DST 문법

DiBA PLC 사용자는 확장자가 DST인 text file을 작성하여 E5A가 수행할 작업 내용을 조작할 수 있습니다. DST file은 메모장 등의 다양한 일반 편집기를 이용하여 수정할 수 있습니다. DST file에서 대문자/소문자 구분은 문자열 내용에서만 유효합니다. CONFIGURATION이라고 쓴 것과 conFiGuratIon이라고 쓴 것 모두 CONFIGURATION 구조지시자로 인식합니다. DST 예제에서 대문자와 소문자를 구분해 사용한 것은 사용자가 읽기 쉽게 만들려는 목적만 있습니다.

```
CONFIGURATION nameOfConf
END_CONFIGURATION
```

[DST 1] conf0.dst

[DST 1]은 유효한 DST의 가장 작은 예입니다. CONFIGURATION은 반드시 정의되어야 하며, 1개만 존재할 수 있습니다. CONFIGURATION의 이름은 달리 쓰이는 곳이 없지만, 반드시 지정해야 합니다. 그 끝은 END_CONFIGURATION으로 표시합니다.

1. DST의 지시자

DST에서 미리 정의된 지시자들을 열거합니다. 지시자들은 다른 용도로 사용될 수 없습니다. 사용자는 변수명이나 함수명 등 새로운 지시자를 생성할 때, 미리 정의된 것을 피해서 정의해야 합니다. 지시자에 대한 자세한 설명은 분류별로 나눠서 합니다. 다음 표는 지시자 이름을 알파벳 순서로 정렬하고, 각 지시자에 대한 설명이 있는 본문의 위치를 나타냅니다.

참고: 주석문은 "(*"로 시작하고, "*)"로 끝납니다. 주석문과 문자열의 내부에는 주석문을 넣을 수 없지만, DST의 다른 어느 곳이든 주석문을 넣을 수 있습니다.

지시자 이름	분류	설명	지시자 이름	분류	설명
+	작업	1.4.3.	-	작업	1.4.3.
*	작업	1.4.3.	/	작업	1.4.3.
&	작업	1.4.2.	|	작업	1.4.2.
!	작업	1.4.2.	^	작업	1.4.2.
ABS	작업	1.4.3.	ACOS	작업	1.4.3.
ADDROF	작업	1.4.5.	AND	작업	1.4.2.
ARRAY	변수	1.3.	ASIN	작업	1.4.3.
AT	변수	1.3	ATAN	작업	1.4.3.
BACKUP	작업	1.4.7.	BOOL	형	1.1.
BY	작업	1.4.6.	BYTE, USINT	형	1.1.
CASE	작업	1.4.6.	CHG_ENDIAN	작업	1.4.5.
CONCAT	작업	1.4.4.	CONFIGURATION	구조	1.2.

지시자 이름	분류	설명	지시자 이름	분류	설명
COS	작업	1.4.3.	CPU	구조	1.2
CTD	FB형	1.5.1.	CTU	FB형	1.5.1.
CTUD	FB형	1.5.1.	DATE	형	1.1.
DATE_AND_TIME, DT	형	1.1.	DELETE	작업	1.4.4.
DINT	형	1.1.	DO	작업	1.4.6.
DWORD, UDINT	형	1.1.	EDC_CRC_16ARC	작업	1.4.5.
EDC_CRC_16DDS110	작업	1.4.5.	EDC_CRC_16DECTR	작업	1.4.5.
EDC_CRC_16DNP	작업	1.4.5.	EDC_CRC_16EN13757	작업	1.4.5.
EDC_CRC_16MODBUS	작업	1.4.5.	EDC_CRC_16UMTS	작업	1.4.5.
EDC_CRC_16USB	작업	1.4.5.	EDC_CRC_32ISOHDLC	작업	1.4.5.
EDC_CRC_8DARC	작업	1.4.5.	EDC_CRC_8I4321	작업	1.4.5.
EDC_CRC_8ICODE	작업	1.4.5.	EDC_CRC_8MAXIMDOW	작업	1.4.5.
EDC_CRC_8ROHC	작업	1.4.5.	EDC_CRC_8SMBUS	작업	1.4.5.
EDC_CRC_8WCDMA	작업	1.4.5.	EDC_SUM_BYTE	작업	1.4.5.
EDC_SUM_DWORD	작업	1.4.5.	EDC_SUM_WORD	작업	1.4.5.
EDC_XOR_BYTE	작업	1.4.5.	EDC_XOR_DWORD	작업	1.4.5.
EDC_XOR_WORD	작업	1.4.5.	ELSE	작업	1.4.6.
ELSIF	작업	1.4.6.	END_CASE	작업	1.4.6.
END_CONFIGURATION	구조	1.2	END_FOR	작업	1.4.6.
END_FUNCTION	구조	1.2.	END_FUNCTION_BLOCK	구조	1.2.
END_IF	작업	1.4.6.	END_PROGRAM	구조	1.2.
END_REPEAT	작업	1.4.6.	END_RESOURCE	구조	1.2.
END_STRUCT	형	1.1.	END_TYPE	형	1.1.
END_VAR	변수	1.3.	END_WHILE	작업	1.4.6.
EQ, =	작업	1.4.2.	ETH_1	구조	1.2.
ETH_2	구조	1.2.	EXIT	작업	1.4.6.
EXP	작업	1.4.3.	EXPT, **	작업	1.4.3.
F_TRIG	FB형	1.5.1.	FALSE	값	1.1.
FIND	작업	1.4.4.	FOR	작업	1.4.6.
FUNCTION	구조	1.2.	FUNCTION_BLOCK	구조	1.2.
GE, >=	작업	1.4.2.	GET_BOOL	작업	1.4.5.
GET_BYTE	작업	1.4.5.	GET_DATE	작업	1.4.5.
GET_DINT	작업	1.4.5.	GET_DT	작업	1.4.5.

지시자 이름	분류	설명	지시자 이름	분류	설명
GET_DWORD	작업	1.4.5.	GET_INT	작업	1.4.5.
GET_REAL	작업	1.4.5.	GET_SINT	작업	1.4.5.
GET_TIME	작업	1.4.5.	GET_TOD	작업	1.4.5.
GET_WORD	작업	1.4.5.	GT, >	작업	1.42.
IF	작업	1.4.6.	INSERT	작업	1.4.4.
INT	형	1.1.	LE, <=	작업	1.42.
LEFT	작업	1.4.4.	LEN	작업	1.4.4.
LIMIT	작업	1.4.3.	LN	작업	1.4.3.
LOG	작업	1.4.3.	LT, <	작업	2.2
MAX	작업	1.4.3.	MID	작업	1.4.4.
MIN	작업	1.4.3.	MODULO	작업	1.4.3.
MUX	작업	1.4.6.	NE, <>	작업	1.4.2.
NOT	작업	1.4.2.	OF	변수, 작업	1.3, 1.4.6.
ON	구조	1.2	OR	작업	1.4.2.
PID	FB형	1.5.2.	PROGRAM	구조	1.2.
R_TRIG	FB형	1.5.1.	REAL	형	1.1.
REPEAT	작업	1.4.6.	REPLACE	작업	1.4.4.
RESOURCE	구조	1.2.	RETURN	작업	1.4.6.
RIGHT	작업	1.4.4.	ROL	작업	1.4.3.
ROR	작업	1.4.3.	RS	FB형	1.5.1.
RTC	FB형	1.5.1.	SEE_NW_DOMAIN	작업	1.4.7.
SEE_NW_IP	작업	1.4.7.	SEE_NW_MODE	작업	1.4.7.
SEE_NW_SSID	작업	1.4.7.	SEL	작업	1.4.6.
SEMA	FB형	1.5.1.	SER_1	구조	1.2.
SER_2	구조	1.2.	SER_3	구조	1.2.
SER_4	구조	1.2.	SER_5	구조	1.2.
SER_6	구조	1.2.	SET_BOOL	작업	1.4.5.
SET_BYTE	작업	1.4.5.	SET_DATE	작업	1.4.5.
SET_DINT	작업	1.4.5.	SET_DT	작업	1.4.5.
SET_DWORD	작업	1.4.5.	SET_INT	작업	1.4.5.
SET_REAL	작업	1.4.5.	SET_SINT	작업	1.4.5.
SET_TIME	작업	1.4.5.	SET_TOD	작업	1.4.5.
SET_WORD	작업	1.4.5.	SHL	작업	1.4.3.

지시자 이름	분류	설명	지시자 이름	분류	설명
SHR	작업	1.4.3.	SIN	작업	1.4.3.
SINT	형	1.1.	SIZEOF	작업	1.4.5.
SQRT	작업	1.4.3.	SR	FB형	1.5.1.
STR_FROM_BOOL	작업	1.4.4.	STR_FROM_BYTE	작업	1.4.4.
STR_FROM_DATE	작업	1.4.4.	STR_FROM_DINT	작업	1.4.4.
STR_FROM_DT	작업	1.4.4.	STR_FROM_DWORD	작업	1.4.4.
STR_FROM_INT	작업	1.4.4.	STR_FROM_REAL	작업	1.4.4.
STR_FROM_SINT	작업	1.4.4.	STR_FROM_TIME	작업	1.4.4.
STR_FROM_TOD	작업	1.4.4.	STR_FROM_WORD	작업	1.4.4.
STR_TO_BOOL	작업	1.4.4.	STR_TO_BYTE	작업	1.4.4.
STR_TO_DATE	작업	2.4	STR_TO_DINT	작업	1.4.4.
STR_TO_DT	작업	2.4	STR_TO_DWORD	작업	1.4.4.
STR_TO_INT	작업	2.4	STR_TO_REAL	작업	1.4.4.
STR_TO_SINT	작업	1.4.4.	STR_TO_TIME	작업	1.4.4.
STR_TO_TOD	작업	1.4.4.	STR_TO_WORD	작업	2.4
STRING	형	1.1.	STRUCT	형	1.1.
TAN	작업	1.4.3.	TASK	구조	1.2.
THEN	작업	1.4.6.	TIME	형	1.1.
TIME_OF_DAY, TOD	형	1.1.	TO	작업	1.4.6.
TOF	FB형	1.5.1.	TON	FB형	1.5.1.
TP	FB형	1.5.1.	TRUE	값	1.1.
TYPE	형	1.1.	UINT, WORD	형	1.1
UNTIL	작업	1.4.6.	VAR	변수	1.3.
VAR_GLOBAL	변수	1.3.	VAR_INPUT	변수	1.3.
WA_ABLE_GEN	작업	1.4.7.	WA_ABLE_OUT	작업	1.4.7.
WA_DISABLE_GEN	작업	1.4.7.	WA_DISABLE_OUT	작업	1.4.7.
WA_ENABLE_GEN	작업	1.4.7.	WA_ENABLE_OUT	작업	1.4.7.
WHILE	작업	1.4.6.	XOR	작업	1.42.

다음 표에 있는 지시자는 표준(IEC 61131-3)에서는 정의되었으나 DST에서는 사용하지 않는 지시자로 사용자가 다른 의미로 정의할 수 없도록 제한합니다.

지시자 이름	분류	지시자 이름	분류	지시자 이름	분류	지시자 이름	분류
LINT	형	LREAL	형	LWORD	형	ULINT	형
VAR_ACCESS	변수	VAR_EXTERNAL	변수	VAR_IN_OUT	변수	VAR_OUTPUT	변수

DST file은 행 단위로 입력합니다. 구조지시자 및 변수지시자의 일부를 제외하고 대부분의 내용은 행 구분자를 사용해야 합니다. DST file 전체에서 동일한 의미로 사용되는 지시자를 설명합니다.

;	행 구분자. 작업 행의 끝을 표시
,	항목 구분자. 작업 행 내에서 동일한 수준의 복수 항목이 사용되는 경우, 항목과 항목의 사이를 표시
:=	값 복사. 지시자의 오른쪽에는 연산 결과를 가지는 식을 배치, 왼쪽에는 결과 값을 넣을 변수를 배치

1.1. 형/값지시자

형/값지시자 중 DST에서 사용하는 것만 모아서 다음에 정리합니다. 형지시자와 값지시자가 다른 경우, 값지시자를 추가로 표시합니다.

지시자 이름	설명
FALSE	값을 0으로 지정
TRUE	값을 1로 지정
BOOL	변수의 형을 BOOL로 지정
BYTE, USINT	변수의 형을 BYTE로 지정
WORD, UINT	변수의 형을 WORD로 지정
DWORD, UDINT	변수의 형을 DWORD로 지정
SINT	변수의 형을 SINT로 지정
INT	변수의 형을 INT로 지정
DINT	변수의 형을 DINT로 지정
REAL	변수의 형을 REAL로 지정
DATE (값지시자: D)	변수의 형을 DATE로 지정, 값의 형을 DATE로 지정 (값의 예) d#2019-1-2
DT, DATE_AND_TIME (값지시자: DT)	변수의 형을 DT로 지정, 값의 형을 DT로 지정 (값의 예 1) dt#2019-1-2-3:4:5.678 (값의 예 2) dt#2019-1-2-3:4:5
TIME (값지시자: T)	변수의 형을 TIME으로 지정, 값의 형을 TIME으로 지정 (값의 예 1) t#1d2h3m4s567ms (값의 예 2) t#800ms (값의 예 3) t#9h
TOD, TIME_OF_DAY (값지시자: TOD)	변수의 형을 TOD로 지정, 값의 형을 TOD로 지정 (값의 예 1) tod#1:2:3.456 (값의 예 2) tod#7:8:9

지시자 이름	설명
STRING	변수의 형을 STRING로 지정 (값의 예) 'AbCd? EF ! ^&*@lr' (특수 문자) ' = '$'', $ = '$$', Line Feed = '$L' or '$l', Carriage Return = '$R' or '$r'
TYPE	사용자 형 선언 개시
END_TYPE	사용자 형 선언 종료
STRUCT	구조체 선언 개시
END_STRUCT	구조체 선언 종료

BOOL부터 REAL까지 숫자를 나타내는 형의 값은 다음과 같이 표현할 수 있습니다. 숫자의 자릿수 등을 사용자가 읽기 쉽도록 '_'를 삽입할 수 있습니다. 숫자 표현에서는 대소문자를 구분하지 않습니다.

숫자 표현	표현 예
정수 (진수를 지정하지 않으면 10진수로 인식)	0, 1, -2, 3_456(= 3456)
실수	0.0, 1.2, -3.456_789, 1.2E3, 4.5E-6, -7.8E-9
2진수	2#1111_0000(= 240), 2#10(= 2)
8진수	8#73(= 59), 8#10_00(= 512)
10진수	10#12_345(= 12345)
16진수	16#FEDC(= 65244), 16#89A(= 2202)

E5A가 지원하는 기본 변수형은 13가지입니다.

