" button becomes a "Stop" button.
- Sampling: In the "Sampling" area, you can select the graphical display sampling rate from 1 to 480 seconds per sample.

You can freeze the graph by clicking the "Pause" button. Click the "Resume" button to resume sampling data at the selected rate. To clear the graph, select "Clear" from the right-mouse button within the graph.

- Advanced Options: You can use the "Options" button to further configure parameters for the auto-tuning process. (See the figure below.)

From the advanced screen, you can check the box that causes the auto-tuner to automatically determine the values for the Hysteresis and Deviation (default setting), or you can enter the values for these fields that minimize the disturbance to your process during the auto-tune procedure.



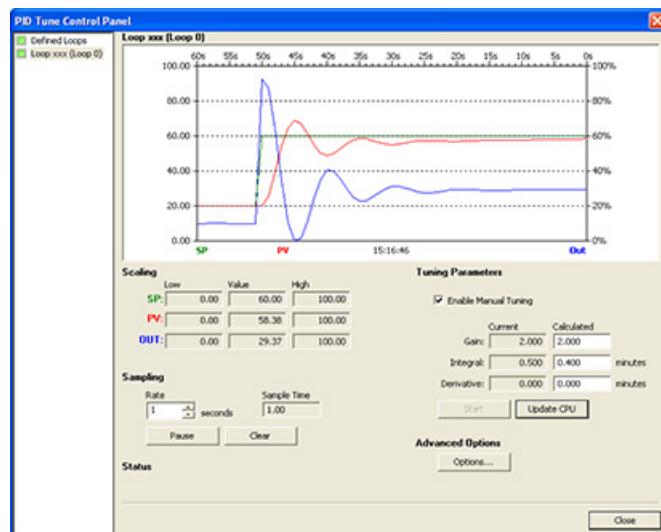
In the "Dynamic Response" field, use the dropdown button to select the type of loop response (Fast, Medium, Slow, or Very Slow) that you wish to have for your process. Depending upon your process, a fast response may have overshoot and would correspond to an underdamped tuning condition. A medium speed response may be on the verge of having overshoot and would correspond to a critically damped tuning condition. A slow response may not have any overshoot and would correspond to an overdamped tuning condition. A very slow response may not have overshoot and would correspond to a heavily overdamped tuning condition.

Once you have made the desired selections, click OK to return to the main screen of the PID Tune Control Panel.

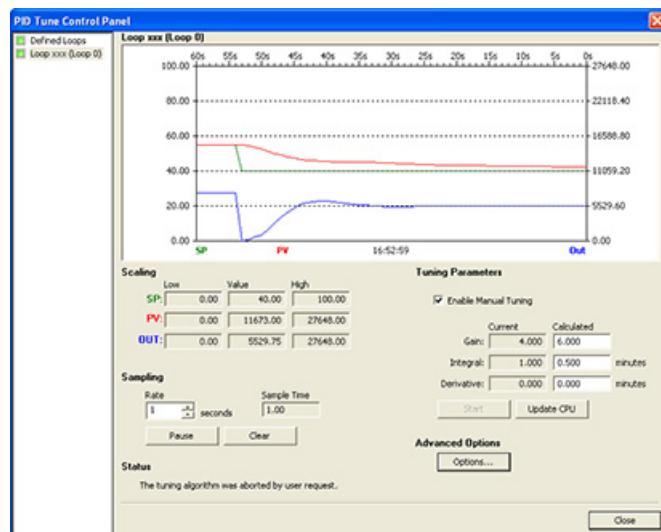
Loop monitoring

After you have completed the auto-tune sequence and have transferred the suggested tuning parameters to the CPU, you can use the control panel to monitor your loop's response to a step change in the setpoint.

In the figure below, the loop responds to a setpoint change with the original tuning parameters (before running auto-tune). Notice the overshoot and the long, damped ringing behavior of the process using the original tuning parameters.



The loop responds to the same setpoint change after applying the values determined by the auto-tune process using the selection for a fast response. Notice that for this process there is no overshoot, but there is just a little bit of ringing.



To eliminate the ringing at the expense of the speed of response, select a medium or a slow response and re-run the auto-tuning process.

After you have a good starting point for the tuning parameters for your loop, you can use the control panel to tweak the parameters. Then you can monitor the loop's response to a setpoint change. In this way, you can fine tune your process for an optimum response in your application.

To simplify the use of PID loop control in your application, STEP 7-Micro/WIN SMART provides a PID wizard (Page 251) to configure your PID loops.

Open loop motion control

The S7-200 SMART CPU provides two methods of open loop motion control:

- Pulse Width Modulation (PWM): Built into the CPU for speed, position, or duty cycle control
- Axis of Motion: Built into the CPU for speed and position control

The CPU provides three digital outputs (Q0.0, Q0.1, and Q0.3) that can be configured as PWM outputs by the PWM wizard or as motion control outputs by the Motion wizard.

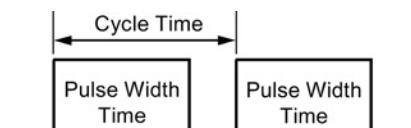
When an output is configured for PWM operation, the cycle time of the output is fixed and the pulse width or duty cycle of the pulse is controlled by your program. The variations in pulse width can be used to control the speed or position in your application.

The Axis of Motion provides a single pulse train output with integrated direction control and disable outputs. It also includes programmable inputs which allow the CPU to be configured for several modes of operation, including automatic reference point seek. The Axis of Motion provides a unified solution for open loop control of the speed and position for either stepper motors or servo motors.

To simplify the use of motion control in your application, STEP 7-Micro/WIN SMART provides a Motion wizard to configure Axis of Motion and a PWM wizard to configure PWM. The wizards generate motion instructions that you can use to provide dynamic control of speed and motion in your application. For the Axis of Motion, STEP 7-Micro/WIN SMART also provides a control panel that allows you to control, monitor, and test your motion operations.

12.1 Using the PWM output

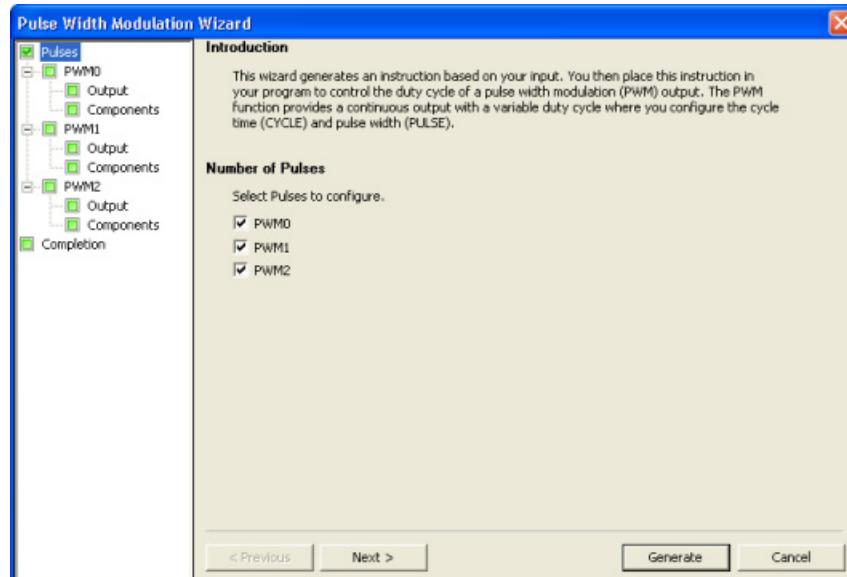
PWM provides a fixed cycle time output with a variable duty cycle. The PWM output runs continuously after being started at the specified frequency (cycle time). The pulse width is varied as required to affect the desired control. Duty cycle can be expressed as a percentage of the cycle time or as a time value corresponding to pulse width. The pulse width can vary from 0% (no pulse, always off) to 100% (no pulse, always on). See the following figure.



Since the PWM output can be varied from 0% to 100%, it provides a digital output that in many ways is analogous to an analog output. For example the PWM output can be used to control the speed of a motor from stop to full speed or it can be used to control the position of a valve from closed to full open.

12.1.1 Configuring the PWM output

To configure one of the built-in outputs for PWM control, use the PWM wizard.



Use one of the following methods to open the PWM wizard:

- Click the "PWM" button from the Wizards area of the Tools menu.



- Open the Wizards folder in the project tree and double-click "PWM", or select "PWM" and press the Enter key.
 1. Select a pulse generator.
 2. Change the name of a PWM channel, if required.
 3. Configure the PWM channel output time base.
 4. Generate project components.
 5. Use the PWMx_RUN subroutine to control the duty cycle of your PWM output.

Note

PWM channels are hard-coded to specific outputs:

- PWM0 is assigned to Q0.0.
- PWM1 is assigned to Q0.1.
- PWM2 is assigned to Q0.3.

12.1.2 PWMx_RUN subroutine

LAD / FBD	STL	Description
<pre> Pwm0_Run ----- EN ----- RUN ----- Cycle Pulse ----- Error </pre>	CALL PWMx_RUN, Cycle, Pulse, Error	The PWMx_RUN subroutine allows you to control the duty cycle of the output by varying the pulse width from 0 to the pulse width of the cycle time.

Table 12- 1 Parameters for the PWMx_RUN subroutine

Inputs/Outputs	Data types	Operands
Cycle, Pulse	Word	IW, QW, VW, MW, SMW, SW, T, C, LW, AC, AIW, *VD, *AC, *LD, Constant
Error	Byte	IB, QB, VB, MBV, SMB, LB, AC, *VD, *AC, *LD, Constant

The Cycle input is a word value that defines the cycle time for the pulse width modulation (PWM) output. The allowed range is from 2 to 65535 when milliseconds is the timebase and 10 to 65535 when microseconds is the timebase.

The Pulse input is a word value that defines the pulse width (Duty Cycle) for the PWM output. The allowed range of values is from 0 to 65535 units of the time base (microseconds or milliseconds) that was specified within the PWM wizard.

The Error is a byte value returned by the PWMx_RUN subroutine that indicates the result of execution. See the following table for a description of the possible error codes.

Table 12- 2 PWMx_RUN instruction error codes

Error code	Description
0	No error, normal completion
131	Pulse generator already in use by another PWM or by a motion axis, or illegal timebase change

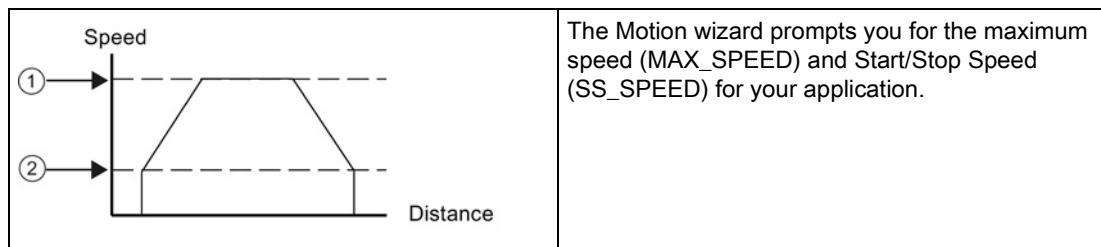
To simplify the use of PWM control in your application, STEP 7-Micro/WIN SMART provides a PWM wizard (Page 438) to configure your on-board PWM generators and control the duty cycle of a PWM output.

12.2 Using motion control

The motion control built into the CPU uses an Axis of Motion to control both the speed and motion of a stepper motor or a servo motor.

Using the Axis of Motion requires expertise in the field of motion control. This chapter is not meant to educate the novice in this subject. However, it provides fundamental information that will help as you use the Motion wizard to configure the Axis of Motion for your application.

12.2.1 Maximum and start/stop speeds

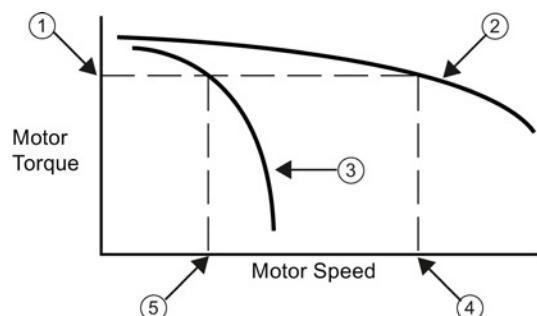


① MAX_SPEED

② SS_SPEED

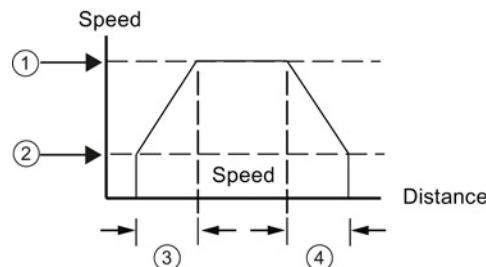
- MAX_SPEED: Enter the value for the optimum operating speed of your application within the torque capability of your motor. The torque required to drive the load is determined by friction, inertia, and the acceleration/deceleration times.
- The Motion wizard calculates and displays the minimum speed that can be controlled by the Axis of Motion based upon the MAX_SPEED that you specify.
- SS_SPEED: Enter a value within the capability of your motor to drive your load at low speeds. If the SS_SPEED value is too low, the motor and load could vibrate or move in short jumps at the beginning and end of travel. If the SS_SPEED value is too high, the motor could lose pulses on start up, and the load could overdrive the motor when attempting to stop.

Motor data sheets have different ways of specifying the start/stop (or pull-in/pull-out) speed for a motor and given load. Typically, a useful SS_SPEED value is 5% to 15% of the MAX_SPEED value. To help you select the correct speeds for your application, refer to the data sheet for your motor. The following figure shows a typical motor torque/speed curve.



- ① Torque required to drive the load
- ② Motor torque versus speed characteristic
- ③ Start/Stop speed versus torque: This curve moves towards lower speed as the load inertia increases.
- ④ Maximum speed that the motor can drive the load: MAX_SPEED should not exceed this value.
- ⑤ Start/Stop speed (SS_SPEED) for this load

12.2.2 Entering the acceleration and deceleration times



As part of the configuration, you set the acceleration and deceleration times. The default setting for both the acceleration time and the deceleration time is 1 second. Typically, motors can work with less than 1 second. See the following figure.

- ① MAX_SPEED
- ② SS_SPEED
- ③ ACCEL_TIME
- ④ DECEL_TIME

Note

Motor acceleration and deceleration times are determined by trial and error. You should start by entering a large value. Optimize these settings for the application by gradually reducing the times until the motor starts to stall.

Specify the following times in milliseconds:

- ACCEL_TIME: Time required for the motor to accelerate from SS_SPEED to MAX_SPEED. Default = 1000 ms
- DECEL_TIME: Time required for the motor to decelerate from MAX_SPEED to SS_SPEED. Default = 1000 ms

12.2.3 Configuring the motion profiles

A profile is a pre-defined motion description consisting of one or more speeds of movement that effect a change in position from a starting point to an ending point. You do not have to define a profile in order to use the Axis of Motion. The Motion wizard provides instructions for you to use to control moves without running a profile.

A profile is programmed in steps consisting of an acceleration/deceleration to a target speed followed by a fixed number of pulses at the target speed. In the case of single step moves or the last step in a move, there is also a deceleration from the target speed (last target speed) to stop.

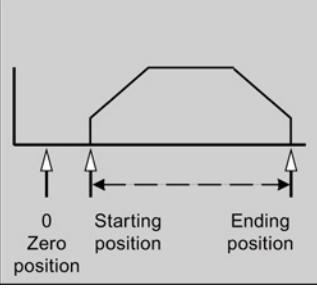
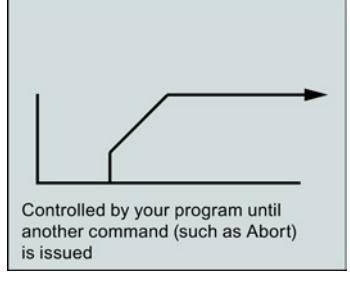
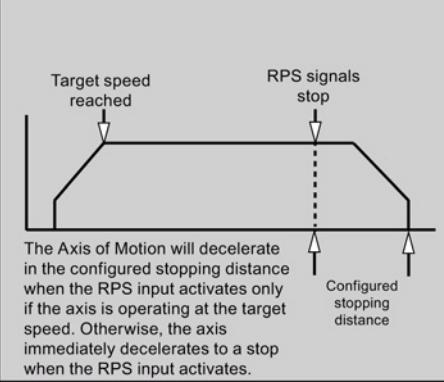
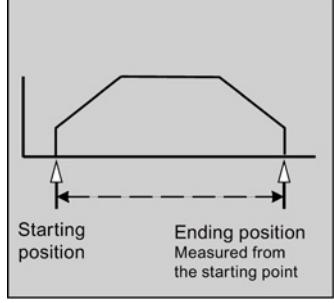
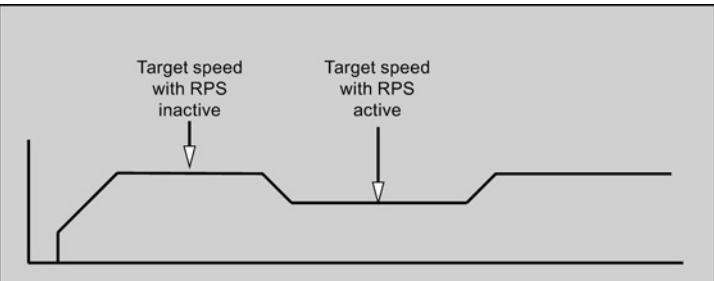
The Axis of Motion supports a maximum of 32 profiles.

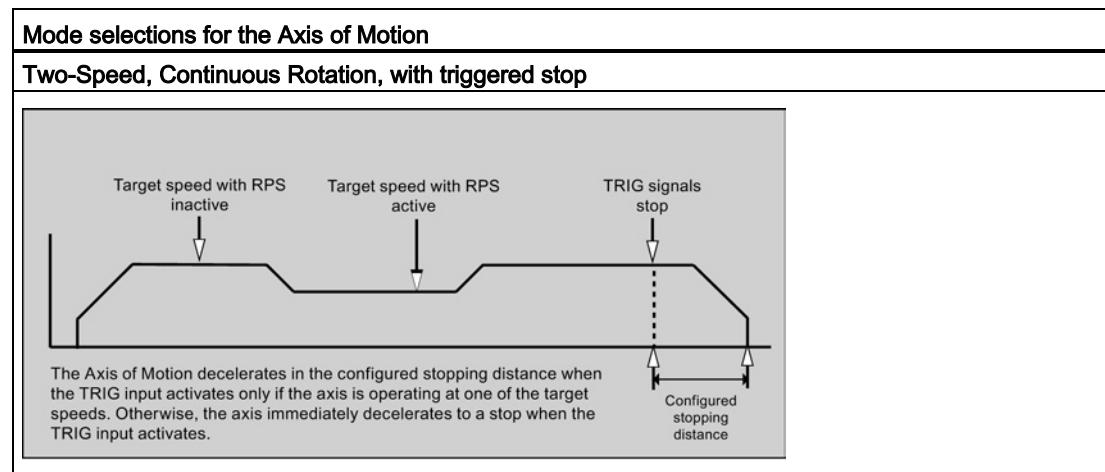
Defining the motion profile

The Motion wizard guides you through a motion profile definition, where you define each motion profile for your application. For each profile, you select the operating mode and define the specifics of each individual step for the profile. The Motion wizard also allows you to define a symbolic name for each profile by simply entering the symbol name as you define the profile.

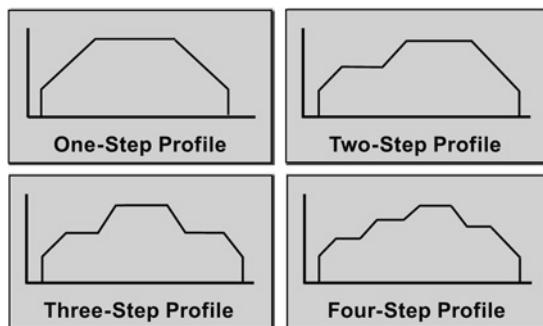
Selecting the mode of operation for the profile

You configure the profile according to the mode of operation desired. The Axis of Motion supports absolute position, relative position, single-speed continuous rotation, and two-speed continuous rotation. The following figure shows the different modes of operation.

Mode selections for the Axis of Motion	
Absolute Position	Single-Speed, Continuous Rotation
 <p>0 Zero position Starting position Ending position</p>	 <p>Controlled by your program until another command (such as Abort) is issued</p>
Single-speed, Continuous Rotation, with triggered stop	Relative Position
 <p>The Axis of Motion will decelerate in the configured stopping distance when the RPS input activates only if the axis is operating at the target speed. Otherwise, the axis immediately decelerates to a stop when the RPS input activates.</p>	 <p>Starting position Ending position Measured from the starting point</p>
Two-Speed, Continuous Rotation	
	



Creating the steps of the profile



A step is a fixed distance that a tool moves, including the distance covered during acceleration and deceleration times. The Axis of Motion supports a maximum of 16 steps in each profile.

You specify the target speed and ending position or number of pulses for each step. Additional steps are entered one at a time. The figure illustrates a one-step, two-step, three-step, and a four-step profile.

Notice that a one-step profile has one constant speed segment, a two-step profile has two constant speed segments, and so on. The number of steps in the profile matches the number of constant speed segments of the profile.

12.3 Features of motion control

Motion control provides the functionality and performance that you need for open-loop position control in up to three Axes of Motion:

- Provides high-speed control, with a range from 20 pulses per second up to 100,000 pulses per second
- Supports both jerk (S curve) or linear acceleration and deceleration
- Provides a configurable measuring system that allows you to enter data either as engineering units (such as inches or centimeters) or as a number of pulses
- Provides configurable backlash compensation
- Supports absolute, relative, and manual methods of position control
- Provides continuous operation
- Provides up to 32 motion profiles, with up to 16 speed changes per profile
- Provides four different reference-point seek modes, with a choice of the starting seek direction and the final approach direction for each sequence
- Provides support for SINAMICS V90 drives

You use STEP 7-Micro/WIN SMART to create all of the configuration and profile information used by the Axis of Motion. This information is downloaded to the CPU with your program blocks.

Motion control provides six digital inputs and four digital outputs that provide the interface to your motion application. See the following table. These inputs and outputs are local to the CPU. The CPU technical specifications (Page 513) provide detailed information for the CPUs and include wiring diagrams for connecting each CPU to some of the more common motor driver/amplifier units.

Table 12-3 Motion control CPU inputs to configure

Signal	Description
STP	The STP input causes the CPU to stop the motion in progress. You can select the desired operation of STP within the Motion wizard.
RPS	The RPS (Reference Point Switch) input establishes the reference point or home position for absolute move operations. In some modes, the RPS input is also used to stop the motion in progress after travelling a specified distance.
ZP	The ZP (Zero Pulse) input helps establish the reference point or home position. Typically, the motor driver/amplifier pulses ZP once per motor revolution. Note: Only used in RP Seek Modes of 3 and 4.
LMT+ LMT-	LMT+ and LMT- inputs establish the maximum limits for motion travel. The Motion wizard allows you to configure the operation of LMT+ and LMT- inputs.
TRIG	The TRIG (Trigger) input causes the CPU, in some modes, to stop the motion in progress after travelling a specified distance.

⚠ WARNING

Risks with changes to filter time for digital input channel

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 12.8 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 12.8 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, cycle the power of the CPU.

Table 12- 4 Motion control CPU hard-coded outputs

Signal	Description
P0	P0 and P1 are pulse outputs that control the movement and direction of movement of the motor.
P1	
DIS	DIS is an output used to disable or enable the motor driver/amplifier.

12.4 Programming an Axis of Motion

STEP 7-Micro/WIN SMART provides easy-to-use tools for configuring and programming the Axis of Motion. Simply follow these steps:

1. Configure the Axis of Motion: STEP 7-Micro/WIN SMART provides a Motion wizard for creating the configuration/profile table and the position instructions. See "Configuring the Axis of Motion" for information about configuring the Axis of Motion.
2. Test the operation of the Axis of Motion: STEP 7-Micro/WIN SMART provides a Motion control panel for testing the wiring of the inputs and outputs, the configuration of the Axis of Motion, and the operation of the motion profiles. See "Monitoring the Axis of Motion with the Motion control panel" for information about the Motion control panel.

3. Create the program to be executed by the CPU: The Motion wizard automatically creates the motion instructions that you insert into your program. See "Instructions created by the Motion wizard for the Axis of Motion" for information about the motion instructions. Insert the following instructions into your program:
 - To enable the Axis of Motion, insert an AXISx_CTRL instruction. Use SM0.0 (Always On) to ensure that this instruction is executed every scan.
 - To move the motor to a specific location, use an AXISx_GOTO or an AXISx_RUN instruction. The AXISx_GOTO instruction moves to a location specified by the inputs from your program. The AXISx_RUN instruction executes the motion profiles you configured with the Motion wizard.
 - To use absolute coordinates for your motion, you must establish the zero position for your application. Use an AXISx_RSEEK or an AXISx_LDPOS instruction to establish the zero position.
 - The other instructions that are created by the Motion wizard provide functionality for typical applications and are optional for your specific application.
4. Compile your program and download the system block, data block, and program block to the CPU.

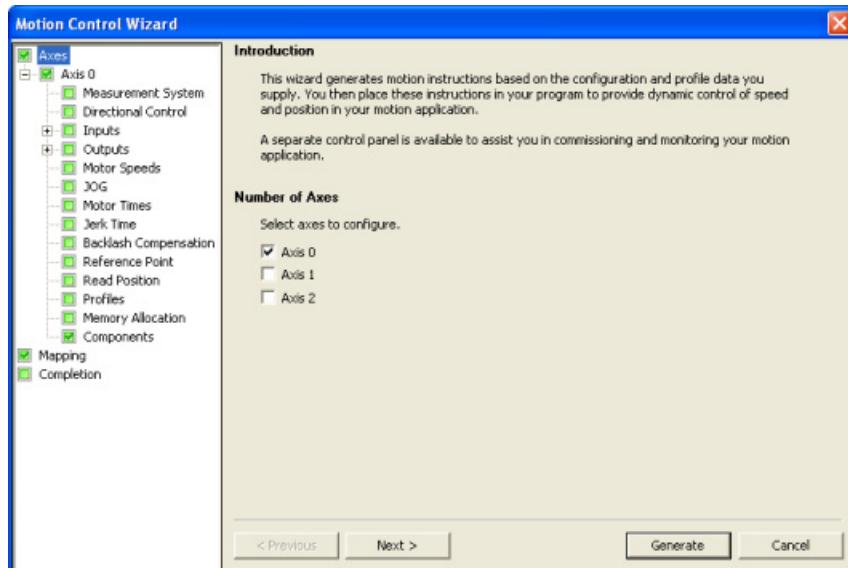
Note

Make sure to match the measurement system configuration to the pulses/revolution and distance/revolution specifications of your stepper/servo motor controller system.

12.5 Configuring an Axis of Motion

Configuration/profile table

You must create a configuration/profile table for the Axis of Motion in order for the CPU to control your motion application. The Motion wizard makes the configuration process quick and easy by leading you step-by-step through the configuration process. Refer to the "Advanced Topics" section of this chapter for detailed information about the configuration/profile table.



The Motion wizard also allows you to create the configuration/profile table offline. You can create the configuration without being connected to a CPU.

Starting the Motion wizard

To start the Motion wizard, either click the Tools icon in the navigation bar and then double-click the Motion wizard icon, or select the Tools > Motion wizard menu command.

Selecting type of measurement

Select the measurement system: You can select either engineering units or pulses:

- If you select pulses, no other information is required.
- If you select engineering units, enter the number of pulses required to produce one revolution of the motor (refer to the data sheet for your motor or drive), the base unit of measurement (such as inch, foot, millimeter, or centimeter), and the distance traveled in one revolution of the motor.

If you change the measurement system later, you must delete the entire configuration including any instructions generated by the Motion wizard. You must then enter your selections consistent with the new measurement system.

Configuring the input pin locations

You can program inputs related to motion control, to include STP, LMT-, LMT+, RPS, TRIG, and ZP, with a configuration in SDB0.

Table 12- 5 STP, RPS, LMT+, LMT-, TRIG, and ZP pin locations

Pin definition for inputs	Description
LMT+, LMT-, STP, RPS, TRIG	Input pin 0 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.0).
	Input pin 1 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.1).
	Input pin 2 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.2).
	Input pin 3 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.3).
	Input pin 4 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.4).
	Input pin 5 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.5).
	Input pin 6 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.6).
	Input pin 7 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I0.7).
	Input pin 8 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.0).
	Input pin 9 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.1).
	Input pin 10 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.2).
	Input pin 11 of the CPU can act as the LMT+, LMT-, STP, RPS, TRIG input (I1.3).
ZP HSC	HSC0 of the CPU acts as the ZP input (I0.0).
	HSC1 of the CPU acts as the ZP input (I0.1).
	HSC2 of the CPU acts as the ZP input (I0.2).
	HSC3 of the CPU acts as the ZP input (I0.3).

Note

After you configure an input to a specific function (for example, RPS) for a particular Axis of Motion, you cannot use that input for any other Axis of Motion or function.

Note

High-speed input wiring must use shielded cables

Use shielded cable with a maximum length of 50 m, when connecting HSC input channels I0.0, I0.1, I0.2, and I0.3.

Mapping the I/O

STEP 7-Micro/WIN SMART enforces a fixed output assignment for PWM and Axis of Motion.

P0 and P1 outputs

At a minimum, any axis that is enabled has a P0 output pin configured for it. It may also have P1 output if its "Phase" configuration is anything other than "Single-phase (1 output)". Refer to the "Editing default input and output configuration" section for more information. These output pins are hard-coded to a specific output, depending on the following criteria:

Axis 0	<ul style="list-style-type: none">• P0 for Axis 0 is always configured for Q0.0.• P1 for Axis 0 is configured for Q0.2 if the "Phase" for the axis is not configured to "Single phase (1 output)".
Axis 1	<ul style="list-style-type: none">• P0 for Axis 1 is always configured for Q0.1.• P1 for Axis 1 is mapped in two possible locations based upon axis configuration as follows:<ul style="list-style-type: none">– If the "Phase" for Axis 1 is configured for "Single-phase (1 output)", then no P1 output is assigned.– If the "Phase" for Axis 1 is configured for "Two-phase (2 output)" or "AB quadrature phase (2 output)", then P1 is configured for Q0.3.– Else, P1 for Axis 1 is configured for Q0.7.
Axis 2	<ul style="list-style-type: none">• P0 for Axis 2 is always configured for Q0.3.• P1 for Axis 2 is configured for Q1.0 if the "Phase" for the axis is not configured to "Single phase (1 output)".• Axis 2 is not available for use if the configured "Phase" for Axis 1 is configured to "Two-phase (2 output)" or "AB quadrature phase (2 output)".

DIS outputs

If an axis has a DIS output configured, then an entry is present in the mapping table for that output. The DIS output is also hard-coded to specific outputs based upon the following rules:

- DIS for Axis 0 is always configured for Q0.4.
- DIS for Axis 1 is always configured for Q0.5.
- DIS for Axis 2 is always configured for Q0.6.

Pulse output units are connected to standard 24V outputs.

Editing default input and output configuration

To change or view the default configuration of the integrated inputs/outputs select the required input/output node:

- In the "Active level" field, use the dropdown list to select the active level (High or Low). When the level is set to High, a logic 1 is read when current is flowing in the input. When the level is set to Low, a logic 1 is read when there is no current flow in the input. A logic 1 level is always interpreted as meaning the condition is active. The LEDs are illuminated when current flows in the input, regardless of activation level. (Default = active high)
- In the "System Block", "Digital Inputs" node, you can select the filter time constant (0.20 ms to 12.80 ms) for the STP, RPS, LMT+, LMT-, and TRIG inputs. Increasing the filter time constant eliminates more noise, but it also slows down the response time to a signal state change. (Default = 6.4 ms)

WARNING

Risks with changes to filter time for digital input channel

If the filter time for a digital input channel is changed from a previous setting, a new "0" level input value may need to be presented for up to 12.8 ms accumulated duration before the filter becomes fully responsive to new inputs. During this time, short "0" pulse events of duration less than 12.8 ms may not be detected or counted.

This changing of filter times can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

To ensure that a new filter time goes immediately into effect, a power cycle of the CPU must be applied.

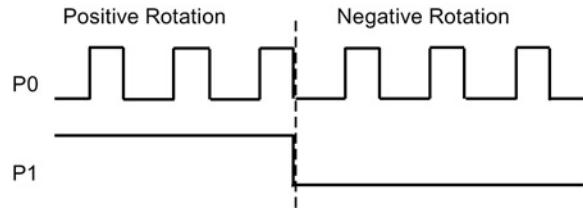
- In the "Directional Control" node, you can select the following "Phasing" modes:
 - Single phase (2 output)
 - Two-phase (2 output)
 - AB quadrature phase (2 output)
 - Single phase (1 output)

You can also select the "Polarity" (positive or negative) of the outputs.

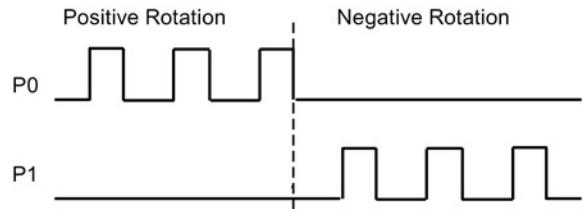
Phasing

You have four options for the "Phasing" interface to the stepper/servo drive. These options are as follows:

- Single phase (2 output): If you select the single phase (2 output) option, then one output (P0) controls the pulsing, and one output (P1) controls the direction. P1 is high (active) if pulsing is in the positive direction. P1 is low (inactive) if pulsing is in the negative direction. Single phase (2 output) is shown in the figure below (assuming positive polarity):



- Two-phase (2 output): If you select the Two-phase (2 output) option, then one output (P0) pulses for positive directions, and a different output (P1) pulses for negative directions. Two-phase (2 output) is shown in the figure below (assuming positive polarity):



- AB quadrature phase (2 output): If you select the AB quadrature phase (2 output) option, then both outputs pulse at the speed specified, but 90 degrees out-of-phase. The AB quadrature phase (2 output) is a 1X configuration, meaning a generated pulse is measured from one positive transition of the output to the next positive transition of the same output. In this case, the direction is determined by which output transitions high first. P0 leads P1 for the positive direction. P1 leads P0 for the negative direction. AB quadrature phase (2 output) is shown in the figures below (assuming positive polarity):

AB quadrature phase (2 output)	
(Positive polarity): positive rotation	(Positive polarity): negative rotation
P0	P0
P1	P1
P0 leads P1	
P1 leads P0	

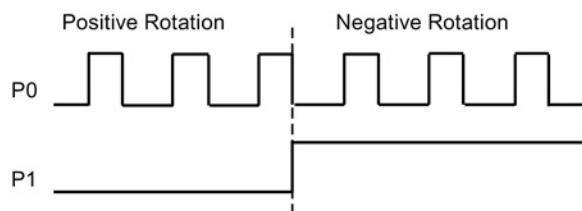
- Single phase (1 output): If you select the single phase (1 output) option, then the output (P0) controls the pulsing. Only positive motion commands are accepted by the CPU in this mode. The Motion wizard restricts you from making illegal negative configurations when you select this mode. You can save an output if your motion application is in one direction only. Single phase (1 output) is shown in the figure below (assuming positive polarity):



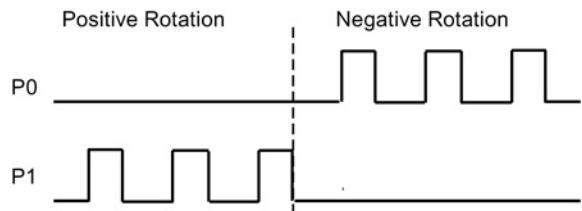
Polarity

You can switch positive and negative directions with the "Polarity" parameter. If the motor is wired in the wrong direction, this is typically done. You can avoid re-wiring the hardware by setting this parameter to negative. The negative setting changes the output operation as follows:

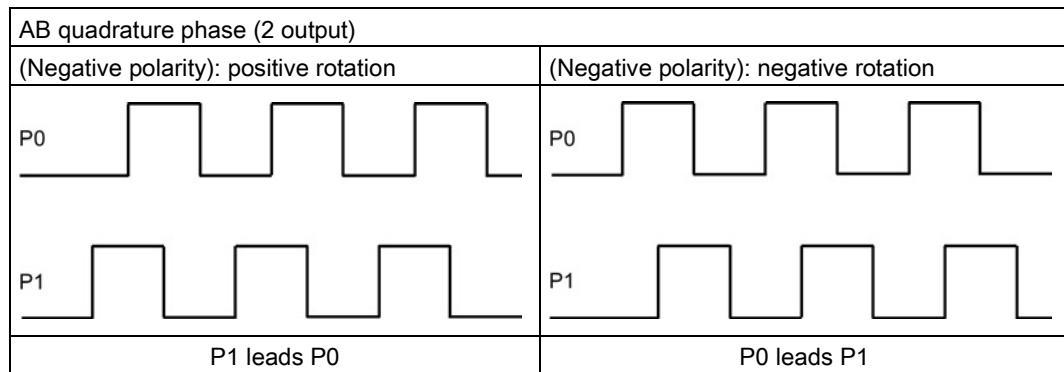
- Single phase (2 output): P1 is low (inactive) if pulsing is in the positive direction. P1 is high (active) if pulsing is in the negative direction. This is shown in the figure below:



- Two-phase (2 output): P0 pulses for negative directions. P1 pulses for positive directions. This is shown in the figure below:



- AB quadrature phase (2 output): P0 leads P1 for a negative direction. P1 leads P0 for a positive direction. This is shown in the figure below:



- Single phase (1 output): Negative polarity is not allowed in this phasing mode.

The default setting for the "Directional Control" dialog is "Single phase (2 output)" and "Positive polarity".

Note

You cannot choose to which pins P0 and P1 are configured; this is hardcoded to a specific pin. Refer to the Mapping I/O section (preceding this section) for the pin mapping list.



WARNING

Safety precautions when using an Axis of Motion

The limit and stop functions in the Axis of Motion are electronic logic implementations that do not provide the level of protection provided by electromechanical controls.

Control devices and Axis of Motion functions can fail in unsafe conditions, which can result in unpredictable operation of controlled equipment. Such unpredictable operations could result in death or serious personal injury, and/or property damage.

Consider using an emergency stop function, electromechanical overrides, or redundant safeguards that are independent of the Axis of Motion and the CPU.

Configure response to physical inputs

1. Select the response to the LMT+, the LMT-, and the STP inputs.
2. Use the dropdown list to select: decelerate to a stop (default) or immediate stop.

Entering maximum start and stop speed

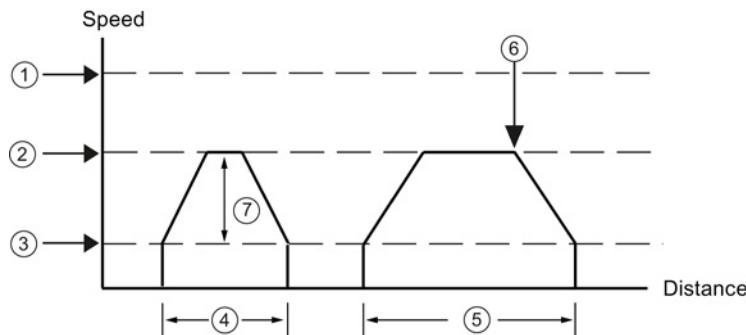
Enter the maximum speed (MAX_SPEED) and Start/Stop Speed (SS_SPEED) for your application.

Entering jog parameters

Enter the JOG_SPEED and the JOG_INCREMENT values:

- JOG_SPEED: The JOG_SPEED (Jog speed for the motor) is the maximum speed that can be obtained while the JOG command remains active.
- JOG_INCREMENT: Distance that the tool is moved by a momentary JOG command.

The following figure shows the operation of the Jog command. When the Axis of Motion receives a Jog command, it starts a timer. If the Jog command is terminated before 0.5 seconds has elapsed, the Axis of Motion moves the tool the amount specified in the JOG_INCREMENT at the speed defined by JOG_SPEED. If the Jog command is still active when the 0.5 seconds have elapsed, the Axis of Motion accelerates to the JOG_SPEED. Motion continues until the Jog command is terminated. The Axis of Motion then performs a decelerated stop. You can enable the Jog command either from the Motion Control Panel or with a motion instruction. A representation of a JOG operation is shown in the figure below.



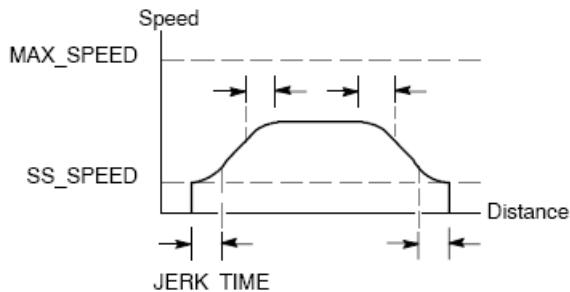
- ① MAX_SPEED
- ② JOG_SPEED
- ③ SS_SPEED
- ④ JOG_INCREMENT: JOG command is active for less than 0.5 seconds.
- ⑤ JOG command is active for more than 0.5 seconds.
- ⑥ JOG command is terminated (Starts ramp from JOG_SPEED to SS_SPEED).
- ⑦ The speed reached can be anywhere from the SS_SPEED to the JOG_SPEED, depending on the length of the JOG_INCREMENT.

Entering acceleration time

Enter the acceleration and deceleration times in the edit boxes.

Entering jerk time

Available on certain types of moves, jerk compensation provides smoother position control by reducing the jerk (rate of change) in the acceleration and deceleration parts of the motion profile. See the following figure:



Jerk compensation is also known as "S curve profiling". This compensation is applied equally to the beginning and ending portions of both the acceleration and deceleration curve. Jerk compensation is not applied to the initial and final step between zero speed and SS_SPEED.

Specify the jerk compensation by entering a time value (JERK_TIME). This is the time required for acceleration to change from zero to the maximum acceleration rate. A longer jerk time yields smoother operation with a smaller increase in total cycle time than would be obtained by decreasing the ACCEL_TIME and DECEL_TIME. A value of 0 ms (the default value) indicates that no compensation is to be applied.

Note

A good first value for JERK_TIME is 40% of ACCEL_TIME.

Note

Jerk compensation is not available for two speed moves, manual speed change moves, aborted moves, and automatic deceleration reactions upon reaching a limit or STP input.

Configuring the Backlash compensation

Backlash compensation: Distance that the motor must move to eliminate the backlash (slack) in the system on a direction change. Backlash compensation is always a positive value.

- Default = 0
- Choose a Reference Point search sequence to use backlash.

Configuring reference point and seek parameters

1. Select using a reference point or not using a reference point for your application:
 - If your application requires that movements start from or be referenced to an absolute position, you must establish a reference point (RP) or zero position that fixes the position measurements to a known point on the physical system.
 - If a reference point is used, you will want to define a way to automatically relocate the reference point. The process of automatically locating the reference point is called Reference Point Seek. Defining the Reference Point Seek process requires two steps in the wizard.
2. Enter the Reference Point seek speeds (a fast seek speed and a slow seek speed):
 - **RP_FAST** is the initial speed the module uses when performing an RP seek command. Typically, the RP_FAST value is approximately 2/3 of the MAX_SPEED value.
 - **RP_SLOW** is the speed of the final approach to the RP. A slower speed is used on approach to the RP, so as not to miss it. Typically, the RP_SLOW value is the SS_SPEED value.
3. Define the initial seek direction and the final reference point approach direction:
 - **RP_SEEK_DIR** is the initial direction for the RP seek operation. Typically, this is the direction from the work zone to the vicinity of the RP. Limit switches play an important role in defining the region that is searched for the RP. When performing a RP seek operation, encountering a limit switch can result in a reversal of the direction, which allows the search to continue. (Default = Negative)
 - **RP_APPR_DIR** is the direction of the final approach to the RP. To reduce backlash and provide more accuracy, the reference point should be approached in the same direction used to move from the RP to the work zone. (Default = Positive)
4. The Motion wizard provides advanced reference point options that allow you to specify an RP offset (RP_OFFSET), which is the distance from the RP to the zero position. See the following figure:
 - **RP_OFFSET**: Distance from the RP to the zero position of the physical measuring system.
 - Default = 0



5. The Axis of Motion provides a reference point switch (RPS) input that is used when seeking the RP. The RP is identified by a method of locating an exact position with respect to the RPS. The RP can be centered in the RPS Active zone, the RP can be located on the edge of the RPS Active zone, or the RP can be located a specified number of zero pulse (ZP) input transitions from the edge of the RPS Active zone.
6. You can configure the sequence that the Axis of Motion uses to search for the reference point. The following figure shows a simplified diagram of the default RP search sequence. Select one of the following options for the RP search sequence:
 - **RP Seek mode 0:** Does not perform a RP seek sequence
 - **RP Seek mode 1:** The RP is where the RPS input goes active on the approach from the work zone side. (Default)
 - **RP Seek mode 2:** The RP is centered within the active region of the RPS input.
 - **RP Seek mode 3:** The RP is located outside the active region of the RPS input. RP_Z_CNT specifies how many ZP (Zero Pulse) input counts should be received after the RPS becomes inactive.
 - **RP Seek mode 4:** The RP is generally within the active region of the RPS input. RP_Z_CNT specifies how many ZP (Zero Pulse) input counts should be received after the RPS becomes active.

RP seek mode 1

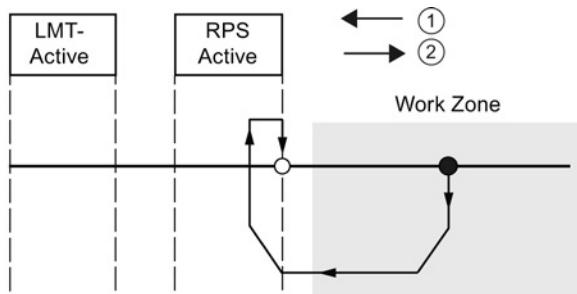


Figure 12-1 ①: RP seek direction
 ②: RP approach direction

Note

The RPS Active region (which is the distance that the RPS input remains active) must be greater than the distance required to decelerate from the RP_FAST speed to the RP_SLOW speed. If the distance is too short, the Axis of Motion generates an error.

Defining the motion profile

1. In the motion profile definition screen, click the new profile button to enable defining the profile.
2. Choose the desired mode of operation:
 - For an absolute position profile:
Fill in the target speed and the ending position.
If more than one step is needed, click the new step button and fill in the step information as required.
 - For a relative position profile:
Fill in the target speed and the ending position.
If more than one step is needed, click the new step button and fill in the step information as required.
 - For a single-speed, continuous rotation:
Enter the target speed value in the edit box.
Select the direction of rotation.
If you wish to terminate the single speed, continuous rotation move using the RPS input, click the checkbox.
Fill in the distance to move after the RPS input goes active (RPS input must be enabled).
 - For a two-speed, continuous rotation (RPS input must be enabled):
Enter the target speed value when RPS is inactive in the edit box.
Enter the target speed value when RPS is active in the edit box.
Select the direction of rotation.
If you wish to terminate the two speed, continuous rotation move using the TRIG input, click the checkbox. (TRIG input must be enabled.)
Fill in the distance to move after the TRIG input goes active.
3. Define as many profiles and steps as you need to perform the desired movement.

Finishing the configuration

1. After you have configured the operation of the Axis of Motion, you simply click "Generate".

The Motion wizard performs the following tasks:

- Inserts the axis configuration and profile table into the system block and data block for your CPU program
- Creates a global symbol table for the motion parameters
- Adds the motion instruction subroutines into the project program block for you to use in your application

2. You can run the Motion wizard again in order to modify any configuration or profile information.

Note

Because the Motion wizard makes changes to the program block, the data block, and the system block, you must download all three blocks to the CPU. Otherwise, the Axis of Motion will not have all the program components that it needs for proper operation.

12.6 Subroutines created by the Motion wizard for the Axis of Motion

You must ensure for each motion action that only one motion subroutine is active at a time in addition to the AXISx_CTRL, which must be active every scan. Each motion subroutine is prefixed with an "AXISx_" where "x" is the axis number channel. There are 13 motion subroutines.

Motion subroutine	Description
AXISx_CTRL	Provides initialization and overall control of the axis
AXISx_MAN	Used for manual mode operation of the axis
AXISx_GOTO	Commands the axis to go to a specified location
AXISx_RUN	Commands the axis to execute a configured motion profile
AXISx_RSEEK	Initiates a reference point seek operation
AXISx_LDOFF	Establishes a new zero position that is offset from the reference point position
AXISx_LDPOS	Changes the axis position to a new value
AXISx_SRATE	Modifies the configured acceleration, deceleration, and jerk compensation times
AXISx_DIS	Controls the DIS output
AXISx_CFG	Reads the configuration block and updates the axis setup as required
AXISx_CACHE	Pre-caches a configured motion profile
AXISx_RDPOS	Returns the current axis position
AXISx_ABSPOS	Reads the absolute position value from a SINAMICS V90 servo drive

Note

The motion subroutines increase the amount of memory required for your program by up to 1700 bytes. You can delete unused motion subroutines to reduce the amount of memory required. To prevent the generation of unneeded motion subroutines, uncheck the "Generate" box for each unneeded subroutine in the "Components" node of the Motion wizard. To restore generation of a particular motion subroutine, start the Motion wizard again, navigate to the "Components" node, and check the "Generate" box for the subroutine. Click the "Generate" button to rebuild the wizard-generated subroutines.

See also

Using the Motion wizard

12.6.1 Guidelines for using the Motion subroutines

You must ensure that only one motion subroutine is active at a time.

You can execute the AXISx_RUN and AXISx_GOTO from an interrupt routine as long as the interrupt is called cyclically. However, it is very important that you do not attempt to start a motion subroutine in an interrupt routine if the Axis of Motion is busy processing another command. If you start a subroutine in an interrupt routine, then you can use the outputs of the AXISx_CTRL subroutine to monitor when the Axis of Motion has completed the movement.

The Motion wizard automatically configures the values for the speed parameters (Speed and C_Speed) and the position parameters (Pos or C_Pos) according to the measurement system that you selected. For pulses, these parameters are DINT values. For engineering units, the parameters are REAL values for the type of unit that you selected. For example: selecting centimeters (cm) stores the position parameters as REAL values in centimeters and stores the speed parameters as REAL values in centimeters per second (cm/sec).

Some "generate" guidelines when using motion subroutines are as follows:

- Insert the AXISx_CTRL subroutine in your program, and use the SM0.0 contact to execute it every scan.
- To specify motion to an absolute position, you must first use either an AXISx_RSEEK or a AXISx_LDPOS subroutine to establish the zero position.
- To move to a specific location, based upon inputs from your program, use the AXISx_GOTO subroutine.
- To run the motion profiles you configured with the Motion wizard, use the AXISx_RUN subroutine.

12.6.2 AXISx_CTRL subroutine

Table 12- 6 AXISx_CTRL

LAD / FBD	STL	Description
<pre> AXIS0_CTRL -EN -MOD_~ Done Error C_Pos C_Spe^ C_Dir </pre>	<pre> CALL AXISx_CTRL, MOD_EN, Done, Error, C_Pos, C_Speed, C_Dir </pre>	<p>The AXISx_CTRL subroutine (Control) enables and initializes the Axis of Motion by automatically commanding the Axis of Motion to load the configuration/profile table each time the CPU changes to RUN mode.</p> <p>Use this subroutine only once in your project per motion axis, and ensure that your program calls this subroutine every scan. Use SM0.0 (Always On) as the input for the EN parameter.</p>

Table 12- 7 Parameters for the AXISx_CTRL subroutine

Inputs/Outputs	Data type	Operands
MOD_EN	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done, C_Dir	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

The MOD_EN parameter must be on to enable the other motion subroutines to send commands to the Axis of Motion. If the MOD_EN parameter turns off, then the Axis of Motion aborts any command that is in progress.

