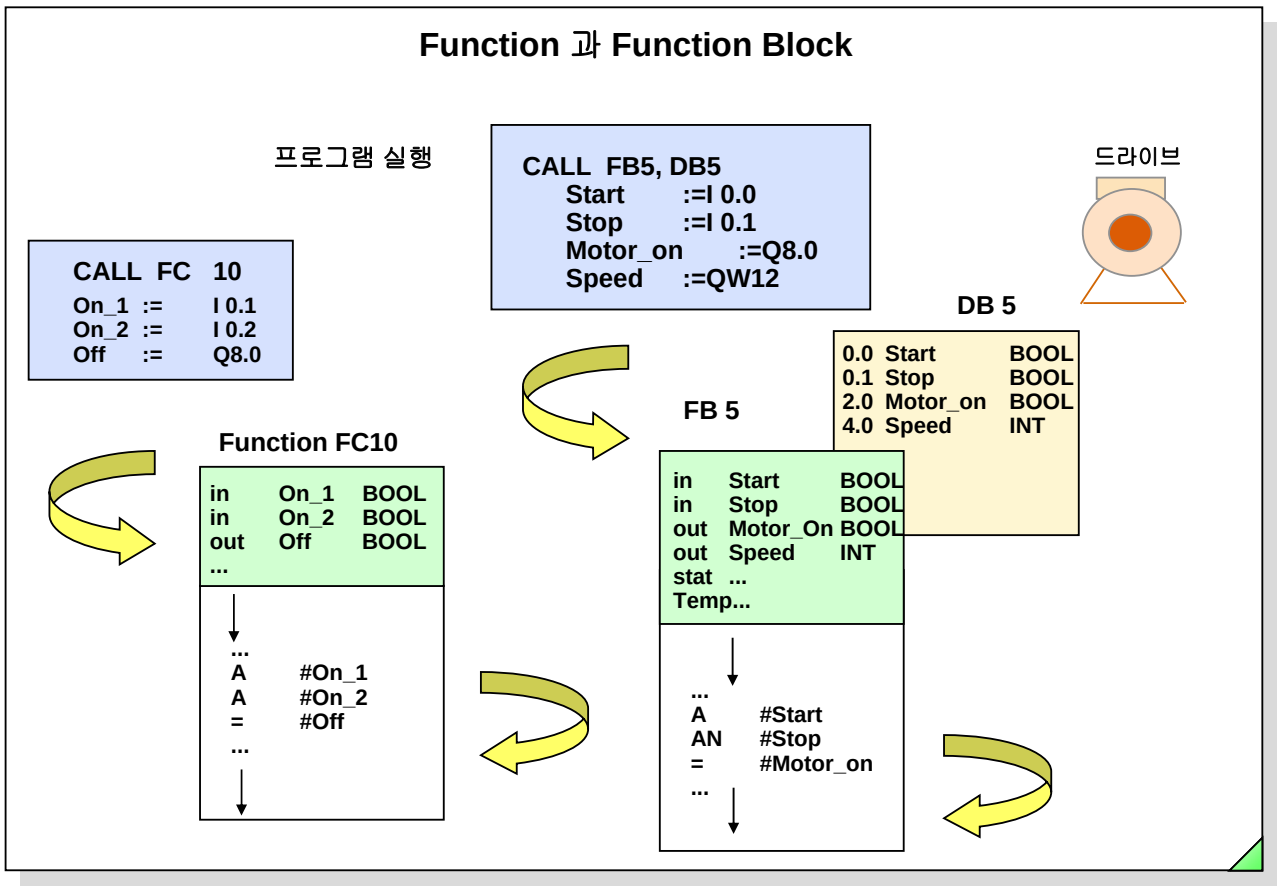


Date: 22.03.2006
File: PRO1_12E.1

차 례

교육 목표

... 파라미터를 지정하는 블록을 이해합니다.

... 파라미터를 지정하는 FC 를 생성하고 호출합니다.

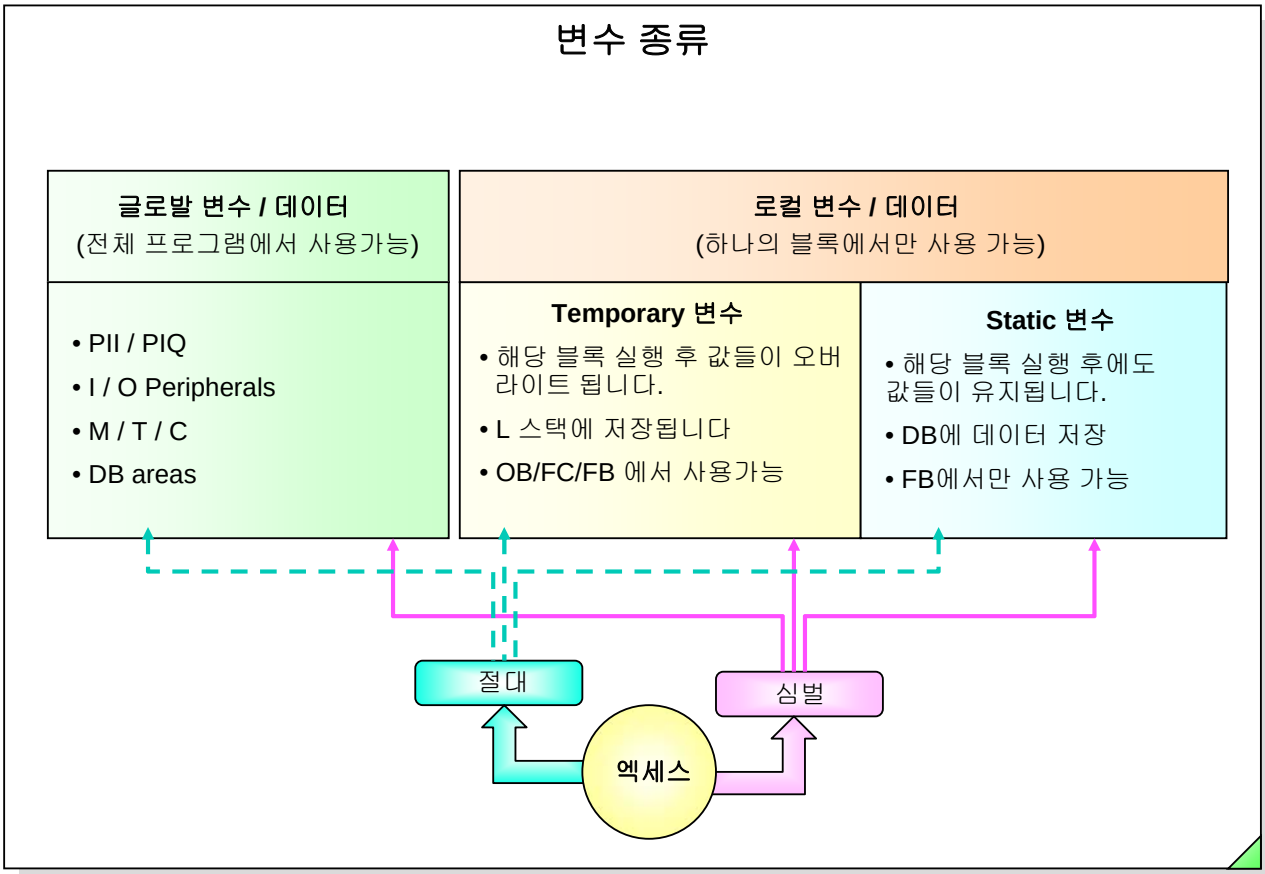
... FC 와 FB 차이를 이해합니다

... Static 변수 사용을 이해합니다.

... Static 변수를 선언하고 프로그램에 적용합니다.

... 파라미터를 지정할 수 있는 FB 를 생성하고 호출합니다.

Date: 22.03.2006
File: PRO1_12E.2



개 요

지금까지 프로그램에서는 입력과 출력이 실제 어드레스로 직접 할당이 된채 작 성되었습니다. 이러한 유형의 프로그램은 어드레스 위치에 고정이 되기 때문에 반복적인 프로세스에는 잘 맞지 않습니다. 파라미터를 지정하지 않는 블록들은 반복적인 프로세스가 없는 기계 제작에 적합합니다.

대형 시스템에서 반복적인 프로세스는 재사용할 수 있고 파라미터를 설정할 수 있는 블록들(Function 과 Function Block) 을 사용하는 것이 효과적입니다. 이러한 블록들은 심벌 입력과 출력 파라미터(로컬 심벌 이름)를 사용하며 블록이 호출될 때 실제 피연산자와 함께 사용됩니다.

사용자는 이러한 피연산자를 Function 이나 Function Block 을 호출할 때 할당 해 야 합니다. FC/FB 의 프로그램 로직은 변경되지 않기 때문에 여러번 재 사용할 수 있습니다.

로컬 변수

지금까지는 생산 데이터 등을 저장하기 위해 글로벌 변수 (비트 메모리와 데이터 블록)를 사용했습니다. 이 단원에서는 로컬 변수라는 데이터 저장 장소를 접하게 됩니다.

로컬 변수는 그것들이 생성된 블록에서만 읽을 수 있으며 이로 인해 다른 프로그램 블록들의 로컬 변수와는 데이타 교환을 할 수 없습니다.

두가지 유형의 로컬 변수가 있습니다. - Temporary 와 Static

Temporary 변수 :

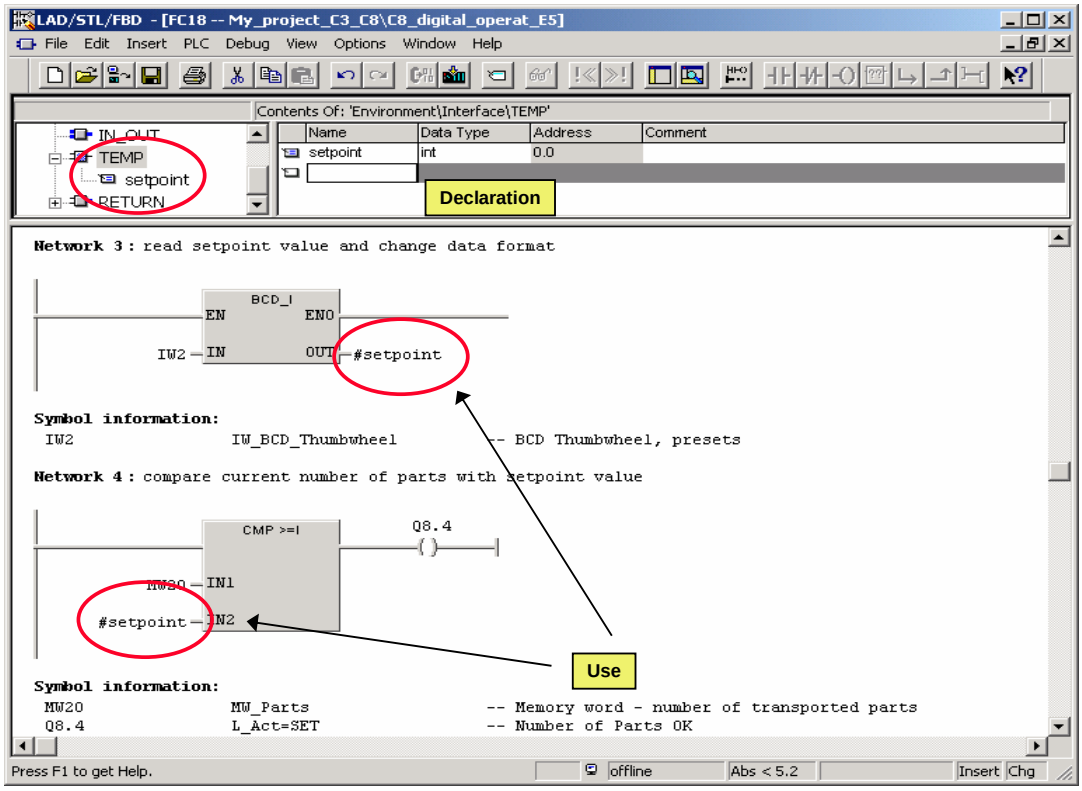
Temporary 변수는 블록이 실행되는 동안에만 저장되는 변수입니다. 모든 프로그 램 블록(OB, FB, FC)에서 선언될 수 있습니다.

Static 변수 :

데이터가 블록이 수행이 된 이후에도 유지가 되어야 한다면 데이터는 Static 변수 에 저장이 되어야 합니다.

Static 변수는 Function Block 에서만 선언이 됩니다. FB에 할당된 인스턴스 DB 가 이러한 Static 변수를 저장하는 장소로 사용이 됩니다.