변수형	Byte 크기	Bit 크기	최솟값	최댓값
BOOL	-	1	0	1
BYTE USINT	1	8	0	255
SINT	1	8	-128	127
WORD UINT	2	16	0	65,535
INT	2	16	-32,768	32,767
DWORD UDINT	4	32	0	4,294,967,295
DINT	4	32	-2,147,483,648	2,147,483,647
REAL	4	32	-3.4e38	3.4e38
TIME	8	64	-	-
TOD	8	64	tod#0:0:0.0	tod#23:59:59.999
DATE	8	64	d#1900-1-1	-
DT	8	64	dt#1900-1-1-0:0:0.0	-
STRING	64(지정 가능)	-	-	-

사용자가 새로운 형을 선언하는 내용을 설명합니다. TYPE은 CONFIGURATION 내에서만 사용할 수 있고, STRUCT는 TYPE 내에서만 사용할 수 있습니다. TYPE 내의 변수들은 같은 주소에 겹쳐서 배치됩니다. STRUCT를 사용하면 변수들이 일렬로 이어지는 주소에 서로 겹치지 않도록 배치됩니다. TYPE과 STRUCT를 조합하여 사용자가 원하는 자리에 변수를 배치할 수 있습니다. TYPE을 선언하면서 지정한 새로운 형의 이름은 이후 변수를 선언할 때 변수형으로 사용할 수 있습니다.

사용자 형 선언 형식	TYPE {새로운 형의 이름}: {변수 선언}; {변수 선언}; {구조체 선언} {구조체 선언} END_TYPE {새로운 형의 이름}은 사용자가 지정합니다. DST file 전체에서 유일한 이름을 사용해야 합니다.
ENUM 선언 형식	TYPE {새로운 형의 이름}: ({ENUM명}, {ENUM명},); END_TYPE ENUM으로 선언하면, 새로운 형은 DINT와 동일한 크기 및 범위를 가집니다. {ENUM명}은 사용자가 지정합니다. DST file 전체에서 유일한 이름을 사용해야 합니다. 값을 지정하지 않은 경우, 처음의 값은 0이고, 이후 1씩 더한 값을 가집니다. {ENUM명} := {지정 값},의 형식으로 값을 지정할 수 있습니다. 이후의 ENUM은 1씩 더한 값을 가집니다.
구조체 선언 형식	STRUCT {변수 선언}; {변수 선언}; END_STRUCT
1 byte의 길이와 10 byte의 버퍼를 묶은 형을 선언한 예	TYPE T_UDF: STRUCT length : BYTE; buffer : ARRAY [10] OF BYTE; END_STRUCT END_TYPE

IP V4 address를 다양한 형태로 묶은 형을 선언한 예	TYPE T_IPv4:
	  ip_value : UDINT;
	  ip_byte : ARRAY [4] OF BYTE;
	  ip_class_a : USINT;
	  ip_class_b : UINT;
	END_TYPE

T_IPv4를 사용하는 예	PROGRAM aTestPgm
	  VAR
	  vIpAddress : T_IPv4;
	  vClassA : USINT;
	  END_VAR
	  vIpAddress.ip_value := 16#f11f001a;
	  vClassA := vIpAddress.ip_class_a;
	END_PROGRAM

사용자는 변수들과 작업 내용의 조합으로도 새로운 변수형을 생성할 수 있습니다. E5A가 가지는 PLC의 특성으로 설정을 기억하면서 단계적 작업 진행 및 작업과 실행 세부 내용의 분리 등의 한계를 극복하기 위해 필요합니다. 통신 기능의 운용을 위해 만들어진 내장 FB(Function Block)를 사용하여 쉽고 간단하게 작업을 처리한 예에서 그 효용을 볼 수 있습니다.

구조지시자인 FUNCTION_BLOCK을 사용하여 구축한 FB를 전역 변수의 형으로 지정하면 FB변수가 만들어집니다.

1.2. 구조지시자

DST file은 text file이라는 기본 바탕 위에 CONFIGURATION, PROGRAM, FUNCTION_BLOCK, FUNCTION의 구조를 올려서 구축됩니다. CONFIGURATION 구조는 DST file 전체에서 반드시 1개가 존재해야 합니다. PROGRAM 구조, FUNCTION_BLOCK 구조, FUNCTION 구조 각각은 필요에 따라 구축하므로 존재하지 않거나, 여러 개가 존재할 수 있습니다. 다음은 DST file의 모든 구조 요소가 하나씩 포함된 구조 형식의 예입니다.

```
CONFIGURATION {CONF의 이름}
  RESOURCE {resource 영역의 이름} ON {resource의 이름}
    TASK {task의 이름} (SINGLE := TRUE, PRORITY := 1);
    PROGRAM {program의 별명} WITH {task의 이름} : {PROG의 이름 혹은 FB변수의 이름} ();
  END_RESOURCE
END_CONFIGURATION

PROGRAM {PROG의 이름}
END_PROGRAM

FUNCTION_BLOCK {FB의 이름}
END_FUNCTION_BLOCK

FUNCTION {FUN의 이름} : {출력변수형}
END_FUNCTION
```

CONFIGURATION 구조에서는 사용자 형 정의, 전역 변수 선언, 자원(RESOURCE) 배정 등이 가능합니다. 자원 배정을 통해 실행될 작업객체를 정의합니다. 개별 작업객체는 서로 독립적이며, 실행되는 동안 E5A를 단독으로 점유합니다. 다음은 CONFIGURATION 구조를 구축하기 위해 필요한 내용을 정리한 것입니다.

{CONF의 이름}은 사용자가 지정합니다. 사용하는 곳은 없지만 반드시 입력해야 합니다. CONFIGURATION은 DST file 내에서 반드시 한 번 구축되어야 합니다.
{resource 영역의 이름}은 사용자가 지정합니다. 이후 작업 영역에서 자원의 영역을 지정하는 데에 사용합니다. DST file 전체에서 유일한 이름을 사용해야 합니다.
{resource의 이름}은 resource를 선택하는 다음의 구조지시자 중 하나를 사용합니다. 하나의 resource를 복수의 RESOURCE에 지정할 수는 없습니다. ● CPU = E5A를 지정합니다. ● ETH_1 = server port를 지정합니다. ● ETH_2 = client port를 지정합니다. ● SER_1 = RS485 port를 지정합니다. ● SER_2 = 지정된 것이 없습니다. 예비용입니다. ● SER_3 = 지정된 것이 없습니다. 예비용입니다. ● SER_4 = 지정된 것이 없습니다. 예비용입니다. ● SER_5 = 지정된 것이 없습니다. 예비용입니다. ● SER_6 = 지정된 것이 없습니다. 예비용입니다.
TASK는 RESOURCE 내에서 여러 개를 선언할 수 있습니다. 특성은 1회성과 주기성이 있습니다. 1회성으로 선언하려면 SINGLE을 TRUE로 지정하고, 주기성으로 선언하려면 INTERVAL을 지정합니다. SINGLE은 BOOL 형이고, INTERVAL은 TIME 형입니다. TASK가 여러 개 선언되었을 때, 동시에 실행 조건이 만족하는 경우도 발생할 수 있습니다. PRIORITY는 이 경우의 실행 우선순위를 판단하는 값입니다. 설정 가능한 범위는 1~9입니다. {task의 이름}은 사용자가 지정합니다. 해당 RESOURCE 내에서만 유효합니다.
PROGRAM은 TASK로 만들어진 실행 조건과 실행 우선순위를 {PROG의 이름 혹은 FB변수의 이름}과 묶어서 작업객체를 구축합니다. {program의 별명}은 사용자가 지정합니다. 사용하는 곳이 없으며 지정하지 않아도 됩니다. {PROG의 이름 혹은 FB변수의 이름}은 PROGRAM 구조의 {PROG의 이름}이나 FB형을 전역 변수에 지정한 {FB변수의 이름}을 지정합니다. 뒤에 오는 괄호는 PROG 혹은 FB가 호출된다는 것을 뜻할 뿐이지, 파라미터를 입력받기 위한 용도가 아닙니다. 그러므로 사용자는 PROGRAM행에서 호출하는 FB에게 주는 파라미터를 다른 작업에서 미리 설정해야 합니다.

PROGRAM 구조, FUNCTION_BLOCK 구조, FUNCTION 구조에서는 지역 변수 선언, 작업 내역 작성 등이 가능합니다. 다음은 각 구조를 구축하기 위해 필요한 내용을 정리한 것입니다.

PROGRAM 구조에서 {PROG의 이름}은 사용자가 지정합니다. DST file 전체에서 유일한 이름을 사용해야 합니다. PROGRAM은 파라미터(입력 변수)나 출력 변수가 없습니다. 구축된 작업은 PROG이라고 정의합니다. PROG의 호출 형식은 {PROG의 이름}();입니다.

FUNCTION_BLOCK 구조에서 {FB의 이름}은 사용자가 지정합니다. DST file 전체에서 유일한 이름을 사용해야 합니다. 구축된 작업은 FB라고 정의하고, 생성된 형은 FB형이라고 정의합니다. FB형은 전역 변수로만 선언이 가능하고, 이렇게 만들어진 변수를 FB변수라고 정의합니다. FB변수의 모든 지역(내부) 변수들은 파라미터(입력변수)나 출력 변수로 사용될 수 있습니다. FB의 호출 형식은 {FB변수명}({FB 지역변수명} := {파라미터 값});입니다. 파라미터를 입력하지 않을 수도 있고, 여러 개를 입력할 수도 있습니다. 또한 작업 행을 분리하여 FB의 호출 전에 {FB변수명}.{FB 지역변수명} := {파라미터 값};과 같이 입력할 수도 있습니다.
FUNCTION 구조에서 {FUN의 이름}은 사용자가 지정합니다. DST file 전체에서 유일한 이름을 사용해야 합니다. {출력변수형}은 기본 변수형 중 하나를 선택해야 합니다. 출력 변수의 이름은 {FUN의 이름}과 동일합니다. 구축된 작업은 FUN이라고 정의합니다. FUN은 파라미터(입력 변수), 지역(내부) 변수, 출력 변수를 가집니다. 입력 변수와 내부 변수는 선언하지 않을 수 있으나 출력 변수는 하나를 반드시 선언해야 합니다. FUN의 호출 형식은 {변수명} := {FUN의 이름}({파라미터 값});입니다. PROG이나 FB와 달리 반드시 반환 값을 작업에서 접근 가능한 변수에게 넘겨줘야 합니다. 파라미터는 FUNCTION 구조에서 선언한 입력 변수와 개수 및 순서가 일치해야 합니다.

구축된 작업은 상하관계를 형성합니다. 최상위에 작업객체가 있고, 차례로 PROG, FB, FUN이 위치합니다. 호출하려는 작업에서 동위의 작업과 하위의 작업을 호출할 수 있습니다. 단, 동위 작업의 호출이 돌아서 자신을 호출하는 것(순환 호출)은 금지됩니다.

1.3. 변수지시자

DST file에서 정의할 수 있는 변수의 종류는 전역 변수, 지역 변수, 입력 변수입니다. 변수의 종류는 변수가 선언되는 영역에 의해 결정됩니다. 변수 영역을 우선 설명하고, 다음으로 변수 선언을 설명합니다.

전역 변수 영역의 형식	VAR_GLOBAL {변수 선언}; END_VAR
전역 변수 영역 설명	전역 변수 영역은 CONFIGURATION 구조와 RESOURCE 구조 내에서만 선언이 가능합니다. CONFIGURATION 구조 내에서 선언한 경우, DST file의 모든 작업 영역에서 {변수명}으로 참조가 가능합니다. RESOURCE 구조 내에서 선언한 경우, DST file의 모든 작업 영역에서 {resource 영역의 이름}.{변수명}으로 참조가 가능합니다.
지역 변수 영역의 형식	VAR {변수 선언}; END_VAR

지역 변수 영역 설명	지역 변수 영역은 PROGRAM 구조, FUNCTION_BLOCK 구조, FUNCTION 구조에서만 선언이 가능합니다. 일반적으로 지역 변수는 그 지역 변수가 선언된 구조 내의 작업에서만 {변수명}으로 참조가 가능합니다. 예외적으로 FB변수의 지역 변수는 DST file의 모든 작업 영역에서 {FB변수명}.{변수명}으로 참조가 가능합니다. 만약 FB변수가 RESOURCE 구조 내에 선언되었다면 {resource 영역의 이름}.{FB변수명}.{변수명}으로 참조해야 합니다.
입력 변수 영역의 형식	VAR_INPUT {변수 선언}; END_VAR
입력 변수 영역 설명	입력 변수 영역은 FUNCTION 구조에서만 선언이 가능합니다. 입력 변수는 그 입력 변수가 선언된 구조 내의 작업에서만 {변수명}으로 참조가 가능합니다. 다른 PROG이나 FB에서 FUN를 호출할 때, 모든 입력 변수는 선언된 순서대로 파라미터로써 입력되어야 합니다.

{변수 선언}은 0개 이상의 행으로 선언할 수 있습니다. 기본 형식은 {변수명} : {변수형};이고, 사용의 편의를 위해 다양한 기능을 가지고 있습니다. 각 기능에 대한 제약 사항은 E5A의 제한된 자원을 안정적으로 사용하기 위해 아래 기술된 것 외에도 더 존재할 수 있습니다.

변수 선언의 형식	{변수명} : {변수형} ; {변수명}은 사용자가 지정합니다. 변수의 참조 범위 내에서 유일한 이름을 사용해야 합니다. {변수형}은 기본 변수형, 사용자 형, FB형 중 하나를 사용할 수 있습니다. 단, FB형은 변수 선언이 전역 변수 영역에 포함될 때만 사용가능합니다. {변수형}이 STRING인 경우, STRING(10)과 같이 최대 길이를 지정할 수도 있습니다. 길이를 지정하지 않은 경우, 기본으로 64 byte가 최대 길이로 지정됩니다. (사용 예) vInt1 : INT; (사용 예) vStr1 : STRING; (사용 예) vStr2 : STRING(30);
복수 변수 선언 기능	{변수명1}, {변수명2}, {변수명3} : {변수형} ; 동일한 변수형으로 변수명을 여러 개 선언할 수 있습니다. (사용 예) vInt1, vInt2 : INT;
배열 선언 기능	{변수형} : ARRAY[{배열의 크기}] OF {변수형} ; 형식 1에서 배열 선언이 추가된 형식입니다. STRING을 제외한 모든 변수형에 대해 사용가능합니다. 배열은 1차부터 10차까지 선언이 가능합니다. 차수의 구분은 ","를 사용합니다. (사용 예) vInt1 : ARRAY[6] OF INT; (사용 예) vInt2 : ARRAY[2,3,4] OF INT;

초기값 지정 기능	{변수명} : {변수형} := {초기값} ; 기본 변수형에 대해 초기값을 지정할 수 있습니다. 사용자 형과 FB형은 초기값을 지정할 수 없습니다. (사용 예) vInt1 : INT := 0;
범위 지정 기능	{변수명} : {변수형} ({최솟값} .. {최댓값}) ; STRING을 제외한 기본 변수형에 대해 범위를 지정할 수 있습니다. (사용 예) vInt1 : INT(-2000..8000);
주소 지정 기능	{변수명} AT {변수의 주소} : {변수형} ; 전역 변수 영역에서 모든 변수에 대해 주소를 지정할 수 있습니다. {변수의 주소}로 사용가능한 메모리 영역은 입력, 출력, 내부 등이 모두 가능합니다. 주소의 크기는 지정할 변수형과 동일한 크기를 가져야 합니다. 단, STRING, 사용자 형, FB형은 byte 주소를 사용하여 주소를 지정합니다. (사용 예) gInt1 AT %IW16 : INT; (사용 예) gStr1 AT %MB1024 : STRING;

1.4. 작업지시자

PROGRAM, FUNCTION_BLOCK, FUNCTION 구조 내에서 변수 영역을 제외한 나머지는 작업 영역입니다.