The output parameters of the AXISx_CTRL subroutine provide the current status of the Axis of Motion.

The Done parameter turns on when the Axis of Motion completes any subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

The C_Pos parameter is the current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter provides the current speed of the Axis of Motion. If you configured the measurement system for the Axis of Motion for pulses, C_Speed is a DINT value containing the number of pulses/second. If you configured the measurement system for engineering units, C_Speed is a REAL value containing the selected engineering units/second (REAL).

The C_Dir parameter indicates the current direction of the motor:

- Signal state of 0 = positive
- Signal state of 1 = negative

Note

The Axis of Motion reads the configuration/profile table only at power-up or when commanded to load the configuration.

- If you use the Motion wizard to modify the configuration, then the AXISx_CTRL subroutine automatically commands the Axis of Motion to load the configuration/profile table every time the CPU changes to RUN mode.
- If you use the Motion control panel to modify the configuration, clicking the Update Configuration button commands the Axis of Motion to load the new configuration/profile table.
- If you use another method to modify the configuration, then you must also issue an AXISx_CFG command to the Axis of Motion to load the configuration/profile table. Otherwise, the Axis of Motion continues to use the old configuration/profile table.

12.6.3 AXISx_MAN subroutine

Table 12- 8 AXISx_MAN

LAD / FBD	STL	Description
<pre> LAD / FBD diagram for AXISx_MAN: - Inputs: EN, RUN, JOG_P, JOG_N, Speed, Dir, Error, C_Pos, C_Speed, C_Dir - Structure: A main block labeled 'AXISx_MAN' containing the inputs. </pre>	<pre> CALL AXISx_MAN, RUN, JOG_P, JOG_N, Speed, Dir, Error, C_Pos, C_Speed, C_Dir </pre>	<p>The AXISx_MAN subroutine (Manual Mode) puts the Axis of Motion into manual mode. This allows the motor to be run at different speeds or to be jogged in a positive or negative direction.</p> <p>You can enable only one of the RUN, JOG_P, or JOG_N inputs at a time.</p>

Table 12- 9 Parameters for the AXISx_MAN subroutine

Inputs/Outputs	Data type	Operands
RUN, JOG_P, JOG_N	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant

Inputs/Outputs	Data type	Operands
Dir, C_Dir	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Enable the RUN (Run/Stop) parameter to command the Axis of Motion to accelerate to the specified speed (Speed parameter) and direction (Dir parameter). You can change the value of the Speed parameter while the motor is running, but the Dir parameter must remain constant. Disabling the RUN parameter commands the Axis of Motion to decelerate until the motor comes to a stop.

Enable the JOG_P (Jog Positive Rotation) or the JOG_N (Jog Negative Rotation) parameter to command the Axis of Motion to jog in either a positive or negative direction. If the JOG_P or JOG_N parameter remains enabled for less than 0.5 seconds, the Axis of Motion issues pulses to travel the distance specified in JOG_INCREMENT. If the JOG_P or JOG_N parameter remains enabled for 0.5 seconds or longer, the Axis of Motion begins to accelerate to the specified JOG_SPEED.

The Speed parameter determines the speed when RUN is enabled. If you configured the measuring system of the Axis of Motion for pulses, the speed is a DINT value for pulses/second. If you configured the measuring system of the Axis of Motion for engineering units, the speed is a REAL value for units/second. You can change this parameter while the motor is running.

Note

The Axis of Motion may not react to small changes in the Speed parameter, especially if the configured acceleration or deceleration time is short and the difference between the configured maximum speed and start/stop speed is large.

The Dir parameter determines the direction to move when RUN is enabled. You cannot change this value when the RUN parameter is enabled.

The Error parameter (Page 491) contains the result of this subroutine.

The C_Pos parameter contains the current position of the Axis of Motion. Based upon the units of measurement selected, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter contains the current speed of the Axis of Motion. Based upon the units of measurement selected, the value is either the number of pulses/second (DINT) or the engineering units/second (REAL).

The C_Dir parameter indicates the current direction of the motor:

- Signal state of 0 = positive
- Signal state of 1 = negative

12.6.4 AXISx_GOTO subroutine

Table 12- 10 AXISx_GOTO

LAD / FBD	STL	Description
<pre> LAD Diagram for AXISx_GOTO: - EN (Input) connects to the main block. - Inside the block, START (Output) connects to an edge detection element. - The edge detection element has four outputs: - Pos connects to Done - Speed connects to Error - Mode connects to C_Pos - Abort connects to C_Spe^ </pre>	CALL AXISx_GOTO, START, Pos, Speed, Mode, Abort, Done, Error, C_Pos, C_Speed	The AXISx_GOTO subroutine commands the Axis of Motion to go to a desired location.

Table 12- 11 Parameters for the AXISx_GOTO subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Pos, Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Mode	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Abort, Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the DONE bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a GOTO command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a GOTO command to the Axis of Motion. To ensure that only one GOTO command is sent, use an edge detection element to pulse the START parameter on.

The Pos parameter contains a value that signifies either the location to move (for an absolute move) or the distance to move (for a relative move). Based upon the units of measurement selected, the value is either a number of pulses (DINT) or the engineering units (REAL).

The Speed parameter determines the maximum speed for this movement. Based upon the units of measurement, the value is either a number of pulses/second (DINT) or the engineering units/second (REAL).

The Mode parameter selects the type of move:

- 0: Absolute position
- 1: Relative position
- 2: Single-speed, continuous positive rotation
- 3: Single-speed, continuous negative rotation

12.6 Subroutines created by the Motion wizard for the Axis of Motion

The Done parameter turns on when the Axis of Motion completes this subroutine.

Turn on the Abort parameter to command the Axis of Motion to stop execution of this command and decelerate until the motor comes to a stop.

The Error parameter (Page 491) contains the result of this subroutine.

The C_Pos parameter contains current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter contains the current speed of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses/second (DINT) or the engineering units/second (REAL).

12.6.5 AXISx_RUN subroutine

Table 12- 12 AXISx_RUN

LAD / FBD	STL	Description
<pre> LAD / FBD Diagram for AXISx_RUN: - EN (Input) - START (Output) - Done (Output) - Abort (Output) - Error (Output) - C_Profile (Output) - C_Step (Output) - C_Pos (Output) - C_Spe~ (Output) </pre>	<pre> CALL AXISx_RUN, START, Profile, Abort, Done, Error, C_Profile, C_Step, C_Pos, C_Speed </pre>	<p>The AXISx_RUN subroutine (Run Profile) commands the Axis of Motion to execute the motion operation in a specific profile stored in the configuration/profile table.</p>

Table 12- 13 Parameters for the AXISx_RUN subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Profile	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Abort, Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error, C_Profile, C_Step	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a RUN command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a RUN command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

12.6 Subroutines created by the Motion wizard for the Axis of Motion

The Profile parameter contains the number or the symbolic name for the motion profile. The "Profile" input must be between 0 - 31. If not, the subroutine will return an error.

Turn on the Abort parameter to command the Axis of Motion to stop the current profile and decelerate until the motor comes to a stop.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

The C_Profile parameter contains the profile currently being executed by the Axis of Motion.

The C_Step parameter contains the step of the profile currently being executed.

The C_Pos parameter contains the current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

The C_Speed parameter contains the current speed of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses/second (DINT) or the engineering units/second (REAL).

12.6.6 AXISx_RSEEK subroutine

Table 12- 14 AXISx_RSEEK

LAD / FBD	STL	Description
	CALL AXISx_RSEEK, START, Done, Error	<p>The AXISx_RSEEK subroutine (Seek Reference Point Position) initiates a reference point seek operation, using the search method in the configuration/profile table. When the Axis of Motion locates the reference point and motion has stopped, the Axis of Motion loads the RP_OFFSET parameter value into the current position.</p>

Table 12- 15 Parameters for the AXISx_RSEEK subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

The default value for RP_OFFSET is 0. You can use the Motion wizard, the Motion Control Panel, or the AXISx_LDOFF (Load Offset) subroutine to change the RP_OFFSET value.

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

12.6 Subroutines created by the Motion wizard for the Axis of Motion

Turn on the START parameter to send a RSEEK command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a RSEEK command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

12.6.7 AXISx_LDOFF subroutine

Table 12- 16 AXISx_LDOFF

LAD / FBD	STL	Description
	CALL AXISx_LDOFF, START, Done, Error	<p>The AXISx_LDOFF subroutine (Load Reference Point Offset) establishes a new zero position that is at a different location from the reference point position.</p> <p>Before executing this subroutine, you must first determine the position of the reference point. You must also move the machine to the starting position. When the subroutine sends the LDOFF command, the Axis of Motion computes the offset between the starting position (the current position) and the reference point position. The Axis of Motion then stores the computed offset to the RP_OFFSET parameter and sets the current position to 0. This establishes the starting position as the zero position.</p> <p>In the event that the motor loses track of its position (such as on loss of power or if the motor is repositioned manually), you can use the AXISx_RSEEK subroutine to re-establish the zero position automatically.</p>

Table 12- 17 Parameters for the AXISx_LDOFF subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a LDOFF command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a LDOFF command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

12.6.8 AXISx_LDPOS subroutine

Table 12- 18 AXISx_LDPOS

LAD / FBD	STL	Description
<pre> CALL AXIS0_LDPOS, START, New_Pos, Done, Error, C_Pos </pre>	<pre> CALL AXISx_LDPOS, START, New_Pos, Done, Error, C_Pos </pre>	The AXISx_LDPOS subroutine (Load Position) changes the current position value in the Axis of Motion to a new value. You can also use this subroutine to establish a new zero position for any absolute move command.

Table 12- 19 Parameters for the AXISx_LDPOS subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
New_Pos, C_Pos	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a LDPOS command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a LDPOS command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The New_Pos parameter provides the new value to replace the current position value that the Axis of Motion reports and uses for absolute moves. Based upon the units of measurement, the value is either a number of pulses (DINT) or the engineering units (REAL).

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

The C_Pos parameter contains the current position of the Axis of Motion. Based upon the units of measurement, the value is either a number of pulses (DINT) or the number of engineering units (REAL).

12.6.9 AXISx_SRATE subroutine

Table 12- 20 AXISx_SRATE

LAD / FBD	STL	Description
<pre> AXISx_SRATE EN START ACCEL~ Done DECEL~ Error JERK ~ </pre>	<pre> CALL AXISx_SRATE, START, ACCEL_Time, DECEL_Time, JERK_Time, Done, Error </pre>	The AXISx_SRATE subroutine (Set Rate) commands the Axis of Motion to change the acceleration, deceleration, and jerk times.

Table 12- 21 Parameters for the AXISx_SRATE subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L
ACCEL_Time, DECEL_Time, JERK_Time	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to copy the new time values to the configuration/profile table and sends a SRATE command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends an SRATE command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The ACCEL_Time, DECEL_Time, and JERK_Time parameters determine the new acceleration time, deceleration time, and jerk time in milliseconds (ms).

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

12.6.10 AXISx_DIS subroutine

Table 12- 22 AXISx_DIS

LAD / FBD	STL	Description
	<code>CALL AXISx_DIS, DIS_ON, Error</code>	The AXISx_DIS subroutine turns the DIS output of the Axis of Motion on or off. This allows you to use the DIS output for disabling or enabling a motor controller. If you use the DIS output on the Axis of Motion, then this subroutine can be called every scan or only when you need to change the value of the DIS output.

Table 12- 23 Parameters for the AXISx_DIS subroutine

Inputs/Outputs	Data type	Operands
DIS_ON	BOOL	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

When the EN bit turns on to enable the subroutine, the DIS_ON parameter controls the DIS output of the Axis of Motion.

Note

If you have not defined a "DIS" output in the Motion wizard, the AXISx_DIS subroutine will return an error.

The Error parameter (Page 491) contains the result of this subroutine.

12.6.11 AXISx_CFG subroutine

Table 12- 24 AXISx_CFG

LAD / FBD	STL	Description
	<code>CALL AXISx_CFG, START, Done, Error</code>	The AXISx_CFG subroutine (Reload Configuration) commands the Axis of Motion to read the configuration block from the location specified by the configuration/profile table pointer. The Axis of Motion then compares the new configuration with the existing configuration and performs any required setup changes or recalculations.

12.6 Subroutines created by the Motion wizard for the Axis of Motion

Table 12- 25 Parameters for the AXISx_CFG subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

Turn on the START parameter to send a CFG command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a CFG command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

12.6.12 AXISx_CACHE subroutine

Table 12- 26 AXISx_CACHE

LAD / FBD	STL	Description
<pre> LD AXIS0_CACHE EN OUT START LD AXIS0_CACHE IN Profile OUT Done OUT Error </pre>	<pre> CALL AXISx_CACHE, START, Profile, Done, Error </pre>	<p>The AXISx_CACHE subroutine (Cache Profile) commands a caching of a motion profile before the profile is executed. This allows you to pre-cache needed commands before execution. Pre-caching reduces the amount of time and provides consistency between executing a motion instruction and starting the move.</p>

Table 12- 27 Parameters for the AXISx_CACHE subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
Profile	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Abort, Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error, C_Profile, C_Step	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
C_Pos, C_Speed	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the Done bit signals that the execution of the subroutine has completed.

12.6 Subroutines created by the Motion wizard for the Axis of Motion

Turn on the START parameter to send a CACHE command to the Axis of Motion. For each scan when the START parameter is on and the Axis of Motion is not currently busy, the subroutine sends a CACHE command to the Axis of Motion. To ensure that only one command is sent, use an edge detection element to pulse the START parameter on.

The Profile parameter contains the number or the symbolic name for the motion profile. The "Profile" input must be between 0 - 31. If not, the subroutine will return an error.

The Done parameter turns on when the Axis of Motion completes this subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

12.6.13 AXISx_RDPOS subroutine

Table 12- 28 AXISx_RDPOS

LAD / FBD	STL	Description
	CALL AXISx_RDPOS, Error, I_Pos	The AXISx_RDPOS subroutine returns the current motion axis position.

Table 12- 29 Parameters for the AXISx_RDPOS subroutine

Inputs/Outputs	Data type	Operands
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
I_Pos	DINT, REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine.

The Error parameter (Page 491) contains the result of this subroutine.

The I_Pos parameter contains the current motion axis position.

Note

Execution of this command returns the actual current position of the axis. Position status values provided in other motion subroutines, such as AXISx_CTRL, AXISx_GOTO, and so forth, are updated periodically. Therefore, the position values reported by those commands and the position value reported by this command may differ somewhat as a result and is normal.

12.6.14 AXISx_ABSPOS subroutine

Table 12- 30 AXISx_ABSPOS

LAD / FBD	STL	Description
<pre> CALL AXISx_ABSPOS, EN START RDY INP Res Drive Port Done Error D_Pos </pre>	<pre> CALL AXISx_ABSPOS, START, RDY, INP, Res, Drive, Port, Done, Error, D_Pos </pre>	The AXISx_ABSPOS subroutine reads the absolute position from certain Siemens servo drives, such as the V90. You read the absolute position value in order to update the current position value in the Axis of Motion. This capability is supported when using a SINAMICS V90 servo drive combined with a SIMOTICS-1FL6 servo motor that has an absolute encoder installed.

Table 12- 31 Parameters for the AXISx_ABSPOS subroutine

Inputs/Outputs	Data type	Operands
START	BOOL	I, Q, V, M, SM, S, T, C, L, Power Flow
RDY, INP	BOOL	I, Q, V, M, SM, S, T, C, L
Res	DINT	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD, Constant
Drive	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Port	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD, Constant
Done	BOOL	I, Q, V, M, SM, S, T, C, L
Error	BYTE	IB, QB, VB, MB, SMB, SB, LB, AC, *VD, *AC, *LD
D_Pos	REAL	ID, QD, VD, MD, SMD, SD, LD, AC, *VD, *AC, *LD

Turn on the EN bit to enable the subroutine. Ensure that the EN bit stays on until the DONE bit signals that the execution of the subroutine has completed.

Turn on the START parameter to obtain the current absolute position from the specified drive. To ensure that only one operation to read the current position is performed, use an edge detection element to pulse the START parameter on.

The RDY parameter indicates the readiness state of the servo drive, which is typically provided by a digital output signal from the drive. This subroutine will read the absolute position from the drive only if this parameter is on.

The INP parameter indicates the standstill state of the motor, which is typically provided by a digital output signal from the drive. This subroutine will read the absolute position from the drive only if this parameter is on.

The Res parameter must be set to the resolution of the absolute encoder connected to your servo motor. For example, the single turn resolution of a SIMOTICS S-1FL6 servo motor with absolute encoder is 20 bits or 1048576.

Set the Drive parameter to match the RS485 address of the servo drive to be accessed by this subroutine. Valid addresses of individual drives are 0 to 31.

Set the Port parameter to designate the CPU port to be used to communicate with the servo drive:

- 0: Onboard RS485 port (Port 0)
- 1: RS485/RS232 signal board (if present, Port 1)

The Done parameter turns on when the subroutine's work is complete.

The Error parameter (Page 491) contains the result of this subroutine.

The D_Pos parameter contains the current absolute position returned by the servo drive.

Note

The configured measurement system setting for this Axis of Motion must be set to Engineering Units to use this subroutine.

Setup needed to harmonize the engineering unit system between the Axis of Motion and the V90 servo drive

12.7 Sample programs for the Axis of Motion

The first sample program shows a simple relative move that uses the AXISx_CTRL and AXISx_GOTO subroutines to perform a cut-to-length operation. This program does not require an RP seek mode or a motion profile, and the length can be measured in either pulses or engineering units. Enter the length (VD500) and target speed (VD504). When I0.0 (Start) turns on, the machine starts. When I0.1 (Stop) turns on, the machine finishes the current operation and stops. When I0.2 (E_Stop) turns on, the machine aborts any motion and immediately stops.

The second sample program provides an example of the AXISx_CTRL, AXISx_RUN, AXISx_RSEEK, and AXISx_MAN subroutines. You must configure the RP seek mode and a motion profile.

Open loop motion control

12.7 Sample programs for the Axis of Motion

Table 12- 32 Sample program 1: Simple relative move (cut to length application)

LAD/FBD	Description	STL
Network 1: 	Control instruction	LD SM0.0 = L60.0 LD NI0.2 = L63.7 LD L60.0 CALL AXIS0_CTRL, L63.7, M1.0, VB900, VD902, VD906, V910.0
Network 2: 	Start puts machine into automatic mode.	LD I0.0 AN I0.2 EU SQ 0.2, 1 SM 0.1, 1
Network 3: 	E_Stop: Stops immediately and turns off automatic mode.	LD I0.2 RQ 0.2, 1
Network 4: 	<ul style="list-style-type: none"> 1. Move to a certain point: 2. Enter the length to cut. 3. Enter the target speed into Speed. 4. Set the mode to 1 (Relative mode). 	LD Q0.2 = L60.0 LD M0.1 EU = L63.7 LD L60.0 CALL AXIS0_GOTO, L63.7, VD500, VD504, 1, I0.2, Q0.4, VB920, VD922, VD926

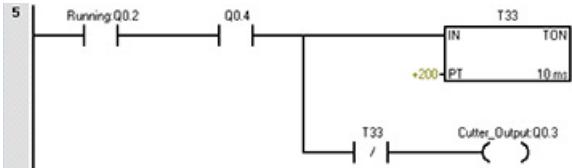
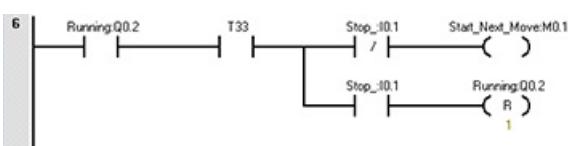
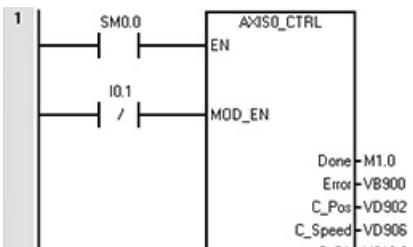
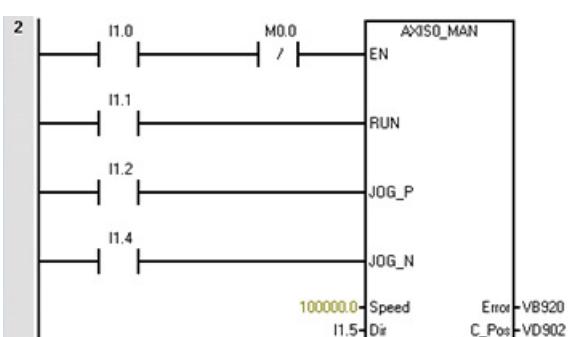
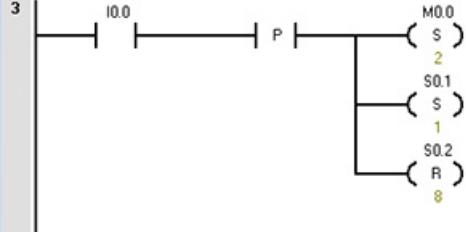
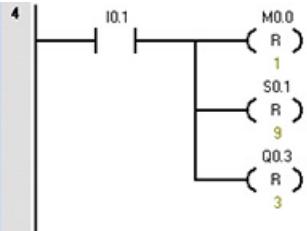
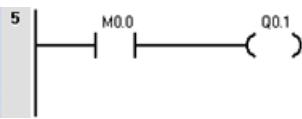
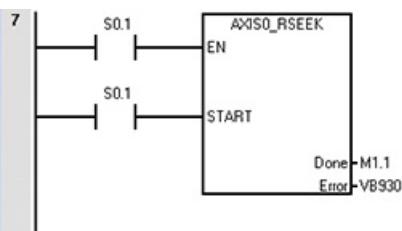
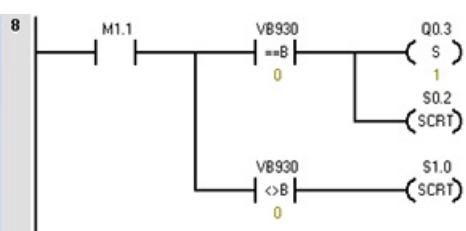
LAD/FBD	Description	STL
Network 5: 	When in position, turn on the cutter for 2 seconds to finish the cut.	LDQ0.2 AQ0.4 TONT33, +200 ANT33 = Q0.3
Network 6: 	When the cut is finished, then restart unless the Stop is active.	LDQ0.2 AT33 LPS ANIO.1 = M0.1 LPP AI0.1 RQ0.2, 1

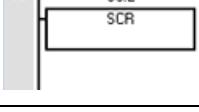
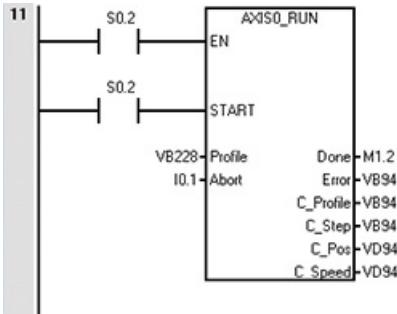
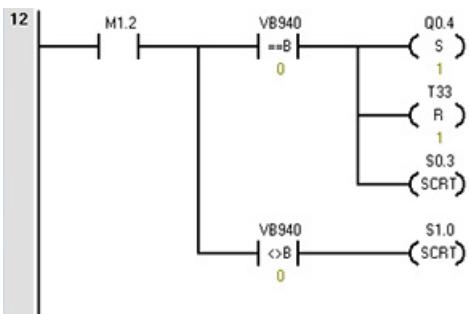
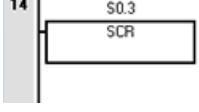
Table 12- 33 Sample program 2: Program with AXISx_CTRL, AXISx_RUN, AXISx_SEEK, and AXISx_MAN

LAD/FBD	Description	STL
Network 1 	Enable the Axis of Motion.	LDSM0.0 = L60.0 LDNI0.1 = L63.7 LDL60.0 CALL AXIS0_CTRL, L63.7, M1.0, VB900, VD902, VD906, V910.0
Network 2 	Manual mode if not in auto mode	LDI1.0 ANM0.0 = L60.0 LDI1.1 = L63.7 LDI1.2 = L63.6 LDI1.4 = L63.5 LDL60.0 CALL AXIS0_MAN, L63.7, L63.6, L63.5, +100000, I1.5, VB920, VD902, VD906, V910.0

Open loop motion control

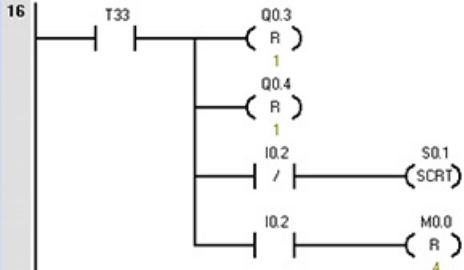
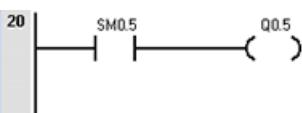
12.7 Sample programs for the Axis of Motion

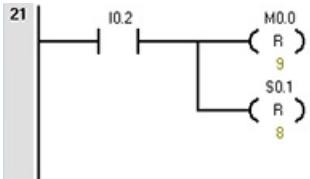
LAD/FBD	Description	STL
Network 3	Enable auto mode.	LDI0.0 EU SM0.0, 2 SS0.1, 1 RS0.2, 8
		
Network 4	Emergency Stop: Disable the Axis of Motion and auto mode.	LDI0.1 RMO.0, 1 RS0.1, 9 RQ0.3, 3
		
Network 5	When in auto mode, turn on the Running light.	LDM0.0 = Q0.1
		
Network 6		LSCR S0.1
		
Network 7	Find the reference point (RP).	LDS0.1 = L60.0 LDS0.1 = L63.7 LDL60.0 CALL AXISO_RSEEK, L63.7, M1.1, VB930
		
Network 8	When at reference point (RP): 1. Clamp the material. 2. Go to the next step.	LDM1.1 LPS AB= VB930, 0 SQ0.3, 1 SCRTS0.2 LPP AB<> VB930, 0 SCRTS1.0
		

LAD/FBD	Description	STL
Network 9 		SCRE
Network 10 		LSCR0.2
Network 11 	Use profile 1 to move into position.	LDS0.2 = L60.0 LDS0.2 = L63.7 LDL60.0 CALL AXIS0_RUN, L63.7, VB228, I0.1, M1.2, VB940, VB941, VB942, VD944, VD948
Network 12 	When positioned: 1. Turn on the cutter. 2. Go to the next step.	LDM1.2 LPS AB= VB940, 0 SQ0.4, 1 RT33, 1 SCRTS0.3 LPP AB<> VB940, 0 SCRTS1.0
Network 13 		SCRE
Network 14 	Wait for the cut to finish.	LSCR0.3

Open loop motion control

12.7 Sample programs for the Axis of Motion

LAD/FBD	Description	STL
Network 15 		LDS0.3 TONT33, +200
Network 16 	Unless STOP is on, restart when the cut is finished.	LDT33 LPS RQ0.3, 1 RQ0.4, 1 ANIO.2 SCRTS0.1 LPP A I0.2 RMO.0, 4
Network 17 		SCRE
Network 18 		LSCRS1.0
Network 19 	Reset the outputs.	LDS1.0 RQ0.3, 2
Network 20 	Flash the error light.	LDSM0.5 = Q0.5

LAD/FBD	Description	STL
Network 21 	Exit the error routine if STOP is on.	<code>LDI 0.2 RM 0.0, 9 RS 0.1, 8</code>
Network 22 		<code>SCRE</code>

12.8 Monitoring the Axis of Motion

To aid you in the development of your motion control solution, STEP 7-Micro/WIN SMART provides the Motion control panel.

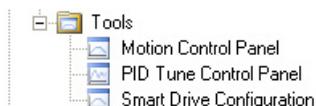
Opening the Motion control panel

To open the Motion control panel, use one of the following methods:

- Click the "Motion Control Panel" button from the Tools area of the Tools menu ribbon strip.

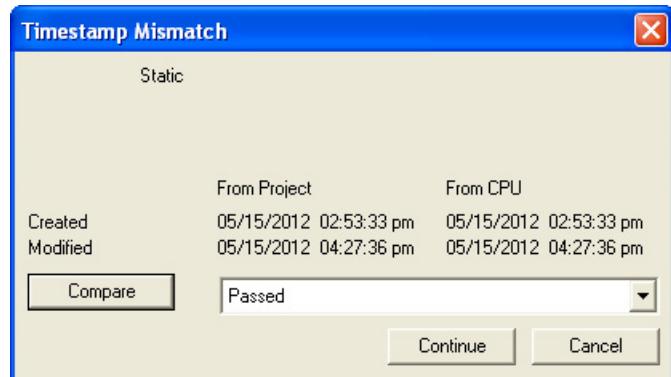


- Open the Tools folder in the project tree, select the "Motion Control Panel" node and press Enter; or double-click the "Motion Control Panel" node.



12.8 Monitoring the Axis of Motion

At this point, a comparison between the CPU and STEP 7-Micro/WIN SMART is executed to ensure that the configurations are the same. (See the figure below.)



The Axis of Motion Operation (Page 483), Configuration (Page 488), and Profile Configuration (Page 488) settings make it easy for you to monitor and control the operation of the Axis of Motion during the startup and test phases of your development process.

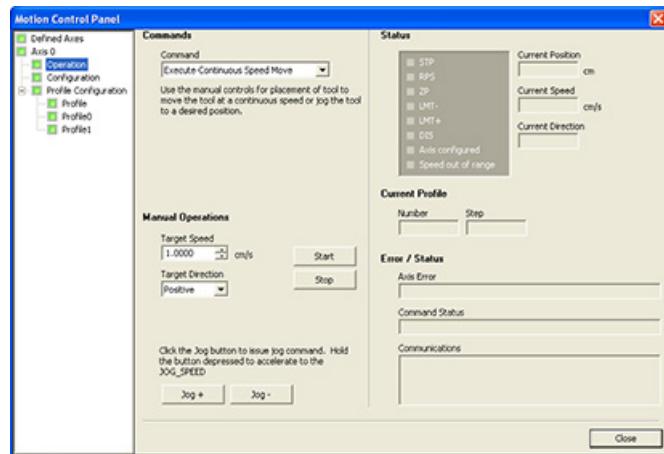
Use the Motion control panel to verify that the Axis of Motion is wired correctly, to adjust the configuration data, and to test each movement profile.

If additional changes need to be made in the Axis of Motion, refer to the Motion wizard (Page 448).

For error code listings, refer to the Axis of Motion error codes (Page 489) and the Motion instruction error codes (Page 491).

12.8.1 Displaying and controlling the operation of the Axis of Motion

In the Operation node, you can interact with the operations of the Axis of Motion. The control panel displays the current speed, the current position, and the current direction of the Axis of Motion. You can also see the status of the input and output LEDs (excluding the Pulse LEDs).



The control panel allows you to interact with the Axis of Motion by changing the speed and direction, by stopping and starting the motion, and by jogging the tool (if the CPU is stopped).

Note

You cannot execute a motion command while the CPU is running. The CPU must be in STOP mode in order to change the speed and direction, stop and start the motion, and use the jog tool.

Note

Exiting the Motion control panel or a loss of communications while a motion command is active causes the axis to stop its motion after a 5 second timeout.

Motion commands

You can also generate the following motion commands:

Table 12- 34 Motion control panel commands

Command	Description
<p>Commands</p> <p>Command <input type="button" value="Execute Continuous Speed Move"/></p> <p>Use the manual controls for placement of tool to move the tool at a continuous speed or jog the tool to a desired position.</p> <p>Manual Operations</p> <p>Target Speed <input type="text" value="0.2000"/> cm/s <input type="button" value="Start"/></p> <p>Target Direction <input type="button" value="Positive"/></p> <p>Click the Jog button to issue jog command. Hold the button depressed to accelerate to the JOG_SPEED</p> <p><input type="button" value="Jog +"/> <input type="button" value="Jog -"/></p>	Execute continuous speed move: This command allows you to use the manual controls for positioning the tool. Enter "Target Speed" and "Direction", and click "Start" to execute continuous move. Motion will continue until "Stop" is clicked (or error condition).
<p>Commands</p> <p>Command <input type="button" value="Seek to reference point"/></p> <p>Click Execute and the axis will be commanded to seek a reference point, using the search algorithm specified in the module configuration.</p> <p><input type="button" value="Execute"/> <input type="button" value="Abort"/></p>	Seek to a reference point: This command finds the reference point by using the configured search mode. Click "Execute", and the axis will command a "Seek to Reference Point", using the search algorithm specified in the axis configuration. Click "Abort" to stop a seek process before the reference point has been found.

Command	Description
<p>Commands</p> <p>Command <input type="button" value="Load reference point offset"/></p> <p>Use the manual controls to place the tool at the new zero position. Click Execute button to save this position as the RP_OFFSET. The current position will be zero.</p> <p><input type="button" value="Execute"/> <input type="button" value="Abort"/></p> <p>Manual Operations</p> <p>Target Speed <input type="text" value="0.2000"/> cm/s <input type="button" value="Start"/></p> <p>Target Direction <input type="button" value="Positive"/> <input type="button" value="Stop"/></p> <p>Click the Jog button to issue jog command. Hold the button depressed to accelerate to the JOG_SPEED</p> <p><input type="button" value="Jog +"/> <input type="button" value="Jog -"/></p>	<p>Load reference point offset: After you use the manual controls to jog the tool to the new position, you then load the "Reference Point Offset". Use the manual controls to place the tool at the new position. Click "Execute" to save this position as the "RP_OFFSET". The current position will be set to zero.</p>
<p>Commands</p> <p>Command <input type="button" value="Reload current position"/></p> <p>Enter the position to set and click execute button to update current position. This will also establish a new zero position.</p> <p>New Position <input type="text" value="1.0000"/> cm</p> <p><input type="button" value="Execute"/></p>	<p>Reload current position: This command updates the current position value and establishes a new zero position. Enter the position to set and click "Execute" to update the current position. This will also establish a new zero position.</p>
<p>Commands</p> <p>Command <input type="button" value="Activate DIS output"/></p> <p>Click execute button to activate the DIS output.</p> <p><input type="button" value="Execute"/></p>	<p>Activate the DIS output: This command turns the DIS output of the Axis of Motion on. Click "Execute" to activate the DIS output.</p>

Open loop motion control

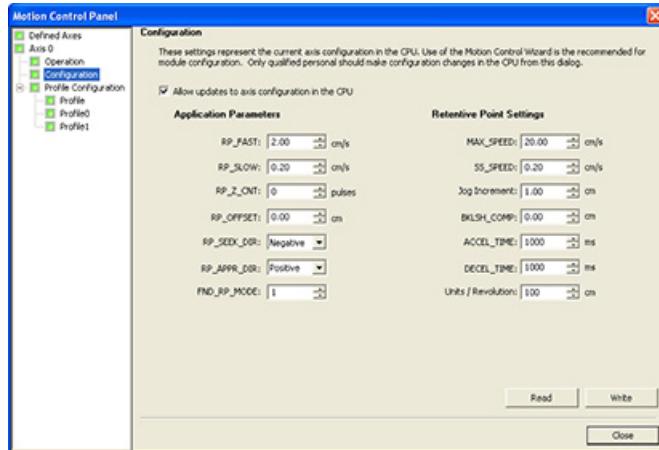
12.8 Monitoring the Axis of Motion

Command	Description
Commands <p>Command <input type="button" value="Deactivate DIS output"/></p> <p>Click execute to deactivate the DIS output.</p> <p><input type="button" value="Execute"/></p>	De-activate the DIS output: This command turns the DIS output of the Axis of Motion off. Click "Execute" to de-activate the DIS output.
Commands <p>Command <input type="button" value="Load axis configuration"/></p> <p>Click execute button to have the module read its configuration from V-Memory.</p> <p><input type="button" value="Execute"/></p>	Load axis configuration: This command loads a new configuration by commanding the Axis of Motion to read the configuration block from the V memory of the CPU. Click "Execute" to have the axis read its configuration from V memory.
Commands <p>Command <input type="button" value="Move to an absolute position"/></p> <p>Specify a target speed and the absolute position to move to. This option requires that the zero position be defined.</p> <p> </p> Manual Operations <p>Target Speed <input type="text" value="0.2000"/> cm/s <input type="button" value="Start"/> <input type="button" value="Stop"/></p> <p>Target Position <input type="text" value="1.0000"/> cm</p> <p>Click the Jog button to issue jog command. Hold the button depressed to accelerate to the JOG_SPEED</p> <p><input type="button" value="Jog +"/> <input type="button" value="Jog -"/></p>	Move to an absolute position: This command allows you to move to a specified position at a target speed. Before using this command, you must have already established the zero position. Assign a "Target Speed" and the "Absolute Position" to move to. This option requires that the zero position be defined.

Command	Description
<p>Commands</p> <p>Command <input type="button" value="Move by a relative amount"/></p> <p>Specify a target speed and the distance from the current position that you wish to move. You may specify either a positive or negative distance.</p> <p>Manual Operations</p> <p>Target Speed <input type="text" value="0.2000"/> cm/s <input type="button" value="Start"/> <input type="button" value="Stop"/></p> <p>Target Position <input type="text" value="1.0000"/> cm Click the Jog button to issue jog command. Hold the button depressed to accelerate to the JOG_SPEED <input type="button" value="Jog +"/> <input type="button" value="Jog -"/></p>	Move by a relative amount: This command allows you to move a specified distance from the current position at a target speed. You can enter a positive or negative distance. Assign a "Target Speed" and a "Target Position" to move to.
<p>Commands</p> <p>Command <input type="button" value="Reset the axis command interface"/></p> <p>This option is useful if the axis appears unresponsive to commands. This option clears the command byte for the axis and sets the bit.</p> <p><input type="button" value="Execute"/></p>	Reset the axis command interface: This command clears the axis command interface for the Axis of Motion and sets the "Done" bit. Use this command if the Axis of Motion appears to not be responding to commands.
<p>Commands</p> <p>Command <input type="button" value="Execute profile"/></p> <p>Set profile number and click execute button to start running the profile.</p> <p>Profile Number <input type="button" value="Profile"/></p> <p><input type="button" value="Execute"/> <input type="button" value="Abort"/></p>	Execute profile: This command allows you to select a profile to be executed. The control panel displays the status of the profile which is being executed by the Axis of Motion. Select the profile that is to be executed, then click "Execute", and the axis will command the profile. Note: This command is only available if a profile has been defined in the Motion wizard.

12.8.2 Displaying and modifying the configuration of the Axis of Motion

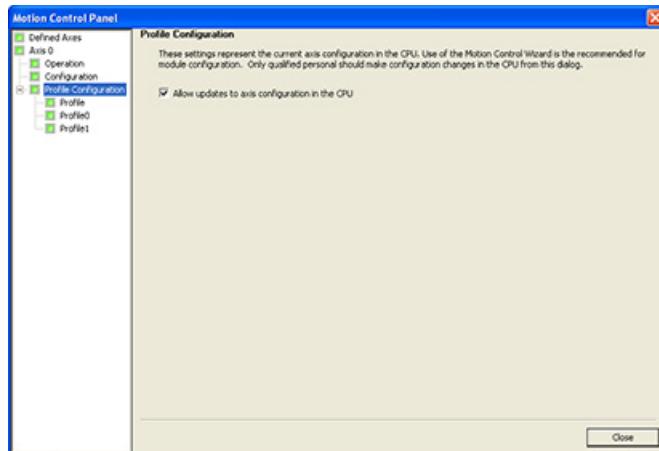
In the Configuration node, you can view and modify the configuration settings for the Axis of Motion that is stored in the data block of the CPU.



After you modify the configuration settings, you simply click the write button to send the data values to the CPU. These data values are not saved in your STEP 7-Micro/WIN SMART project. You must manually make changes to your project that reflects the final values of these fields.

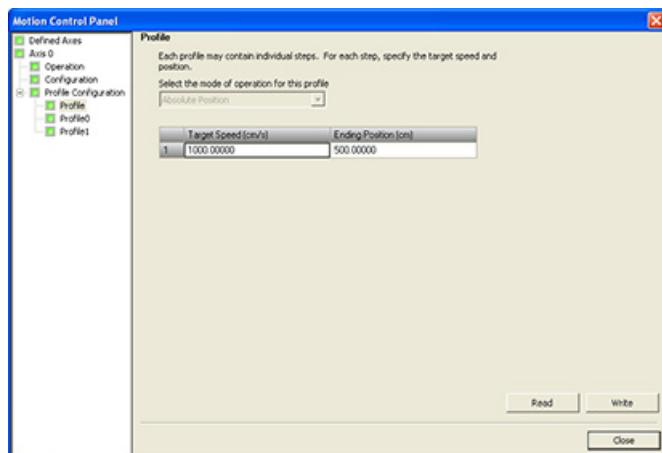
12.8.3 Displaying the profile configuration for the Axis of Motion

In the Profile Configuration node, you can view the configuration of each profile for the Axis of Motion.



Click each profile to view its mode of operation and data values.

Some data values of the profile can be modified in this dialog. After you modify the configuration settings, you simply click the write button to send the data values to the CPU. These data values are not saved in your STEP 7-Micro/WIN SMART project. You must manually make changes to your project that reflects the final values of these fields.



12.8.4 Error codes for the Axis of Motion (WORD at SMW620, SMW670, or SMW720)

Table 12- 35 Axis of Motion error codes

Error code	Description
0	No error
1	Reserved
2	Configuration block not present
3	Configuration block pointer error
4	Size of configuration block exceeds available V memory
5	Illegal configuration block format
6	Too many profiles specified
7	Illegal STP_RSP specification
8	Illegal LIM- specification
9	Illegal LIM+ specification
10	Illegal FILTER_TIME specification
11	Illegal MEAS_SYS specification
12	Illegal RP_CFG specification
13	Illegal PLS/REV value
14	Illegal UNITS/REV value
15	Illegal RP_ZP_CNT value
16	Illegal JOG_INCREMENT value
17	Illegal MAX_SPEED value

Error code	Description
18	Illegal SS_SPD value
19	Illegal RP_FAST value
20	Illegal RP_SLOW value
21	Illegal JOG_SPEED value
22	Illegal ACCEL_TIME value
23	Illegal DECEL_TIME value
24	Illegal JERK_TIME value
25	Illegal BKLSH_COMP value
26	AXIS not available
27	Invalid LMT+ location
28	Invalid LMT- location
29	Invalid STP location
30	Invalid RPS location
31	Invalid ZP location
32	Illegal output phase
33	No RPS input defined (If Homing is defined, then RPS must be defined also.)
34	Invalid TRIG location
35	Reserved
36	No ZP input defined (If Homing mode 3 or 4 is defined, then ZP must be defined also.)
37	Phase A (P0) output conflict
38	Phase B (P1) output conflict
39	DIS output conflict
40	Reserved
41	Invalid SDB0 record size
42	Illegal SDB0 format
43 to 127	Reserved

To verify that the Axis of Motion is wired correctly, to adjust the configuration data, and to test each movement profile, use the Motion control panel.

If additional changes need to be made in the Axis of Motion, go to the Motion wizard.

12.8.5 Error codes for the Motion instruction (seven LS bits of SMB634, SMB684, or SMB734)

In the SM table for each axis there is a byte reserved to display the result of the motion instruction (Offset 34). This byte indicates when an instruction is complete and if there was an error in the instruction.

Table 12- 36 Motion instruction error codes

Error code	Description
0	No error
1	Aborted by user
2	Configuration error (This error occurs if there is an error in the SDB0 configuration.)
3	Illegal command
4	Aborted due to no valid configuration (This error occurs if there is an error in the configuration table.)
5	Reserved
6	Aborted due to no defined reference point
7	Aborted due to STP input active
8	Aborted due to LMT- input active
9	Aborted due to LMT+ input active
10	Aborted due to problem executing motion
11	No profile block configured for specified profile
12	Illegal operation mode
13	Operation mode not supported for this command
14	Illegal number of steps in profile block
15	Illegal direction change
16	Illegal distance
17	RPS/TRIG trigger occurred before target speed reached
18	Insufficient RPS active region width
19	Speed out-of-range
20	Insufficient distance to perform desired speed change
21	Illegal position
22	Zero position unknown
23	No DIS output is defined
24	Reserved
25	Aborted due to CPU going to stop
26	Aborted due to expiration of Motion control panel heartbeat
27 to 127	Reserved
128	Axis of Motion cannot process this instruction: either the Axis of Motion is busy with another instruction, or there was no Start pulse on this instruction.
129	Reserved
130	Axis of Motion is not enabled
131	Reserved

Error code	Description
132	Reserved
133	Illegal profile specified. The AXISx_RUN and AXISx_CACHE instructions profile number range must be between 0 - 31.
134	Illegal mod specified in AXISx_GOTO instruction

To verify that the Axis of Motion is wired correctly, to adjust the configuration data, and to test each movement profile, use the Motion control panel.

If additional changes need to be made in the Axis of Motion, go to the Motion wizard.

12.9 Advanced topics

12.9.1 Understanding the configuration/profile table for the Axis of Motion

Overview

The Motion wizard has been developed to make motion applications easy by automatically generating the configuration and profile information based upon the answers you give about your motion control system. Configuration/profile table information is provided for advanced users who want to create their own motion control routines.

The configuration/profile table is located in the V memory area of the S7-200 SMART CPU. As shown in the table below, the configuration settings are stored in the following types of information:

- **Configuration block:** Contains information used to setup the Axis of Motion in preparation for executing position commands
- **Interactive block:** Supports direct setup of position parameters by the user program
- **Profile block:** Describes a pre-defined move operation to be performed by the Axis of Motion. You can configure up to 32 profile blocks.

Note

The profile block of the configuration/profile table can contain up to 32 motion profiles. To create more than 32 move profiles, you can exchange configuration/profile tables by changing the value stored in the configuration/profile table pointer.

Table 12- 37 Configuration/Profile table: Configuration block

Configuration/Profile table			
Byte offset	Name	Function description	Type
Configuration block			
0	MOD_ID	Axis of Motion identification field	--
5	CB_LEN	Length of the configuration block in bytes (1 byte)	--
6	IB_LEN	Length of the interactive block in bytes (1 byte)	--
7	PF_LEN	Length of a single profile in bytes (1 byte)	--
8	STP_LEN	Length of a single step in bytes (1 byte)	--
9	STEPS	Number of steps allowed per profile (1 byte)	--
10	PROFILES	Number of profiles from 0 to 32 (1 byte)	--
11	--	Reserved: Set to 0	--
13	--	Reserved: Set to 0	--
14	STOP_RSP	Specifies the drive's response to the STP input (1 byte): <ul style="list-style-type: none"> • 0: No action. Ignore input condition. • 1: Decelerate to a stop and indicate STP input active. • 2: Terminate pulses and indicate STP input. • 3 to 255: Reserved (error if specified) 	--
15	LMT-_RSP	Specifies the drive's response to the negative limit input (1 byte): <ul style="list-style-type: none"> • 0: No action. Ignore input condition. • 1: Decelerate to a stop and indicate limit reached. • 2: Terminate pulses and indicate limit reached. • 3 to 255: Reserved (error if specified) 	--
16	LMT+_RSP	Specifies the drive's response to the positive limit input (1 byte): <ul style="list-style-type: none"> • 0: No action. Ignore input condition. • 1: Decelerate to a stop and indicate limit reached. • 2: Terminate pulses and indicate limit reached. • 3 to 255: Reserved (error if specified) 	--
17	--	Reserved: Set to 0	--
18	MEAS_SYS	Specifies the measurement system used to describe moves (1 byte): <ul style="list-style-type: none"> • 0: Pulses (speed measured in pulses/sec and position values measured in pulses; values are double integer) • 1: Engineering units (speed measured in units/sec and position values measured in units; values are single precision real) • 2 to 255: Reserved (error if specified) 	--

Configuration/Profile table																																																	
Byte offset	Name	Function description	Type																																														
Configuration block																																																	
19	--	Reserved: Set to 0	--																																														
20	PLS/REV	Specifies the number of pulses per revolution of the motor, (only applicable when MEAS_SYS is set to 1) - (4 bytes)	DInt																																														
24	UNITS/REV	Specifies the engineering units per revolution of the motor, (only applicable when MEAS_SYS is set to 1) - (4 bytes)	Real																																														
28	UNITS	Reserved for STEP 7-Micro/WIN SMART to store a custom units string (4 bytes)	--																																														
32	RP_CFG	<p>Specifies the reference point search configuration (1 byte):</p> <table border="1"> <tr> <td>MSB 7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>LSB 0</td> </tr> <tr> <td></td> <td></td> <td>0</td> <td>0</td> <td colspan="3">MODE</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>↑</td> <td>↑</td> <td>↑</td> <td></td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td></td> <td>RP_SEEK_DIR</td> <td>RP_APPR_DIR</td> <td></td> </tr> </table> <p>RP_SEEK_DIR: This bit specifies the starting direction for a reference point search (0 - positive direction, 1 - negative direction).</p> <p>RP_APPR_DIR: This bit specifies the approach direction for terminating the reference point search (0 - positive direction, 1 - negative direction).</p> <table border="1"> <tr> <td>MODE</td> <td>Specifies the reference point search method</td> </tr> <tr> <td>'0000'</td> <td>Reference point search disabled.</td> </tr> <tr> <td>'0001'</td> <td>The reference point is where the RPS input goes active.</td> </tr> <tr> <td>'0010'</td> <td>The reference point is centered within the active region of the RPS input.</td> </tr> <tr> <td>'0011'</td> <td>The reference point is outside the active region of the RPS input.</td> </tr> <tr> <td>'0100'</td> <td>The reference point is within the active region of the RPS input.</td> </tr> <tr> <td>'0101' to ' '1111'</td> <td>Reserved (error if selected)</td> </tr> </table>	MSB 7	6	5	4	3	2	1	LSB 0			0	0	MODE								↑	↑	↑							RP_SEEK_DIR	RP_APPR_DIR		MODE	Specifies the reference point search method	'0000'	Reference point search disabled.	'0001'	The reference point is where the RPS input goes active.	'0010'	The reference point is centered within the active region of the RPS input.	'0011'	The reference point is outside the active region of the RPS input.	'0100'	The reference point is within the active region of the RPS input.	'0101' to ' '1111'	Reserved (error if selected)	--
MSB 7	6	5	4	3	2	1	LSB 0																																										
		0	0	MODE																																													
				↑	↑	↑																																											
					RP_SEEK_DIR	RP_APPR_DIR																																											
MODE	Specifies the reference point search method																																																
'0000'	Reference point search disabled.																																																
'0001'	The reference point is where the RPS input goes active.																																																
'0010'	The reference point is centered within the active region of the RPS input.																																																
'0011'	The reference point is outside the active region of the RPS input.																																																
'0100'	The reference point is within the active region of the RPS input.																																																
'0101' to ' '1111'	Reserved (error if selected)																																																
33	--	Reserved: Set to 0	--																																														
34	RP_Z_CNT	Number of pulses of the ZP input used to define the reference point (4 bytes)	DInt																																														
38	RP_FAST	Fast speed for the RP seek operation: MAX_SPD or less (4 bytes)	DInt/Real																																														

Configuration/Profile table			
Byte offset	Name	Function description	Type
Configuration block			
42	RP_SLOW	Slow speed for the RP seek operation: Maximum speed from which the motor can instantly go to a stop or less (4 bytes)	DInt/Real
46	SS_SPEED	Start/Stop Speed. (4 bytes): The starting speed is the maximum speed to which the motor can instantly go from a stop and the maximum speed from which the motor can instantly go to a stop. Operation below this speed is allowed, but the acceleration and deceleration times do not apply.	DInt/Real
50	MAX_SPEED	Maximum operating speed of the motor (4 bytes)	DInt/Real
54	JOG_SPEED	Jog speed (4 bytes): MAX_SPEED or less (4 bytes)	DInt/Real
58	JOG_INCREMENT	Jog increment value: The distance (or number of pulses) to move in response to a single jog pulse (4 bytes).	DInt/Real
62	ACCEL_TIME	Time required to accelerate from minimum to maximum speed in msec (4 bytes)	DInt
66	DECEL_TIME	Time required to decelerate from maximum to minimum speed in msec (4 bytes)	DInt
70	BKLSH_COMP	Backlash compensation: The distance used to compensate for the system backlash on a direction change (4 bytes).	DInt/Real
74	JERK_TIME	Time during which jerk compensation is applied to the beginning and ending portions of an acceleration/deceleration curve (S-curve). Specifying a value of 0 disables jerk compensation. The jerk time is given in milliseconds (Range: 0 ms to 32000 ms. (4 bytes)	DInt

Table 12- 38 Configuration/Profile table: Interactive block

Configuration/Profile table			
Byte offset	Name	Function description	Type
Interactive block			
78	MOVE_CMD	Selects the mode of operation (1 byte): <ul style="list-style-type: none"> • 0: Absolute position • 1: Relative position • 2: Single-speed, continuous positive rotation • 3: Single-speed, continuous negative rotation • 4: Manual speed control, positive rotation • 5: Manual speed control, negative rotation • 6: Single-speed, continuous positive rotation with triggered stop (Activation of RPS triggers the stop; TARGET_POS contains distance to travel after signal) • 7: Single-speed, continuous negative rotation with triggered stop (Activation of RPS triggers the stop; TARGET_POS contains distance to travel after signal) • 8 to 255: Reserved (error if specified) 	--
79	--	Reserved: Set to 0	--
80	TGT_POS	Target position to go to in this move (4 bytes)	DInt/Real
84	TGT_SPEED	Target speed for this move (4 bytes)	DInt/Real
88	RP_OFFSET	Absolute position of the reference point (4 bytes)	DInt/Real

Table 12- 39 Configuration/Profile table: Profile block 0

Configuration/Profile table			
Byte offset	Name	Function description	Type
Profile block 0			
92(+0)	STEPS	Number of steps in this move sequence (1 byte)	--
93(+1)	MODE	<p>Selects the mode of operation for this profile block (1 byte):</p> <ul style="list-style-type: none"> • 0: Absolute position • 1: Relative position • 2: Single-speed, continuous positive rotation • 3: Single-speed, continuous negative rotation • 4: Reserved (error if specified) • 5: Reserved (error if specified) • 6: Single-speed, continuous positive rotation with triggered stop (RPS input signals stop) • 7: Single-speed, continuous negative rotation with triggered stop (RPS input signals stop) • 8: Two-speed, continuous positive rotation RPS selects speed) • 9: Two-speed, continuous negative rotation (RPS selects speed) • 10: Two-speed, continuous positive rotation with triggered stop (RPS selects speed, TRIG input signals stop) • 11: Two-speed, continuous negative rotation with triggered stop (RPS selects speed, TRIG input signals stop) • 12 to 255: Reserved (error if specified) 	--
94(+2)	Step 0: POS	Position to go to in move step 0 (4 bytes)	DInt/Real
98(+6)	Step 0: SPEED	Target speed for move step 0 (4 bytes)	DInt/Real
102(+10)	Step 1: POS	Position to go to in move step 1 (4 bytes)	DInt/Real
106(+14)	Step 1: SPEED	Target speed for move step 1 (4 bytes)	DInt/Real
110(+18)	Step 2: POS	Position to go to in move step 2 (4 bytes)	DInt/Real
114(+22)	Step 2: SPEED	Target speed for move step 2 (4 bytes)	DInt/Real
118(+26)	Step 3: POS	Position to go to in move step 3 (4 bytes)	DInt/Real
122(+30)	Step 3: SPEED	Target speed for move step 3 (4 bytes)	DInt/Real
...	Step ...: POS	Position to go to in move step ... (4 bytes)	DInt/Real
...	Step ...: SPEED	Target speed for move step ... (4 bytes)	DInt/Real
214(+122)	Step 15: POS	Position to go to in move step 3 (4 bytes)	DInt/Real
218(+126)	Step 15: SPEED	Target speed for move step 3 (4 bytes)	DInt/Real

Note

You can have between 1 and 16 steps in the Configuration/Profile table, Profile block 0.