Temporary 변수



SIMATIC S7

Date: 22.03.2006
File: PRO1_12E.4

SITRAIN Training for Automation and Drives

개 요

Temporary 변수는 모든 프로그램 블록에서 사용될 수 있습니다. 이 변수는 블록이 수행되는 동안 정보를 임시로 저장하기 위해 사용됩니다. 데이터는 블록 프로그램 을 마치고 나면 잃어버리게 됩니다.

데이터는 L 스택 (로컬 데이타 스택) 에 저장됩니다. 이것은 CPU 내 별도의 메모리 영역입니다.

선언부

블록의 선언 테이블에서 변수들을 선언합니다. "TEMP" 행에서 변수 이름과 관련 데이타 타입을 입력합니다. 여기서는 초기값(Initial Value)을 미리 선언할 수 없습니다.

블록을 저장한 후에 L 스택의 메모리 위치가 "Address" 컬럼에 디스플레이됩니다.

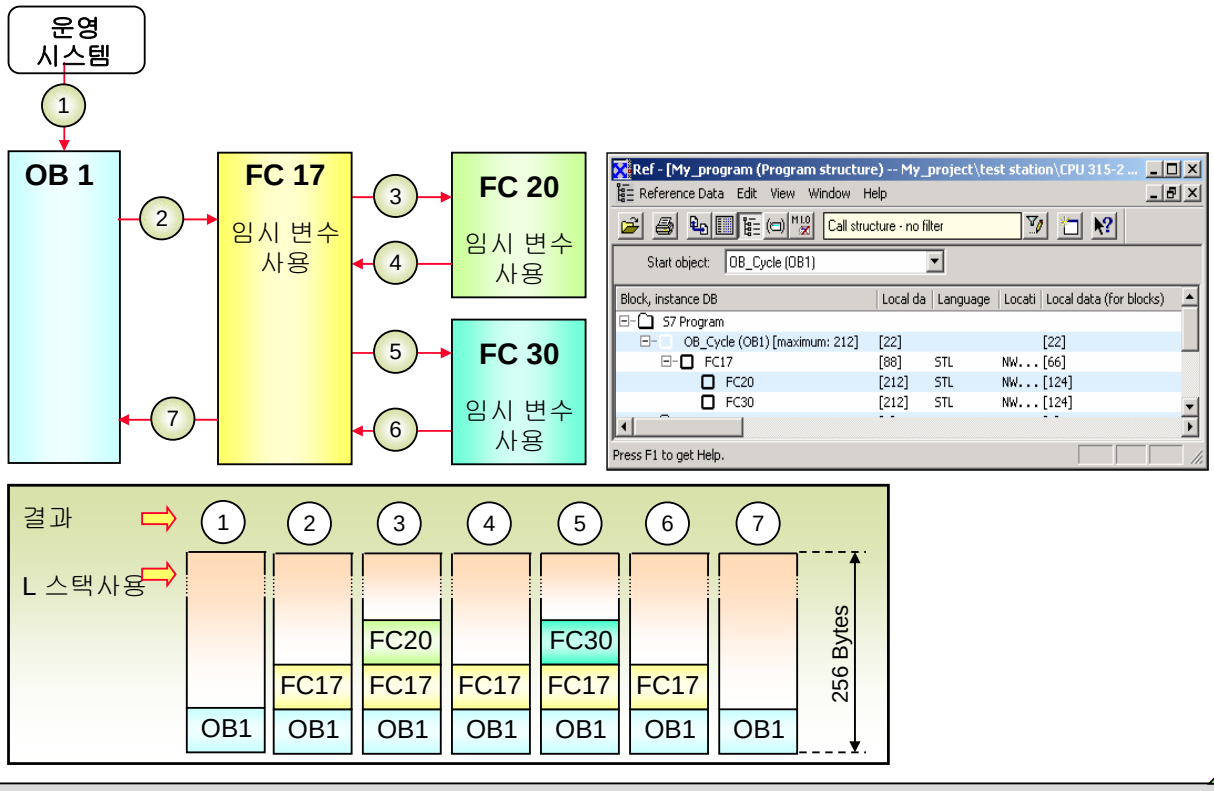
엑세스

네트워크(렁) 1 에서 Temporary 변수에 대해 심벌로 엑세스를 하는 예를 볼 수 있습니다. 빼기의 결과가 Temporary 변수 "Result" 에 저장됩니다. 사용자는 절대 어드레스를 입력할 수 있습니다(T LW0). 그러나 프로그램이 읽기가 더 어렵기 때문 에 절대 어드레스를 피하는 것이 좋습니다.

참 조

문자로 시작하는 변수는 로컬 변수이며 선언 테이블에서 정의된 블록에서 만 사용할 수 있습니다. 프로그램 편집기가 자동적으로 # 글자를 입력시켜줍니다.

로컬 데이터 스택 전체 사용량



전체 사용량

모든 프로그램 실행이나 혹은 우선권 등급 (예를들면 OB1과 그안에서 호출된 모든 블록)을 위해 별도의 L 스택이 예비되어 있습니다.(CPU 의 L 스택에 할당 된 세그먼트 크기) 즉, OB1에 의해 호출된 블록들 가운데 사용된 로컬 변수나 Temporary 변수 뿐 아니라 OB1 의 로컬 변수가 L 스택에 저장이 됩니다. 레퍼런스 데이터 기능을 사용하여 S7 프로그램이 L 스택에 얼마나 많은 부하를 주고 있는지를 볼 수 있습니다. 레퍼런스 데이타는 "고장 탐구 (Troubleshooting)" 단원에서 더 자세히 다룹니다.

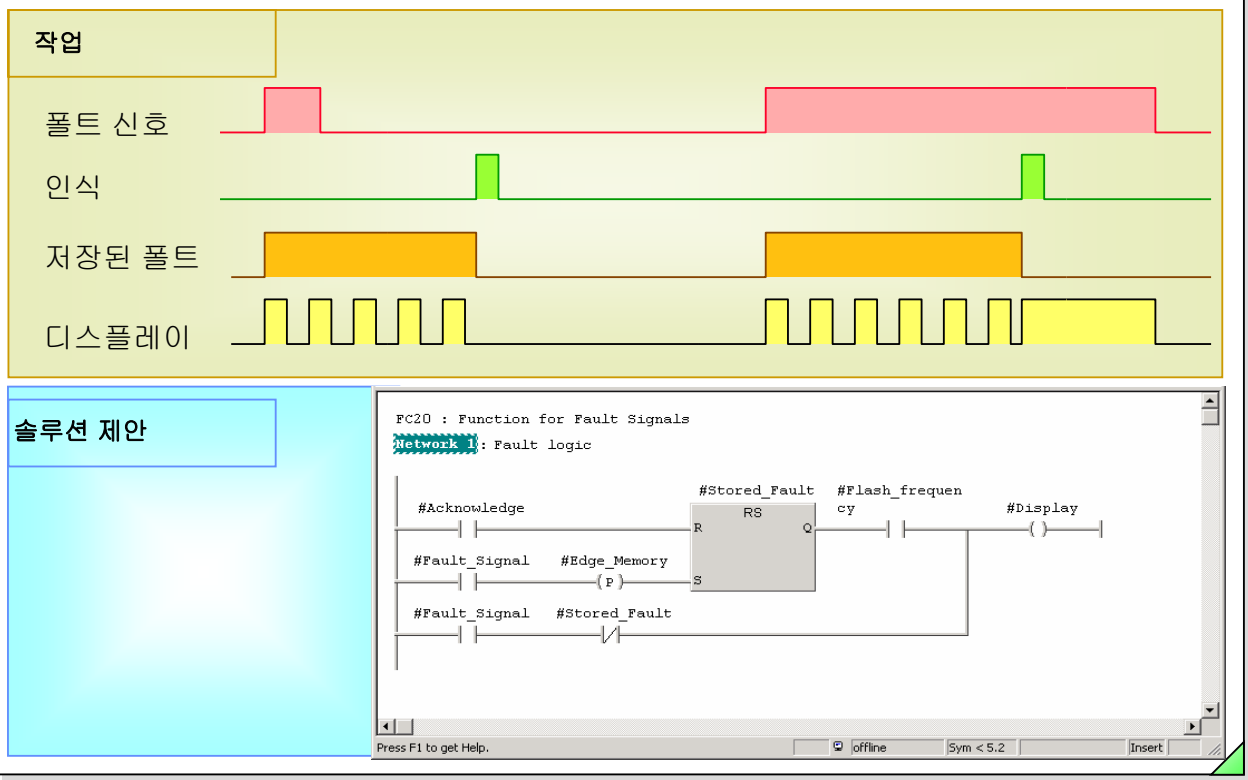
레퍼런스 데이타

SIMATIC Manager에서 블록 폴더를 선택하고 다음 메뉴를 선택합니다.
Options -> Reference Data -> Display -> Program Structure

참 조

프로그램 실행 중에 최대 로컬 데이터가 초과가 되면(L 스택 오버플로우) CPU 가 STOP 이 됩니다. 진단 버퍼(Diagnostic buffer) 에는 "STOP caused by error when allocating local data" 가 에러의 원인으로 입력이 됩니다.

예 : 출력 LED 로 오류 신호 디스플레이



Date: 22.03.2006
File: PRO1_12E.6

설 명

발생한 문제는 오퍼레이터 콘솔에 있는 LED 에 디스플레이가 됩니다. 문제가 발생하면 (I 0.0), LED (Q 5.1) 이 2 Hz 로 깜박거리게 됩니다.

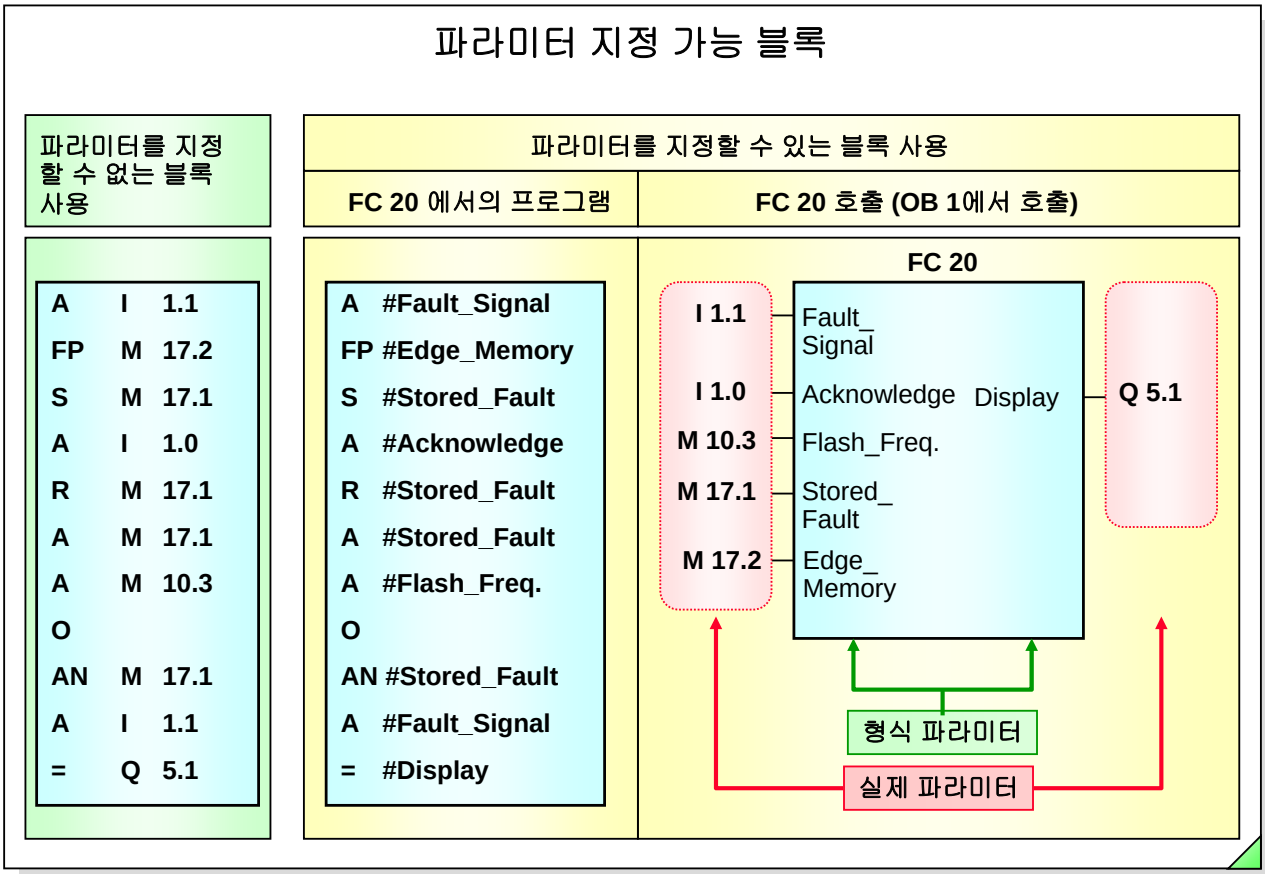
문제는 입력 I1.2 에서 인식이 됩니다. 문제가 수정이 되면 LED 는 깜박거리지 않 게 됩니다. 문제가 계속되면 LED 는 문제가 해결될 때까지 켜진 상태로 멈추게 됩니다.

