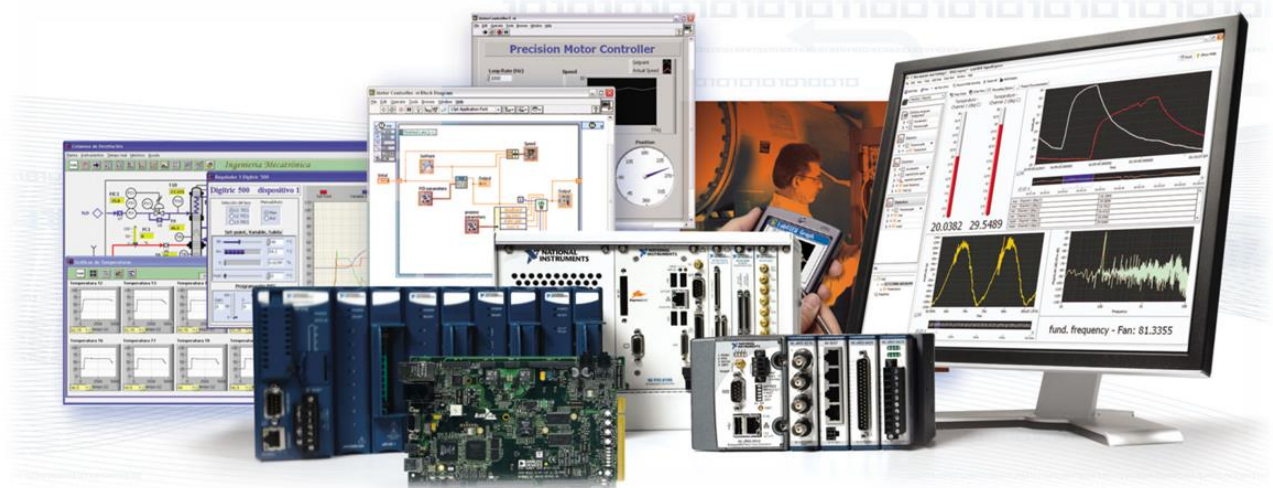


8. 디자인 패턴





디자인패턴의 개요

- ❖ 디자인 패턴이란 **LabVIEW**에서 사용하는 정형화된 프로그램의 전체 구조를 의미
- ❖ **VI** 에 **While** 루프가 하나 이하가 있을 때 단일 루프 디자인 패턴이라 하고 **2**개 이상이면 멀티 루프 디자인 패턴이라 한다.
- ❖ 단일 루프 디자인 패턴에는 단순, 일반, 상태 머신, 사용자 인터페이스 이벤트 핸들러, 메시지 핸들러등이 있다.
- ❖ 멀티 루프 디자인 패턴에는 병렬, 마스터 / 슬레이브, 생산자/소비자(데이터), 생산자/소비자(이벤트) 등이 있다



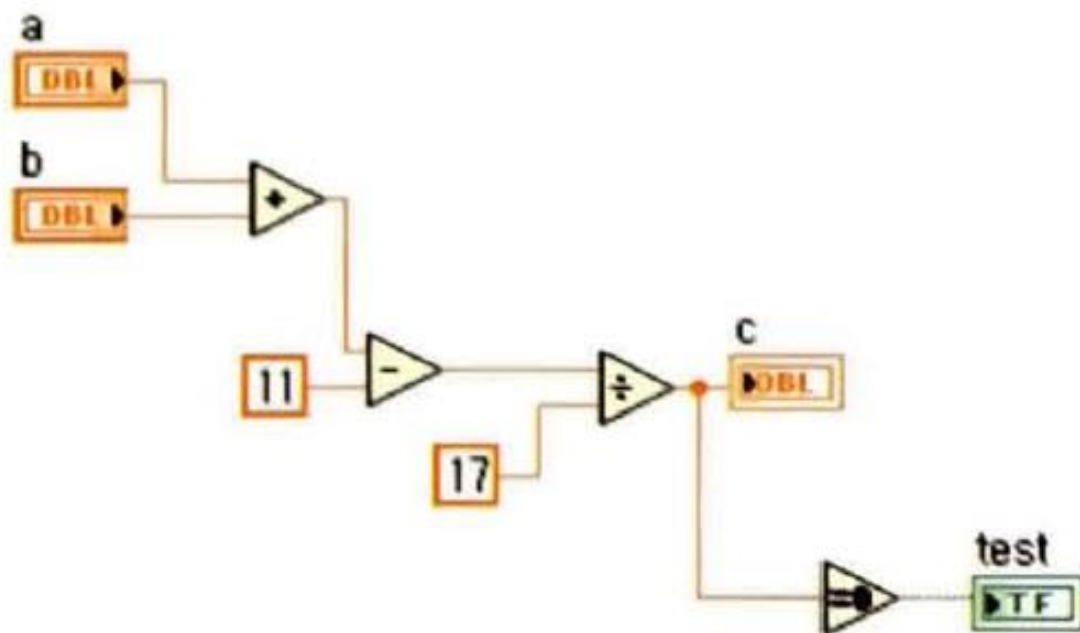
디자인패턴의 개요





단순(Simple) VI 디자인 패턴

- ❖ **While** 루프가 없는 구조로써 한 번 실행하고 종료되는 구조.
- ❖ 반복할 필요가 없는 코드의 경우 적합하며 **subVI**로 많이 사용하는 디자인 패턴





일반(General) VI 디자인 패턴

- ❖ **While** 루프가 하나 있는 구조로써 초기화-반복-종료로 구성
- ❖ 연속 데이터 수집에서 많이 사용하는 디자인 패턴.



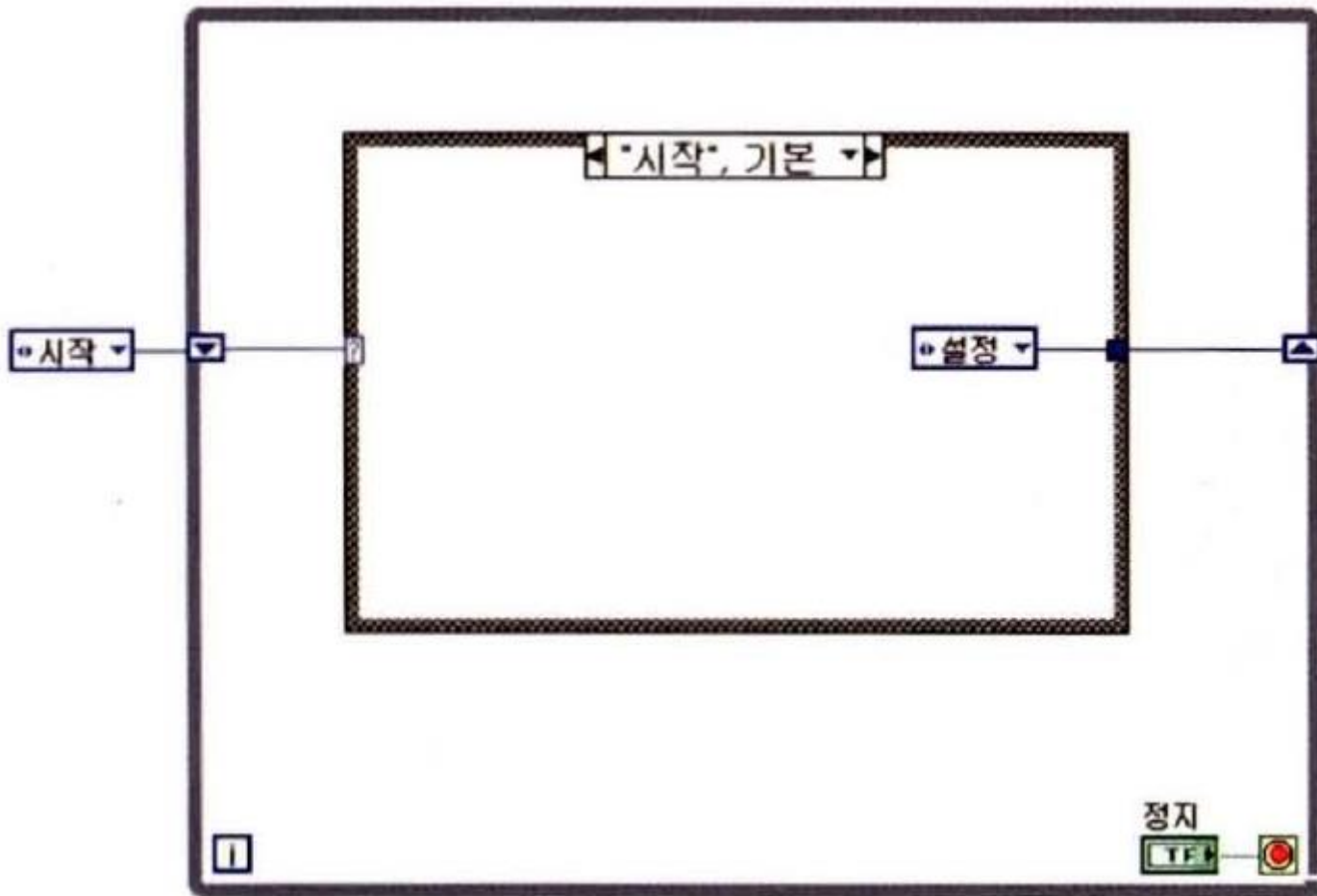


상태 머신(State Machine) 디자인 패턴

- ❖ 상태 머신 디자인 패턴은 프로젝트가 **A**를 실행한 뒤 **B**를 실행하고 **B**의 결과에 따라 **C** 또는 **D**를 실행해야 하는 등의 프로그램을 만들 때 적합한 디자인 패턴
- ❖ 다음 상태를 어떤 상태로든 이동할 수 있다.
- ❖ 그림과 같이 **While** 루프와 케이스 구조를 조합하여 구성
- ❖ 특히 **While** 루프의 시프트 레지스터 기능을 이용하여 다음 상태의 값을 전달
- ❖ 그림에서 시프트 레지스터가 '시작'으로 초기화되어 있어 **While** 루프의 실행 시 '시작' 케이스가 실행되고, 다음 '설정'이 시프트 레지스터에 저장되어 다음 반복할 때 '설정' 케이스가 실행



상태 머신(State Machine) 디자인 패턴





상태 머신(State Machine) 디자인 패턴

- ❖ 상태 머신 디자인 패턴은 프로젝트가 **A**를 실행한 뒤 **B**를 실행하고 **B**의 결과에 따라 **C** 또는 **D**를 시행해야 하는 등의 프로그램을 만들 때 적합한 디자인 패턴
- ❖ 다음 상태를 어떤 상태로든 이동할 수 있다.
- ❖ 그림과 같이 **While** 루프와 케이스 구조를 조합하여 구성
- ❖ 특히 **While** 루프의 시프트 레지스터 기능을 이용하여 다음 상태의 값을 전달

실습8-4-1) 상태머신 디자인 패턴

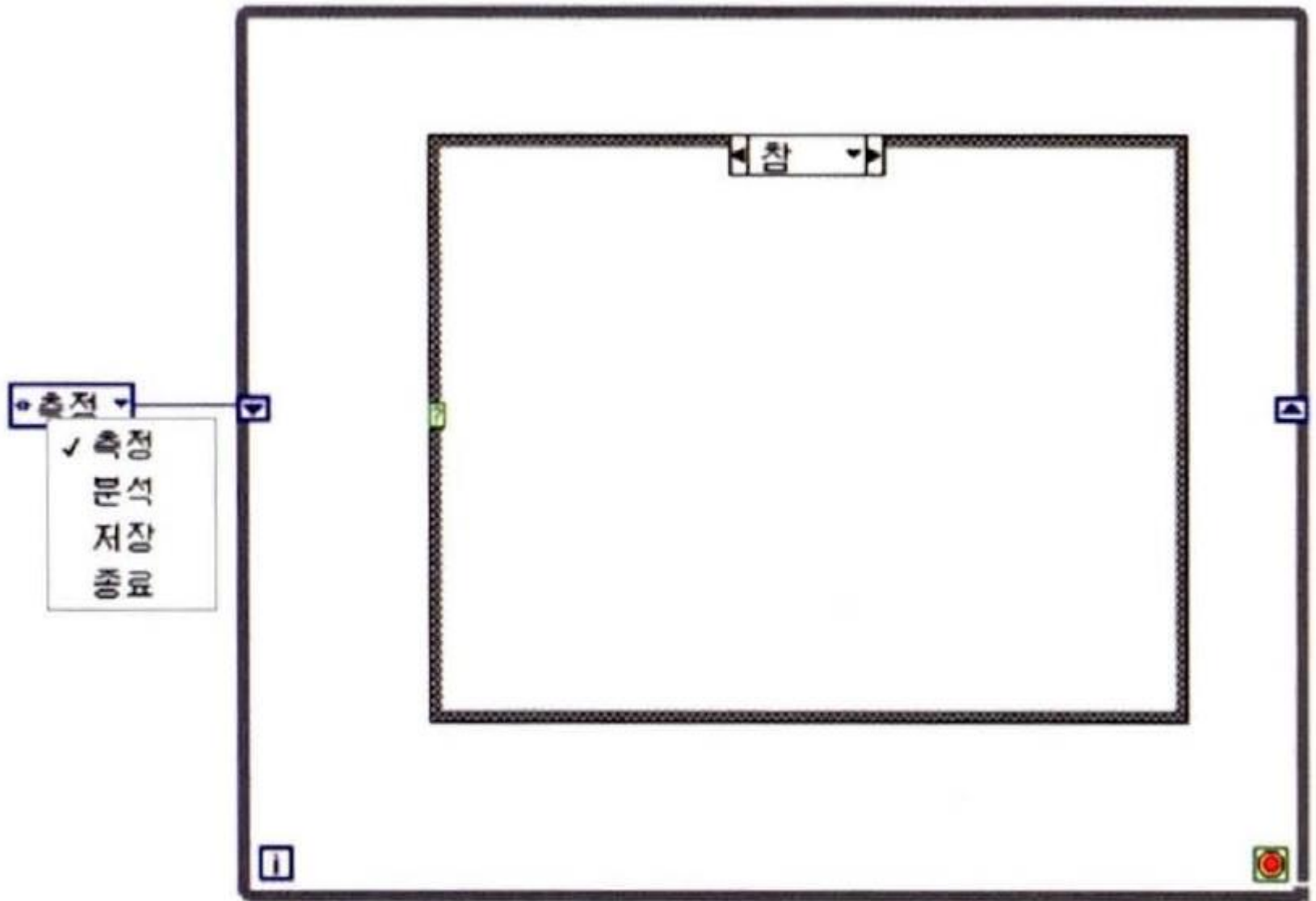


실습8-4-1) 상태머신 디자인 패턴

- ❖ 새 **VI**의 블록다이어그램 에 **While** 루프와 시프트 레지스터, 케이스 구조, 열거형 상수를 배치.
- ❖ 열거형 상수의 바로가기메뉴 > 아이템 편집 ...을 하여 ‘측정’, ‘분석’, ‘저장’, ‘종료’를 정의
- ❖ 그리고 그림과 같이 와이어링한다.