1.4.1. 호출

작업 영역은 행 단위로 구분하며, 식에 의한 연산을 수행하거나 PROG, FB, FUN 등을 호출할 수 있습니다.

{PROG의 이름}();	PROG의 이름으로 호출. 파라미터는 지정할 수 없습니다. 호출 작업이 단독으로 1행을 구성합니다.
{FB변수명} ({FB지역변수명} := {값});	FB변수명으로 호출. 파라미터는 FB의 내에 선언한 지역변수에 대해 값을 지정하는 형식으로 입력할 수 있습니다. 파라미터의 개수나 순서는 특별한 제한이 없습니다. 파라미터가 복수인 경우, ","로 구분합니다. 호출 작업이 단독으로 1행을 구성합니다.
{FUN의 이름} ({파라미터 값})	FUN의 이름으로 호출. 파라미터는 FUN 내에 선언한 입력 변수의 순서 및 개수와 일치해야 합니다. 파라미터가 복수인 경우, ","로 구분합니다.

1.4.2. 논리 연산

식에 사용할 수 있는 논리 연산용 작업지시자를 설명합니다.

&	1 Bit AND. BOOL 연산으로 AND를 수행한 결과 값을 구합니다. (사용 예) TRUE & FALSE, 결과 값 FALSE
|	1 Bit OR. BOOL 연산으로 OR를 수행한 결과 값을 구합니다. (사용 예) TRUE | FALSE, 결과 값 TRUE
!	1 Bit NOT. BOOL 연산으로 NOT을 수행한 결과 값을 구합니다. (사용 예) ! TRUE, 결과 값 FALSE
^	1 Bit XOR. BOOL 연산으로 XOR를 수행한 결과 값을 구합니다. (사용 예) TRUE ^ FALSE, 결과 값 TRUE
AND	32 Bit AND. DWORD 연산으로 AND를 수행한 결과 값을 구합니다. (사용 예) 2#1010 AND 2#1100, 결과 값 16#8
OR	32 Bit OR. DWORD 연산으로 OR를 수행한 결과 값을 구합니다. (사용 예) 2#1010 OR 2#1100, 결과 값 16#E
NOT	32 Bit NOT. DWORD 연산으로 NOT을 수행한 결과 값을 구합니다. (사용 예) NOT 2#1010, 결과 값 16#FFFF_FFF5
XOR	32 Bit XOR. DWORD 연산으로 XOR를 수행한 결과 값을 구합니다. (사용 예) 2#1010 XOR 2#1100, 결과 값 2#0110
>=, GE	크거나 같음. 지시자의 오른쪽보다 왼쪽의 값이 크거나 같으면 TRUE, 아니면 FALSE로 결과 값을 구합니다. (사용 예) 1 >= 2, 결과 값 FALSE
>, GT	큼. 지시자의 오른쪽보다 왼쪽의 값이 크면 TRUE, 아니면 FALSE로 결과 값을 구합니다. (사용 예) 1 GT 2, 결과 값 FALSE
<=, LE	작거나 같음. 지시자의 오른쪽보다 왼쪽의 값이 작거나 같으면 TRUE, 아니면 FALSE로 결과 값을 구합니다. (사용 예) 1 <= 2, 결과 값 TRUE
<, LT	작음. 지시자의 오른쪽보다 왼쪽의 값이 작으면 TRUE, 아니면 FALSE로 결과 값을 구합니다. (사용 예) 1 LT 2, 결과 값 TRUE

=, EQ	같음. 지시자의 오른쪽과 왼쪽의 값이 동일하면 TRUE, 아니면 FALSE로 결과 값을 구합니다. (사용 예) 1 = 2, 결과 값 FALSE
<>, NE	다름. 지시자의 오른쪽과 왼쪽의 값이 다르면 TRUE, 아니면 FALSE로 결과 값을 구합니다. (사용 예) 1 NE 2, 결과 값 TRUE

1.4.3. 산술 연산

식에 사용할 수 있는 산술 연산용 작업지시자를 설명합니다.

+	더하기. (사용 예) 1 + 2, 결과 값 3
-	빼기. (사용 예) 1 - 2, 결과 값 -1
*	곱하기. (사용 예) 1 * 2, 결과 값 2
/	나누기. (사용 예) 1 / 2, 결과 값 0.5
MODULO	나머지. (사용 예) 5 MODULO 4, 결과 값 1
**, EXPT	자승. (사용 예) 5 ** 2, 결과 값 25
ABS({파라미터1})	절대 값. (사용 예) ABS(-5), 결과 값 5
SQRT({파라미터1})	제곱근. (사용 예) SQRT(4), 결과 값 2
EXP({파라미터1})	e의 승. (사용 예) EXP(1), 결과 값(근사) 2.71828
LN({파라미터1})	자연로그. (사용 예) LN(EXP(1)), 결과 값 1
LOG({파라미터1})	로그. (밑수는 10) (사용 예) LOG(10), 결과 값 1
MAX({파라미터1}, ...)	최댓값. 파라미터는 숫자 형으로 2개 이상 입력해야 합니다. 개수 제한은 없습니다. 입력된 값 중 가장 큰 값을 반환합니다. (사용 예) MAX(2, 3, 4), 결과 값 4 (사용 예) MAX(2, 1, 4), 결과 값 4

MIN({파라미터1}, ...)	최솟값. 파라미터는 숫자 형으로 2개 이상 입력해야 합니다. 개수 제한은 없습니다. 입력된 값 중 가장 작은 값을 반환합니다. (사용 예) MIN(2, 3, 4), 결과 값 2 (사용 예) MIN(2, 1, 4), 결과 값 1
LIMIT({파라미터1}, {파라미터2}, {파라미터3})	값 제한. {파라미터1}은 최솟값이고, {파라미터2}는 입력 값, {파라미터3}은 최댓값입니다. 입력 값을 최솟값과 최댓값으로 제한하여 결과 값으로 반환합니다. (사용 예) LIMIT(2, 3, 4), 결과 값 3 (사용 예) LIMIT(2, 1, 4), 결과 값 2
ROL({파라미터1}, {파라미터2})	Rotate Left. DWORD 연산으로 입력 값을 지정한 수만큼 왼쪽으로 회전시킵니다. {파라미터1}은 입력 값, {파라미터2}는 이동 bit 수입니다. (사용 예) ROL(1, 2), 결과 값 4
ROR({파라미터1}, {파라미터2})	Rotate Right. DWORD 연산으로 입력 값을 지정한 수만큼 오른쪽으로 회전시킵니다. {파라미터1}은 입력 값, {파라미터2}는 이동 bit 수입니다. (사용 예) ROR(1, 2), 결과 값 16#4000_0000
SHL({파라미터1}, {파라미터2})	Shift Left. DWORD 연산으로 입력 값을 지정한 수만큼 왼쪽으로 이동시킵니다. {파라미터1}은 입력 값, {파라미터2}는 이동 bit 수입니다. (사용 예) SHL(1, 2), 결과 값 4
SHR({파라미터1}, {파라미터2})	Shift Right. DWORD 연산으로 입력 값을 지정한 수만큼 오른쪽으로 이동시킵니다. {파라미터1}은 입력 값, {파라미터2}는 이동 bit 수입니다. (사용 예) SHR(1, 2), 결과 값 0
ACOS({파라미터1})	Arc-cosine. {파라미터1}은 –1~1 사이의 값을 입력해야 합니다. 결과 값은 0~180 [°] 범위에서 구해집니다. (사용 예) ACOS(0.5), 결과 값 60
ASIN({파라미터1})	Arc-sine. {파라미터1}은 –1~1 사이의 값을 입력해야 합니다. 결과 값은 -90~90 [°] 범위에서 구해집니다. (사용 예) ASIN(0.5), 결과 값 30
ATAN({파라미터1})	Arc-tangent. 결과 값은 -90~90 [°] 범위에서 구해집니다. (사용 예) ATAN(0.5), 결과 값(근사) 26.565

COS({파라미터1})	Cosine. {파라미터1}은 [°] 단위의 값을 입력해야 합니다. 결과 값은 –1~1 범위에서 구해집니다. (사용 예) COS(50), 결과 값(근사) 0.643
SIN({파라미터1})	Sine. {파라미터1}은 [°] 단위의 값을 입력해야 합니다. 결과 값은 -1~1 범위에서 구해집니다. (사용 예) SIN(50), 결과 값(근사) 0.766
TAN({파라미터1})	Tangent. {파라미터1}은 [°] 단위의 값을 입력해야 합니다. (사용 예) TAN(50), 결과 값(근사) 1.192

1.4.4. 문자열 연산

CONCAT({파라미터1}, ...)	이어붙이기. 파라미터는 문자열 형으로 2개 이상 입력해야 합니다. 개수 제한은 없습니다. 입력 순서대로 이어붙인 문자열을 반환합니다. (사용 예) CONCAT('Ab', ' Cd'), 결과 값 'Ab Cd'
DELETE({파라미터1}, {파라미터2}, {파라미터3})	잘라버리기. 입력 문자열에서 지정 위치부터 지정 길이만큼 잘라버린 문자열을 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 자를 위치, {파라미터3}은 자를 길이입니다. 위치는 문자열 내에 있어야 합니다. 길이가 음수거나 위치+길이가 문자열을 벗어나면, 문자열의 끝까지 잘라 버려집니다. (사용 예) DELETE('Ab Cd', 2, -1), 결과 값 'Ab'
FIND({파라미터1}, {파라미터2})	찾기. 입력 문자열에서 검색 문자열과 일치하는 첫 위치를 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 검색 문자열입니다. (사용 예) FIND('Ab Bb ', 'b '), 결과 값 1
INSERT({파라미터1}, {파라미터2}, {파라미터3})	끼워 넣기. 입력 문자열의 지정 위치에 삽입 문자열을 끼워 넣은 문자열을 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 삽입 문자열, {파라미터3}은 위치입니다. (사용 예) INSERT('Ab Cd', ' x', 2), 결과 값 'Ab x Cd'
LEFT({파라미터1}, {파라미터2})	왼쪽에서 잘라내기. 입력 문자열의 왼쪽부터 지정 길이만큼 잘라낸 문자열을 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 길이입니다. 길이가 음수거나 문자열을 벗어나면, 문자열의 끝까지 잘라냅니다. (사용 예) LEFT('Ab Cd', 2), 결과 값 'Ab'

LEN({파라미터1})	길이[byte]구하기. 입력 문자열의 길이를 반환합니다. (사용 예) LEN('Ab Cd'), 결과 값 5
MID({파라미터1}, {파라미터2}, {파라미터3})	중간에서 잘라내기. 입력 문자열의 지정 위치부터 지정 길이만큼 잘라낸 문자열을 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 위치, {파라미터3}은 길이입니다. 길이가 음수거나 위치+길이가 문자열을 벗어나면, 문자열의 끝까지 잘라냅니다. (사용 예) MID('Ab Cd', 2, 2), 결과 값 ' C'
REPLACE({파라미터1}, {파라미터2}, {파라미터3}, {파라미터4})	바꿔 넣기. 입력 문자열의 지정 위치부터 지정 길이만큼을 삽입 문자열로 바꾼 문자열을 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 삽입 문자열, {파라미터3}은 위치, {파라미터4}는 길이입니다. 길이가 음수거나 위치+길이가 문자열을 벗어나면, 문자열의 끝까지 잘라버립니다. (사용 예) REPLACE('Ab Cd', 'x', 1, 3), 결과 값 'Axd'
RIGHT({파라미터1}, {파라미터2})	오른쪽에서 잘라내기. 입력 문자열의 오른쪽부터 지정 길이만큼 잘라낸 문자열을 반환합니다. {파라미터1}은 입력 문자열, {파라미터2}는 길이입니다. 길이가 음수거나 문자열을 벗어나면, 문자열의 시작까지 잘라냅니다. (사용 예) RIGHT('Ab Cd', 2), 결과 값 'Cd'
STR_FROM_BOOL ({파라미터1})	BOOL을 문자열로 변환. 입력된 BOOL 값을 문자열로 변환합니다. (사용 예) STR_FROM_BOOL(1), 결과 값 'TRUE'
STR_FROM_BYTE ({파라미터1})	BYTE를 문자열로 변환. 입력된 BYTE 값을 문자열로 변환합니다. (사용 예) STR_FROM_BYTE(1), 결과 값 '1'
STR_FROM_SINT ({파라미터1})	SINT를 문자열로 변환. 입력된 SINT 값을 문자열로 변환합니다. (사용 예) STR_FROM_SINT(1), 결과 값 '1'
STR_FROM_WORD ({파라미터1})	WORD를 문자열로 변환. 입력된 WORD 값을 문자열로 변환합니다. (사용 예) STR_FROM_WORD(1), 결과 값 '1'
STR_FROM_INT ({파라미터1})	INT를 문자열로 변환. 입력된 INT 값을 문자열로 변환합니다. (사용 예) STR_FROM_INT(1), 결과 값 '1'
STR_FROM_DWORD ({파라미터1})	DWORD를 문자열로 변환. 입력된 DWORD 값을 문자열로 변환합니다. (사용 예) STR_FROM_DWORD(1), 결과 값 '1'