문 제

오류 신호의 RLO 에지에 대한 검출이 이루어지게 됩니다. 이는 존재하는 문제가 인식이 되면 메모리가 즉시 리셋이 되기 때문입니다.

메모리가 세트가 되면 (메시지는 아직 인식이 되지 않았습니다) 상부의 AND 명령어로 인해 LED가 깜박거리게 됩니다. 이로 인해 하드웨어 설정에서 클럭 메모 리 (Clock Memory) 로 정의된 비트 메모리 M 10.3 이 동작을 하게 됩니다.

하부 AND 명령어로 인해 인식은 되었지만 아직 존재하는 문제가 있을 때 LED 가 켜진 상태로 멈추게 됩니다.



SIMATIC S7

Date: 22.03.2006
File: PRO1_12E.7

SITRAIN Training for Automation and Drives

어플리케이션

자주 반복되는 프로그램에는 파라미터 지정 블록을 사용할 수 있습니다.

- 프로그램이 한번만 생성이 되면 되기 때문에 프로그래밍 시간을 크게 줄일 수 있습니다.
- 블록이 사용자 메모리에 한번만 저장이 되면 되기 때문에 전체 메모리 양을 크게 줄일 수 있습니다.
- 블록은 사용자가 원하면 언제든 호출될 수 있으며 각각 다른 어드레스가 할당이 됩니다. 이를 위해 형식 파라미터 (Formal Parameter)에는 블록이 호출될 때 마다 다른 실제 어드레스 (Actual address)가 할당이 됩니다.

프로그램 실행

위 예제에서 STL 언어를 볼 수 있으며 프로그램 실행을 따라가기가 더 쉽습니다. STL 코드는 이전 예제와 같은 오류 로직을 수행합니다. STL 프로그래밍 언어는 고급 과정에서 더 자세히 다루어질 예정입니다.

위에서 보는 블록이 실행이 되고 A #Acknowledge" 가 수행이 되면 Acknowledge 파라미터가 실제 파라미터로 바뀌어지게 됩니다. 입력 I 1.1 이 Acknowledge 파라미터의 실제 파라미터로 주어지게 되면 FC 20 프로그램 블록에서 보는 "A #Acknowledge" 대신에 A I 1.1 이 들어가게 됩니다.

파라미터 설정

FC 나 FB 블록을 파라미터 지정 블록으로 프로그램할 수 있습니다. OB 블록은 운영 시스템에서 직접 호출이 되기 때문에 파라미터를 지정할 수 있는 프로그램 을 생성할 수 없습니다. 사용자 프로그램에서는 OB 블록 호출이 이루어질 수 없 기 때문에 실제 어드레스를 OB 로 전달하는 것이 불가능합니다.

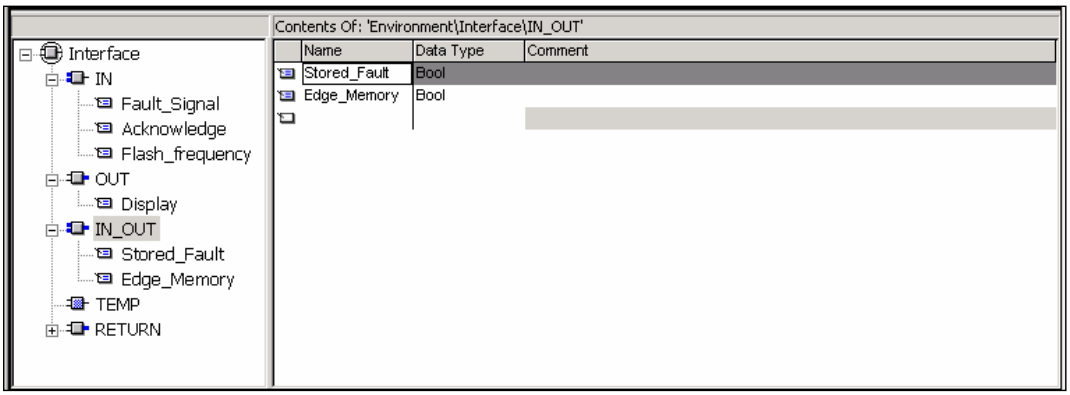
예

시스템에서 서브루틴이 두번 필요하다고 해도 파라미터 지정 블록의 FC20 은 한번만 프로그램하면 됩니다. FC 20 은 두개의 서로 다른 오류에 대해 호출할 수 있고 매번 다른 실제 어드레스를 할당하게 됩니다.

FC 20 에서의 형식 파라미터 정의

형식 파라미터

파라미터 타입	선언	사용	그래픽 디스플레이
입력 파라미터	in	읽기만 해당	블록의 왼쪽
출력 파라미터	out	쓰기만 해당	블록의 오른쪽
입/출력 파라미터	In_out	읽기 / 쓰기	블록 왼쪽



Formal Parameter

파라미터 지정 블록 프로그램을 생성하기 전에 먼저 선언부 테이블에서 형식 파라미터(Formal Parameter)를 정의해야 합니다.

파라미터 유형

슬라이드의 테이블에서 세개의 다른 파라미터 타입이 블록에 사용되고 있는 것 을 볼 수 있습니다. 사용자는 원하는 대로 형식 파라미터에 대한 유형 선언을 할 수 있습니다. "in" 선언 유형은 서브루틴에서 "읽기" 선언 유형이 지정되어야 합니다. "out" 선언 유형은 서브루틴에서 "쓰기" 선언 유형이 지정되어야 합니다. 읽기 엑세스(A,O,L 명령어 사용)를 위한 형식 파라미터과 동시에 쓰기 엑세스 (S,R,T) 를 위한 형식 파라미터는 "in/out" 파라미터에 선언이 되어야 합니다.

인터페이스

블록의 인터페이스는 IN, OUT, IN_OUT 파라미터를 갖습니다. RETURN 파라미 터는 IEC 61131-3 에 따라 특정하게 지정된 부가적인 OUT 파라미터입니다. 이 파라미터는 FC 에서만 존재합니다.

TEMP 변수는 블록이 호출이 될 때 눈에 보이지 않으며 실제 어드레스가 호출되는 블록의 TEMP 변수에 전달되는 것이 아니기 때문에 블록 인터페이스의 요소 가 될 수 없습니다.

파라미터와 TEMP 변수를 선언하려면 파라미터 유형이나 TEMP 변수 유형이 "interface" (아래쪽 그림 참조) 에서 선택되어야 합니다. 그후 오른쪽에 나타나는 테이블에서 이름이 관련 데이터 유형과 주석과 함께 편집될 수 있습니다.

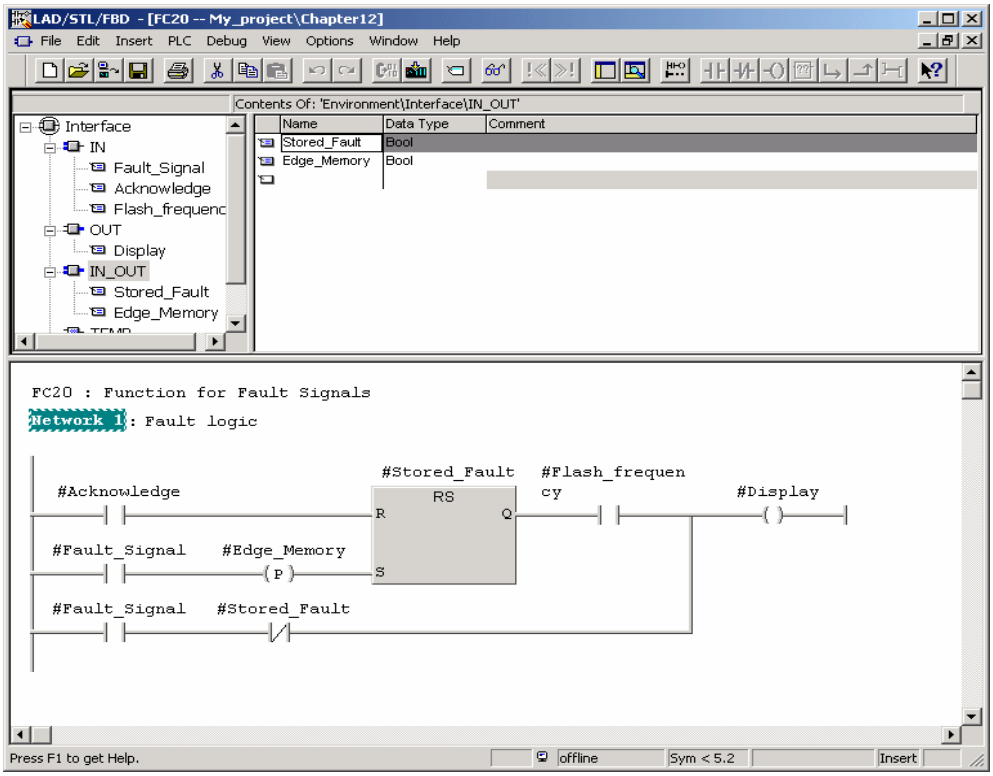
FC 20 의 예

슬라이드의 아래쪽 부분에서 FC 블록 "Fault Signal" (이전 페이지 참조) 의 인터 페이스 와 선언 테이블을 볼 수 있습니다. 형식 파라미터 #Stored_Fault 와 #Edge_Memory 가 FP 명령어에 연결되어 읽기와 쓰기 명령어로 사용이 되기 때 문에 이러한 것을 in/out 파라미터에 선언을 해야 합니다.

주 의!

선언된 블록의 형식 파라미터 (IN, OUT, IN_OUT, TEMP는 예외) 는 "outside" 로의 인터페이스입니다. 즉 그것들은 눈에 보이고 다른 블록과 연관이 있기 때문에 그렇게 부릅니다. 블록 인터페이스가 추후 형식 파라미터를 삭제하거나 추가를 한다면 호출하는 것도 업데이트되어야 합니다. 이 블록을 호출하는 모든 블록도 업데이트되어야 합니다.

파라미터 지정 가능 블록 편집



Date: 22.03.2006
File: PRO1_12E.9

참 조

형식 파라미터 이름이 소문자로 쓰여졌든지 대문자로 쓰여졌든지 상관이 없습니다. # 글자가 이름 앞에 자동적으로 삽입이 됩니다. 이 글자는 파라미터가 이 블록 의 변수 테이블에서 정의된 로컬 변수임을 가리키는데 사용이 됩니다.

프로그램을 LAD/FBD 에서 작성을 할 때 이름이 하나의 라인에 모두 나오지 않게 하는 것이 가능합니다. 이는 프로그램 편집기에서 필요에 맞게 설정하면 됩니다. (Options -> Customize -> LAD/FBD 탭 -> Address Field Width)

심 벌

1. 블록을 편집할 때 심벌 이름을 사용한다면 편집기는 가장 먼저 블록의 변수 선 언 테이블을 통해 그 이름을 찾습니다.

심벌이름이 있다면 심벌과 이름 앞에 있는 # 문자가 로컬 변수로 수용이 됩니 다. 소문자와 대문자는 심벌이 선언 테이블에서 입력된 것에 맞게 수정될 것입니다.