Table 12- 40 Configuration/Profile table: Profile block 1

Configuration/Profile table			
Byte offset	Name	Function description	Type
Profile block 1			
X ¹	STEPS	Number of steps in this move sequence (1 byte) Note: There can be up to 16 steps.	--
(X + 1)	MODE	Selects the mode of operation for this profile block (1 byte)	--
(X + 2)	Step 0: POS	Position to go to in move step 0 (4 bytes)	DInt/Real
(X + 4)	Step 0: SPEED	The target speed for move step 0 (4 bytes)	DInt/Real
...

¹ The offset of Profile block 1 and subsequent blocks is variable and dependent upon the number of steps configured in the largest profile. The offset is determined by the following formula:

$$\text{Offset of Profile block } x = \text{CB_LEN} + \text{IB_LEN} + (x * \text{PF_LEN})$$

Table 12- 41 Profile detail for Mode 0 (Absolute position)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	n = Number of steps configured in this profile
+1		MODE	byte	0 = Absolute position
+2	0	POS	dint/fp	Destination position in step 0
+6		SPEED	dint/fp	Target speed for step 0
•				
•				
•				
(4 * n) + 2	n	POS	dint/fp	Destination position in step n
(\$ * N) + 6		SPEED	dint/fp	Target speed for step n

Table 12- 42 Profile detail for Mode 1 (Relative position)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	n = Number of steps configured in this profile
+1		MODE	byte	0 = Relative position
+2	0	POS	dint/fp	Distance to travel in step 0
+6		SPEED	dint/fp	Target speed for step 0
•				
•				
•				
(4 * n) + 2	n	POS	dint/fp	Distance to travel in step n
(\$ * N) + 6		SPEED	dint/fp	Target speed for step n

Table 12- 43 Profile detail for Mode 2 (Single-speed, continuous positive rotation) and Mode 3 (Single-speed, continuous negative rotation)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	1
+1		MODE	byte	2 = Single-speed, continuous positive rotation or 3 = Single-speed, continuous negative rotation
+2	0	POS	dint/fp	n.a. (must be set to 0)
+6		SPEED	dint/fp	Target speed

Table 12- 44 Profile detail for Mode 6 (Single-speed, continuous positive rotation with triggered stop) and Mode 7 (Single-speed, continuous negative rotation with triggered stop)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	1
+1		MODE	byte	6 = Single-speed, continuous positive rotation with triggered stop or 7 = Single-speed, continuous negative rotation with triggered stop
+2	0	POS	dint/fp	Distance to travel after activation of RPS signal (value must be positive)
+6		SPEED	dint/fp	Target speed

Table 12- 45 Profile detail for Mode 8 (Two-speed, continuous positive rotation) and Mode 9 (Two-speed, continuous negative rotation)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	2
+1		MODE	byte	8 = Two-speed, continuous positive rotation or 9 = Two-speed, continuous negative rotation
+2	0	POS	dint/fp	n.a. (must be set to 0)
+6		SPEED	dint/fp	Target speed if RPS signal is inactive
+10	1	POS	dint/fp	n.a. (must be set to 0)
+14		SPEED	dint/fp	Target speed if RPS signal is active

Table 12- 46 Profile detail for Mode 10 (Two-speed, continuous positive rotation with triggered stop) andMode 11 (Two-speed, continuous negative rotation with triggered stop)

Byte offset from start of profile	Step number	Name	Field size	Value
+0		STEPS	byte	2
+1		MODE	byte	10 = Two-speed, continuous positive rotation with triggered stop or 11 = Two-speed, continuous negative rotation with triggered stop
+2	0	POS	dint/fp	Distance to travel after activation of TRIG signal (value must be positive)
+6		SPEED	dint/fp	Target speed if RPS signal is inactive
+10	1	POS	dint/fp	n.a. (must be set to 0)
+14		SPEED	dint/fp	Target speed if RPS signal is active

12.9.2 Special memory (SM) locations for the Axis of Motion

The CPU allocates 50 bytes of special memory (SM) to each Axis of Motion. (See the following table.) When the Axis of Motion detects an error condition or a change in status of the data, the Axis of Motion updates these SM locations. The first Axis of Motion updates SMB600 through SMB649 as required to report the error condition, the second Axis of Motion updates SMB650 through SMB699, and so on.

Table 12- 47 Special memory bytes SMB600 to SMB749

SM bytes for Axes of Motion:		
Axis of Motion 0	Axis of Motion 1	Axis of Motion 2
SMB600 to SMB649	SMB650 to SMB699	SMB700 to SMB749

The following table shows the structure of the SM data area allocated for an Axis of Motion. The definition uses Axis 0 as an example.

Table 12- 48 Special memory area definition for the Axis of Motion 0

SM address	Description																		
SMB600 to SMB615	Axis name (16 ASCII characters). SMB600 is the first character: "Axis 0"																		
SMB616 to SMB619	Reserved																		
SMW620	Axis 0: Error code (See "Axis of Motion error codes" (Page 489) list.)																		
SMB622	<p>Axis 0: Input/output status: Reflects the status of the inputs and outputs</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: right;">MSB</td> <td>7</td> <td>6</td> <td>5</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td style="text-align: left;">LSB</td> </tr> <tr> <td></td> <td>DIS</td> <td>0</td> <td>TRIG</td> <td>STP</td> <td>LMT-</td> <td>LMT+</td> <td>RPS</td> <td>ZP</td> </tr> </table> <ul style="list-style-type: none"> • DIS (Disable outputs): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow • TRIG (Stop input): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow • STP (Stop input): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow • LMT- (Negative travel limit input): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow • LMT+ (Positive travel limit input): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow • RPS (Reference point switch input): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow • ZP (Zero pulse input): <ul style="list-style-type: none"> – 0 = No current flow – 1 = Current flow 	MSB	7	6	5	4	3	2	1	LSB		DIS	0	TRIG	STP	LMT-	LMT+	RPS	ZP
MSB	7	6	5	4	3	2	1	LSB											
	DIS	0	TRIG	STP	LMT-	LMT+	RPS	ZP											

SM address	Description																		
SMB623	<p>Axis 0 Instantaneous status: Reflects the status of the configuration and rotation direction status</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">MSB</td> <td style="text-align: center;">7</td> <td style="text-align: center;">6</td> <td style="text-align: center;">5</td> <td style="text-align: center;">4</td> <td style="text-align: center;">3</td> <td style="text-align: center;">2</td> <td style="text-align: center;">1</td> <td style="text-align: center;">LSB</td> </tr> <tr> <td></td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>OR</td> <td>R</td> <td>CFG</td> </tr> </table> <ul style="list-style-type: none"> • OR (Target speed out of range): <ul style="list-style-type: none"> – 0 = In range – 1 = Out of range • R (Direction of rotation): <ul style="list-style-type: none"> – 0 = Positive rotation – 1 = Negative rotation • CFG (Module configured): <ul style="list-style-type: none"> – 0 = Not configured – 1 = Configured 	MSB	7	6	5	4	3	2	1	LSB		0	0	0	0	0	OR	R	CFG
MSB	7	6	5	4	3	2	1	LSB											
	0	0	0	0	0	OR	R	CFG											
SMB624	Axis 0: CUR_PF is a byte that indicates the profile currently being executed.																		
SMB625	Axis 0: CUR_STP is a byte that indicates the step currently being executed in the profile.																		
SMD626	Axis 0: CUR_POS is a double-word value that indicates the current position of the Axis of Motion.																		
SMD630	Axis 0: CUR_SPD is a double-word value that indicates the current speed of the Axis of Motion.																		
SMB634	<p>Axis 0: Result of the instruction. Error conditions above 127 are generated by the instruction subroutines created by the Motion wizard.</p> <table style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">MSB</td> <td style="text-align: center;">7</td> <td style="text-align: center;">6</td> <td style="text-align: center;">LSB</td> <td style="text-align: center;">0</td> </tr> <tr> <td></td> <td>D</td> <td>ERROR</td> <td></td> <td></td> </tr> </table> <ul style="list-style-type: none"> • D (Done bit): <ul style="list-style-type: none"> – 0= Operation in progress – 1= Operation complete (set by the Axis of Motion during initialization) • ERROR: (See "Motion instruction error codes" (Page 491) list.) 	MSB	7	6	LSB	0		D	ERROR										
MSB	7	6	LSB	0															
	D	ERROR																	
SMB635 to SMB645	Reserved																		
SMD646	Axis 0: Pointer to the V memory location of the configuration/profile table. A pointer value to an area other than V memory is not valid. The Axis of Motion monitors this location until it receives a non-zero pointer value.																		

12.10 Understanding the RP Seek modes of the Axis of Motion

The following figures provide diagrams of the different options for each RP Seek mode:

- RP Seek: Mode 1 shows two of the options for RP Seek mode 1. This mode locates the RP where the RPS input goes active on the approach from the work zone side.
- RP Seek: Mode 2 shows two of the options for RP Seek mode 2. This mode locates the RP in the center within the active region of the RPS input.
- RP Seek: Mode 3 shows two of the options for RP Seek mode 3. This mode locates the RP a specified number of zero pulses (ZP) outside the active region of the RPS input.
- RP Seek: Mode 4 shows two of the options for RP Seek mode 4. This mode locates the RP a specified number of zero pulses (ZP) within the active region of the RPS input.

For each mode, there are four combinations of RP Seek direction and RP Approach direction. (Only two of the combinations are shown.) These combinations determine the pattern for the RP Seek operation. For each of the combinations, there are also four different starting points:

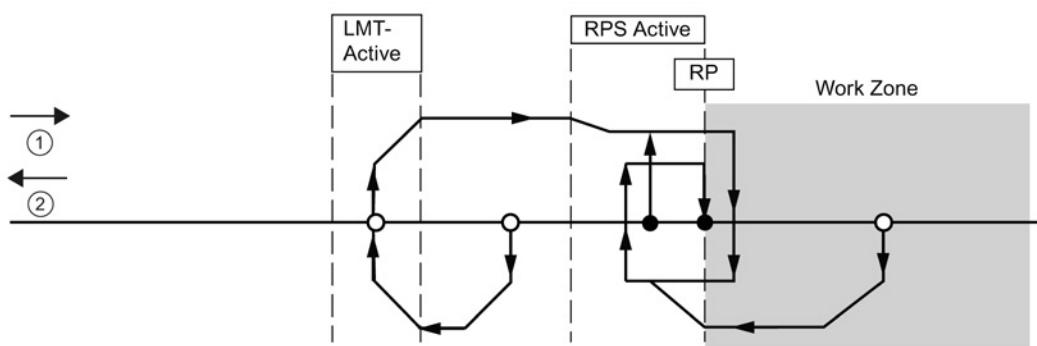
The work zones for each diagram have been located so that moving from the reference point to the work zone requires movement in the same direction as the RP Approach Direction. By selecting the location of the work zone in this way, all the backlash of the mechanical gearing system is removed for the first move to the work zone after a reference point seek.

Note

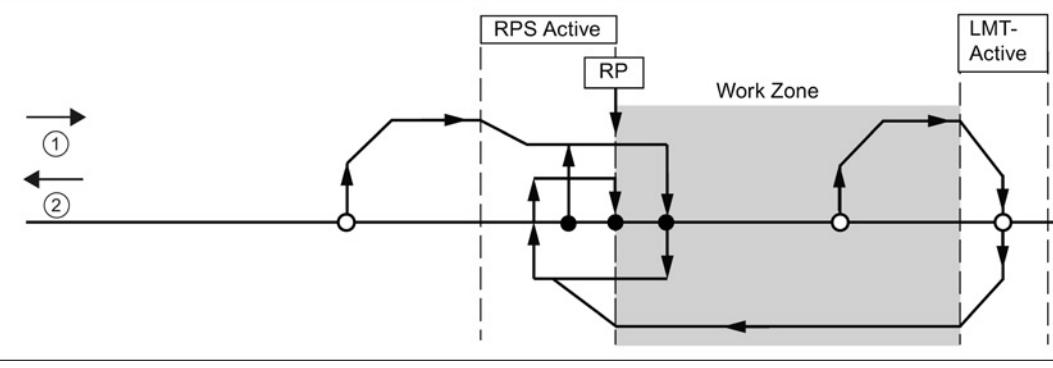
The RPS input must be enabled to use the RP Seek functionality. The ZP input must also be enabled if RP Seek modes 3 or 4 are to be used, unless you configure the number of ZP pulses to be received after entering the RPS active region to a value of "0".

RP seek mode 1

Default configuration: RP seek direction: negative and RP approach direction: positive

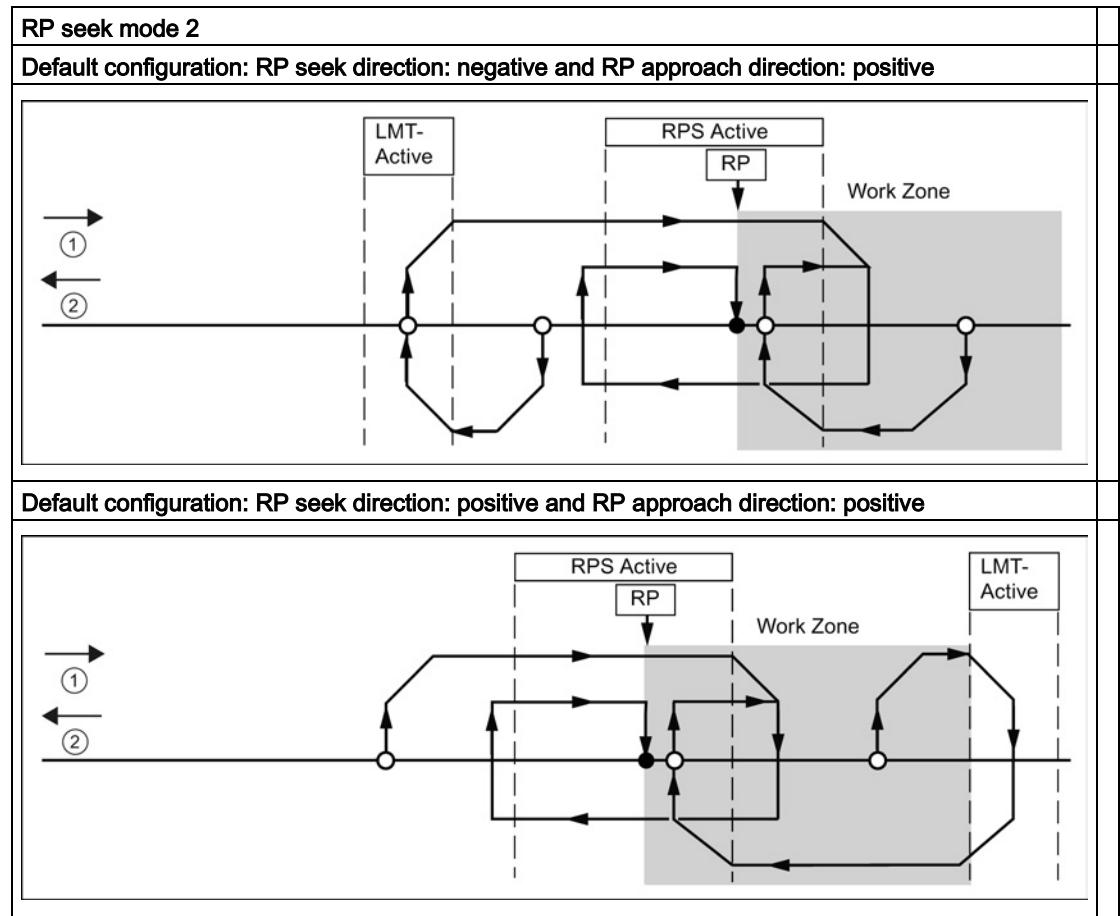


Default configuration: RP seek direction: positive and RP approach direction: positive



①: Positive motion

②: Negative motion

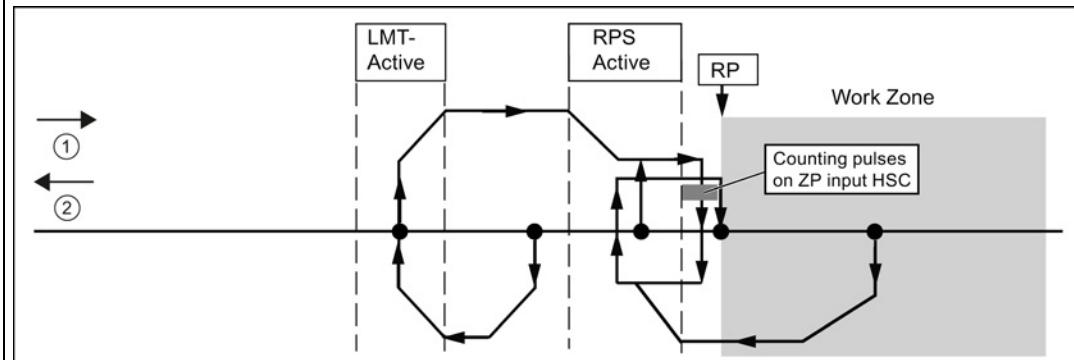


①: Positive motion

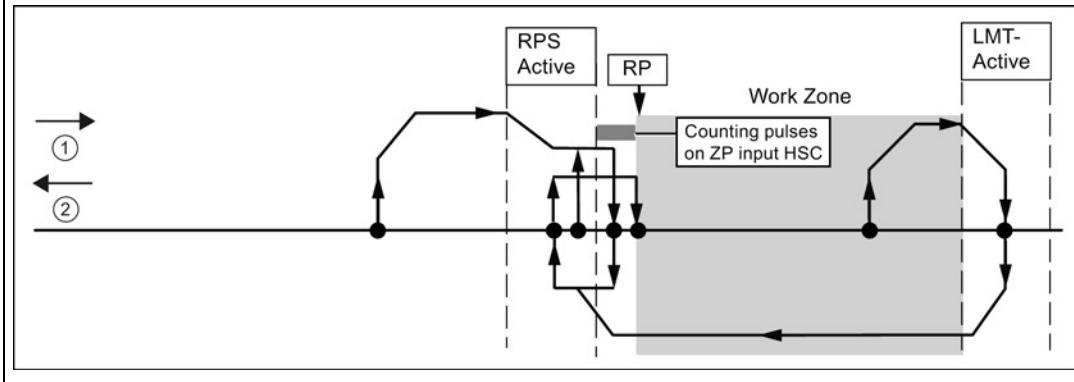
②: Negative motion

RP seek mode 3

Default configuration: RP seek direction: negative and RP approach direction: positive

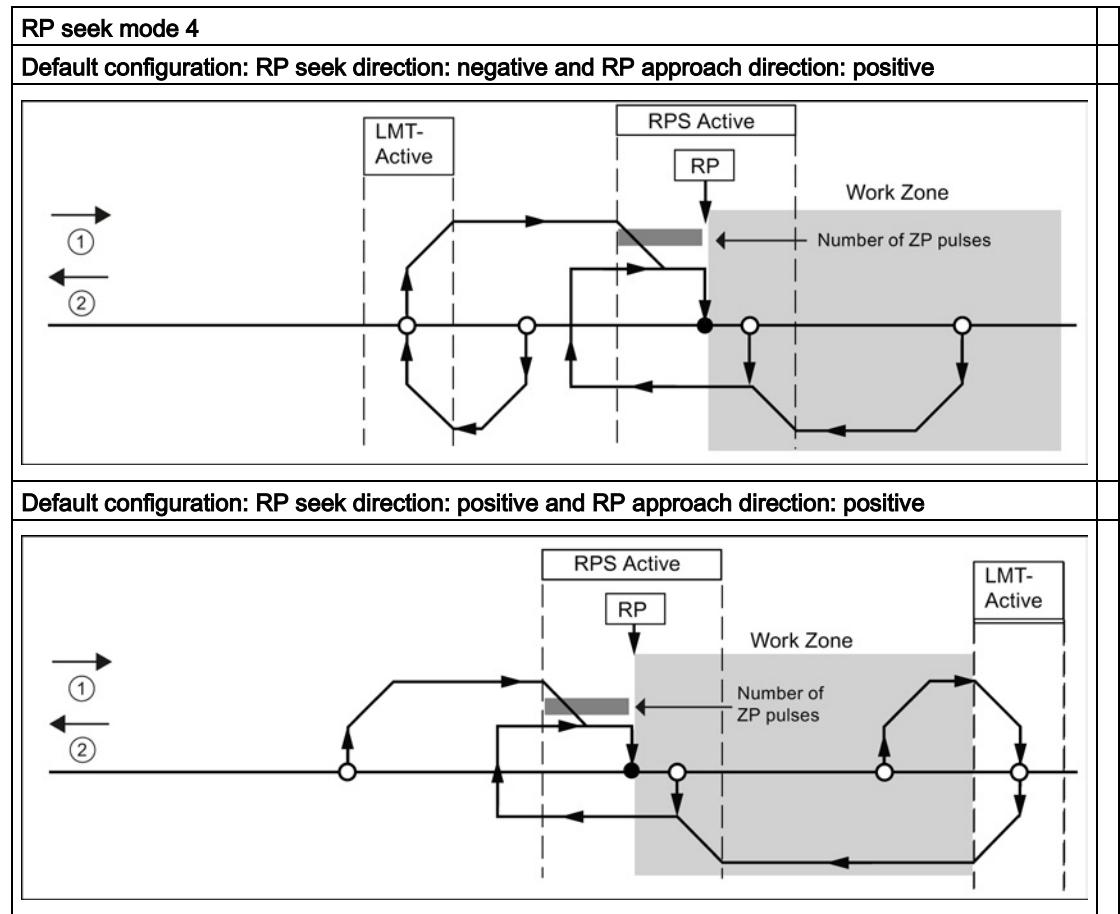


Default configuration: RP seek direction: positive and RP approach direction: positive



①: Positive motion

②: Negative motion

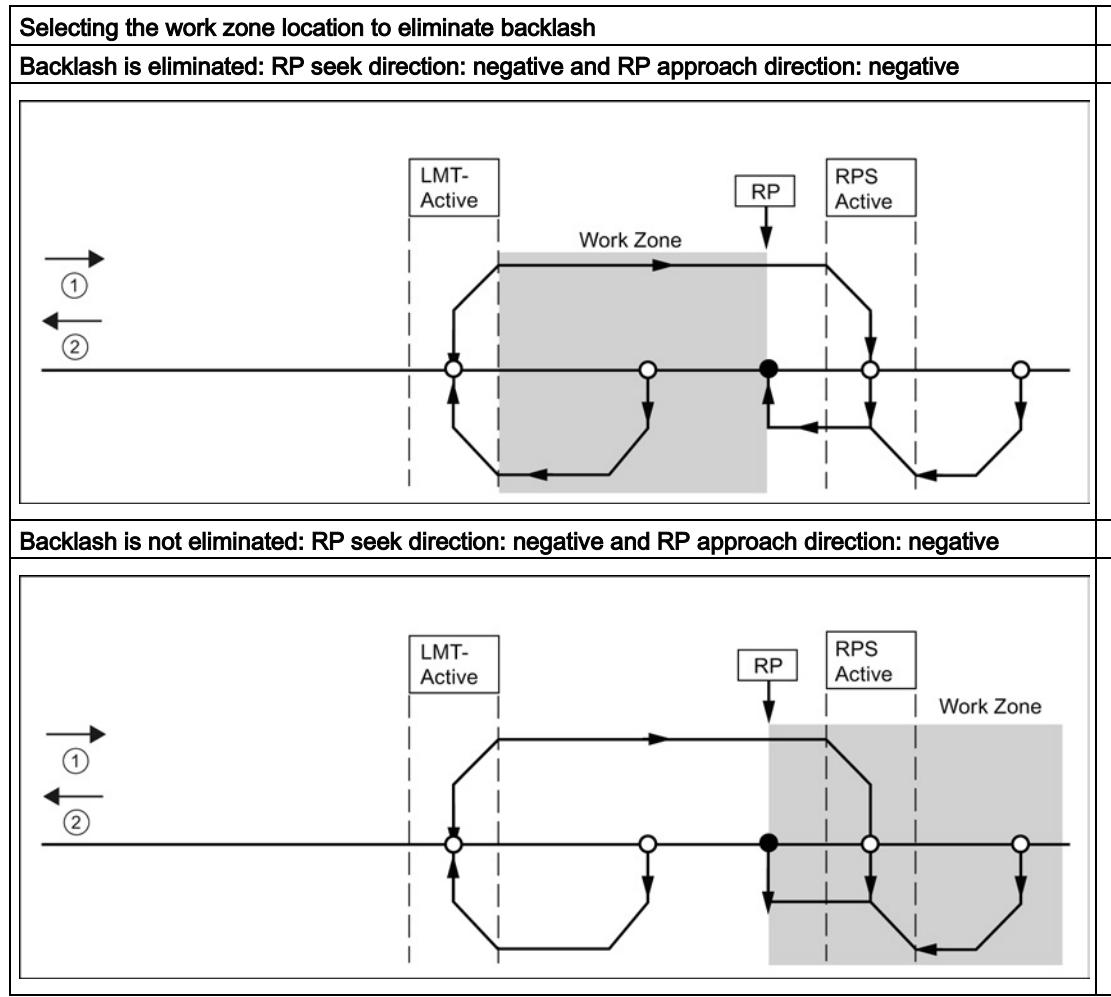


①: Positive motion

②: Negative motion

12.10.1 Selecting the work zone location to eliminate backlash

The following figure shows the work zone in relationship to the reference point (RP), the RPS Active zone, and the limit switches (LMT+ and LMT-) for an approach direction that eliminates the backlash. The second part of the illustration places the work zone so that the backlash is not eliminated. The following figure shows RP seek mode 3. A similar placement of the work zone is possible, although not recommended, for each of the search sequences for each of the other RP seek modes.



①: Positive motion

② Negative motion

Technical specifications

A.1 General specifications

A.1.1 General technical specifications

Standards compliance

The S7-200 SMART automation system complies with the following standards and test specifications. The test criteria for the S7-200 SMART automation system are based on these standards and test specifications.

CE approval



The S7-200 SMART Automation System satisfies requirements and safety related objectives according to the EC directives listed below, and conforms to the harmonized European standards (EN) for the programmable controllers listed in the Official Journals of the European Community.

- EC Directive 2006/95/EC (Low Voltage Directive) "Electrical Equipment Designed for Use within Certain Voltage Limits"
 - EN 61131-2:2007 Programmable controllers - Equipment requirements and tests
- EC Directive 2004/108/EC (EMC Directive) "Electromagnetic Compatibility"
 - Emission standard
EN 61000-6-4:2007: Industrial Environment
 - Immunity standard
EN 61000-6-2:2005: Industrial Environment

The CE Declaration of Conformity is held on file available to competent authorities at:

Siemens AG
IA AS RD ST PLC Amberg
Werner-von-Siemens-Str. 50
D92224 Amberg
Germany

Technical specifications

A.1 General specifications

Industrial environments

The S7-200 SMART automation system is designed for use in industrial environments.

Table A- 1 Industrial environments

Application field	Noise emission requirements	Noise immunity requirements
Industrial	EN 61000-6-4:2007	EN 61000-6-2:2005

Electromagnetic compatibility

Electromagnetic Compatibility (EMC) is the ability of an electrical device to operate as intended in an electromagnetic environment and to operate without emitting levels of electromagnetic interference (EMI) that may disturb other electrical devices in the vicinity.

Table A- 2 Immunity per EN 61000-6-2

Electromagnetic compatibility - Immunity per EN 61000-6-2	
EN 61000-4-2 Electrostatic discharge	8 kV air discharge to all surfaces ±4 kV contact discharge to exposed conductive surfaces
EN 61000-4-3 Radiated, radio-frequency, electromagnetic field immunity test	80 to 1000 MHz, 10 V/m, 80% AM at 1 kHz 1.4 to 2.0 GHz, 3 V/m, 80% AM a 1 kHz 2.0 to 2.7 GHz, 1 V/m, 80% AM at 1 kHz
EN 61000-4-4 Fast transient bursts	2 kV, 5 kHz with coupling network to AC and DC system power 2 kV, 5 kHz with coupling clamp to I/O
EN 61000-4-5 Surge immunity	AC systems - 2 kV common mode, 1kV differential mode DC systems - 2 kV common mode, 1kV differential mode For DC systems (I/O signals, DC power systems) external protection is required.
EN 61000-4-6 Conducted disturbances	150 kHz to 80 MHz, 10 V RMS, 80% AM at 1kHz
EN 61000-4-11 Voltage dips	AC systems 0% for 1 cycle, 40% for 12 cycles and 70% for 30 cycles at 60 Hz

Table A- 3 Conducted and radiated emissions per EN 61000-6-4

Electromagnetic compatibility - Conducted and radiated emissions per EN 61000-6-4		
Conducted Emissions EN 55011, Class A, Group 1	0.15 MHz to 0.5 MHz	<79dB (µV) quasi-peak; <66 dB (µV) average
	0.5 MHz to 5 MHz	<73dB (µV) quasi-peak; <60 dB (µV) average
	5 MHz to 30 MHz	<73dB (µV) quasi-peak; <60 dB (µV) average
Radiated Emissions EN 55011, Class A, Group 1	30 MHz to 230 MHz	<40dB (µV/m) quasi-peak; measured at 10m
	230 MHz to 1 GHz	<47dB (µV/m) quasi-peak; measured at 10m

Environmental conditions

Table A- 4 Transport and storage

Environmental conditions - Transport and storage	
EN 60068-2-2, Test Bb, Dry heat and EN 60068-2-1, Test Ab, Cold	-40° C to +70° C
EN 60068-2-30, Test Db, Damp heat	25° C to 55° C, 95% humidity
EN 60068-2-14, Test Na, temperature shock	-40° C to +70° C, dwell time 3 hours, 2 cycles
EN 60068-2-32, Free fall	0.3 m, 5 times, product packaging
Atmospheric pressure	1080 to 660h Pa (corresponding to an altitude of -1000 to 3500m)

Table A- 5 Operating conditions

Environmental conditions - Operating	
Ambient temperature range (Inlet Air 25 mm below unit)	0° C to 55° C horizontal mounting 0° C to 45° C vertical mounting 95% non-condensing humidity
Atmospheric pressure	1080 to 795 hPa (corresponding to an altitude of -1000 to 2000m)
Concentration of contaminants	S0 ₂ : < 0.5 ppm; H ₂ S: < 0.1 ppm; RH < 60% non-condensing
EN 60068-2-14, Test Nb, temperature change	5° C to 55° C, 3° C/minute
EN 60068-2-27 Mechanical shock	15 G, 11 ms pulse, 6 shocks in each of 3 axis
EN 60068-2-6 Sinusoidal vibration	DIN rail mount: 3.5 mm from 5-9 Hz, 1G from 9 - 150 Hz Panel Mount: 7.0 mm from 5-9 Hz, 2G from 9 to 150 Hz 10 sweeps each axis, 1 octave per minute

Table A- 6 High potential isolation test

High potential isolation test	
24 V/5 V nominal circuits	520 VDC (type test of optical isolation boundaries)
115/230 V circuits to ground	1,500 VAC routine test/1950 VDC type test
115/230 V circuits to 115/230 V circuits	1,500 VAC routine test/1950 VDC type test
115 V/230V circuits to 24 V/5 V circuits	1,500 VAC routine test/3250 VDC type test
Ethernet port to 24 V/5 V circuits and ground ¹	1,500 VAC (type test only)

¹ Ethernet port isolation is designed to limit hazard during short term network faults to hazardous voltages. It does not conform to safety requirements for routine AC line voltage isolation.

Technical specifications

A.1 General specifications

Protection class

- Protection Class II according to EN 61131-2 (Protective conductor not required)

Degree of protection

- IP20 Mechanical Protection, EN 60529
- Protects against finger contact with high voltage as tested by standard probe. External protection required for dust, dirt, water and foreign objects of < 12.5mm in diameter.

Rated voltages

Table A- 7 Rated voltages

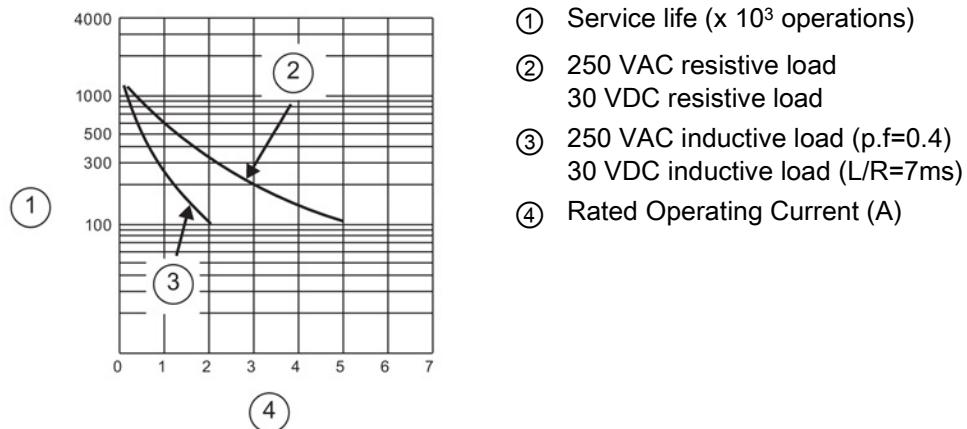
Rated voltage	Tolerance
24 VDC	20.4 VDC to 28.8 VDC
120/240 VAC	85 VAC to 264 VAC, 47 to 63 Hz

Note

When a mechanical contact turns on output power to the S7-200 SMART CPU, or any digital signal module, it sends a "1" signal to the digital outputs for approximately 150 microseconds. This could cause unexpected machine or process operation which could result in death or serious injury to personnel and/or damage to equipment. You must plan for this, especially if you are using devices which respond to short duration pulses.

Relay electrical service life

The typical performance data supplied by relay vendors is shown below. Actual performance may vary depending upon your specific application. An external protection circuit that is adapted to the load will enhance the service life of the contacts.



A.2 S7-200 SMART CPUs

A.2.1 CPU ST20 and CPU SR20

A.2.1.1 General specifications and features

General specifications and features of the CPU ST20 and CPU SR20

Table A- 8 General specifications

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay
Order number	6ES7 288-1ST20-0AA0	6ES7 288-1SR20-0AA0
Dimensions W x H x D (mm)	90 x 100 x 81	90 x 100 x 81
Weight	320 grams	367.3 grams
Power dissipation	20 W	14 W
Current available (EM bus)	1110 mA max.(5 VDC)	740 mA max. (5 VDC)
Current available (24 VDC)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 VDC)	4 mA/input used	4 mA/input used

Technical specifications

A.2 S7-200 SMART CPUs

Table A- 9 CPU features

Technical data		Description
CPU ST20 DC/DC/DC, CPU SR20 AC/DC/Relay		
User memory ¹	Program	12 Kbytes
	User data (V)	8 Kbytes
	Retentive	10 Kbytes max. ¹
On-board digital I/O		12 inputs/8 outputs
Process image		256 bits of inputs (I) / 256 bits of outputs (Q)
Analog image		56 words of inputs (AI) / 56 words of outputs (AQ)
Bit memory (M)		256 bits
Temporary (local) memory (L)		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine
Sequential control relays (S)		256 bits
Signal modules expansion		6
Signal board expansion		1 max.
High-speed counters		4 total <ul style="list-style-type: none"> • 4 at 200K Hz for single phase • 2 at 100 K Hz for A/B phase
Pulse outputs ²		2 at 100 KHz ²
Pulse catch inputs		12
Cyclic interrupts		2 at 1 ms resolution
Edge interrupts		4 rising and 4 falling (6 and 6 with optional signal board)
Memory card		microSDHC Card (optional)
Real time clock accuracy		+/- 120 seconds/month
Real time clock retention time		7 days typ./6 days min. at 25°C (maintenance-free Super Capacitor)

¹ You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive times) to be retentive, up to the specified maximum amount.

² The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 10 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 11 User program elements supported

Element	Description	
POUs	Type/ quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
Timers	Type/ quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
	Counters	Quantity 256

Table A- 12 Communication

Technical data	Description
Number of ports	Ethernet: 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)
HMI device	4 per port : RS485, SB CM01(RS232/485 signal board) 8 per port: Ethernet
Programming device (PG)	Ethernet: 1
Connections	Ethernet: 1 for programming device, 8 for HMIs, 8 for CPUs, 8 active GET/PUT, 8 passive GET/PUT RS485: 4 for HMIs per port
CPUs (peer-to-peer)	Ethernet: 8
Data rates	Ethernet: 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	Ethernet: Transformer isolated, 1500 VAC RS485: none
Cable type	Ethernet: CAT5e shielded RS485: PROFIBUS network cable

Technical specifications

A.2 S7-200 SMART CPUs

Table A- 13 Power supply

Technical data		CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay
Voltage range		20.4 to 28.8 VDC	85 to 264 VAC
Line frequency		--	47 to 63 Hz
Input current	CPU only at max. load	160 mA at 24 VDC (without driving 300 mA sensor power) 430 mA at 24 VDC (with driving 300 mA sensor power)	210 mA at 120 VAC (with 300 mA power sensor output) 90 mA at 120 VAC (without 300 mA power sensor output) 120 mA at 240 VAC (with 300 mA power sensor output) 60 mA at 240 VAC (without 300 mA power sensor output)
	CPU with all expansion accessories at max. load	720 mA at 24 VDC	290 mA at 120 VAC 170 mA at 240 VAC
Inrush current (max.)		11.7 A at 28.8 VDC	9.3 A at 264 VAC
Isolation (input power to logic)	--		1500 VAC
Ground leakage, AC line to functional earth	--		0.5 mA max.
Hold up time (loss of power)		20 ms at 24 VDC	30 ms at 120 VAC 200 ms at 240 VAC
Internal fuse, not user replaceable		3 A, 250 V slow blow	3 A, 250 V, slow blow

Table A- 14 Sensor power

Technical data		CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay
Voltage range		20.4 to 28.8 VDC	20.4 to 28.8 VDC
Output current rating (max.)		300 mA (short circuit protected)	300 mA (short circuit protected)
Maximum ripple noise (<10 MHz)		< 1 V peak to peak	< 1 V peak to peak
Isolation (CPU logic to sensor power)		Not isolated	Not isolated

A.2.1.2 Digital inputs and outputs

Table A- 15 Digital inputs

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay
Number of inputs	12	12
Type	SinkSource (IEC Type 1 sink, except I0.0 to I0.3)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec	35 VDC for 0.5 sec.
Logic 1 signal (min.)	I0.0 to I0.3, I0.6 to I0.7: 4 VDC at 8 mA Other inputs: 15 VDC at 2.5 mA	15 VDC at 2.5 mA
Logic 0 signal (max.)	I0.0 to I0.3, I0.6 to I0.7: 1 VDC at 1 mA Other inputs: 5 VDC at 1 mA	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1	1
Filter times	Individually selectable on each channel (points I0.0 to I1.3): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.3): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	4 HSC at 200 KHz for single phase 2 HSC at 100 KHz for A/B phase	4 HSC at 200 KHz for single phase 2 HSC at 100 KHz for A/B phase
Number of inputs on simultaneously	12	12
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): <ul style="list-style-type: none"> • 500 m normal (low-speed) inputs • 50 m HSC (high-speed) inputs <hr/> I0.6 to I0.7: <ul style="list-style-type: none"> • Shielded (only): 500 m normal inputs <hr/> All other inputs: <ul style="list-style-type: none"> • Shielded: 500 m normal inputs • Unshielded: 300 m normal inputs 	All inputs: <ul style="list-style-type: none"> • Shielded: 500 m normal inputs, 50 m HSC inputs • Unshielded: 300 m normal inputs

Technical specifications

A.2 S7-200 SMART CPUs

Table A- 16 Digital outputs

Technical data	CPU ST20 DC/DC/DC	CPU SR20 AC/DC/Relay
Number of outputs	8	8
Type	Solid state - MOSFET (sourcing)	Relay, dry contact
Voltage range	20.4 to 28.8 VDC	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	20 VDC min.	--
Logic 0 signal with 10 KΩ load	0.1 VDC max.	--
Rated current per point (max.)	0.5 A	2.0 A
Rated current per common (max.)	6 A	10.0 A
Lamp load	5 W	30 W DC/200 W AC
ON state resistance	0.6 Ω max.	0.2 Ω max. when new
Leakage current per point	10 μA max.	--
Surge current	8 A for 100 ms max.	7 A with contacts closed
Overload protection	No	No
Isolation (field side to logic)	500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new
Isolation between open contacts	--	750 VAC for 1 minute
Isolation groups	2	1
Inductive clamp voltage	L+ minus 48 VDC, 1 W dissipation	Not recommended
Switching delay (Qa.0 to Qa.3)	1.0 μs max., off to on 3.0 μs max., on to off	10 ms max.
Switching delay (Qa.4 to Qa.7)	50 μs max., off to on 200 μs max., on to off	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles
Output state in STOP mode	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8
Cable length (max.), in meters	Shielded: 500 m Unshielded: 300 m	Shielded: 500 m Unshielded: 300 m

A.2.1.3 CPU ST20 and CPU SR20 wiring diagrams

Table A- 17 Wiring diagram for the CPU ST20 DC/DC/DC (6ES7 288-1ST20-0AA0)

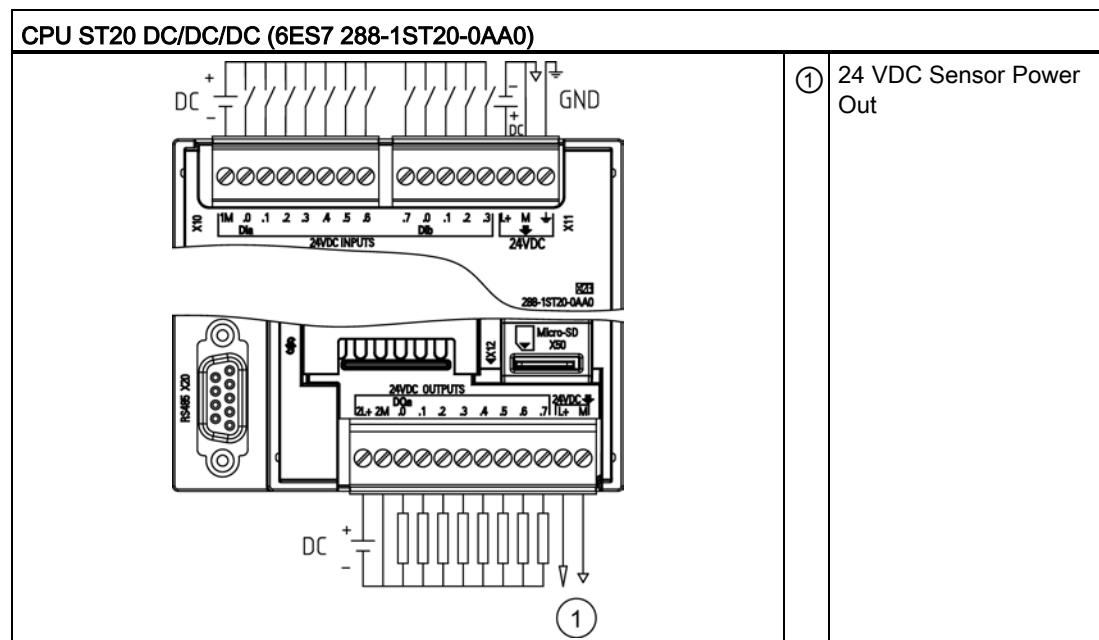


Table A- 18 Connector pin locations for CPU ST20 DC/DC/DC (6ES7 288-1ST20-0AA0)

Pin	X10	X11	X12
1	1M	DI a.7	2L+
2	DI a.0	DI b.0	2M
3	DI a.1	DI b.1	DQ a.0
4	DI a.2	DI b.2	DQ a.1
5	DI a.3	DI b.3	DQ a.2
6	DI a.4	L+ / 24 VDC	DQ a.3
7	DI a.5	M / 24 VDC	DQ a.4
8	DI a.6	Functional Earth	DQ a.5
9	--	--	DQ a.6
10	--	--	DQ a.7
11	--	--	L+ / 24 VDC
12	--	--	M / 24 VDC

Technical specifications

A.2 S7-200 SMART CPUs

Table A- 19 Wiring diagram for the CPU SR20 AC/DC/Relay (6ES7 288-1SR20-0AA0)

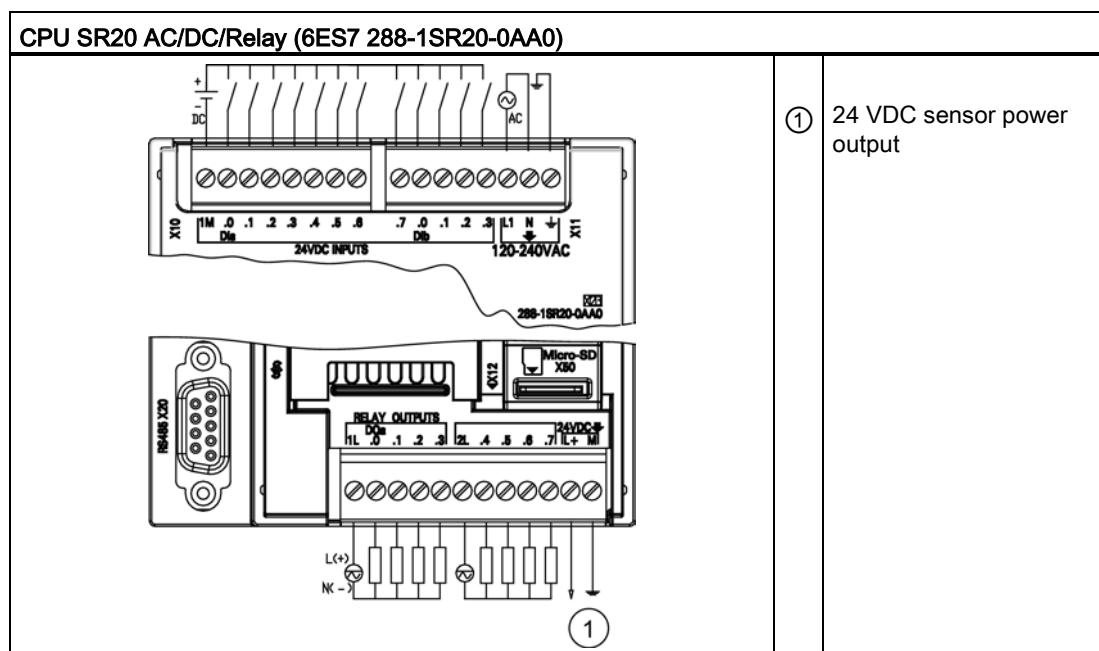


Table A- 20 Connector pin locations for CPU SR20 AC/DC/Relay (6ES7 288-1SR20-0AA0)

Pin	X10	X11	X12
1	1M	DI a.7	1L
2	DI a.0	DI b.0	DQ a.0
3	DI a.1	DI b.1	DQ a.1
4	DI a.2	DI b.2	DQ a.2
5	DI a.3	DI b.3	DQ a.3
6	DI a.4	L1 / 120 - 240 VAC	2L
7	DI a.5	N / 120 - 240 VAC	DQ a.4
8	DI a.6	Functional Earth	DQ a.5
9	--	--	DQ a.6
10	--	--	DQ a.7
11	--	--	L+ / 24 VDC Out
12	--	--	M / 24 VDC Out