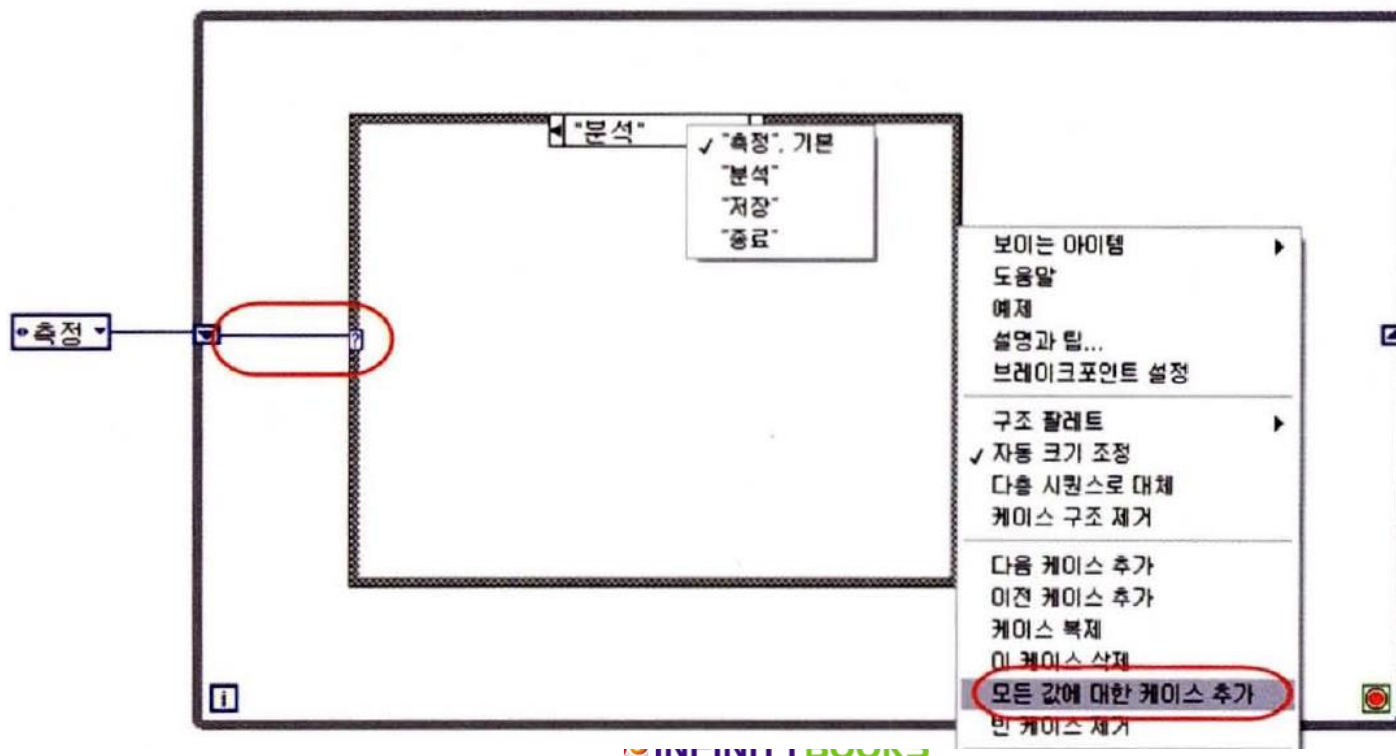
ALGO 4.2. ALPHABETIC FITTING TEST





실습8-4-1) 상태머신 디자인 패턴

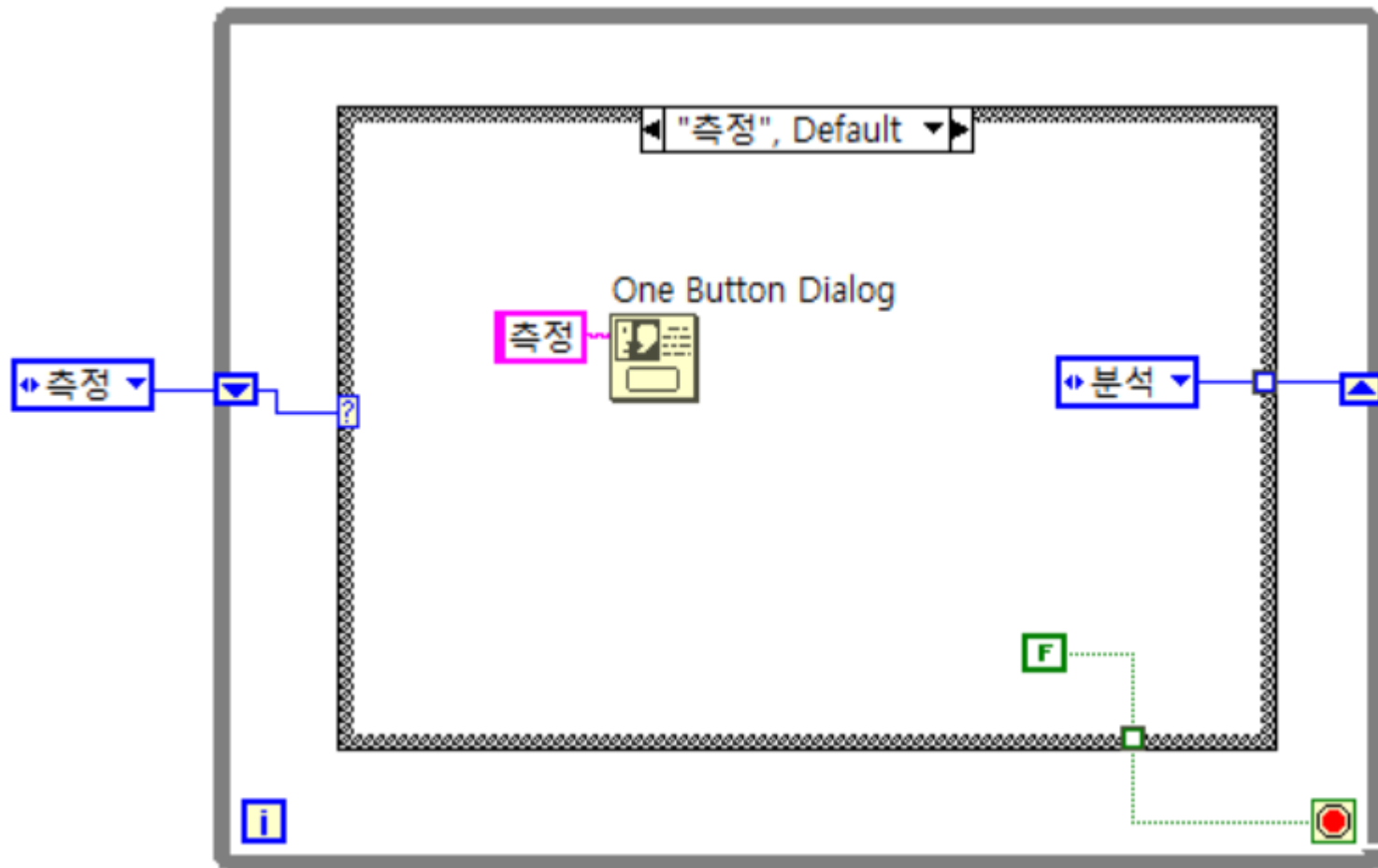
- ❖ 시프트 레지스터를 케이스 구조의 선택자 터미널에 연결한 후, 케이스 구조의 바로가기메뉴 > 모든 값에 대한 케이스 추가를 하게 되면 열거형 상수가 가지고 있는 아이템에 대한 4개의 케이스가 생성





실습8-4-1) 상태머신 디자인 패턴

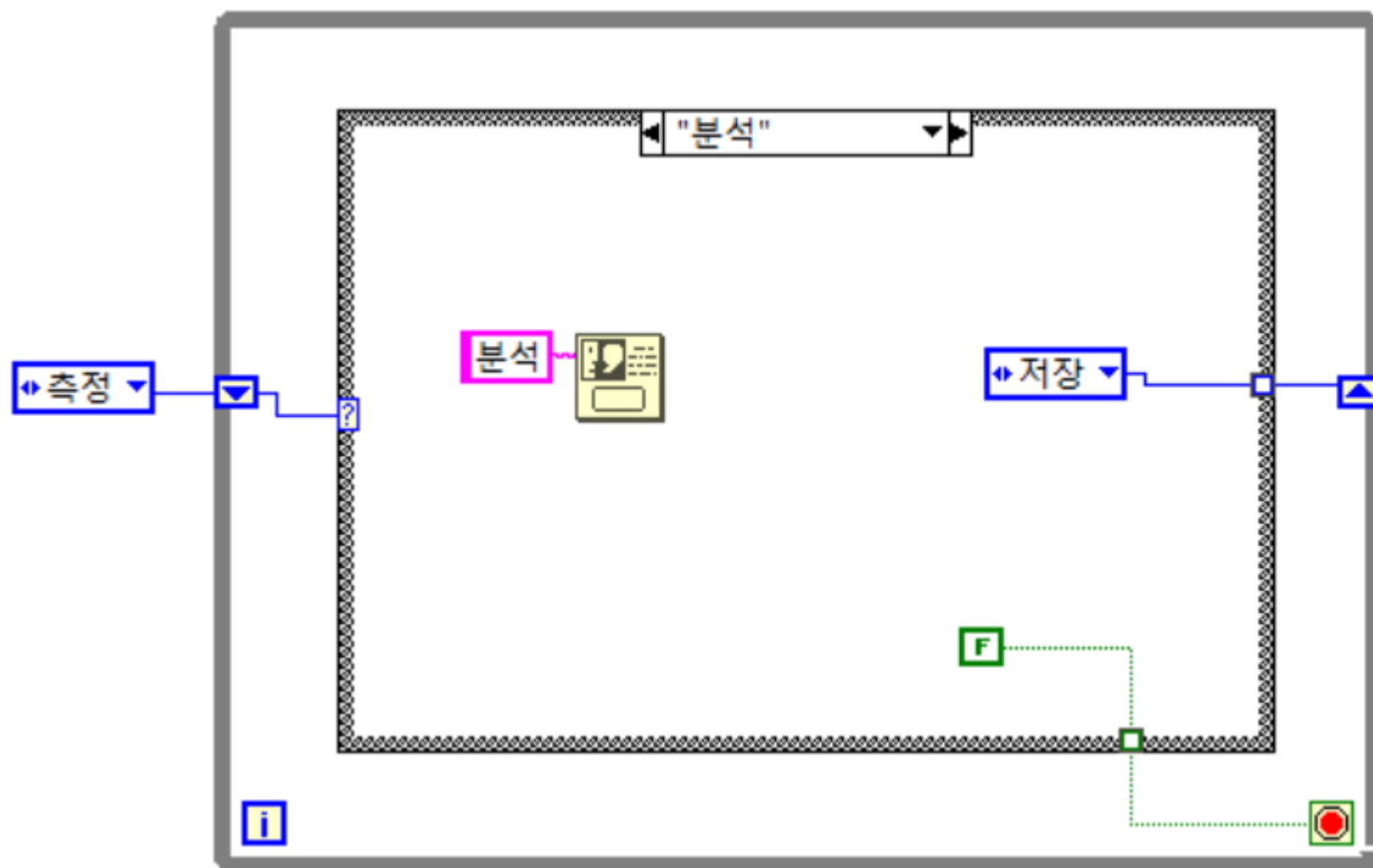
- ❖ ‘측정’ 케이스에 단일 버튼 대화상자.VI와 거짓 상수 그리고 열거형 상수를 그림과 같이 연결





실습8-4-1) 상태머신 디자인 패턴

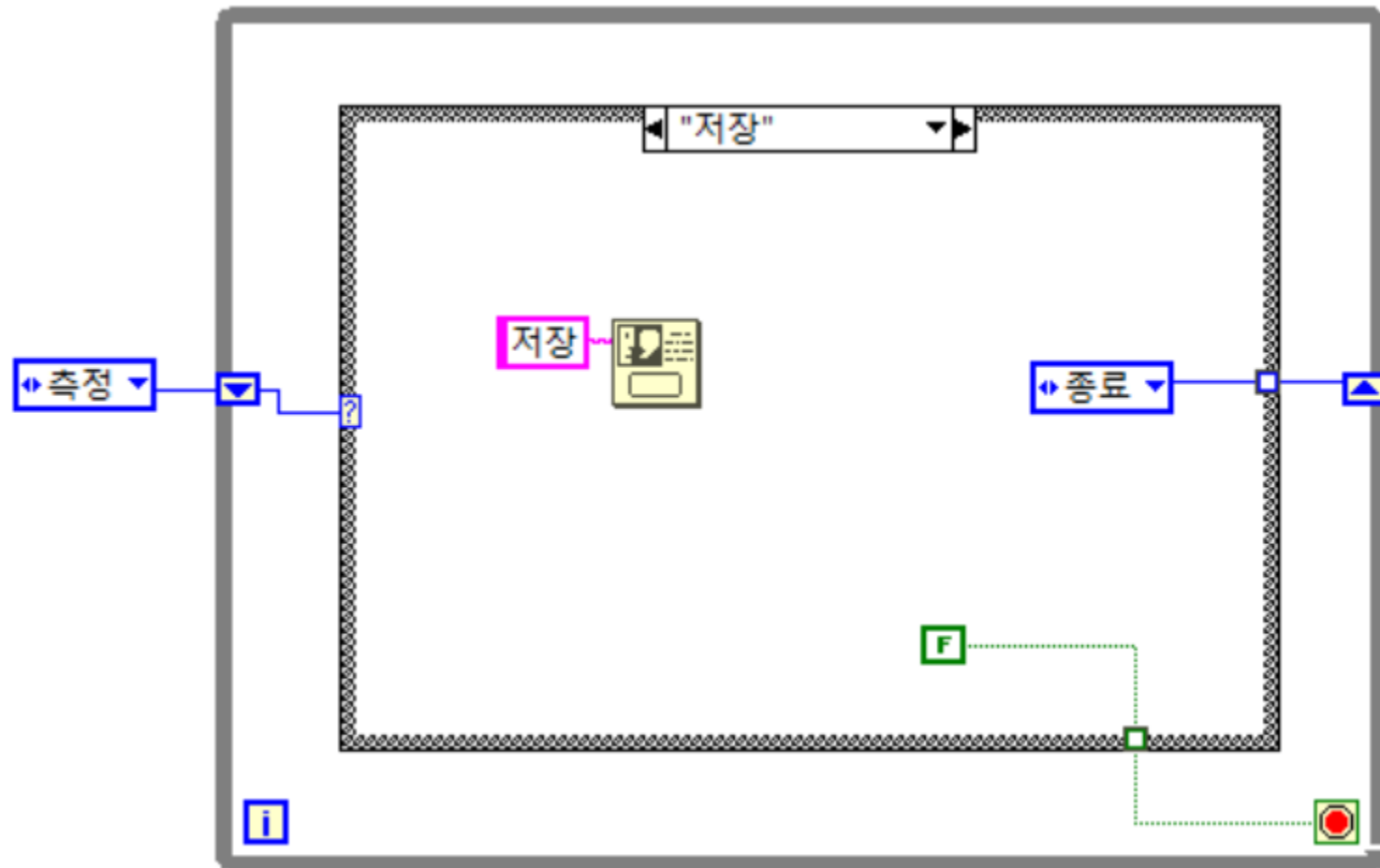
- ❖ ‘분석’ 케이스에 단일 버튼 대화상자.vi와 거짓 상수 그리고 열거형 상수를 그림과 같이 연결.





실습8-4-1) 상태머신 디자인 패턴

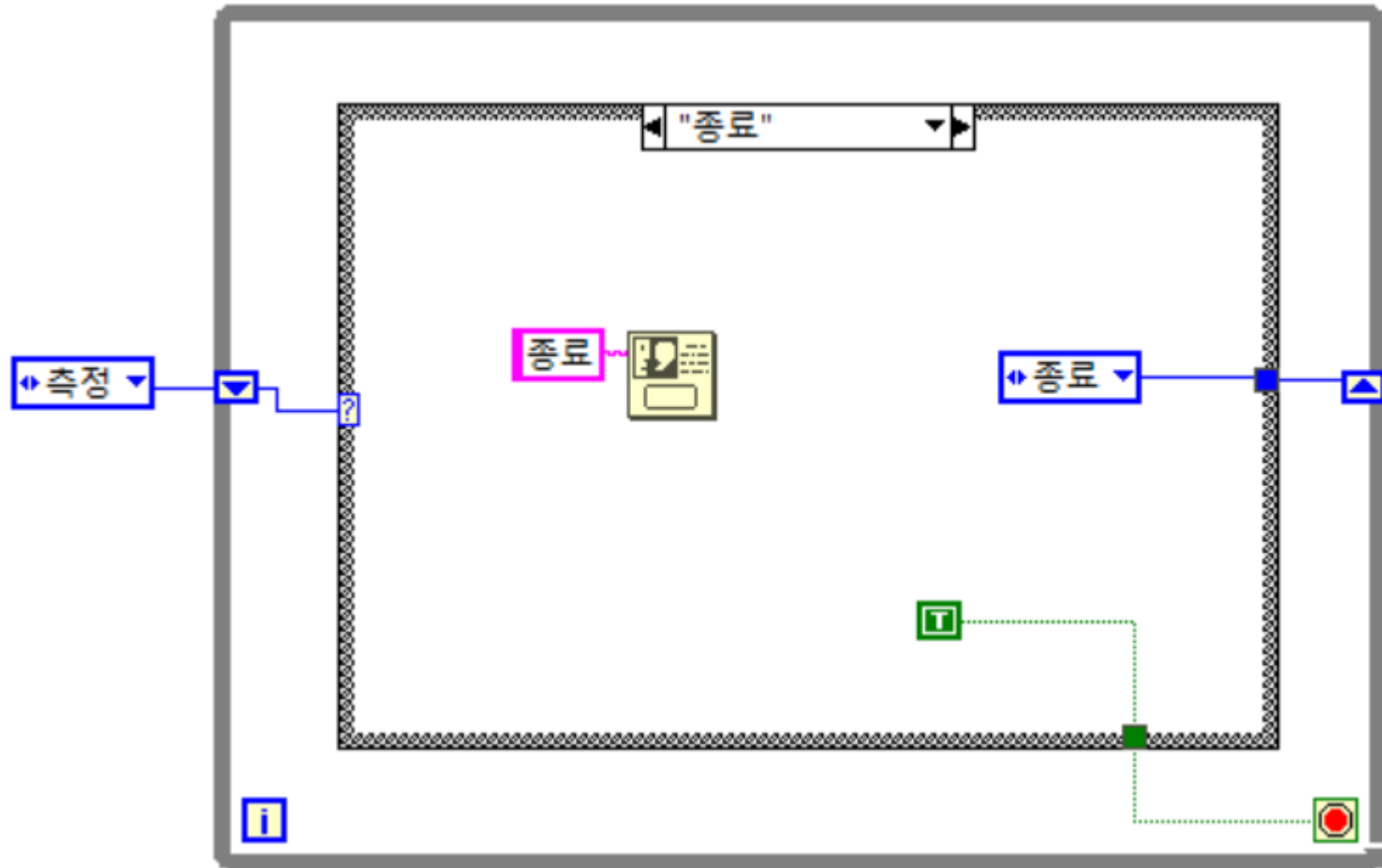
- ❖ ‘저장’ 케이스에 단일 버튼 대화상자.vi와 거짓 상수 그리고 열거형 상수를 그림과 같이 연결.





실습8-4-1) 상태머신 디자인 패턴

- ❖ ‘종료’ 케이스에 단일 버튼 대화상자.vi와 참 상수 그리고 열거형 상수를 그림과 같이 연결.