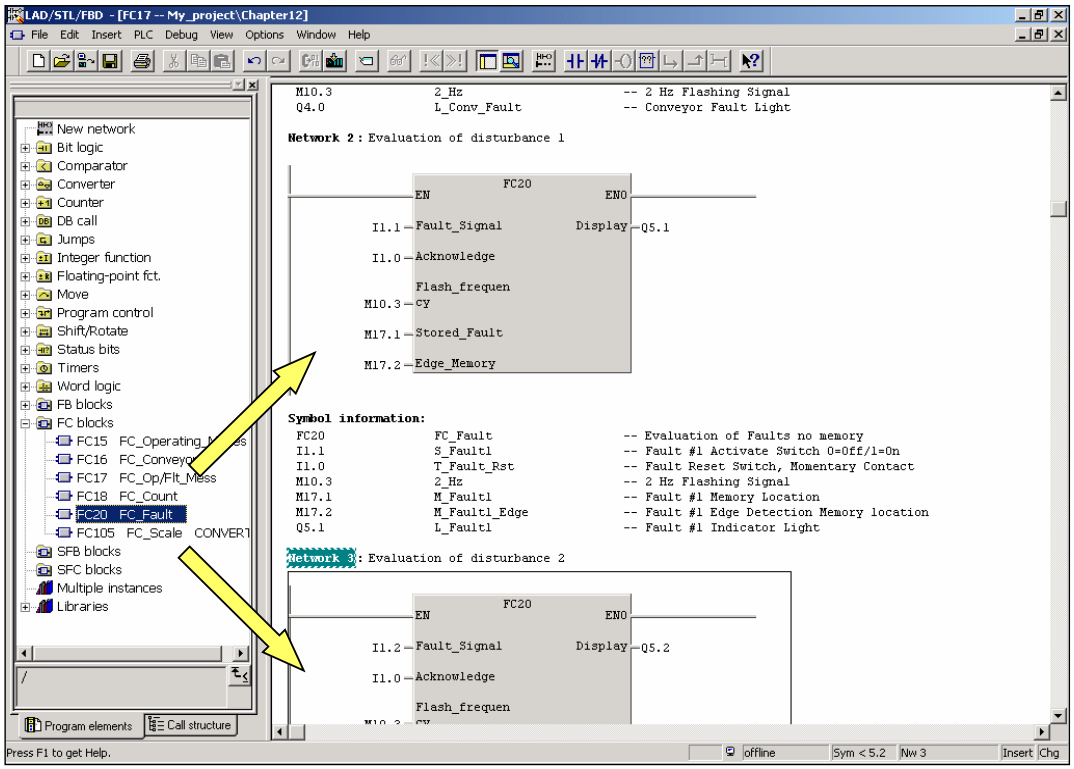
2. 심벌을 로컬 변수에서 찾을 수 없다면 편집기는 글로벌 심벌 테이블에서 심벌 을 찾습니다.

심벌이 거기 있다면 심벌은 인용부호 표시로 놓여지게 되고 프로그램에서 글 로벌 변수로 수용이 됩니다.

3.. 심벌 테이블과 로컬 선언 테이블에서 같은 심벌 이름이 지정이 되면 편집기는 언제나 로컬 변수를 채택합니다.

그러나 글로발 심벌로 작업을 하기 원하면 입력시 심벌 이름을 인용부호 표시 에 넣습니다.

파라미터 지정 가능 블록 호출



블록 호출 프로그래밍

파라미터를 지정하는 블록의 호출은 원하는 블록의 심벌을 복사해서 호출하는 블록의 프로그램 입력 부분에 입력할 수 있습니다. 드래그 앤 드롭을 사용하면 편리합니다. 이 심벌은 프로그램 편집기의 "Program Element Catalog" 부분의 "FC Blocks" 혹은 "FB Blocks" 부분 에서 찾을 수 있습니다. 인용부호 영역은 호출된 블록의 각 형식 파라미터에 자동적으로 나타납니다.

참 조

파라미터를 지정하는 FC 가 호출이 되면 실제 어드레스는 모든 형식 파라미터로 전송이 됩니다.

예외 :
LAD 와 FBD 와 같은 그래픽 프로그램 언어에서 EN과 ENO 파라미터는 편집기 에 의해 자동적으로 추가가 됩니다. 여기에는 형식 파라미터를 지정하지 않으며 이 파라미터들을 통해 조건적으로 블록을 호출할 수 있게 할 수는 있습니다.

파라미터 설정

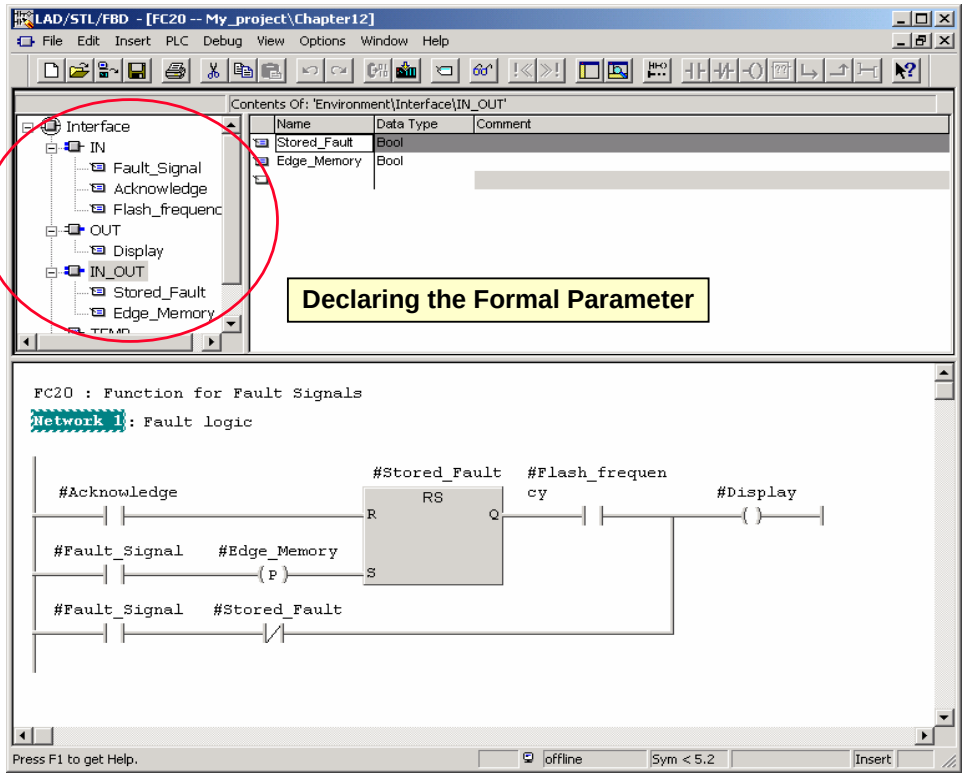
입력하는 데이타 타입이 호출된 블록의 형식 파라미터에 맞으면 글로발 어드레스 혹은 로컬 어드레스가 실제 파라미터(어드레스)로 전달됩니다.

실제 파라미터는 절대 어드레스나 전역 심벌 테이블에서 정의한 심벌 이름 혹은 호출하는 블록에서의 선언 테이블에서 정의한 심벌 이름으로 전달 됩니다.

파라미터 전달

기본적으로 "파라미터 전송 (Passing on of parameters)"이 가능합니다. 즉 호출 하는 블록의 형식 파라미터가 호출된 블록의 실제 파라미터로 전달이 되는 것입니다. 복합 데이터 타입의 파리미터인 경우 한계가 있긴 하지만 가능합니다. 이 내용은 고급 과정에서 다루어질 것입니다.

연습 문제 1 : 파라미터 지정 가능 Function 편집



Date: 22.03.2006
File: PRO1_12E.11

오류 해석 Function

오류가 발생을 하면 (시뮬레이터의 스위치 동작) 시뮬레이터의 출력 LED가 2 Hz 로 깜박거리기 시작합니다. 사용자는 시뮬레이터의 푸시 버튼을 사용하여 오류 를 인식할 수 있습니다.

오류가 인식이 되면 두가지 중 하나의 경우일 것입니다. 오류가 더이상 존재하지 않다면(시뮬레이터의 스위가 꺼짐) 출력 LED 도 꺼지게 될 것입니다. 오류가 계 속 존재한다면 (시뮬레이터의 스위치가 계속 켜짐) 출력 LED는 켜진 상태에서 멈 출 것입니다. 오류가 더 이상 존재하지 않으면 LED 는 꺼지게 됩니다.

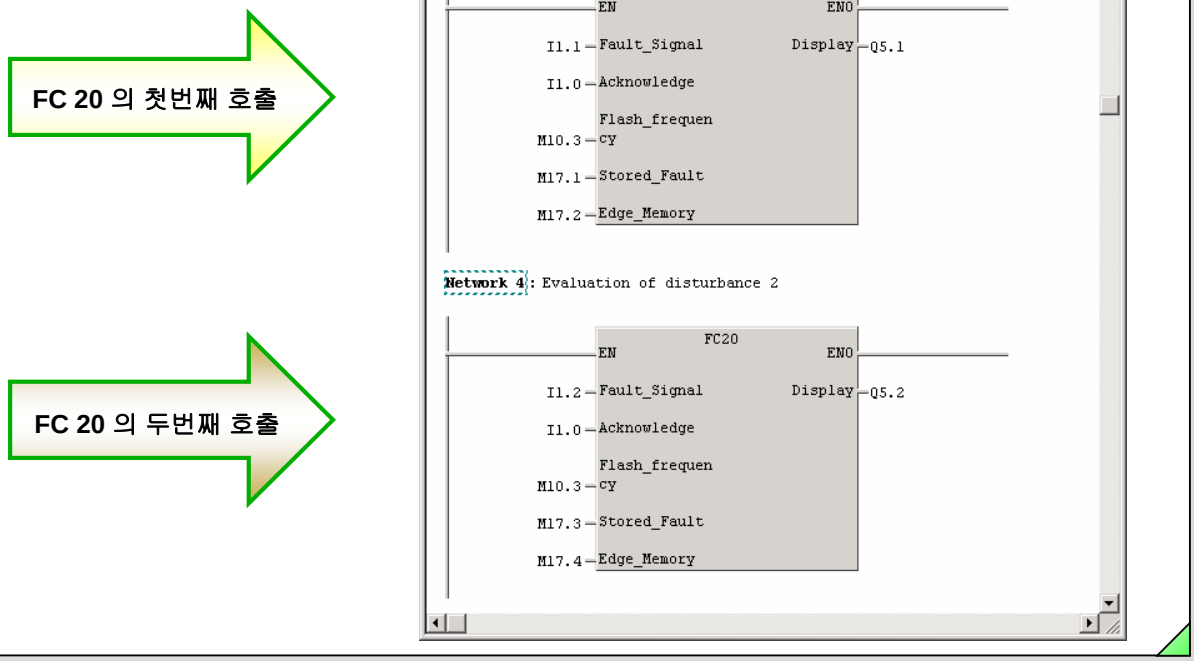
과 제

오류 해석 프로그램을 파라미터 지정 FC 20 "Evaluate_Fault" 안에 생성하십시오.

절 차

1. FC 20 블록을 S7 블록 폴더에 삽입하십시오.
2. 위 슬라이드에 보여진 것처럼 형식 파라미터를 선언하십시오.
3. 위 슬라이드에 보여진 것 처럼 프로그램을 생성하십시오.
4. 블록을 저장하고 CPU로 다운로드하십시오.

연습 문제 2 : 파라미터 지정 가능 Function 호출



Date: 22.03.2006
File: PRO1_12E.12

과 제

두개의 프로세스 에러 (시뮬레이터의 두개의 스위치)가 해석이 되고 시뮬레이터 의 LED 를 통해 디스플레이됩니다. FC 20 을 두번 호출하는 프로그램을 작성하 고 위 슬라이드에 보여진 것처럼 실제 파라미터를 입력합니다.

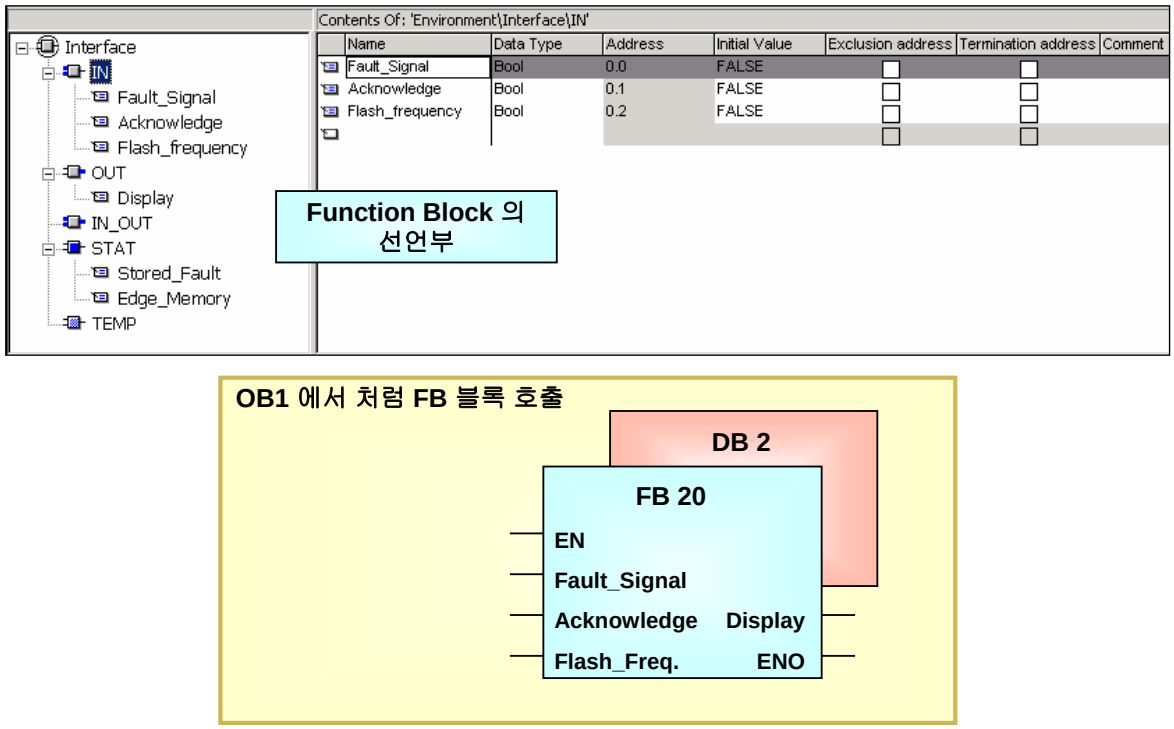
절 차

1. FC 20 을 프로그램하십시오 - FC 17 블록에 두개의 새로운 네트워크(렁)에서 호출하십시오.
2. FC 17 을 저장하고 CPU 로 다운로드하십시오.

참 조

HWConfig 화면에서 클럭 메모리 (Clock Memery) 로 MB 10을 설정하였습니다. 메모리 리셋을 수행한다면 M10.3 비트 메모리가 깜박거리게 하기 위해서는 HWConfig 에서 생성된 시스템 데이터(system data) 를 다시 다운로드시켜야 합니다.