A.2.2 CPU ST30 and CPU SR30

A.2.2.1 General specifications and features

General specifications and features of the CPU ST30 and CPU SR30

Table A- 21 General specifications

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Order number	6ES7 288-1ST30-0AA0	6ES7 288-1SR30-0AA0
Dimensions W x H x D (mm)	110 x 100 x 81	110 x 100 x 81
Weight	375 g	435 g
Power dissipation	12 W	14 W
Current available (EM bus)	740 mA max. (5 VDC)	740 mA max. (5 VDC)
Current available (24 VDC)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 VDC)	4 mA/ input used	4 mA/ input used

Table A- 22 CPU features

Technical data		Description	
		CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
User memory ¹	Program	18 Kbytes	18 Kbytes
	User data (V)	12 Kbytes	12 Kbytes
	Retentive	10 Kbytes max. ¹	10 Kbytes max. ¹
On-board digital I/O		18 inputs/12 outputs	18 inputs/12 outputs
Process image		256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)
Analog image		56 words of inputs (AI) / 56 words of outputs (AQ)	56 words of inputs (AI) / 56 words of outputs (AQ)
Bit memory (M)		256 bits	256 bits
Temporary (local) memory (L)		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine
Sequential control relays (S)		256 bits	256 bits
Signal modules expansion		6	6
Signal board expansion		1 max.	1 max.
High-speed counters		4 total <ul style="list-style-type: none"> • 4 at 200 KHz for single phase • 2 at 100 KHz for A/B phase 	4 total <ul style="list-style-type: none"> • 4 at 200 KHz for single phase • 2 at 100 KHz for A/B phase
Pulse outputs ²		3 at 100 KHz	--
Pulse catch inputs		12	12

Technical specifications

A.2 S7-200 SMART CPUs

Technical data	Description	
	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Cyclic interrupts	2 at 1 ms resolution	2 at 1 ms resolution
Edge interrupts	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling (6 and 6 with optional signal board)
Memory card	microSDHC Card (optional)	microSDHC Card (optional)
Real time clock accuracy	+/- 120 seconds/month	+/- 120 seconds/month
Real time clock retention time	7 days typ./6 days min. at 25°C (maintenance-free Super Capacitor)	7 days typ./6 days min. at 25°C (maintenance-free Super Capacitor)

- ¹ You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values on retentive times) to be retentive, up to the specified maximum amount.
- ² The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 23 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 24 User program elements supported

Element	Description	
POUs	Type/quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
	Type/quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Table A- 25 Communication

Technical data	Description
Number of ports	Ethernet: 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)
HMI device	4 per port : RS485, SB CM01(RS232/485 signal board) 8 per port: Ethernet
Programming device (PG)	Ethernet: 1

Technical data	Description
Connections	Ethernet: 1 for programming device, 8 for HMIs, 8 for CPUs, 8 active GET/PUT, 8 passive GET/PUT RS485: 4 for HMIs per port
CPUs (peer-to-peer)	Ethernet: 8
Data rates	Ethernet: 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	Ethernet: Transformer isolated, 1500 VAC RS485: none
Cable type	Ethernet: CAT5e shielded RS485: PROFIBUS network cable

Table A- 26 Power supply

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Voltage range	20.4 to 28.8 VDC	85 to 264 VAC
Line frequency	--	47 to 63 Hz
Input current	CPU only at max. load CPU with all expansion accessories at max. load	64 mA at 24 VDC (without driving 300 mA sensor power) 365 mA at 24 VDC (with driving 300 mA sensor power) 92 mA at 120 VAC (with power sensor) 40 mA at 120 VAC (without power sensor) 52 mA at 240 VAC (with power sensor) 27 mA at 240 VAC (without power sensor) 136 mA at 120 VAC 72 mA at 240 VAC
Inrush current (max.)	6A at 28.8 VDC	8.9 A at 264 VAC
Isolation (input power to logic)	--	1500 VAC
Ground leakage, AC line to functional earth	--	0.5 mA max.
Hold up time (loss of power)	20 ms at 24 VDC	30 ms at 120 VAC 200 ms at 240 VAC
Internal fuse, not user replaceable	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 27 Sensor power

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Voltage range	20.4 to 28.8 VDC	20.4 to 28.8 VDC
Output current rating (max.)	300 mA (short circuit protected)	300 mA (short circuit protected)
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak
Isolation (CPU logic to sensor power)	Not isolated	Not isolated

A.2.2.2 Digital inputs and outputs

Table A- 28 Digital inputs

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Number of inputs	18	18
Type	Sink/Source (IEC Type 1 sink, except I0.0 to I0.3)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.
Logic 1 signal (min.)	I0.0 to I0.3, I0.6 to I0.7: 4 VDC at 8 mA Other inputs: 15 VDC at 2.5 mA	15 VDC at 2.5 mA
Logic 0 signal (max.)	I0.0 to I0.3, I0.6 to I0.7: 1 VDC at 1 mA Other inputs: 5 VDC at 1 mA	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1	1
Filter times	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	4 HSC at 200 KHz for single phase 2 HSC at 100 KHz for A/B phase	4 HSC at 200 KHz for single phase 2 HSC at 100 KHz for A/B phase
Number of inputs on simultaneously	18	18
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): • 500 m normal (low-speed) inputs • 50 m HSC (high-speed) inputs I0.6 to I0.7: • Shielded (only): 500 m normal inputs All other inputs: • Shielded: 500 m normal inputs • Unshielded: 300 m normal inputs	All inputs: • Shielded: 500 m normal inputs, 50 m HSC inputs • Unshielded: 300 m normal inputs

Table A- 29 Digital outputs

Technical data	CPU ST30 DC/DC/DC	CPU SR30 AC/DC/Relay
Number of outputs	12	12
Type	Solid state - MOSFET (sourcing)	Relay, dry contact
Voltage range	20.4 to 28.8 VDC	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	20 VDC min.	--
Logic 0 signal with 10 KΩ load	0.1 VDC max.	--
Rated current per point (max.)	0.5 A	2.0 A
Rated current per common (max.)	6 A	10.0 A
Lamp load	5 W	30 W DC/200 W AC
ON state resistance	0.6 Ω max.	0.2 Ω max. when new
Leakage current per point	10 μA max.	--
Surge current	8 A for 100 ms max.	7 A with contacts closed
Overload protection	No	No
Isolation (field side to logic)	500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new
Isolation between open contacts	--	750 VAC for 1 minute
Isolation groups	1	1
Inductive clamp voltage	L+ minus 48 VDC, 1 W dissipation	Not recommended
Switching delay (Qa.0 to Qa.3)	1.0 μs max., off to on 3.0 μs max., on to off	10 ms max.
Switching delay (Qa.4 to Qb.7)	50 μs max., off to on, 200 μs max., on to off	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles
Output state in STOP mode	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	12	12
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

A.2.2.3 CPU ST30 and CPU SR30 wiring diagrams

Table A- 30 Wiring diagram for the CPU ST30 DC/DC/DC (6ES7 288-1ST30-0AA0)

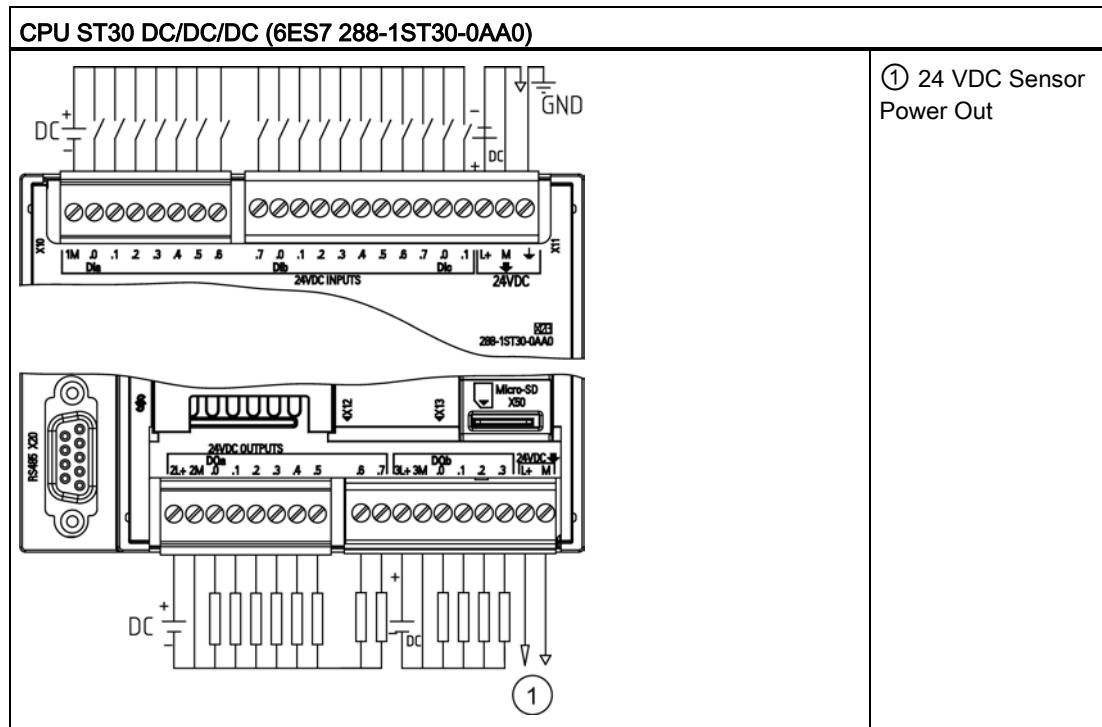


Table A- 31 Connector pin locations for CPU ST30 DC/DC/DC (6ES7 288-1ST30-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	2L+	DQ a.6
2	DI a.0	DI b.0	2M	DQ a.7
3	DI a.1	DI b.1	DQ a.0	3L+
4	DI a.2	DI b.2	DQ a.1	3M
5	DI a.3	DI b.3	DQ a.2	DQb.0
6	DI a.4	Dlb.4	DQa.3	DQb.1
7	DI a.5	Dlb.5	DQ a.4	DQb.2
8	DI a.6	Dlb.6	DQ a.5	DQb.3
9	--	Dlb.7	--	L+ / 24 VDC
10	--	Dlc.0	--	M / 24 VDC
11	--	Dlc.1	--	--
12	--	L+24 VDC	--	--
13	--	M / 24 VDC	--	--
14	--	Functional Earth	--	--

Table A- 32 Wiring diagram for the CPU SR30 AC/DC/Relay (6ES7 288-1SR30-0AA0)

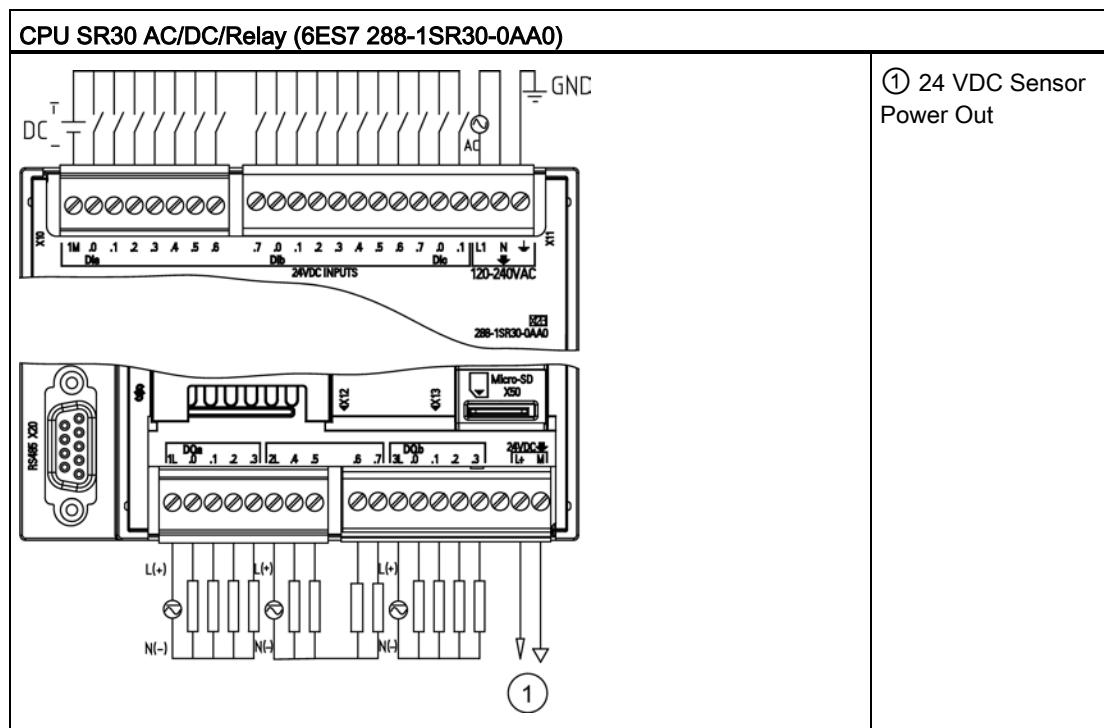


Table A- 33 Connector pin locations for CPU SR30 AC/DC/Relay (6ES7 288-1SR30-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ a.6
2	DI a.0	DI b.0	DQ a.0	DQ a.7
3	DI a.1	DI b.1	DQ a.1	3L
4	DI a.2	DI b.2	DQ a.2	DQ b.0
5	DI a.3	DI b.3	DQ a.3	DQ b.1
6	DI a.4	DIb.4	2L	DQ b.2
7	DI a.5	DIb.5	DQ a.4	DQ b.3
8	DI a.6	DIb.6	DQ a.5	--
9	--	DIb.7	--	L+ / 24 VDC Out
10	--	DIC.0	--	M / 24 VDC Out
11	--	DIC.1	--	--
12	--	L1 / 120 - 240 VAC	--	--
13	--	N / 120 - 240 VAC	--	--
14	--	Functional Earth	--	--

Technical specifications

A.2 S7-200 SMART CPUs

A.2.3 CPU ST40, CPU SR40, and CPU CR40

A.2.3.1 General specifications and features

General specifications and features of the CPU ST40, CPU SR40, and CPU CR40

Table A- 34 General specifications

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
Order number	6ES7 288-1ST40-0AA0	6ES7 288-1SR40-0AA0	6ES7 288-1CR40-0AA0
Dimensions W x H x D (mm)	125 x 100 x 81	125 x 100 x 81	125 x 100 x 81
Weight	410.3 grams	441.3 grams	440 grams
Power dissipation	18 W	23 W	18 W
Current available (EM bus)	740 mA max. (5 VDC)	740 mA max. (5 VDC)	--
Current available (24 VDC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 VDC)	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 35 CPU features

Technical data		Description	
		CPU ST40 DC/DC/DC, CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
User memory	Program	24 Kbytes	12 Kbytes
	User data (V)	16 Kbytes	8 Kbytes
	Retentive	10 Kbytes max. ¹	10 Kbytes max. ¹
On-board digital I/O	24 inputs/16 outputs		24 inputs/16 outputs
Process image	256 bits of inputs (I) / 256 bits of outputs (Q)		256 bits of inputs (I) / 256 bits of outputs (Q)
Analog image	56 words of inputs (AI) / 56 words of outputs (AQ)		--
Bit memory (M)	256 bits		256 bits
Temporary (local) memory (L)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine		64 bytes in the main program and 64 bytes in each subroutine and interrupt routine
Sequential control relays (S)	256 bits		256 bits
Signal modules expansion	6 max.		--
Signal board expansion	1 max.		--

Technical data	Description	
	CPU ST40 DC/DC/DC, CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
High-speed counters	4 total <ul style="list-style-type: none">• 4 at 200 KHz single phase• 2 at 100 KHz A/B phase	4 total <ul style="list-style-type: none">• 4 at 100 KHz single phase• 2 at 50 KHz A/B phase
Pulse outputs ²	3 at 100 KHz	--
Pulse catch inputs	14	14
Cyclic interrupts	2 at 1 ms resolution	2 at 1 ms resolution
Edge interrupts	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling
Memory card	microSDHC Card (optional)	microSDHC Card (optional)
Real time clock accuracy	120 seconds/month	--
Real time clock retention time	7 days typ./6 days min. at 25°C	--

¹ You can configure areas of V memory, M memory, C memory (current values), and portions of T memory (current values) on retentive, up to the specified maximum amount.

² The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 36 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 37 User program elements supported

Element	Description	
POUs	Type/ quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
Timers	Type/ quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Technical specifications

A.2 S7-200 SMART CPUs

Table A- 38 Communication

Technical data	Description
Number of ports	Ethernet: 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board) on SR40/ST40 only
HMI device ¹	4 per port
Programming device (PG)	Ethernet: 1
Connections	Ethernet: 1 for programming device, 4 for HMIs, 8 for CPUs, 8 active GET/PUT, 8 passive GET/PUT RS485: 4 for HMIs per port
CPUs (peer-to-peer)	Ethernet: 8
Data rates	Ethernet: 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	Ethernet: Transformer isolated, 1500 VDC RS485: none
Cable type	Ethernet: CAT5e shielded RS485: PROFIBUS network cable

Table A- 39 Power supply

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
Voltage range	20.4 to 28.8 VDC	85 to 264 VAC	85 to 264 VAC
Line frequency	--	47 to 63 Hz	47 to 63 Hz
Input current (max. CPU only load)	190 mA at 24 VDC (without driving 300 mA sensor power) 470 mA at 24 VDC (with driving 300 mA sensor power)	130 mA at 120 VAC (without driving 300 mA sensor power) 250 mA at 120V (with driving 300 mA sensor power) 80 mA at 240 VAC (without driving 300 mA sensor power) 150 mA at 240 VAC (with driving 300 mA sensor power)	130 mA at 120 VAC (without driving 300 mA sensor power) 250 mA at 120V (with driving 300 mA sensor power) 80 mA at 240 VAC (without driving 300 mA sensor power) 150 mA at 240 VAC (with driving 300 mA sensor power)
CPU with all expansion accessories	680 mA at 24 VDC	300 mA at 120 VAC 190 mA at 240 VAC	--
Inrush current (max.)	11.7 A at 28.8 VDC	16.3 A at 264 VAC	7.3 A at 264 VAC
Isolation (input power to logic)	--	1500 VAC	1500 VAC
Ground leakage, AC line to functional earth	--	0.5 mA	0.5 mA

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
Hold up time (loss of power)	20 ms at 24 VDC	30 ms at 120 VAC 200 ms at 240 VAC	50 ms at 120 VAC 400 ms at 240 VAC
Internal fuse, not user replaceable	3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 40 Sensor power

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
Voltage range	20.4 to 28.8 VDC	20.4 to 28.8 VDC	20.4 to 28.8 VDC
Output current rating (max.)	300 mA	300 mA	300 mA
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak	< 1 V peak to peak
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated

A.2.3.2 Digital inputs and outputs

Table A- 41 Digital inputs

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
Number of inputs	24	24	24
Type	Sink/Source (IEC Type 1 sink, except I0.0 to I0.3)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.	30 VDC, max.	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.
Logic 1 signal (min.)	I0.0 to I0.3: 4 VDC at 8 mA Other inputs: 15 VDC at 2.5 mA	I0.0 to I0.3: 4 VDC at 8 mA 15 VDC at 2.5 mA	15 VDC at 2.5 mA
Logic 0 signal (max.)	I0.0 to I0.3: 1 VDC at 1 mA Other inputs: 5 VDC at 1 mA	5 VDC at 1 mA	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute	500 VAC for 1 minute
Isolation group	1	1	1
Filter times	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8

Technical specifications

A.2 S7-200 SMART CPUs

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
	Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	4 HSC at 200 kHz for single phase 2 HSC at 100 kHz for A/B phase	4 HSC at 200 kHz for single phase 2 HSC at 100 kHz for A/B phase	4 HSC at 100 kHz for single phase 2 HSC at 50 kHz for A/B phase
Number of inputs on simultaneously	24	24	24
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): <ul style="list-style-type: none"> • 500 m normal (low-speed) inputs, • 50 m HSC (high-speed) inputs All other inputs: Shielded: <ul style="list-style-type: none"> • 500 m normal inputs Unshielded: <ul style="list-style-type: none"> • 300 m normal inputs 	All inputs: Shielded: <ul style="list-style-type: none"> • 500 m normal inputs • 50 m HSC inputs Unshielded: <ul style="list-style-type: none"> • 300 m normal inputs 	All inputs: Shielded: <ul style="list-style-type: none"> • 500 m normal inputs • 50 m HSC inputs Unshielded: <ul style="list-style-type: none"> • 300 m normal inputs

Table A- 42 Digital outputs

Technical data	CPU ST40 DC/DC/DC	CPU SR40 AC/DC/Relay	CPU CR40 AC/DC/Relay
Number of outputs	16	16	16
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact
Voltage range	20.4 to 28.8 VDC	5 to 30 VDC or 5 to 250 VAC	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	20 VDC min.	--	--
Logic 0 signal with 10 kΩ load	0.1 VDC max.	--	--
Rated current per point (max.)	0.5 A	2 A	2 A
Rated current per common (max.)	6 A	10 A	10 A
Lamp load	5 W	30 W DC / 200 W AC	30 W DC / 200 W AC
ON state resistance	0.6 Ω max.	0.2 Ω max. when new	0.2 Ω max. when new
Leakage current per point	10 μA max.	--	--
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed
Overload protection	No	No	No

Technical data	CPU ST40	CPU SR40	CPU CR40
	DC/DC/DC	AC/DC/Relay	AC/DC/Relay
Isolation (field side to logic)	500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 MΩ min. when new	100 MΩ min. when new
Isolation between open contact	--	750 VAC for 1 minute	750 VAC for 1 minute
Isolation groups	2	4	4
Inductive clamp voltage	L+ minus 48 VDC, 1 W dissipation	--	--
Switching delay (Qa.0 to Qa.3)	1.0 µs max., off to on 3.0 µs max., on to off	10 ms max.	10 ms max.
Switching delay (Qa.4 to Qb.7)	50 µs max., off to on 200 µs max., on to off	10 ms max.	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles
Output state in STOP mode	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	16	16	16
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

A.2.3.3 CPU ST40, SR40 and CR40 wiring diagrams

Table A- 43 Wiring diagram for the CPU ST40 DC/DC/DC (6ES7 288-1ST40-0AA0)

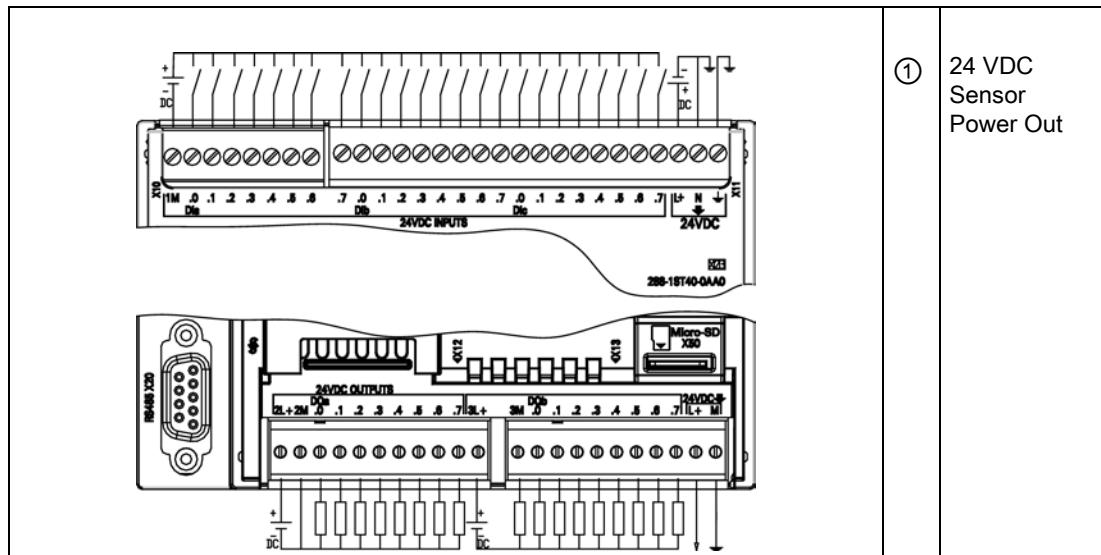


Table A- 44 Connector pin locations for CPU ST40 DC/DC/DC (6ES7 288-1ST40-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	2L+	3M
2	DI a.0	DI b.0	2M	DQ b.0
3	DI a.1	DI b.1	DQ a.0	DQ b.1
4	DI a.2	DI b.2	DQ a.1	DQ b.2
5	DI a.3	DI b.3	DQ a.2	DQ b.3
6	DI a.4	DI b.4	DQ a.3	DQ b.4
7	DI a.5	DI b.5	DQ a.4	DQ b.5
8	DI a.6	DI b.6	DQ a.5	DQ b.6
9	--	DI b.7	DQ a.6	DQ b.7
10	--	DI c.0	DQ a.7	L+ / 24 VDC Out
11	--	DI c.1	3L+	M / 24 VDC Out
12	--	DI c.2	--	--
13	--	DI c.3	--	--
14	--	DI c.4	--	--
15	--	DI c.5	--	--
16	--	DI c.6	--	--
17	--	DI c.7	--	--
18	--	L+ / 24 VDC	--	--
19	--	M / 24 VDC	--	--
20	--	Functional Earth	--	--

Table A- 45 Wiring diagram for the CPU SR40 AC/DC/Relay (6ES7 288-1SR40-0AA0)

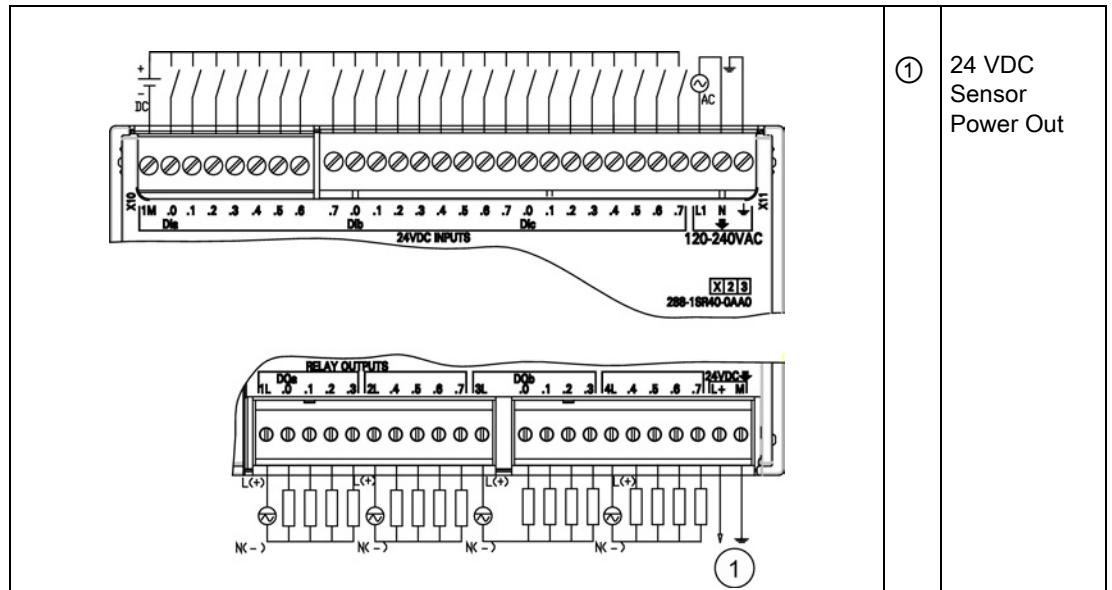


Table A- 46 Connector pin locations for CPU SR40 AC/DC/Relay (6ES7 288-1SR40-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ b.0
2	DI a.0	DI b.0	DQ a.0	DQ b.1
3	DI a.1	DI b.1	DQ a.1	DQ b.2
4	DI a.2	DI b.2	DQ a.2	DQ b.3
5	DI a.3	DI b.3	DQ a.3	4L
6	DI a.4	DI b.4	2L	DQ b.4
7	DI a.5	DI b.5	DQ a.4	DQ b.5
8	DI a.6	DI b.6	DQ a.5	DQ b.6
9	--	DI b.7	DQ a.6	DQ b.7
10	--	DI c.0	DQ a.7	L+ / 24 VDC Out
11	--	DI c.1	3L	M / 24 VDC Out
12	--	DI c.2	--	--
13	--	DI c.3	--	--
14	--	DI c.4	--	--
15	--	DI c.5	--	--
16	--	DI c.6	--	--
17	--	DI c.7	--	--
18	--	L1 / 120 - 240 VAC	--	--
19	--	N / 120 - 240 VAC	--	--
20	--	Functional Earth	--	--

Technical specifications

A.2 S7-200 SMART CPUs

Table A- 47 Wiring diagram for the CPU CR40 AC/DC/Relay (6ES7 288-1CR40-0AA0)

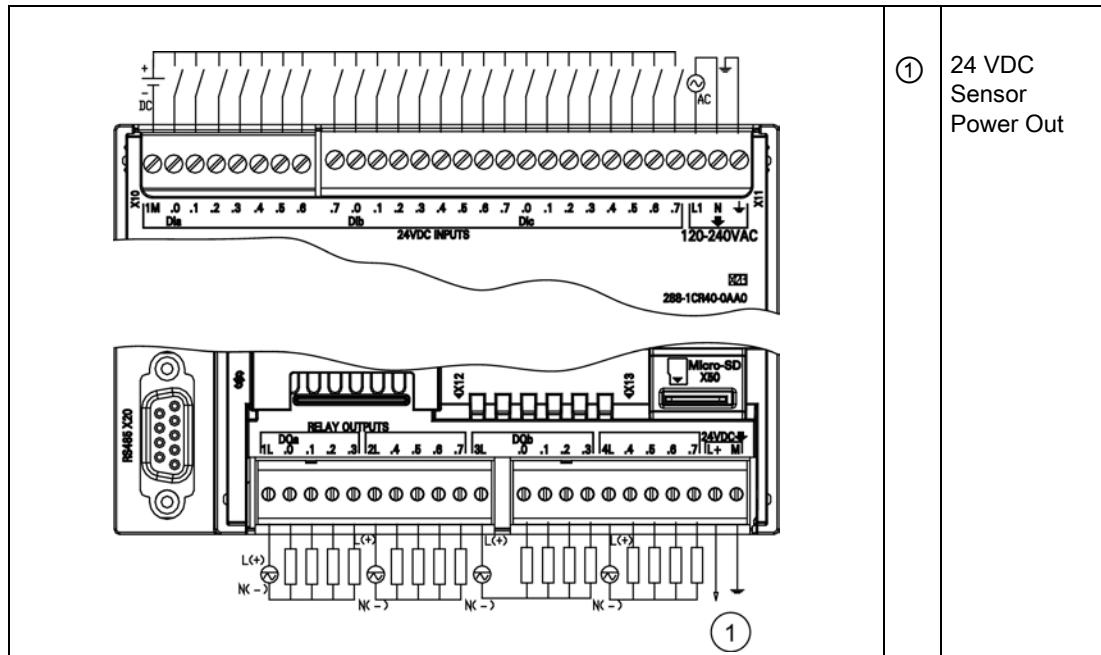


Table A- 48 Connector pin locations for CPU CR40 AC/DC/Relay (6ES7 288-1CR40-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI a.7	1L	DQ b.0
2	DI a.0	DI b.0	DQ a.0	DQ b.1
3	DI a.1	DI b.1	DQ a.1	DQ b.2
4	DI a.2	DI b.2	DQ a.2	DQ b.3
5	DI a.3	DI b.3	DQ a.3	4L
6	DI a.4	DI b.4	2L	DQ b.4
7	DI a.5	DI b.5	DQ a.4	DQ b.5
8	DI a.6	DI b.6	DQ a.5	DQ b.6
9	--	DI b.7	DQ a.6	DQ b.7
10	--	DI c.0	DQ a.7	L+ / 24 VDC Out
11	--	DI c.1	3L	M / 24 VDC Out
12	--	DI c.2	--	--
13	--	DI c.3	--	--
14	--	DI c.4	--	--
15	--	DI c.5	--	--
16	--	DI c.6	--	--
17	--	DI c.7	--	--
18	--	L1 / 120 - 240 VAC	--	--

Pin	X10	X11	X12	X13
19	--	N / 120 - 240 VAC	--	--
20	--	Functional Earth	--	--

A.2.4 CPU ST60, CPU SR60, and CPU CR60

A.2.4.1 General specifications and features

Table A- 49 General specifications

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
Order number	6ES7 288-1ST60-0AA0	6ES7 288-1SR60-0AA0	6ES7 288-1CR60-0AA0
Dimensions W x H x D (mm)	175 x 100 x 81	175 x 100 x 81	175 x 100 x 81
Weight	528.2 grams	611.5 grams	620 grams
Power dissipation	20 W	25 W	20 W
Current available (EM bus)	740 mA max. (5 VDC)	740 mA max. (5 VDC)	--
Current available (24 VDC)	300 mA max. (sensor power)	300 mA max. (sensor power)	300 mA max. (sensor power)
Digital input current consumption (24 VDC)	4 mA/input used	4 mA/input used	4 mA/input used

Table A- 50 CPU features

Technical data	Description	
	CPU ST60, CPU SR60	CPU CR60 AC/DC/Relay
User memory	Program	30 Kbytes
	User data (V)	20 Kbytes
	Retentive	10 Kbytes max. ¹
On-board digital I/O	36 inputs/24 outputs	36 inputs/24 outputs
Process image	256 bits of inputs (I) / 256 bits of outputs (Q)	256 bits of inputs (I) / 256 bits of outputs (Q)
Analog image	56 words of inputs (AI) / 56 words of outputs (AQ)	--
Bit memory (M)	256 bits	256 bits
Temporary (local) memory (L)	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine	64 bytes in the main program and 64 bytes in each subroutine and interrupt routine
Sequential control relays (S)	256 bits	256 bits
Signal modules expansion	6 max.	--
Signal board expansion	1 max.	--

Technical specifications

A.2 S7-200 SMART CPUs

Technical data	Description	
	CPU ST60, CPU SR60	CPU CR60 AC/DC/Relay
High-speed counters	4 total <ul style="list-style-type: none"> • 4 at 200 KHz single phase • 2 at 100 KHz A/B phase 	4 total <ul style="list-style-type: none"> • 4 at 100 KHz single phase • 2 at 50 KHz A/B phase
Pulse outputs ²	3 at 100 KHz	--
Pulse catch inputs	14	14
Cyclic interrupts	2 at 1 ms resolution	2 at 1 ms resolution
Edge interrupts	4 rising and 4 falling (6 and 6 with optional signal board)	4 rising and 4 falling
Memory card	microSDHC card (optional)	microSDHC card (optional)
Real time clock accuracy	120 seconds/month	--
Real time clock retention time	7 days typ./6 days min. at 25°C	--

- ¹ You can configure areas of V memory, M memory, C memory (current values) and portions of T memory (current values on retentive timers) to be retentive, up to the specified maximum amount.
- ² The specified maximum pulse frequency is possible only for CPU models with transistor outputs. Pulse output operation is not recommended for CPU models with relay outputs.

Table A- 51 Performance

Type of instruction	Execution speed
Boolean	150 ns instruction
Move Word	1.2 µs/instruction
Real math	3.6 µs/instruction

Table A- 52 User program elements supported

Element	Description	
POUs	Type/ quantity	Main program: 1 Subroutines: 128 (0 to 127) Interrupt routines: 128 (0 to 127)
	Nesting depth	From main program: 8 subroutine levels From interrupt routine: 4 subroutine levels
Accumulators	Quantity	4
Timers	Type/ quantity	Non-retentive (TON, TOF): 192 Retentive (TONR): 64
Counters	Quantity	256

Table A- 53 Communication

Technical data	Description
Number of ports	Ethernet: 1 Serial ports: 1 (RS485) Add-on serial ports: 1 (with optional RS232/485 signal board)
HMI device	4 per port : RS485, SB CM01(RS232/485 signal board) 8 per port: Ethernet
Programming device (PG)	Ethernet: 1
Connections	Ethernet: 1 for programming device, 8 for HMIs, 8 for CPUs, 8 active GET/PUT, 8 passive GET/PUT RS485: 4 for HMIs per port
CPUs (peer-to-peer)	Ethernet: 8
Data rates	Ethernet: 10/100 Mb/s RS485 system protocols: 9600, 19200, and 187500 b/s RS485 freeport: 1200 to 115200 b/s
Isolation (external signal to PLC logic)	Ethernet: Transformer isolated, 1500 VAC RS485: none
Cable type	Ethernet: CAT5e shielded RS485: PROFIBUS network cable

Table A- 54 Power supply

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
Voltage range	20.4 to 28.8 VDC	85 to 264 VAC	85 to 264 VAC
Line frequency	--	47 to 63 Hz	47 to 63 Hz
Input current (max. CPU only load)	220 mA at 24 VDC (without driving 300 mA sensor power) 500 mA at 24 VDC (with driving 300 mA sensor power)	160 mA at 120 VAC (without driving 300 mA sensor power) 280 mA at 120 VAC (with driving 300 mA sensor power) 90 mA at 240 VAC (without driving 300 mA sensor power) 160 mA at 240 VAC (with driving 300 mA sensor power)	160 mA at 120 VAC (without driving 300 mA sensor power) 280 mA at 120 VAC (with driving 300 mA sensor power) 90 mA at 240 VAC (without driving 300 mA sensor power) 160 mA at 240 VAC (with driving 300 mA sensor power)
CPU with all expansion accessories	710 mA at 24 VDC	370 mA at 120 VAC 220 mA at 240 VAC	--
Inrush current (max.)	11.5 A at 28.8 VDC	16.3 A at 264 VDC	7.3 A at 264 VAC
Isolation (input power to logic)	None	1500 VAC	1500 VAC
Ground leakage, AC line to functional earth	None	None	None

Technical specifications

A.2 S7-200 SMART CPUs

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
Hold up time (loss of power)	20 ms at 24 VDC	30 ms at 120 VAC 200 ms at 240 VAC	50 ms at 120 VAC 400 ms at 240 VAC
Internal fuse, not user replaceable	3 A, 250 V, slow blow	3 A, 250 V, slow blow	3 A, 250 V, slow blow

Table A- 55 Sensor power

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR 60 AC/DC/Relay
Voltage range	20.4 to 28.8 VDC	20.4 to 28.8 VDC	20.4 to 28.8 VDC
Output current rating (max.)	300 mA	300 mA	300 mA
Maximum ripple noise (<10 MHz)	< 1 V peak to peak	< 1 V peak to peak	< 1 V peak to peak
Isolation (CPU logic to sensor power)	Not isolated	Not isolated	Not isolated

A.2.4.2 Digital inputs and outputs

Table A- 56 Digital inputs

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
Number of inputs	36	36	36
Type	Sink/Source (IEC Type 1 sink, except I0.0 to I0.3)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.	30 VDC, max.	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.
Logic 1 signal (min.)	I0.0 to I0.3: 4 VDC at 8 mA Other inputs: 15 VDC at 2.5 mA	I0.0 to I0.3: 4 VDC at 8 mA 15 VDC at 2.5 mA	15 VDC at 2.5 mA
Logic 0 signal (max.)	I0.0 to I0.3: 1 VDC at 1 mA Other inputs: 5 VDC at 1 mA	5 VDC at 1 mA	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	1	1	1
Filter times	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8	Individually selectable on each channel (points I0.0 to I1.5): μs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
	Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8	Individually selectable on each channel (points I0.6 and greater): ms: 0, 6.4, 12.8
HSC clock input rates (max.) (Logic 1 Level = 15 to 26 VDC)	4 HSC at 200 K Hz for single phase 2 HSC at 100 K Hz for A/B phase	4 HSC at 200 K Hz for single phase 2 HSC at 100 K Hz for A/B phase	4 HSC at 100 K Hz for single phase 2 HSC at 50 K Hz for A/B phase
Number of inputs on simultaneously	36	36	36
Cable length (max.), in meters	I0.0 to I0.3: Shielded (only): <ul style="list-style-type: none"> • 500 m normal (low-speed) inputs • 50 m HSC (high-speed) inputs <hr/> All other inputs: Shielded: <ul style="list-style-type: none"> • 500 m normal inputs Unshielded: <ul style="list-style-type: none"> • 300 m normal inputs 	All inputs: Shielded: <ul style="list-style-type: none"> • 500 m normal inputs • 50 m HSC inputs Unshielded: <ul style="list-style-type: none"> • 300 m normal inputs 	All inputs: Shielded: <ul style="list-style-type: none"> • 500 m normal inputs • 50 m HSC inputs Unshielded: <ul style="list-style-type: none"> • 300 m normal inputs

Table A- 57 Digital outputs

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
Number of outputs	24	24	24
Type	Solid state - MOSFET (sourcing)	Relay, dry contact	Relay, dry contact
Voltage range	20.4 to 28.8 VDC	5 to 30 VDC or 5 to 250 VAC	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	20 VDC min.	--	--
Logic 0 signal with 10 KΩ load	0.1 VDC max.	--	--
Rated current per point (max.)	0.5 A	2 A	2 A
Rated current per common (max.)	6 A	10 A	10 A
Lamp load	5 W	30 W DC / 200 W AC	30 W DC / 200 W AC
ON state resistance	0.6 Ω max.	0.2 Ω max. when new	0.2 Ω max. when new
Leakage current per point	10 µA max.	--	--
Surge current	8 A for 100 ms max.	7 A with contacts closed	7 A with contacts closed
Overload protection	No	No	No
Isolation (field side to logic)	500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)	1500 VAC for 1 minute (coil to contact) None (coil to logic)

Technical specifications

A.2 S7-200 SMART CPUs

Technical data	CPU ST60 DC/DC/DC	CPU SR60 AC/DC/Relay	CPU CR60 AC/DC/Relay
Isolation resistance	--	100 M Ω min. when new	100 M Ω min. when new
Isolation between open contacts	--	750 VAC for 1 minute	750 VAC for 1 minute
Isolation groups	3	6	6
Inductive clamp voltage	L+ minus 48 VDC, 1 W dissipation	--	-
Switching delay (Qa.0 to Qa.3)	1.0 µs max., off to on 3.0 µs max., on to off	10 ms max.	10 ms max.
Switching delay (Qa.4 to Qc.7)	50 µs max., off to on 200 µs max., on to off	10 ms max.	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles	100,000 open/close cycles
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	24	24	24
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

A.2.4.3 CPU ST60, SR60 and CR60 wiring diagrams

Table A- 58 Wiring diagram for CPU ST60 DC/DC/DC (6ES7 288-1ST60-0AA0)

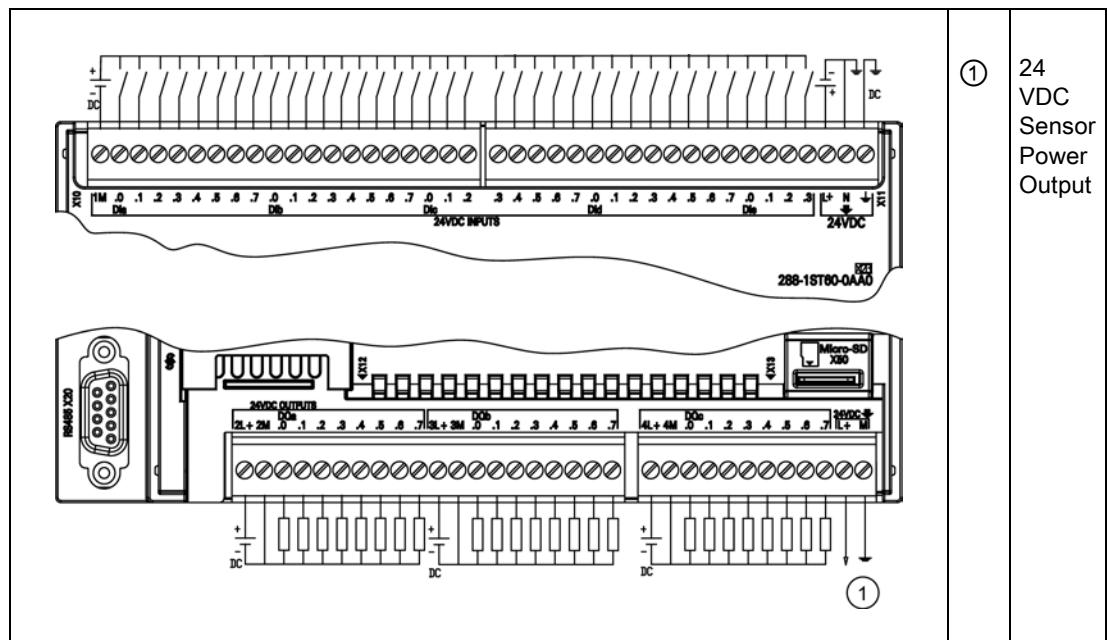


Table A- 59 Connector pin locations for CPU ST60 DC/DC/DC (6ES7 288-1ST60-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI c.3	2L+	4L+
2	DI a.0	DI c.4	2M	4M
3	DI a.1	DI c.5	DQ a.0	DQ c.0
4	DI a.2	DI c.6	DQ a.1	DQ c.1
5	DI a.3	DI c.7	DQ a.2	DQ c.2
6	DI a.4	DI d.0	DQ a.3	DQ c.3
7	DI a.5	DI d.1	DQ a.4	DQ c.4
8	DI a.6	DI d.2	DQ a.5	DQ c.5
9	DI a.7	DI d.3	DQ a.6	DQ c.6
10	DI b.0	DI d.4	DQ a.7	DQ c.7
11	DI b.1	DI d.5	3L+	L+ / 24 VDC Out
12	DI b.2	DI d.6	3M	M / 24 VDC Out
13	DI b.3	DI d.7	DQ b.0	--
14	DI b.4	DI e.0	DQ b.1	--
15	DI b.5	DI e.1	DQ b.2	--
16	DI b.6	DI e.2	DQ b.3	--
17	DI b.7	DI e.3	DQ b.4	--
18	DI c.0	L+ / 24 VDC	DQ b.5	--

Technical specifications

A.2 S7-200 SMART CPUs

Pin	X10	X11	X12	X13
19	DI c.1	M / 24 VDC	DQ b.6	--
20	DI c.2	Functional Earth	DQ b.7	--

Table A- 60 Wiring diagram for CPU SR60 AC/DC/Relay (6ES7 288-1SR60-0AA0)

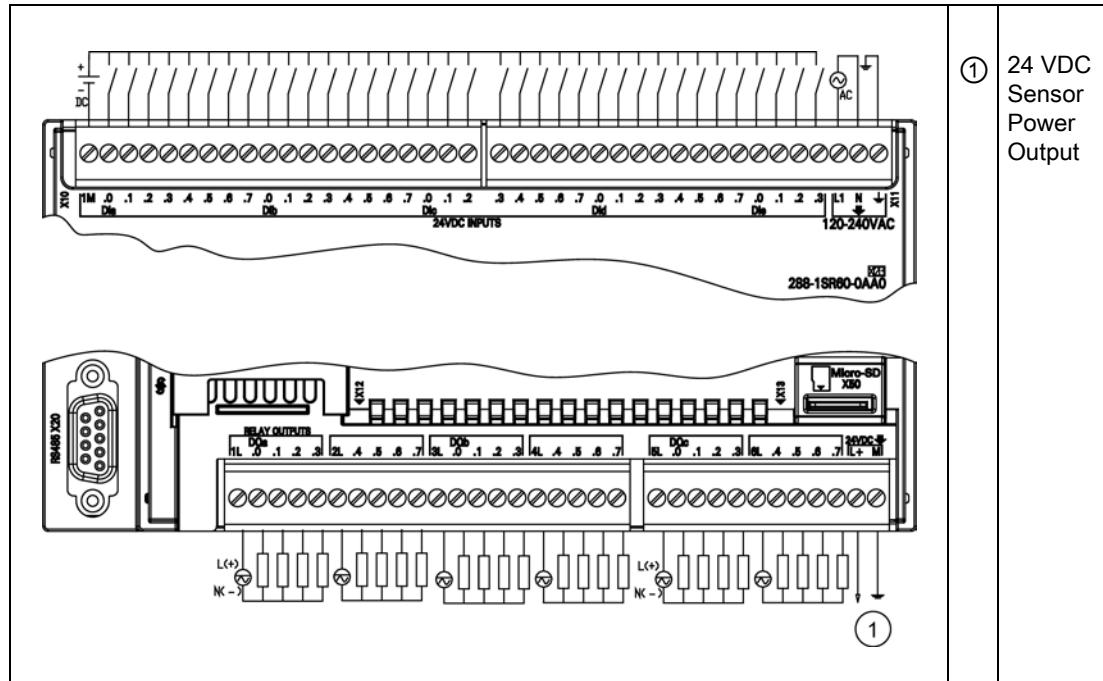


Table A- 61 Connector pin locations for CPU SR60 AC/DC/Relay (6ES7 288-1SR60-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI c.3	1L	5L
2	DI a.0	DI c.4	DQ a.0	DQ c.0
3	DI a.1	DI c.5	DQ a.1	DQ c.1
4	DI a.2	DI c.6	DQ a.2	DQ c.2
5	DI a.3	DI c.7	DQ a.3	DQ c.3
6	DI a.4	DI d.0	2L	6L
7	DI a.5	DI d.1	DQ a.4	DQ c.4
8	DI a.6	DI d.2	DQ a.5	DQ c.5
9	DI a.7	DI d.3	DQ a.6	DQ c.6
10	DI b.0	DI d.4	DQ a.7	DQ c.7
11	DI b.1	DI d.5	3L	L+ / 24 VDC Out
12	DI b.2	DI d.6	DQ b.0	M / 24 VDC Out
13	DI b.3	DI d.7	DQ b.1	--
14	DI b.4	DI e.0	DQ b.2	--
15	DI b.5	DI e.1	DQ b.3	--

Pin	X10	X11	X12	X13
16	DI b.6	DI e.2	4L	--
17	DI b.7	DI e.3	DQ b.4	--
18	DI c.0	L1 / 120 - 240 VAC	DQ b.5	--
19	DI c.1	N / 120 - 240 VAC	DQ b.6	--
20	DI c.2	Functional Earth	DQ b.7	--

Table A- 62 Wiring diagram for CPU CR60 AC/DC/Relay (6ES7 288-1CR60-0AA0)