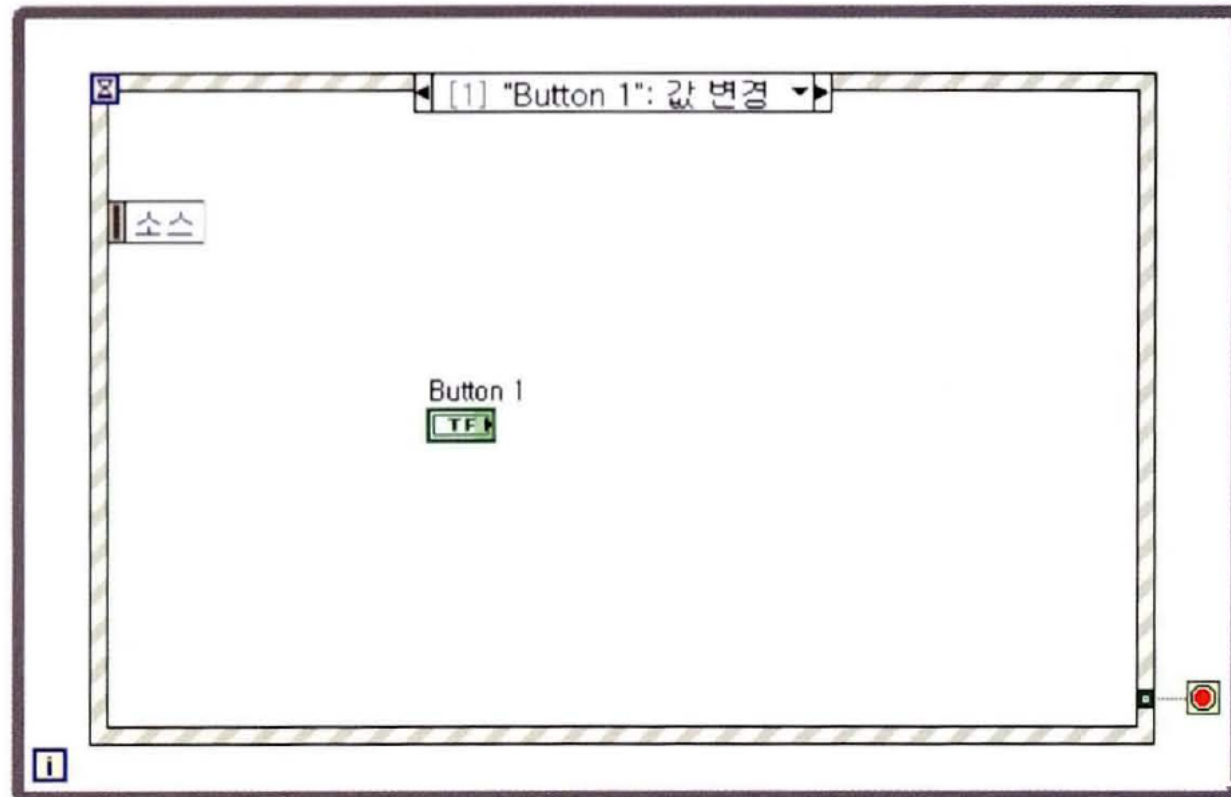
실습8-4-1) 상태머신 디자인 패턴

- ❖ 실행하면 측정 -> 분석 -> 저장 -> 종료 순서로 대화 상자가 나오면서 종료



사용자 인터페이스 이벤트 핸들러 디자인 패턴

- ❖ 사용자 인터페이스인 프런트패널에서 마우스나 키보드의 이벤트(마우스 다운, 마우스 업, 키 다운 등)가 발생할 때 동기화를 맞추어 특정 코드를 실행하고자 할 때 사용하기에 적합한 디자인 패턴





메시지 핸들러 디자인 패턴

- ❖ 메시지 핸들러 디자인 패턴은 프로젝트가 **A**를 실행한 뒤 **B**를 실행하고 **B**를 실행한 뒤 **C**를 실행해야 하는 식의 프로그램을 만들 때 적합한 디자인 패턴.
- ❖ 그림과 같이 **While** 루프와 케이스 구조를 조합하여 구성
- ❖ 특히 **While** 루프의 시프트 레지스터 기능을 이용하여 다음 상태의 값을 전달.
- ❖ 상태 머신 디자인 패턴과 유사.



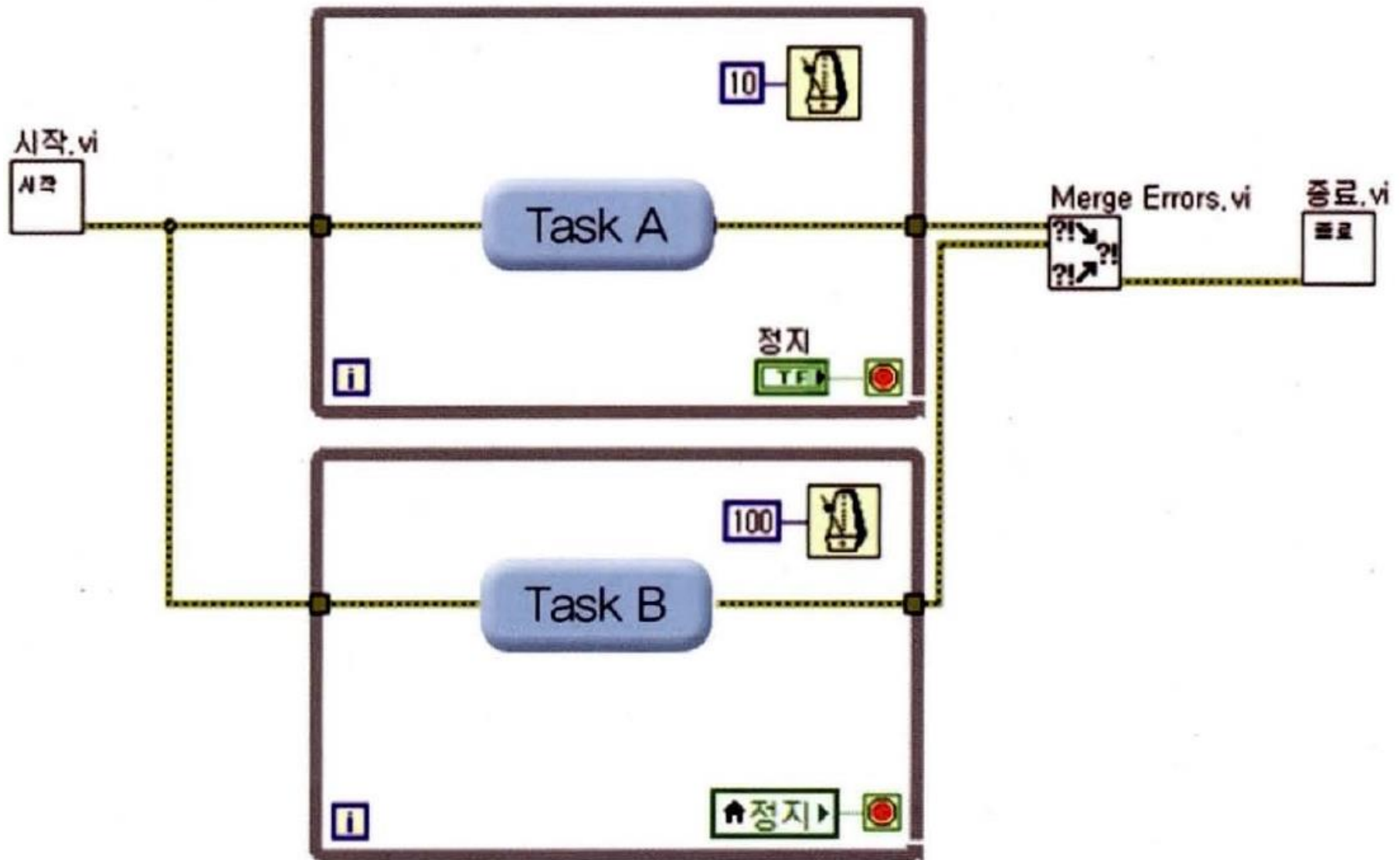


병렬 루프 디자인 패턴

- ❖ 동시에 태스크를 실행하고 응답해야 할 경우 각 태스크를 서로 다른 **While** 루프에 할당하여 병렬 실행.
- ❖ 병렬 루프 디자인 패턴은 두 개 이상의 **While** 루프가 서로 다른 실행 주기를 가지고 동시에 다양한 태스크를 처리할 때 주로 사용.
- ❖ **While** 루프 간에 실행 중에는 실시간으로 데이터를 주고받고 하는 관계가 없는 즉 **While** 루프 각각 독립적인 일을 처리하고자 할 때 사용하기에 적합



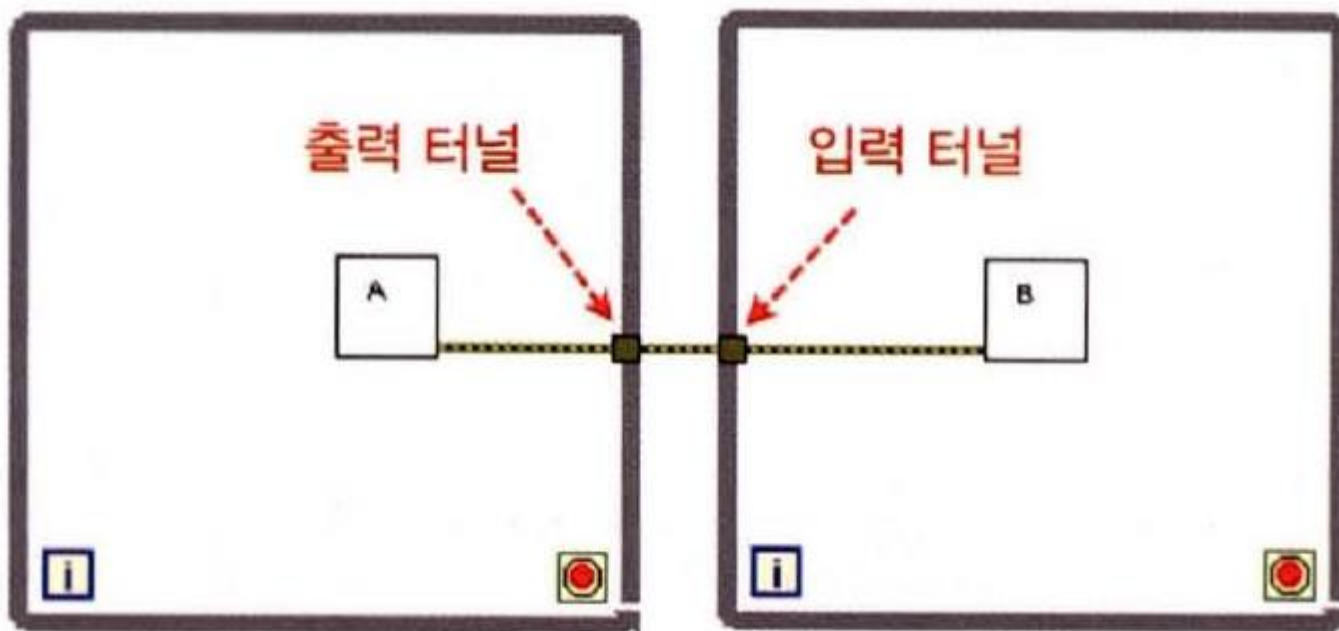
병렬 루프 디자인 패턴





멀티 루프 간의 실시간 데이터 공유

- ❖ 그림에서 보면 첫 번째 **While** 루프 노드의 출력 터널에서 값이 출력되려면 **While** 루프 노드의 실행이 끝나야 하고, 두 번째 **While** 루프 노드가 시작될려면 입력 터널로부터 값을 받아야 실행 가능.
- ❖ 즉 와이어를 이용해서는 멀티 루프 간에 실시간으로 데이터를 공유할 수 없다





멀티 루프 간의 실시간 데이터 공유

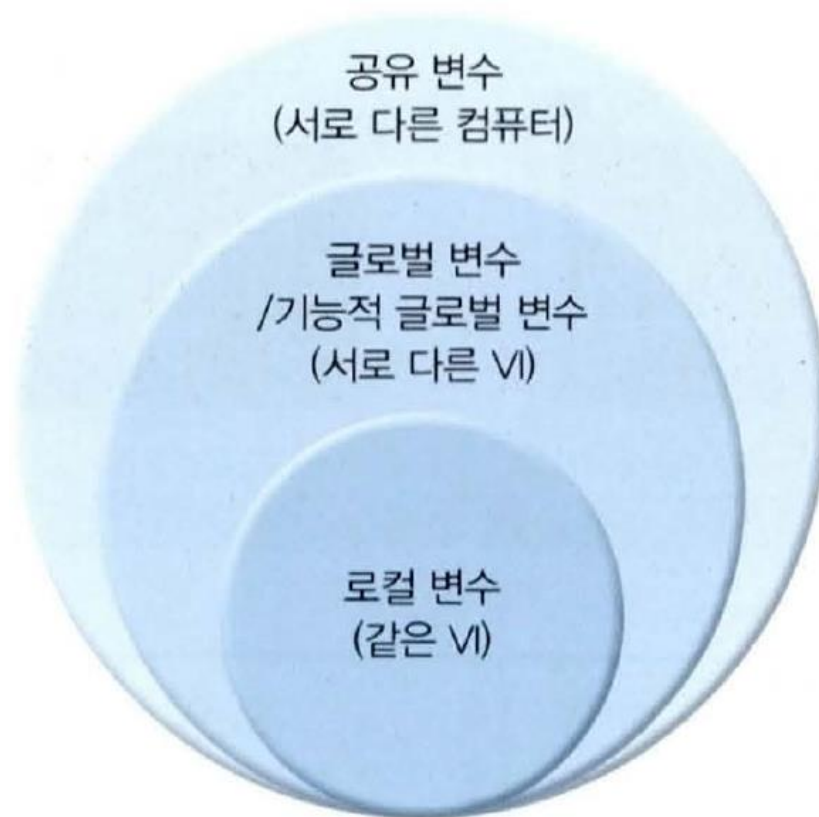
❖ 그럼 멀티 루프 간에 데이터를 실시간으로 공유하려면 어떤 방법을 사용해야 할까?





멀티 루프 간의 실시간 데이터 공유

- ❖ 변수는 읽기와 쓰기 속성을 가지고 있으며, 크게 로컬 변수, 글로벌 변수, 기능적 글로벌 변수, 공유 변수로 구분.
- ❖ 기능적 차이는 없으며, 사용 범위의 차이가 있다.





멀티 루프 간의 실시간 데이터 공유

- ❖ 가장 사용 범위가 좁은 로컬 변수를 생성하는 방법은 공유하고자 하는 컨트롤 또는 인디케이터의 터미널에서 바로가기메뉴 > 생성 > 로컬 변수를 선택
- ❖ 로컬 변수를 처음 생성하면 쓰기 속성을 가지고 있다.
- ❖ 이를 읽기 속성으로 변경하고자 할 때, 로컬 변수의 바로가기메뉴 > 읽기로 변경을 선택.



멀티 루프 간의 실시간 데이터 공유

❖ 로컬변수 생성

a
b
c

- 보이는 아이템 ▶
- 찾기 ▶
- 컨트롤 숨기기
- 인디케이터로 변경
- 상수로 변경
- 설명과 팁...
- 문자열 팔레트 ▶
- 생성 ▶
- 데이터 처리 ▶
- 고급 ▶
- 타입 정의 만들기
- 아이콘으로 보기
- 프로퍼티

- 상수
- 컨트롤
- 인디케이터
- 로컬 변수
- 참조
- 프로퍼티 노드 ▶
- 인보크 노드 ▶

a b c

- 보이는 아이템 ▶
- 찾기 ▶
- 아이템 선택 ▶
- 읽기로 변경
- 도움말
- 예제
- 설명과 팁...
- 브레이크포인트 ▶
- 문자열 팔레트 ▶
- 생성 ▶
- 대체 ▶
- 프런트패널 열기
- 프로퍼티

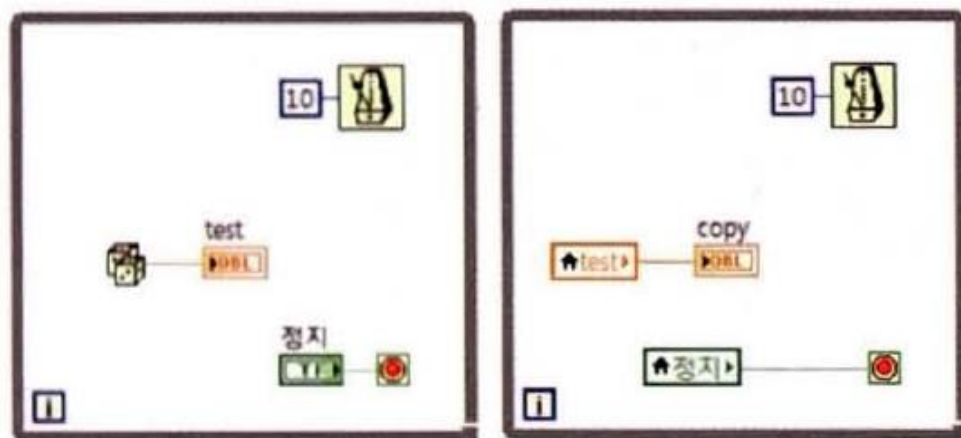
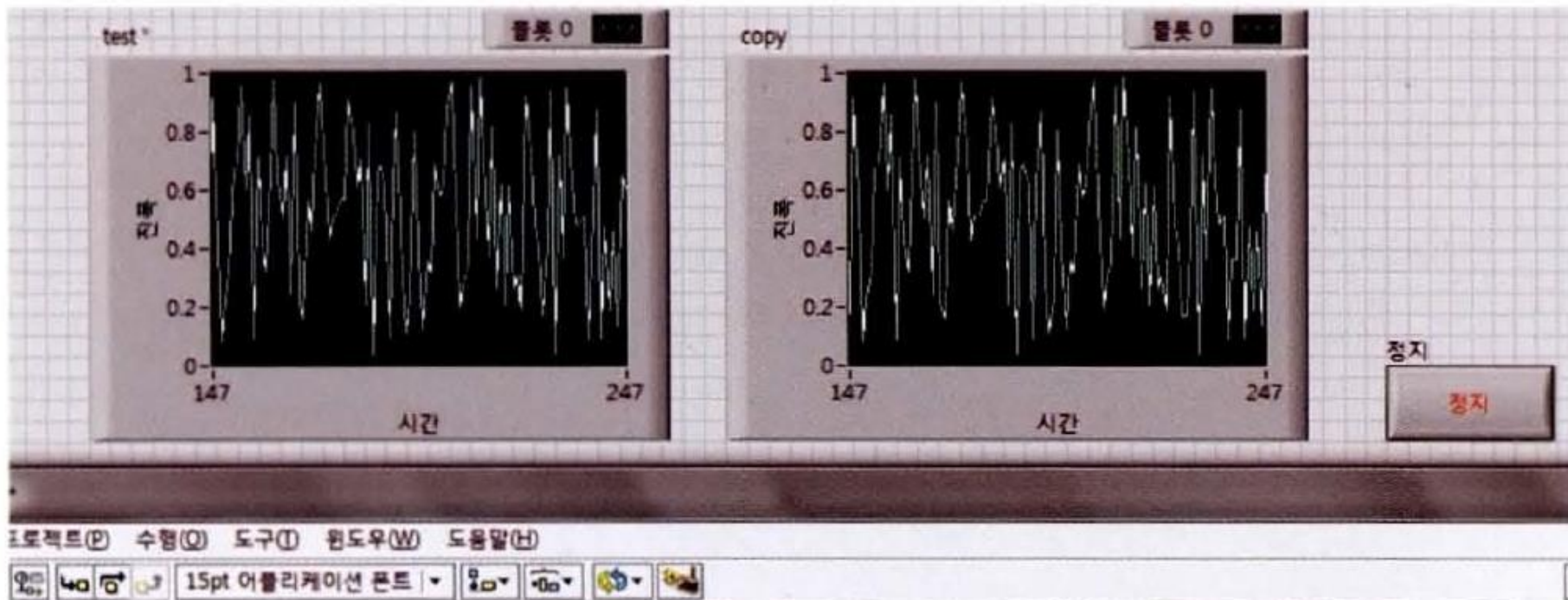


멀티 루프 간의 실시간 데이터 공유

- ❖ 그림에서 로컬 변수를 이용하여 같은 **VI** 내에서 두 개의 **While** 루프를 하나의 정지 버튼을 이용하여 모든 **While** 루프를 정지하고 'test'에 디스플레이되는 값을 로컬 변수로 만들어서 'copy'에 연결하여 두 웨이브폼 차트에 서 같은 값이 디스플레이되도록 하였다.
- ❖ 이때 두 로컬 변수의 속성은 로컬 변수에 저장된 값을 읽어와야 하므로 읽기 속성을 사용한다.