Function Blocks (FB)



SIMATIC S7

Date: 22.03.2006
File: PRO1_12E.13

SITRAIN Training for Automation and Drives

FB의 특징

FC 와는 다르게 FB 에는 Recall 메모리가 있습니다. 이는 로컬 데이타 블록이 Function Block 에 할당이 되는 것을 의미합니다. 이 데이타 블록을 인스턴스 데 이터 블록(Instance DB)이라고 합니다. FB 를 호출할 때 인스턴스 DB 의 번호도 지정해야 합니다. 인스턴스 DB는 Static 변수를 저장하기 위해 사용됩니다. 이러 한 로컬 변수는 FB의 선언부에서 선언을 하여 사용될 수 있습니다. 프로그램이 종료가 되어 블록을 빠져나와도 데이터는 남아 있습니다 .

Exclusion 어드레스와 Termination 어드레스

이 옵션을 사용하면 속성을 프로세스 진단과 연관되어 있는 FB 파라미터와 Static 변수에 할당할 수 있습니다.

파라미터

Function Block 이 호출되면 실제 파라미터의 값들은 인스턴스 DB에 저장이 됩니다. 실제 파라미터가 블록 호출에서 형식 파라미터에 할당이 되면 이 파 라미터에 대해 인스턴스 DB에 저장된 최종 값이 프로그램 실행시 사용이 됩니다. 사용자는 모든 FB 호출시 다른 실제 파라미터를 지정할 수 있습니다.

Function Block 을 빠져 나와도 인스턴스 DB 에는 데이터가 남아 있습니다.

Static 변수

Static 로컬 변수는 Function Block 에서 엑세스할 수 없는 특정 데이터를 저장합니다. 다시 말해서 변수는 형식 파라미터로서 다른 블록에 입력되거나 출력되지 못한다는 뜻입니다.

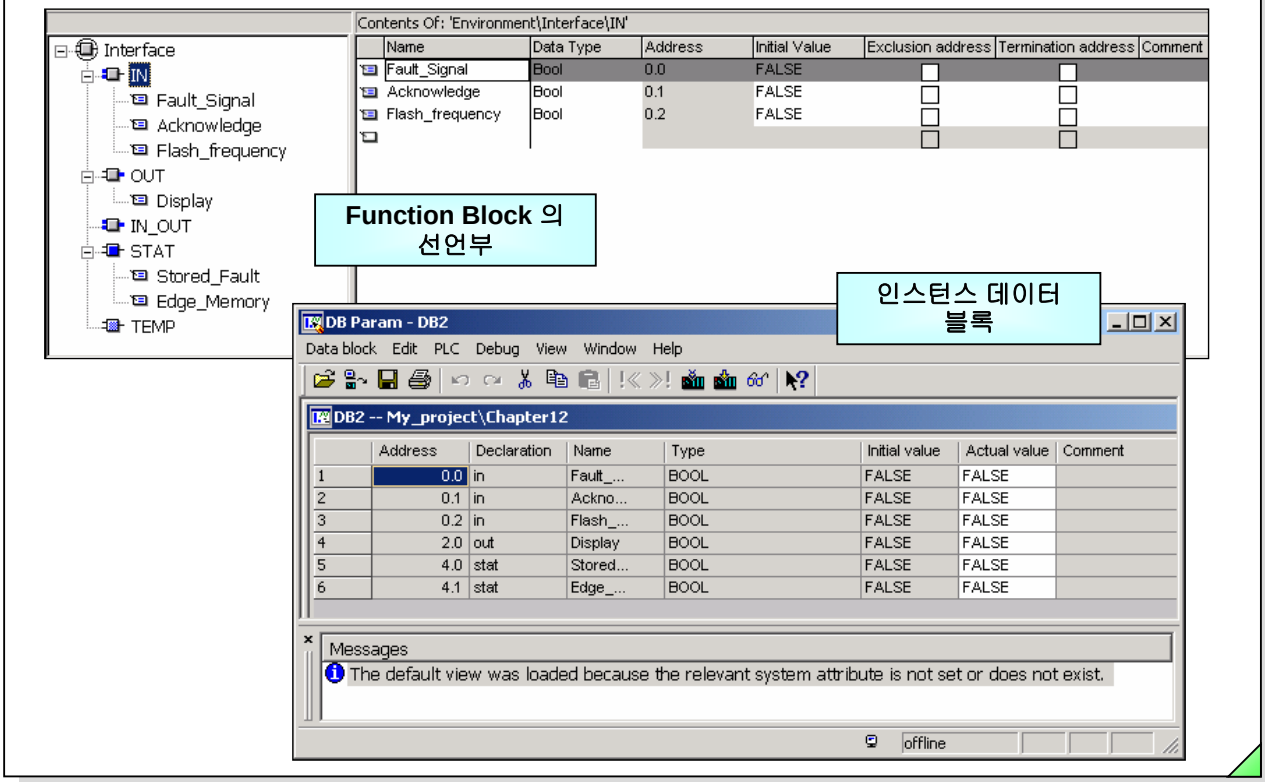
FB 장점

- FC에서 프로그램을 할 때는 사용하지 않은 비트 메모리 어드레스 영역과 데이 터 영역을 찾아야만 합니다. FB에서의 Static 변수는 STEP 7 소프트웨어에서 관 리를 해줍니다.

- Static 변수를 사용하면 비트 메모리 영역이나 데이터 영역을 두번 지정하는 일 을 피할 수 있습니다.

- 형식 파라미터 FC 20의 "Store Fault" 와 "Edge_Memory" 대신에 Static 변수 "Stored_Fault" 와 "Edge_Memory" 를 사용할 수 있습니다. 이는 두개의 형식 파 라미터를 삭제시킬 수 있기 때문에 블록 호출이 더 간단해 집니다.

오류 디스플레이를 위한 **Function Block**



Date: 22.03.2006
File: PRO1_12E.14

오류 디스플레이

이전의 연습문제에서 오류 메시지를 디스플레이하기 위한 파라미터 지정 블록 FC 20 을 생성하였습니다.

비트 메모리 대신에 오류 신호와 RLD 에지 검출을 저장하기 위해 FC 20 을 사용 하였고 FB 에서는 이것이 Static 변수라고 합니다. 그것은 FB 와 연관된 인스턴스 DB 에 저장이 되었습니다.

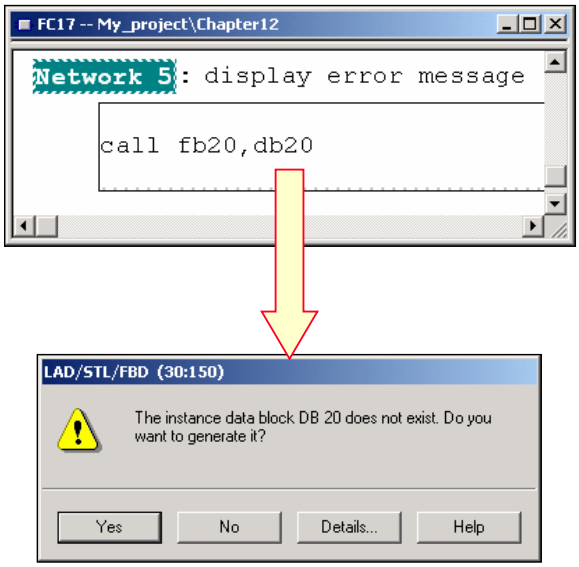
인스턴스 DB 구조

DB가 생성이 되고 FB 에 연결이 될 때 STEP 7 은 Function Block 의 로컬 선언 테이블에서 지정된 구조를 사용하여 데이터 블록의 데이터 구조를 생성합니다. DB 를 저장한 후에 DB가 생성이 되고 인스턴스 DB 로 사용할 수 있습니다.

인스턴스 데이터 블록 생성

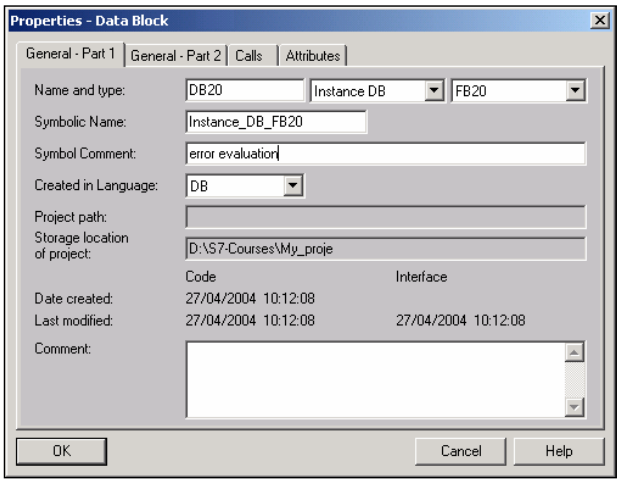
1. FB 호출시 인스턴스 데이터 블록 생성

In the LAD/STL/FBD Editor



2. 별도로 인스턴스 데이터 블록 생성

In the SIMATIC Manager



인스턴스 DB 생성

새로운 인스턴스 DB를 생성하는 데는 두가지 방법이 있습니다.

- FB 를 호출할 때 어느 인스턴스 DB 를 FB에 사용할 것인지 지정해야 합니다. 다음과 같은 메시지가 나타납니다.

"The instance data block DBx does not exist. Do you want to generate it ?„

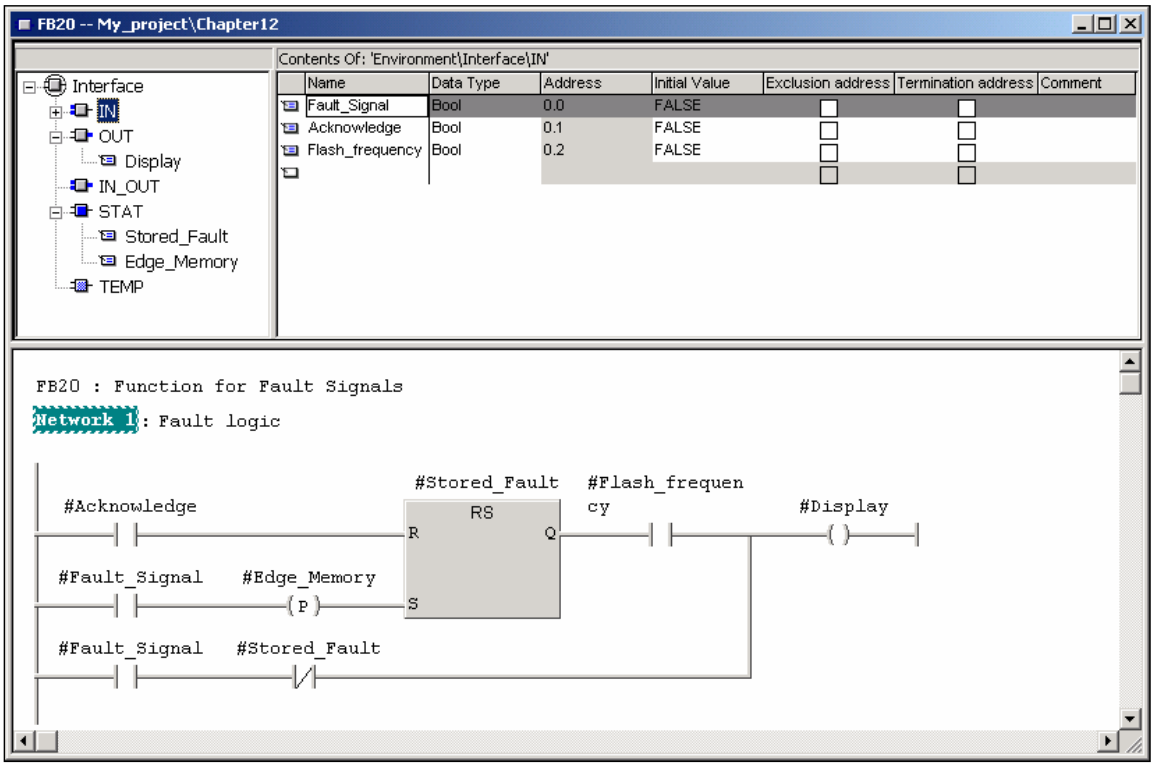
- 새로운 DB를 생성할때 "Data block referencing a function block" 옵션을 선택 합니다.