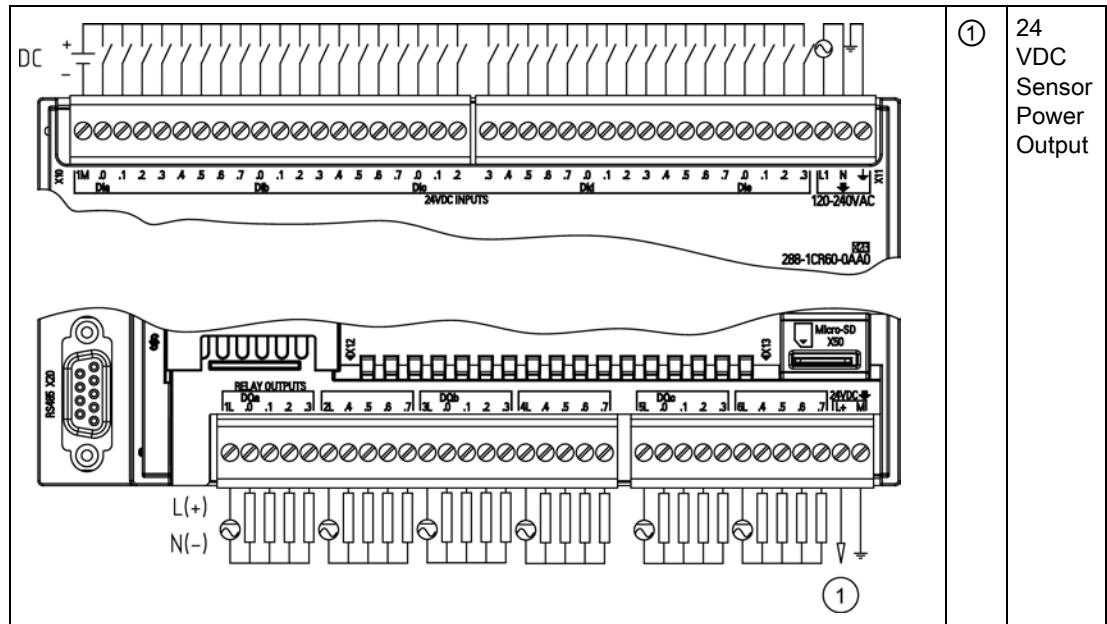


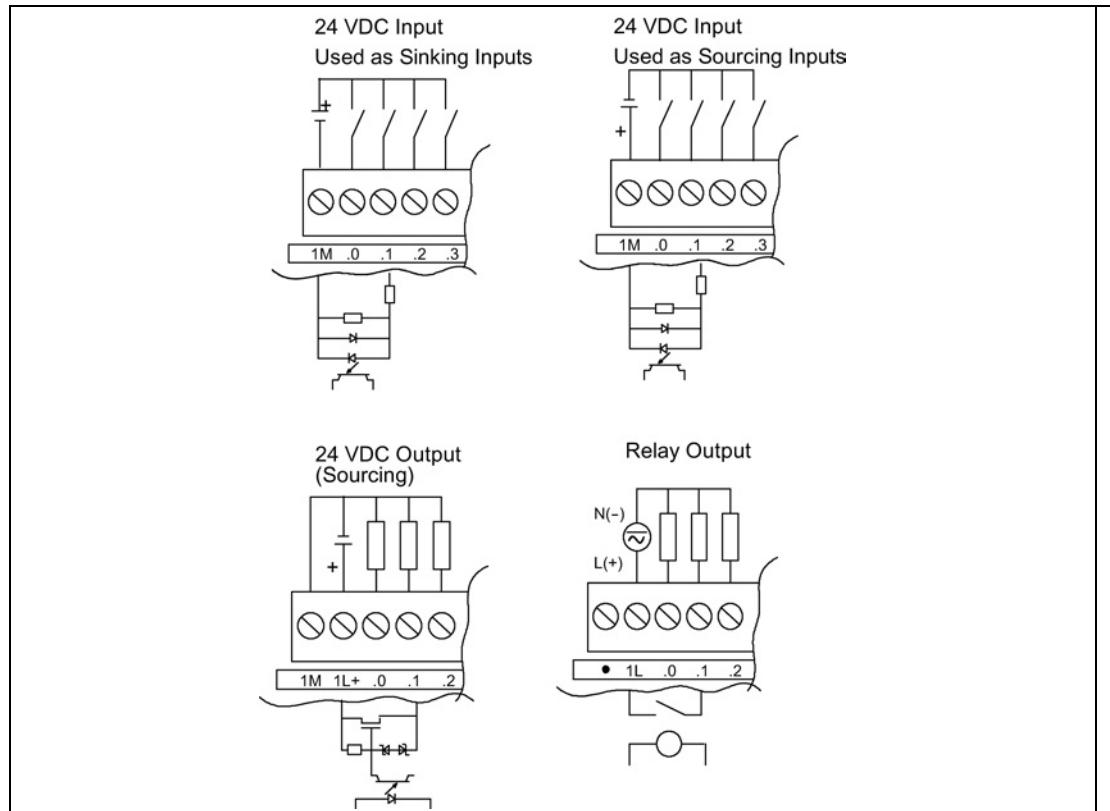
Table A- 63 Connector pin locations for CPU CR60 AC/DC/Relay (6ES7 288-1CR60-0AA0)

Pin	X10	X11	X12	X13
1	1M	DI c.3	1L	5L
2	DI a.0	DI c.4	DQ1.0	DQc.0
3	DI a.1	DI c.5	DQ a.1	DQ c.1
4	DI a.2	DI c.6	DQ a.2	DQ c.2
5	DI a.3	DI c.7	DQ a.3	DQ c.3
6	DI a.4	DI d.0	2L	6L
7	DI a.5	DI d.1	DQ a.4	DQ c.4
8	DI a.6	DI d.2	DQ a.5	DQ c.5
9	DI a.7	DI d.3	DQ a.6	DQ c.6
10	DI b.0	DI d.4	DQ a.7	DQ c.7
11	DI b.1	DI d.5	3L	L+ / 24 VDC Out
12	DI b.2	DI d.6	DQb.0	M / 24 VDC Out
13	DI b.3	DI d.7	DQ b.1	--

Pin	X10	X11	X12	X13
14	DI b.4	DI e.0	DQ b.2	--
15	DI b.5	DI e.1	DQ b.3	--
16	DI b.6	DI e.2	4L	--
17	DI b.7	DI e.3	DQ b.4	--
18	DI c.0	L1 / 24 VDC	DQ b.5	--
19	DI c.1	N / 24 VDC	DQ b.6	--
20	DI c.2	Functional Earth	DQ b.7	--

A.2.5 Wiring diagrams for sink and source input, and relay output

Table A- 64 Wiring diagrams for sink input, source input, and relay output



A.3 Digital inputs and outputs expansion modules (EMs)

A.3.1 EM DI08 digital input specifications

Table A- 65 General specifications

Model	EM Digital 8 x Inputs (EM DI08)
Order number	6ES7 288-2DE08-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	141.4 grams
Power dissipation	1.5 W
Current consumption (SM Bus)	105 mA
Current consumption (24 VDC)	4 mA / input used

Table A- 66 Digital inputs

Model	EM Digital 8 x Inputs (EM DI08)
Number of inputs	8
Type	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC, max.
Surge voltage	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC at 2.5 mA
Logic 0 signal (max.)	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	2
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms (selectable in groups of 4)
Number of inputs on simultaneously	8
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs

Technical specifications

A.3 Digital inputs and outputs expansion modules (EMs)

Table A- 67 Wiring diagram for the EM DI08 Digital 8 x Input (6ES7 288-2DE08-0AA0)

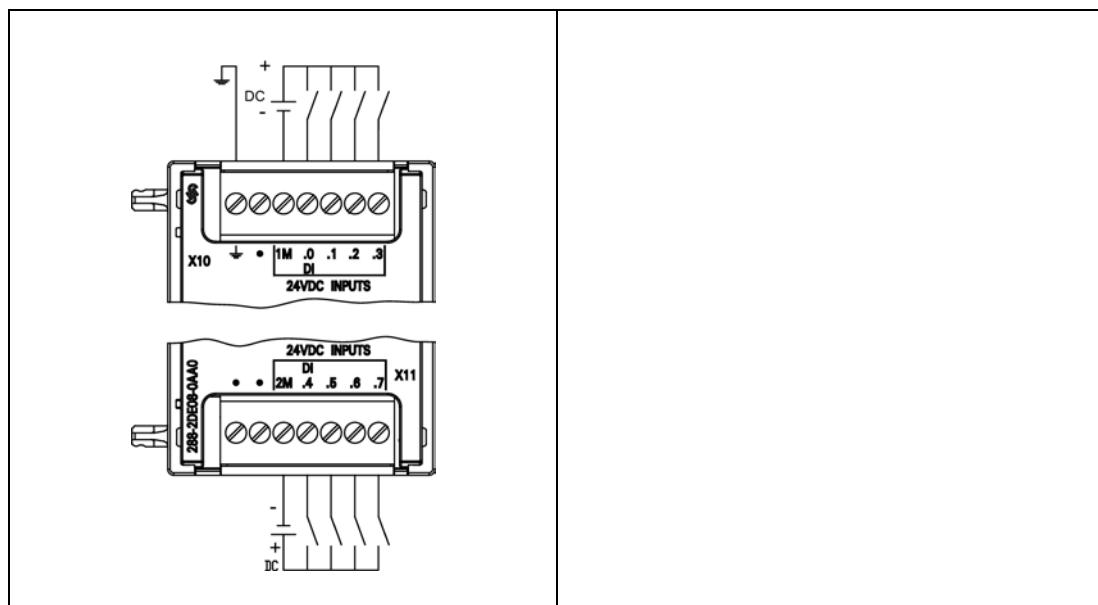


Table A- 68 Connector pin locations for EM DI08 Digital 8 x Input (6ES7 288-2DE08-0AA0)

Pin	X10	X11
1	Functional Earth	No connection
2	No connection	No connection
3	1M	2M
4	DI a.0	DI a.4
5	DI a.1	DI a.5
6	DI a.2	DI a.6
7	DI a.3	DI a.7

A.3.2 EM DT08 and EM DR08 digital output specifications

Table A- 69 General specifications

Model	EM Digital 8 x Outputs (EM DT08)	EM Digital Output 8 x Relay (EM DR08)
Order number	6ES7 288-2DT08-0AA0	6ES7 288-2DR08-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81
Weight	147 grams	166.3 grams
Power dissipation	1.5 W	4.5 W
Current consumption (SM Bus)	120 mA	120 mA
Current consumption (24 VDC)	--	11 mA / Relay coil used

Table A- 70 Digital outputs

Model	EM Digital 8 x Outputs (EM DT08)	EM Digital Output 8 x Relay (EM DR08)
Number of outputs	8	8
Type	Solid state - MOSFET (sourcing)	Relay, dry contact
Voltage range	20.4 to 28.8 V	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	20 V	-
Logic 0 signal with 10K Ω load	0.1 V	-
Rated current per point (max.)	0.75 A	2.0 A
Rated current per common (max.)	3 A	8 A
Lamp load	5 W DC	30 W DC / 200 W AC
ON state contact resistance	0.6 Ω	0.2 Ω max. when new
Leakage current per point	10 μA	--
Surge current	8 A for 100 ms	7 A with contacts closed
Overload protection	No	No
Isolation (field side to logic)	Optical, 500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new
Isolation between open contacts	--	750 VAC for 1 minute
Isolation groups	2	2
Inductive clamp voltage	Minus 48 VDC	--
Switching delay	Switch on less than 50 μs and switch off less than 200 μS	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

Technical specifications

A.3 Digital inputs and outputs expansion modules (EMs)

Table A- 71 Wiring diagrams for the EM DT08 and EM DR08

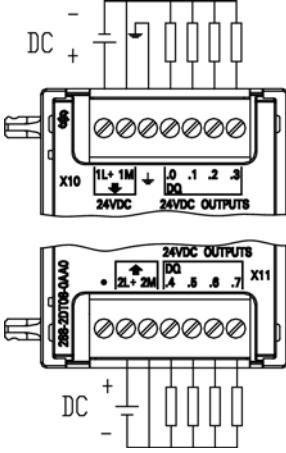
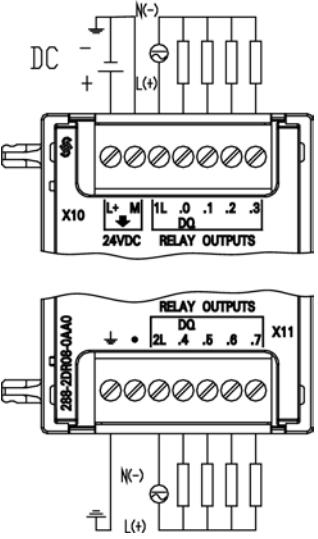
EM DT08 Digital 8 x Outputs (6ES7 288-2DT08-0AA0)	EM DR08 Digital 8 x Outputs x Relay (6ES7 288-2DR08-0AA0)
	

Table A- 72 Connector pin locations for EM DT08 Digital 8 x Outputs (6ES7 288-2DT08-0AA0)

Pin	X10	X11
1	1L+ / 24 VDC	No connection
2	1M / 24 VDC	2L+ / 24 VDC
3	Functional Earth	2M / 24 VDC
4	DQ a.0	DQ a.4
5	DQ a.1	DQ a.5
6	DQ a.2	DQ a.6
7	DQ a.3	DQ a.7

Table A- 73 Connector pin locations for EM DR08 Digital 8 x Outputs x Relay (6ES7 288-2DR08-0AA0)

Pin	X10	X11
1	L+ / 24 VDC	Functional Earth
2	M / 24 VDC	No connection
3	1L	2L
4	DQ a.0	DQ a.4
5	DQ a.1	DQ a.5

Pin	X10	X11
6	DQ a.2	DQ a.6
7	DQ a.3	DQ a.7

A.3.3 EM DT16, EM DR16, EM DT32, and EM DR32 digital input/output specifications

Table A- 74 General specifications

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Order number	6ES7 288-2DT16-0AA0	6ES7 288-2DR16-0AA0	6ES7 288-2DT32-0AA0	6ES7 288-2DR32-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81	45 x 100 x 81	70 x 100 x 81	70 x 100 x 81
Weight	179.7g grams	201.9 grams	257.3 grams	295.4 grams
Power dissipation	2.5 W	5.5 W	4.5 W	10 W
Current consumption (SM Bus)	145 mA	145 mA	185 mA	180 mA
Current consumption (24 VDC)	4 mA / Input used 11 mA / Relay coil used	4 mA / Input used 11 mA / Relay coil used	4 mA / Input used	4 mA / Input used

Table A- 75 Digital inputs

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Number of inputs	8	8	16	16
Type	Sink/Source (IEC Type 1 sink)	SinkSource (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)	Sink/Source (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC max.	30 VDC max.	30 VDC max.	30 VDC max.
Surge voltage	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC	15 VDC	15 VDC	15 VDC
Logic 0 signal (max.)	5 VDC	5 VDC	5 VDC	5 VDC
Isolation (field side to logic)	500 VAC for 1 minute	500 VAC for 1 minute	500 VAC for 1 minute	500 VAC for 1 minute
Isolation groups	2	2	2	2

Technical specifications

A.3 Digital inputs and outputs expansion modules (EMs)

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Filter times	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4	0.2, 0.4, 0.8, 1.6, 3.2, 6.4, and 12.8 ms, selectable in groups of 4
Number of inputs on simultaneously	8	8	16	16
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs

Table A- 76 Digital outputs

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs / 16 x Relay Outputs (EM DR32)
Number of outputs	8	8	16	16
Type	Solid state- MOSFET (sourcing)	Relay, dry contact	Solid state- MOSFET (sourcing)	Relay, dry contact
Voltage range	20.4 to 28.8 VDC 5 to 250 VAC	5 to 30 VDC or 5 to 250 VAC	20.4 to 28.8 VDC	5 to 30 VDC or 5 to 250 VAC
Logic 1 signal at max. current	20 VDC, min.	--	20 VDC, min.	--
Logic 0 signal with 10 KΩ load	0.1 VDC, max.	--	0.1 VDC, max.	--
Rated current per point (max.)	0.75 A	2 A	0.75 A	2 A
Rated current per common (max.)	3 A	8 A	6 A	8 A
Lamp load	5 W	30 W DC/200 W AC	5 W	30 W DC/200 W AC
ON state contact resistance	0.6 Ω max.	0.2 Ω max. when new	0.6 Ω max.	0.2 Ω max. when new
Leakage current per point	10 µA max.	--	10 µA max.	--
Surge current	8 A for 100 ms max.	7 A with contacts closed	8 A for 100 ms max.	7 A with contacts closed
Overload protection	No	No	No	No
Isolation (field side to logic)	500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)	500 VAC for 1 minute	1500 VAC for 1 minute (coil to contact) None (coil to logic)
Isolation resistance	--	100 M Ω min. when new	--	100 M Ω min. when new

A.3 Digital inputs and outputs expansion modules (EMs)

Model	EM Digital 8 x Inputs/ Digital 8 x Outputs (EM DT16)	EM Digital 8 x Inputs/ 8 x Relay Outputs (EM DR16)	EM Digital 16 x Inputs/ Digital 16 x Outputs (EM DT32)	EM Digital 16 x Inputs/ 16 x Relay Outputs (EM DR32)
Isolation between open contacts	--	750 VAC for 1 minute	--	750 VAC for 1 minute
Isolation groups	2	2	3	4
Inductive clamp voltage	Minus 48 V	--	Minus 48 V	--
Switching delay	50 µs max., off to on 200 µs max., on to off	10 ms max.	50 µs max., off to on 200 µs max., on to off	10 ms max.
Lifetime mechanical (no load)	--	10,000,000 open/close cycles	--	10,000,000 open/close cycles
Lifetime contacts at rated load	--	100,000 open/close cycles	--	100,000 open/close cycles
Output behavior in STOP	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)	Last value or substitute value (default value 0)
Number of outputs on simultaneously	8	8	16	16
Cable length (max.), in meters	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m	Shielded: 500 m Unshielded: 150 m

Technical specifications

A.3 Digital inputs and outputs expansion modules (EMs)

Table A- 77 Wiring diagrams for the EM DT16 and EM DR16

EM DT16 Digital 8 x Inputs/Digital 8 x Outputs (6ES7 288-2DT16-0AA0)	EM DR16 Digital 8 x Inputs/ 8 x Relay Outputs (6ES7 288-2DR16-0AA0)

Table A- 78 Connector pin locations for EM DT16 Digital 8 x Inputs/Digital 8 x Outputs (6ES7 288-2DT16-0AA0)

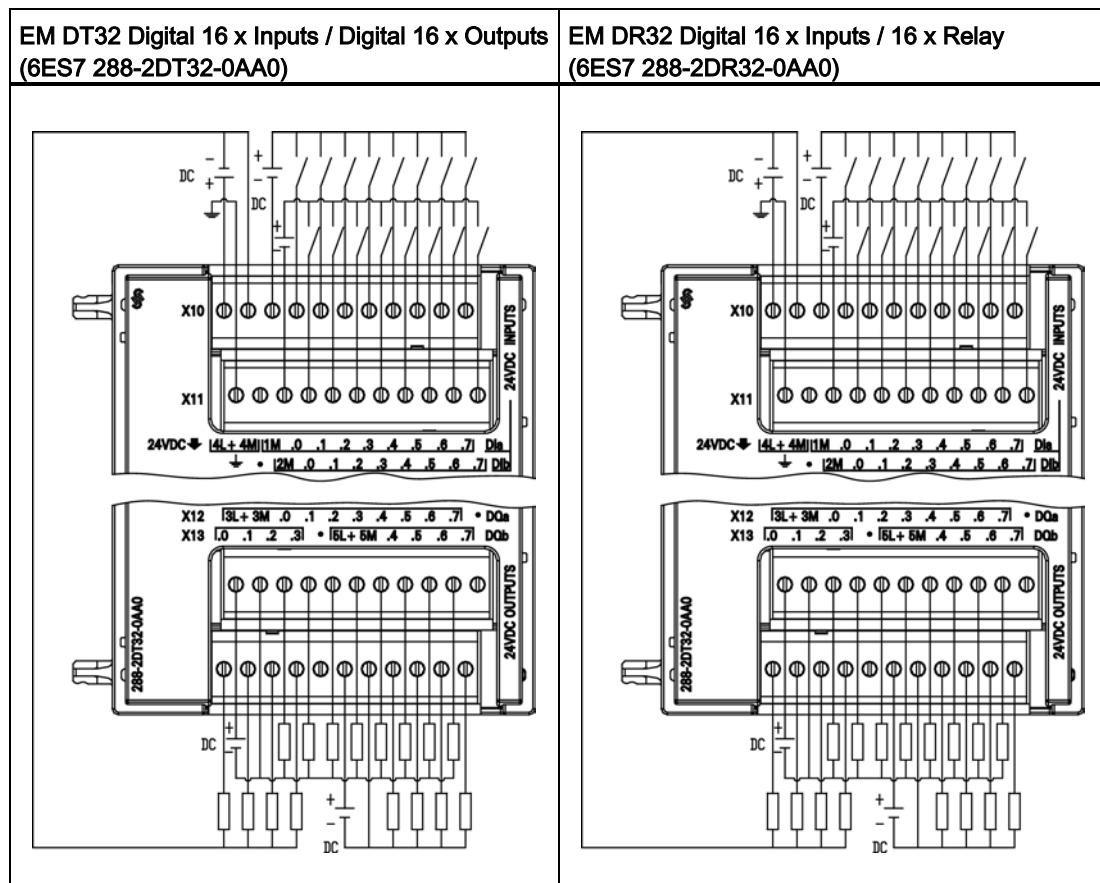
Pin	X10	X11	X12	X13
1	No connection	Functional Earth	No connection	No connection
2	No connection	No connection	3L+ / 24 VDC	4L+ / 24 VDC
3	1M	2M	3M / 24 VDC	4M / 24 VDC
4	DI a.0	DI a.4	DQ a.0	DQ a.4
5	DI a.1	DI a.5	DQ a.1	DQ a.5
6	DI a.2	DI a.6	DQ a.2	DQ a.6
7	DI a.3	DI a.7	DQ a.3	DQ a.7

A.3 Digital inputs and outputs expansion modules (EMs)

Table A- 79 Connector pin locations for EM DR16 Digital 8 x Inputs/ 8 x Relay Outputs (6ES7 288-2DR16-0AA0)

Pin	X10	X11	X12	X13
1	L+ / 24VDC	Functional Earth	No connection	No connection
2	M / 24VDC	No connection	No connection	No connection
3	1M	2M	1L	2L
4	DI a.0	DI a.4	DQ a.0	DQ a.4
5	DI a.1	DI a.5	DQ a.1	DQ a.5
6	DI a.2	DI a.6	DQ a.2	DQ a.6
7	DI a.3	DI a.7	DQ a.3	DQ a.7

Table A- 80 Wiring diagrams for the EM DT32 and EM DR32



Technical specifications

A.4 Analog inputs and outputs expansion modules (EMs)

Table A- 81 Connector pin locations for EM DT32 Digital 16 x Inputs / Digital 16 x Outputs (6ES7 288-2DT32-0AA0)

Pin	X10	X11	X12	X13
1	4L+ / 24 VDC ¹	Functional Earth	3L+ / 24 VDC	DQ b.0 ¹
2	4M / 24 VDC ¹	No connection	3M / 24 VDC	DQ b.1 ¹
3	1M	2M	DQ a.0	DQ b.2 ¹
4	DI a.0	DI b.0	DQ a.1	DQ b.3 ¹
5	DI a.1	DI b.1	DQ a.2	No connection
6	DI a.2	DI b.2	DQ a.3	5L+ / 24 VDC
7	DI a.3	DI b.3	DQ a.4	5M / 24 VDC
8	DI a.4	DI b.4	DQ a.5	DQ b.4
9	DI a.5	DI b.5	DQ a.6	DQ b.5
10	DI a.6	DI b.6	DQ a.7	DQ b.6
11	DI a.7	DI b.7	No connection	DQ b.7

¹ In same isolation group.

Table A- 82 Connector pin locations for EM DR32 Digital 16 x Inputs / 16 x Relay (6ES7 288-2DR32-0AA0)

Pin	X10	X11	X12	X13
1	L+ / 24VDC	Functional Earth	1L	3L
2	M / 24VDC	No connection	DQ a.0	DQ b.0
3	1M	2M	DQ a.1	DQ b.1
4	DI a.0	DI b.0	DQ a.2	DQ b.2
5	DI a.1	DI b.1	DQ a.3	DQ b.3
6	DI a.2	DI b.2	No connection	No connection
7	DI a.3	DI b.3	2L	4L
8	DI a.4	DI b.4	DQ a.4	DQ b.4
8	DI a.5	DI b.5	DQ a.5	DQ b.5
10	DI a.6	DI b.6	DQ a.6	DQ b.6
11	DI a.7	DI b.7	DQ a.7	DQ b.7

A.4 Analog inputs and outputs expansion modules (EMs)

A.4.1 EM AE04 analog input specifications

Table A- 83 General specifications

Model	EM Analog 4 x inputs (EM AE04)
Order number	6ES7 288-3AE04-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	147 grams
Power dissipation	1.5 W (no load)
Current consumption (SM Bus)	80 mA
Current consumption (24 VDC)	40 mA (no load)

Table A- 84 Analog inputs

Model	EM Analog 4 x inputs (EM AE04)
Number of inputs	4
Type	Voltage or current (differential), selectable in groups of 2
Range	±10 V, ±5 V, ±2.5 V, or 0 to 20 mA
Full scale range (data word)	-27,648 to 27,648
Overshoot/undershoot range (data word)	Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4864 to 0
Overflow/underflow (data word)	Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768
Resolution	Voltage mode: 11 bits + sign bit Current mode: 11 bits
Maximum withstand voltage/current	±35 V / ±40 mA
Smoothing	None, weak, medium or strong
Noise rejection	400, 60, 50, or 10 Hz
Input impedance	≥9 M Ω (voltage) / 250 Ω (current)
Isolation (field side to logic)	None
Accuracy (25°C / 0 to 55°C)	Voltage mode: ±0.1% / ±0.2% of full scale Current mode: ±0.2% / ±0.3% of full scale
Measuring principle	625 µS (400 Hz rejection)
Common mode rejection	40 dB, DC to 60 Hz
Operational signal range	Signal plus common mode voltage must be less than +12 V and greater than -12 V
Cable length (max.), in meters	100 m twisted and shielded

Technical specifications

A.4 Analog inputs and outputs expansion modules (EMs)

Table A- 85 Diagnostics

Model	EM Analog 4 x inputs (EM AE04)
Overflow/underflow	Yes
24 VDC low voltage	Yes

Table A- 86 Wiring diagram EM AE04 Analog 4 x Inputs (6ES7 288-3AE04-0AA0)

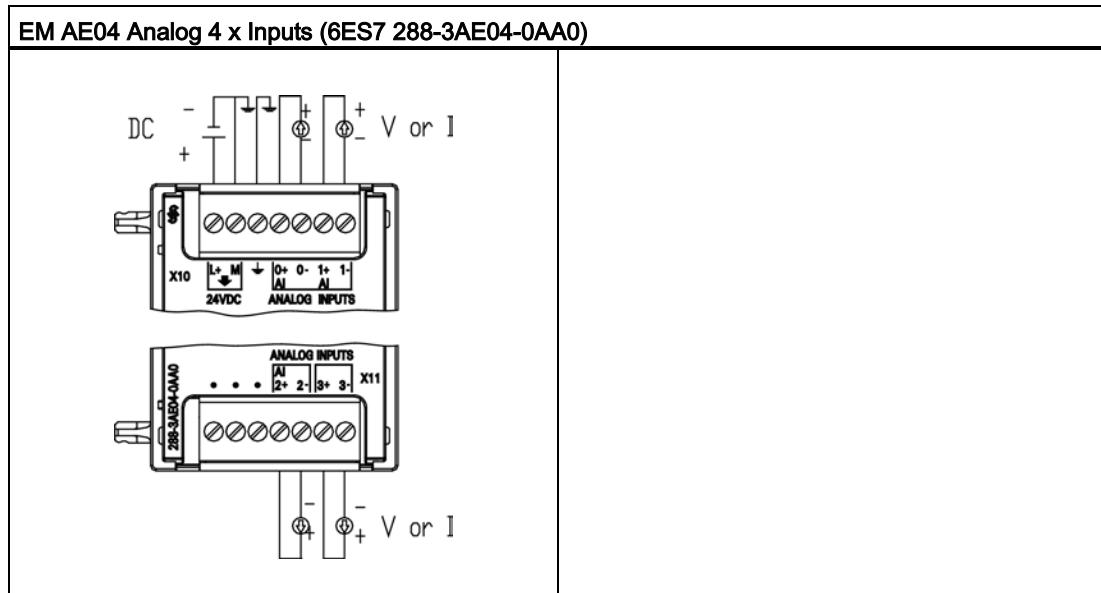


Table A- 87 Connector pin locations for EM AE04 Analog 4 x Inputs (6ES7 288-3AE04-0AA0)

Pin	X10	X11
1	L+ / 24 VDC	No connection
2	M / 24 VDC	No connection
3	Functional Earth	No connection
4	AI 0+	AI 2+
5	AI 0-	AI 2-
6	AI 1+	AI 3+
7	AI 1-	AI 3-

A.4.2 EM AQ02 analog output module specifications

Table A- 88 General specifications

Technical data	EM Analog 2 x outputs (EM AQ02)
Order number	6ES7 288-3AQ02-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	147.1 grams
Power dissipation	1.5 W (no load)
Current consumption (SM Bus)	80 mA
Current consumption (24 VDC)	50 mA (no load)

Table A- 89 Analog outputs

Technical data	EM Analog 2 x outputs (EM AQ02)
Number of outputs	2
Type	Voltage or current
Range	± 10 V or 0 to 20 mA
Resolution	Voltage mode: 10 bits + sign bit Current mode: 10 bits
Full scale range (data word)	Voltage: -27,648 to 27,648 Current: 0 to 27,648
Accuracy (25°C / 0 to 55°C)	$\pm 0.5\%$ / $\pm 1.0\%$ of full scale
Settling time (95% of new value)	Voltage: 300 μ s (R), 750 μ s (R), 750 μ s (1 μ F) Current: 600 μ s (1 mH), 2 ms (10 mH)
Load impedance	Voltage: $\geq 1000 \Omega$ Current: $\leq 500 \Omega$
Output behavior in STOP	Last value or substitute value (default value 0)
Isolation (field side to logic)	None
Cable length (max.), in meters	100 m twisted and shielded

Table A- 90 Diagnostics

Technical data	EM Analog 2 x outputs (EM AQ02)
Overflow/underflow	Yes
Short to ground (voltage mode only)	Yes
Wire break (current mode only)	Yes
24 VDC low voltage	Yes

Technical specifications

A.4 Analog inputs and outputs expansion modules (EMs)

Table A- 91 Wiring diagram for the EM AQ02 Analog 2 x Outputs (6ES7 288-3AQ02-0AA0)

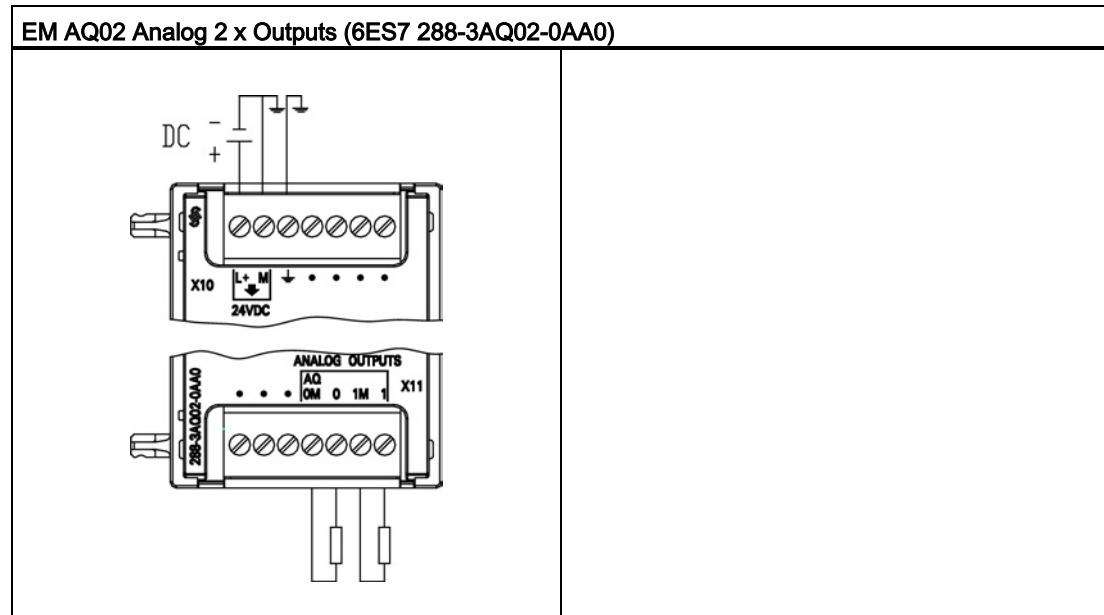


Table A- 92 Connector pin locations for EM AQ02 Analog 2 x Outputs (6ES7 288-3AQ02-0AA0)

Pin	X10	X11
1	L+ / 24 VDC	No connection
2	M / 24 VDC	No connection
3	Functional Earth	No connection
4	No connection	AQ 0M
5	No connection	AQ 0
6	No connection	AQ 1M
7	No connection	AQ 1

A.4.3 EM AM06 analog input/output module specifications

Table A- 93 General specifications

Technical data	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Order number	6ES7 288-3AM06-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	173.4 grams
Power dissipation	2.0 W (no load)
Current consumption (SM Bus)	80 mA
Current consumption (24 VDC)	60 mA (no load)

Table A- 94 Analog inputs

Model	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Number of inputs	4
Type	Voltage or current (differential): Selectable in groups of 2
Range	± 10 V, ± 5 V, ± 2.5 V, or 0 to 20 mA
Full scale range (data word)	-27,648 to 27,648
Overshoot/undershoot range (data word)	Voltage: 27,649 to 32,511 / -27,649 to -32,512 Current: 27,649 to 32,511 / -4,864 to 0
Overflow/underflow (data word)	Voltage: 32,512 to 32,767 / -32,513 to -32,768 Current: 32,512 to 32,767 / -4,865 to -32,768
Resolution	Voltage mode: 11 bits + sign Current mode: 11 bits
Maximum withstand voltage/current	± 35 V / ± 40 mA
Smoothing	None, weak, medium, or strong
Noise rejection	400, 60, 50, or 10 Hz
Input impedance	≥ 9 M Ω
Isolation (field side to logic)	None
Accuracy (25°C / 0 to 55°C)	Voltage mode: $\pm 0.1\%$ / $\pm 0.2\%$ of full scale Current mode: $\pm 0.2\%$ / $\pm 0.3\%$ of full scale
Analog to digital conversion time	625 μ s (400 Hz rejection)
Common mode rejection	40 dB, DC to 60 Hz
Operational signal range	Signal plus common mode voltage must be less than +12 V and greater than -12 V
Cable length (max.), in meters	100 m twisted and shielded

Technical specifications

A.4 Analog inputs and outputs expansion modules (EMs)

Table A- 95 Analog outputs

Technical data	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Number of outputs	2
Type	Voltage or current
Range	± 10 V or 0 to 20 mA
Resolution	Voltage mode: 10 bits + sign Current mode: 10 bits
Full scale range (data word)	Voltage: -27,648 to 27,648 Current: 0 to 27,648
Accuracy (25°C / 0 to 55°C)	$\pm 0.5\%$ / $\pm 1.0\%$ of full scale
Settling time (95% of new value)	Voltage: 300 μ s (R), 750 μ s (1 μ F) Current: 600 μ s (1 mH), 2 ms (10 mH)
Load impedance	Voltage: $\geq 1000 \Omega$ Current: $\leq 500 \Omega$
Output behavior in STOP	Last value or substitute value (default value 0)
Isolation (field side to logic)	None
Cable length (max.), in meters	100 m twisted and shielded

Table A- 96 Diagnostics

Model	EM 4 x Analog Inputs / 2 x Analog Outputs (AM06)
Overflow/underflow	Yes
Short to ground (voltage mode only)	Yes
Wire break (current mode only)	Yes
24 VDC low voltage	Yes

A.4 Analog inputs and outputs expansion modules (EMs)

Table A- 97 Wiring diagram for the AM06 4 x Analog Inputs / 2 x Analog Outputs (6ES7 288-3AM06-0AA0)

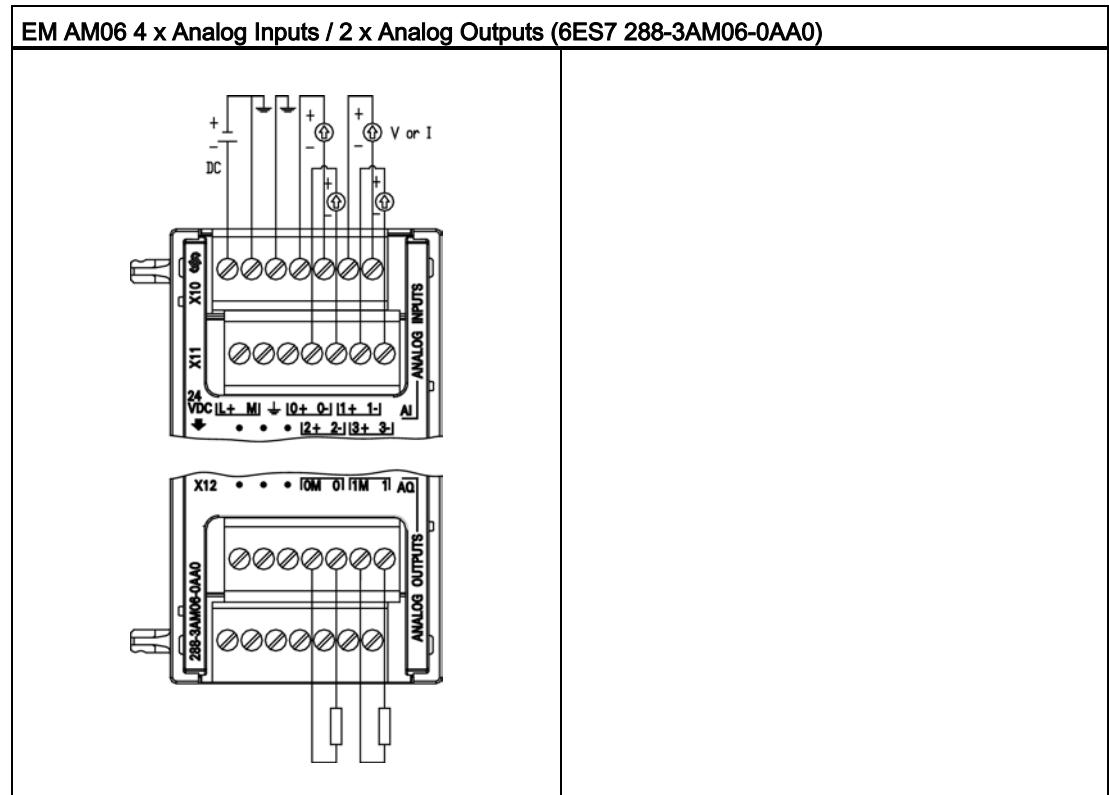


Table A- 98 Connector pin locations for AM06 4 x Analog Inputs / 2 x Analog Outputs (6ES7 288-3AM06-0AA0)

Pin	X10	X11	X12
1	L+ / 24 VDC	No connection	No connection
2	M / 24 VDC	No connection	No connection
3	Functional Earth	No connection	No connection
4	AI 0+	AI 2+	AQ 0M
5	AI 0-	AI 2-	AQ 0
6	AI 1+	AI 3+	AQ 1M
7	AI 1-	AI 3-	AQ 1

Technical specifications

A.5 Thermocouple and RTD expansion modules (EMs)

A.5 Thermocouple and RTD expansion modules (EMs)

A.5.1 Thermocouple expansion modules (EMs)

A.5.1.1 EM AT04 thermocouple specifications

Table A- 99 General specifications

Model	EM AT04 AI 4 x 16 bit TC
Order number	6ES7 288-3AT04-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	125 grams
Power dissipation	1.5 W
Current consumption (SM Bus)	80 mA
Current consumption (24 VDC) ¹	40 mA

¹ 20.4 to 28.8 VDC (Class 2, Limited Power, or sensor power from PLC)

Table A- 100 Analog inputs

Model	EM AT04 AI 4 x 16 bit TC
Number of inputs	4
Range	See Thermocouple selection table.
Nominal range (data word)	
Overrange/underrange (data word)	
Overflow/underflow (data word)	
Resolution	Temperature 0.1° C / 0.1° F
	Voltage 15 bits plus sign
Maximum withstand voltage	± 35
Noise rejection	85 dB for selected filter setting (10 Hz, 50 Hz, 60 Hz or 400 Hz)
Common mode rejection	> 120 dB at 120 VAC
Impedance	≥ 10 MΩ
Isolation	Field to logic 500 VAC
	Field to 24 VDC 500 VAC
	24 VDC to logic 500 VAC
Channel to channel isolation	--
Accuracy	See Thermocouple selection table.
Repeatability	±0.05% FS

Model	EM AT04 AI 4 x 16 bit TC
Measuring principle	Integrating
Module update time	See Filter selection table.
Cold junction error	±1.5°C
Cable length (meters)	100 m to sensor max.
Wire resistance	100 Ω max.

Table A- 101 Diagnostics

Model	EM AT04 AI 4 x 16 bit TCSM 1231 AI 4 x 16 bit TC
Overflow/underflow ¹	Yes
Wire break (current mode only) ²	Yes
24 VDC low voltage ¹	Yes

¹ The overflow, underflow and low voltage diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

² When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The EM AT04 Thermocouple (TC) analog signal module measures the value of voltage connected to the module inputs. The temperature measurement type can be either "Thermocouple" or "Voltage".

- "Thermocouple": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).
- "Voltage": The nominal range full scale value will be decimal 27648.

Technical specifications

A.5 Thermocouple and RTD expansion modules (EMs)

Table A- 102 Wiring diagrams for the thermocouple EM AT04

EM AT04 4 x 16 bit (6ES7 288-3AT04-0AA0)	

Note: Connectors must be gold. See Appendix C, Spare Parts for order number.

① TC 2, 3, 4, and 5 not shown connected for clarity.

Table A- 103 Connector pin locations for EM AT04 4 x 16 bit (6ES7 288-3AT04-0AA0)

Pin	X10 (gold)	X11 (gold)
1	L+ / 24VDC	No connection
2	M / 24VDC	No connection
3	GND	No connection
4	AI 0+ /TC	AI 2+ /TC
5	AI 0- /TC	AI 2- /TC
6	AI 1+ /TC	AI 3+ /TC
7	AI 1- /TC	AI 3- /TC

Note

Unused analog inputs should be shorted.

The thermocouple unused channels can be deactivated. No error will occur if an unused channel is deactivated.

Thermocouples are formed whenever two dissimilar metals are electrically bonded to each other. A voltage is generated that is proportional to the junction temperature. This voltage is small; one microvolt could represent many degrees. Measuring the voltage from a thermocouple, compensating for extra junctions, and then linearizing the result forms the basis of temperature measurement using thermocouples.

When you connect a thermocouple to the EM AT04 Thermocouple module, the two dissimilar metal wires are attached to the module at the module signal connector. The place where the two dissimilar wires are attached to each other forms the sensor thermocouple.

Two more thermocouples are formed where the two dissimilar wires are attached to the signal connector. The connector temperature causes a voltage that adds to the voltage from the sensor thermocouple. If this voltage is not corrected, then the temperature reported will deviate from the sensor temperature.

Cold junction compensation is used to compensate for the connector thermocouple. Thermocouple tables are based on a reference junction temperature, usually zero degrees Celsius. The cold junction compensation compensates the connector to zero degrees Celsius. The cold junction compensation restores the voltage added by the connector thermocouples. The temperature of the module is measured internally, and then converted to a value to be added to the sensor conversion. The corrected sensor conversion is then linearized using the thermocouple tables.

For optimum operation of the cold junction compensation, the thermocouple module must be located in a thermally stable environment. Slow variation (less than 0.1° C/minute) in ambient module temperature is correctly compensated within the module specifications. Air movement across the module will also cause cold junction compensation errors.

If better cold junction error compensation is needed, an external iso-thermal terminal block may be used. The thermocouple module provides for use of a 0° C referenced or 50° C referenced terminal block.

The ranges and accuracy for the different thermocouple types supported by the SM 1231 Thermocouple signal module are shown in the table below.

Table A- 104 EM AT04 Thermocouple selection table

Type	Under-range minimum ¹	Nominal range low limit	Nominal range high limit	Over-range maximum ²	Normal range ^{3, 4} accuracy @ 25°C	Normal range ^{1, 2} accuracy -20°C to 55°C
J	-210.0°C	-150.0°C	1200.0°C	1450.0°C	±0.3°C	±0.6°C
K	-270.0°C	-200.0°C	1372.0°C	1622.0°C	±0.4°C	±1.0°C
T	-270.0°C	-200.0°C	400.0°C	540.0°C	±0.5°C	±1.0°C
E	-270.0°C	-200.0°C	1000.0°C	1200.0°C	±0.3°C	±0.6°C
R & S	-50.0°C	100.0°C	1768.0°C	2019.0°C	±1.0°C	±2.5°C
N	-270.0°C	-200.0°C	1300.0°C	1550.0°C	±1.0°C	±1.6°C
Voltage	-32512	-27648 -80mV	27648 80mV	32511	±0.05%	±0.1%

¹ Thermocouple values below the under-range minimum value are reported as -32768.

² Thermocouple values above the over-range minimum value are reported as 32767.

³ Internal cold junction error is ±1.5°C for all ranges. This adds to the error in this table. The module requires at least 30 minutes of warm-up time to meet this specification.

⁴ In the presence of radiated radio frequency of 970 MHz to 990 MHz, the accuracy of the EM AT04 AI 4 x 16 bit TC may be degraded.

Technical specifications

A.5 Thermocouple and RTD expansion modules (EMs)

Note

Thermocouple channel

Each channel on the Thermocouple signal module can be configured with a different thermocouple type (selectable in the software during configuration of the module).

Table A- 105 Noise reduction and update times for the EM AT04 Thermocouple

Rejection frequency selection	Integration time	4 Channel module update time (seconds)
400 Hz (2.5 ms)	10 ms ¹	0.143
60 Hz (16.6 ms)	16.67 ms	0.223
50 Hz (20 ms)	20 ms	0.263
10 Hz (100 ms)	100 ms	1.225

¹ To maintain module resolution and accuracy when 400 Hz rejection is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

It is recommended for measuring thermocouples that a 100 ms integration time be used. The use of smaller integration times will increase the repeatability error of the temperature readings.

Note

After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior of the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

Representation of analog values for Thermocouple Type J

A representation of the analog values of thermocouples type J is shown in the table below.

Table A- 106 Representation of analog values of thermocouples type J

Type J in °C	Units		Type J in °F	Units		Range
	Decimal	Hexadecimal		Decimal	Hexadecimal	
> 1450.0	32767	7FFF	> 2642.0	32767	7FFF	Overflow
1450.0	14500	38A4	2642.0	26420	6734	Overrange
:	:	:	:	:	:	
1200.1	12001	2EE1	2192.2	21922	55A2	
1200.0	12000	2EE0	2192.0	21920	55A0	Rated range
:	:	:	:	:	:	
-150.0	-1500	FA24	-238.0	-2380	F6B4	

Type J in °C	Units		Type J in °F	Units		Range
	Decimal	Hexadecimal		Decimal	Hexadecimal	
-150.1	-1501	FA23	-238.2	-2382	F6B2	Underrange
:	:	:	:	:	:	
-210.0	-2100	F7CC	-346.0	-3460	F27C	
< -210.0	-32768	8000	< -346.0	-32768	8000	Underflow ¹

¹ Faulty wiring (for example, polarity reversal, or open inputs) or sensor error in the negative range (for example, wrong type of thermocouple) may cause the thermocouple module to signal underflow.

A.5.2 RTD expansion modules (EMs)

EM RTD specifications

Table A- 107 General specifications

Technical data	EM RTD 2 x 16 bit (EM AR02)
Order number	6ES7 288-3AR02-0AA0
Dimensions W x H x D (mm)	45 x 100 x 81
Weight	148.7 grams
Power dissipation	1.5 W
Current consumption (SM Bus)	80 mA
Current consumption (24 VDC) ¹	40 mA

Table A- 108 Analog inputs

Technical data	EM RTD 2 x 16 bit (EM AR02)
Number of inputs	2
Type	Module referenced RTD and Ω
Range	See RTD Sensor selection table
Nominal range (data word)	
Overshoot/undershoot range (data word)	
Overflow/underflow (data word)	
Resolution	Temperature 0.1° C / 0.1° F
	Resistance 15 bits + sign bit
Maximum withstand voltage	± 35 V
Noise rejection	85 dB at 10 Hz / 50 Hz / 60 Hz / 400 Hz
Common mode rejection	>120 dB
Impedance	≥ 10 M Ω

Technical specifications

A.5 Thermocouple and RTD expansion modules (EMs)

Technical data			EM RTD 2 x 16 bit (EM AR02)
Isolation	Field side to logic	500 VAC	
	Field to 24 VDC	500 VAC	
	24 VDC to logic	500 VAC	
Channel to channel isolation	0		
Accuracy	See RTD Sensor selection table		
Repeatability	$\pm 0.05\%$ FS		
Maximum sensor dissipation	0.5 m W		
Measuring principle	Sigma-delta		
Module update time	See Noise reduction selection table		
Cable length (max.), in meters	100 m to sensor max.		
Wire resistance (max.)	except 10 Ω RTD	20 Ω	
	10 Ω RTD	2.7 Ω	

Table A- 109 Diagnostics

Technical data		EM RTD 2 x 16 bit (EM AR02)
Overflow/underflow ^{1,2}		Yes
Wire break ³		Yes
24 VDC low voltage ¹		Yes

¹ The overflow, underflow and low voltage diagnostic alarm information will be reported in the analog data values even if the alarms are disabled in the module configuration.

² For resistance ranges underflow detection is never enabled.

³ When wire break alarm is disabled and an open wire condition exists in the sensor wiring, the module may report random values.

The EM RTD analog signal module measures the value of resistance connected to the module inputs. The measurement type can be selected as either "Resistor" or "Thermal resistor".

- "Resistor": The nominal range full scale value will be decimal 27648.
- "Thermal resistor": The value will be reported in degrees multiplied by ten (for example, 25.3 degrees will be reported as decimal 253).

The EM RTD module supports measurements with 2-wire, 3-wire and 4-wire connections to the sensor resistor.

Table A- 110 Ranges and accuracy for the different sensors supported by the RTD expansion module

Temperature coefficient	RTD type	Under range minimum ¹	Nominal range low limit	Nominal range high limit	Over range maximum ²	Normal range accuracy @ 25°C	Normal range accuracy -20°C to 60°C
Pt 0.003850 ITS90 DIN EN 60751	Pt 10	-243.0°C	-200.0°C	850.0°C	1000.0°C	±1.0°C	±2.0°C
	Pt 50	-243.0°C	-200.0°C	850.0°C	1000.0°C	±0.5°C	±1.0°C
	Pt 100						
	Pt 200						
	Pt 500						
	Pt 1000						
Pt 0.003902 Pt 0.003916 Pt 0.003920	Pt 100	-243.0°C	-200.0°C	850.0°C	1000.0°C	± 0.5°C	±1.0°C
	Pt 200	-243.0°C	-200.0°C	850.0°C	1000.0°C	± 0.5°C	±1.0°C
	Pt 500						
	Pt 1000						
	Pt 10	-273.2°C	-240.0°C	1100.0°C	1295°C	±1.0°C	±2.0°C
	Pt 50	-273.2°C	-240.0°C	1100.0°C	1295°C	±0.8°C	±1.6°C
	Pt 100						
	Pt 500						
	Ni 100	-105.0°C	-60.0°C	250.0°C	295.0°C	±0.5°C	±1.0°C
Ni 0.006720 Ni 0.006180	Ni 120						
	Ni 200						
	Ni 500						
	Ni 1000						
LG-Ni 0.005000	LG-Ni 1000	-105.0°C	-60.0°C	250.0°C	295.0°C	±0.5°C	±1.0°C
Ni 0.006170	Ni 100	-105.0°C	-60.0°C	180.0°C	212.4°C	±0.5°C	±1.0°C
Cu 0.004270	Cu 10	-240.0°C	-200.0°C	260.0°C	312.0°C	±1.0°C	±2.0°C
Cu 0.004260	Cu 10	-60.0°C	-50.0°C	200.0°C	240.0°C	±1.0°C	±2.0°C
	Cu 50	-60.0°C	-50.0°C	200.0°C	240.0°C	±0.6°C	±1.2°C
	Cu 100						
Cu 0.004280	Cu 10	-240.0°C	-200.0°C	200.0°C	240.0°C	±1.0°C	±2.0°C
	Cu 50	-240.0°C	-200.0°C	200.0°C	240.0°C	±0.7°C	±1.4°C
	Cu 100						

¹ RTD values below the under-range minimum value report -32768.