머티 큰따 가이 시시가 데이타 고오





멀티 루프 간의 실시간 데이터 공유

- ❖ 그림에서 로컬 변수를 이용하여 같은 **VI** 내에서 두 개의 **While** 루프를 하나의 정지 버튼을 이용하여 모든 **While** 루프를 정지하고 'test'에 디스플레이되는 값을 로컬 변수로 만들어서 'copy'에 연결하여 두 웨이브폼 차트에 서 같은 값이 디스플레이되도록 하였다.
- ❖ 이때 두 로컬 변수의 속성은 로컬 변수에 저장된 값을 읽어와야 하므로 읽기 속성을 사용한다.

실습8-8-1) 로컬 변수 사용법



실습8-8-1) 로컬 변수 사용법

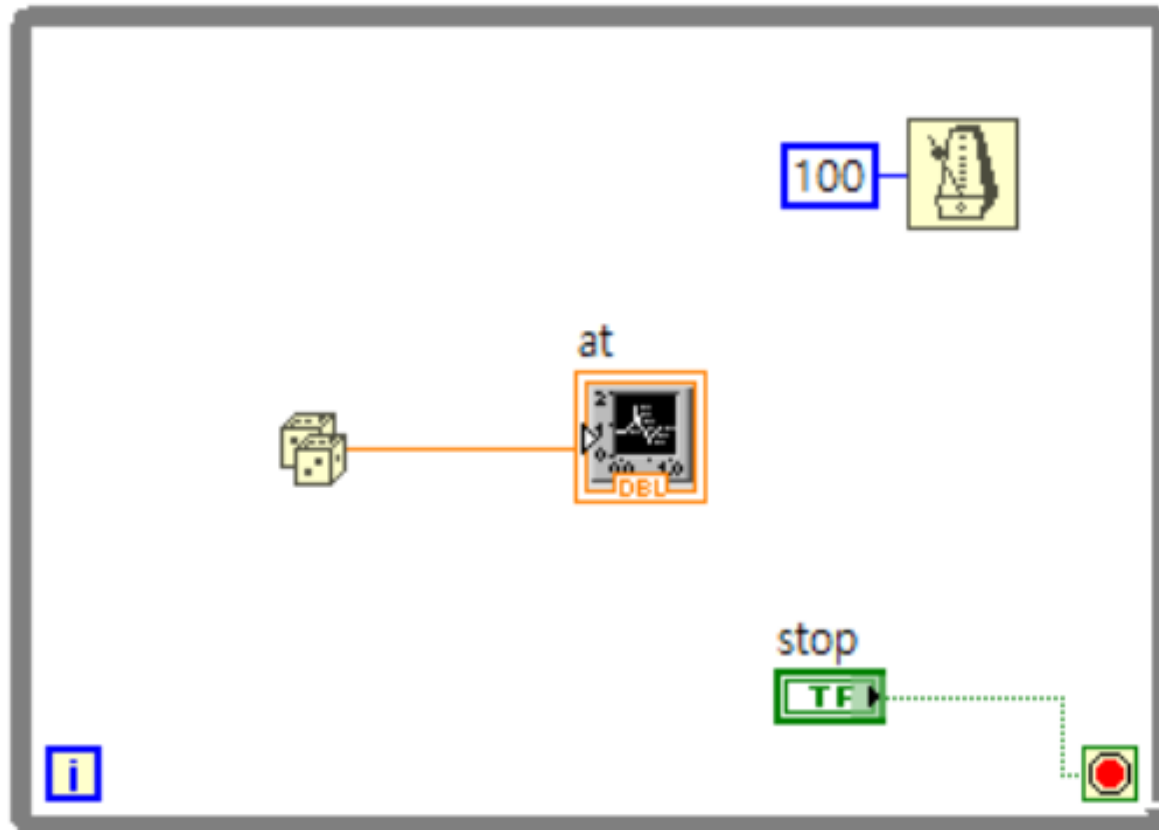
❖ 새 **VI**를 열고 그림과 같이 프런트패널을 구성





실습8-8-1) 로컬 변수 사용법

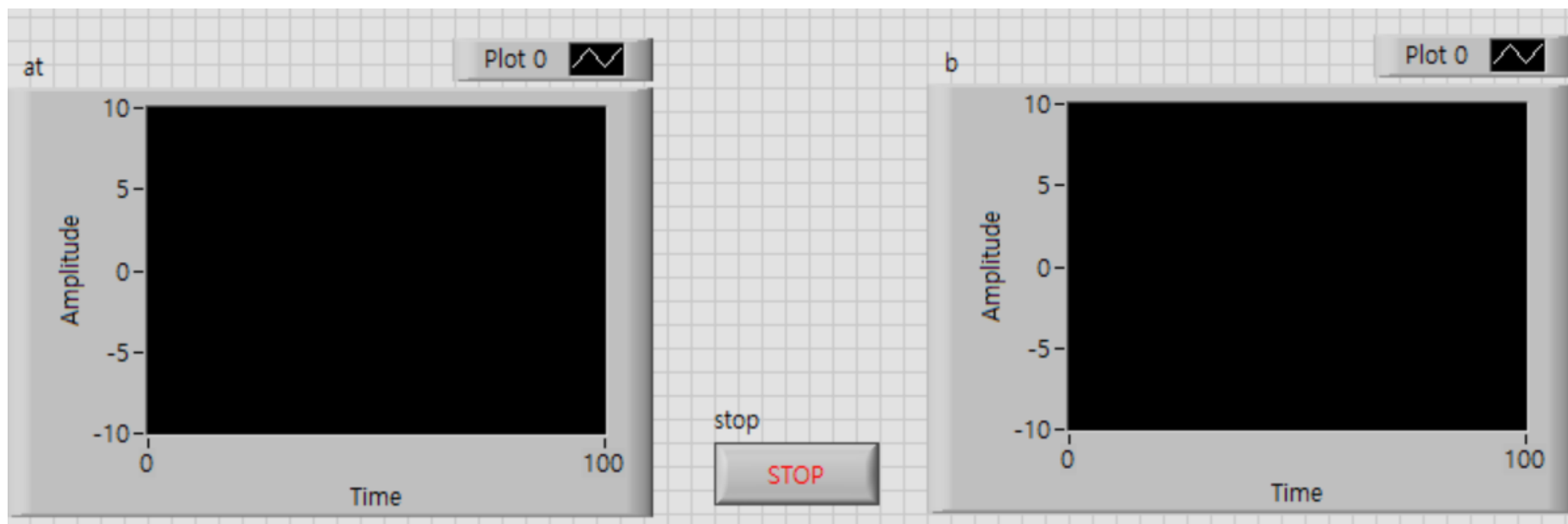
- ❖ 그림과 같이 블록다이어그램을 구성
- ❖ 실행을 하면 **0**에서 **1** 사이 의 임의의 숫자가 웨이브폼 차트에 출력





실습8-8-1) 로컬 변수 사용법

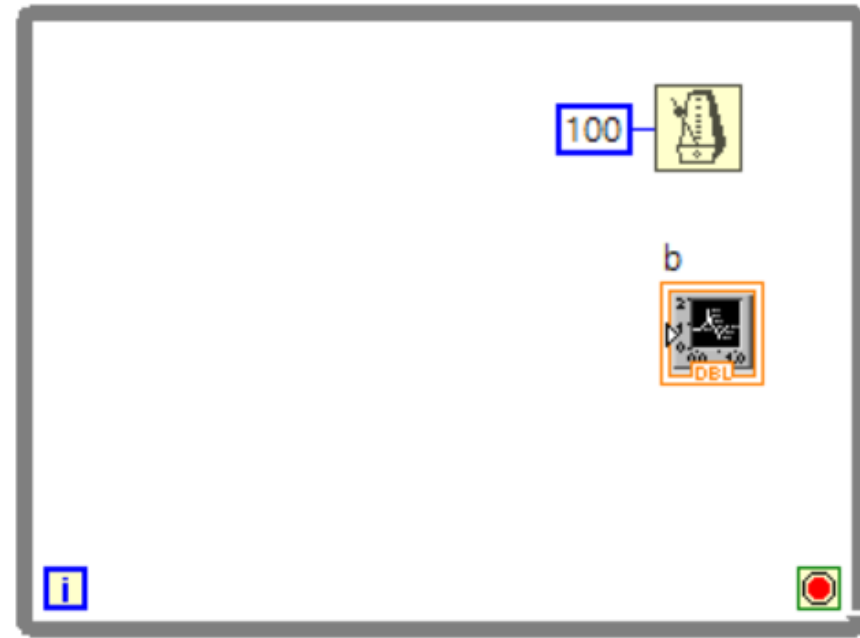
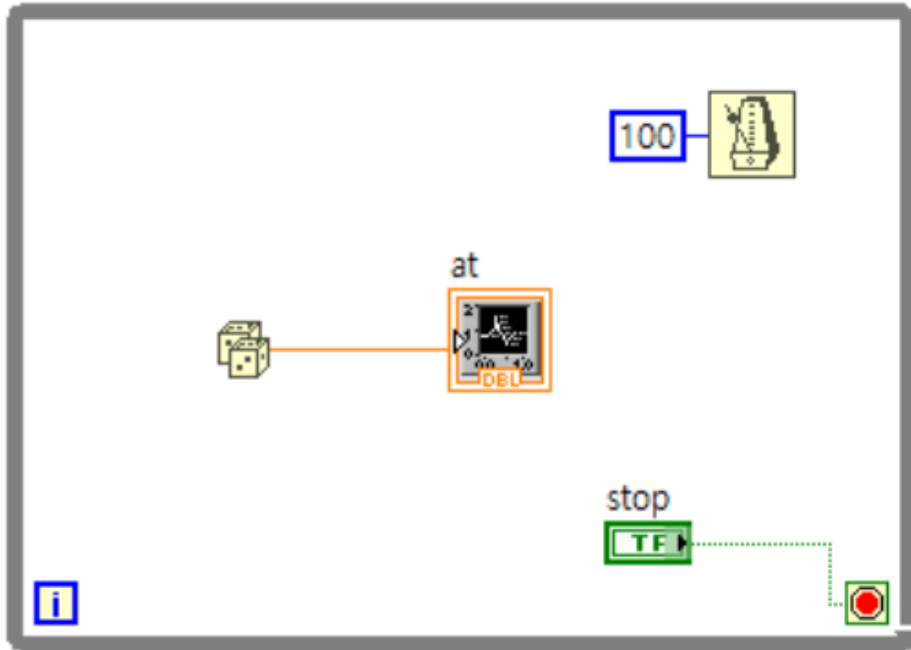
❖ 프런트패널에 라벨이 'b'인 웨이브폼 차트를 추가.





실습8-8-1) 로컬 변수 사용법

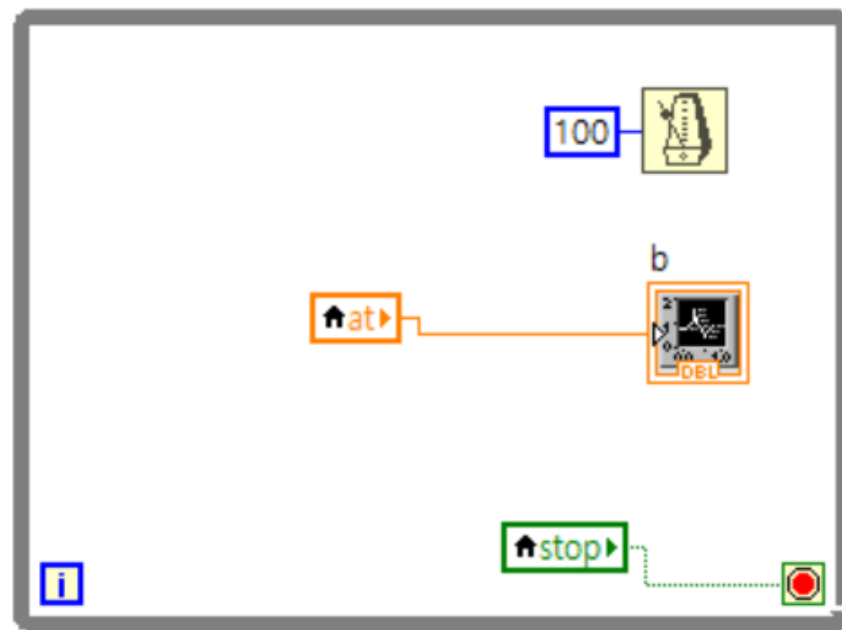
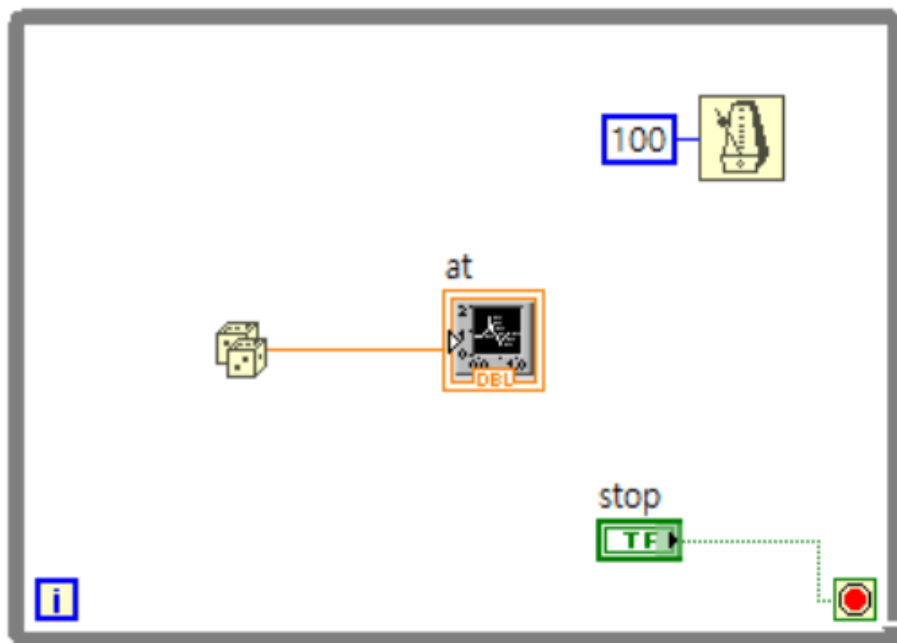
❖ 블록다이어그램을 그림과 같이 수정





실습8-8-1) 로컬 변수 사용법

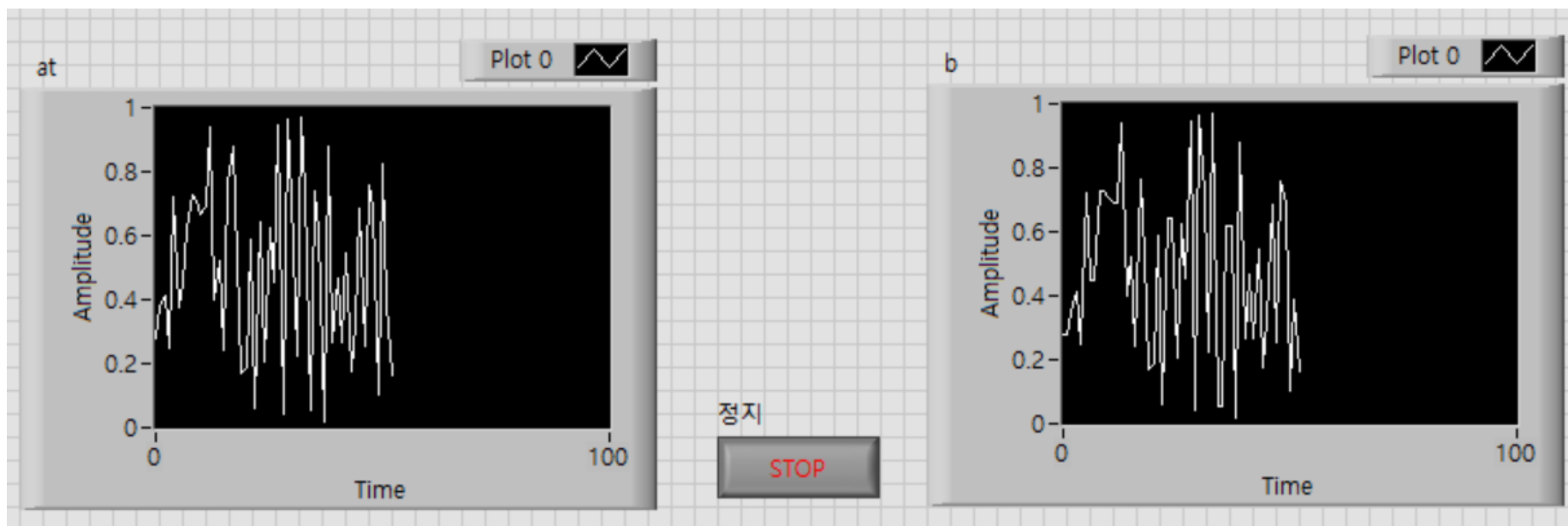
- ❖ ‘a’와 ‘정지’ 터미널에서 각각 바로가기메뉴 > 생성 > 로컬 변수를 생성하고 생성된 각각의 로컬 변수들을 바로가기메뉴 > 읽기로 변경하여 그림과 같이 와이어링한다