참 조

하나의 인스턴스 DB가 하나의 FB 에 연결됩니다. 그러나 FB 는 호출될 때 마다 다른 여러 인스턴스 DB 에 연결될 수 있습니다.

FB 를 파라미터나 Static 변수를 추가하여 변경할 때는 인스턴스 DB를 다시 생성 해 주어야만 합니다.

연습 문제 3 : Function Block 편집



Date: 22.03.2006
File: PRO1_12E.16

과 제

추가 오류(시뮬레이터 스위치)가 해석되어야 합니다. 이것을 하기에 가장 쉬운 방법은 FC 20 호출을 또 한번 하는 것입니다.

그러나 FB 솔루션의 장점을 살리기 위해 파라미터 지정 FB 20 을 프로그램하여 세번째 오류를 해석합니다.

Static 변수는 Edge Memory 와 Stored Fault 를 저장하는 데 사용됩니다. 이러한 Static 변수는 FB 의 인스턴스 DB에 저장될 것입니다.

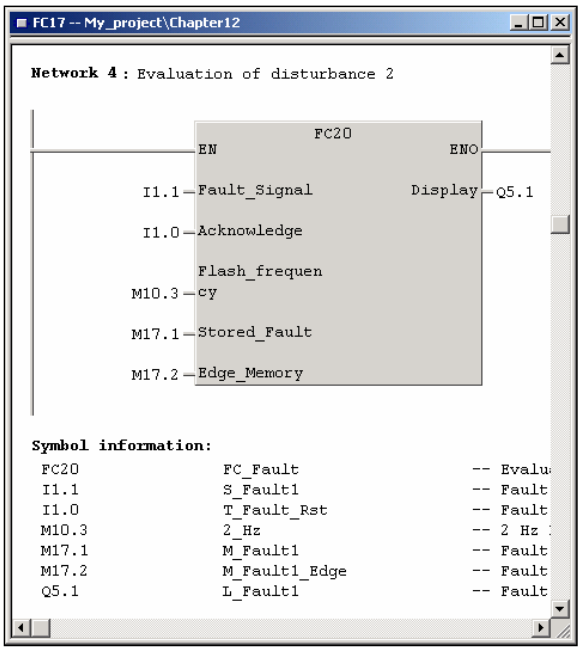
위 슬라이드에서 FB 20의 선언 테이블 입력과 출력 파라미터와 프로그램 시작을 볼 수 있습니다.

절 차

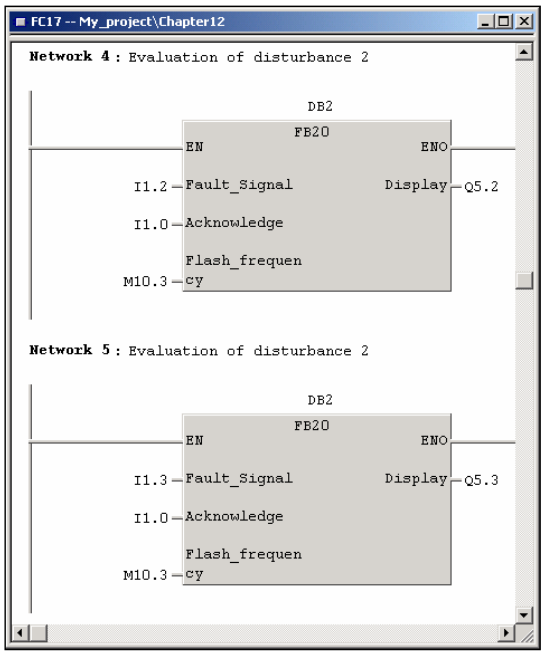
1. FB 20 블록을 S7 블록 폴더에 삽입하십시오.
2. 위 슬라이드에 나온 것 처럼 형식 파라미터와 Static 변수를 선언하십시오.
3. FB 20 의 프로그램을 작성하십시오. 이미 생성된 FC 20 에서 필요한 네트워 크(렁)을 복사해서 FB 20 에 붙여넣기를 하십시오.
4. 새로운 블록을 저장하고 CPU 로 다운로드하십시오.

연습 문제 4 : Function Block 호출 및 테스트

현재 상태



현습 문제 실행 후 상태



SIMATIC S7

Date: 22.03.2006
File: PRO1_12E.17

SITRAIN Training for Automation and Drives

과 제

오류 #2의 해석 (지금까지 FC 20 에서 프로그램을 하였음) 과 새로운 오류 #3 의 해석이 새롭게 생성된 FB 20 으로 구현이 될 것입니다.

파라미터 지정 블록 FB 20 이 FC 17 에서 두번 호출이 되어야 합니다.

FB 20 이 호출될 때마다 다른 인스턴스 DB 가 할당되어야 합니다.

절 차

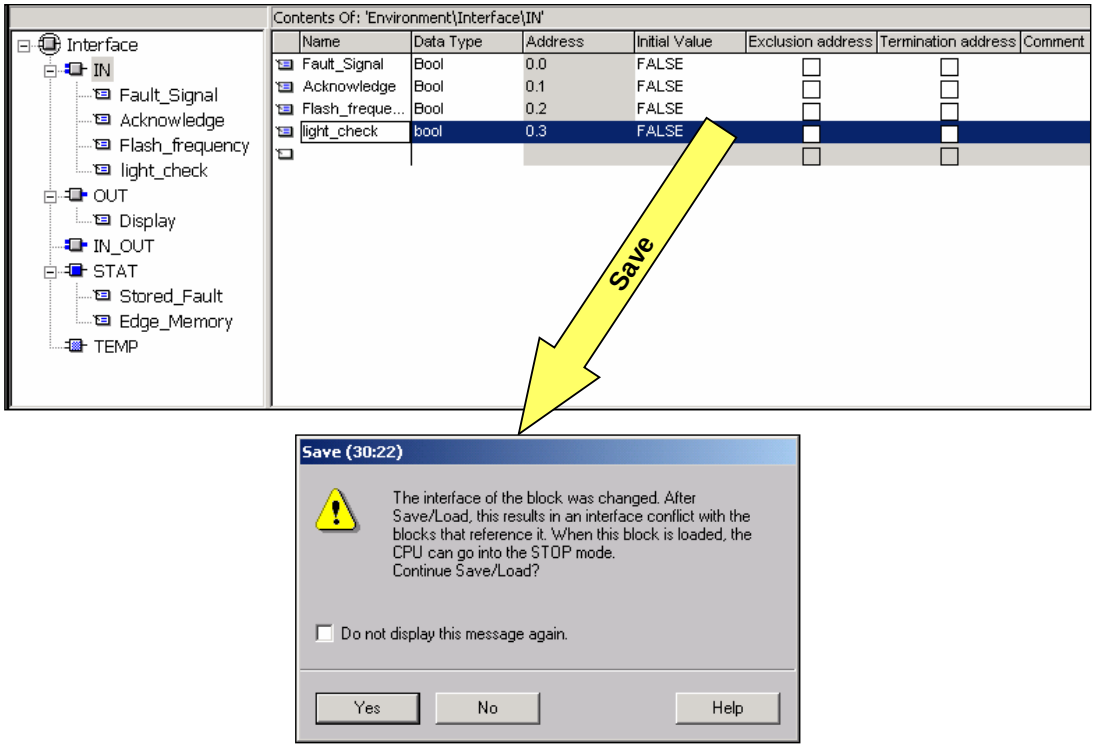
1. FC 17 에서 두번째 호출된 FC 20 을 삭제하십시오. 오류 #2가 FB 20 에서 구현되기 때문입니다.
2. 슬라이드와 같이 FC 17 에서 두개의 새로운 렁에 FB 20 이 두번 호출됩니다. 편집기에서 인스턴스 DB2 와 DB3 을 생성하십시오.
3. 변경된 FC 17을 오프라인에서 저장하십시오.
4. 생성된 DB2 와 DB3 을 SIMATIC Manager 에서 CPU 로 다운로드하십시오. 이 후 변경된 FC 17 을 다운로드하십시오.
5. 사용자 프로그램을 테스트하십시오.

과 제 2

오류 해석에 대한 Function Block 을 성공적으로 테스트한 후에 다음 프로세스 신 호를 시뮬레이터 스위치 대신에 오류 신호로 사용할 수 있습니다.

- 비트 메모리 신호 M 17.0 "Conveyor belt fault condition"
- 근접 스위치 1번 베이와 2번 베이를 자동 모드로 동시에 동작을 시키십시오. 두 번째 작업은 다음 연습 문제로 이어지지 않습니다.

블록 파라미터 삽입과 삭제



Date: 22.03.2006
File: PRO1_12E.18

문 제

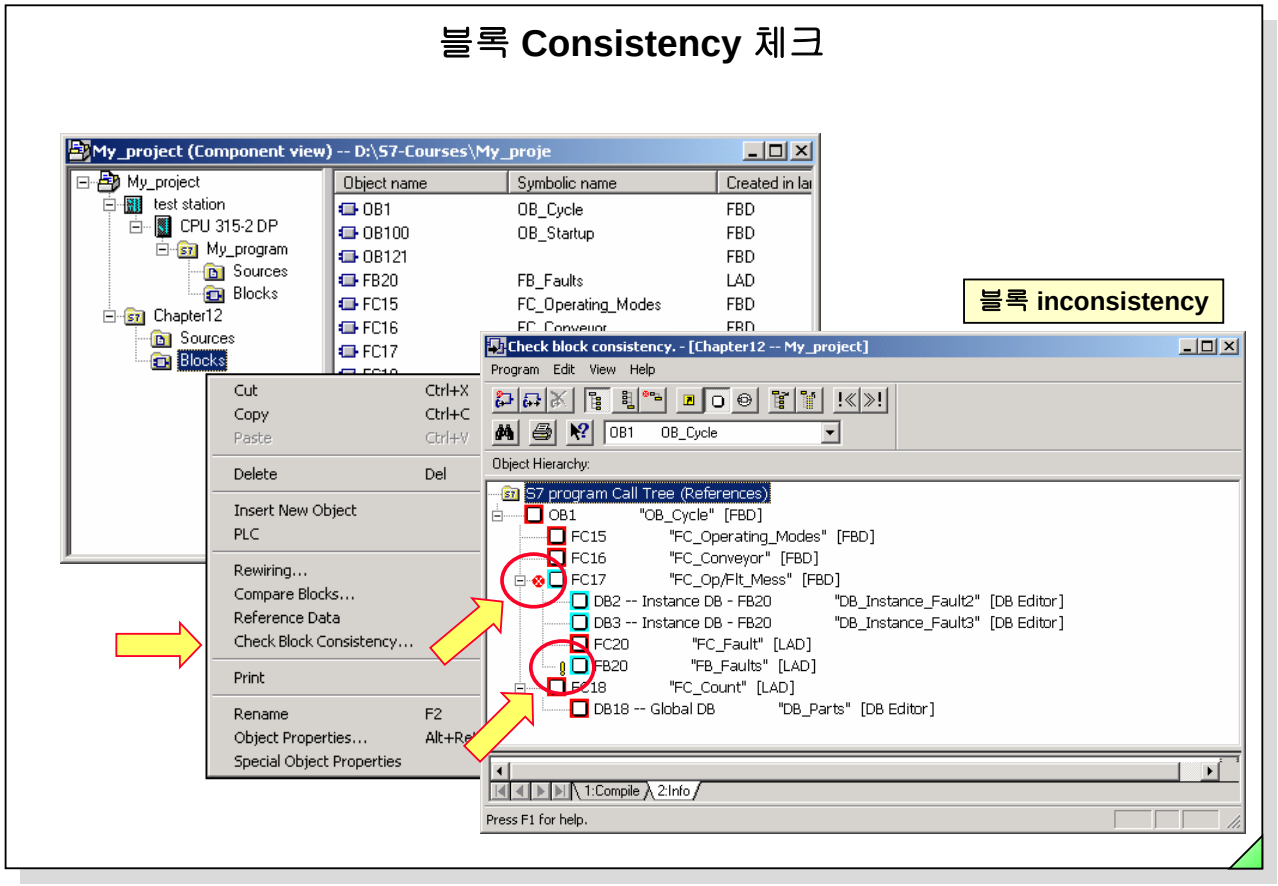
프로그램 생성중에 혹은 프로그램 실행 중에 각 블록의 선언부나 프로그램을 추 후에 조정하거나 추가하려고 할 때 "Time Stamp Conflict" 가 발생합니다. Time Stamp Conflicts 는 호출 블록과 호출된 블록이나 관련 블록 사이의 내용이 서로 같지않은 문제를 차례로 발생시킬 수 있기 때문에 많은 수정 작업이 필요해집니다.