STR_FROM_DINT ({파라미터1})	DINT를 문자열로 변환. 입력된 DINT 값을 문자열로 변환합니다. (사용 예) STR_FROM_DINT(1), 결과 값 '1'
STR_FROM_REAL ({파라미터1})	REAL을 문자열로 변환. 입력된 REAL 값을 문자열로 변환합니다. (사용 예) STR_FROM_REAL(1), 결과 값 '1E000'
STR_FROM_TIME ({파라미터1})	TIME을 문자열로 변환. 입력된 TIME 값을 문자열로 변환합니다. (사용 예) STR_FROM_TIME(t#1s), 결과 값 'T#0d0h0m1s0ms'
STR_FROM_TOD ({파라미터1})	TOD를 문자열로 변환. 입력된 TOD 값을 문자열로 변환합니다. (사용 예) STR_FROM_TOD(tod#1:1:1), 결과 값 'TOD#1:1:1.000'
STR_FROM_DATE ({파라미터1})	DATE를 문자열로 변환. 입력된 DATE 값을 문자열로 변환합니다. (사용 예) STR_FROM_DATE(d#2019-1-2), 결과 값 'D#2019-1-2'
STR_FROM_DT ({파라미터1})	DT를 문자열로 변환. 입력된 DT 값을 문자열로 변환합니다. (사용 예) STR_FROM_DT(dt#2019-1-2-3:4:5), 결과 값 'DT#2019-1-2-3:4:5.000'
STR_TO_BOOL ({파라미터1})	문자열을 BOOL로 변환. 입력된 문자열을 BOOL 값으로 변환합니다. (사용 예) STR_TO_BOOL('TRUE'), 결과 값 1
STR_TO_BYTE ({파라미터1})	문자열을 BYTE로 변환. 입력된 문자열을 BYTE 값으로 변환합니다. (사용 예) STR_TO_BYTE('1'), 결과 값 1
STR_TO_SINT ({파라미터1})	문자열을 SINT로 변환. 입력된 문자열을 SINT 값으로 변환합니다. (사용 예) STR_TO_SINT('1'), 결과 값 1
STR_TO_WORD ({파라미터1})	문자열을 WORD로 변환. 입력된 문자열을 WORD 값으로 변환합니다. (사용 예) STR_TO_WORD('1'), 결과 값 1
STR_TO_INT ({파라미터1})	문자열을 INT로 변환. 입력된 문자열을 INT 값으로 변환합니다. (사용 예) STR_TO_INT('1'), 결과 값 1
STR_TO_DWORD ({파라미터1})	문자열을 DWORD로 변환. 입력된 문자열을 DWORD 값으로 변환합니다. (사용 예) STR_TO_DWORD('1'), 결과 값 1
STR_TO_DINT ({파라미터1})	문자열을 DINT로 변환. 입력된 문자열을 DINT 값으로 변환합니다. (사용 예) STR_TO_DINT('1'), 결과 값 1

STR_TO_REAL ({파라미터1})	문자열을 REAL로 변환. 입력된 문자열을 REAL 값으로 변환합니다. (사용 예) STR_TO_REAL('1'), 결과 값 1E000
STR_TO_TIME ({파라미터1})	문자열을 TIME로 변환. 입력된 문자열을 TIME 값으로 변환합니다. (사용 예) STR_TO_TIME('t#1s'), 결과 값 T#0d0h0m1s0ms
STR_TO_TOD ({파라미터1})	문자열을 TOD로 변환. 입력된 문자열을 TOD 값으로 변환합니다. (사용 예) STR_TO_TOD('tod#1:1:1'), 결과 값 TOD#1:1:1.000
STR_TO_DATE ({파라미터1})	문자열을 DATE로 변환. 입력된 문자열을 DATE 값으로 변환합니다. (사용 예) STR_TO_DATE('d#2019-1-2'), 결과 값 D#2019-1-2
STR_TO_DT ({파라미터1})	문자열을 DT로 변환. 입력된 문자열을 DT 값으로 변환합니다. (사용 예) STR_TO_DT('dt#2019-1-2-3:4:5'), 결과 값 DT#2019-1-2-3:4:5.000

1.4.5. 메모리 연산

ADDROF({파라미터1})	변수의 주소[bit]구하기. 내부 메모리와 출력 메모리만 적용 가능합니다. (사용 예) ADDROF(%QX10), 결과 값 10
SIZEOF({파라미터1})	변수의 크기[bit]구하기. (사용 예) SIZEOF(%QX10), 결과 값 1
CHG_ENDIAN ({파라미터1})	Endian 변환. (사용 예) CHG_ENDIAN(16#1122), 결과 값 16#2211
EDC_SUM_BYTE ({파라미터1}, {파라미터2})	BYTE SUM 값 구하기. 통신 오류검사를 위해 BYTE 단위로 SUM한 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_SUM_WORD ({파라미터1}, {파라미터2})	WORD SUM 값 구하기. 통신 오류검사를 위해 WORD 단위로 SUM한 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 word count 입니다.
EDC_SUM_DWORD ({파라미터1}, {파라미터2})	DWORD SUM 값 구하기. 통신 오류검사를 위해 DWORD 단위로 SUM한 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 dword count 입니다.

EDC_XOR_BYTE ({파라미터1}, {파라미터2})	BYTE XOR 값 구하기. 통신 오류검사를 위해 BYTE 단위로 XOR한 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_XOR_WORD ({파라미터1}, {파라미터2})	WORD XOR 값 구하기. 통신 오류검사를 위해 WORD 단위로 XOR한 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 word count 입니다.
EDC_XOR_DWORD ({파라미터1}, {파라미터2})	DWORD XOR 값 구하기. 통신 오류검사를 위해 DWORD 단위로 XOR한 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 dword count 입니다.
EDC_CRC_8DARC ({파라미터1}, {파라미터2})	8bit DARC CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_8I4321 ({파라미터1}, {파라미터2})	8bit I-432-1(ITU) CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_8ICODE ({파라미터1}, {파라미터2})	8bit I-CODE CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_8MAXIMDOW ({파라미터1}, {파라미터2})	8bit MAXIM(DOW-CRC) CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_8ROHC ({파라미터1}, {파라미터2})	8bit ROHC CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_8SMBUS ({파라미터1}, {파라미터2})	8bit SMBus CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.

EDC_CRC_8WCDMA ({파라미터1}, {파라미터2})	8bit WCDMA CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16ARC ({파라미터1}, {파라미터2})	16bit ARC(CRC-16, CRC-16/LHA, CRC-IBM) CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16DDS110 ({파라미터1}, {파라미터2})	16bit DDS-110 CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16DECTR ({파라미터1}, {파라미터2})	16bit DECT-R(R-CRC-16) CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16DNP ({파라미터1}, {파라미터2})	16bit DNP CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16EN13757 ({파라미터1}, {파라미터2})	16bit EN-13757 CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16MODBUS ({파라미터1}, {파라미터2})	16bit MODBUS CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16UMTS ({파라미터1}, {파라미터2})	16bit UMTS(CRC-16/BUYPASS, CRC-16/VERIFONE) CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
EDC_CRC_16USB ({파라미터1}, {파라미터2})	16bit USB CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.

EDC_CRC_32ISOHDLC ({파라미터1}, {파라미터2})	32bit ISO-HDLC(CRC-32, CRC-32/ADCCP, PKZIP, CRC-32/V-42) CRC 값 구하기. 통신 오류검사용 CRC 값을 구합니다. 계산 대상인 데이터는 내부 메모리에 저장되어 있어야 합니다. {파라미터1}은 주소, {파라미터2}는 byte count 입니다.
GET_BOOL ({파라미터1})	BOOL 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_BYTE ({파라미터1})	BYTE 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_SINT ({파라미터1})	SINT 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_WORD ({파라미터1})	WORD 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_INT ({파라미터1})	INT 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_DWORD ({파라미터1})	DWORD 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_DINT ({파라미터1})	DINT 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_REAL ({파라미터1})	REAL 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_TIME ({파라미터1})	TIME 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_TOD ({파라미터1})	TOD 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_DATE ({파라미터1})	DATE 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
GET_DT ({파라미터1})	DT 값 읽기. {파라미터1}은 내부 메모리의 주소입니다.
SET_BOOL ({파라미터1}, {파라미터2})	BOOL 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_BYTE ({파라미터1}, {파라미터2})	BYTE 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_SINT ({파라미터1}, {파라미터2})	SINT 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.

SET_WORD ({파라미터1}, {파라미터2})	WORD 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_INT ({파라미터1}, {파라미터2})	INT 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_DWORD ({파라미터1}, {파라미터2})	DWORD 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_DINT ({파라미터1}, {파라미터2})	DINT 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_REAL ({파라미터1}, {파라미터2})	REAL 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_TIME ({파라미터1}, {파라미터2})	TIME 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_TOD ({파라미터1}, {파라미터2})	TOD 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_DATE ({파라미터1}, {파라미터2})	DATE 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.
SET_DT ({파라미터1}, {파라미터2})	DT 값 쓰기. {파라미터1}은 내부 메모리의 주소입니다. {파라미터2}는 쓸 값입니다.

1.4.6. 흐름 제어

SEL({파라미터1}, {파라미터2}, {파라미터3})	2개 중 하나 선택. {파라미터1}이 0이면 {파라미터2}를 반환하고, 나머지 경우는 {파라미터3}을 반환합니다. {파라미터2}와 {파라미터3}의 형은 어떤 것이든 가능합니다.
MUX({파라미터1}, ...)	여러 개 중 하나 선택. {파라미터1}은 0일 때 {파라미터2}를 선택하고, +1마다 다음 파라미터를 선택하여 반환합니다.

RETURN;		작업 구조에서 나감. 현재 수행 중인 작업 구조(PROG, FB, FUN)를 빠져나갑니다.(현재 작업을 호출한 곳으로 복귀합니다.) 호출 작업이 단독으로 1행을 구성합니다.
EXIT;		반복 작업에서 나감. 현재 수행 중인 FOR, REPEAT, WHILE 등의 반복 작업에서 빠져나갑니다.

IF	형식	IF {선택조건1} THEN
		{작업 행1};
		ELSIF {선택조건2} THEN
		{작업 행2};
		ELSE
		{작업 행e};
		END_IF;

	설명	IF 행은 IF로 시작해서 END_IF;로 끝납니다. ELSIF와 ELSE는 필요한 경우 사용합니다. ELSIF는 여러 개를 넣을 수 있고, ELSE는 한 개까지 넣을 수 있습니다. 각 {작업 행}은 0행 이상을 넣을 수 있습니다. {선택조건1}의 결과가 TRUE면, {작업 행1}을 수행합니다. {선택조건1}의 결과가 FALSE이고, {선택조건2}의 결과가 TRUE면, {작업 행2}를 수행합니다. 앞의 {선택조건}이 모두 FALSE이면, {작업 행e}를 수행합니다.
CASE	형식	CASE {선택조건} OF {선택 값1}: {작업 행1}; {선택 값2}: {작업 행2}; ELSE {작업 행e}; END_CASE;
	설명	CASE 행은 CASE로 시작해서 END_CASE;로 끝납니다. ELSE는 필요한 경우 사용합니다. {선택 값}은 1개 이상으로 지정할 수 있습니다. {선택 값} 내에서도 여러 개를 동시에 지정할 수 있습니다. {선택조건}의 값과 {선택 값1}이 일치하면, {작업 행1}을 수행합니다. 나머지는 IF에서와 같습니다.

FOR	형식	FOR {변수 지정} TO {끝 값} BY {증가 값} DO {작업 행}; END_FOR;
	설명	FOR 행은 FOR로 시작해서 END_FOR;로 끝납니다. {변수 지정}에서 반복 수행 조건의 변수와 초기값을 지정합니다. (예, vInt1 := 1) {끝 값}은 반복 수행을 멈출 조건이 되는 값입니다. {증가 값}은 반복 수행이 끝날 때마다 반복 변수에 더할 값입니다. {작업 행}은 0행 이상을 넣을 수 있습니다. 반복 조건이 유지되는 동안 반복 수행됩니다. 반복 조건은 {증가 값}이 양수면, 변수의 값이 {끝 값}보다 작거나 같은 동안 유지됩니다. {증가 값}이 음수면, 변수의 값이 {끝 값}보다 크거나 같은 동안 유지됩니다. {증가 값}은 기본이 1입니다. BY {증가 값}을 사용하지 않으면 1로 설정됩니다.
REPEAT	형식	REPEAT {작업 행}; UNTIL {반복조건} END_REPEAT;
	설명	REPEAT 행은 REPEAT으로 시작해서 END_REPEAT;로 끝납니다. {작업 행}은 0행 이상을 넣을 수 있습니다. {반복조건}이 TRUE인 동안 반복 수행됩니다.
WHILE	형식	WHILE {반복조건} DO {작업 행}; END_WHILE;
	설명	WHILE 행은 WHILE로 시작해서 END_WHILE;로 끝납니다. {작업 행}은 0행 이상을 넣을 수 있습니다. {반복조건}이 TRUE인 동안 반복 수행됩니다.

1.4.7. 시스템 명령

BACKUP ({파라미터1}, {파라미터2})	지정된 내부 메모리를 backup합니다. 재기동할 때, E5A는 내부 메모리를 backup한 값들이 없다면 모든 값을 0으로 초기화하고, 아니면 backup한 값들로 초기화합니다. 반환값은 기록된 byte 수 입니다. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.
SEE_NW_DOMAIN()	E5A의 현재 hostname을 문자열로 구합니다.
SEE_NW_IP()	E5A의 현재 IP address를 문자열로 구합니다.
SEE_NW_MODE()	E5A의 현재 WiFi 모드(AP, Station)를 문자열로 구합니다.
SEE_NW_SSID()	E5A의 현재 SSID를 문자열로 구합니다.

WA_ABLE_GEN ({파라미터1}, {파라미터2})	E5A외부에서 내부 메모리로 쓰기 가능 여부 확인. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.
WA_ABLE_OUT ({파라미터1}, {파라미터2})	E5A외부에서 출력 메모리로 쓰기 가능 여부 확인. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.
WA_DISABLE_GEN ({파라미터1}, {파라미터2})	E5A외부에서 내부 메모리로 쓰기 불가능하게 변경. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.
WA_DISABLE_OUT ({파라미터1}, {파라미터2})	E5A외부에서 출력 메모리로 쓰기 불가능하게 변경. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.
WA_ENABLE_GEN ({파라미터1}, {파라미터2})	E5A외부에서 내부 메모리로 쓰기 가능하게 변경. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.
WA_ENABLE_OUT ({파라미터1}, {파라미터2})	E5A외부에서 출력 메모리로 쓰기 가능하게 변경. {파라미터1}은 메모리의 주소입니다. {파라미터2}는 bit 단위의 길이입니다.

1.5. FB형

E5A는 표준에 정의된 FB외에도 사용자 편의를 위해 몇 가지 FB형이 추가되어 있습니다.

1.5.1. 표준 기반

IEC 61131-3 표준은 1993년 처음 발표되었고, DST의 기초가 된 2nd edition은 2003년에 발표되었습니다. 가장 최근에 발표된 것은 3rd edition으로 2013년에 발표되었습니다.

E5A는 DST라는 프로그래밍언어가 동작하는 작은 컴퓨터처럼 설명되고 있지만, 표준이 만들어질 때에는 PLC를 logic gate의 뭉치로 보는 경향이 강했습니다. 이는 PLC의 원래 표현이 Programmable Logic Controller인 것에서도 알 수 있습니다. 그래서 표준 FB들은 내부에 logic gate가 존재하는 것처럼 설계되었습니다.