실습8-8-1) 로컬 변수 사용법

❖ 실행하면 'b'는 'a'와 같은 값이 디스플레이되고, '정지' 버튼이 눌러지면 두 개의 **While** 루프가 동시에 정지된다



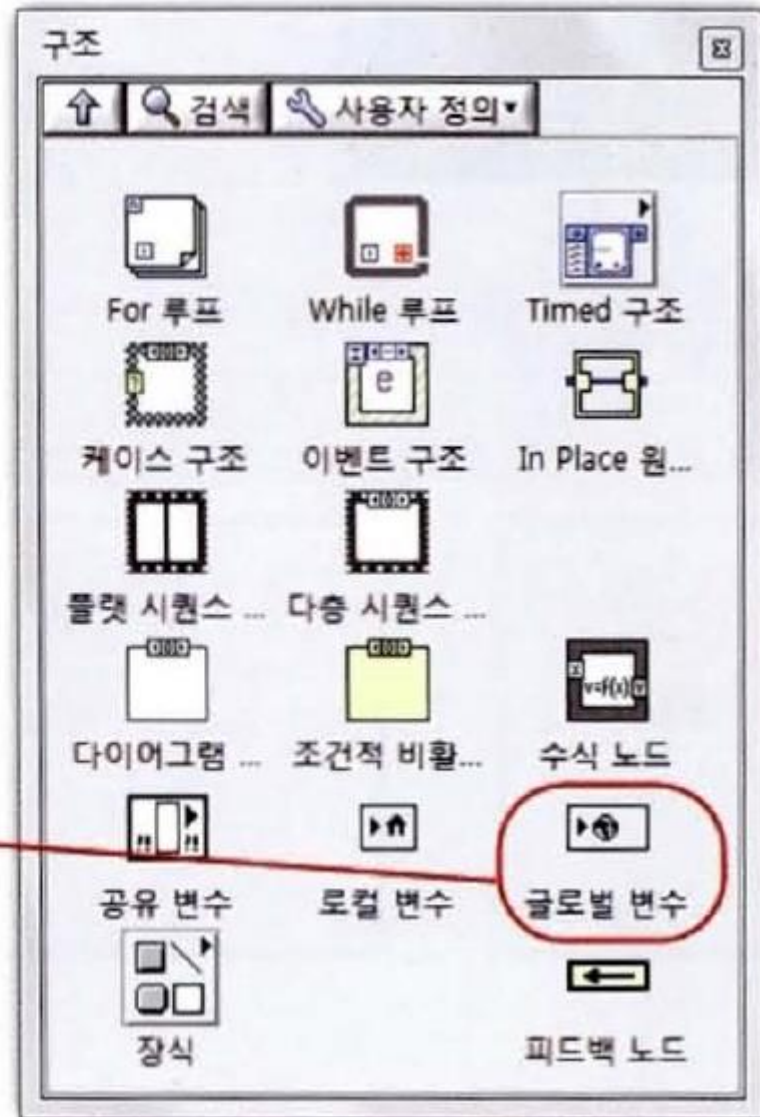
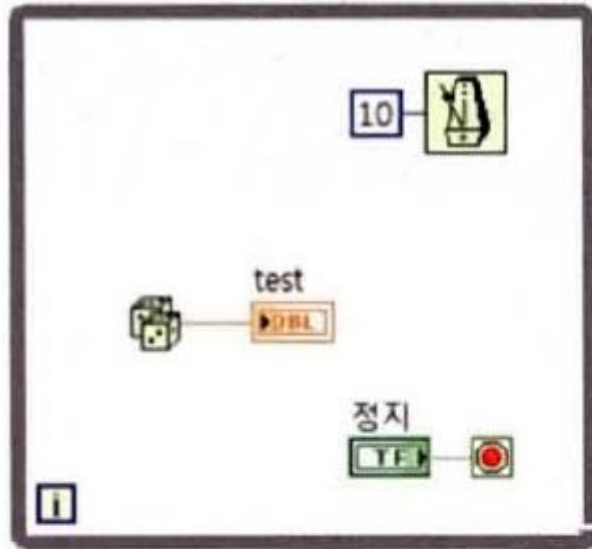


멀티 루프 간의 실시간 데이터 공유

- ❖ 글로벌 변수는 서로 다른 **VI**간에 데이터를 공유할 수 있다
- ❖ 예를 들어 **a.vi**의 정지 버튼이 눌러지면 **b.vi**의 **While** 루프도 종료될 수 있도록 할 때는 글로벌 변수를 이용하여 구현한다.
- ❖ 글로벌 변수를 만드는 방법은 그림과 같이 함수 팔레트 > 구조 > 글로벌 변수를 블록다이어그램에 위치시킨다.



멀티 루프 간의 실시간 데이터 공유



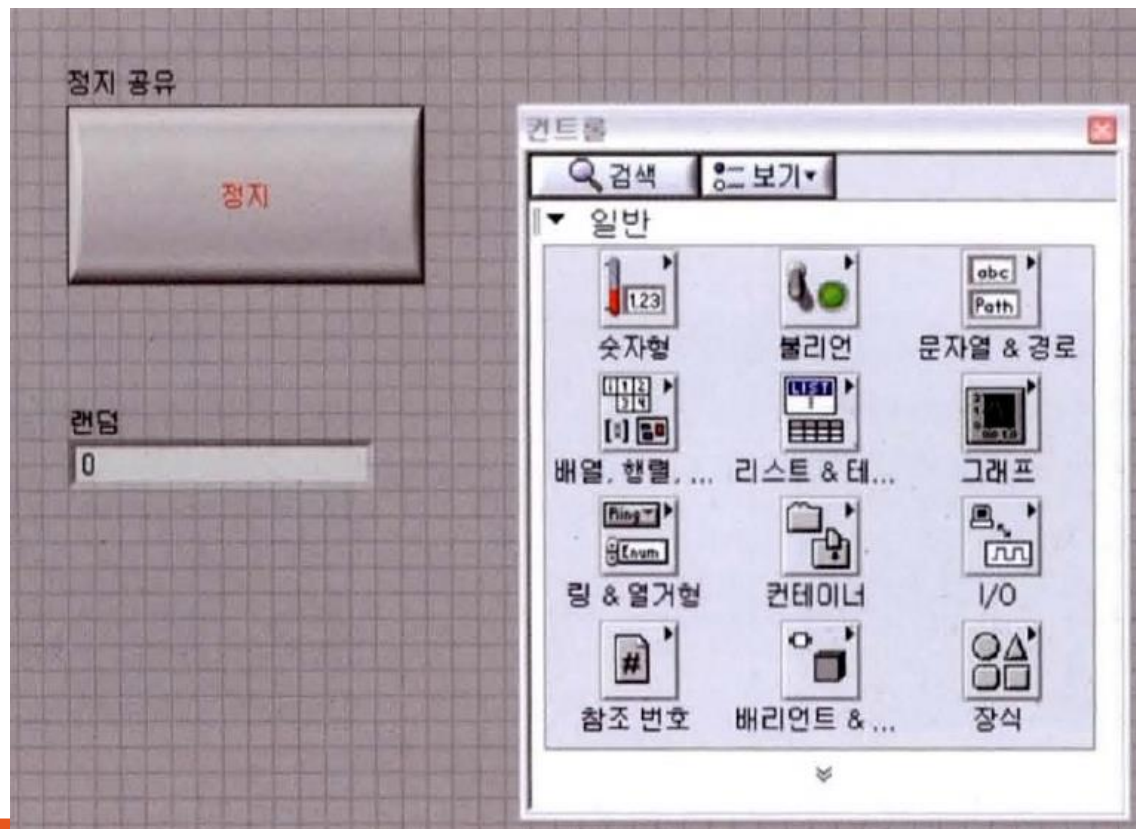
있다
le 루
여 구

≡ >



멀티 루프 간의 실시간 데이터 공유

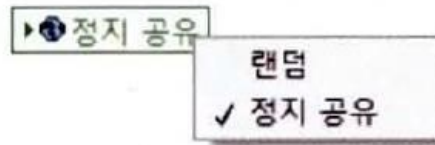
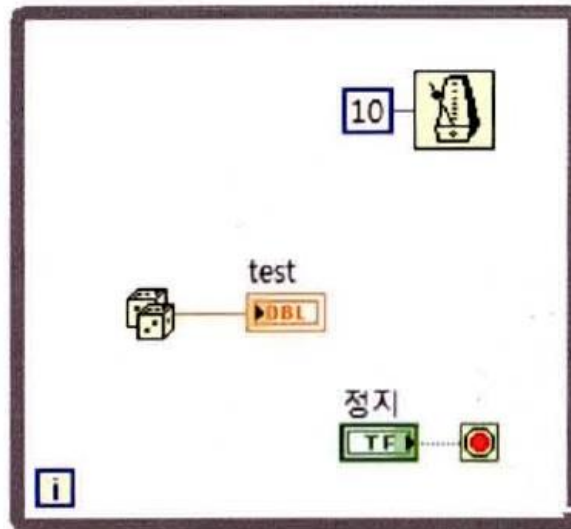
- ❖ 위치시킨 글로벌 변수를 더블 클릭하면 그림과 같이 프런트 패널만 있는 새로운 창이 하나 뜬다.
- ❖ 공유할 데이터 타입을 컨트롤 팔레트를 이용하여 프런트패널에 가져다 놓고 저장한 후 창을 닫는다.





멀티 루프 간의 실시간 데이터 공유

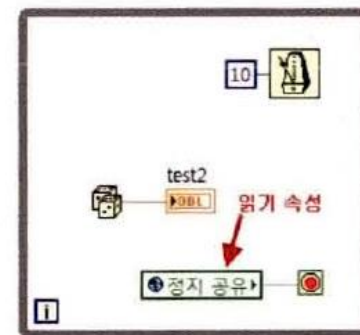
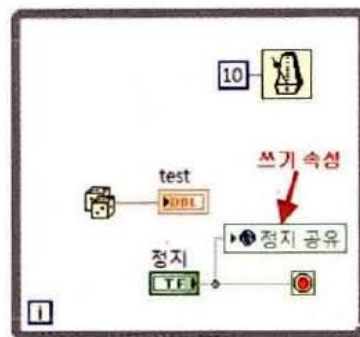
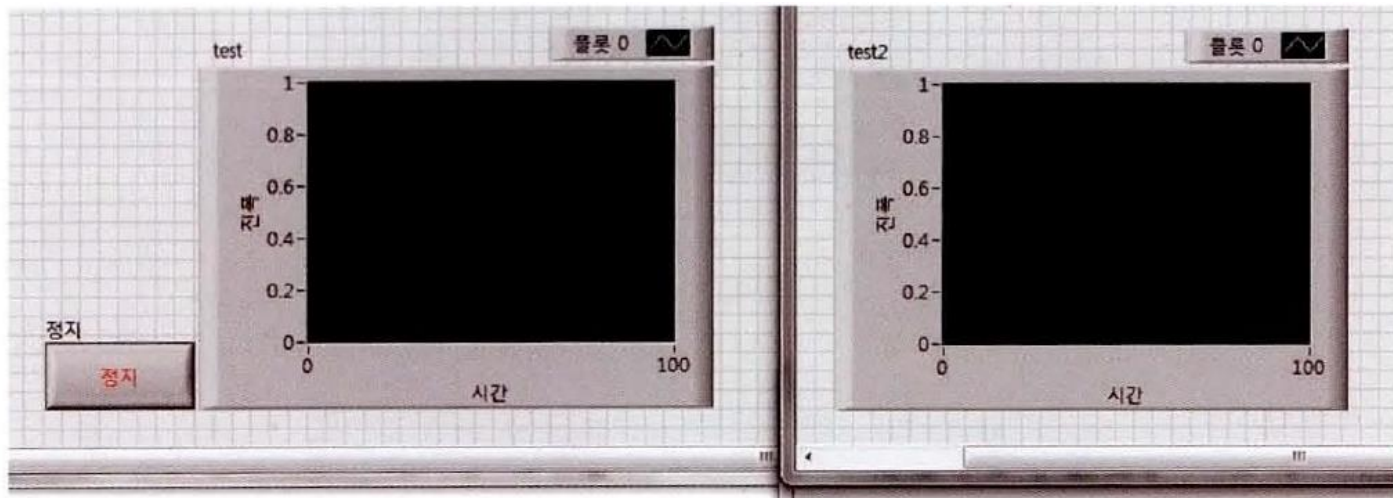
- ❖ 다시 블록다이어그램으로 돌아와서 글로벌 변수의 바로가기 메뉴 > 아이템 선택을 하면 앞에서 만든 컨트롤 또는 인디케이터의 라벨이 나타난다
- ❖ 그 중 사용하고 싶은 것을 선택하면 된다.





멀티 루프 간의 실시간 데이터 공유

- ❖ 그림에서 왼쪽 **VI**에서 정지 버튼을 눌렀을 때, 오른쪽 **VI** 외 왼쪽 **VI** 둘 다 프로그램을 종료시킬 수 있도록 하기 위해 정지 버튼의 값을 글로벌 변수를 이용하여 공유





멀티 루프 간의 실시간 데이터 공유

- ❖ 왼쪽 **VI**에서 블록다이어그램을 보면 ‘정지 공유’ 글로벌 변수를 쓰기 속성으로 설정하여 ‘정지’의 값을 글로벌 변수에 쓴다.
- ❖ 오른쪽 **VI**의 블록다이어그램에서 ‘정지 공유’ 글로벌 변수를 읽기 속성으로 설정하여 왼쪽 **VI**에서 글로벌 변수에 쓴 값을 읽어 온 후 오른쪽 **VI**의 조건 터미널에 연결한다.