블록 파라미터가 추후에 프로그램에서 이미 호출된 블록에서 추가되거나 삭제될 때 다른 블록에서의 블록 호출도 업데이트되어야만 합니다. 이것이 무시되면 CPU는 STOP 으로 전환됩니다. 혹은 추가적으로 선언된 형식 파라미터가 호출 될 때 실제 파라미터에 적용되지 않기 때문에 블록의 기능은 더이상 보증할 수 없 게 됩니다.

예제에서 추가적인 입력 파라미터 "Check_Light" 가 삽입이 되었고 나중에 모든 블록 호출시 실제 파라미터가 할당이 됩니다.

형식 파라미터가 추가되거나 삭제된 블록을 저장할 때 일어날 수 있는 경고 메시지가 나타납니다.

블록 Consistency 체크



Date: 22.03.2006
File: PRO1_12E.19

사용 영역

Check block consistency -> Compile 기능은 모든 Time stamp conflict 를 제거하 고 블록 불일치를 체크합니다.

인터페이스 충돌은 파라미터 지정 블록이 생성이 되고 다른 블록에서 호출이 된 이후 추가적으로 수정이 될 때 발생합니다. 블록 불일치는 또한 어드레스가 심벌 로 엑세스되고 그후 글로발 심벌 테이블이나 데이타 블록에서 심벌 <-> 절대 어 드레스가 변경되었을 때 발생합니다.

슬라이드에서는 일부 블록들에 불일치가 생기고 (예 : 인터페이스 충돌) 이것을 가리키는 심벌이 나타나 있습니다. (온라인 도움말 참조) 이러한 블록들은 오픈 이 될 수 있고 오른쪽 마우스 버튼을 사용하여 교정이 될 수 있습니다. (다음 페이지 참조)

트리 구조 보기

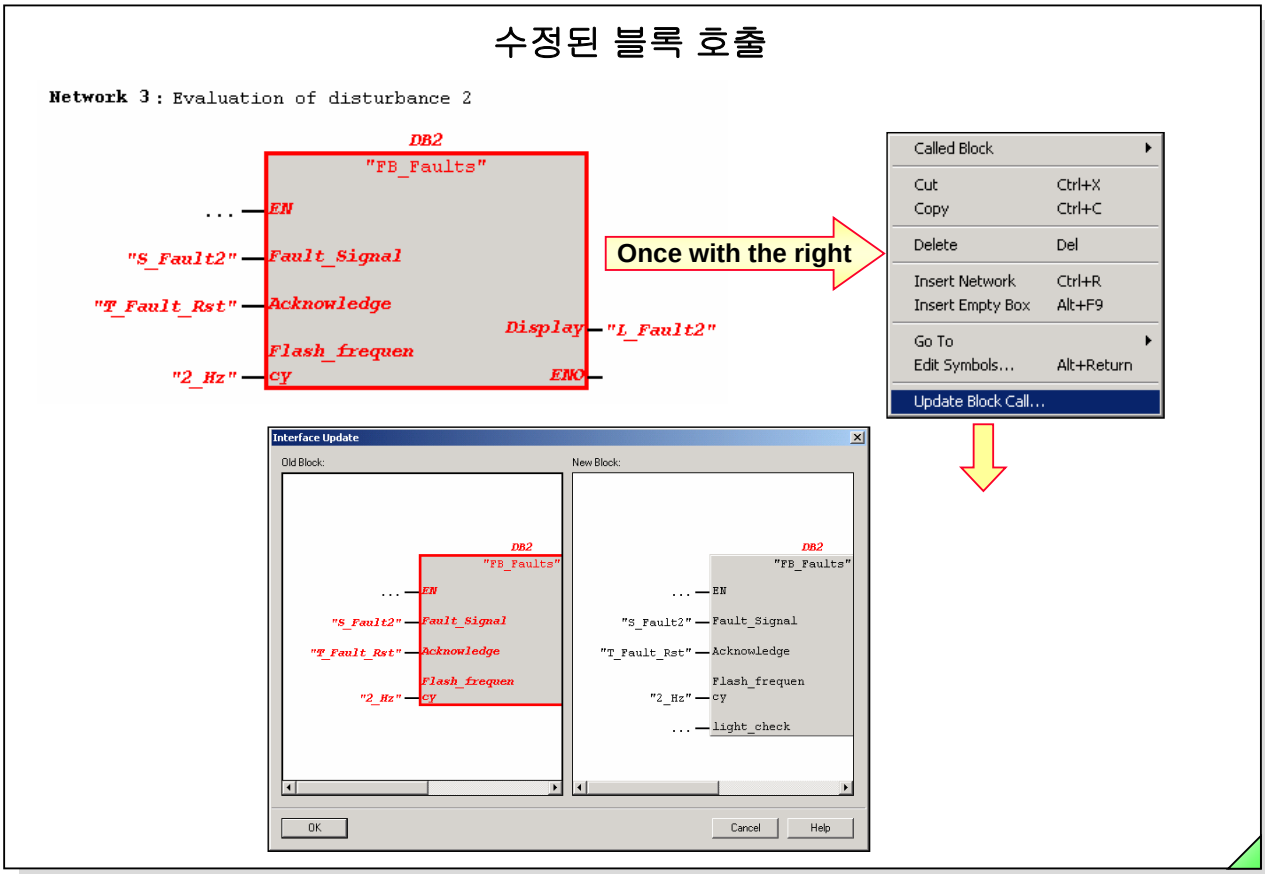
트리 구조 보기는 선택된 블록 폴더에 있는 블록들의 연관성을 보여줍니다. 트리 구조 보기는 View -> Reference Treee / Dependency Tree 를 통해 Depency Tree 나 혹은 Referece Treee 로 표시될 수 있습니다.

Reference Tree (레퍼런스 트리)

레퍼런스 트리는 왼쪽에서 오른쪽으로 모든 블록과 호출 관계를 레벨 별로 보여 줍니다. Reference Data Program Structure 처럼 호출 관계가 왼쪽에서 오른쪽 으로 처음 호출 레벨 1번 부터 보여줍니다. 그러므로 레퍼런스 트리는 각 프로그 램 실행 레벨의 호출에 대해 전체 보기를 제공해 줍니다.

Dependency Tree (디펜던시 트리)

디펜던시 트리는 왼쪽에서 오른쪽으로 모든 블록과 호출관계를 레벨 별로 보여 줍니다. 이경우 보여지는 호출 관계는 호출 레벨 1번부터는 아니고 개별적인 블 록에서부터 보여줍니다. 따라서 블록 폴더에 있는 모든 블록은 왼쪽 레벨에 모두 리스트됩니다. 다음 레벨 (오른쪽으로)은 호출되는 블록들 혹은 의존성을 보여줍니다. Reference Data -Cross reference list 와 같이 디펜던시 트리는 어느 블록을 호출하였는가에 대한 정보를 제공합니다 .

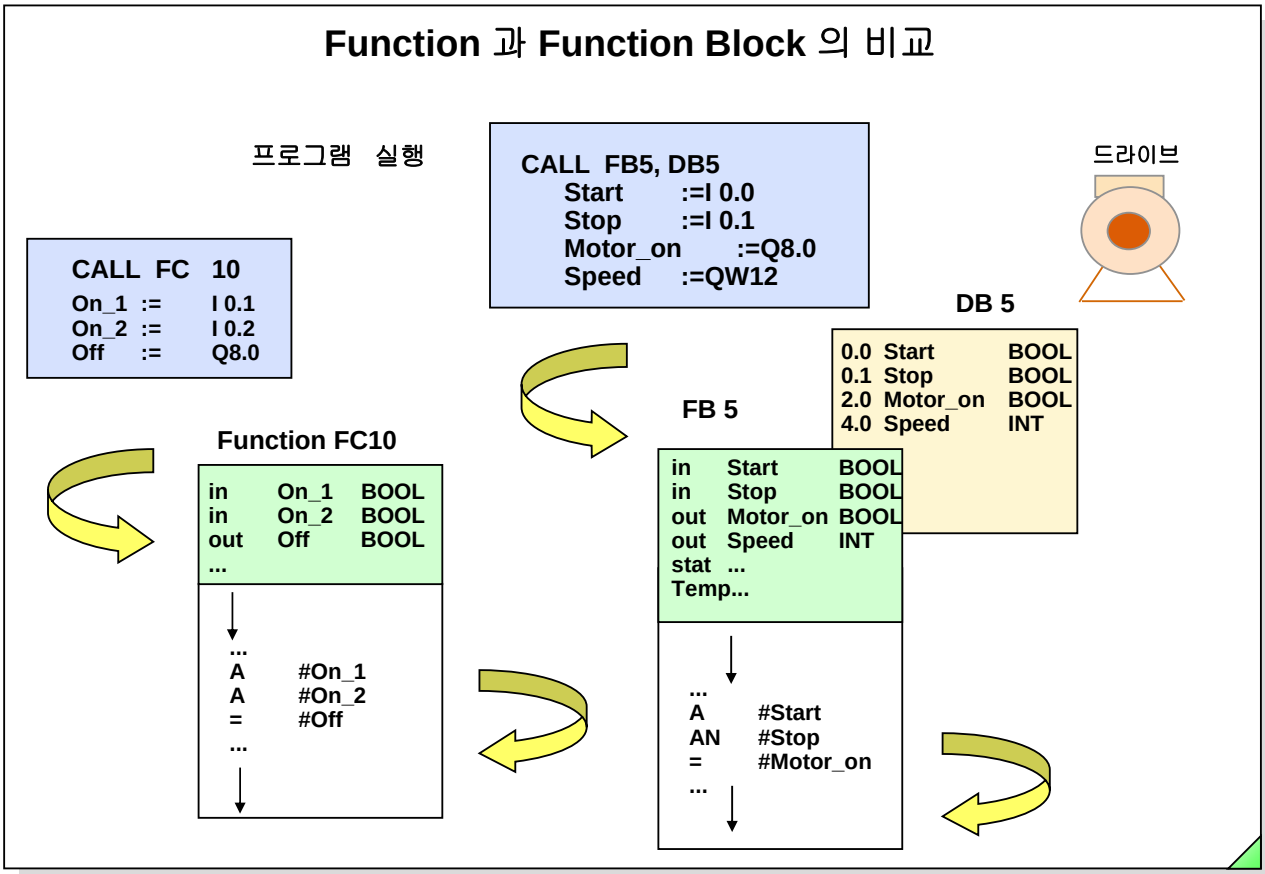


Date: 22.03.2006
File: PRO1_12E.20

호출 업데이트

일치하지 않는 블록 호출은 호출한 블록에서 적색으로 표시됩니다. 일치하지 않는 호출을 오른쪽 마우스로 클릭하고 나타나는 다이알로그 박스에서 Update Block Call 을 선택하십시오. 이전 블록(오류가 있는 블록)과 새로운 블록(슬라이드에서 "Check_Lights" 의 파라미터가 추가됨) 호출을 보여주는 윈도 우가 나타납니다. "OK" 로 확인을 한 후 빠져있는 "Check_Lights" 형식 파라미터에 실제 파라미터를 입력합니다.

인스턴스 DB가 Function Block 에 대해 다시 생성이 됩니다.



Functions

Fuctions 은 메모리가 없는 파라미터 지정 블록입니다. STEP 7 에서 Function 은 필요한 대로 입력, 출력,입/출력 파라미터를 가질 수 있습니다.

Functions는 자체적으로 결과를 저장하는 데이터 영역을 가지고 있지 않기 때문 에 메모리가 없습니다. Function 을 실행하면서 발생하는 중간 결과는 로컬 데이 터 스택의 Temporary 변수에 저장이 되면 됩니다.

Function 은 기존의 방법대로 프로세서의 명령어 군을 확장합니다.

사용 영역

Functions 는 블록을 호출하면서 Function 값이 다시 돌아올 때 주로 사용합니다. (수학적 Function 이나 이진 로직이 있는 개별 제어와 같은 Function)

Function Blocks

Function Blocks (FB) 은 사용자 프로그램 블록이며 IEC 61131-3 을 따르고 있는 메모리가 있는 로직 블록입니다. OB, FB,FC 에서 호출될 수 있습니다. Function Blocks 는 필요한대로 Static 변수, Temporary 변수를 비롯 입력, 출력, 입출력 변수를 가질 수 있습니다.