Hardware의 관점에서 digital 신호는 0[V](= Low, 줄여서 L) 혹은 Vcc[V](= High, 줄여서 H)로 구분되는 2진 신호입니다. Software의 관점에서 변수형 중 BOOL은 digital 신호에 대응하는 용도로 가장 많이 사용합니다. Digital 신호를 BOOL에 대응시킬 때, (L = FALSE, H = TRUE)로 처리하는 것이 일반적이지만, (L = TRUE, H = FALSE)로 처리하는 경우도 있습니다. Falling edge(↘)는 신호가 H에서 L로 변경되는 순간이고, rising edge(↗)는 L에서 H로 변경되는 순간입니다.

Logic gate의 상태를 변경하는 기준은 level trigger 혹은 edge trigger가 될 수 있습니다. Level trigger는 신호가 H 또는 L일 때 지정한 동작을 수행하는 것을 의미하며, 순서대로 H active 또는 L active라고 부릅니다. Edge trigger는 신호가 ↘ 또는 ↗에서 지정한 동작을 수행하는 것을 의미하며, 순서대로 falling edge trigger 또는 rising edge trigger라고 부릅니다.

E5A의 표준 FB들은 FALSE를 L로, TRUE를 H로 인식합니다. 다음의 설명에서 H active는 (H), L active는 (L), falling edge trigger는 (↘), rising edge trigger는 (↗)로 표시합니다. 표준 FB는 호출될 때마다 상태를 처리하므로 호출 주기를 적절하게 선택해야 합니다.

CTD	이름	Down Counter.		
	지역 변수	입력	CD : BOOL; LD : BOOL; PV : INT;	count down 입력 신호. (↗) 초기값 설정 신호. (H) 초기값.
		출력	Q : BOOL; CV : INT;	상태 출력 신호. 현재값.
	설명	LD가 H일 때, PV를 CV에 복사. CD가 ↗일 때, CV를 -1. CV <= 0일 때, Q는 1. 나머지 경우, Q는 0.		
CTU	이름	Up Counter.		
	지역 변수	입력	CU : BOOL; R : BOOL; PV : INT;	count up 입력 신호. (↗) 초기값 설정 신호. (H) 기준값.
		출력	Q : BOOL; CV : INT;	상태 출력 신호. 현재값.
	설명	R이 H일 때, CV를 0으로 설정. CU가 ↗일 때, CV를 +1. CV >= PV일 때, Q는 1. 나머지 경우, Q는 0.		
CTUD	이름	Up/Down Counter.		
	지역 변수	입력	CU : BOOL; CD : BOOL; R : BOOL; LD : BOOL PV : INT;	count up 입력 신호. (↗) count down 입력 신호. (↗) 초기값 설정 신호. (H) 초기값 설정 신호. (H) 기준값.
		출력	QU : BOOL; QD : BOOL; CV : INT;	상태 출력 신호. 상태 출력 신호. 현재값.
	설명	R이 H일 때, CV를 0으로 설정. LD가 H일 때, PV를 CV에 복사. CU가 ↗일 때, CV를 +1. CD가 ↗일 때, CV를 -1. CV >= PV일 때, QU는 1. 나머지 경우, QU는 0. CV <= 0일 때, QD는 1. 나머지 경우, QD는 0.		

F_TRIG	이름	Falling Edge Detector.		
	지역 변수	입력	CLK : BOOL;	입력 신호. (↘)
		출력	Q : BOOL;	상태 출력 신호.
	설명	CLK가 ↘일 때, Q는 1. 나머지 경우, Q는 0.		
R_TRIG	이름	Rising Edge Detector.		
	지역 변수	입력	CLK : BOOL;	입력 신호. (↗)
		출력	Q : BOOL;	상태 출력 신호.
	설명	CLK가 ↗일 때, Q는 1. 나머지 경우, Q는 0.		
RS	이름	RS Flipflop.		
	지역 변수	입력	S : BOOL; R1 : BOOL;	입력 신호. (H) 입력 신호. (H)
		출력	Q : BOOL;	상태 출력 신호.
	설명	R1이 H일 때, Q는 0. R1은 L, S는 H일 때, Q는 1. 나머지 경우, Q는 값 유지.		
RTC	이름	Real Time Clock.		
	지역 변수	입력	EN : BOOL; PDT : DT;	입력 신호. (↗) 설정시각.
		출력	Q : BOOL; CDT : DT;	상태 출력 신호. 현재시각.
	설명	EN이 ↗일 때, PDT로 시각 재설정. Q는 EN을 복사. CDT는 호출한 순간의 시각.		
SEMA	이름	Semaphore.		
	지역 변수	입력	CLAIM : BOOL; RELEASE:BOOL;	입력 신호. (H) 입력 신호. (H)
		출력	BUSY : BOOL;	상태 출력 신호.
	설명	CLAIM이 H이고, 요청이 성공하면, BUSY는 0. CLAIM이 H이고, 요청이 실패하면, BUSY는 1. RELEASE가 H이고, 요청이 성공하면, BUSY는 0. 공유 자원의 접근을 통제하기 위해 사용하는 기능으로 작업객체에 할당됩니다. 권한을 획득한 작업객체에서 해제를 수행해야 합니다.		

<table>
<tr><td rowspan="4">SR</td><td>이름</td><td colspan="3">SR Flipflop.</td></tr>
<tr><td rowspan="2">지역 변수</td><td>입력</td><td>S1 : BOOL;
R : BOOL;</td><td>입력 신호. (H)
입력 신호. (H)</td></tr>
<tr><td>출력</td><td>Q : BOOL;</td><td>상태 출력 신호.</td></tr>
<tr><td>설명</td><td colspan="3">S1이 H일 때, Q는 1.
S1은 L, R는 H일 때, Q는 0. 나머지 경우, Q는 값 유지.</td></tr>
<tr><td rowspan="4">TOF</td><td>이름</td><td colspan="3">Timer OFF Delay.</td></tr>
<tr><td rowspan="2">지역 변수</td><td>입력</td><td>IN : BOOL;
PT : TIME;</td><td>입력 신호. (H)
지연시간.</td></tr>
<tr><td>출력</td><td>Q : BOOL;
ET : TIME;</td><td>상태 출력 신호.
경과시간.</td></tr>
<tr><td>설명</td><td colspan="3">IN이 H일 때, Q는 1, ET는 0.
IN이 ↘일 때, ET 측정 시작.
IN이 L이고, ET < PT일 때, ET 계속 측정.
IN이 L이고, ET >= PT일 때, ET 측정 중지, Q는 0.
IN의 신호에 따라 PT만큼 지연된 시간 동안 Q의 ON을 유지하는 timer를 운영합니다.</td></tr>
<tr><td rowspan="4">TON</td><td>이름</td><td colspan="3">Timer ON Delay.</td></tr>
<tr><td rowspan="2">지역 변수</td><td>입력</td><td>IN : BOOL;
PT : TIME;</td><td>입력 신호. (H)
지연시간.</td></tr>
<tr><td>출력</td><td>Q : BOOL;
ET : TIME;</td><td>상태 출력 신호.
경과시간.</td></tr>
<tr><td>설명</td><td colspan="3">IN이 L일 때, Q는 0, ET는 0.
IN이 ↗일 때, ET 측정 시작.
IN이 H이고, ET < PT일 때, ET 계속 측정.
IN이 H이고, ET >= PT일 때, ET 측정 중지, Q는 1.
IN의 신호에 따라 PT만큼 지연된 시간 후, Q를 ON하는 timer를 운영합니다.</td></tr>
<tr><td rowspan="4">TP</td><td>이름</td><td colspan="3">Pulse Generator.</td></tr>
<tr><td rowspan="2">지역 변수</td><td>입력</td><td>IN : BOOL;
PT : TIME;</td><td>입력 신호. (H)
지연시간.</td></tr>
<tr><td>출력</td><td>Q : BOOL;
ET : TIME;</td><td>상태 출력 신호.
경과시간.</td></tr>
<tr><td>설명</td><td colspan="3">Q가 0, IN이 ↗일 때, ET 측정 시작. Q는 1.
ET < PT일 때, ET 계속 측정.
ET >= PT일 때, ET 측정 중지, Q는 0.
IN이 L이고, ET >= PT일 때, ET는 0.
IN의 신호에 따라 PT만큼 지연된 시간 동안만 Q를 ON하는 timer를 운영합니다.</td></tr>
</table>

1.5.2. PID 제어

PID 제어는 19세기 후반부터 발전되어온 자동제어기술(공식적인 명칭은 제어이론, Control Theory) 중 1922년 처음 소개되어 지금까지 가장 잘 알려지고, 널리 쓰이는 자동제어기술입니다.

PID 제어에 대해 간략하게 정리된 내용은 위키백과의 한글판(https://ko.wikipedia.org/wiki/PID_제어기)에 있고, 더 자세한 내용은 위키백과의 영문판(https://en.wikipedia.org/wiki/PID_controller)에 있습니다.

PID	이름	PID controller.		
	지역 변수	입력	KP : USINT; KI : USINT; KD : USINT; HEND : INT; LEND : INT; HIW : INT; LIW : INT; SP : INT; PV : INT;	비례계수. 범위: 1~100 적분계수. 범위: 0~100 미분계수. 범위: 0~100 제어범위 상한. 제어범위 하한. 적분항적용 상한. 적분항적용 하한. 목표값. 현재값.
		출력	EC : USINT; CV : INT;	오류코드. 제어값[%]. 범위: -100~100

PID FB는 표준 FB와 마찬가지로 호출될 때마다 상태를 처리합니다. 입력되는 값들 중 PV만 주기적으로 갱신하는 것이 일반적인 사용법입니다. 호출 주기가 반응 속도를 결정하므로 사용자는 호출 주기를 적절하게 선택해야 합니다.

CV는 백분율로 표시되므로 사용자는 제어 대상에 적용할 때 실제의 제어범위에 맞춰서 크기변환(scaling)을 해야 합니다. CV가 100이면 최대 출력으로 제어하고, 0이면 출력을 끄고, -100이면 역방향 최대 출력으로 제어합니다.

KP, KI, KD의 영향력은 서로의 비율로 결정됩니다. (KP=2, KI=1, KD=0)로 설정된 것과 (KP=100, KI=50, KD=0)로 설정된 것은 동일한 특성을 보입니다.

HEND와 LEDN로 지정하는 제어범위는 SP 및 PV가 가질 수 있는 값의 한계입니다. SP나 PV가 제어범위를 벗어나면 오류가 발생됩니다.

HIW와 LIW는 PID 제어 과정에서 적분항이 과도하게 작용해서 발생하는 문제를 방지하기 위해 적용한 Integral Windup 기능에 사용하는 값입니다. PV가 LIW~HIW의 범위를 벗어나면 적분항을 계산에 반영하지 않습니다.

오류가 발생한 경우, 해당 작업객체는 중단되므로 FB 호출 이후의 작업은 실행되지 않습니다. 오류코드는 EC에 저장되므로 작업객체의 다음 실행 주기나 다른 작업객체에서 오류 점검을 해야 문제 분석이 가능합니다. 단, E5aLoader를 이용한 시험 단계에서는 화면에 오류 내용과 위치가 표시되므로 사용자는 이를 참고하여 DST를 수정할 수 있습니다.

1.5.3. Serial 통신

RS485는 RESOURCE 중 SER_1로 연동됩니다. SER_1의 초기 운용모드는 closed이므로 통신을 위해서 CONF_SER1 FB로 설정을 해야 합니다.

CONF_SER1	이름	SER_1 통신 설정.		
	지역 변수	입력	GET_SET : BOOL; RATE : UDINT; PARITY : USINT; DATABITS : USINT; STOPBIT : BOOL; MODE : USINT; SLAVE_ID : USINT;	설정방향. Bit Rate[bps]. 범위: 300~3M Parity. 범위: 0~4 Data Bits[bit]. 범위: 5~8 Stop Bit. 운용모드. 범위: 0~3 Slave ID. 범위: 1~247
		출력	EC : UINT;	오류코드.

FB가 호출되었을 때, GET_SET이 0이면 현재 설정된 상태로부터 설정값들을 읽고, GET_SET이 1이면 새로운 설정값을 씁니다. 설정값을 읽는 경우 RATE, PARITY, DATABITS, STOPBIT, MODE, SLAVE_ID 등이 갱신됩니다. 설정값을 쓰는 경우 요청한 대로 설정하지 못하면 오류가 발생합니다.

RATE은 SER_1의 bit rate 설정이고, 단위는 [bps]입니다.

PARITY 값의 의미는 0(= no parity), 1(= even parity), 2(=odd parity), 3(=zero parity), 4(=one parity)입니다.

DATABITS는 serial 통신의 data bit 크기입니다.

STOPBIT 값의 의미는 0(=no stop bit), 1(=1 stop bit)입니다.

SER_1의 운용모드는 closed, MODBUS RTU master, MODBUS RTU slave, 사용자 정의 프로토콜(user defined protocol) serial 중 하나입니다. 현재의 운용모드가 closed일 때만 운용모드를 변경할 수 있습니다. 이는 RATE만 변경하려는 경우도 포함합니다.

MODE 값의 의미는 0(=closed), 1(=MODBUS RTU master), 2(=MODBUS RTU slave), 3(=사용자 정의 프로토콜 serial)입니다.

SLAVE_ID는 운용모드가 MODBUS RTU slave일 때만 유효한 값입니다. 이외의 운용모드에서는 범위를 벗어난 값을 가져도 오류가 발생하지 않습니다.

COMM_SER1	이름	MODBUS RTU master 통신.		
	지역 변수	입력	READ_WRITE : BOOL; SLAVE_ID : USINT; SLAVE_AREA : USINT; SLAVE_ADDR : UINT; DATA_COUNT : UINT; MEM_ADDR : UINT;	Data방향. Slave ID. 범위: 1~247 접근영역. 범위: 0~4 Slave의 영역내부 주소. Data수. 내부 메모리 주소.
		출력	EC : UINT;	오류코드.

운용모드가 MODBUS RTU master인 경우, E5A는 MODBUS RTU slave인 상대 장치의 값을 읽거나 쓸 수 있습니다. 이 때 사용하는 것이 COMM_SER1 FB입니다.

READ_WRITE 값의 의미는 0(=read), 1(=write)입니다. Data방향이 read라면 E5A는 상대 장치로부터 data를 가져오고, data방향이 write이라면 E5A는 상대 장치에게 data를 보냅니다.

SLAVE_ID는 RS485 위에서 상대 장치를 구분하는 값입니다.

SLAVE_AREA 값의 의미는 0(=general register), 1(=coil status), 2(=input status), 3(=holding register), 4(=input register)입니다. 상대 장치가 E5A인 경우 메모리 영역을 구분하는 값입니다.

SLAVE_ADDR은 접근영역에서의 주소입니다. Coil status나 input status에서는 bit 단위이고, general register, holding register, input register에서는 word(16bit) 단위를 사용합니다.

DATA_COUNT는 읽거나 쓸 data수입니다.

MEM_ADDR은 내부 메모리의 주소입니다.

입력 지역 변수에서 요청한 대로 처리가 불가능하면 오류가 발생합니다. 단, 상대 장치와의 통신 과정에서 문제가 발생하면, EC에만 반영되고 작업객체의 실행에는 영향을 주지 않습니다.