실습8-8-2] 글로벌 변수 사용법



실습8-8-2) 글로벌 변수 사용법

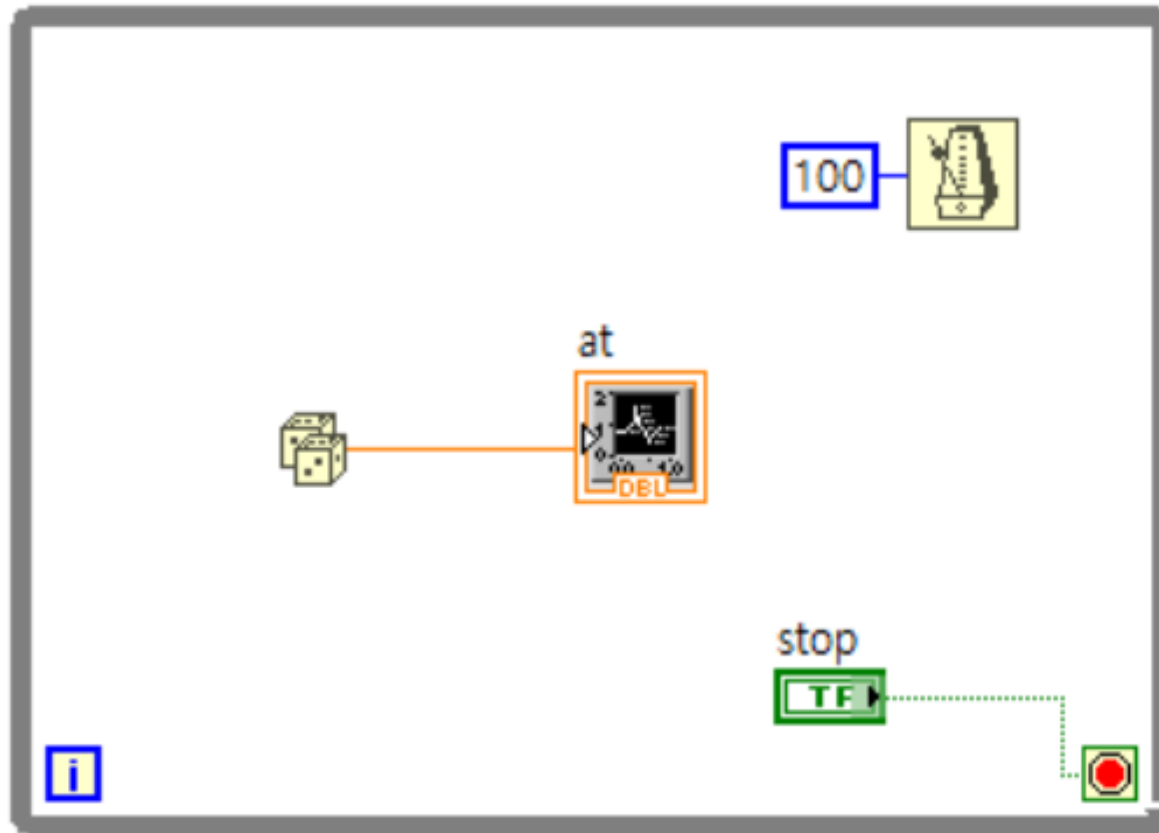
❖ 새 **VI**를 열고 그림과 같이 프런트패널을 구성





실습8-8-2) 글로벌 변수 사용법

- ❖ 그림과 같이 블록다이어그램을 구성
- ❖ 실행을 하면 **0**에서 **1** 사이 의 임의의 숫자가 웨이브폼 차트에 출력





실습8-8-2) 글로벌 변수 사용법

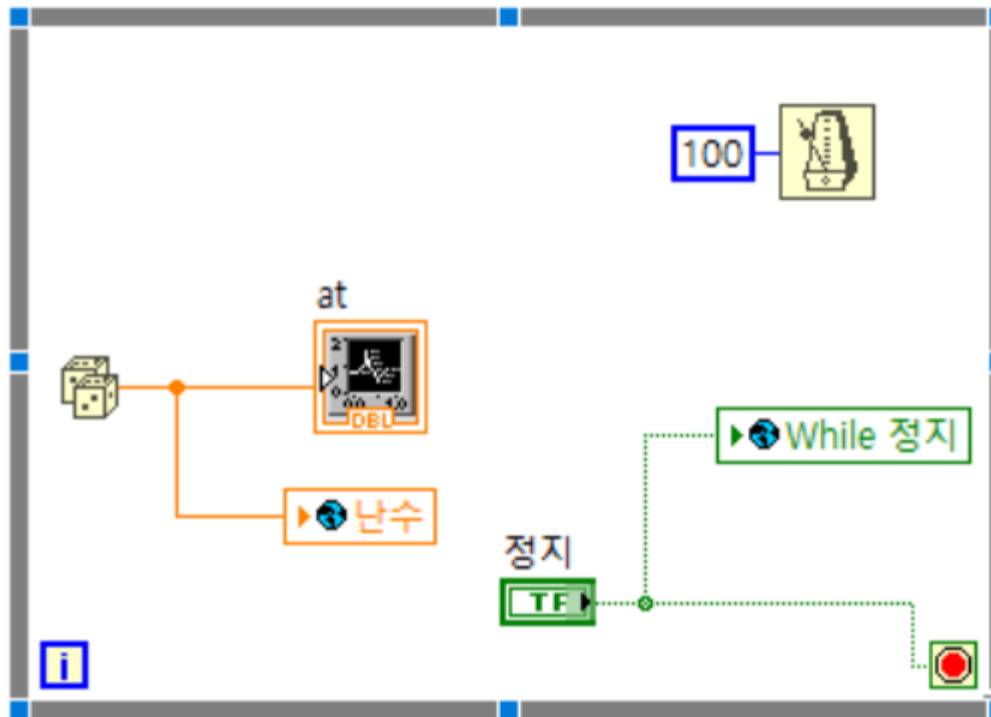
- ❖ 블록다이어그램에 글로벌 변수를 하나 가져다 놓고 더블 클릭하여 글로벌 변수를 열어 그림과 같이 '난수' 웨이브폼 차트와 '**While** 정지' 불리언 버튼을 추가하고 '글로벌 변수'라고 저장하고 달는다.





실습8-8-2) 글로벌 변수 사용법

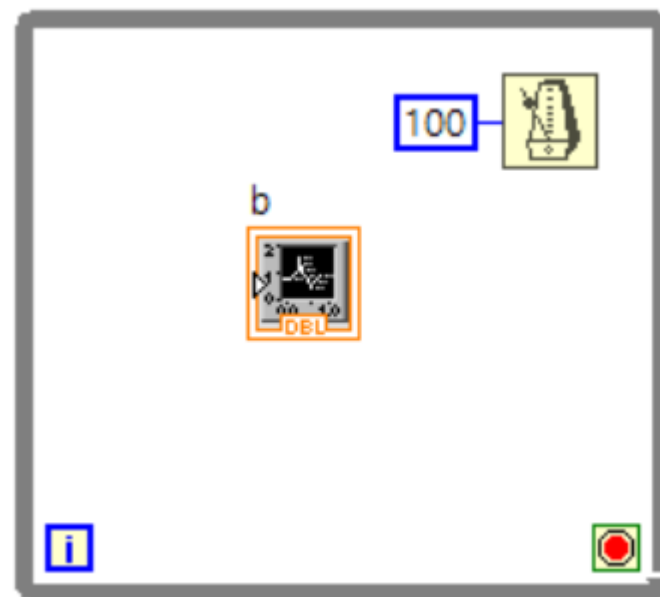
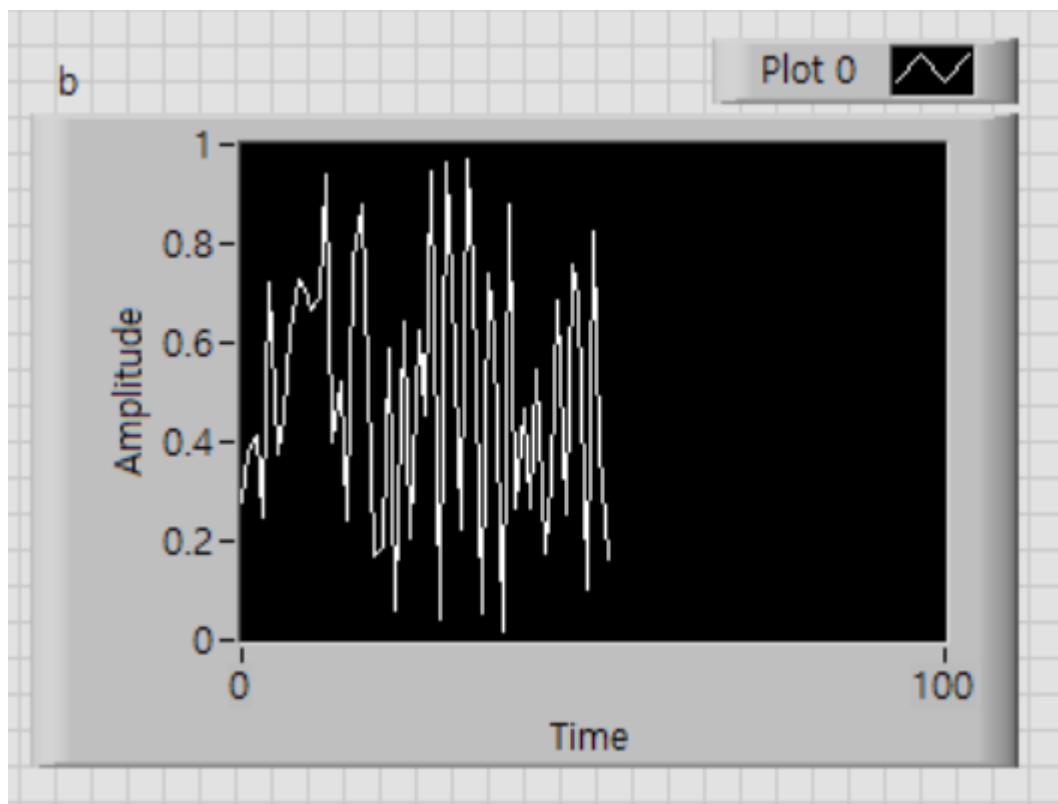
- 
- Houghton Mifflin Books





실습8-8-2) 글로벌 변수 사용법

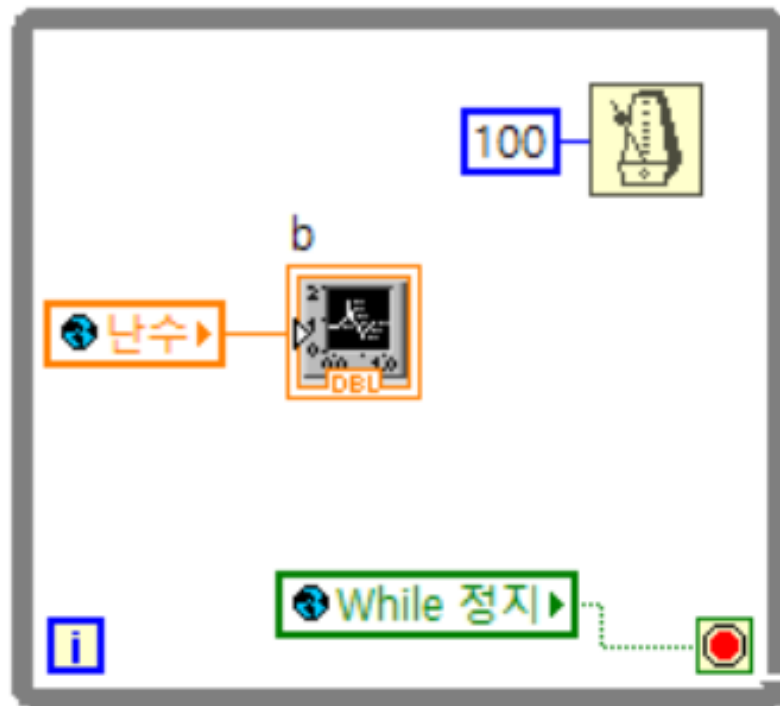
- ❖ 새 **VI**에 그림과 같이 프런트패널과 블록다이어그램을 구현하고 '**b.vi**'로 저장한다.





실습8-8-2) 글로벌 변수 사용법

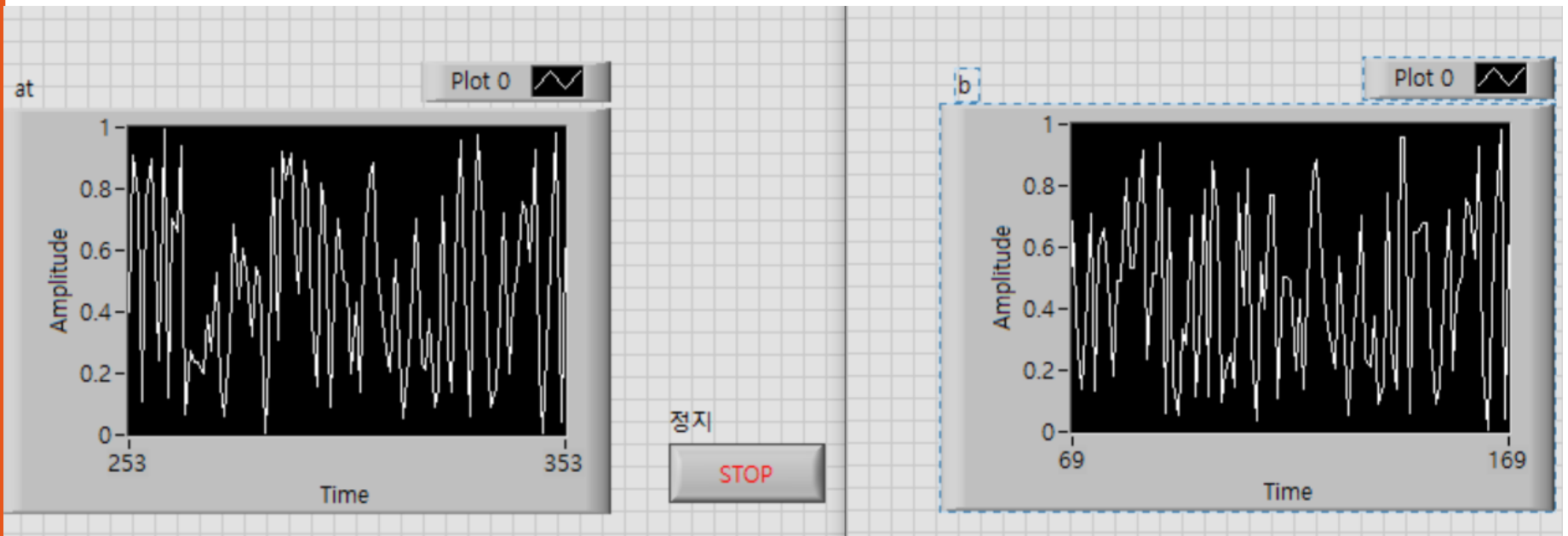
- ❖ 즉 글로벌 변수에 저장된 값을 읽어온다





실습8-8-2) 글로벌 변수 사용법

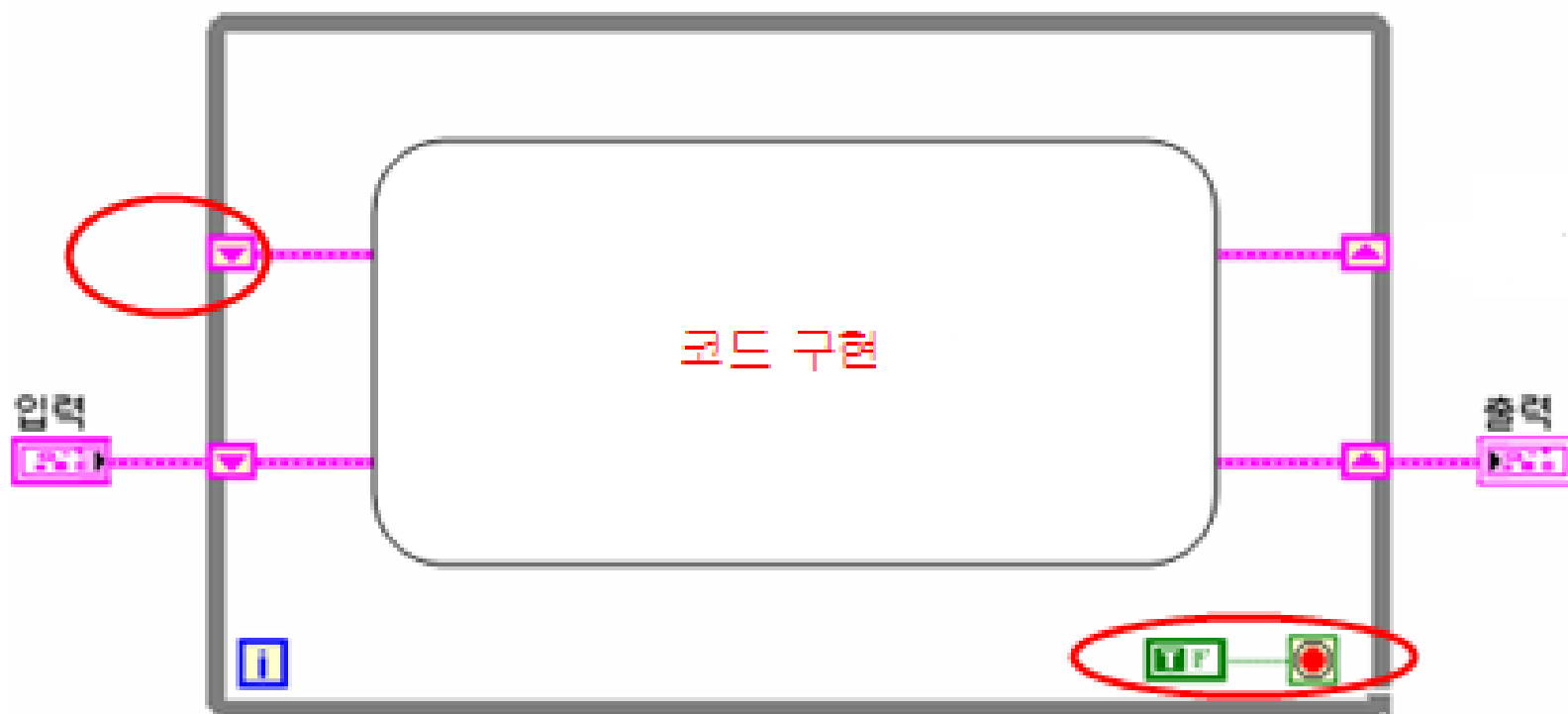
- ❖ 두 **VI**를 실행하여 ‘a’와 ‘b’는 같은 값이 디스플레이되고 ‘정지’ 버튼이 눌러지면 두 **VI**가 동시에 정지된다.





멀티 루프 간의 실시간 데이터 공유

- ❖ 기능적 글로벌 변수는 이름 그대로 글로벌 변수의 기능을 하는 **VI**이다.
- ❖ 구조는 그림과 같다.