² RTD values above the over-range maximum value report +32767.

Technical specifications

A.5 Thermocouple and RTD expansion modules (EMs)

Table A- 111 Resistance

Range	Under range minimum	Nominal range low limit	Nominal range high limit	Over range maximum ¹	Normal range accuracy @ 25°C	Normal range accuracy -20°C to 60°C
150 Ω	n/a	0 (0 Ω)	27648 (150 Ω)	176.383 Ω	±0.05%	±0.1%
300 Ω	n/a	0 (0 Ω)	27648 (300 Ω)	352.767 Ω	±0.05%	±0.1%
600 Ω	n/a	0 (0 Ω)	27648 (600 Ω)	705.534 Ω	±0.05%	±0.1%

¹ Resistance values above the over-range minimum value are reported as +32767.

Note

The module reports 32767 on any activated channel with no sensor connected. If open wire detection is also enabled, the module flashes the appropriate red LEDs.

When 500 Ω and 1000 Ω RTD ranges are used with other lower value resistors, the error may increase to two times the specified error.

Best accuracy will be achieved for the 10 Ω RTD ranges if 4 wire connections are used.

The resistance of the connection wires in 2 wire mode will cause an error in the sensor reading and therefore accuracy is not guaranteed.

Table A- 112 Noise reduction and update times for the RTD module

Rejection frequency selection	Integration time	Update time (seconds)
400 Hz (2.5 ms)	10 ms ¹	4-/2-wire: 0.142 3-wire: 0.285
60 Hz (16.6 ms)	16.67 ms	4-/2-wire: 0.222 3-wire: 0.445
50 Hz (20 ms)	20 ms	4-/2-wire: 0.262 3-wire: .505
10 Hz (100 ms)	100 ms	4-/2-wire: 1.222 3-wire: 2.445

¹ To maintain module resolution and accuracy when the 400 Hz filter is selected, the integration time is 10 ms. This selection also rejects 100 Hz and 200 Hz noise.

Note

After power is applied, the module performs internal calibration for the analog-to-digital converter. During this time the module reports a value of 32767 on each channel until valid data is available on that channel. Your user program may need to allow for this initialization time. Because the configuration of the module can vary the length of the initialization time, you should verify the behavior of the module in your configuration. If required, you can include logic in your user program to accommodate the initialization time of the module.

Table A- 113 Wiring diagram for the EM AR02 RTD 2 x 16 bit (6ES7 288-3AR02-0AA0)

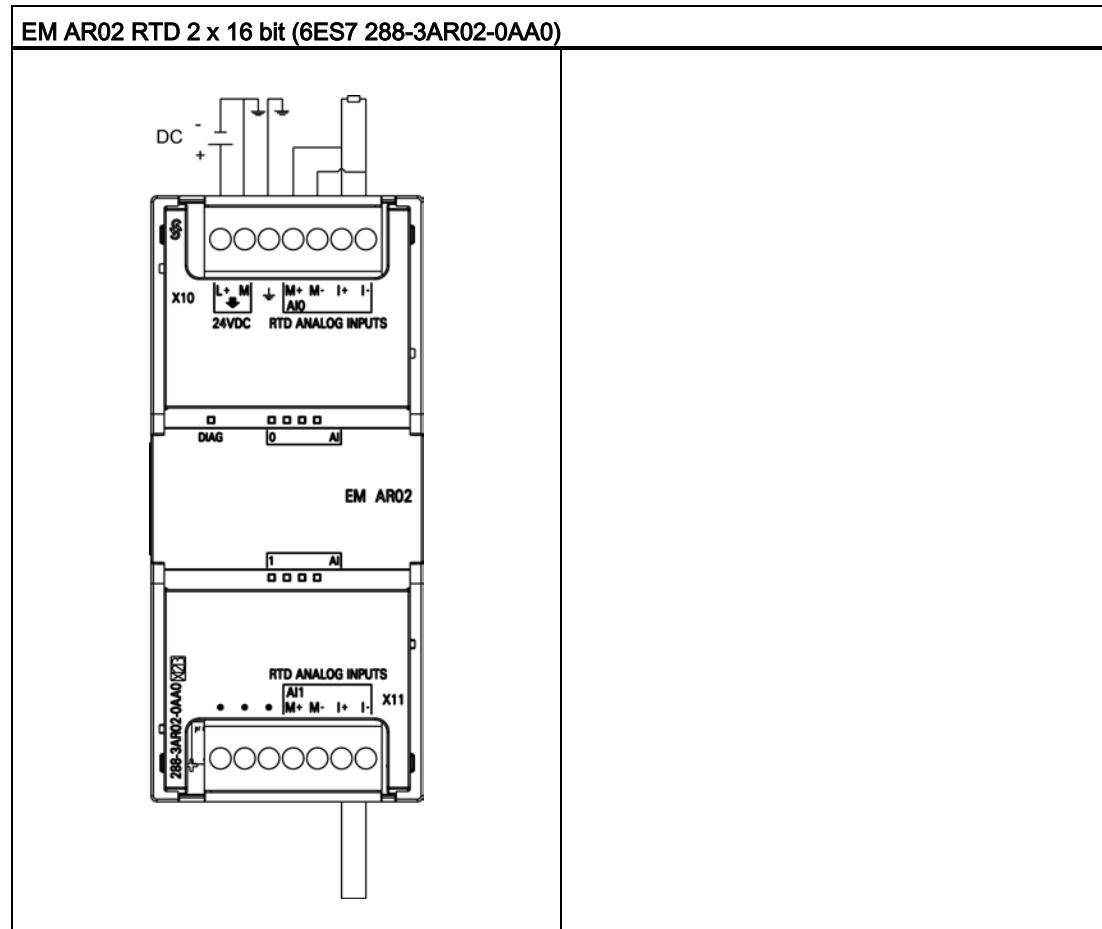


Table A- 114 Connector pin locations for EM AR02 RTD 2 x 16 bit (6ES7 288-3AR02-0AA0)

Pin	X10	X11
1	L+ / 24 VDC	No connection
2	M / 24 VDC	No connection
3	Functional Earth	No connection
4	AI0 M+	AI1 M+
5	AI0 M-	AI1 M-
6	AI0 I+	AI1 I+
7	AI0 I-	AI1 I-

Technical specifications

A.6 Digital signal boards

A.6 Digital signal boards

A.6.1 SB DT04 digital input/output specifications

Table A- 115 General specifications

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Order number	6ES7 288-5DT04-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	18.1 grams
Power dissipation	1.0 W
Current consumption (5 VDC)	50 mA
Current consumption (24 VDC)	4 mA / Input used

Table A- 116 Digital inputs

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Number of inputs	2
Type	Sink (IEC Type 1 sink)
Rated voltage	24 VDC at 4 mA, nominal
Continuous permissible voltage	30 VDC max.
Surge voltage	35 VDC for 0.5 sec.
Logic 1 signal (min.)	15 VDC at 2.5 mA
Logic 0 signal (max.)	5 VDC at 1 mA
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	1
Filter times	Individually selectable on each channel: µs: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8 ms: 0.2, 0.4, 0.8, 1.6, 3.2, 6.4, 12.8
Number of inputs on simultaneously	2
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 300 m normal inputs

Table A- 117 Digital outputs

Technical data	SB Digital 2 x Inputs / 2 x Digital Outputs (DT04)
Number of outputs	2
Output type	Solid state - MOSFET (sourcing)
Voltage range	20.4 to 28.8 VDC
Logic 1 signal at max. current	20 VDC min.
Logic 0 signal at max. current	0.1 VDC max.
Rated current per point (max.)	0.5 A
Rated current per common (max.)	1 A
Lamp load	5 W
On state contact resistance	0.6 Ω max.
Leakage current per point	10 μA max.
Surge current	5 A for 100 ms max.
Overload protection	No
Isolation (field side to logic)	500 VAC for 1 minute
Isolation groups	1
Inductive clamp voltage	L+ minus 48 V, 1 W dissipation
Switching delay	2 μs max. off to on 10 μs max. on to off
Output behavior in STOP	Last value or substitute value (default value 0)
Number of outputs on simultaneously	2
Cable length (max.), in meters	Shielded: 500 m normal inputs Unshielded: 150 m normal inputs

Technical specifications

A.6 Digital signal boards

Table A- 118 Wiring diagram for the SB DT04 2 Digital Input/2 Digital Output (6ES7 288-5DT04-0AA0)

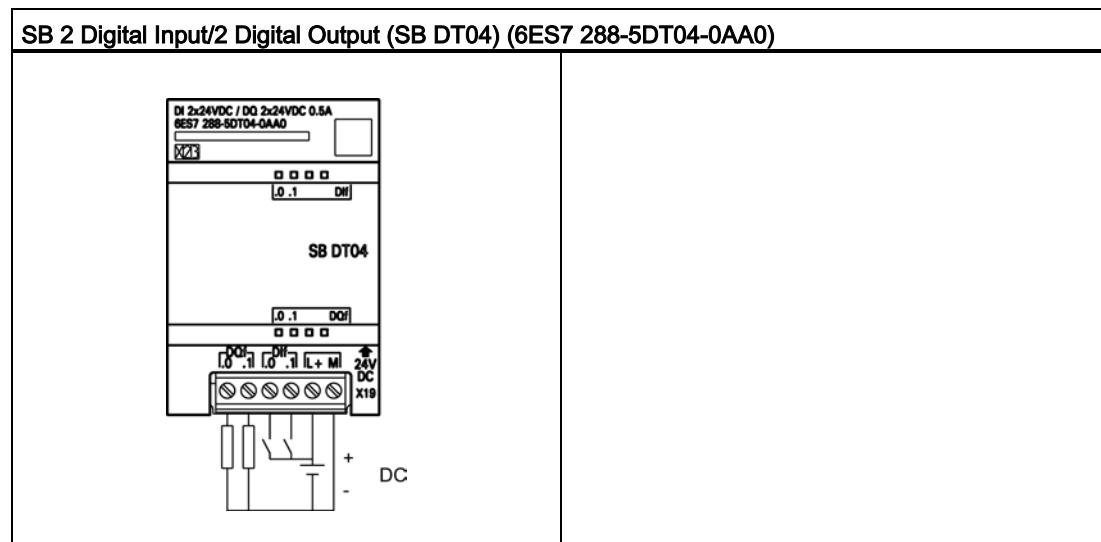


Table A- 119 Connector pin locations for SB DT04 2 Digital Input/2 Digital Output (6ES7 288-5DT04-0AA0)

Pin	X19
1	DQ f.0
2	DQ f.1
3	DI f.0
4	DI f.1
5	L+ / 24 VDC
6	M-/ 24 VDC

A.7 Analog signal boards

A.7.1 SB AQ01 analog output specifications

Table A- 120 General specifications

Technical data	SB Analog 1 x Output (SB AQ01)
Order number	6ES7 288-5AQ01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	17.4 grams
Power dissipation	1.5 W
Current consumption (5 VDC)	15 mA
Current consumption (24 VDC)	40 mA (no load)

Table A- 121 Analog outputs

Technical data	SB Analog 1 x Output (SB AQ01)
Number of outputs	1
Type	Voltage or current
Range	±10 V, 0 to 20 mA
Resolution	Voltage: 11 bits + sign Current: 11 bits
Full scale range (data word)	-27,648 to 27,648 (-10 V to 10 V)
Refer to the output ranges for voltage and current.	0 to 27,648 (0 to 20 mA)
Accuracy (25°C / 0 to 55°C)	±0.5% / ±1%
Settling time (95% of new value)	Voltage: 300 µs (R), 750 µs (1 µF) Current: 600 µs (1mH), 2 ms (10 mH)
Load impedance	Voltage: ≥ 1000 ohms Current: ≤ 600 ohms
Output behavior in STOP	Last value, substitute value (default value 0)
Isolation (field side to logic)	None
Cable length (max.), in meters	10 m twisted and shielded

Technical specifications

A.7 Analog signal boards

Table A- 122 Diagnostics

Technical data	SB Analog 1 x Output (SB AQ01)
Overflow/underflow	Yes
Short to ground (voltage mode only)	Yes
Wire break (current mode only)	Yes

Table A- 123 Wiring diagram for the SB AQ01 SB Analog 1 x Output (6ES7 288-5AQ01-0AA0)

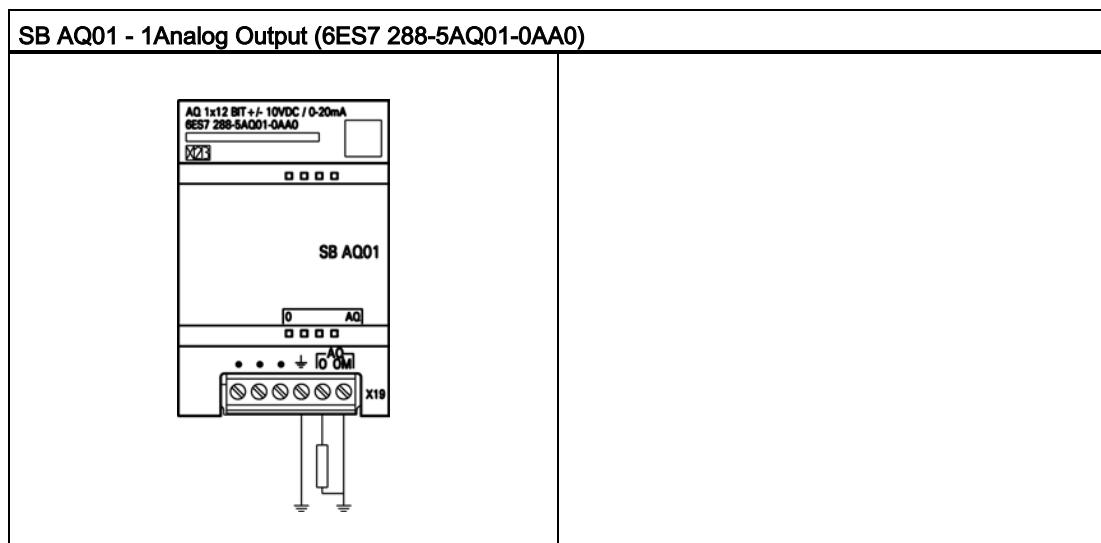


Table A- 124 Connector pin locations for SB AQ01 Analog 1 x Output (6ES7 288-5AQ01-0AA0)

Pin	X19
1	No connection
2	No connection
3	No connection
4	Functional Earth
5	AQ 0
6	AQ 0M

A.8 RS485/RS232 signal boards

A.8.1 SB RS485/RS232 specifications

Table A- 125 General specifications

Technical data	SB RS485/RS232
Order number	6ES7 288-5CM01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	18.2 grams
Power dissipation	0.5 W
Current consumption (5 VDC)	50 mA
Current consumption (24 VDC)	N/A

Table A- 126 RS485 Transmitter and receiver

Technical data	SB RS485/RS232
Common mode voltage range	-7 V to +12 V, 1 second, 3 VRMS continuous
Transmitter differential output voltage	2 V min. at RL = 100 Ω 1.5 V min. at RL = 54 Ω
Termination and bias	4.7 K Ω to +5 V on TxD 4.7 K Ω to GND on RxD
Receiver input impedance	12 K Ω min.
Receiver threshold/sensitivity	+/- 0.2 V min. 60 mV typical hysteresis
Isolation	None
RS 485 signal to chassis ground	
RS485 signal to CPU logic common	
Cable length, shielded	With isolated repeater: 1000 m up to 187.5 kbaud Without isolated repeater: 50 m

Technical specifications

A.8 RS485/RS232 signal boards

Table A- 127 RS232 Transmitter and receiver

Technical data	SB RS485/RS232
Transmitter output voltage	+/- 5 V min. at RL = 3K Ω
Transmit output voltage	+/- 15 VDC max.
Receiver input impedance	3 K Ω min.
Receiver threshold/sensitivity	0.8 V min. low, 2.4 max. high 0.5 V typical hysteresis
Receiver input voltage	+/- 30 VDC max.
Isolation	None
RS232 signal to chassis ground	
RS232 signal to CPU logic common	
Cable length, shielded	10 m max.

Table A- 128 Wiring diagram for the SB CM01 RS485RS232 (6ES7 288-5CM01-0AA0)

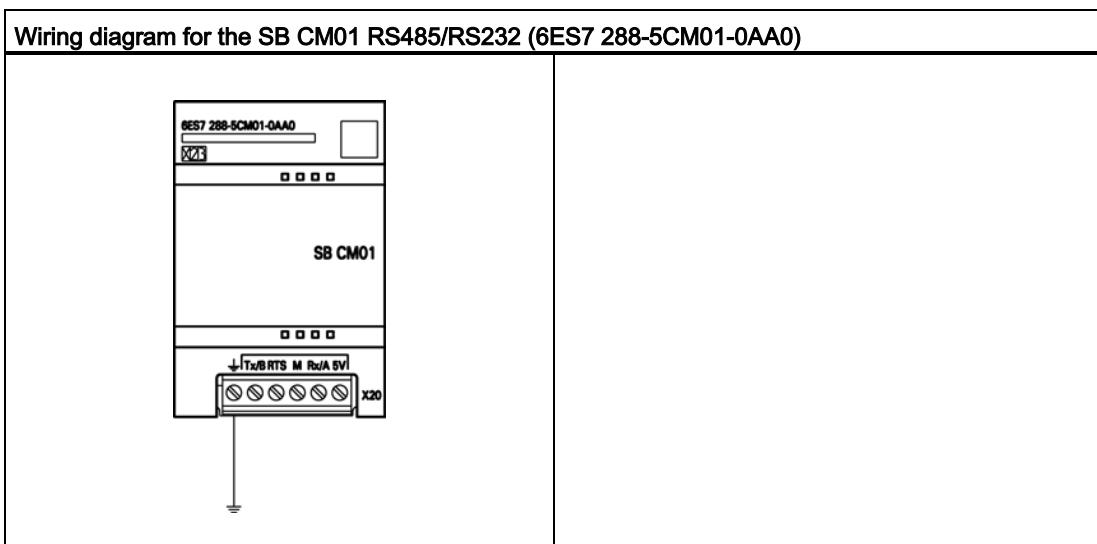


Table A- 129 Connector pin locations for SB CM01 RS485/RS232 (6ES7 288-5CM01-0AA0)

Pin	X20
1	Functional Earth
2	Tx/B
3	RTS
4	M
5	Rx/A
6	5V Out (Bias Voltage)

A.9 **Battery board signal boards (SBs)**

A.9.1 **SB BA01 Battery board**

SB BA01 Battery board

The S7-200 SMART SB BT01 battery board is designed for long-term backup of the Real-time clock. It is pluggable in the signal board slot of the S7-200 SMART CPU (firmware V2.0 and later versions). You must add the SB BT01 to the device configuration and download the hardware configuration to the CPU for the SB BT01 to gain access to the additional battery health reporting options.

The battery (type CR1025) is not included with the SB BT01 and must be purchased by the user.

Note

The SB BT01 is mechanically designed to fit the CPUs with the firmware V2.0 and later versions.

Table A- 130 General specifications

Technical data	SB BA01 Battery Board
Order number	6ES7 288-5BA01-0AA0
Dimensions W x H x D (mm)	35 x 52.2 x 16
Weight	20 grams
Power dissipation	0.6 W
Current consumption (5 VDC)	18 mA
Current consumption (24 VDC)	None

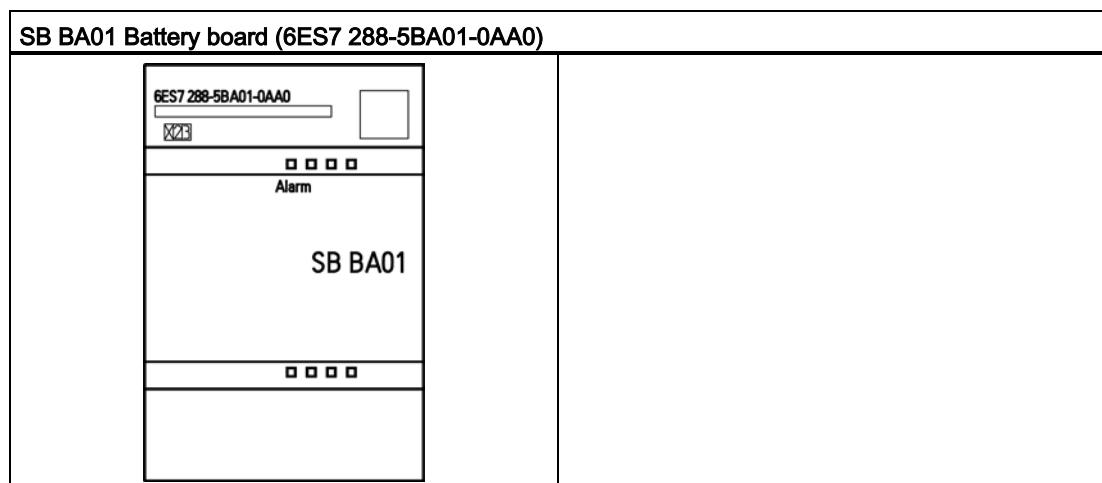
Battery (not included)	SB BA01 Battery Board
Hold up time	Approximately 1 year
Battery type	CR1025 Refer to Installing or replacing a battery in the SB BA01 battery board (Page 43)
Nominal voltage	3 V
Nominal capacity	30 mAH

Technical specifications

A.9 Battery board signal boards (SBs)

Diagnostics	SB BA01 Battery Board
Critical battery level	< 2.5 V
Battery diagnostic	<p>Low voltage indicator:</p> <ul style="list-style-type: none"> • Low battery voltage causes the LED on the BA01 panel to illuminate with the red light continuously ON. • Diagnostic alarm and/or digital output status of battery low condition available
Battery status	<p>Battery status bit provided</p> <p>0 = Battery OK 1 = Battery low</p>
Battery status update	Battery status is updated at power up and then once per day while CPU is in RUN mode.

Table A- 131 Insertion diagram for the SB BA01 Battery board



Calculating a power budget

B.1 Power budget

Your CPU has an internal power supply that provides power for the CPU, the expansion modules, signal boards, and other 24 VDC user power requirements. Use the following information as a guide for determining how much power (or current) the CPU can provide for your configuration.

Refer to the technical specifications for your particular CPU to determine the 24 VDC sensor supply power budget, the 5 VDC logic budget supplied by your CPU and the 5 VDC power requirements of the expansion modules and signal boards. Refer to the Calculating a power budget to determine how much power (or current) the CPU can provide for your configuration.

The CPU provides the 5 VDC logic power needed for any expansion in your system. Pay careful attention to your system configuration to ensure that the CPU can supply the 5 VDC power required by your selected expansion modules. If your configuration requires more power than the CPU can supply, you must remove a module.

Note

If the CPU power budget is exceeded, you may not be able to connect the maximum number of modules allowed for your CPU.

The CPU also provides a 24 VDC sensor supply that can supply 24 VDC for input points, for relay coil power on the expansion modules, or for other requirements. If your power requirements exceed the budget of the sensor supply, then you must add an external 24 VDC power supply to your system. You must manually connect the 24 VDC supply to the input points or relay coils.

If you require an external 24 VDC power supply, ensure that the power supply is not connected in parallel with the sensor supply of the CPU. For improved electrical noise protection, it is recommended that the commons (M) of the different power supplies be connected.



WARNING

Connecting power supplies safely

Connecting an external 24 VDC power supply in parallel with the 24 VDC sensor supply of the CPU can result in a conflict between the two supplies as each seeks to establish its own preferred output voltage level.

The result of this conflict can be shortened lifetime or immediate failure of one or both power supplies, with consequent unpredictable operation of the PLC system. Unpredictable operation could result in death or serious injury to personnel, and/or damage to equipment.

The DC sensor supply of the CPU and any external power supply should provide power to different points. A single connection of the commons is allowed.

Some of the 24 VDC power input ports in the S7-200 SMART system are interconnected, with a common logic circuit connecting multiple M terminals. For example, the following circuits are interconnected when designated as "not isolated" in the data sheets: the 24 VDC power supply of the CPU, the power input for the relay coil of an EM, or the power supply for a non-isolated analog input. All non-isolated M terminals must connect to the same external reference potential.



WARNING

Avoiding unwanted current flow

Connecting non-isolated M terminals to different reference potentials will cause unintended current flows that may cause damage or unpredictable operation in the PLC and any connected equipment.

Failure to comply with these guidelines could cause damage or unpredictable operation which could result in death or severe personal injury and/or property damage.

Always ensure that all non-isolated M terminals in an S7-200 SMART system are connected to the same reference potential.

Refer to the technical specifications for your particular CPU (Page 513) to determine the 24 VDC sensor supply power budget, the 5 VDC logic budget supplied by your CPU and the 5 VDC power requirements of the expansion modules and signal boards.

B.2 Calculating a sample power requirement

Calculating a sample power requirement

The following table shows a sample calculation of the power requirements for a CPU that includes the following:

- CPU SR40 AC/DC/Relay
- 3 each EM Digital Output 8 x Relay (EM DR08)
- 1 each EM Digital 8 x Inputs (EM DI08)

This installation has a total of 32 inputs and 40 outputs.

Note

The CPU has already allocated the power required to drive the CPUs internal relay coils. You do not need to include the internal relay coil power requirements in a power budget calculation.

The CPU in this example provides sufficient 5 VDC current, but does not provide enough 24 VDC current from the sensor supply for all of the inputs and expansion relay coils. The I/O requires 392 mA and the CPU provides 300 mA. This installation requires an additional source of at least 92 mA of 24 VDC power to operate all the included 24 VDC inputs and outputs.

Table B- 1 Calculation of the power budget for a sample configuration

CPU power budget	5 VDC	24 VDC
CPU SR40 AC/DC/Relay	740 mA	300 mA
minus		
System requirements	5 VDC	24 VDC
CPU SR40, 24 inputs		24 * 4 mA = 96 mA
Slot 0: EM DR08	120 mA	8 * 11 mA = 88 mA
Slot 1: EM DR08	120 mA	8 * 11 mA = 88 mA
Slot 2: EM DR08	120 mA	8 * 11 mA = 88 mA
Slot 3: EM DI08	105 mA	8 * 4 mA = 32 mA
Total requirements	465 mA	392 mA
equals		
Current balance	5 VDC	24 VDC
Current balance total	275 mA	(92 mA)

B.3 Calculating your power requirement

Calculating your power requirement

Use the table below to determine how much power (or current) the CPU can provide for your configuration. Refer to the technical specifications (Page 509) for the power budgets of your CPU model and the power requirements of your digital modules, analog modules or signal boards.

Table B- 2 Power budget

Power budget	5 VDC	24 VDC
minus		
System requirements	5 VDC	24 VDC
Total requirements		
equals		
Current balance	5 VDC	24 VDC
Current balance total		

Error codes

C.1 PLC non-fatal error codes

PLC compiler and run-time errors are non-fatal errors. Non-fatal errors can degrade some aspect of the performance of your PLC, but do not render the PLC incapable of executing the user program or updating the I/O.

- **Run-time programming errors** are non-fatal error conditions created by you or your program while the program is being executed. An example of this is an indirect-address pointer, which was valid when the program compiled, modified by program execution to point to an out-of-range address. Access the PLC Information from the PLC menu ribbon strip to determine what type of error has occurred.

You can correct run-time programming errors only by modifying the user program. The run-time programming errors are cleared at the next transition from STOP to RUN mode.

- **PLC compiler errors** (or program-compile errors) prevent you from downloading the program to the PLC. STEP 7-Micro/WIN SMART detects compile errors when you compile or download (Page 33) the program, and shows errors in the output window. If there is a compile error, the PLC retains the current program that is resident in the PLC.

I/O errors are also non-fatal errors. When problems occur with the I/O of the CPU, signal board, and expansion modules, the PLC records the error information in special memory (SM) bits that your program can monitor and evaluate.

Non-fatal error codes

Hexadecimal error code	Non-fatal PLC program compiler errors
0080	The program is too large for the CPU; please reduce the program size
0081	Logic stack underflow; split the network into multiple networks
0082	Illegal instruction; check instruction mnemonics
0083	Illegal instruction before end of main program; remove incorrect instruction
0085	Illegal combination of FOR/NEXT; add FOR instruction or delete NEXT instruction
0086	Illegal combination of FOR/NEXT; add NEXT instruction or delete FOR instruction
0087	Missing label or POU; add the appropriate label
0088	Illegal instruction before end of subroutine; add RET to the end of the subroutine or remove incorrect instruction
0089	Illegal instruction before end of interrupt routine; add RETI to the end of the interrupt routine or remove incorrect instruction
008B	Illegal jump in or out of a SCR segment
008C	Duplicate label or POU name

Error codes

C.1 PLC non-fatal error codes

Hexadecimal error code	Non-fatal PLC program compiler errors
008D	Exceeded maximum label or POU number; ensure that the number of labels allowed has not been exceeded
0090	Illegal operand
0091	Memory range error; check the operand ranges
0092	Illegal count operand; verify the maximum count size
0093	FOR/NEXT nesting level exceeded
0095	Missing LSCR instruction
0096	Missing SCRE instruction or illegal instruction before SCRE
0099	Too many password-protected POUs
009B	Illegal index for a string operation
009D	Illegal parameter detected in system block
009F	Illegal program organization

Hexadecimal error code	Transition to RUN mode prevented (run inhibit conditions)
0070	Run inhibit due to memory card inserted
0071	Run inhibit due to missing configured device
0072	Run inhibit due to mismatched device configuration (note: this error also includes device parameterization errors)
0073	Run inhibit due to attempted firmware update
0074	Run inhibit due to serious HW error on signal module or signal board

Hexadecimal error code	Non-fatal run-time programming problem
0000	No non-fatal errors present
0001	HSC instruction enabled before executing HDEF instruction
0002	Input interrupt point already assigned to an HSC
0003	HSC input point already assigned to an input interrupt or other HSC
0004	Instruction not allowed in an interrupt routine
0005	Simultaneous HSC/PLS/motion instructions
0006	Indirect addressing error
0007	Time of day instructions data error
0008	Maximum user subroutine nesting level exceeded
0009	Simultaneous XMT/RCV instructions on Port 0
000A	Execution of an HDEF instruction of a previously configured HSC
000B	Simultaneous execution of XMT/RCV instructions on Port 1
000F	Illegal numeric value encountered in compare contact instruction
0013	Illegal PID loop table

Hexadecimal error code	Non-fatal run-time programming problem
0014	<p>Data log error:</p> <ul style="list-style-type: none"> • There are too many DATx_WRITE subroutine executions in one program scan. Only 10 to 15 data log writes per second can be sustained. When there are too many DATx-WRITE executions per second, then the allotted memory will be full and for a short period of time no new data log records are stored. • Executing a data log write subroutine without first configuring a data log with the data log wizard
0016	HSC or interrupt input point already assigned to motion
0017	PWM output point already assigned to motion
0019	Signal Board not present or not configured
001A	Scan watchdog timeout.
001B	Attempt to change time base on enabled PWM
001C	Serious hardware error on signal module or signal board
0090	Illegal operand
0091	Operand range error; check the operand ranges
0092	Illegal count operand; verify the maximum count size
0098	Illegal program edit in RUN mode
009A	Attempt to switch into freeport mode inside a user interrupt routine
009B	Illegal index for a string operation (user requested index=0)

See also

Special memory (SM) and system symbol names (Page 593)

C.2 PLC non-fatal error SM flags

Overview

Non-fatal errors are those that may degrade some aspect of PLC performance, but do not render the PLC incapable of executing the user program and updating I/O. To help you in debugging your program, information associated with error conditions is stored in Special Memory (SM) locations (Page 593), which can be accessed by the user program. For example, if you do not want to continue in RUN mode with certain non-fatal error conditions, you can have the user program force a transition to STOP mode when the undesirable condition occurs.

The following table lists and describes the Special Memory non-fatal error information.

SM bit	Non-fatal error description	SM byte	Non-fatal error description
SM0.2	Retentive Data Lost	SMB9	Module 0 I/O Error Byte
SM0.7	RTC_Lost	SMB11	Module 1 I/O Error Byte
SM1.3	Divide by Zero Error	SMB13	Module 2 I/O Error Byte
SM3.0	Parity Error	SMB15	Module 3 I/O Error Byte
SM4.0	Comm. Interrupt Queue Overflow		
SM4.1	Input Interrupt Queue Overflow		
SM4.2	Timed Interrupt Queue Overflow		
SM4.3	Run-Time Programming Problem		
SM5.0	I/O Error (any I/O error bit set)		

C.3 PLC fatal error codes

Overview

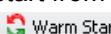
Fatal errors cause the PLC to stop the execution of your program. Depending on the severity of the error, a fatal error can render the PLC incapable of performing any or all functions. The objective for handling fatal errors is to bring the PLC to a safe state from which the PLC can respond to interrogations about the existing error conditions.

The PLC performs the following tasks when a fatal error is detected.

- Changes to STOP mode
- Turns on both the System Fault LED and the STOP LED
- Turns off the outputs

The PLC remains in this condition until the fatal error is corrected. The table shown below provides a list with descriptions for the fatal error codes that can be read from the PLC.

STEP 7-Micro/WIN SMART displays the error codes generated by the PLC, along with a brief description, in the PLC Information dialog. To access PLC information, click the PLC button  from the Information area of the PLC menu ribbon strip.

Once you have corrected the conditions that caused the fatal error, power-cycle the PLC or perform a warm restart from STEP 7-Micro/WIN SMART. To perform a warm start, click the Warm Start button  in the Modify area of the PLC menu ribbon strip.

Restarting the PLC clears the fatal error condition and causes power-up diagnostic testing. If another fatal error condition occurs, the PLC sets the System Fault LED again; otherwise, the PLC begins normal operation.

There are several possible error conditions that can render the PLC incapable of communication, in which case you cannot view the PLC error code. This type of error indicates a hardware failure requiring the PLC module to be repaired; it cannot be fixed by changes to the program or by clearing the PLC memory.

Fatal error codes

Hexadecimal error code	Description
0000	No fatal errors present
0001	System firmware checksum error
0002	Compiled user program checksum error
0004	Permanent memory failed
0005	Permanent memory error on user program
0006	Permanent memory error on system block
0007	Permanent memory error on force data
0009	Permanent memory error on user data, DB1
000A	Memory card failed
000B	Memory card error on user program
000C	Memory card error on system block
000D	Memory card error on force data
000F	Memory card error on user data, DB1
0010	Internal firmware error
0015	User program has compile error on power-up
0016	User data has compile error on power-up
0017	System block has compile error on power-up
0018	CPU HW identification data not available or corrupted
0019	HW watchdog timeout error

C.4 **Timestamp mismatch**

This warning message indicates that the timestamps for the project do not match the timestamps for the program in the PLC. This may indicate that the programs are different, in which case it would be dangerous to continue the current operation. However, the programs might be functionally identical and still have different timestamps.

What actions modify the program timestamps?

Each program contains two distinct timestamps; the "Created" timestamp and the "Last Modified" timestamp. The created timestamp is set when the project is created by the New Project option. The Created timestamp is not affected by any user edits or program compilation.

The Last Modified timestamp is used to indicate when the user last modified the program. There are many conditions that cause the Last Modified timestamp to be set:

1. An edit of instructions or operands in the program block editor.
2. Adding, deleting, or modifying a variable or global symbol.
3. Adding or deleting a POU.
4. Compiling the program block.
5. Downloading the program block (this automatically compiles the program block and therefore sets the last modified timestamp).

Note that although all of these actions will cause the last modified timestamp to be set, this does not necessarily mean that the programs are different. For this reason, STEP 7-Micro/WIN SMART provides the "Compare" option, to allow you to determine whether the programs are really different

How do I tell if the programs are really different?

You can choose to compare the program block in the PLC with the project's program block by clicking the "Compare" button. The results of this comparison allow you to determine whether to continue the status operation.

How do I synchronize timestamps?

Downloading a new project to the PLC synchronizes the timestamps, which allows you to run status.

Special memory (SM) and system symbol names

D.1 SM (Special Memory) overview

SMB0 to SMB29, SMB480 to SMB515, and SMB1000 to SMB1399 (S7-200 SMART read-only special memory)



The CPU Operating System writes new changes to the system data stored in Special Memory.

Read system status from the CPU



SMB0 to SMB29, SMB480 to SMB515, and SMB1000 to SMB1399 are read-only from your program. If a program attempts a write to a read-only SM address, STEP 7-Micro/WIN SMART will compile the program without error. The CPU program compiler, however, will reject the program and display "Operand range error, Download failed".

Your program can read data stored in special memory addresses, evaluate the current system status, and use conditional logic to decide how to respond. In run mode, the continuous scanning of your program logic provides continuous monitoring of system data.

- SMB0 (Page 595) System status bits
- SMB1 (Page 596) Instruction execution status bits
- SMB2 (Page 597) Freeport receive character
- SMB3 (Page 597) Freeport parity error
- SMB4 (Page 598) Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and value forced
- SMB5 (Page 598) I/O error status bits
- SMB6-SMB7 (Page 599) CPU ID, error status, and digital I/O points
- SMB8-SMB19 (Page 599) I/O module ID and errors
- SMW22-SMW26 (Page 600) Scan times
- SMB28-SMB29 (Page 601) Signal board ID and errors
- SMB480-SMB515 (Page 612) Data log status (read only)
- SMB1000-SMB1049 (Page 615) CPU hardware/firmware ID
- SMB1050-SMB1099 (Page 615) SB (signal board) hardware/firmware ID
- SMB1100-1399 (Page 616) EM (expansion module) hardware/firmware ID

SMB30 to SMB194 and SMB566 to SMB749 (S7-200 SMART read/write special memory)



As required, the S7-200 CPU Operating System reads configuration/control data from special memory and writes new changes to the system data stored in Special Memory.

Read system status Your program can read and write all SM addresses in this range, but the normal usage of SM data varies according to the function of each address.
Write (send) control commands to the CPU



SM addresses provide a means to access system status data, configure system options, and control system functions. In run mode, continuous scanning of your program provides for continuous access to special system features.

- SMB30 (port 0) and SMB130 (port 1) (Page 601) Port configuration for the integrated RS485 port (Port 0) and the CM01 Signal Board (SB) RS232/RS485 port (Port 1)
- SMB34-SMB35 (Page 602) Time intervals for timed interrupts
- SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3) (Page 603) High-speed counter configuration and operation
- SMB66-SMB85 (Page 606) PWM0 and PWM1 high-speed outputs
- SMB86-SMB94 and SMB186-SMB194 (Page 608) Receive message control
- SMW98 (Page 610) I/O expansion bus communication errors
- SMW100-SMW114 (Page 611) System alarms
- SMB130 (Page 601) Port configuration for the CM01 Signal Board (SB) RS232/RS485 port (Port 1) (See SMB30)
- SMB136-SMB145 (Page 612) HSC3 High-speed counter (See SMB36)
- SMB186-SMB194 (Page 608) Receive message control (See SMB86-SMB94)
- SMB566-SMB575 (Page 613) PWM2 high-speed output
- SMB600-SMB649 (Page 613) Axis 0 open loop motion control
- SMB650-SMB699 (Page 615) Axis 1 open loop motion control
- SMB700-SMB749 (Page 615) Axis 2 open loop motion control

 **WARNING**
Risks with STEP 7-Micro/WIN Version 4.0 or greater (.mwp files) with absolute special memory (SM) addressing

If an earlier version of STEP 7-Micro/WIN (.mwp file) uses symbolic SM addressing in the OB, and the System Symbols table is generated, the symbols will map properly to the new addresses. However, if the .mwp file uses absolute SM addressing in the OB, those absolute SM addresses will not map to the new SM addresses.

This incorrect mapping of SM addresses can result in unexpected machine or process operation, which may cause death or serious injury to personnel, and/or damage to equipment.

Delete the "S7-200 Symbols" table and generate a SMART "System Symbols" table. The symbols in the OB will map to the new SM address scheme in the SMART System Symbols table..

D.2 SMB0: System status

Special Memory Byte 0 (SM0.0 - SM0.7) provides eight bits that are updated by the S7-200 SMART CPU at the end of each scan cycle. Your program can read the status of these bits and make decisions based on a bit's value.

Table D- 1 SMB0 system status bits

S7-200 SMART symbol name	SM address	Description
Always_On	SM0.0	This bit is always ON. (set to 1)
First_Scan_On	SM0.1	This bit is set ON, for the first scan cycle, and is set OFF thereafter. One use for this bit is to call an initialization subroutine.
Retentive_Lost	SM0.2	<p>This bit is turned ON for one scan cycle after:</p> <ul style="list-style-type: none"> • Reset to factory communication command • Reset to factory memory card evaluation • Evaluation of program transfer card in which a new system block was loaded from the card. • Problem with retentive record on NAND flash <p>This bit can be used as either an error memory bit or as a mechanism to invoke a special start-up sequence.</p>
RUN_Power_Up	SM0.3	This bit is turned ON for one scan cycle when RUN mode is entered from a power-up or warm restart condition. This bit can be used to provide machine warm-up time before starting an operation.
Clock_60s	SM0.4	This bit provides a clock pulse that is OFF for 30 seconds and ON for 30 seconds, for a cycle time of 1-minute. It provides an easy-to-use delay or a 1-minute clock pulse.
Clock_1s	SM0.5	This bit provides a clock pulse that is OFF for 0.5 seconds and then ON for 0.5 seconds for a cycle time of 1 second. It provides an easy-to-use delay or a 1-second clock pulse.

S7-200 SMART symbol name	SM address	Description
Clock_Scan	SM0.6	This bit is a scan cycle clock that is ON for one scan and then OFF for the next scan, alternating ON and OFF on subsequent scans. This bit can be used as a scan counter input.
RTC_Lost	SM0.7	This bit is turned ON for one scan cycle, if the time on the real time clock device was reset or lost at power-up (resulting in system time lost). This bit can be used as either an error memory bit, or to invoke a special start-up sequence.

D.3 SMB1: Instruction execution status

Special memory byte 1 (SM1.0 - SM1.7) provides execution status for various instructions, such as table and math operations. These bits are set and reset by instructions at execution time. Your program can read the bit values and then make decisions based on the values.

Table D- 2 SMB1 instruction execution status bits

S7-200 SMART symbol name	SM address	Description
Result_0	SM1.0	This bit is set ON by the execution of certain instructions when the result of the operation is zero.
Overflow_Illegal	SM1.1	This bit is set ON by the execution of certain instructions either when an overflow results or when an illegal number value is detected.
Neg_Result	SM1.2	This bit is set ON when a negative result is produced by a math operation.
Divide_By_0	SM1.3	This error bit is set ON when division by zero is attempted.
Table_Overflow	SM1.4	This bit is set ON by Add to Table (ATT) instruction execution when the referenced data table is full.
Table_Empty	SM1.5	This bit is set ON when either LIFO or FIFO instructions attempt to read from an empty table.
Not_BCD	SM1.6	This bit is set ON for an illegal value (non-BCD) in a BCD to binary conversion.
Not_Hex	SM1.7	This bit is set ON for an illegal value (non-hex ASCII digit) during ASCII to Hex (ATH) conversion.

D.4 SMB2: Freeport receive character

Special memory byte 2 is the Freeport receive character buffer. Each character that is received while in Freeport mode is placed in this location for easy access by your program.

Table D- 3 SMB2 Freeport received character

S7-200 SMART symbol name	SM address	Description
Receive_Char	SMB2	This byte contains each character received from Port 0 or Port 1 during Freeport communication.

Note

SMB2 and SMB3 are shared between port 0 and port 1

When the reception of a character on port 0 results in the execution of the interrupt routine attached to that event (interrupt event 8), SMB2 contains the character received on port 0, and SMB3 contains the parity status of that character.

When the reception of a character on port 1 results in the execution of the interrupt routine attached to that event (interrupt event 25), SMB2 contains the character received on port 1, and SMB3 contains the error status of that character.

D.5 SMB3: Freeport character error

SMB3 is used for Freeport mode and contains an error bit that is set when a parity, framing, break, or overrun error is detected on a received character. SM3.0 is set ON when a parity, framing, break, or overrun error is detected. Use this bit to discard the message.

Table D- 4 SMB3 Freeport character error

S7-200 SMART symbol name	SM address	Description
Parity_Err	SM3.0	This bit indicates a parity, framing, break, or overrun error received from Port 0 or Port 1. (0=no error; 1=error)

Special memory (SM) and system symbol names

D.6 SMB4: Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and value forced

SMB4: Interrupt queue overflow, run-time program error, interrupts enabled, freeport transmitter idle, and value forced

Special memory byte 4 (SM4.0 - SM4.7) contains the interrupt queue overflow bits and a bit (SM 4.4) that shows whether interrupts are enabled or disabled. These bits indicate either that interrupts are occurring at a rate greater than can be processed or that interrupts were disabled with the global interrupt disable instruction.

Other bits indicate:

- A run-time program error
- Freeport transmitter status
- If any PLC memory value is currently forced.

Table D- 5 SMB4 system status

S7-200 SMART symbol name	SM address	Description
Comm_Int_Ovr	** SM4.0	1 = Communication interrupt queue has overflowed.
Input_Int_Ovr	** SM4.1	1 = Input interrupt queue has overflowed.
Timed_Int_Ovr	** SM4.2	1 = Timed interrupt queue has overflowed.
RUN_Err	SM4.3	1 = Run-time programming non-fatal error is detected.
Int_Enable	SM4.4	1 = Interrupts are enabled.
Xmit0_Idle	SM4.5	1 = Port 0 transmitter is idle (0=transmission in progress).
Xmit1_Idle	SM4.6	1 = Port 1 transmitter is idle (0=transmission in progress).
Force_On	SM4.7	1 = Any memory location is forced.

** Use status bits 4.0, 4.1, and 4.2 only inside an interrupt routine. These status bits are reset when the queue is emptied, and control is returned to the main program.

SMB5: I/O error status

Special Memory Byte 5 (SM5.0 - SM5.7) contains status bits which indicate error conditions that were detected in the I/O system. These bits provide an overview of the I/O errors detected.

Table D- 6 SMB5 I/O error status

S7-200 SMART symbol name	SM address	Description
IO_Err	SM5.0	This bit is set ON if any I/O errors are present.

D.8 SMB6-SMB7: CPU ID, error status, and digital I/O points

Special memory bytes 6 and 7 provide CPU information.

S7-200 SMART symbol name	SM address	Read-only SMB6 and SMB7 (CPU ID, error status, and digital I/O points)							
CPU_ID	SMB6	MSB				LSB			
		7					0		
		1	x	x	x	c	d	0	0
	SM6.4 – SM6.6	0	0	0	= reserved				
		0	0	1	= CPU CR40				
		0	1	0	= CPU CR60				
		0	1	1	= CPU SR20 / ST20				
		1	0	0	= CPU SR40 / ST40				
		1	0	1	= CPU SR60 / ST60				
		1	1	0	= reserved				
		1	1	1	= CPU SR30 / ST30				
					c				Configuration / parameterization error 0 = no error, 1 = error
					d				Diagnostic alarm (See SMW100 for alarm codes) 0 = no error, 1 = error
CPU_IO	SMB7	MSB				LSB			
		7					0		
		i	i	i	i	q	q	q	q
	SM7.0 – SM7.7	i	i	i	i				0 - 15 bytes of digital input points
					q	q	q	q	0 - 15 bytes of digital output points

See also SMW100-SMW114 System alarm codes (Page 611)

D.9 SMB8-SMB19: I/O module ID and errors

SMB8 through SMB19 are organized in byte pairs for expansion modules 0 to 5.

The even-numbered byte of each pair is the module-identification register. These bytes identify the module type, the I/O type, and the number of inputs and outputs.

The odd-numbered byte of each pair is the module error register. These bytes provide an indication of any errors detected in the I/O for that module.

Special memory (SM) and system symbol names

D.10 SMW22-SMW26: Scan times

S7-200 SMART symbol name	SM address	Read-only SMB8-SMB21 module ID and error data											
EM0_ID	SMB8	Expansion module 0 ID register											
EM0_Err	SMB9	Expansion module 0 error register (See SMW104 for diagnostic alarm code)											
EM1_ID	SMB10	Expansion module 1 ID register											
EM1_Err	SMB11	Expansion module 1 error register (See SMW106 for diagnostic alarm code)											
EM2_ID	SMB12	Expansion module 2 ID register											
EM2_Err	SMB13	Expansion module 2 error register (See SMW108 for diagnostic alarm code)											
EM3_ID	SMB14	Expansion module 3 ID register											
EM3_Err	SMB15	Expansion module 3 error register (See SMW110 for diagnostic alarm code)											
EM4_ID	SMB16	Expansion module 4 ID register											
EM4_Err	SMB17	Expansion module 4 error register (See SMW112 for diagnostic alarm code)											
EM5_ID	SMB18	Expansion module 5 ID register											
EM5_Err	SMB19	Expansion module 5 error register (See SMW114 for diagnostic alarm code)											

	Even numbered byte-I/O module ID register								Odd numbered byte-I/O module error register								
	MSB				LSB				MSB				LSB				
	7						0		7					0			
	m	0	0	a	i	i	q	q	c	d	0	b	0	0			
m: Module present	0	= Present								c:	0	No error					
	1	= Not present									1	Configuration / parameterization error					
									d:	0	No error						
										1	Diagnostic alarm						
a: I/O type	0	= Digital								b:	0	No error					
	1	= Analog									1	Bus access error					
ii: Inputs	0	0	= No inputs														
	0	1	= 2 AI or 8 DI														
	1	0	= 4 AI or 16 DI														
	1	1	= 8 AI or 32 DI														
qq: Outputs	0	0	= No outputs														
	0	1	= 2 AQ or 8 DQ														
	1	0	= 4 AQ or 16 DQ														
	1	1	= 8 AQ or 32 DQ														
									m:	0	OK						
										1	Configured module missing						

See also SMW100-SMW114 System alarm codes (Page 611)

D.10 SMW22-SMW26: Scan times

SMW22, SMW24, and SMW26 contain information on the scan time. You can read the last scan time, minimum scan time, and maximum scan time (millisecond values).

Table D- 7 SMW22-SMW26 PLC scan times

S7-200 SMART symbol name	SM address	Description
Last_Scan	SMW22	Scan time of the last scan.
Minimum_Scan	SMW24	Minimum scan time recorded since entering the RUN mode or since resetting these values from the PLC Information dialog.
Maximum_Scan	SMW26	Maximum scan time recorded since entering the RUN mode or since resetting these values from the PLC Information dialog.

D.11 SMB28-SMB29: Signal board ID and errors

SMB28-SMB29 byte addresses store the signal board type and error status.

SMB28-SMB29

S7-200 SMART symbol name	SM address	Read-only SMB28-SMB29 signal board ID and error data																	
SB_ID	SMB28	Signal board ID register																	
SB_Err	SMB29	Signal board error register (See SMW102 for diagnostic alarm code)																	
		SMB28 signal board ID register							SMB29 signal board error register										
		MSB				LSB			MSB				LSB						
		7					0		7				0						
		m	0	0	a	i	i	q	q			c	d	0	b	0	0	0	m
m: Module present	0 = Present							c: 0 No error											
	1 = Not present							1 Configuration / parameterization error											
									d:		0	No error							
									1 Diagnostic alarm										
a: I/O type			0	= Digital	b:							0	No error						
			1	= Analog		1	Bus access error												
ii: Inputs			0	0	m:							0	OK						
			0	1		1	Configured signal board missing												
			1	0															
			1	1															
qq: Outputs			0	0															
			0	1															
			1	0															
			1	1															

See also SMW100-SMW110 System alarm codes (Page 611)

D.12 SMB30: (port 0) and SMB130: (port 1)

SMB30 configures port 0 (onboard RS485 port).

SMB130 configures port 1 (optional CM01 signal board).