COMM_SERUSR	이름	사용자 정의 프로토콜 serial 통신.		
	지역 변수	입력	RECV_SEND : USINT; SEND_ADDR : UINT; SEND_LEN : UINT; RECV_ADDR : UINT; RECV_LEN : UINT;	송수신구성. 송신블록 주소. 송신길이[byte]. 범위: 1~255 수신블록 주소. 수신최대길이[byte]. 범위: 1~255
		출력	EC : UINT;	오류코드.

운용모드가 사용자 정의 프로토콜 serial인 경우, E5A는 사용자 정의 프로토콜 serial로 운용되는 상대 장치와 data를 주고받을 수 있습니다. 이때 사용하는 것이 COMM_SERUSR FB입니다.

RECV_SEND 값의 의미는 0(=receive), 1(=send), 2(=send & receive)입니다. Receive일 때는 RECV_ADDR과 RECV_LEN만 유효하고, send일 때는 SEND_ADDR과 SEND_LEN만 유효합니다. Send & receive일 때, send를 먼저 수행하고 receive를 합니다.

송신블록과 수신블록은 첫 byte에 실제 송수신한 byte 수가 저장됩니다. 송수신되는 data는 2번째 byte부터 시작합니다. 모두 내부 메모리 영역에 위치해야 합니다.

SEND_LEN은 송신할 data의 byte수이고, RECV_LEN은 수신할 data buffer의 크기(byte수)입니다.

1.5.4. Ethernet Server 통신

E5A에 구축된 TCP server는 RESOURCE의 ETH_1로 연동됩니다. 초기 운용모드는 closed입니다.

CONF_ETH1	이름	ETH_1 통신 설정.		
	지역 변수	입력	GET_SET : BOOL; IPV4_6 : BOOL; PORT : UINT; MODE : USINT; E5D_HOST : STRING;	설정방향. IP종류. Port번호. 운용모드. 범위: 0~3 (예비용).
		출력	EC : UINT;	오류코드.

FB가 호출되었을 때, GET_SET이 0이면 현재 설정된 상태로부터 설정값들을 읽고, GET_SET이 1이면 새로운 설정값을 씁니다. 설정값을 읽는 경우 IPV4_6, PORT, MODE, E5D_HOST 등이 갱신됩니다. 설정값을 쓰는 경우 요청한 대로 설정하지 못하면 오류가 발생합니다.

IPV4_6 값의 의미는 0(=IPv4), 1(=IPv6)입니다.

PORT는 TCP server의 port번호입니다. 1024번보다 작은 well-known port 영역에서는 운용모드에 할당된 port만 열 수 있습니다(MODBUS TCP server = 502, 사용자 정의 프로토콜 TCP server = 23). 임의로 port번호를 지정하려면 1024~49151 사이의 값을 사용합니다.

MODE 값의 의미는 0(=closed), 1(=MODBUS TCP server), 2(=사용자 정의 프로토콜 TCP server), 3(=E5D server)입니다. E5D server는 예비용으로 아직 사용할 수 없습니다.

COMM_ETHUSR	이름	사용자 정의 프로토콜 TCP server 통신.		
	지역 변수	입력	RECV_SEND : USINT; SEND_ADDR : UINT; SEND_LEN : UINT; RECV_ADDR : UINT; RECV_LEN : UINT;	송수신구성. 송신블록 주소. 송신길이[byte]. 범위: 1~255 수신블록 주소. 수신최대길이[byte]. 범위: 1~255
		출력	EC : UINT;	오류코드.

운용모드가 사용자 정의 프로토콜 TCP server인 경우, E5A는 사용자 정의 프로토콜 TCP client로 운용되는 상대 장치와 data를 주고받을 수 있습니다. 이 때 사용하는 것이 COMM_ETHUSR FB입니다.

RECV_SEND 값의 의미는 0(=receive), 1(=send), 2(=send & receive)입니다. Receive일 때는 RECV_ADDR과 RECV_LEN만 유효하고, send일 때는 SEND_ADDR과 SEND_LEN만 유효합니다. Send & receive일 때, send를 먼저 수행하고, receive를 합니다.

송신블록과 수신블록은 첫 byte에 실제 송수신한 byte 수가 저장됩니다. 송수신되는 data는 2번째 byte부터 시작합니다. 모두 내부 메모리 영역에 위치해야 합니다.

SEND_LEN은 송신할 data의 byte수이고, RECV_LEN은 수신할 data buffer의 크기(byte수)입니다.

1.5.5. Ethernet Client 통신

E5A에 구축된 TCP client는 RESOURCE의 ETH_2로 연동됩니다. 초기 운용모드는 closed입니다.

CONF_ETH2	이름	ETH_2 통신 설정.		
	지역 변수	입력	GET_SET : BOOL; IPV4_6 : BOOL; HOST : STRING; PORT : UINT; MODE : USINT; UNIT_ID : USINT;	설정방향. IP종류. Server 주소. Port번호. 운용모드. 범위: 0~2 Unit ID.
		출력	EC : UINT;	오류코드.

FB가 호출되었을 때, GET_SET이 0이면 현재 설정된 상태로부터 설정값들을 읽고, GET_SET이 1이면 새로운 설정값을 씁니다. 설정값을 읽는 경우 IPV4_6, PORT, MODE, UNIT_ID, HOST 등이 갱신됩니다. 설정값을 쓰는 경우 요청한 대로 설정하지 못하면 오류가 발생합니다.

IPV4_6 값의 의미는 0(=IPv4), 1(=IPv6)입니다.

HOST는 상대 장치의 IP 주소나 domain name입니다.

PORT는 TCP server의 port번호입니다. 상대 장치가 열어 놓은 port번호를 입력해야 합니다.

MODE 값의 의미는 0(=closed), 1(=MODBUS TCP client), 2(=사용자 정의 프로토콜 TCP client)입니다.

UNIT_ID는 상대 장치가 MODBUS TCP server이고, 특정 unit ID를 지정해야만 응답하는 경우에 사용합니다. 상대가 E5A이거나 unit ID에 무관하게 응답한다면, 값을 지정하지 않습니다.

COMM_ETH2	이름	MODBUS TCP client 통신.		
	지역 변수	입력	READ_WRITE : BOOL; SERVER_AREA : USINT; SERVER_ADDR : UINT; DATA_COUNT : UINT; MEM_ADDR : UINT;	Data방향. 접근영역. 범위: 0~4 Slave의 영역내부 주소. Data수. 내부 메모리 주소.
		출력	EC : UINT;	오류코드.

운용모드가 MODBUS TCP client인 경우, E5A는 MODBUS TCP server인 상대 장치의 값을 읽거나 쓸 수 있습니다. 이 때 사용하는 것이 COMM_ETH2 FB입니다.

READ_WRITE 값의 의미는 0(=read), 1(=write)입니다. Data방향이 read라면 E5A는 상대 장치로부터 data를 가져오고, data방향이 write이라면 E5A는 상대 장치에게 data를 보냅니다.

SERVER_AREA 값의 의미는 0(=general register), 1(=coil status), 2(=input status), 3(=holding register), 4(=input register)입니다. 상대 장치가 E5A인 경우 메모리 영역을 구분하는 값입니다.

SERVER_ADDR은 접근영역에서의 주소입니다. Coil status나 input status에서는 bit 단위이고, general register, holding register, input register에서는 word(16bit) 단위를 사용합니다.

DATA_COUNT는 읽거나 쓸 data수입니다.

MEM_ADDR은 내부 메모리의 주소입니다.

입력 지역 변수에서 요청한 대로 처리가 불가능하면 오류가 발생합니다. 단, 상대 장치와의 통신 과정에서 문제가 발생하면, EC에만 반영되고, 작업객체의 실행에는 영향을 주지 않습니다.

	이름	사용자 정의 프로토콜 TCP client 통신.		
COMM_ETHUSR	지역 변수	입력	RECV_SEND : USINT; SEND_ADDR : UINT; SEND_LEN : UINT; RECV_ADDR : UINT; RECV_LEN : UINT;	송수신구성. 송신블록 주소. 송신길이[byte]. 범위: 1~255 수신블록 주소. 수신최대길이[byte]. 범위: 1~255
		출력	EC : UINT;	오류코드.

운용모드가 사용자 정의 프로토콜 TCP client인 경우, E5A는 사용자 정의 프로토콜 TCP server로 운용되는 상대 장치와 data를 주고받을 수 있습니다. 이때 사용하는 것이 COMM_ETHUSR FB입니다.

RECV_SEND 값의 의미는 0(=receive), 1(=send), 2(=send & receive)입니다. Receive일 때는 RECV_ADDR과 RECV_LEN만 유효하고, send일 때는 SEND_ADDR과 SEND_LEN만 유효합니다. Send & receive일 때, send를 먼저 수행하고 receive를 합니다.

송신블록과 수신블록은 첫 byte에 실제 송수신한 byte 수가 저장됩니다. 송수신되는 data는 2번째 byte부터 시작합니다. 모두 내부 메모리 영역에 위치해야 합니다.

SEND_LEN은 송신할 data의 byte수이고, RECV_LEN은 수신할 data buffer의 크기(byte수)입니다.