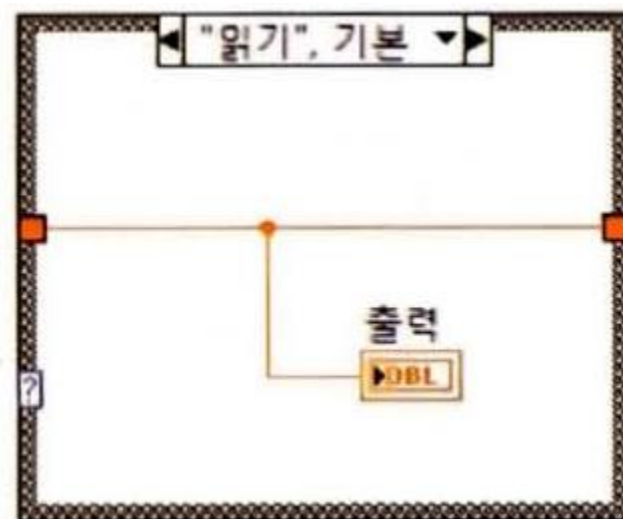
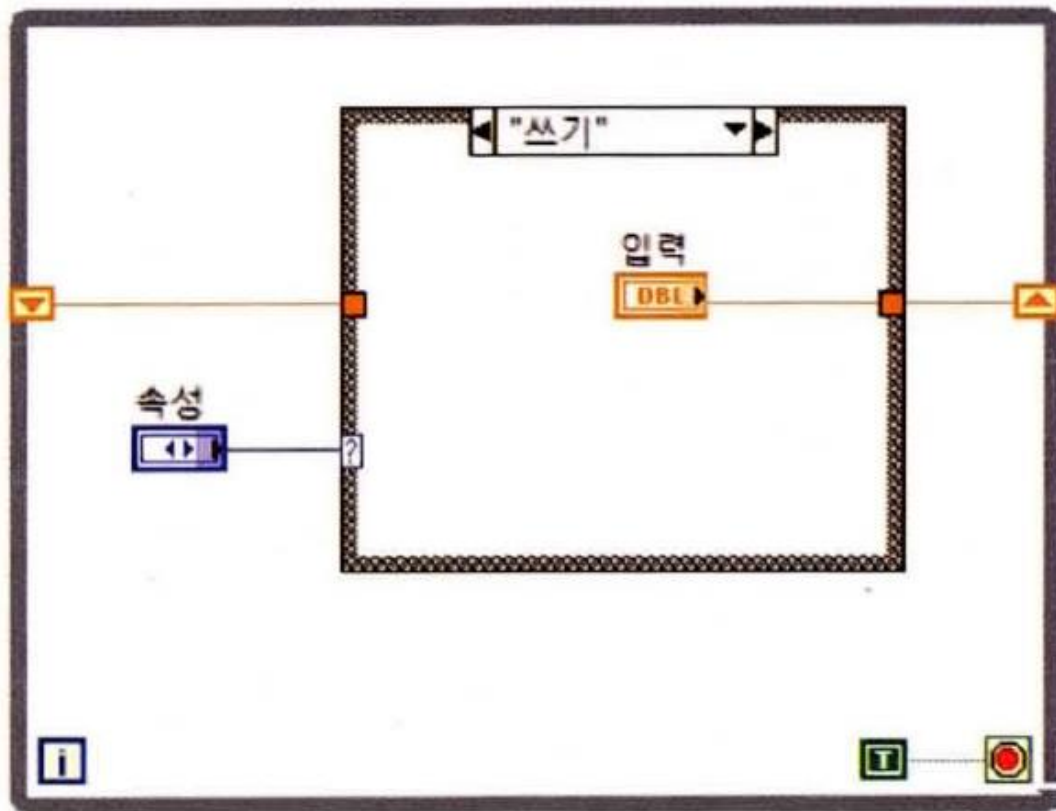
멀티 루프 간의 실시간 데이터 공유

- ❖ 특징은 **While** 루프의 조건 터미널에 참 상수가 연결되어 있고 초기화되지 않은 시프트 레지스터를 가지고 있다면 기능적 글로벌 변수이다.
- ❖ 조건 터미널에 참 상수가 연결되어 있으므로 **While** 루프는 한 번만 실행되고 끝나며 시프트 레지스터가 초기화되지 않았으므로 마지막 값을 저장했다가 다음 번 실행 때 그 값을 사용할 수 있다.
- ❖ 즉 저장 기능이 가능하다



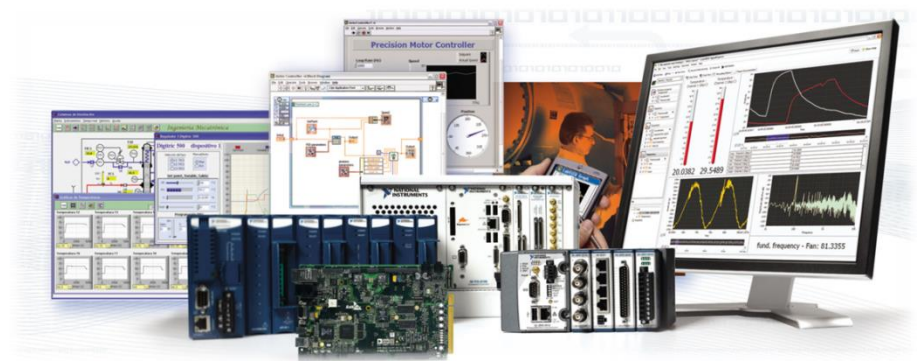
멀티 루프 간의 실시간 데이터 공유

- ❖ 그림은 기능적 글로벌 변수의 예이다.
- ❖ 변수는 읽기 / 쓰기 속성이 가능해야 하기 때문에 이 기능을 케이스 구조를 사용하여 분리한다.



실습8-8-3)

기능적 글로벌 변수 사용법



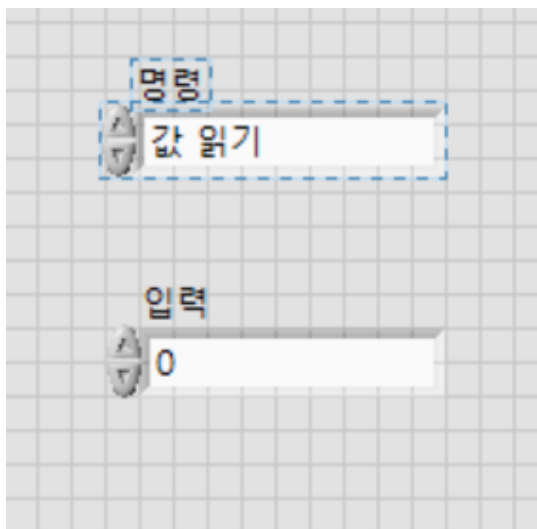


실습8-8-3) 기능적 글로벌 변수 사용법

- ❖ 새 **VI**의 프런트패널에 열거형 컨트롤과 숫자형 컨트롤을 배치
- ❖ 열거형 컨트롤의 아이템을 그림과 같이 추가



Enum Properties: 명령



Enum Properties: 명령

Appearance | Data Type | Display Format | **Edit Items** | Documentation

Items	Values
값 읽기	0
값 쓰기	1

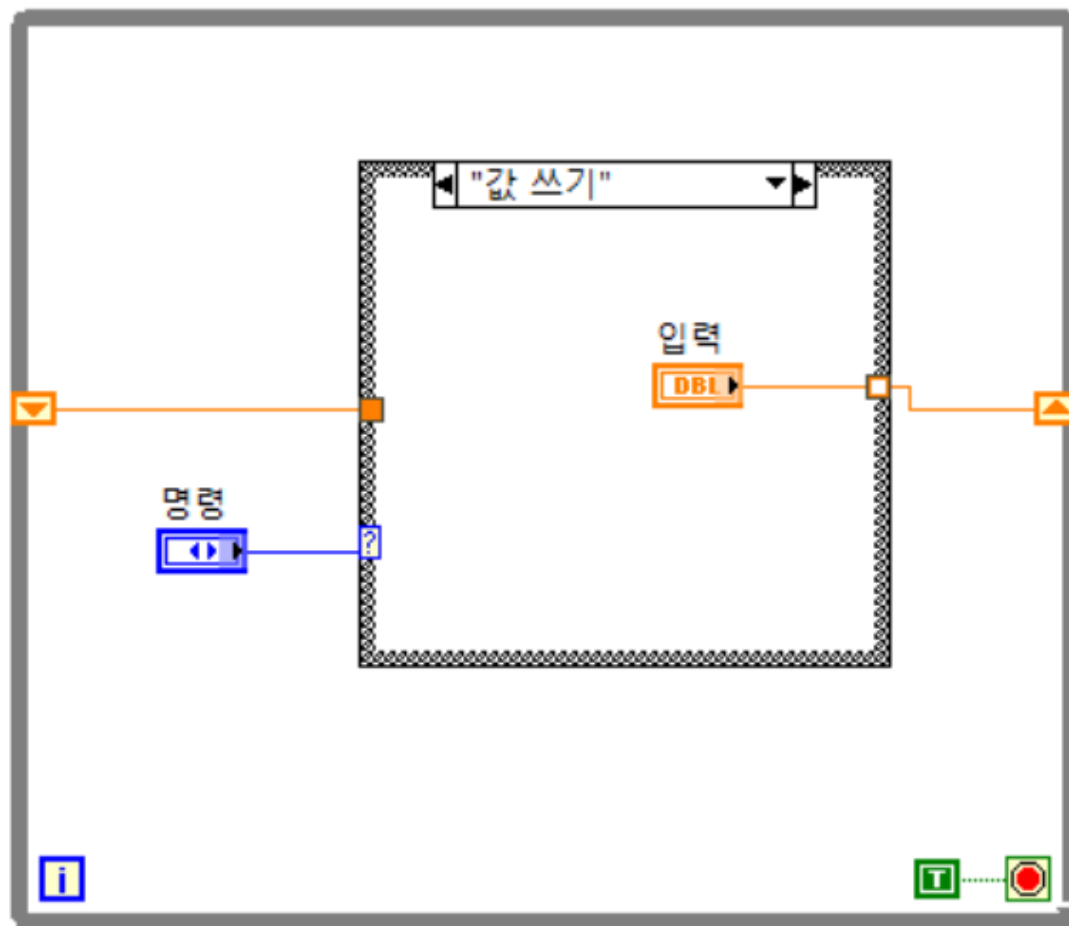
☐ Allow undefined values at run time

Buttons: Insert, Delete, Move Up, Move Down, Disable Item



실습8-8-3) 기능적 글로벌 변수 사용법

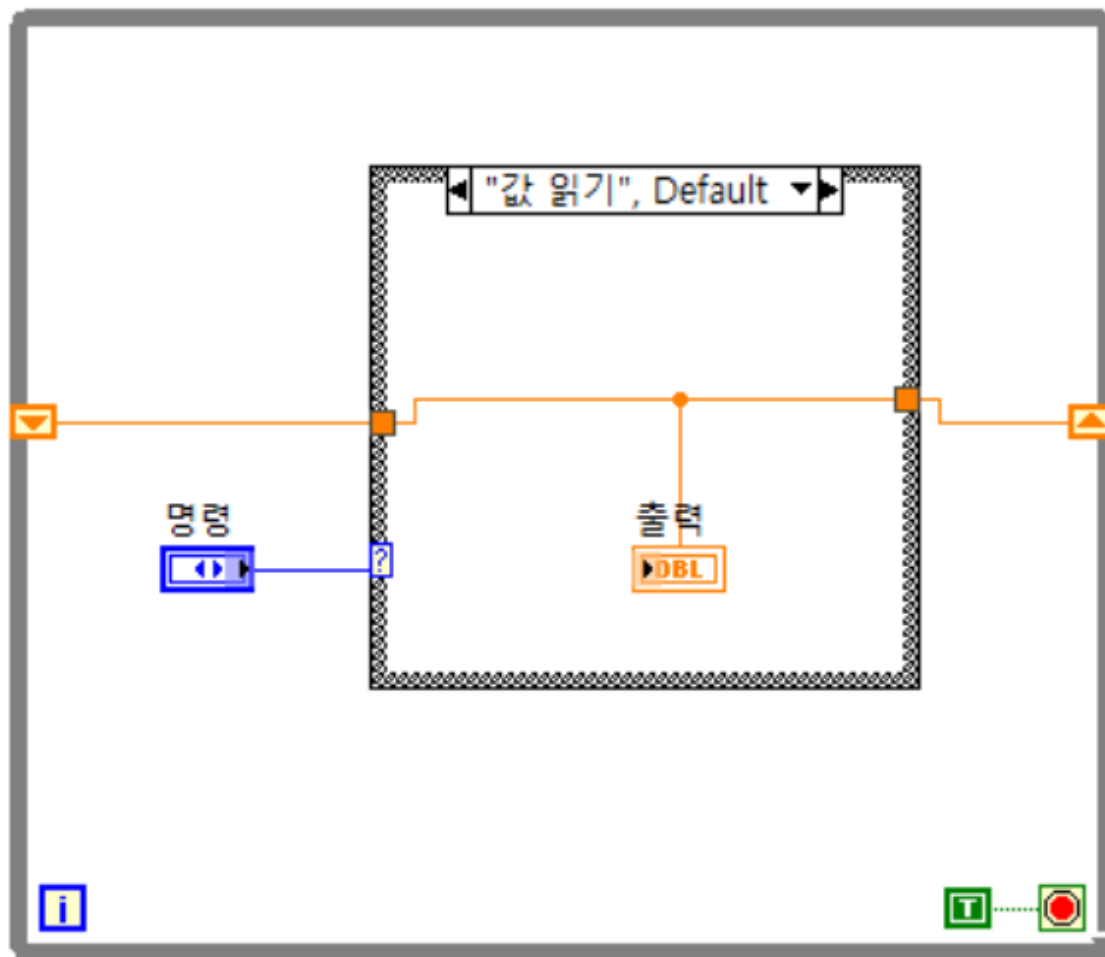
- ❖ 블록 다이어그램에서 '명령' 터미널을 케이스 구조에 연결한 다음 그림과 같이 구성





실습8-8-3) 기능적 글로벌 변수 사용법

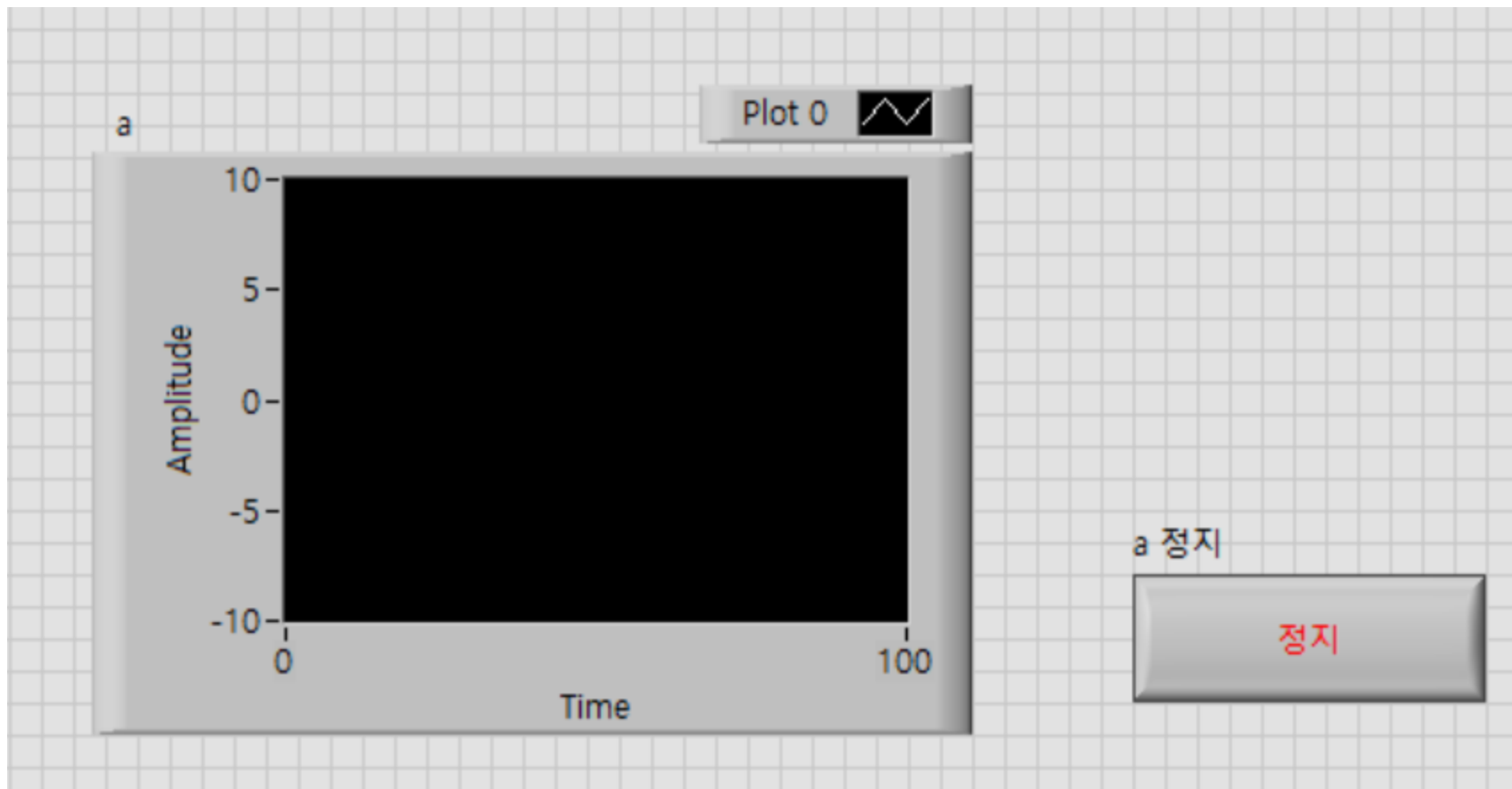
- ❖ 블록 다이어그램에서 '값 읽기' 케이스를 그림과 같이 수정하고 저장한 뒤 **VI**를 닫는다.