You can read and write to SMB30 and SMB130. These bytes configure the respective communication port for Freeport operation and provide selection of either Freeport or system protocol support.

Special memory (SM) and system symbol names

D.13 SMB34-SMB35: Time intervals for timed interrupts

S7-200 SMART symbol name	SM address		Bit Format	Bit format							
	Port 0	Port 1		MSB				LSB			
P0_Config	SMB30			7					0		
P1_Config		SMB130		p	p	d	b	b	b	m	m
	SM30.6 – SM30.7	SM130.6 – SM130.7	pp:	0	0	= No parity					
				0	1	= Even parity					
				1	0	= No parity					
				1	1	= Odd parity					
	SM30.5	SM130.5	d:	0		= 8 data bits per character					
				1		= 7 data bits per character					
	SM30.2 – SM30.4	SM130.2 – SM130.4	bbb:	0	0	0	= 38,400 bps				
				0	0	1	= 19,200 bps				
				0	1	0	= 9,600 bps				
				0	1	1	= 4,800 bps				
				1	0	0	= 2,400 bps				
				1	0	1	= 1,200 bps				
				1	1	0	= 115,200 bps				
				1	1	1	= 57,600				
P0_Config_0	SM30.0		mm:	0	0		= PPI slave mode				
P1_Config_0		SM130.0		0	1		= Freeport protocol				
	SM30.1	SM130.1		1	0		= Reserved (defaults to PPI slave mode)				
				1	1		= Reserved (defaults to PPI slave mode)				
Note: Bits 2 through 7 are ignored in PPI modes											

D.13 SMB34-SMB35: Time intervals for timed interrupts

Special memory bytes 34 and 35 control the time interval of timed interrupts 0 and 1. You can assign time intervals (in 1-ms increments) from 1 ms to 255 ms. The time-interval value is captured by the CPU at the time that the corresponding timed interrupt event is attached to an interrupt routine. To change the time interval, you must reattach the timed interrupt event to the same interrupt routine or to a different interrupt routine. You can terminate a timed interrupt event by detaching the event.

Table D- 8 SMB34-SMB35 timed interrupt intervals

S7-200 SMART symbol name	SM address	Description
Time_0_Intrvl	SMB34	Timed interrupt 0: Time interval value (in 1 ms increments from 1 ms to 255 ms).
Time_1_Intrvl	SMB35	Timed interrupt 1: Time interval value (in 1 ms increments from 1 ms to 255 ms).

D.14 **SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3): high-speed counters**

These addresses provide high-speed counter configuration and operation, for HSC0, HSC1, HSC2, and HSC3.

Table D- 9 High-speed counter 0 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC0_Status	SMB36	HSC0 counter status Note: Counter status bits are valid only while an interrupt routine triggered by a high-speed counter event is being executed.
	SM36.0–SM36.4	Reserved
HSC0_Status_5	SM36.5	HSC0 current counting direction status bit: 1 = counting up
HSC0_Status_6	SM36.6	HSC0 current value equals preset value status bit: 1 = equal
HSC0_Status_7	SM36.7	HSC0 current value is greater than preset value status bit: 1 = greater than
HSC0_Ctrl	SMB37	HSC0 counter control
HSC0_Reset_Level	SM37.0	Active level control bit for Reset: 0 = Reset is active high, 1 = Reset is active low
	SM37.1	Reserved
HSC0_Rate	SM37.2	HSC0 count rate selection for AB quadrature phase counters: 0 = 4x counting rate; 1= 1x counting rate
HSC0_Dir	SM37.3	HSC0 direction control bit: 1 = count up
HSC0_Dir_Update	SM37.4	HSC0 update direction: 1 = update direction
HSC0_PV_Update	SM37.5	HSC0 update preset value: 1 = write new preset value to HSC0 preset
HSC0_CV_Update	SM37.6	HSC0 update current value: 1 = write new current value to HSC0 current
HSC0_Enable	SM37.7	HSC0 enable bit: 1 = enable
HSC0_CV	SMD38	HSC0 new current value SMD38 is used to set HSC0 current value to any value you choose. To update the current value, write SMD38 with the desired new current value, write SM37.6 to 1, and execute the HSC instruction. The new current value is then written to HSC0's current count register.
HSC0_PV	SMD42	HSC0 new preset value SMD42 is used to set HSC0 preset value to any value you choose. To update the current value, write SMD42 with the desired new current value, write SM37.5 to 1, and execute the HSC instruction. The new preset value is then written to HSC0's preset register.

Special memory (SM) and system symbol names

D.14 SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3): high-speed counters

Table D- 10 High-speed counter 1 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC1_Status	SMB46	HSC1 counter status Note: Counter status bits are valid only while an interrupt routine triggered by a high-speed counter event is being executed
	SM46.0-SM46.4	Reserved
HSC1_Status_5	SM46.5	HSC1 current counting direction status bit: 1 = counting up
HSC1_Status_6	SM46.6	HSC1 current value equals preset value status bit: 1 = equal
HSC1_Status_7	SM46.7	HSC1 current value is greater than preset value status bit: 1 = greater than
HSC1_Ctrl	SMB47	HSC1 control
	SM47.0-SM47.2	Reserved
HSC1_Dir	SM47.3	HSC1 direction control bit: 1 = count up 0 = count down
HSC1_Dir_Update	SM47.4	HSC1 update direction: 1 = update direction
HSC1_PV_Update	SM47.5	HSC1 update preset value: 1 = write new preset value to HSC1 preset
HSC1_CV_Update	SM47.6	HSC1 update current value: 1 = write new current value to HSC1 current
HSC1_Enable	SM47.7	HSC1 enable bit: 1 = enable HSC 0 = disable HSC
HSC1_CV	SMD48	HSC1 new current value SMD48 is used to set HSC1 current value to any value you choose. To update the current value, write SMD48 with the desired new current value, write SM47.6 to 1, and execute the HSC instruction. The new current value is then written to HSC1's current count register.
HSC1_PV	SMD52	HSC1 new present value SMD52 is used to set HSC1 preset value to any value you choose. To update the current value, write SMD52 with the desired new current value, write SM47.5 to 1, and execute the HSC instruction. The new preset value is then written to HSC1's preset register.

Table D- 11 High-speed counter 2 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC2_Status	SMB56	HSC2 counter status Note: Counter Status bits are valid only while an interrupt routine triggered by a high-speed counter event is being executed.
	SM56.0-SM56.4	Reserved
HSC2_Status_5	SM56.5	HSC2 current counting direction status bit: 1 = counting up
HSC2_Status_6	SM56.6	HSC2 current value equals preset value status bit: 1 = equal
HSC2_Status_7	SM56.7	HSC2 current value is greater than preset value status bit: 1 = greater than
HSC2_Ctrl	SMB57	HSC2 control
HSC2_Reset_Level	SM57.0	HSC2 active level control bit for Reset: 0 = Reset is active high; 1 = Reset is active low
	SM57.1	Reserved
HSC2_Rate	SM57.2	HSC2 counting rate selection for AB quadrature phase counters: 0 = 4x counting rate; 1 = 1x counting rate

S7-200 SMART symbol name	SM address	Description
HSC2_Dir	SM57.3	HSC2 direction control bit: 1 = count up
HSC2_Dir_Update	SM57.4	HSC2 update direction: 1 = update direction
HSC2_PV_Update	SM57.5	HSC2 update preset value: 1 = write new preset value to HSC2 preset
HSC2_CV_Update	SM57.6	HSC2 update current value: 1 = write new current value to HSC2 current
HSC2_Enable	SM57.7	HSC2 enable bit: 1 = enable
HSC2_CV	SMD58	HSC2 new current value SMD58 is used to set HSC2 current value to any value you choose. To update the current value, write SMD58 with the desired new current value, write SM57.6 to 1, and execute the HSC instruction. The new current value is then written to HSC2's current count register.
HSC2_PV	SMD62	HSC2 new preset value SMD62 is used to set HSC2 preset value to any value you choose. To update the current value, write SMD62 with the desired new current value, write SM57.5 to 1, and execute the HSC instruction. The new preset value is then written to HSC2's preset register.

Table D- 12 High-speed counter 3 configuration and operation

S7-200 SMART symbol name	SM address	Description
HSC3_Status	SMB136	HSC3 counter status Note: Counter status bits are valid only while an interrupt routine triggered by a high-speed counter event is being executed.
	SM136.0-SM136.4	Reserved
HSC3_Status_5	SM136.5	HSC3 current counting direction status bit: 1 = counting up
HSC3_Status_6	SM136.6	HSC3 current value equals preset value status bit: 1 = equal
HSC3_Status_7	SM136.7	HSC3 current value is greater than preset value status bit: 1 = greater than
HSC3_Ctrl	SMB137	HSC3 counter control
	SM137.0-SM137.2	Reserved
HSC3Dir	SM137.3	HSC3 direction control bit: 1 = count up
HSC3_Dir_Update	SM137.4	HSC3 update direction: 1 = update direction
HSC3_PV_Update	SM137.5	HSC3 update preset value: 1 = write new preset value to HSC3 preset
HSC3_CV_Update	SM137.6	HSC3 update current value: 1 = write new current value to HSC3 current
HSC3_Enable	SM137.7	HSC3 enable bit: 1 = enable

Special memory (SM) and system symbol names

D.15 SMB67-SMB71, SMB77-SMB81, and SMB567-SMB571: PWM0, PWM1, and PWM2 high-speed outputs

S7-200 SMART symbol name	SM address	Description
HSC3_CV	SMD138	HSC3 new current value SMD138 is used to set HSC3 current value to any value you choose. To update the current value, write SMD138 with the desired new current value, write SM137.6 to 1, and execute the HSC instruction. The new current value is then written to HSC3's current count register.
HSC3_PV	SMD142	HSC3 new preset value SMD142 is used to set HSC3 preset value to any value you choose. To update the current value, write SMD142 with the desired new current value, write SM137.5 to 1, and execute the HSC instruction. The new preset value is then written to HSC3's preset register.

D.15 SMB67-SMB71, SMB77-SMB81, and SMB567-SMB571: PWM0, PWM1, and PWM2 high-speed outputs

SMB67 through SMB81 are used to monitor and control the pulse width modulation outputs (PWM0 and 1), for the PLS (Pulse) instruction.

SMB567 through SMB571 are used to monitor and control pulse-width modulation output for PWM2.

Table D- 13 High-speed output 0 configuration and control

S7-200 SMART symbol name	SM address	Function
PLS0_Ctrl	SMB67	Control pulse width modulation for Q0.0
PLS0_Cycle_Update	SM67.0	PWM0 update cycle time value: 1=write new cycle time
PWM0_PW_Update	SM67.1	PWM0 update pulse width value: 1=write new pulse width
	SM67.2	Reserved
PLS0_TimeBase	SM67.3	PWM0 time base: 0=1 µs/tick, 1=1 ms/tick
	SM67.4-SM67.6	Reserved
PLS0_Enable	SM67.7	PWM0 enable bit: 1=enable
PLS0_Cycle	SMW68	Cycle time value, pulse width modulation output 2 Word data: PWM0 cycle time value (2 to 65535 units of the time base)
PWM0_PW	SMW70	Pulse width value of pulse width modulation output 2 Word data: PWM0 pulse width value (0 to 65535 units of the time base)

Table D- 14 High-speed output 1 configuration and control

S7-200 SMART symbol name	SM address	Function
PLS1_Ctrl	SMB77	Control pulse width modulation for Q0.1
PLS1_Cycle_Update	SM77.0	PWM1 update cycle time value: 1=write new cycle time
PWM1_PW_Update	SM77.1	PWM1 update pulse width value: 1=write new pulse width
	SM77.2	Reserved
PLS1_TimeBase	SM77.3	PWM1 time base: 0=1 µs/tick, 1=1 ms/tick
	SM77.4-SM77.6	Reserved
PLS1_Enable	SM77.7	PWM1 enable bit: 1=enable
PLS1_Cycle	SMW78	Cycle time value, pulse width modulation output 1 Word data: PWM1 cycle time value (2 to 65535 units of the time base)
PWM1_PW	SMW80	Pulse width value of pulse width modulation output 1 Word data: PWM1 pulse width value (0 to 65535 units of the time base)

Table D- 15 High-speed output 2 configuration and control

S7-200 SMART symbol name	SM address	Description
PLS2_Ctrl	SMB567	Control pulse width modulation for Q0.3
PLS2_Cycle_Update	SM567.0	PWM2 update cycle time value: 1=write new cycle time
PWM2_PW_Update	SM567.1	PWM2 update pulse width value: 1=write new pulse width
	SM567.2	Reserved
PLS2_TimeBase	SM567.3	PWM2 time base: 0=1 µs/tick, 1=1 ms/tick
	SM567.4-SM567.6	Reserved
PLS2_Enable	SM567.7	PWM2 enable bit: 1 = enable
PLS2_Cycle	SMW568	Cycle time value, pulse width modulation output 2 Word data: PWM2 cycle time value (2 to 65535 units of the time base)
PWM2_PW	SMW570	Pulse width value of pulse width modulation output 2 Word data: PWM2 pulse width value (0 to 65535 units of the time base)

D.16 SMB86-SMB94 and SMB186-SMB194: Receive message control

SMB86-SMB94 and SMB186-SMB194 are used to control and read status of the RCV (Receive message) instruction.

S7-200 SMART symbol name	SM address		Bit Format	Receive message status byte									
	Port 0	Port 1		MSB				LSB					
				7	n	r	e	0	0	t	c	p	
P0_Stat_Rcv	SMB86												
P1_Stat_Rcv		SMB186											
P0_Stat_Rcv_7	SM86.7		n:	1	= Receive message was terminated by user disable command, else n = 0								
P1_Stat_Rcv_7		SM186.7											
P0_Stat_Rcv_6	SM86.6		r:	1	= Receive message terminated: (No start condition defined, character count of 0, or execution of receive message with transmit active) else, r = 0								
P1_Stat_Rcv_6		SM186.6											
P0_Stat_Rcv_5	SM86.5		e:	1	= End character received else, e = 0								
P1_Stat_Rcv_5		SM186.5											
P0_Stat_Rcv_2	SM86.2		t:		1	= Receive message terminated: timer expired else, t = 0							
P1_Stat_Rcv_2		SM186.2											
P0_Stat_Rcv_1	SM86.1		c:		1	= Receive message terminated: maximum character count reached else, c = 0							
P1_Stat_Rcv_1		SM186.1											
P0_Stat_Rcv_0	SM86.0		p:		1	= Receive message was terminated because of a parity, framing, break, or overrun error else, p = 0							
P1_Stat_Rcv_0		SM186.0											

			Bit Format	Receive message control byte								
				MSB				LSB				
	Port 0	Port 1		7	en	sc	ec	il	c/m	Tmr	bk	0
P0_Ctrl_Rcv	SMB87											
P1_Ctrl_Rcv		SMB187										
P0_Ctrl_Rcv_7	SM87.7		en:	0	= Receive message function is disabled							
P1_Ctrl_Rcv_7		SM187.7		1	= Receive message function is enabled							
P0_Ctrl_Rcv_6	SM87.6		sc:	0	= Ignore SMB88 or SMB188							
P1_Ctrl_Rcv_6		SM187.6		1	= Use the value of SMB88 or SMB188 to detect start of message							
P0_Ctrl_Rcv_5	SM87.5		ec:	0	= Ignore SMB89 or SMB189							
P1_Ctrl_Rcv_5		SM187.5		1	= Use the value of SMB89 or SMB189 to detect end of message							
P0_Ctrl_Rcv_4	SM87.4		il:	0	= Ignore SMW90 or SMW190							
P1_Ctrl_Rcv_4		SM187.4		1	= Use the value of SMW90 or SMW190 to detect an idle line condition							
P0_Ctrl_Rcv_3	SM87.3		c/m:	0	= Timer is an inter-character timer							
P1_Ctrl_Rcv_3		SM187.3		1	= Timer is a message timer							
P0_Ctrl_Rcv_2	SM87.2		tmr:	0	= Ignore SMW92 or SMW192							
P1_Ctrl_Rcv_2		SM187.2		1	= Terminate receive if the time period in SMW92 or SMW192 is exceeded							
P0_Ctrl_Rcv_1	SM87.1		bk:	0	= Ignore break conditions							
P1_Ctrl_Rcv_1		SM187.1		1	= Use break condition as start of message detection							

The bits of the message control byte are used to define the criteria by which the message is identified. The start of message and end of message criteria are defined. To determine the start of a message, either of two sets of logically ANDed start of message criteria must be true and must occur in sequence (in sequence means idle line followed by start character or break followed by start character). To determine the end of a message, the enabled end of message criteria is logically ORed. The equations for start and stop criteria are given below:

$$\text{Start of Message} = (\text{il AND sc}) \text{ OR } (\text{bk AND sc})$$

$$\text{End of Message} = \text{ec OR tmr OR maximum character count reached}$$

S7-200 SMART symbol name	SM address		Bit Format	Programming the start of a message criteria							
				MSB				LSB			
	Port 0	Port 1	7						0		
P0_Ctrl_Rcv	SMB87		en	sc	ec	il	c/m	Tmr	bk	0	
P1_Ctrl_Rcv		SMB187									
Idle line detection			0		1				0		SMW90 > 0
Start character detection			1		0				0		SMW90 = don't care
Break detection			0		0				1		SMW90 = don't care
Any response to a request			0		1				0		SMW90 = 0 (Message timer can be used to terminate receive, if there is no response)
Break and a start character			1		0				1		SMW90 = don't care
Idle line and start character			1		1				0		SMW90 > 0
Idle line and start character (illegal)			1		1				0		SMW90 = 0
Note: Receive will automatically be terminated by a parity, framing, overrun or break error.											

S7-200 SMART symbol name	SM address											
			Port 0	Port 1								
P0_Start_Char	SMB88				Start of message character.							
P1_Start_Char		SMB188										
P0_End_Char	SMB89				End of message character.							
P1_End_Char		SMB189										
P0_Idle_Time	SMW90				Word data: Idle line time period given in milliseconds. The first character received after the idle line time has expired is the start of a new message.							
P1_Idle_Time		SMW190										
P0_Timeout	SMW92				Word data: Inter-character/message timer timeout value given in milliseconds. If the time period is exceeded, the receive message is terminated.							
P1_Timeout		SMW192										
P0_Max_Char	SMB94				Maximum number of characters to be received (1 to 255 bytes).							
P1_Max_Char		SMB194			Note: This range must be set to the expected maximum buffer size, even if the character count message termination is not used.							

D.17 **SMW98: I/O expansion bus communication errors**

SMW98 gives you information about errors on the expansion I/O bus.

Table D- 16 SMW98 I/O expansion bus communication error counter

S7-200 SMART symbol name	SM address (Read/Write)	Description
EM_Parity_Err	SMW98	This word is incremented each time a parity error is detected on the expansion I/O bus. It is cleared upon power up and by the user writing a zero.

D.18 SMW100-SMW114 System alarms

Special memory words SMW100-SMW114 provide alarm and diagnostic error codes for CPU, SB (signal board), and EM (expansion modules).

S7-200 SMART symbol name	SM address								Current diagnostic alarm source																	
CPU_Alarm	SMW100								CPU diagnostic alarm code																	
SB_Alarm	SMW102								Signal board board diagnostic alarm code																	
EM0_Alarm	SMW104								Expansion module bus slot 0 diagnostic alarm code																	
EM1_Alarm	SMW106								Expansion module bus slot 1 diagnostic alarm code																	
EM2_Alarm	SMW108								Expansion module bus slot 2 diagnostic alarm code																	
EM3_Alarm	SMW110								Expansion module bus slot 3 diagnostic alarm code																	
EM4_Alarm	SMW112								Expansion module bus slot 4 diagnostic alarm code																	
EM5_Alarm	SMW114								Expansion module bus slot 5 diagnostic alarm code																	
Alarm code format		MSB								LSB																
		15	14	13				8	7						0											
		d	s	c	c	c	c	c	a	a	a	a	a	a	a	a										
d: Alarm direction	0	Input channel or not applicable																								
	1	Output channel																								
s: Alarm scope	0	On individual channel																								
	1	On entire module																								
c: Channel number		c	c	c	c	c	c	Number of the affected channel, if alarm scope = "On individual channel" or 0, if alarm scope = "On entire module"																		
a: Alarm type																	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0H: No alarm									
																	0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 01H: Short circuit									
																	0 0 0 0 0 0 x x x x 0 2H to 05H: Reserved									
																	0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 06H: Wire break									
																	0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 07H: Upper limit exceeded									
																	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 08H: Lower limit exceeded									
																	0 0 0 0 0 x x x x x x 0 0 0 0 0 09H to 0FH: Reserved									
																	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 10H: Paramaterization error									
																	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 11H: Sensor or load voltage missing									
																	0 0 0 x x x x x x 0 0 0 0 0 0 12H to 1FH: Reserved									
																	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 20H: Internal error (MID problem)									
																	0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 21H: Internal error (IID problem)									
																	0 0 1 0 0 0 0 1 0 0 22H: Reserved									
																	0 0 1 0 0 0 0 1 1 1 0 1 1 1 0 23H: Configuration error									
																	0 0 1 0 0 0 1 0 0 0 24H: Reserved									
																	0 0 1 0 0 0 1 0 1 0 25H: Bad or missing firmware									
																	0 0 1 0 x x x x x x 0 0 0 0 0 0 26H to 2AH: Reserved									
																	0 0 1 0 1 0 0 1 0 1 1 1 0 1 1 2BH: Battery voltage low									
																	x x x x x x x x x x 0 0 0 0 0 0 2CH to FFH: Reserved									

D.19 SMB130: Freeport control for port 1 (See SMB30)

Refer to "SMB30: (port 0) and SMB130: (port 1)" (Page 601) for details.

D.20 SMB136-SMB145: HSC3 high-speed counter

Refer to "SMB36-45 (HSC0), SMB46-55 (HSC1), SMB56-65 (HSC2), SMB136-145 (HSC3): high-speed counters" (Page 603) for details.

D.21 SMB186-SMB194: Receive message control (See SMB86-SMB94)

Refer to "SMB86-SMB94 and SMB186-SMB194" (Page 608) for details.

D.22 SMB480-SMB515: Data log status

SMB480 through SMB515 are read-only special memory addresses that are used to monitor the status of Data log operations.

S7-200 SMART symbol name	SM address	Function
DL0_InitResult	SMB480	Initialization result code for Data Log 0: The data log analysis is performed after power-up and after System block download. <ul style="list-style-type: none">• 00H: data log OK• 01H: initialization in progress• 02H: data log file not found• 03H: data log initialization error• 04H to FEH: reserved• FFH: data log not configured
DL1_InitResult	SMB481	Initialization result code for Data Log 1 (See SMB 480 for result codes)
DL2_InitResult	SMB482	Initialization result code for Data Log 2 (See SMB 480 for result codes)
DL3_InitResult	SMB483	Initialization result code for Data Log 3 (See SMB 480 for result codes)
DL0_Maximum	SMW500	Data log 0: Configured maximum allowed number of records
DL0_Current	SMW502	Data log 0: Actual maximum allowed number of records
DL1_Maximum	SMW504	Data log 1: Configured maximum allowed number of records
DL1_Current	SMW506	Data log 1: Actual maximum allowed number of records
DL2_Maximum	SMW508	Data log 2: Configured maximum allowed number of records
DL2_Current	SMW510	Data log 2: Actual maximum allowed number of records
DL3_Maximum	SMW512	Data log 3: Configured maximum allowed number of records
DL3_Current	SMW514	Data log 3: Actual maximum allowed number of records

D.23 **SMB567-SMB571: PWM2 high-speed output (See SMB67-SMB71)**

Refer to "SMB66-SMB71, SMB77-SMB81, and SMB567-SMB570: PWM0, PWM1, and PWM2 high-speed outputs" (Page 606) for details.

D.24 **SMB600-SMB749: Axis (0, 1, and 2) open loop motion control**

Axis configuration and control SM addresses

This axis special memory data is normally read and written by wizard generated program code.

Axis data SM address			Axis function
Axis 0	Axis 1	Axis 2	
SMB600-SMB615	SMB650-SMB665	SMB600-SMB615	Axis name (16 ASCII characters). The first character is the lowest numbered byte in the sequence.
SMB616-SMB619	SMB616-SMB619	SMB616-SMB619	Reserved
SMW620	SMW670	SMW720	Error code - See Axis of Motion error codes (Page 489)

Special memory (SM) and system symbol names

D.24 SMB600-SMB749: Axis (0, 1, and 2) open loop motion control

Axis 0 SMB622	Axis 1 SMB672	Axis 2 SMB722		MSB								LSB						
				7	6	5	4	3	2	1	0							
DIS:		0	No current flow															
SM622.7	SM672.7	SM722.7	DIS: disable outputs	1	Current flow													
					0	Not active												
SM622.5	SM672.5	SM722.5	TRIG: Trigger input	0	Active													
				1	Active													
SM622.4	SM672.4	SM722.4	STP : Stop input	0	Not active													
				1	Active													
SM622.3	SM672.3	SM722.3	LMT - : Negative travel limit input	0	Not active													
				1	Active													
SM622.2	SM672.2	SM722.2	LMT + : Positive travel limit input	0	Not active													
				1	Active													
SM622.1	SM672.1	SM722.1	RPS : Reference point switch input	0	Not active													
				1	Active													
SM622.0	SM672.0	SM722.0	ZP : Zero pulse input	0	Not active													
				1	Active													

Axis 0 SMB623	Axis 1 SMB673	Axis 2 SMB723		MSB								LSB								
				7	6	5	4	3	2	1	0									
OR: Target speed out of range		0	In range																	
SM622.1	SM672.1	SM722.1	R: Direction of rotation	0	Positive rotation															
				1	Negative rotation															
SM622.0	SM672.0	SM722.0	CFG: Axis configured	0	Not configured															
				1	Configured															

Axis data SM address			Axis function											
Axis 0	Axis 1	Axis 2												
SMB624	SMB674	SMB724	CUR_PF is a byte that indicates the profile currently being executed.											
SMB625	SMB675	SMB725	CUR_STP is a byte that indicates the step currently being executed in the profile.											
SMD626	SMD676	SMD726	CUR_POS is a double-word value that indicates the current position of the axis.											
SMD630	SMD680	SMD730	CUR_SPD is a double-word value that indicates the current speed of the axis.											

Axis 0 SMB634	Axis 1 SMB684	Axis 2 SMB734		MSB								LSB								
				7	6	5	4	3	2	1	0									
D: Done bit		0	Error:																	
SM634.7	SM684.7	SM734.7	D: Done bit	1	Operation complete (set by the axis during initialization)															
				1	Operation complete (set by the axis during initialization)															

Axis data SM address			Axis function											
Axis 0	Axis 1	Axis 2												
SMB635-SMB645	SMB685-SMB695	SMB735-SMB745	Reserved											
SMD646	SMD646	SMD746	V memory pointer to the configuration/profile table for the axis. A pointer value to an area other than V memory will not be accepted.											

D.25 SMB650-SMB699: Axis 1 open loop motion control (See SMB600-SMB740)

Refer to "SMB600-SMB749: Axis (0, 1, and 2) open loop motion control (Page 613)" for details.

D.26 SMB700-SMB749: Axis 2 open loop motion control (See SMB600-SMB740)

Refer to "SMB600-SMB749: Axis (0, 1, and 2) open loop motion control (Page 613)" for details.

D.27 SMB1000-SMB1049: CPU hardware/firmware ID

This CPU information is written to special memory after a power-up or warm restart transition. The SMB1000-SMB1049 section of special memory is read-only.

SM address	Description
SMW1000	CPU vendor ID: (always 0x002A)
SMB1002 to SMB1021	CPU order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1022 to SMB1037	CPU serial number: ASCII characters, left-justified in field, padded with spaces
SMW1038	CPU hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFFD (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1040	CPU firmware version: Byte 0 is ASCII 'V'; byte 1=functional version; byte 2=minor change version; byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1044	CPU firmware version counter (range 0x0000 to 0x00FF)
SMW1046	Reserved: Always 0x0000
SMW1048	CPU device type: Always 0x0001

D.28 SMB1050-SMB1099: SB (signal board) hardware/firmware ID

This signal board information is written to special memory after a power-up or warm restart transition. The SMB1050-SMB1099 section of special memory is read-only.

SM address	Description
SMW1050	Signal board vendor ID: Set to 0x002A if a Siemens SB is present and 0x0000 if no SB is present
SMB1052 to SMB1071	Signal board order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1072 to SMB1087	Signal board serial number: ASCII characters, left-justified in field, padded with spaces

Special memory (SM) and system symbol names

D.29 SMB1100-SMB1399: EM (expansion module) hardware/firmware ID

SM address	Description
SMW1088	Signal board hardware version: Represents the hardware E-stand; range=0x0001 to 0xFFFFD, (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1090	Signal board firmware version; Byte 0 is ASCII 'V'; byte 1=functional version; byte 2=minor change version; byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1094	Signal board firmware version counter (range 0x0000 to 0x00FF)
SMW1096	Reserved, always 0x0000
SMW1098	Signal board device type: I/O=0x0003, communications=0x0004, all other values reserved

D.29 SMB1100-SMB1399: EM (expansion module) hardware/firmware ID

Expansion module information is written to special memory after a power-up or warm restart transition. The SMB1100-SMB1399 section of special memory is read-only.

SM addresses for slot 0	Description
SMW1100	EM bus Slot 0 vendor ID: Set to 0x002A if a Siemens EM is present and 0x0000 if no EM is present
SMB1102 to SMB1121	EM bus Slot 0 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1122 to SMB1137	EM bus slot 0 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1138	EM bus slot 0 hardware version: represents the hardware E-stand; range = 0x0001 to 0xFFFFD (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1140	EM bus slot 0 firmware version: Byte 0 is ASCII 'V', byte 1=functional version, byte 2= minor change version, byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1144	EM bus slot 0 firmware version counter (range 0x0000 to 0x00FF)
SMW1146	Reserved, always 0x0000
SMW1148	EM bus slot 0 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 1	Description
SMW1150	EM bus slot 1 vendor ID: Set to 0x002A if a Siemens EM is present and 0x0000 if no EM is present
SMB1152 to SMB1171	EM bus slot 1 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1172 to SMB1187	EM bus slot 1: serial number: ASCII characters, left-justified in field, padded with spaces
SMW1188	EM bus slot 1 hardware version: Represents the hardware E-stand; range=0x0001 to 0xFFFFD (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1190	EM bus slot 1 firmware version: Byte 0 is ASCII 'V'; byte 1=functional version, byte 2=minor change version, byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1194	EM bus slot 1 firmware version counter (range 0x0000 to 0x00FF)
SMW1196	Reserved, always 0x0000
SMW1198	EM bus slot 1 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 2	Description
SMW1200	EM bus slot 2 vendor ID: Set to 0x002A if a Siemens EM is present and 0x0000 if no EM is present
SMB1202 to SMB1221	EM bus slot 2 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1222 to SMB1237	EM bus slot 2 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1238	EM bus slot 2 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFF (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1240	EM bus slot 2 firmware version: Byte 0 is ASCII 'V'; byte 1=functional version, byte 2=minor change version, byte 3 = bug fix version (range of bytes 1-3=0x00 to 0xFF)
SMW1244	EM bus slot 2 firmware version counter (range 0x0000 to 0x0OFF)
SMW1246	Reserved, always 0x0000
SMW1248	EM bus slot 2 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 3	Description
SMW1250	EM bus slot 3 vendor ID: Set to 0x002A if a Siemens EM is present and 0x0000 if no EM is present
SMB1252 to SMB1271	EM bus slot 3 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1272 to SMB1287	EM bus slot 3 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1288	EM bus slot 3 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFF (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1290	EM bus slot 3 firmware version: Byte 0 is ASCII 'V'; byte 1=functional version, byte 2=minor change version, byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1294	EM bus slot 3 firmware version counter (range 0x0000 to 0x0OFF)
SMW1296	Reserved, always 0x0000
SMW1298	EM bus slot 3 device type: I/O=0x0003, communications=0x0004, all other values reserved

SM addresses for slot 4	Description
SMW1300	EM bus slot 4 vendor ID: Set to 0x002A if a Siemens EM is present and 0x0000 if no EM is present
SMB1302 to SMB1321	EM bus slot 4 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1322 to SMB1327	EM bus slot 4 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1338	EM bus slot 4 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFF (0x0000, 0xFFFFE, and 0xFFFF are reserved values)
SMD1340	EM bus slot 4 firmware version: Byte 0 is ASCII 'V'; byte 1=functional version, byte 2=minor change version, byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1344	EM bus slot 4 firmware version counter (range 0x0000 to 0x0OFF)
SMW1346	Reserved, always 0x0000
SMW1348	EM bus slot 4 device type: I/O=0x0003, communications=0x0004, all other values reserved

Special memory (SM) and system symbol names

D.29 SMB1100-SMB1399: EM (expansion module) hardware/firmware ID

SM addresses for slot 5	Description
SMW1350	EM bus slot 5 vendor ID: Set to 0x002A if a Siemens EM is present and 0x0000 if no EM is present
SMB1352 to SMB1371	EM bus slot 5 order ID (MLFB): ASCII characters, left-justified in field, padded with spaces
SMB1372 to SMB1387	EM bus slot 5 serial number: ASCII characters, left-justified in field, padded with spaces
SMW1388	EM bus slot 5 hardware version: Represents the hardware E-stand; range = 0x0001 to 0xFFFFD (0x0000, 0xFFFE, and 0xFFFF are reserved values)
SMD1390	EM bus slot 5 firmware version: Byte 0 is ASCII 'V'; byte 1=functional version, byte 2=minor change version, byte 3=bug fix version (range of bytes 1-3 = 0x00 to 0xFF)
SMW1394	EM bus slot 5 firmware version counter (range 0x0000 to 0x00FF)
SMW1396	Reserved, always 0x0000
SMW1398	EM bus slot 5 device type: I/O=0x0003, communications=0x0004, all other values reserved

References

E.1 Often-used special memory bits

The complete list of pre-defined special memory program symbols is available to your project, in the System Symbol table.

Table E- 1 Often-used special memory bits

SM address	System symbol name	Description
SM0.0	Always_On	Always ON
SM0.1	First_Scan_On	ON for the first scan cycle only
SM0.2	Retentive_Lost	ON for one scan cycle if retentive data is lost
SM0.3	RUN_Power_Up	ON for 1 scan cycle when RUN mode is entered from a power-up condition
SM0.4	Clock_60s	Clock pulse that is ON for 30 s, OFF for 30 s, for a cycle time of 1 min.
SM0.5	Clock_1s	Clock pulse that is ON for 0.5 s, OFF for 0.5 s, for a cycle time of 1 s.
SM0.6	Clock_Scan	Scan cycle clock which is ON for one scan cycle and OFF for the next scan cycle
SM0.7	RTC_Lost	This bit is turned ON for one scan cycle, if the time on the real time clock device was reset or lost at power-up (resulting in system time lost). This bit can be used as either an error memory bit, or to invoke a special start-up sequence.
SM1.0	Result_0	Set to 1 by the execution of certain instructions when the result of the operation = 0
SM1.1	Overflow_Illegal	Set to 1 by the execution of certain instructions on overflow or illegal numeric value
SM1.2	Neg_Result	Set to 1 when a math operation produces a negative result
SM1.3	Divide_By_0	Set to 1 when an attempt is made to divide by zero
SM1.4	Table_Overflow	Set to 1 when the Add to Table instruction attempts to overfill the table
SM1.5	Table_Empty	Set to 1 when a LIFO or FIFO instruction attempts to read from an empty table
SM1.6	Not_BCD	Set to 1 when an attempt is made to convert a non-BCD value to a binary value
SM1.7	Not_Hex	Set to 1 when an ASCII value cannot be converted to a valid hexadecimal value

References

E.2 Interrupt events in priority order

E.2 Interrupt events in priority order

Table E- 2 Priority order for interrupt events

Priority group	Event	Description
Communications <i>Highest Priority</i>	8	Port 0 Receive character
	9	Port 0 Transmit complete
	23	Port 0 Receive message complete
	24	Port 1 Receive message complete
	25	Port 1 Receive character
	26	Port 1 Transmit complete
Discrete <i>Medium Priority</i>	0	I0.0 Rising edge
	2	I0.1 Rising edge
	4	I0.2 Rising edge
	6	I0.3 Rising edge
	35	I7.0 Rising edge (signal board)
	37	I7.1 Rising edge (signal board)
	1	I0.0 Falling edge
	3	I0.1 Falling edge
	5	I0.2 Falling edge
	7	I0.3 Falling edge
	36	I7.0 Falling edge (signal board)
	38	I7.1 Falling edge (signal board)
	12	HSC0 CV=PV (current value = preset value)
	27	HSC0 Direction changed
	28	HSC0 External reset
	13	HSC1 CV=PV (current value = preset value)
	16	HSC2 CV=PV (current value = preset value)
	17	HSC2 Direction changed
	18	HSC2 External reset
	32	HSC3 CV=PV (current value = preset value)
Timed <i>Lowest Priority</i>	10	Timed interrupt 0 SMB34
	11	Timed interrupt 1 SMB35
	21	Timer T32 CT=PT interrupt
	22	Timer T96 CT=PT interrupt

E.3 High-speed counter summary

Table E- 3 S7-200 SMART HSC input assignments and capabilities

	Clock A	Dir / Clock B	Reset	Single phase max. clock/input rate	Dual phase / AB quadrature phase max. clock/input rate
HSC0	I0.0	I0.1	I0.4	200 kHz (S model CPUs) ¹ 100 kHz (C model CPUs) ²	100 kHz (S model CPUs) (Maximum 1x count rate = 100 kHz) (Maximum 4x count rate = 400 kHz) 50 kHz (C model CPUs) (Maximum 1x count rate = 50 kHz) (Maximum 4x count rate = 200 kHz)
HSC1	I0.1			200 kHz (S model CPUs) 100 kHz (C model CPUs)	
HSC2	I0.2	I0.3	I0.5	200 kHz (S model CPUs) 100 kHz (C model CPUs)	100 kHz (S model CPUs) (Maximum 1x count rate = 100 kHz) (Maximum 4x count rate = 400 kHz) 50 kHz (C model CPUs) (Maximum 1x count rate = 50 kHz) (Maximum 4x count rate = 200 kHz)
HSC3	I0.3			200 kHz (S model CPUs) 100 kHz (C model CPUs)	

¹ S model CPUs: SR20/ST20, SR30/ST30, SR40/ST40, SR60/ST60

² C model CPUs: CR40, CR60

E.4 Instructions

Instructions

The STL instruction names and descriptions are shown in the tables below. See the chapter on program instructions (Page 139) for the LAD and FBD instructions.

Boolean instructions	
STL	Description
LD bit	Load
LDI bit	Load Immediate
LDN bit	Load Not
LDNI bit	Load Not Immediate
A bit	AND
AI bit	AND Immediate
AN bit	AND Not
ANI bit	AND Not Immediate

References

E.4 Instructions

Boolean instructions	
STL	Description
O bit	OR
OI bit	OR Immediate
ON bit	OR Not
ONI bit	OR Not Immediate
LDBx IN1, IN2	Load result of Byte Compare IN1 (x:<, <=, =, >, >=) IN2
ABx IN1, IN2	AND result of Byte Compare IN1 (x:<, <=, =, >, >=) IN2
OBx IN1, IN2	OR result of Byte Compare IN1 (x:<, <=, =, >, >=) IN2
LDWx IN1, IN2	Load result of Word Compare IN1 (x:<, <=, =, >, >=) IN2
AWx IN1, IN2	AND result of Word Compare IN1 (x:<, <=, =, >, >=) IN2
OWxIN1, IN2	OR result of Word Compare IN1 (x:<, <=, =, >, >=) IN2
LDDx IN1, IN2	Load result of DWord Compare IN1 (x:<, <=, =, >, >=) IN2
ADx IN1, IN2	AND result of DWord Compare IN1 (x:<, <=, =, >, >=) IN2
ODx IN1, IN2	OR result of DWord Compare IN1 (x:<, <=, =, >, >=) IN2
LDRx IN1, IN2	Load result of Real Compare IN1 (x:<, <=, =, >, >=) IN2
ARx IN1, IN2	AND result of Real Compare IN1 (x:<, <=, =, >, >=) IN2
ORx IN1, IN2	OR result of Real Compare IN1 (x:<, <=, =, >, >=) IN2
NOT	Stack Negation
EU	Detection of Rising Edge
ED	Detection of Falling Edge
= bit	Assign Value
=I bit	Assign Value Immediate
S bit, N	Set bit Range
R bit, N	Reset bit Range
SI bit, N	Set bit Range Immediate
RI bit, N	Reset bit Range Immediate
Not available in STL	SR (Set dominate bistable) RS (Reset dominate bistable)

Boolean instructions	
STL	Description
LDSx IN1, IN2	Load result of String Compare IN1 (x: =, <>) IN2
ASx IN1, IN2	AND result of String Compare IN1 (x: =, <>) IN2
OSx IN1, IN2	OR result of String Compare IN1 (x: =, <>) IN2
ALD	AND Load
OLD	OR Load
LPS	Logic Push (stack control)
LRD	Logic Read (stack control)
LPP	Logic Pop (stack control)
LDS n	Load Stack (stack control), n= 0 to 8
AENO	AND ENO

Math, increment, and decrement instructions	
STL	Description
+I IN1, OUT	Add Integer, Double Integer or Real
+D IN1, OUT	IN1+OUT=OUT
+R IN1, OUT	
-I IN1, OUT	Subtract Integer, Double Integer, or Real
-D IN1, OUT	OUT-IN1=OUT
-R IN1, OUT	
MUL IN1, OUT	Multiply Integer (16*16->32)
*I IN1, OUT	Multiply Integer, Double Integer, or Real
*D IN1, OUT	IN1 * OUT = OUT
*R IN1, IN2	
DIV IN1, OUT	Divide Integer (16/16->32)
/I IN1, OUT	Divide Integer, Double Integer, or Real
/D,IN1, OUT	OUT / IN1 = OUT
/R IN1, OUT	
SQRT IN, OUT	Square Root
LN IN, OUT	Natural Logarithm
EXP IN, OUT	Natural Exponential
SIN IN, OUT	Sine
COS IN, OUT	Cosine
TAN IN, OUT	Tangent
INCB OUT	Increment Byte, Word or DWord
INCW OUT	
INCD OUT	

References

E.4 Instructions

Math, increment, and decrement instructions	
STL	Description
DECB OUT	Decrement Byte, Word, or DWord
DECW OUT	
DECD OUT	
PID TBL, LOOP	PID Loop

Timer and counter instructions	
STL	Description
TON Txxx, PT	On-Delay Timer
TOF Txxx, PT	Off-Delay Timer
TONR Txxx, PT	Retentive On-Delay Timer
BITIM OUT	Beginning Interval Timer
CITIM IN, OUT	Calculate Interval Timer
CTU Cxxx, PV	Count Up
CTD Cxxx, PV	Count Down
CTUD Cxxx, PV	Count Up/Down

High-speed instructions	
STL	Description
HDEF HSC, MODE	Define High-Speed Counter mode
HSC n	Activate High-Speed Counter
PLS n	Pulse Output:

Real time clock instructions	
STL	Description
TODR T	Read Time of Day clock
TODW T	Write Time of Day clock
TODRX T	Read Real Time Clock Extended
TODWX T	Set Real Time Clock Extended

Program control instructions	
STL	Description
END	Conditional End of Program
STOP	Transition to STOP Mode
WDR	WatchDog Reset (500 ms)
JMP N	Jump to defined Label
LBL N	Define a Label

Program control instructions	
STL	Description
CALL N [N1,...]	Call a Subroutine [N1, ... up to 16 optional parameters]
CRET	Conditional Return from SBR
FOR INDX,INIT,FINAL NEXT	For/Next Loop
LSCR N SCRT N CSCRE SCRE	Load, Transition, Conditional End, and End Sequence Control Relay segment
GERR ECODE	Get Error code

Move, Shift, and Rotate instructions	
STL	Description
MOVB IN, OUT	Move Byte, Word, DWord, Real
MOVW IN, OUT	
MOVD IN, OUT	
MOVR IN, OUT	
BIR IN, OUT	Move Byte Immediate Read
BIW IN, OUT	Move Byte Immediate Write
BMB IN, OUT, N	Block Move Byte, Word, DWord
BMW IN, OUT, N	
BMD IN, OUT, N	
SWAP IN	Swap Bytes
SHRB DATA, S_BIT, N	Shift Register Bit
SRB OUT, N	Shift Right Byte, Word, DWord
SRW OUT, N	
SRD OUT, N	
SLB OUT, N	Shift Left Byte, Word, DWord
SLW OUT, N	
SLD OUT, N	
RRB OUT, N	Rotate Right Byte, Word, DWord
RRW OUT, N	
RRD OUT, N	
RLB OUT, N	Rotate Left Byte, Word, DWord
RLW OUT, N	
RLD OUT, N	

References

E.4 Instructions

Logical instructions	
STL	Description
ANDB IN1, OUT ANDW IN1, OUT ANDD IN1, OUT	Logical AND of Byte, Word, and DWord
ORB IN1, OUT ORW IN1, OUT ORD IN1, OUT	Logical OR of Byte, Word, and DWord
XORB IN1, OUT XORW IN1, OUT XORD IN1, OUT	Logical XOR of Byte, Word, and DWord
INVB OUT INVW OUT INVD OUT	Invert Byte, Word and DWord (1's complement)

String instructions	
STL	Description
SLEN IN, OUT	String Length
SCAT N, OUT	Concatenate String
SCPY IN, OUT	Copy String
SSCPY IN, INDX, N, OUT	Copy Substring from String
CFND IN1, IN2, OUT	Find First Character within String
SFND IN1, IN2, OUT	Find String within String

Table, Find, and Conversion instructions	
STL	Description
ATT DATA, TBL	Add data to table
LIFO TBL, DATA FIFO TBL, DATA	Get data from table
FND= TBL, PTN, INDX FND<> TBL, PTN, INDX FND< TBL, PTN, INDX FND> TBL, PTN, INDX	Find data value in table that matches comparison
FILL IN, OUT, N	Fill memory space with pattern
BCDI OUT IBCD OUT	Convert BCD to Integer Convert Integer to BCD
BTI IN, OUT ITB IN, OUT ITD IN, OUT DTI IN, OUT	Convert Byte to Integer Convert Integer to Byte Convert Integer to Double Integer Convert Double Integer to Integer

Table, Find, and Conversion instructions	
STL	Description
DTR IN, OUT	Convert DWord to Real
TRUNC IN, OUT	Convert Real to Double Integer
ROUND IN, OUT	Convert Real to Double Integer
ATH IN, OUT, LEN	Convert ASCII to Hex
HTA IN, OUT, LEN	Convert Hex to ASCII
ITA IN, OUT, FMT	Convert Integer to ASCII
DTA IN, OUT, FM	Convert Double Integer to ASCII
RTA IN, OUT, FM	Convert Real to ASCII
DECO IN, OUT	Decode
ENCO IN, OUT	Encode
SEG IN, OUT	Generate 7-segment pattern
ITS IN, FMT, OUT	Convert Integer to String
DTS IN, FMT, OUT	Convert Double Integer to String
RTS IN, FMT, OUT	Convert Real to String
STI STR, INDEX, OUT	Convert Substring to Integer
STD STR, INDEX, OUT	Convert Substring to Double Integer
STR STR, INDEX, OUT	Convert Substring to Real

Interrupt instructions	
STL	Description
CRETI	Conditional Return from Interrupt
ENI	Enable Interrupts
DISI	Disable Interrupts
ATCH INT, EVNT	Attach Interrupt routine to event
DTCH EVNT	Detach event
CEVENT EVNT	Clear all interrupt events of type EVNT

Communications instructions	
STL	LAD/FBD
GET	Reads remote station data
PUT	Writes data to a remote station
XMT TBL, PORT	Freeport transmission
RCV TBL, PORT	Freeport receive message
GIP ADDR, MASK, GATE	Get CPU address, subnet mask, and gateway
SIP ADDR, MASK, GATE	Set CPU address, subnet mask, and gateway
GPA ADDR, PORT	Get Port Address
SPA ADDR, PORT	Set Port Address

E.5 Memory ranges and features

Description	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU CR40, CPU CR60	CPU SR60, CPU ST60
User program size	12 Kbytes	18 Kbytes	24 Kbytes	12 Kbytes	30 Kbytes
User data size	8 Kbytes	12 Kbytes	16 Kbytes	8 Kbytes	20 Kbytes
Process-image input register	I0.0 to I31.7	I0.0 to I3.7	I0.0 to I31.7	I0.0 to I31.7	I0.0 to I31.7
Process-image output register	Q0.0 to Q31.7	Q0.0 to Q31.7	Q0.0 to Q31.7	Q0.0 to Q31.7	Q0.0 to Q31.7
Analog inputs (read only)	AIW0 to AIW110	AIQ0 to AIW110	AIW0 to AIW110	--	AIW0 to AIW110
Analog outputs (write only)	AQW0 to AQW110	AQW0 to AQW110	AQW0 to AQW110	---	AQW0 to AQW110
Variable memory (V)	VB0 to VB8191	VB0 to VB12287	VB0 to VB16383	VB0 to VB8191	VB0 to VB20479
Local memory (L) ¹	LB0 to LB63	LB0 to LB63	LB0 to LB63	LB0 to LB63	LB0 to LB63
Bit memory (M)	M0.0 to M31.7	M0 to M31.7	M0.0 to M31.7	M0.0 to M31.7	M0.0 to M31.7
Special Memory (SM)	SM0.0 to SM1535.7	SM0.0 to SM1535.7	SM0.0 to SM1535.7	SM0.0 to SM1535.7	SM0.0 to SM1535.7
Read only	SM0.0 to SM29.7 SM1000.0 to SM1535.7	SM0 to SM29.7 SM1000.0 to SM1535.7	SM0.0 to SM29.7 SM1000.0 to SM1535.7	SM0.0 to SM29.7 SM1000.0 to SM1535.7	SM0.0 to SM29.7 SM1000.0 to SM1535.7
Timers	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)	256 (T0 to T255)
Retentive on-delay	1 ms	T0, T64	T0, T64	T0, T64	T0, T64
	10 ms	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68	T1 to T4, and T65 to T68
	100 ms	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95	T5 to T31, and T69 to T95
On/Off delay	1 ms	T32, T96	T32, T96	T32, T96	T32, T96
	10 ms	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100	T33 to T36, and T97 to T100
	100 ms	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255	T37 to T63, and T101 to T255
Counters	256 (C0 to C255)	256 (C0 to C255)	256 (C0 to C255)	256 (C0 to C255)	256 (C0 to C255)
High-speed counters	HC0 to HC3	HC0 to HC3	HC0 to HC3	HC0 to HC3	HC0 to HC3
Sequential control relays (S)	S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7	S0.0 to S31.7
Accumulator registers	AC0 to AC3	AC0 to AC3	AC0 to AC3	AC0 to AC3	AC0 to AC3
Jumps/Labels	0 to 255	0 to 255	0 to 255	0 to 255	0 to 255
Call/Subroutine	0 to 127	0 to 127	0 to 127	0 to 127	0 to 127
Interrupt routines	0 to 127	0 to 127	0 to 127	0 to 127	0 to 127
Positive/negative transitions	1024	1024	1024	1024	1024

Description	CPU SR20, CPU ST20	CPU SR30, CPU ST30	CPU SR40, CPU ST40	CPU CR40, CPU CR60	CPU SR60, CPU ST60
PID loops	0 to 7				
Ports	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)	Ethernet programming port, Integrated RS485 port (Port 0), CM01 Signal Board (SB) RS232/RS485 port (Port 1)

¹ LB60 to LB63 are reserved by STEP 7-Micro/WIN SMART when programming in LAD or FBD.