2. 사용 예

작업지시자에 대한 사용 예를 모아서 basic0.dst를 작성합니다. E5aLoader에서 행 단위로 실행하면, 결과를 확인할 수 있습니다.

```
CONFIGURATION nameOfConf
  VAR_GLOBAL
    gTest : STRING := '123456789';
  END_VAR

  RESOURCE myDevice ON CPU
    TASK taskInit (SINGLE := TRUE, PRIORITY := 3);
    TASK taskSync (INTERVAL := t#3s, PRIORITY := 7);
    PROGRAM WITH taskInit : ProgInit();
    PROGRAM WITH taskSync : ProgMain();
  END_RESOURCE
END_CONFIGURATION

PROGRAM ProgInit
  VAR
    vLen : INT;
    vAddr, vDword1 : DWORD;
    vDint1 : DINT;
    vReal1 : REAL;
    vStr1 : STRING;
    vTime1 : TIME;
    vTod1 : TOD;
    vDate1 : DATE;
    vDt1 : DT;
  END_VAR

  vDword1 := TRUE & FALSE; (* FALSE *)
  vDword1 := TRUE | FALSE; (* TRUE *)
  vDword1 := !TRUE; (* FALSE *)
  vDword1 := TRUE ^ FALSE; (* TRUE *)
  vDword1 := 2#1010 AND 2#1100; (* 16#8 *)
  vDword1 := 2#1010 OR 2#1100; (* 16#E *)
  vDword1 := NOT 2#1010; (* 16#FFFF_FFF5 *)
  vDword1 := 2#1010 XOR 2#1100; (* 2#0110 *)
  vDword1 := 1 >= 2; (* FALSE *)
  vDword1 := 1 GT 2; (* FALSE *)
  vDword1 := 1 <= 2; (* TRUE *)
```

```
vDword1 := 1 LT 2; (* TRUE *)
vDword1 := 1 = 2; (* FALSE *)
vDword1 := 1 NE 2; (* TRUE *)

vReal1 := 1 + 2; (* 3 *)
vReal1 := 1 - 2; (* -1 *)
vReal1 := 1 * 2; (* 2 *)
vReal1 := 1 / 2; (* 0.5 *)
vReal1 := 5 MODULO 4; (* 1 *)
vReal1 := 5 ** 2; (* 25 *)
vReal1 := ABS(-5); (* 5 *)
vReal1 := SQRT(4); (* 2 *)
vReal1 := EXP(1); (* ~ 2.71828 *)
vReal1 := LN(EXP(1)); (* 1 *)
vReal1 := LOG(10); (* 1 *)
vReal1 := MAX(2,3,4); (* 4 *)
vReal1 := MAX(2,1,4); (* 4 *)
vReal1 := MIN(2,3,4); (* 2 *)
vReal1 := MIN(2,1,4); (* 1 *)
vReal1 := LIMIT(2,3,4); (* 3 *)
vReal1 := LIMIT(2,1,4); (* 2 *)
vDword1 := ROL(1,2); (* 4 *)
vDword1 := ROR(1,2); (* 16#4000_0000 *)
vDword1 := SHL(1,2); (* 4 *)
vDword1 := SHR(1,2); (* 0 *)

vReal1 := ACOS(0.5); (* 60 *)
vReal1 := ASIN(0.5); (* 30 *)
vReal1 := ATAN(0.5); (* ~ 26.565 *)
vReal1 := COS(50); (* ~ 0.643 *)
vReal1 := SIN(50); (* ~ 0.766 *)
vReal1 := TAN(50); (* ~ 1.192 *)

vStr1 := CONCAT('Ab', ' Cd'); (* 'Ab Cd' *)
vStr1 := DELETE('Ab Cd', 2, -1); (* 'Ab' *)
vDword1 := FIND('Ab Bb ', 'b '); (* 1 *)
vStr1 := INSERT('Ab Cd', ' x', 2); (* 'Ab x Cd' *)
vStr1 := LEFT('Ab Cd', 2); (* 'Ab' *)
vDword1 := LEN('Ab Cd'); (* 5 *)
vStr1 := MID('Ab Cd', 2, 2); (* ' C' *)
vStr1 := REPLACE('Ab Cd', 'x', 1, 3); (* 'Axd' *)
vStr1 := RIGHT('Ab Cd', 2); (* 'Cd' *)
```

```
vStr1 := STR_FROM_BOOL(1); (* 'TRUE' *)
vStr1 := STR_FROM_BYTE(1); (* '1' *)
vStr1 := STR_FROM_SINT(1); (* '1' *)
vStr1 := STR_FROM_WORD(1); (* '1' *)
vStr1 := STR_FROM_INT(1); (* '1' *)
vStr1 := STR_FROM_DWORD(1); (* '1' *)
vStr1 := STR_FROM_DINT(1); (* '1' *)
vStr1 := STR_FROM_REAL(1); (* '1E000' *)
vStr1 := STR_FROM_TIME(t#1s); (* 'T#0d0h0m1s0ms' *)
vStr1 := STR_FROM_TOD(tod#1:1:1); (* 'TOD#1:1:1.000' *)
vStr1 := STR_FROM_DATE(d#2019-1-2); (* 'D#2019-1-2' *)
vStr1 := STR_FROM_DT(dt#2019-1-2-3:4:5); (* 'DT#2019-1-2-3:4:5.000' *)
vDword1 := STR_TO_BOOL('TRUE'); (* 1 *)
vDword1 := STR_TO_BYTE('1'); (* 1 *)
vDword1 := STR_TO_SINT('1'); (* 1 *)
vDword1 := STR_TO_WORD('1'); (* 1 *)
vDword1 := STR_TO_INT('1'); (* 1 *)
vDword1 := STR_TO_DWORD('1'); (* 1 *)
vDword1 := STR_TO_DINT('1'); (* 1 *)
vReal1 := STR_TO_REAL('1'); (* 1.000 *)
vTime1 := STR_TO_TIME('t#1s'); (* T#0d0h0m1s0ms *)
vDt1 := STR_TO_TOD('tod#1:1:1'); (* TOD#1:1:1.000 *)
vDt1 := STR_TO_DATE('d#2019-1-2'); (* D#2019-1-2 *)
vDt1 := STR_TO_DT('dt#2019-1-2-3:4:5'); (* DT#2019-1-2-3:4:5.000 *)

vDword1 := ADDROF(%QX10); (* 10 *)
vDword1 := SIZEOF(%QX10); (* 1 *)
vDword1 := CHG_ENDIAN(16#1122); (* 16#2211 *)
vAddr := ADDROF(gTest);
vLen := LEN(gTest);
vDword1 := EDC_SUM_BYTE(vAddr, vLen); (* 16#DD *)
vDword1 := EDC_SUM_WORD(vAddr, SHR(vLen, 1)); (* 16#D4D0 *)
vDword1 := EDC_SUM_DWORD(vAddr, SHR(vLen, 2)); (* 16#6C6A6866 *)
vDword1 := EDC_XOR_BYTE(vAddr, vLen); (* 16#31 *)
vDword1 := EDC_XOR_WORD(vAddr, SHR(vLen, 1)); (* 16#800 *)
vDword1 := EDC_XOR_DWORD(vAddr, SHR(vLen, 2)); (* 16#C040404 *)
vDword1 := EDC_CRC_8DARC(vAddr, vLen); (* 16#15 *)
vDword1 := EDC_CRC_8I4321(vAddr, vLen); (* 16#A1 *)
vDword1 := EDC_CRC_8ICODE(vAddr, vLen); (* 16#7E *)
vDword1 := EDC_CRC_8MAXIMDOW(vAddr, vLen); (* 16#A1 *)
vDword1 := EDC_CRC_8ROHC(vAddr, vLen); (* 16#D0 *)
vDword1 := EDC_CRC_8SMBUS(vAddr, vLen); (* 16#F4 *)
```

```
  vDword1 := EDC_CRC_8WCDMA(vAddr, vLen); (* 16#25 *)
  vDword1 := EDC_CRC_16ARC(vAddr, vLen); (* 16#BB3D *)
  vDword1 := EDC_CRC_16DDS110(vAddr, vLen); (* 16#9ECF *)
  vDword1 := EDC_CRC_16DECTR(vAddr, vLen); (* 16#7E *)
  vDword1 := EDC_CRC_16DNP(vAddr, vLen); (* 16#EA82 *)
  vDword1 := EDC_CRC_16EN13757(vAddr, vLen); (* 16#C2B7 *)
  vDword1 := EDC_CRC_16MODBUS(vAddr, vLen); (* 16#4B37 *)
  vDword1 := EDC_CRC_16UMTS(vAddr, vLen); (* 16#FEE8 *)
  vDword1 := EDC_CRC_32ISOHDLC(vAddr, vLen); (* 16#CBF43926 *)
  vAddr := 64;
  vDword1 := GET_BOOL(vAddr); (* 1 *)
  vDword1 := GET_BYTE(vAddr); (* 57 *)
  vDint1 := GET_SINT(vAddr); (* 57 *)
  vDword1 := GET_WORD(vAddr); (* 57 *)
  vDint1 := GET_INT(vAddr); (* 57 *)
  vDword1 := GET_DWORD(vAddr); (* 57 *)
  vDint1 := GET_DINT(vAddr); (* 57 *)
  vReal1 := GET_REAL(vAddr); (* 0.000 *)
  vTime1 := GET_TIME(vAddr); (* T#0d0h0m0s57ms *)
  vTod1 := GET_TOD(vAddr); (* TOD#0:0:0.057 *)
  vDate1 := GET_DATE(vAddr); (* D#1900-1-1 *)
  vDt1 := GET_DT(vAddr); (* DT#1900-1-1-0:0:0.057 *)
  vDword1 := SET_BOOL(vAddr, 20190102); (* 1 *)
  vDword1 := SET_BYTE(vAddr, 20190102); (* 150 *)
  vDint1 := SET_SINT(vAddr, 20190102); (* -106 *)
  vDword1 := SET_WORD(vAddr, 20190102); (* 5014 *)
  vDint1 := SET_INT(vAddr, 20190102); (* 5014 *)
  vDword1 := SET_DWORD(vAddr, 20190102); (* 20190102 *)
  vDint1 := SET_DINT(vAddr, 20190102); (* 20190102 *)
  vReal1 := SET_REAL(vAddr, 20190102); (* 20190102.000 *)
  vTime1 := SET_TIME(vAddr, t#1d2h3m4s); (* T#1d2h3m4s0ms *)
  vTod1 := SET_TOD(vAddr, dt#2019-1-2-3:4:5.678); (* TOD#3:4:5.678 *)
  vDate1 := SET_DATE(vAddr, dt#2019-1-2-3:4:5.678); (* D#2019-1-2 *)
  vDt1 := SET_DT(vAddr, dt#2019-1-2-3:4:5.678); (* DT#2019-1-2-3:4:5.678 *)

  vDword1 := 1;
END_PROGRAM

PROGRAM ProgMain
  VAR
    vInt1, vInt2, vInt3 : INT := 12;
    vInt4 : ARRAY[3] OF INT := 3;
```

```
  vDword1 : DWORD := 34;
  vStr1 : STRING;
END_VAR

vInt1 := SEL(1, vInt2, vDword1); (* 34 *)
vDword1 := MUX(1, vInt1, vInt2, vInt3); (* 12 *)
vStr1 := FunAct(vInt1); (* '34' *)
vStr1 := FunAct(-1); (* '34' *)
vStr1 := FunAct(1); (* 'Act number is 1.$l$r' *)
vStr1 := FunAct(6); (* '6' *)
vStr1 := FunAct(12); (* 'Just match!$l$r' *)

(* 다음은 반복 작업의 흐름을 이해하기 위한 내용입니다. *)
FOR vInt4[0] := 0 TO 3 DO
  vInt3 := vInt3 + 3;
  REPEAT
    vInt4[1] := vInt4[1] - 2;
    WHILE vInt4[2] <= 100 DO
      vInt4[2] := vInt4[2] + 1;
      EXIT;
    END_WHILE;
  UNTIL vInt4[1] > 0
  END_REPEAT;
END_FOR;

(* 다음은 시스템 명령의 사용 예입니다. *)
vStr1 := SEE_NW_IP();
vStr1 := SEE_NW_SSID();
vStr1 := SEE_NW_MODE();
vStr1 := SEE_NW_DOMAIN();
vDword1 := WA_ABLE_GEN(8, 8); (* FALSE *)
vDword1 := WA_ENABLE_GEN(8, 8); (* TRUE *)
vDword1 := WA_ABLE_GEN(8, 8); (* TRUE *)
vDword1 := WA_ABLE_GEN(7, 1); (* FALSE *)
vDword1 := WA_ABLE_GEN(16, 1); (* FALSE *)
vDword1 := WA_DISABLE_GEN(9, 3); (* TRUE *)
vDword1 := WA_ABLE_GEN(8, 8); (* FALSE *)
vDword1 := WA_ABLE_OUT(0, 2); (* FALSE *)
vDword1 := WA_ENABLE_OUT(0, 2); (* TRUE *)
vDword1 := WA_ABLE_OUT(0, 2); (* TRUE *)
vDword1 := WA_DISABLE_OUT(1, 1); (* TRUE *)
vDword1 := WA_ABLE_OUT(0, 2); (* FALSE *)
```

```
  vDword1 := WA_ABLE_OUT(0, 1); (* TRUE *)

  vDword1 := 0;
END_PROGRAM

FUNCTION FunAct : STRING
  VAR_INPUT
    iAct1 : INT;
  END_VAR
  VAR
    vStr1 : STRING;
  END_VAR

  IF iAct1 > 100 THEN
    RETURN;
  ELSIF iAct1 < 0 THEN
    RETURN;
  ELSE
    CASE iAct1 OF
      0, 1, 2, 3, 4, 5:
        vStr1 := CONCAT('Act number is ', STR_FROM_INT(iAct1), '.');
      12:
        vStr1 := 'Just match!';
    ELSE
      FunAct := STR_FROM_BYTE(iAct1);
      RETURN;
    END_CASE;
  END_IF;
  FunAct := CONCAT(vStr1, '$l$r');
END_FUNCTION
```

표준 FB와 PID FB의 사용 예를 pid0.dst에 작성합니다. UI0에 가변저항을 연결하여 목표 온도를 설정하고, UI1의 PT1000으로 현재 온도 측정합니다. PID를 1:1:1 비율로 제어하고, 온도 범위는 -200~800[℃]입니다. Integral windup 기능은 -100~400[℃] 밖에서 동작합니다. 세기를 제어할 수 있는 히터가 AO0에 연결됩니다. 5초 주기로 PID 제어를 수행합니다.

```
CONFIGURATION nameOfConf
  VAR_GLOBAL
    gPid : PID;
    gTsp AT %IW16 : INT; (* UI0 = 목표값 *)
    gTpv AT %IW17 : INT; (* UI1 = 현재값 *)
  END_VAR
```

```
  RESOURCE myDevice ON CPU
    TASK taskInit (SINGLE := TRUE, PRIORITY := 1);
    TASK taskSync (INTERVAL := t#5s, PRIORITY := 2);
    PROGRAM WITH taskInit : ProgInit();
    PROGRAM WITH taskSync : ProgMain();
  END_RESOURCE
END_CONFIGURATION

PROGRAM ProgInit
  gPid.Kp := 1;
  gPid.Ki := 1;
  gPid.Kd := 1;
  gPid.Hend := 8000;
  gPid.Lend := -2000;
  gPid.Hiw := 4000;
  gPid.Liw := -1000;
  gPid.Sp := 1000;
  gPid.Pv := 1000;
END_PROGRAM

PROGRAM ProgMain
  IF (%IX0 = 0) | (%IX1 = 0) THEN (* 온도 센서가 감지되지 않음. *)
    %QW16 := 0;
    RETURN;
  END_IF;

  gPid(Sp := gTsp, Pv := gTpv);
  IF (gPid.EC <> 0) | (gPid.Cv <= 0) THEN (* 오류 발생, 또는 측정값이 설정값을 넘침. *)
    %QW16 := 0;
    RETURN;
  END_IF;

  %QW16 := gPid.Cv * 100; (* AO0은 제어값 *)
END_PROGRAM
```

[DST 2] pid0.dst

MODBUS RTU 통신의 사용 예를 rtu0.dst(slave)와 rtu1.dst(master)로 작성합니다. 시험을 위해 2개의 E5A가 필요합니다. RS485로 2개의 E5A를 연결합니다. 통신 설정은 9600[bps], N/8/1입니다.

Slave 역할을 하는 E5A는 내부 메모리와 출력 메모리 중 일부를 쓰기 가능으로 변경하고 대기합니다. 장치의 slave ID는 11로 설정합니다.

Master 역할을 하는 E5A는 slave E5A에게 사용 가능한 MODBUS protocol을 순서대로 하나씩 시도합니다.

```
CONFIGURATION nameOfConf
  RESOURCE extLine1 ON SER_1
    VAR_GLOBAL
      gConf : CONF_SER1;
    END_VAR

    TASK taskInit (SINGLE := TRUE, PRIORITY := 1);
    TASK taskSync (INTERVAL := t#5s, PRIORITY := 2);
    PROGRAM pgm1Init WITH taskInit : Prog1Init();
    PROGRAM WITH taskSync : Prog1Sync();
  END_RESOURCE
END_CONFIGURATION

PROGRAM Prog1Init
  VAR
    loBool : BOOL;
  END_VAR

  (* 통신 설정 내용: SET, MODBUS RTU slave mode, slave로 동작할 때 ID는 11, 9600/N/8/1
*)
  extLine1.gConf(GET_SET := 1, MODE := 2, SLAVE_ID := 11, RATE := 9600, PARITY := 0,
DATABITS := 8, STOPBIT := 1);

  (* 작업 영역에 대한 쓰기 방지 기능 해제  *)
  loBool := WA_ENABLE_GEN(0, 16);
  loBool := WA_ENABLE_GEN(2048 * 16, 16);
  loBool := WA_ENABLE_GEN(4095 * 16, 16);

  loBool := WA_ENABLE_OUT(0, 1);
  loBool := WA_ENABLE_OUT(16384, 1);
  loBool := WA_ENABLE_OUT(32767, 1);
  loBool := WA_ENABLE_OUT(32768, 1);

  loBool := WA_ENABLE_OUT(16383, 2);
  IF loBool = FALSE THEN
    loBool := WA_ENABLE_OUT(16383, 1);
  END_IF;
  loBool := WA_ENABLE_OUT(16384, 2);
  loBool := WA_ENABLE_OUT(32766, 2);
  loBool := WA_ENABLE_OUT(32767, 2);
```

```
  loBool := WA_ENABLE_OUT(0, 16);
  loBool := WA_ENABLE_OUT(1024 * 16, 16);
  loBool := WA_ENABLE_OUT(2047 * 16, 16);
  loBool := WA_ENABLE_OUT(4095 * 16, 16);

  loBool := WA_ENABLE_OUT(1023 * 16, 2 * 16);
  IF loBool = FALSE THEN
    loBool := WA_ENABLE_OUT(1023 * 16, 16);
  END_IF;
  loBool := WA_ENABLE_OUT(1024 * 16, 2 * 16);
  loBool := WA_ENABLE_OUT(2046 * 16, 2 * 16);
  loBool := WA_ENABLE_OUT(2047 * 16, 2 * 16);
END_PROGRAM

PROGRAM Prog1Sync
  VAR
    vDw : DWORD;
  END_VAR

  vDw := 0;
END_PROGRAM
```