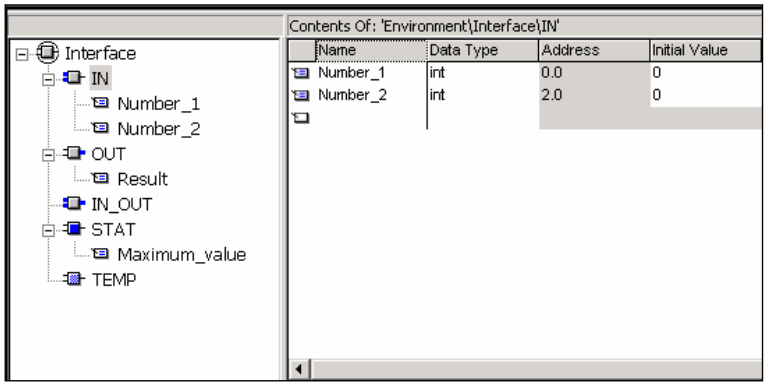
FC 와는 다르게 FB는 메모리가 있습니다. 즉, 하나의 FB는 자체 데이터 영역을 가지고 있습니다. 그래서 프로세스 상태를 하나의 호출에서 다음 호출까지 기억할 수 있습니다. 가장 단순한 형태로 이 자체 데이터 영역이 DB 안에 있게 되며 이 DB를 인스턴스 DB 라고 부릅니다.

"Memory“

Function Blocks 의 선언 테이블에서 사용자는 Static 변수를 선언하고 처음 호출에서 다음 호출까지 이 정보를 기억합니다.

이것이 Function과 주로 다른 점입니다.

연습 문제 5 : 변수 타입 인식



Statement	VARIABLE TYPE					
	Global	Local	Absolute	Symbolic	Static	Parameter
L #Number_1						
L #Number_2						
T #Maximum_value						
L "Number_1"						
T MW 40						
T #Number_2						

과 제

슬라이드의 명령문 영역에서 여러 변수들을 볼 수 있습니다. 아래 테이블에서는 해당 속성을 변수에 할당시킵니다.

절 차

테이블에서 관련 데이터 유형을 X 자로 표시하십시오.

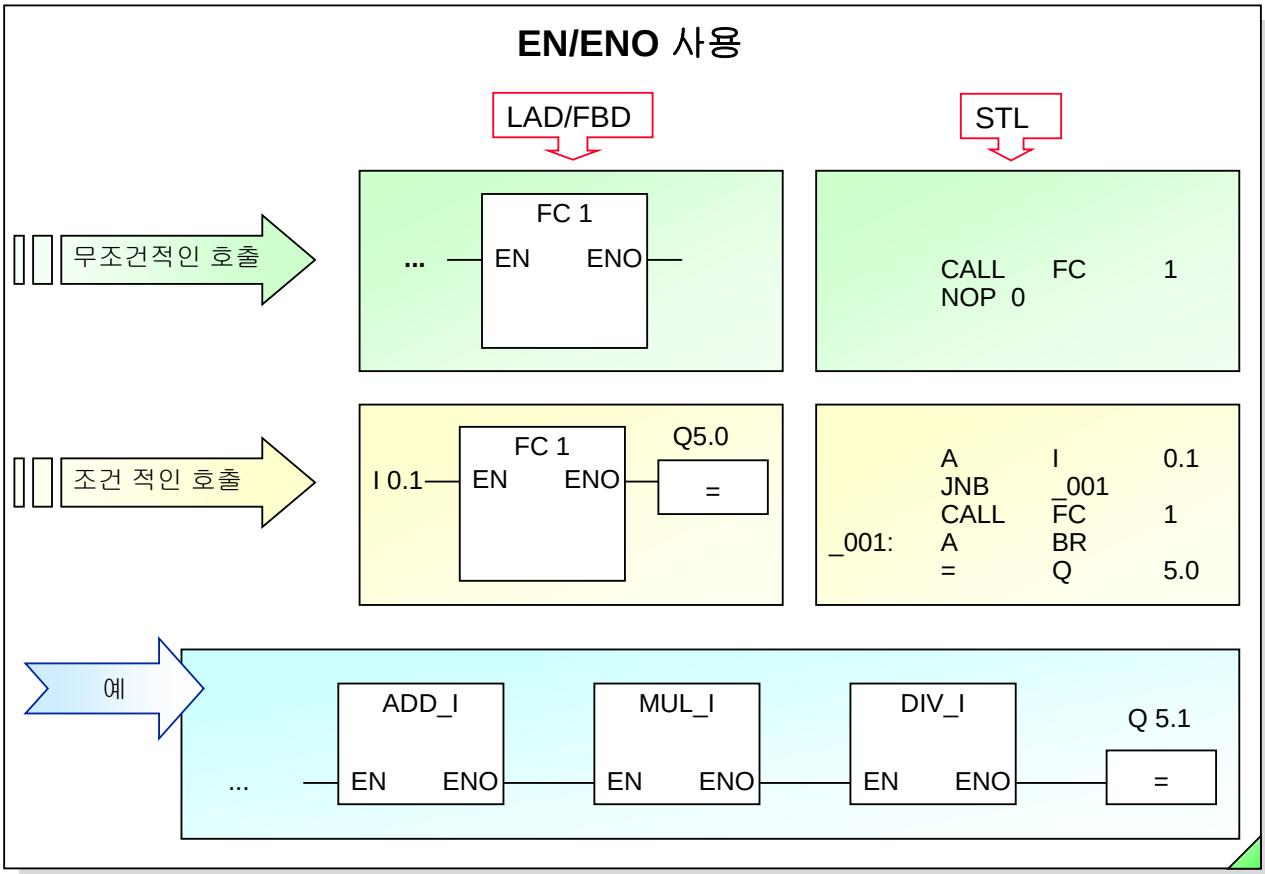
다음 질문에 답하십시오.

명령문 T #Number_2 에서 무엇이 틀렸습니까?

.....

.....

.....



표준 FC

표준 FC 를 실행할때 다음과 같은 규칙이 있습니다.

- EN =0 이면 블록은 실행을 하지 않으며 ENO = 0 입니다.
- EN =1 이면 블록은 실행을 하며 에러없이 실행이 되면 ENO 도 1 입니다.

블록이 실행되는 동안 에러가 발생하면 ENO 는 0가 됩니다.

사용자 FC

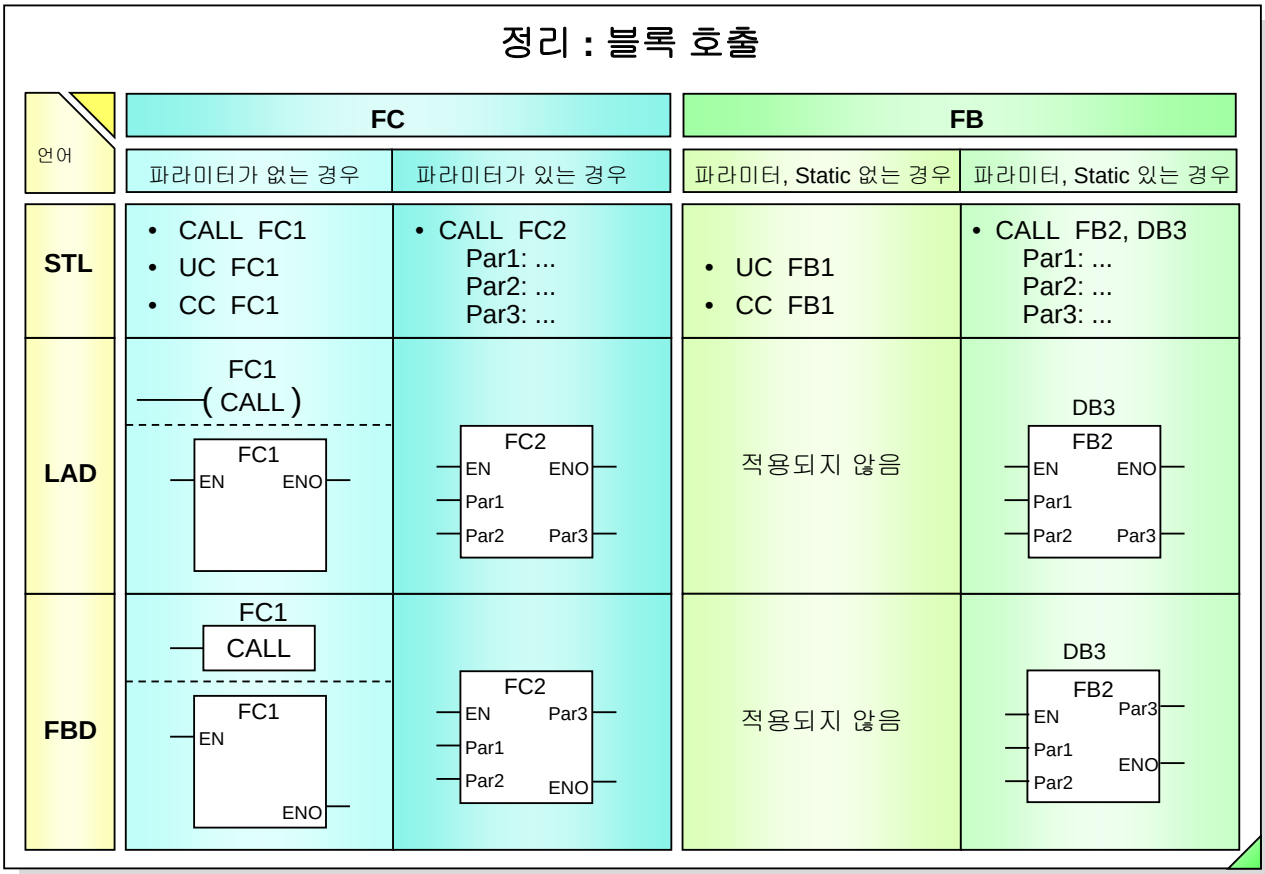
사용자 블록이 LAD/FBD 로 작성이 되었는지 STL 로 작성이 되었는지는 문제가 되지 않습니다. LAD/FBD 에서 호출이 될 때는 EN과 ENO 가 추가가 됩니다.

EN/ENO는 STL 프로그램에서는 존재하지 않습니다.그러나 에뮬레이션을 할 수는 있습니다.

프로그램 언어와 상관없이 에러 처리에 프로그램은 작성되어야 합니다.

내부 연결

LAD/FBD 에서 여러 개의 박스가 그룹이 지어질 수 있으며 EN/ENO 를 통해 논리 적으로 링크될 수 있습니다.



Date: 22.03.2006
File: PRO1_12E.24

CALL

CALL 명령은 프로그램을 호출할 때 사용합니다. 그래픽 프로그래밍 언어인 LAD 와 FBD 에서 블록 호출은 CALL 박스의 EN 입력을 사용하여 RLO 의 조건에 따 라 실행되도록 이루어질 수 있습니다. STL 프로그래밍 언어에서 블록 호출은 RLO 와 상관없이 이루어집니다.

FB 혹은 SFB 를 CALL 명령어로 호출하면 연관되는 인스턴스 DB를 지정해야만 합니다. 이 블록을 절대 이름이나 심벌 이름으로 사용할 수 있습니다.

예: "CALL FB2, DB2" 혹은 "CALL valve, level„

UC

"UC" 명령은 FC,FB 블록을 무조건적으로 호출하는 것입니다. UC 는 호출된 FC 나 FB가 파라미터 지정 블록이 아닌 경우에만 사용됩니다.

또한 UC 로 호출되는 FB에서는 Static 변수가 선언될 수 없습니다.

CC

"CC" 명령은 FC 혹은 FB 블록을 조건적으로 호출할 때 사용됩니다. CC는 호출된 FC 혹은 FB 가 파라미터 지정 블록이 아닌 경우에만 호출됩니다. 또한 CC 로 호출되는 FB 에서는 Static 변수가 선언될 수 없습니다.

절 차

UC 와 CC 명령은 래더와 FBD 로 변환되지 않습니다. 또한 프로그램 편집기에서 파라미터 지정 블록에서 UC 나 CC 호출을 허용합니다. 그러나 이 프로그램이 다운로드가 되면 "AREA ERROR when Reading" 이라는 시스템 오류가 발생합니다.

파라미터

블록의 선언 테이블에서 선언된 형식 파라미터는 블록의 인터페이스입니다. 파라미터 지정 FC 가 호출되면 실제 파라미터는 반드시 모든 형식 파라미터에 할당되어야 합니다. FB가 호출이 될때는 이러한 파라미터는 반드시 입력이 되어야 하는 것은 아닙니다.

Static 변수와 Temporary 변수는 파라미터가 아닙니다. 그리고 블록 인터페이스의 일부분이 아닙니다. 그 결과 블록 호출에서 Static 이나 Temporary 변수로 입력되어야 하는 파라미터는 없습니다.