References

E.5 Memory ranges and features

Order numbers

F.1 Order numbers

CPU models	Order number
CPU SR20, AC/DC/Relay	6ES7 288-1SR20-0AA0
CPU ST20, DC/DC/DC	6ES7 288-1ST20-0AA0
CPU SR30, AC/DC/Relay	6ES7 288-1SR30-0AA0
CPU ST30, DC/DC/DC	6ES7 288-1ST30-0AA0
CPU ST40, DC/DC/DC	6ES7 288-1ST40-0AA0
CPU SR40, AC/DC/Relay	6ES7 288-1SR40-0AA0
CPU CR40, AC/DC/Relay	6ES7 288-1CR40-0AA0
CPU SR60, AC/DC/Relay	6ES7 288-1SR60-0AA0
CPU ST60, DC/DC/DC	6ES7 288-1ST60-0AA0
CPU CR60, AC/DC/Relay	6ES7 288-1CR60-0AA0

Expansion modules and signal boards	Order number
EM Digital 8 x Inputs (EM DI08)	6ES7 288-2DE08-0AA0
EM Digital 8 x Outputs (EM DT08)	6ES7 288-2DT08-0AA0
EM Digital 8 x Outputs Relay (EM DR08)	6ES7 288-2DR08-0AA0
EM Digital 8 x Inputs / Digital 8 x Outputs (EM DT16)	6ES7 288-2DT16-0AA0
EM Digital 8 x Inputs/ Relay 8 x Outputs (EM DR16)	6ES7 288-2DR16-0AA0
EM Digital 16 x Inputs / Digital 16 x Outputs (EM DT32)	6ES7 288-2DT32-0AA0
EM Digital 16 x Inputs / Relay 16 x Outputs (EM DR32)	6ES7 288-2DR32-0AA0
EM Analog 4 x Inputs (EM AE04)	6ES7 288-3AE04-0AA0
EM Analog 2 x Outputs (EM AQ02)	6ES7 288-3AQ02-0AA0
EM Analog 4 x Inputs / Analog 2 x Outputs (EM AM06)	6ES7 288-3AM06-0AA0
EM RTD 2 x 16 bit (EM AR02)	6ES7 288-3AR02-0AA0
EM TC 4 x 16 bit (EM AT04)	6ES7 288-3AT04-0AA0
SB Digital 2 x Inputs / Digital 2 x Outputs (SB DT04)	6ES7 288-5DT04-0AA0
SB Analog 1 x Output (SB AQ01)	6ES7 288-5AQ01-0AA0
SB RS485/RS232 (SB CM01)	6ES7 288-5CM01-0AA0
SB Battery (SB BT01)	6ES7 288-5BA01-0AA0

Order numbers

F.1 Order numbers

Software	Order number
STEP 7-Micro/WIN SMART Individual License (CD-ROM)	6ES7 288-8SW01-0AA0
STEP 7 Basic V12 SP1	6ES7 822-0AA01-0YA0
STEP 7 Professional V12 SP1	6ES7 822-1AA01-0YA5
WinCC flexible 2008 Micro Individual License (DVD-ROM without license key)	6AV6610-0AA01-3CA8
WinCC flexible 2008 Micro Upgrade License (DVD-ROM without license key)	6AV6610-0AA01-3CE8
WinCC flexible 2008 Compact Individual License (DVD-ROM with license key)	6AV6611-0AA51-3CA5
WinCC flexible 2008 Compact Upgrade License (DVD-ROM with license key)	6AV6611-0AA51-3CE5
SINAUT MICRO SC 8 Single License for 1 Installation	6NH9 910-0AA10-0AA3
SINAUT MICRO SC 64 Single License for 1 Installation	6NH9 910-0AA10-0AA6
SINAUT MICRO SC 256 Single License for 1 Installation	6NH9 910-0AA10-0AA8
Drives V90 (PC tools) software	Can be downloaded from the Siemens Services and Support website

Communications cards	Order number
CP 5411: Short AT ISA	6GK 1 541-1AA00
CP 5512: PCMCIA Type II	6GK 1 551-2AA00
CP 5611: PCI card (version 3.0 or greater)	6GK 1 561-1AA00

Manuals	Order number
S7-200 SMART Programmable Controller System Manual (Chinese Traditional))	Included on S7-200 SMART Companion Disk and on Siemens customer support website
S7-200 SMART Programmable Controller System Manual (Chinese Simplified)	
S7-200 SMART Programmable Controller System Manual (English)	
CP 243-2 AS-Interface Master Manual (English)	6GK7 243-2AX00-8BA0
WinCC flexible 2005 Micro User's Manual	6AV6 691-1AA01-0AB0
SIMATIC HMI Manual Collection	6AV6 691-1SA01-0AX0
SIMATIC Text Display (TD) User Manual - Refer to HMI panels for S7-200 SMART	Can be downloaded from the Siemens Services and Support website

Cables, network connectors, repeaters, and end retainers	Order number
MPI Cable	6ES7 901-0BF00-0AA0
PROFIBUS Network Cable	6XV1 830-0AH10
Network Bus Connector with Programming Port Connector, Vertical Cable Outlet	6ES7 972-0BB11-0XA0
Network Bus Connector (no programming port connector), Vertical Cable Outlet	6ES7 972-0BA11-0XA0
RS485 Bus Connector with 35° Cable Outlet (no programming port connector)	6ES7 972-0BA40-0XA0
RS485 Bus Connector with 35° Cable Outlet (with programming port connector)	6ES7 972-0BB40-0XA0
Terminal Block: Tin: 7 terminal, 4 pk	6ES7 292-1AG30-0XA0

Cables, network connectors, repeaters, and end retainers	Order number
Terminal Block: Tin: 8 terminal, 4/pk	6ES7 292-1AH30-0XA0
Terminal Block: Tin: 11 terminal, 4/pk	6ES7 292-1AL30-0XA0
Terminal Block: Tin: 12 terminal, 4/pk	6ES7 292-1AM30-0XA0
Terminal Block: Tin: 20 terminal, 4/pk	6ES7 292-1AV30-0XA0
Terminal Block: Gold: 7 terminal, 4/pk	6ES7 292-1BG30-0XA0
RS485 IP 20 Repeater, Isolated	6ES7 972-0AA00-0XA0
TD/CPU Connecting Cable	6ES7 901-3EB10-0XA0
End Retainer Thermoplast, 10 MM	8WA1808
End Retainer, Steel	8WA1805

Human Machine Interface	Order number
COMFORT HMIs	
SIMATIC HMI TP700 COMFORT	
SIMATIC HMI TP900 COMFORT	6AV2124-0GC01-0AX0
SIMATIC HMI TP1200 COMFORT	6AV2124-0JC01-0AX0
SIMATIC HMI KP400 COMFORT	6AV2124-0MC01-0AX0
SIMATIC HMI KP700 COMFORT	6AV2124-1DC01-0AX0
SIMATIC HMI KP900 COMFORT	6AV2124-1GC01-0AX0
SIMATIC HMI KP1200 COMFORT	6AV2124-1JC01-0AX0
SIMATIC HMI KTP400 COMFORT	6AV2124-1MC01-0AX0
SMART LINE HMIs	
SMART LINE 700 IE	6AV6648-0BC11-3AX0
SMART LINE 1000 IE	6AV6648-0BE11-3AX0
BASIC HMIs	
SIMATIC HMI KTP400 BASIC MONO PN	6AV6647-0AA11-3AX0
SIMATIC HMI KTP600 BASIC MONO PN	6AV6647-0AB11-3AX0
SIMATIC HMI KTP600 BASIC COLOR DP	6AV6647-0AC11-3AX0
SIMATIC HMI KTP600 BASIC COLOR PN	6AV6647-0AD11-3AX0
SIMATIC HMI KTP1000 BASIC COLOR DP	6AV6647-0AE11-3AX0
SIMATIC HMI KTP1000 BASIC COLOR PN	6AV6647-0AF11-3AX0
SIMATIC HMI TP1500 BASIC COLOR PN	6AV6647-0AG11-3AX0
SIMATIC HMI KP300 BASIC MONO PN	6AV6647-0AH11-3AX0
Micro HMIs	
TD 400C TEXT DISPLAY, 4 LINES ¹	6AV6640-0AA00-0AX1
TD400C Blank faceplate material, A4 size (10 sheets/package)	6AV6671-0AP00-0AX0
¹ : Includes one blank faceplate overlay for customization. For additional blank faceplate overlays, order the blank faceplate material for your TD device	

Order numbers

F.1 Order numbers

Index

*

*D (STL-Multiply double integer), 240
*I (STL-Multiply integer), 240
*R (STL-Multiply real), 240

.

.mwp files, 83
.smart files, 83

/

/D (STL-Divide double integer), 240
/I (STL-Divide integer), 240
/R (STL-Divide real), 240

+

+D (STL-Add double integer), 240
+I (STL-Add integer), 240
+R (STL-Add real), 240

<

<=B, 182
<=R, 182
<=W, 182
<>B, 182
<>R, 182
<>S, 186
<>W, 182
<B, 182
<R, 182
<W, 182

=

= (STL-output), 148
==B, 182
==R, 182
==S, 186
==W, 182
=I (STL-output immediate), 148

>

>=B, 182
>=R, 182
>=W, 182
>B, 182
>R, 182
>W, 182

A

A (STL-AND), 139
AB< (STL-AND compare byte less than), 182
AB<= (STL-AND compare byte less than or equal), 182
AB<> (STL-AND compare byte not equal), 182
AB= (STL-AND compare byte equal), 182
AB> (STL-AND compare byte greater than), 182
AB>= (STL-AND compare byte greater than or equal), 182
AC inductive loads, 49
Access rights
 CPU security, 115
 password privilege levels, 115
Active/Passive communication partners, 339
AD< (STL-AND compare double word less than), 182
AD<= (STL-AND compare double word less than or equal), 182
AD<> (STL-AND compare double word not equal), 182
AD= (STL-AND compare double word equal), 182
AD> (STL-AND compare double word greater than), 182
AD>= (STL-AND compare double word greater than or equal), 182
ADD_DI, 240
ADD_I, 240
ADD_R, 240
Addressing
 accumulators, 59
 analog inputs, 61
 analog outputs, 62
 counter memory, 58
 creating pointers and using indirect address, 64
 example of pointer offset to access data, 68
 example of pointer to access data in a table, 67
 flag memory, 57
 high-speed counters, 58

- local and expansion I/O, 64
 local memory, 61
 memory areas, 56
 process-image output register, 56
 sequence control relay (SCR) memory, 62
 special memory (SM) bits, 60
 symbol table, 92
 timer memory, 57
 variable memory, 57
AENO (STL stack-AND ENO bit)
 instruction, 144
 logic stack overview, 143
AI (STL-AND immediate), 141
Air flow, 36
Alarms
 analog input configuration, 122
 from system (SMW100-SMW114), 611
ALD (STL stack-AND Load), 144
AN (STL-AND NOT), 139
Analog inputs
 analog type configuration, 120
 rejection, 120
 smoothing, 120
 system block configuration, 120
Analog outputs
 analog type configuration, 123
 states at RUN/ STOP transition, 123
ANDB (STL-AND byte), 276
ANDD (STL-AND dword), 276
ANDW (STL-AND word), 276
ANI (STL-AND NOT immediate), 141
AR< (STL-AND compare real less than), 182
AR<= (STL-AND compare real less than or equal), 182
AR<> (STL-AND compare real not equal), 182
AR= (STL-AND compare real equal), 182
AR> (STL-AND compare real greater than), 182
AR>= (STL-AND compare real greater than or equal), 182
AS<> (STL-AND string compare not equal), 186
AS= (STL-AND string compare equal), 186
ASCII array conversion instructions, 191
Assigning
 global symbols, 92
 local variables, 98
 variables (local), 95
ATCH, 263
ATH, 191
ATT, 309
Auto-tuning PID, 423
AW< (STL-AND compare word less than), 182
AW<= (STL-AND compare word less than or equal), 182
AW<> (STL-AND compare word not equal), 182
AW= (STL-AND compare word equal), 182
AW> (STL-AND compare word greater than), 182
AW>= (STL-AND compare word greater than or equal), 182
Axis 0 motion control (SMB600-SMB649), 613
Axis of Motion
 ACCEL_TIME, 441
 AXISx_ABSPOS, 474
 AXISx_CACHE, 472
 AXISx_CFG, 471
 AXISx_CTRL, 462
 AXISx_DIS, 471
 AXISx_GOTO, 465
 AXISx_LDOFF, 468
 AXISx_LDPOS, 469
 AXISx_MAN, 463
 AXISx_RDPOS, 473
 AXISx_RSEEK, 467
 AXISx_RUN, 466
 AXISx_SRATE, 470
 configuring, 448
 displaying and controlling the operation of axes, 483
 displaying and modifying the configuration of axes, 488
 displaying the profile configuration for the axes, 488
 eliminating backlash, 508
 error codes, 489
 Motion control panel, 482
 programming, 446
 RP Seek modes, 503
 SM locations, 500
 subroutine guidelines, 461
 subroutines, 460
- B**
- B_I, 188
Baud rate
 communications, 109
 setting, 355
 switch selections:RS232/PPI Multi-Master cable, 367
BCD_I, 188
BCDI (STL-BCD to integer), 188
BGN_ITIME, 327
Biasing and terminating
 CM01 signal board, 363
 network cable, 362
Biassing PID loop, 250
BIR (STL-byte immediate read), 282
Bit logic instructions

- AENO (STL-AND ENO), 143
 contacts, 139
 contacts (Immediate), 141
 edge detectors, 147
 input examples,
 NOP (No operation), 151
 NOT, 146
 output coils, 148
 output examples,
 set and reset bits, 149
 set and reset dominant bistable, 150
 STL logic stack instructions for inputs, 143
 STL logic stack operations,
 BITIM (STL-Begin interval timer), 327
 BIW (STL-byte immediate write), 282
 BLKMOV_B, 279
 BLKMOV_D, 279
 BLKMOV_W, 279
 BMB (STL-block move byte), 279
 BMD (STL-block move double word), 279
 BMW (STL-block move word), 279
 Bookmarks in programs, 409
 BT01 battery signal board, 134
 BTI (STL-Byte to integer), 188
 Building status charts, 416
 Building your communication network, 359
- C**
- CAL_ITIME, 327
 CALL, 328
 Card (memory), 76
 CE approval, 509
 CFND (STL-Find character), 306
 Character error received from Freeport (SMB3), 597
 Charts
 building, 416
 creating, 416
 opening, 416
 CHR_FIND, 306
 CITIM (STL-Calculate interval time), 327
 Clearance for airflow and cooling, 36
 Cleared memory card, 137
 Clearing PLC memory, 135
 Clock instructions
 READ_RTC, 155
 READ_RTCX, 157
 SET_RTC, 155
 SET_RTCX, 157
 CLR_EVNT, 263
 CM01 signal board
 biasing and terminating, 363
- connector pin assignments, 361
Coils
 output, 148
 output (immediate), 148
 reset bits, 149
 reset bits immediate, 149
 set bits, 149
 set bits immediate, 149
Cold junction compensation
 thermocouple, 132
 Thermocouple, 567
Communication drivers, 337
Communication ports, 336
 connector pin assignments, 361
 Freeport mode, 365
 system block configuration, 109
Communications
 choices, 354
 dialog, 344
 Ethernet, RS485, and RS232, 335
 hardware connection, 25
 IP address, 344
 locating MAC address on CPU, 351
 network, 24
 number of connections (Ethernet), 335
 number of connections (RS232), 335
 number of connections (RS485), 335
 order numbers for modules, 632
 point-to-point interface (PPI) protocol, 354
 restricting writes, 115
 RS485 configuration, 354
 STEP 7-Micro/WIN SMART settings, 25
Communications instructions
 receive, 167
 transmit, 167
Compare instructions
 compare character strings, 186
 compare number value, 182
Compiling programs
 downloading, 68
 PLC non-fatal program errors, 587
Configuration
 CPU, system block, 107
 dynamic IP information, 344
 Ethernet, 344
 IP address, 344
 MAC address, 351
 profile table (Axis of Motion), 492
 static IP information, 347
Configuration drawings, 79
Connections
 number of connections (Ethernet),

- number of connections (RS232),
number of connections (RS485),
types of communication, 335
 - Connector, 45
 - Contact information, 3
 - Contacts
 - negative edge detector, 147
 - normally closed, 139
 - normally closed (immediate), 141
 - normally open, 139
 - normally open (immediate), 141
 - NOT, 146
 - positive edge detector, 147
 - Convert instructions
 - ASCII array conversions, 191
 - ASCII sub-string to number value, 202
 - encode and decode, 205
 - number value to string,
 - standard conversion,
 - Cooling, 36
 - COS (cosine), 246
 - Counter instructions
 - high-speed counters, 210
 - high-speed counters (initialization examples), 228
 - high-speed counters (programming examples), 216
 - standard counters, 208
 - CPU
 - accessing data, 55
 - clearing memory, 135
 - configuring communication to HMI, 353
 - connecting power, 24
 - CR40 specifications, 528
 - CR40 wiring diagram, 534
 - CR60, 537
 - CR60 wiring diagram, 543
 - dimensions, 15, 38
 - DIN rail, 41
 - Ethernet communication, 340
 - Ethernet port, 344
 - expansion modules supported, 18
 - fatal errors, 590
 - features, 15
 - installation, 39
 - installation on a panel, 41
 - IP address, 344
 - LEDs, 15
 - MAC address, 351
 - memory card, 76
 - non-fatal error memory locations, 590
 - number of communication connections, 335
 - number of PPI connections, 355
 - process image register, 53
 - setting the type, 31
 - SR20 specifications, 513
 - SR20 wiring diagram, 520
 - SR30 specifications, 521
 - SR40 specifications, 528
 - SR40 wiring diagram, 534
 - SR60 specifications, 537
 - SR60 wiring diagram, 543
 - ST20 wiring diagram, 520
 - ST30 specifications, 521
 - ST30 wiring diagram, 526
 - ST40 specifications, 528
 - ST40 wiring diagram, 534
 - ST60 specifications, 537
 - ST60 wiring diagram, 543
 - system block, 107
 - types of communication, 335
 - wiring guidelines, 48
 - CPU
 - SR30 wiring diagram,
 - CPU hardware/firmware ID (SMB1000-SMB1049), 615
 - CPU ID register (SMB6-SMB7), 599
 - CPU ST20
 - specifications, 513
 - CRET (STL-Conditional return from subroutine), 328
 - Cross reference, 410
 - CTD (count down), 208
 - CTU (count up), 208
 - CTUD (count up/down), 208
 - Customer support, 3
- ## D
- D (STL-Subtract double integer), 240
 - Data
 - receiving, 170
 - retention, 114
 - Data block (DB), 82
 - Data log
 - status (SMB480-SMB515), 612
 - DC inductive loads, 49
 - Debugging and monitoring
 - forcing values, 418
 - program editor status, 412
 - DEC_B, 248
 - DEC_DW, 248
 - DEC_W, 248
 - DECB (STL-Decrement byte), 248
 - DECD (STL-Decrement double word), 248
 - DECO, 205
 - DECW (STL-Decrement word), 248
 - Default gateway IP address, 343

- D**
- Defining
 - local variables, 98
 - Device configuration of CPU and modules, 107
 - DI_I, 188
 - DI_R, 188
 - DI_S, 197
 - Differential term, PID algorithm, 258
 - Digital input filter time, 111
 - Digital input filters, 112
 - Digital inputs, 111
 - Digital outputs, 113
 - Dimensions
 - CPU, 15
 - mounting, 38
 - DIN rail, 39, 41
 - DISI, 263
 - Displaying status, 412
 - DIV, 244
 - DIV_DI, 240
 - DIV_I, 240
 - DIV_R, 240
 - Downloading
 - programs, 68
 - sample program, 33
 - Drive communication
 - calculating time requirement, 372
 - Drives, 445, 474, 632
 - DTA, 191
 - DTCH, 263
 - DTI (STL-Double integer to integer), 188
 - DTR (STL-Double integer to real), 188
 - DTS (STL-Double integer to string), 197
 - Dynamic IP information, 344
- E**
- ED (STL-Edge Down), 147
 - Edge detectors, 147
 - Electromagnetic compatibility (EMC), 510
 - Element usage, 410
 - EM (expansion module) hardware/firmware ID (SMB1100-SMB1299), 616
 - ENCO, 205
 - END, 294
 - ENI, 263
 - Environmental
 - industrial environments, 510
 - operating conditions, 511
 - transport and storage conditions, 511
 - Errors
 - Axis of Motion, 489
 - character error received from Freeport communication (SMB3), 597
 - compile and run-time errors (PLC program), 587
 - data retention, 114
 - fatal (PLC), 590
 - fatal error effect on run-time execution, 103
 - GET_TABLE and PUT_TABLE instructions, 161
 - I/O error status, 598
 - I/O module ID and error registers (SMB8-SMB19), 599
 - memory locations (PLC non-fatal errors), 590
 - Modbus master execution, 397
 - Modbus slave execution, 402
 - Motion instruction, 491
 - non-fatal error effect on run-time execution, 101
 - PID auto-tune, 430
 - PLC error handling, 101
 - PWMx_RUN subroutine, 439
 - signal board ID and error registers (SMB28-SMB29), 601
 - timestamp mismatch (PC/PLC program difference), 592
 - Ethernet
 - configuring communication between CPU and HMI device, 353
 - GET, 160
 - IP address, 343
 - MAC address, 351
 - networks, 338
 - number of communication connections, 335
 - PUT, 160
 - TCP/IP protocol, 339
 - types of communication, 335
 - Ethernet network
 - configuring the IP address for a CPU, 344
 - searching for CPU, 350
 - EU STL-Edge Up), 147
 - Events, interrupts, 265
 - Examples
 - Axis of Motion, 475
 - bit logic input, 152
 - bit logic output, 153
 - count up/down counter instruction, 209
 - GET and PUT instructions, 165
 - Modbus Slave protocol, programming, 402
 - subroutine for sampling the value of an analog input, 83
 - Executing
 - program, 53
 - single or multiple scans, 420
 - EXP (natural exponential), 246
 - Expansion and local I/O addressing, 64

- Expansion bus - communication errors (SMW98), 610
Expansion module (EM)
 EM AE04 specifications, 557
 EM AE04 wiring diagram, 558
 EM AM06 specifications, 561
 EM AM06 wiring diagram, 563
 EM AQ02 specifications, 559
 EM AQ02 wiring diagram, 560
 EM AR02 (RTD) specifications, 569
 EM AR02 (RTD) wiring diagram, 573
 EM AT04 specifications, 564
 EM DI08 specifications, 547
 EM DI08 wiring diagram, 548
 EM DR08 specifications, 549
 EM DR08 wiring diagram, 550
 EM DR16 specifications, 551
 EM DR16 wiring diagram, 554
 EM DR32 specifications, 551
 EM DR32 wiring diagram, 555
 EM DT08 specifications, 549
 EM DT08 wiring diagram, 550
 EM DT16 specifications, 551
 EM DT16 wiring diagram, 554
 EM DT32 specifications, 551
 EM DT32 wiring diagram, 555
installing and removing, 42
 wiring diagram, 564
expansion modules
 EM DI08, 547
Expansion modules, 598
 dimensions, 38
module error status (SMB5), 598
module ID and error registers (SMB8-SMB19), 599
- F**
- Factory defaults memory card, 137
Fatal error effect on run-time execution, 103
Fatal errors (PLC), 590
FBD editor, 88
Features
 CPU, 15
 expansion modules supported, 18
FIFO, 310
File menu
 Download, 68
 Upload, 71
FILL (STL-table fill), 313
FILL_N, 313
Filter time, 111
Filters, digital input configuration, 112
First scan flag (SMB0), 595
- First scan, executing, 420
Flag memory, 57
Floating point values, 259
FND=, <>, <, > (STL-table find), 314
FOR, 282
Force
 writing and forcing outputs in STOP mode, 419
Forced value indicator (SMB4), 598
Forgotten password, 115, 136
Freeport mode
 character interrupt control, 178
 enabling, 168
 example, 365
 Freeport character error (SMB3), 597
 Freeport configuration (SMB30-port 0 and SMB130-port 1), 601
 Freeport receive character (SMB2), 597
 Freeport transmitter idle (SMB4), 598
 interrupts, 268
 SMB86-SMB94 and SMB186-SMB194 receive message control, 608
Freeze outputs
 analog output configuration, 123
 digital output configuration, 113
- G**
- General technical specifications, 509
GERR (STL-Get non-fatal error code), 296
GET, 160
GET_ADDR, 180
GET_ERROR, 296
GIP_ADDR, 181
Global symbols, 92
Grounding, 47
Guidelines
 grounding and circuit, 46
 installation on a panel, 41
 installation procedures, 39
- H**
- Hardware troubleshooting, 421
HDEF (high-speed counter definition), 210
Heat, high voltage, and electrical noise, 35
High-speed counter registers, 603
High-speed counters, 216
High-vibration environment, 41
HMI
 configuring Ethernet communication, 353
 devices, 364

- general guidelines for networks, 364
 multi-master and multi-slave PPI networks, 358
 single-master PPI networks, 358
 supported devices, 337
- Hotline**, 3
HSC (high-speed counter), 137, 210
HSC0, HSC1, and HSC2 high-speed counter registers (SMB36-65, SMB136-145), 603
HTA, 191
- I**
- I (STL-Subtract integer), 240
 - I/O addressing, 64
 - I/O error status
 - PLC non-fatal error codes, 587
 - PLC non-fatal error SM flags, 590
 - SMB5, 598
 - I/O expansion bus - communication errors (SMW98), 610
 - I/O Module ID and error registers (SMB8-SMB19), 599
 - I_B, 188
 - I_BCD, 188
 - I_DI, 188
 - I_S, 197
 - IBCD (STL-Integer to BCD), 188
 - Illegal syntax
 - symbol table, 92
 - Immediate I/O read/writes, 53
 - INC_B, 248
 - INC_DW, 248
 - INC_W, 248
 - INCB (STL-Increment byte), 248
 - INCD (STL-Increment double word), 248
 - INCW (STL-Increment word), 248
 - Indirect addressing
 - creating pointers and using indirect address, 64
 - example of pointer offset to access data, 68
 - example of pointer to access data in a table, 67
 - symbol table, 94
 - Inductive loads, 49
 - Input filter time, 111
 - Input process-image register, 52
 - Inputs
 - edge detectors, 147
 - example bit logic, 152
 - NOT contact, 146
 - physical and in program, 51
 - pulse catch bits (system block), 112
 - reading, 52
 - standard contacts, 139, 141
 - STL logic stack, 143
- Installation
 - clearance for airflow and cooling, 36
 - dimensions, 38
 - DIN rail, 41
 - expansion module (EM), 42
 - grounding, 47
 - grounding and circuit, 46
 - guidelines, 35
 - high-vibration environment, 41
 - inductive loads, 49
 - isolation, 47
 - lamp loads, 48
 - overview, 35, 39
 - panel, 41
 - separate the devices from heat, high voltage, and electrical noise, 35
 - signal board (SB), 43
 - terminal block connector, 45
- Instruction execution status bits (SMB1), 596
- Instruction libraries, 369
- Instructions
 - GET, 160
 - GET_ADDR, 180
 - GIP_ADDR, 181
 - loop control (PID), 250
 - PUT, 160
 - quick reference guide, 621
 - SET_ADDR, 180
 - SIP_ADDR, 181
- Integral term, PID algorithm, 257
- Interrupt routines, 53
 - elements of a user program, 82
- Interrupts
 - attach/detach, enable/disable, conditional return, and clear event instructions, 263
 - event support by CPU model, 265
 - example programs, 269
 - global interrupt enable state (SMB4), 598
 - interrupt queue overflow (SMB4), 598
 - overview, 265
 - priority and queuing, 269
 - programming guidelines, 267
 - time interval values for timed interrupts (SMB34-SMB35), 602
 - types of interrupt events, 268
- INV_B, 275
 - INV_DW, 275
 - INV_W, 275
 - INV_B (STL-invert byte), 275
 - INV_D (STL-invert double word), 275
 - INV_W (STL-invert word), 275
- IP address, 343

- assigning, 341, 349
configuring, 344
MAC address, 351
IP router, 344
Isolation, 47
ITA, 191
ITB (STL-Integer to byte), 188
ITD (STL-Integer to double integer), 188
ITS (STL-Integer to string), 197
- J**
JMP, 284
- L**
L memory, 95
LAD editor, 88
Lamp loads, 48
LBL, 284
LD (STL stack-Load NOT immediate), 143
LD (STL stack-Load), 143
LD (STL-Load), 139
LDB< (STL-Load compare byte less than), 182
LDB<= (STL-Load compare byte less than or equal), 182
LDB<> (STL-Load compare byte not equal), 182
LDB= (STL-Load compare byte equal), 182
LDB> (STL-Load compare byte greater than), 182
LDB>= (STL-Load compare byte greater than or equal), 182
LDD< (STL-Load compare double word less than), 182
LDD<= (STL-Load compare double word less than or equal), 182
LDD<> (STL-Load compare double word not equal), 182
LDD= (STL-Load compare double word equal), 182
LDD> (STL-Load compare double word greater than), 182
LDD>= (STL-Load compare double word greater than or equal), 182
LDI (STL stack-Load immediate), 143
LDI (STL-Load NOT immediate), 141
LDN (STL stack-Load NOT), 143
LDN (STL-Load NOT), 139
LDNI (STL-Load NOT immediate), 141
LDR< (STL-Load compare real less than), 182
LDR<= (STL-Load compare real less than or equal), 182
LDR<> (STL-Load compare real not equal), 182
LDR= (STL-Load compare real equal), 182
- LDR> (STL-Load compare real greater than), 182
LDR>= (STL-Load compare real greater than or equal), 182
LDS (STL stack-Load), 144
LDS<> (STL-Load string compare not equal), 186
LDS= (STL-Load string compare equal), 186
LDW< (STL-Load compare word less than), 182
LDW<= (STL-Load compare word less than or equal), 182
LDW<> (STL-Load compare word not equal), 182
LDW= (STL-Load compare word equal), 182
LDW> (STL-Load compare word greater than), 182
LDW>= (STL-Load compare word greater than or equal), 182
Library
 creating, 369
 USS protocol library, 371
LIFO, 310
LN (natural logarithm), 246
Local and expansion I/O addressing, 64
Local variables, 95
Local/Partner connection, 339
Logic stack
 STL inputs, 143
 STL stack operations, 144
Logic, control, 51
Logical operation instructions
 AND, OR, XOR (byte, word, and dword),
 invert, 275
Loop control (PID)
 adjusting bias, 261
 converting inputs, 259
 converting outputs, 260
 error conditions, 262
 forward/reverse, 261
 loop definition table, 424
 modes, 262
Loop table, 263
Lost password, 136
LPP (STL stack-Logic Pop), 144
LPS (STL stack-Logic Push), 144
LRD (STL stack-Logic Read), 144
LSCR (STL-Load SCR), 285
- M**
MAC address, 351
Main program, 81
Math instructions
 add, subtract, multiply and divide,
 divide integer with remainder, 244
 increment and decrement,

- multiply integers to double integer and divide integer with remainder, 244
- numeric functions,
- MBS_MSG (send message from Modbus master), 394
- MBUS_CTRL (intialize Modbus master communication), 392
- MBUS_SLAVE (slave response to master message), 401
- Memory, 590
 - addresses for non-fatal error indicators, 590
 - clearing PLC, 135
 - retentive range configuration, 114
- Memory card
 - program transfer card, 76
 - reset to factory defaults,
 - type, 72
 - using, 73
- Modbus general
 - addressing, 390
 - advanced user information, 405
 - initialization and execution time for Modbus protocol, 389
 - library features, 386
 - requirements for using Modbus instructions, 387
- Modbus master
 - example program, 403
 - execution error codes, 397
 - MBUS_CTRL (intialize master communication), 392
 - MBUS_MSG (send message from master), 394
 - using Modbus master instructions, 391
- Modbus slave
 - execution error codes, 402
 - MBUS_INIT (intialize slave communication), 400
 - MBUS_SLAVE (slave response to master message), 401
 - using the Modbus slave instructions, 398
- Modules
 - CPU CR40, 528
 - CPU CR60, 537
 - CPU SR20, 513
 - CPU SR30, 521
 - CPU SR40, 528
 - CPU SR60, 537
 - CPU ST20, 513
 - CPU ST40, 528
 - CPU ST60, 537
 - dimensions, 38
 - EM AE04, 557
 - EM AM06, 561
 - EM AQ02, 559
 - EM AR02 (RTD), 569
 - EM DI08, 547
 - EM DR08, 549
 - EM DR16, 551
 - EM DR32, 551
 - EM DT08, 549
 - EM DT16, 551
 - EM DT32, 551
 - SB AQ01, 577, 578
 - SB CM01, 579
 - SB DT04, 574, 576
 - ST30, 521
- Motion control,
 - Motion features, 445
- Motion control panel, 481
 - displaying and controlling the operation of axes, 483
 - displaying and modifying the configuration of axes, 488
 - displaying the profile configuration for the axes, 488
- Motion inputs and outputs
 - CPU, 445
- Motion instruction, error codes, 491
- Motion profile
 - configuring, 442
 - creating steps, 444
 - defining, 442
 - mode of operation, 443
- Motion wizard
 - configuration/profile table, 492
 - maximum and start/stop speeds, 440
 - SM locations, 500
- Mounting
 - DIN rail, 41
 - overview, 39
 - panel, 41
- MOV_B, 278
- MOV_BIR, 282
- MOV_BIW, 282
- MOV_DW, 278
- MOV_R, 278
- MOV_W, 278
- MOVB (STL-move byte), 278
- MOVD (STL-move double word), 278
- Move instructions
 - block move (byte, word, dword), 279
 - move (byte, word, dword, real), 278
 - move byte immediate (read and write), 282
 - SWAP (exchange byte data in a word), 280
- MOVR (STL-move real), 278
- MOVW (STL-move word), 278
- MUL, 244
- MUL_DI, 240
- MUL_I, 240
- MUL_R, 240

Multiple scans, 420

N

Network communications, 24

Networks (communication)

addresses, 356

biasing and terminating the network cable, 362

biasing cable, 362

calculating network distances, 360

general guidelines for building, 359

network configurations, 354

safety concerns, 359

sample RS485 network configurations, 358

selecting the network cable, 361

single-master PPI, 358

types of communication, 335

NEXT, 282

Non-fatal error effect on run-time execution, 101

Non-fatal PLC errors

compile and run-time, 587

Special Memory locations, 590

Non-volatile memory, 72

NOP, 151

NOT (STL), 146

Number value to string conversion instructions, 197

O

O (STL-OR), 139

OB< (STL-OR compare byte less than), 182

OB<= (STL-OR compare byte less than or equal), 182

OB<> (STL-OR compare byte not equal), 182

OB= (STL-OR compare byte equal), 182

OB> (STL-OR compare byte greater than), 182

OB>= (STL-OR compare byte greater than or

equal), 182

OB1, 81

OD< (STL-OR compare double word less than), 182

OD<= (STL-OR compare double word less than or equal), 182

OD<> (STL-OR compare double word not equal), 182

OD= (STL-OR compare double word equal), 182

OD> (STL-OR compare double word greater than), 182

OD>= (STL-OR compare double word greater than or equal), 182

OI (STL-OR immediate), 141

OLD (STL stack-OR Load), 144

ON (STL-OR NOT), 139

ONI (STL-OR NOT immediate), 141

Open loop control, 437

Opening earlier STEP 7-Micro/WIN projects, 83

Operating mode

changing to RUN, 33, 78

changing to STOP, 34, 78

startup options, 119

Operator stations, 79

Options

STL status, 415

OR< (STL-OR compare real less than), 182

OR<= (STL-OR compare real less than or equal), 182

OR<> (STL-OR compare real not equal), 182

OR= (STL-OR compare real equal), 182

OR> (STL-OR compare real greater than), 182

OR>= (STL-OR compare real greater than or equal), 182

ORB (STL-OR byte), 276

ORD (STL-OR dword), 276

Order numbers, 631

ORW (STL-OR word), 276

OS<> (STL-OR string compare not equal), 186

OS= (STL-OR string compare equal), 186

Output image register, 52

Outputs

coils, 148

example bit logic, 153

physical and in program, 51

set and reset bits, 149

set and reset dominant bistable, 150

writing, 52

OW< (STL-OR compare word less than), 182

OW<= (STL-OR compare word less than or equal), 182

OW<> (STL-OR compare word not equal), 182

OW= (STL-OR compare word equal), 182

OW> (STL-OR compare word greater than), 182

OW>= (STL-OR compare word greater than or equal), 182

P

Password

lost or forgotten, 136

privilege levels, 115

Password protection, 115

PC/PPI cable, 361

PID auto-tune

auto-deviation, 428

auto-hysteresis, 428

exception conditions, 430

prerequisites, 427

PV out-of-range, 431

sequence, 428

PID loop control

- loop definition table, 424
- PID Tune control panel, 431
- PID loop instruction**
 - alarm checking, 262
 - loop control types, 259
 - understanding, 256
- PID Tune control panel, 431
- Pin assignments for network connector, 361
- PLC**
 - clearing memory, 135
 - compile and run-time errors, 587
 - fatal errors, 590
 - information (hardware/firmware, error status, run/stop event log), 101
 - installation, 39
 - installation on a panel, 41
 - memory card, 76
 - non-fatal error memory locations, 590
 - system block, 107
- PLC menu**
 - Download, 68
 - Upload, 71
- PLS**
 - instruction, 236
 - Special Memory to monitor and control PWM outputs, 606
- Pointer**
 - creating pointers and using indirect address, 64
 - example of pointer offset to access data, 68
 - example of pointer to access data in a table, 67
- Power interruption (PLC)**, 114
- Power requirements**
 - calculating, 586
 - CPU, 583
 - sample, 584
- Power supply**, 583
- PPI communication**
 - changing to Freeport mode, 169
 - multi-master and multi-slave PPI networks, 358
 - port configuration in system block, 109
 - single-master networks, 358
- Previous STEP 7-Micro/WIN projects**, 83
- Program block**, 81
- Program control instructions**
 - END, STOP, and WDR,
 - FOR-NEXT loop,
 - GET_ERROR,
 - JMP-LBL,
 - SCR (Sequence control relay),
- Program editor**
 - bookmarks, 409
 - debugging and monitoring, 412
- STL status options**, 415
- types**, 87
- using**, 28
- Program instructions**
 - bit logic, 139, 141, 143, 144, 146, 147, 148, 149, 150, 151, 152, 153
 - clock, 155, 157
 - compare, 182, 186
 - convert, 188, 191, 197, 202, 205
 - counters, 208, 210, 216, 228
 - interrupts, 263
 - logical operations, 275, 276
 - math, 240, 244, 246, 248
 - move, 278, 279, 280, 282
 - program control, 282, 284, 285, 294, 296
 - shift and rotate, 297, 300
 - string, 305, 306
 - subroutine, 328, 329
 - table, 309, 310, 313
 - table find, 314
 - timer, 318, 327
- Program status**
 - building a status chart, 416
 - displaying in program editor, 412
 - executing a limited number of scans, 420
- Programs**
 - elements, 81
 - executing limited scans, 420
 - interrupt routines, 82
 - memory card, 76
 - status charts, 416
 - subroutines, 81
- Projects**
 - opening previous STEP 7-Micro/WIN projects, 83
- Proportional term, PID algorithm**, 257
- Protection class**, 512
- Pulse catch bits, digital input configuration in system block**, 112
- Pulse width modulation (PWM)**
 - cycle time, 238
 - instruction, 137
 - output, 437
 - pulse output instruction (PLS), 236
 - pulse outputs, 439
- PUT**, 160
- PWM wizard**
 - open loop motion control, 437
 - PWMx_RUN subroutine, 439
- PWM0, PWM1, PWM2 high-speed outputs (SMB67-SMB81 and SMB567-SMB571)**, 606
- PWMx_RUN**, 439

Q

Queue interrupt overflow (SMB4), 598
quick access toolbar, 21

R

R (STL-Reset), 149
-R (STL-Subtract real), 240
R_S, 197
Rated voltages, 512
RCV (receive message control SMB86-SMB94 and SMB186-SMB194), 608
READ_RTC, 155
READ_RTCX, 157
Real number values, 62
Receive instruction
 break detection, 174
 end character detection, 176
 end conditions, 172
 idle line detection, 172
 intercharacter timer, 176
 maximum character count, 177
 message timer, 177
 parity errors, 177
 start character detection, 173
 user termination, 178
Relay electrical service life, 513
Repeaters, 360
Reset-to-factory-defaults memory card, 137
Restore-to-factory-defaults memory card, 72
Restoring data after power-on, 78
RET, 328
Retentive memory, 72
Retentive ranges, system block configuration, 114
RETI, 263
RI (STL-Reset immediate), 149
ROUND, 188
RS (LAD/FBD Reset dominant bistable), 150
RS232, 368
 Freeport mode, 366
 number of communication connections, 335
 types of communication, 335
RS485
 communication overview, 354
 communication ports configuration, 109
 number of communication connections, 335
 sample network configurations, 358
 setting baud rate and port network address, 356
 types of communication, 335
RTA, 191
RTD analog inputs

 coefficient, 125
 rejection, 125
 resistor, 125
 RTD type, 125
 scale, 125
 smoothing, 125
 system block configuration, 125
RTS (STL-Real to string), 197
RUN mode, 33, 78
RUN to STOP transition
 analog output states, 123
 digital output states, 113
Run-time and PLC compile errors, 587

S

S (STL-Set), 149
S_DI, 202
S_I, 202
S_R, 202
Safety circuits, 79
Sample control program, 27
Sample network configurations, RS485 devices, 358
Saving project, 32
SB (signal board) hardware/firmware ID (SMB1050-SMB1099), 615
Scan cycle
 executing a single scan, 420
 executing multiple scans, 420
 scan times (SMW22-SMW26), 600
SCR, 285
SCRE, 285
SCRT, 285
Security, 115
SEG, 188
Selecting the network cable, 361
Service and support, 3
Set and reset dominant bistable instructions, 150
Set and reset immediate instructions, 149
SET_ADDR, 180
SET_RTC, 155
SET_RTCX, 157
Setting the CPU type, 31
SFND (STL-Find string), 306
Shift and rotate instructions
 bit (SHRB), 300
 byte, word, dword, 297
SHL_B, 297
SHL_DW, 297
SHL_W, 297
SHR_B, 297
SHR_DW, 297

- SHR_W, 297
 SHR_B, 300
 SI (STL-set immediate), 149
 Siemens technical support, 3
 Signal board (SB)
 SB BA01 specifications, 581
 Signal board (SB)
 installing and removing, 43
 SB AQ01 specifications, 577
 SB AQ01 wiring diagram, 578
 SB BA01 wiring diagram, 581
 SB CM01 specifications, 579
 SB CM01 wiring diagram, 580
 SB DT04 specifications, 574
 SB DT04 wiring diagram, 576
 Signal board ID and error registers (SMB28-SMB29), 601
 SIN (sine), 246
 SIP_ADDR, 181
 SLB (STL-Shift left byte), 297
 SLD (STL-Shift left dword), 297
 SLW (STL-Shift left word), 297
 SM (Special memory) assignments and functions, 593
 SM locations (Axis of Motion), 500
 SM memory, PWM operation, 238
 SMB0 system status bits, 595
 SMB1 instruction execution status bits, 596
 SMB1000-SMB1049 CPU hardware/firmware ID, 615
 SMB1050-SMB1099 SB (signal board)
 hardware/firmware ID, 615
 SMB1100-SMB1299 EM (expansion module)
 hardware/firmware ID, 616
 SMB130 port 1 configuration, 601
 SMB136-145 (HSC3) high-speed counter 3, 603
 SMB186-SMB194 receive message control, 608
 SMB2 Freeport receive character, 597
 SMB28-SMB29 signal board ID and error registers, 601
 SMB3 Freeport character error, 597
 SMB30 (port 0) and SMB130 (port 1) configuration, 601
 SMB34-SMB35 time interval values for timed interrupts, 602
 SMB36-45 (HSC0) high-speed counter 0, 603
 SMB4 interrupt queue overflow, run-time program error, interrupts enabled, Freeport transmitter idle, value forced, 598
 SMB46-55 (HSC1) high-speed counter 1, 603
 SMB480-SMB515 Data log status, 612
 SMB5 I/O error status bits, 598
 SMB56-65 (HSC2) high-speed counter 2, 603
 SMB567-SMB571 PWM2 high-speed output, 606
 SMB600-SMB649 Axis 0 motion control, 613
 SMB67-SMB71 PWM0, PWM1 high-speed outputs, 606
 SMB67-SMB81 and SMB567-SMB571 PWM0, PWM1, PWM2 high-speed outputs, 606
 SMB6-SMB7 CPU ID register, 599
 SMB86-SMB94 and SMB186-SMB194 receive message control, 608
 SMB8-SMB19 I/O module ID and error registers, 599
 SMW100-SMW114 System alarms, 611
 SMW22-SMW26 scan times, 600
 SMW98 I/O expansion bus - communication errors, 610
 Software debugging, 409
 Special memory assignments and functions, 593
 Specifications,
 CE approval, 509
 CPU CR40, 528
 CPU SR20, 513
 CPU SR30, 521
 CPU SR40, 528
 CPU SR60, 537
 CPU ST20, 513
 CPU ST30, 521
 CPU ST40, 528
 CPU ST60, 537
 electromagnetic compatibility (EMC), 510
 EM AE04, 557
 EM AM06, 561
 EM AQ02, 559
 EM AR02 (RTD), 569
 EM AT04, 564
 EM DR08, 549
 EM DR16, 551
 EM DR32, 551
 EM DT08, 549
 EM DT16, 551
 EM DT32, 551
 environmental conditions, 511
 general technical specifications, 509
 industrial environments, 510
 protection, 512
 rated voltages, 512
 relay electrical service life, 513
 SB AQ01, 577, 578
 SB CM01, 579
 SB DT04, 574, 576
 Specifications
 CPU CR60, 537
 SQRT (square root), 246
 SR (LAD/FBD Set dominant bistable), 150
 SRB (STL-Shift right byte), 297

- SRD (STL-Shift right dword), 297
SRW (STL-Shift right word), 297
SSCPY (STL-Copy substring), 305
SSTR_CPY, 305
Starting
 startup options, 119
 STEP 7-Micro/WIN SMART, 25
Static IP information, 344
Status
 building a status chart, 416
 displaying in program editor, 412
 executing a limited number of scans, 420
Status error (timestamp mismatch), 592
STD (STL-Sub-string to double integer), 202
STEP 7-Micro/WIN (earlier versions), 83
STEP 7-Micro/WIN SMART
 connecting with the CPU, 26
 equipment requirements, 21
 Ethernet port configuration, 344
 RUN and STOP mode, 33
Steppers in motion control, 440
STI (STL-Sub-string to integer), 202
STL
 logic stack operations, 144
 status options, 415
STL editor, 89
STOP (instruction), 294
STOP mode, 34, 78
 analog output states, 123
 digital output states, 113
 writing and forcing outputs, 419
STR (STL-Sub-string to real), 202
STR_FIND, 306
String instructions
 copy substring, 305
 Find string / character, 306
Strings
 format, 63
 representation, 63
SUB_DI, 240
SUB_I, 240
SUB_R, 240
Subroutine instructions
 Call parameter and return examples, 329
 CALL, RET, 328
Subroutines
 Axis of Motion, 460
 element of user program, 81
 guidelines, 461
 PWMx_RUN, 439
Support, 3
Suppression circuits, 49
SWAP, 280
Symbol table, 92
Symbols (symbolic addressing)
 defining global symbols, 92
 indirect addressing, 94
Synchronous updates (PWM instruction), 238
System alarms (SMW100-SMW114), 611
System block, 82
 BT01 battery signal board, 134
 CPU configuration, 107
 digital input filters, 112
 IP address of CPU, 347
 password and security, 115
 RS485/RS232 CM01 communications signal board, 133
 RTD analog input module, 125
 startup options, 119
 TC analog input module, 129
System status bits (SMB0), 595
- T**
- Table instructions
 ATT, 309
 FIFO/LIFO, 310
 FILL_N, 313
 TBL_FIND, 314
TAN (tangent), 246
TBL_FIND, 314
TC analog input module, 129
TC analog inputs
 rejection, 129
 scale, 129
 smoothing, 129
 system block configuration, 129
 TC type, 129
Technical specifications, 509
Technical support, 3
Terminal block connector, 45
Thermal zone, 36
Thermocouple
 basic operation, 132, 567
 cold junction compensation, 132, 567
 SM 1231 Thermocouple filter selection table, 567
 SM 1231 Thermocouple selection table, 567
Timed interrupt configuration (SMB34-SMB35), 602
Time-of-day
 clock instructions, 155
 extended clock instructions, 157
 protection for reads and writes, 115
Timer instructions
 BITIM, CITIM, 327

interrupts, 269
 programming tips and examples, 321
 TON, TONR, TOF, 318
 Timestamp mismatch (PC/PLC program difference), 592
 TODR (STL-Read time-of-day clock), 155
 TODRX (STL-Read time-of-day clock extended), 157
 TODW (STL-Write time-of-day clock), 155
 TODWX (STL-Write time-of-day clock extended), 157
 TOF (Off-delay timer), 318
 TON (On-delay timer), 318
 TONR (On-delay timer retentive), 318
 Tools (options)
 execution status coloring, 412
 STL status, 415
 Tools menu
 Motion control panel, 481
 PID Tune control panel, 431
 Transmission rate, 360
 Transmit instruction
 example, 179
 transmitting data, 169
 Troubleshooting S7-200 SMART hardware, 421
 TRUNC, 188

U

Uploading programs, 71
 User-defined libraries, 369
 USS protocol
 requirements for use, 371
 USS protocol instructions
 example program, 384
 using, 373
 USS_CTRL, 376
 USS_INIT, 374
 USS_RPM_x, 379
 USS_WPM_x, 381
 USS protocol library, 371
 calculating time for communications, 371
 execution errors, 383
 protocol overview, 371
 requirements, 371
 using the library, 371
 using the USS protocol instructions, 373

V

V90 drives, 22, 445, 474, 632
 Variable table, 95
 Vibration, 41

W

WAND_B, 276
 WAND_DW, 276
 WAND_W, 276
 WDR (watchdog timer reset), 294
 Wiring diagram
 EM AT04, 564
 Wiring diagrams
 CPU CR40, 534
 CPU CR60, 543
 CPU SR20, 520
 CPU SR30, 526
 CPU SR40, 534
 CPU SR60, 543
 CPU ST20, 520
 CPU ST30, 526
 CPU ST40, 534
 CPU ST60, 543
 EM AE04, 558
 EM AM06, 563
 EM AQ02, 560
 EM AR02 (RTD), 573
 EM DI08, 548
 EM DR08, 550
 EM DR16, 554
 EM DR32, 555
 EM DT08, 550
 EM DT16, 554
 EM DT32, 555
 SB AQ01, 578
 SB CM01, 580
 Wiring diagrams
 SB BA01,
 Wiring guidelines
 clearance for airflow and cooling, 36
 DIN rail, 41
 grounding, 47
 grounding and circuit, 46
 inductive loads, 49
 installation, 35
 isolation, 47
 lamp loads, 48
 separate the devices from heat, high voltage, and electrical noise, 35
 terminal block connector, 45
 Wizards
 high-speed counter (HSC), 216
 Text Display, 19
 WOR_B, 276
 WOR_DW, 276
 WOR_W, 276
 Word access, 56

Writing values
outputs, 52
writing and forcing outputs in STOP mode, 419
WXOR_B, 276
WXOR_DW, 276
WXOR_W, 276

X

XORB (STL-XOR byte), 276
XORD (STL-XOR dword), 276
XORW (STL-XOR word), 276