[DST 3] rtu0.dst

```
CONFIGURATION nameOfConf
  TYPE T_CMD:
    STRUCT
      rw : BOOL; (* 읽기/쓰기 *)
      sar : USINT; (* 영역 *)
      sad : UINT; (* 주소 *)
    END_STRUCT
  END_TYPE

  VAR_GLOBAL
    gWait : BOOL;
    gSar, gSad, gRw : SINT;
    gWord0, gWord1 : WORD;
    gRtc : RTC;
    gCmd : ARRAY[5,3,2] OF T_CMD; (* area, addr, r/w *)
  END_VAR

  RESOURCE extLine1 ON SER_1
```

```
    VAR_GLOBAL
      gConf : CONF_SER1;
      gComm : COMM_SER1;
    END_VAR

    TASK taskInit (SINGLE := TRUE, PRIORITY := 1);
    TASK taskSync (INTERVAL := t#5s, PRIORITY := 2);
    PROGRAM pgm1Init WITH taskInit : Prog1Init();
    PROGRAM WITH taskSync : Prog1Sync();
  END_RESOURCE
END_CONFIGURATION

PROGRAM Prog1Init
  gWait := FALSE; (* 통신 상태 초기화 *)
  (* 통신 설정 내용: SET, MODBUS RTU master mode, slave로 동작할 때 ID는 1,
9600/N/8/1 *)
  extLine1.gConf(GET_SET := 1, MODE := 1, SLAVE_ID := 1, RATE := 9600, PARITY := 0,
DATABITS := 8, STOPBIT := 1);
  (* 작업 초기화 *)
  gSar := 0;
  gSad := 0;
  gRw := 1;
  extLine1.gComm.EC := 0; (* 통신 결과 초기화 *)
  extLine1.gComm.SLAVE_ID := 11;
  extLine1.gComm.DATA_COUNT := 1;
  extLine1.gComm.MEM_ADDR := ADDROF(gWord0);

  (* 명령 구조체 초기화 *)
  gCmd[0,0,0].sar := 1; (* coil status, begin of range, read *)
  gCmd[0,0,0].sad := 0;
  gCmd[0,0,0].rw := 0;
  gCmd[0,0,1].sar := 1; (* coil status, begin of range, write *)
  gCmd[0,0,1].sad := 0;
  gCmd[0,0,1].rw := 1;
  gCmd[0,1,0].sar := 1; (* coil status, end of range, read *)
  gCmd[0,1,0].sad := 16383;
  gCmd[0,1,0].rw := 0;
  gCmd[0,1,1].sar := 1; (* coil status, end of range, write *)
  gCmd[0,1,1].sad := 16383;
  gCmd[0,1,1].rw := 1;
  gCmd[0,2,0].sar := 1; (* coil status, out of range, read *)
  gCmd[0,2,0].sad := 16384;
```

```
gCmd[0,2,0].rw := 0;
gCmd[0,2,1].sar := 1; (* coil status, out of range, write *)
gCmd[0,2,1].sad := 16384;
gCmd[0,2,1].rw := 1;
gCmd[1,0,0].sar := 2; (* input status, begin of range, read *)
gCmd[1,0,0].sad := 0;
gCmd[1,0,0].rw := 0;
gCmd[1,0,1].sar := 2; (* input status, begin of range, read *)
gCmd[1,0,1].sad := 1;
gCmd[1,0,1].rw := 0;
gCmd[1,1,0].sar := 2; (* input status, end of range, read *)
gCmd[1,1,0].sad := 16383;
gCmd[1,1,0].rw := 0;
gCmd[1,1,1].sar := 2; (* input status, end of range, read *)
gCmd[1,1,1].sad := 16382;
gCmd[1,1,1].rw := 0;
gCmd[1,2,0].sar := 2; (* input status, out of range, read *)
gCmd[1,2,0].sad := 16384;
gCmd[1,2,0].rw := 0;
gCmd[1,2,1].sar := 2; (* input status, out of range, read *)
gCmd[1,2,1].sad := 16385;
gCmd[1,2,1].rw := 0;
gCmd[2,0,0].sar := 3; (* holding register, begin of range, read *)
gCmd[2,0,0].sad := 0;
gCmd[2,0,0].rw := 0;
gCmd[2,0,1].sar := 3; (* holding register, begin of range, write *)
gCmd[2,0,1].sad := 0;
gCmd[2,0,1].rw := 1;
gCmd[2,1,0].sar := 3; (* holding register, end of range, read *)
gCmd[2,1,0].sad := 2047;
gCmd[2,1,0].rw := 0;
gCmd[2,1,1].sar := 3; (* holding register, end of range, write *)
gCmd[2,1,1].sad := 2047;
gCmd[2,1,1].rw := 1;
gCmd[2,2,0].sar := 3; (* holding register, out of range, read *)
gCmd[2,2,0].sad := 2048;
gCmd[2,2,0].rw := 0;
gCmd[2,2,1].sar := 3; (* holding register, out of range, write *)
gCmd[2,2,1].sad := 2048;
gCmd[2,2,1].rw := 1;
gCmd[3,0,0].sar := 4; (* input register, begin of range, read *)
gCmd[3,0,0].sad := 0;
```

```
  gCmd[3,0,0].rw := 0;
  gCmd[3,0,1].sar := 4; (* input register, end of range, read *)
  gCmd[3,0,1].sad := 1023;
  gCmd[3,0,1].rw := 0;
  gCmd[3,1,0].sar := 4; (* input register, out of range, read *)
  gCmd[3,1,0].sad := 1024;
  gCmd[3,1,0].rw := 0;
  gCmd[3,1,1].sar := 4; (* input register, design year, read *)
  gCmd[3,1,1].sad := 9900;
  gCmd[3,1,1].rw := 0;
  gCmd[3,2,0].sar := 4; (* input register, MAC 5~6, read *)
  gCmd[3,2,0].sad := 9912;
  gCmd[3,2,0].rw := 0;
  gCmd[3,2,1].sar := 4; (* input register, lot, read *)
  gCmd[3,2,1].sad := 9991;
  gCmd[3,2,1].rw := 0;
  gCmd[4,0,0].sar := 0; (* general register, begin of range, read *)
  gCmd[4,0,0].sad := 0;
  gCmd[4,0,0].rw := 0;
  gCmd[4,0,1].sar := 0; (* general register, begin of range, write *)
  gCmd[4,0,1].sad := 0;
  gCmd[4,0,1].rw := 1;
  gCmd[4,1,0].sar := 0; (* general register, end of range, read *)
  gCmd[4,1,0].sad := 8191;
  gCmd[4,1,0].rw := 0;
  gCmd[4,1,1].sar := 0; (* general register, end of range, write *)
  gCmd[4,1,1].sad := 8191;
  gCmd[4,1,1].rw := 1;
  gCmd[4,2,0].sar := 0; (* general register, out of range, read *)
  gCmd[4,2,0].sad := 8192;
  gCmd[4,2,0].rw := 0;
  gCmd[4,2,1].sar := 0; (* general register, out of range, write *)
  gCmd[4,2,1].sad := 8192;
  gCmd[4,2,1].rw := 1;
END_PROGRAM

PROGRAM Prog1Sync
  VAR
    p1sDt : DT;
  END_VAR

  IF gWait = TRUE & extLine1.gComm.EC = 1 THEN
```

```
    (* 통신 결과가 나올 때까지 기다림. *)
    RETURN;
  END_IF;

  IF gWait = FALSE THEN
    (* 통신이 종료된 상태 *)
    extLine1.gComm.SLAVE_AREA := gCmd[gSar, gSad, gRw].sar;
    extLine1.gComm.SLAVE_ADDR := gCmd[gSar, gSad, gRw].sad;
    extLine1.gComm(READ_WRITE := gCmd[gSar, gSad, gRw].rw);
    gWait := TRUE; (* 통신 결과 대기 상태로 전환. *)
    RETURN;
  ELSE
    (* 통신 결과 대기 상태 *)
    IF extLine1.gComm.EC = 0 THEN (* 통신이 정상 종료됨. *)
      gRtc(EN := 0);
      p1sDt := gRtc.CDT; (* 통신 정상 완료 시각 저장 *)
    END_IF;
    gWait := FALSE; (* 통신 종료 상태로 전환. *)

    gWord1 := gWord0 + 1;

    (* 명령 자동 전환 *)
    gRw := gRw + 1;
    IF gRw >= 2 THEN
      gRw := 0;
      gSad := gSad + 1;
      IF gSad >= 3 THEN
        gSad := 0;
        gSar := gSar + 1;
        IF gSar >= 5 THEN
          gSar := 0;
        END_IF;
      END_IF;
    END_IF;

    (* 다음 작업이 write인 경우를 대비해서 쓸 값을 '현재값 + 1'로 지정함. *)
    gWord0 := gWord1;
  END_IF;
END_PROGRAM
```

[DST 4] rtu1.dst

MODBUS TCP 통신의 사용 예를 tcp0.dst(server)와 tcp1.dst(client)로 작성합니다. 시험을 위해 2개의 E5A가 필요합니다. Wifi를 통해 2개의 E5A가 연결됩니다. 통신 설정은 E5A 중 하나를 AP 모드로 운용하거나 하나의 무선 라우터에 station으로 연결합니다.

Server 역할을 하는 E5A는 내부 메모리와 출력 메모리 중 일부를 쓰기 가능으로 변경하고, 대기합니다. RESOURCE ETH_1에서 동작합니다. TCP port는 502를 엽니다.

Master 역할을 하는 E5A는 slave E5A의 한 영역에서 4가지 주소를 지정하여 순서대로 하나씩 시도합니다. RESOURCE ETH_2에서 동작합니다. Server의 IP address는 192.168.0.21입니다.

```
CONFIGURATION nameOfConf
  RESOURCE extLine1 ON ETH_1
    VAR_GLOBAL
      gConf : CONF_ETH1;
    END_VAR

    TASK taskInit (SINGLE := TRUE, PRIORITY := 1);
    TASK taskSync (INTERVAL := t#5s, PRIORITY := 2);
    PROGRAM pgm1Init WITH taskInit : Prog1Init();
    PROGRAM WITH taskSync : Prog1Sync();
  END_RESOURCE
END_CONFIGURATION

PROGRAM Prog1Init
  VAR
    loBool : BOOL;
  END_VAR

  (* 통신 설정 내용: SET, MODBUS TCP server mode, IPv4, port 502 *)
  extLine1.gConf(GET_SET := 1, MODE := 1, IPV4_6 := 0, PORT := 502);

  (* 작업 영역에 대한 쓰기 방지 기능 해제  *)
  loBool := WA_ENABLE_GEN(0, 16);
  loBool := WA_ENABLE_GEN(2048 * 16, 16);
  loBool := WA_ENABLE_GEN(4095 * 16, 16);

  loBool := WA_ENABLE_OUT(0, 1);
  loBool := WA_ENABLE_OUT(16384, 1);
  loBool := WA_ENABLE_OUT(32767, 1);
  loBool := WA_ENABLE_OUT(32768, 1);

  loBool := WA_ENABLE_OUT(16383, 2);
  IF loBool = FALSE THEN
```

```
    loBool := WA_ENABLE_OUT(16383, 1);
  END_IF;
  loBool := WA_ENABLE_OUT(16384, 2);
  loBool := WA_ENABLE_OUT(32766, 2);
  loBool := WA_ENABLE_OUT(32767, 2);

  loBool := WA_ENABLE_OUT(0, 16);
  loBool := WA_ENABLE_OUT(1024 * 16, 16);
  loBool := WA_ENABLE_OUT(2047 * 16, 16);
  loBool := WA_ENABLE_OUT(4095 * 16, 16);

  loBool := WA_ENABLE_OUT(1023 * 16, 2 * 16);
  IF loBool = FALSE THEN
    loBool := WA_ENABLE_OUT(1023 * 16, 16);
  END_IF;
  loBool := WA_ENABLE_OUT(1024 * 16, 2 * 16);
  loBool := WA_ENABLE_OUT(2046 * 16, 2 * 16);
  loBool := WA_ENABLE_OUT(2047 * 16, 2 * 16);
END_PROGRAM

PROGRAM Prog1Sync
  VAR
    vDw : DWORD;
  END_VAR

  vDw := 0;
END_PROGRAM
```

[DST 5] tcp0.dst

```
CONFIGURATION nameOfConf
  TYPE T_CMD:
    STRUCT
      rw : BOOL; (* 읽기/쓰기 *)
      sad : UINT; (* 주소 *)
    END_STRUCT
  END_TYPE

  VAR_GLOBAL
    gWait : BOOL;
    gTest, gCntTest : SINT;
    gWord0, gWord1 : WORD;
    gRtc : RTC;
```

```
    gCmd : ARRAY[4] OF T_CMD;
  END_VAR

  RESOURCE extLine1 ON ETH_2
    VAR_GLOBAL
      gConf : CONF_ETH2;
      gComm : COMM_ETH2;
    END_VAR

    TASK taskInit (SINGLE := TRUE, PRIORITY := 1);
    TASK taskSync (INTERVAL := t#5s, PRIORITY := 2);
    PROGRAM pgm1Init WITH taskInit : Prog1Init();
    PROGRAM WITH taskSync : Prog1Sync();
  END_RESOURCE
END_CONFIGURATION

PROGRAM Prog1Init
  gWait := FALSE; (* 통신 상태 초기화 *)
  (* 통신 설정 내용: SET, MODBUS TCP client mode, unit ID는 1, IPv4, server 주소는
192.168.0.21:502 *)
  extLine1.gConf(GET_SET := 1, MODE := 1, UNIT_ID := 1, IPV4_6 := 0, HOST :=
'192.168.0.21', PORT := 502);
  (* 작업 초기화 *)
  gTest := 0;
  gCntTest := 4;
  extLine1.gComm.EC := 0; (* 통신 결과 초기화 *)
  (* Area 0=general reference, 1=coil status, 2=input status, 3=holding register, 4=input
register *)
  extLine1.gComm.SERVER_AREA := 0;
  extLine1.gComm.DATA_COUNT := 1;
  extLine1.gComm.MEM_ADDR := ADDROF(gWord0);

  (* 명령 구조체 초기화 *)
  (* R/W 0=read, 1=write *)
  gCmd[0].rw := 1; (* step 0 *)
  gCmd[0].sad := 0;
  gCmd[1].rw := 1; (* step 1 *)
  gCmd[1].sad := 2048;
  gCmd[2].rw := 1; (* step 2 *)
  gCmd[2].sad := 4095;
  gCmd[3].rw := 1; (* step 3 *)
  gCmd[3].sad := 4096;
```

```
END_PROGRAM

PROGRAM Prog1Sync
  VAR
    p1sDt : DT;
    p1sDw : DWORD;
  END_VAR

  IF gWait = TRUE & extLine1.gComm.EC = 1 THEN (* 통신 결과가 나올 때까지 기다리는 중 *)
    RETURN;
  END_IF;

  IF gWait = FALSE THEN
    (* 통신이 종료된 상태 *)
    extLine1.gComm.SERVER_ADDR := gCmd[gTest].sad;
    extLine1.gComm(READ_WRITE := gCmd[gTest].rw);
    gWait := TRUE; (* 통신 결과 대기 상태로 전환. *)
    RETURN;
  ELSE
    (* 통신 결과 대기 상태 *)
    IF extLine1.gComm.EC = 0 THEN (* 통신이 정상 종료됨. *)
      gRtc(EN := 0);
      p1sDt := gRtc.CDT; (* 통신 정상 완료 시각 저장 *)
    END_IF;
    gWait := FALSE; (* 통신 종료 상태로 전환. *)

    gWord1 := gWord0 + 1;

    (* 명령 자동 전환 *)
    gTest := gTest + 1;
    IF gTest >= gCntTest THEN
      gTest := 0;
    END_IF;

    (* 다음 작업이 write인 경우를 대비해서 쓸 값을 '현재값 + 1'로 지정함. *)
    gWord0 := gWord1;
  END_IF;
END_PROGRAM
```

[DST 6] tcp1.dst