실습8-8-3) 기능적 글로벌 변수 사용법

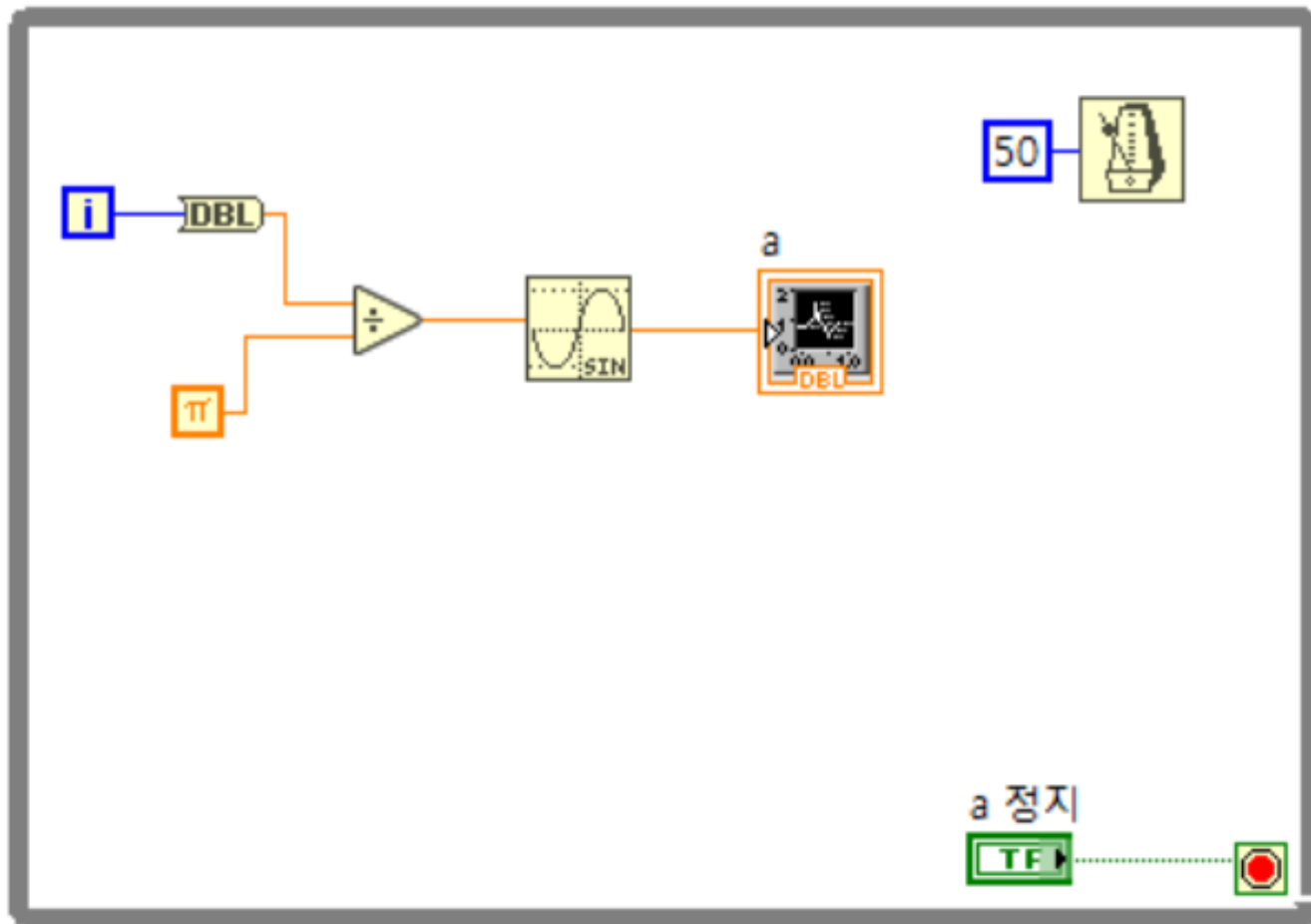
- ❖ 새 **VI**를 생성하고 프런트 패널에 웨이브폼 차트와 정지버튼을 배치한다.





실습8-8-3) 기능적 글로벌 변수 사용법

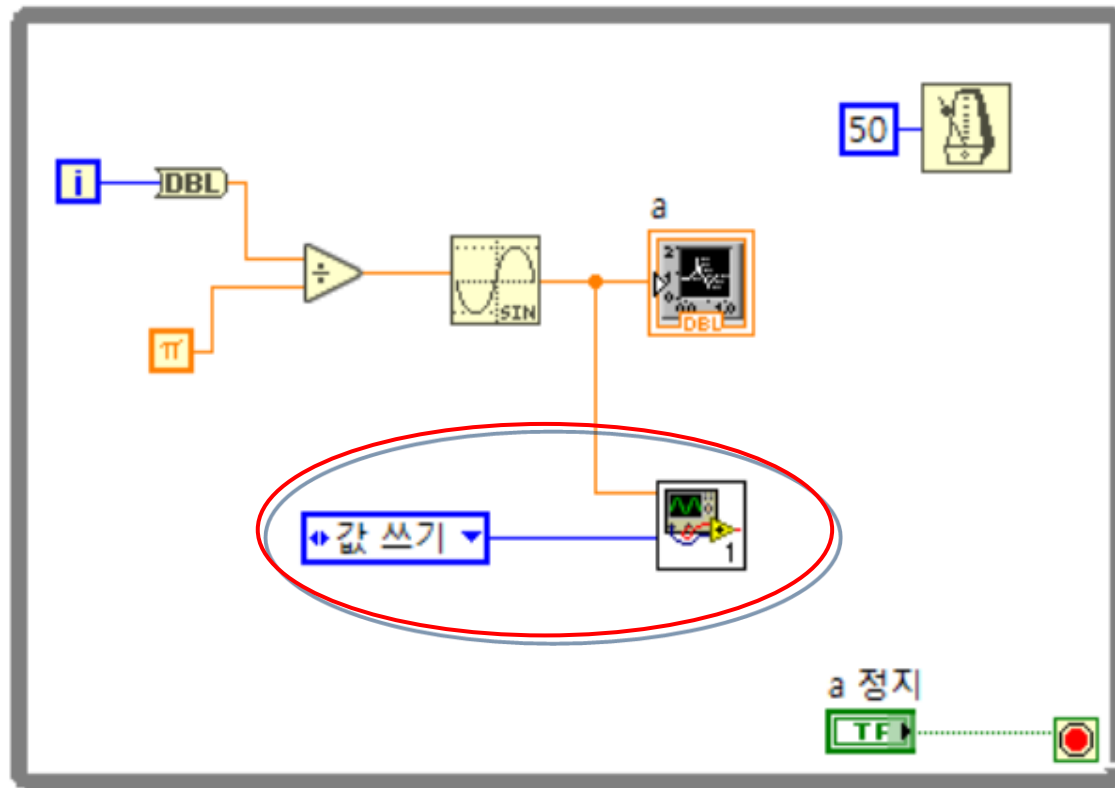
❖ 블록 다이어그램에서 그림과 같이 구성한다.





실습8-8-3) 기능적 글로벌 변수 사용법

- ❖ 블록 다이어그램에 앞에서 만든 기능적 글로벌 변수 **.VI**를 추가하여 그림과 같이 연결한다.
- ❖ 이때 기능적 글로벌 변수 **.VI**의 명령 파라미터에서 바로가기 메뉴 > 생성 > 상수를 선택해 '값 쓰기'로 설정한다.





실습8-8-3) 기능적 글로벌 변수 사용법

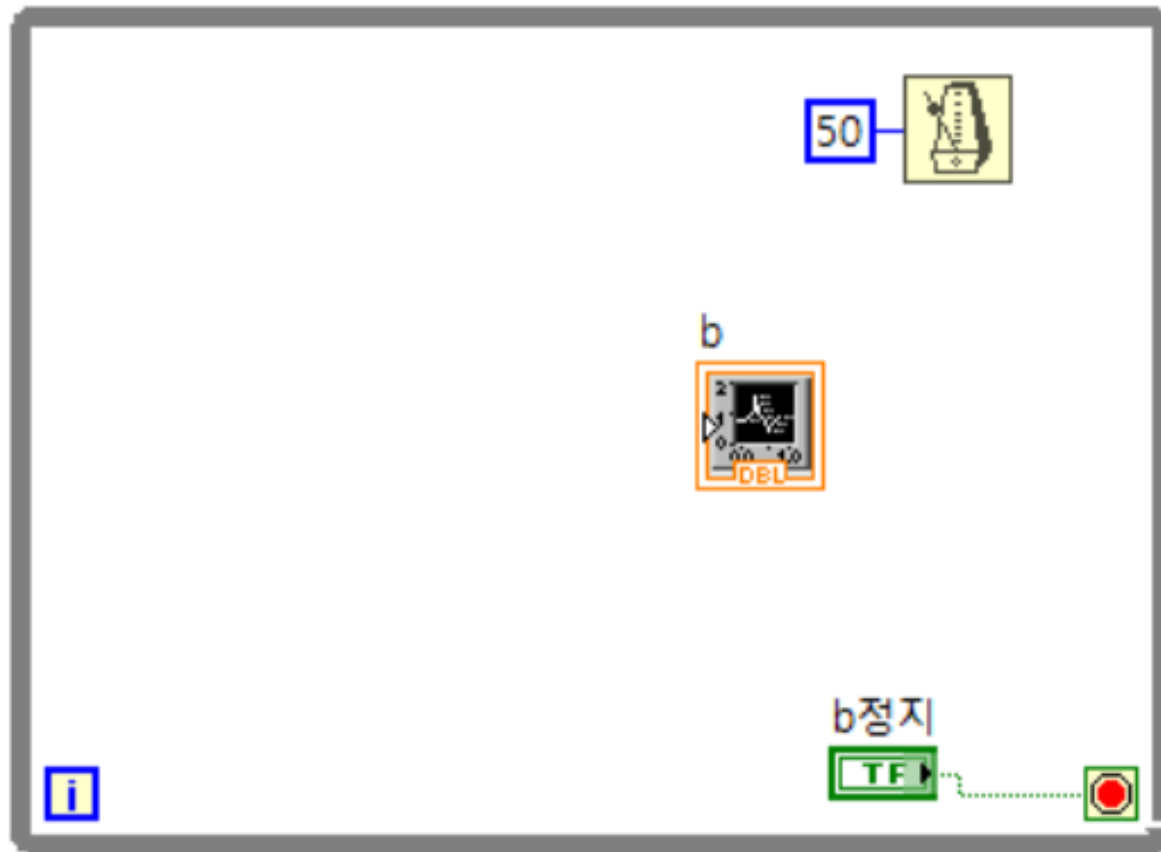
- ❖ 앞에서 만든 기능적 글로벌 변수-**a.vi**와 동일하게 프런트 패널을 구성한다.





실습8-8-3) 기능적 글로벌 변수 사용법

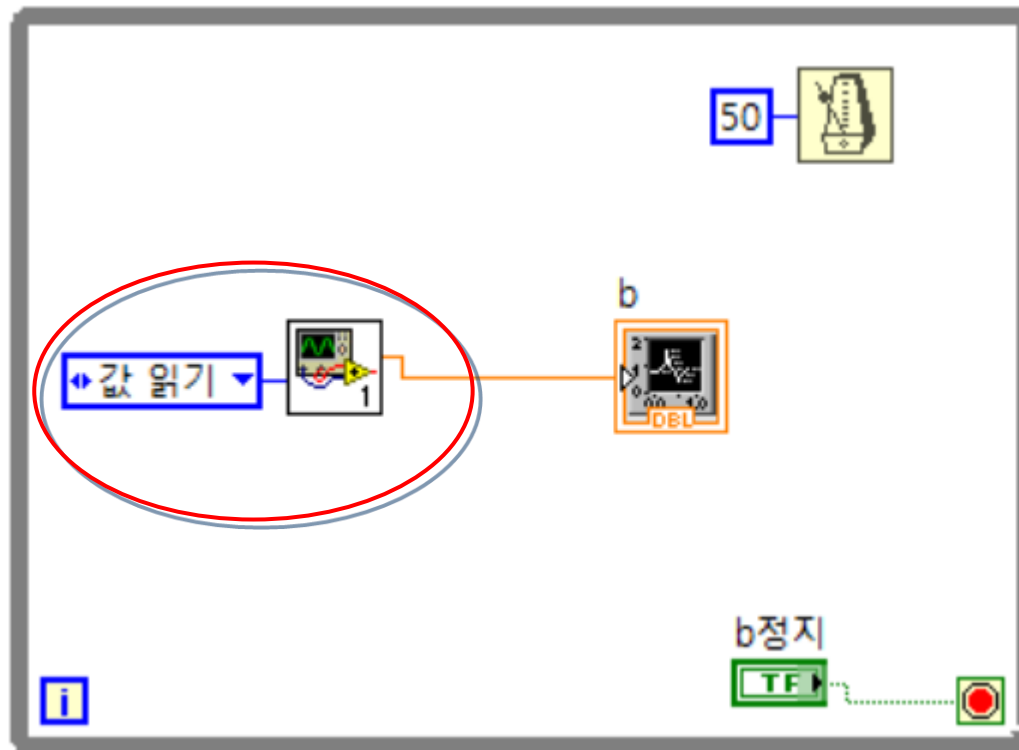
❖ 블록 다이어그램을 그림과 같이 구성한다.





실습8-8-3) 기능적 글로벌 변수 사용법

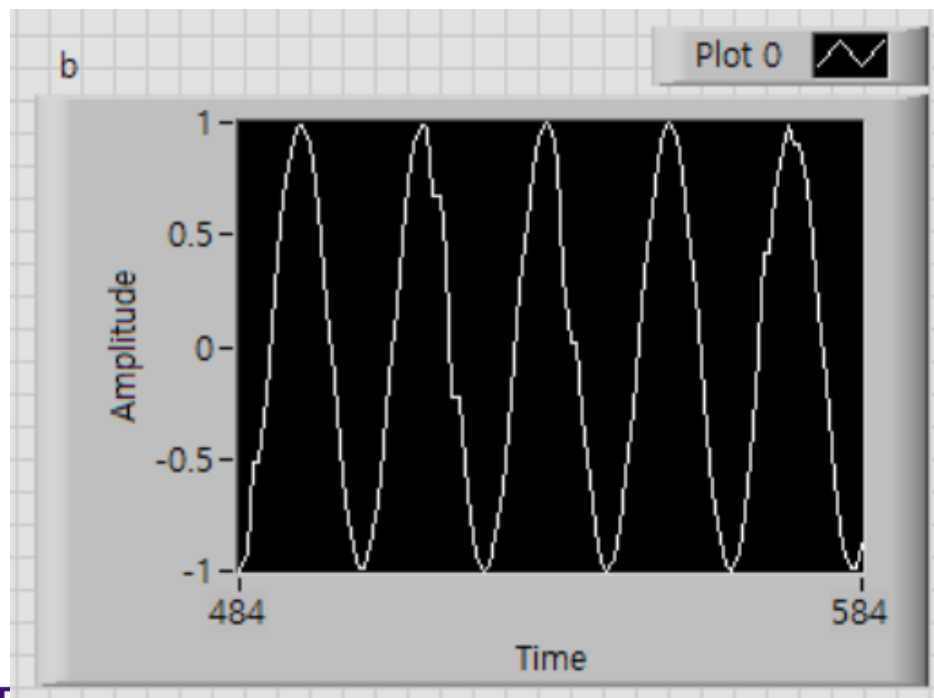
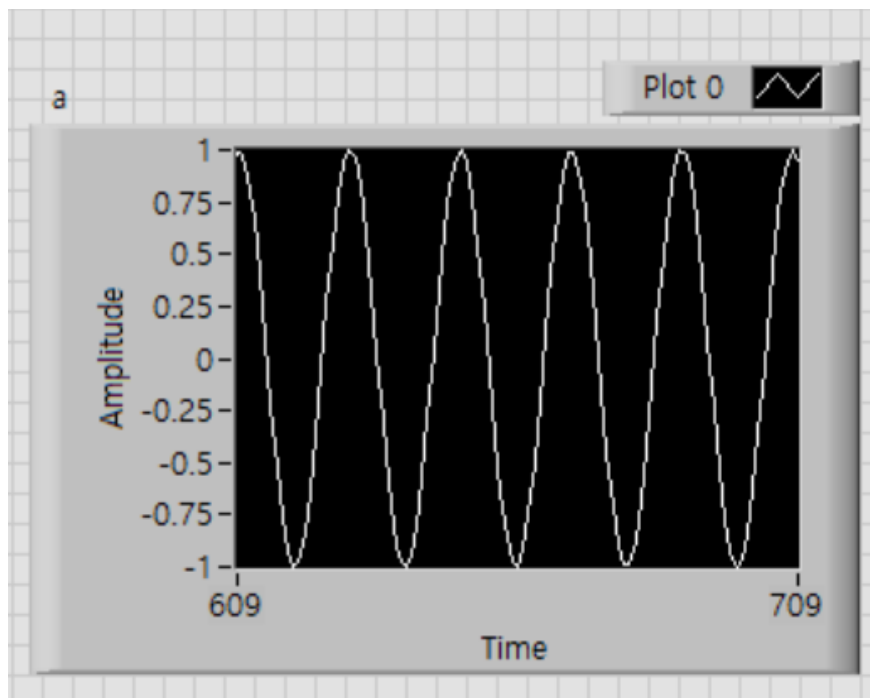
- ❖ 블록 다이어그램에 기능적 글로벌 변수.**VI**를 추가하여 그림과 같이 연결한다.
- ❖ 이때 기능적 글로벌 변수.**VI**의 명령 파라미터에서 바로가기메뉴 > 생성 > 상수를 선택해 '값 읽기'로 설정한다.





실습8-8-3) 기능적 글로벌 변수 사용법

- ❖ 두 **VI**를 각각 실행시키면 ‘기능적 글로벌 변수-**a.VI**’의 ‘**a**’ 웨이브폼 차트에 있는 사인파형이 기능적 글로벌 변수를 통해 ‘기능적 글로벌 변수-**b.vi**’의 ‘**b**’ 웨이브폼 차트에 전달되어 같은 데이터가 디스플레이 되는 것을 확인할 수 있다.





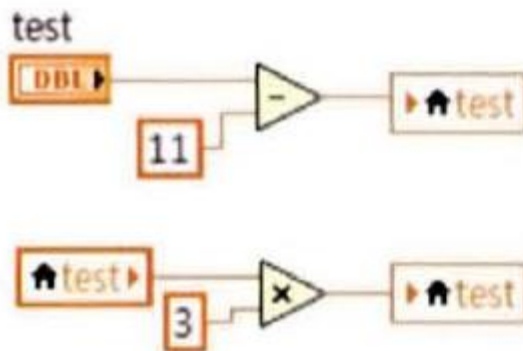
기능적 글로벌 변수

- ❖ 기능적 글로벌 변수를 제외하고 나머지 변수들은 꼭 필요할 때만 사용하는 것이 좋다.
- ❖ 그 이유는 **LabVIEW**의 데이터 흐름 패러다임이 깨지고 경합 조건 (**race condition**)이 발생하여 화이어를 통해 데이터를 전달할 때보다 속도 등의 효율이 떨어지기 때문
- ❖ 경합 조건은 어떤 노드가 맨처음 시작되었는지 또는 **test** 값이 언제 읽혀졌는지에 따라서 다른 결과값을 가져올 수 있다.
- ❖ 이러한 현상을 **경합 조건** 이라고 합니다.



기능적 글로벌 변수

- ❖ 코드에 경합 조건이 있어도 테스트 중 수천 번 같은 결과를 반환하다가 어떤 때는 다른 결과를 반환한다거나 해서 경합 조건은 디버깅이 어렵다.
- ❖ 따라서 변수는 꼭 필요한 곳에서만 사용하고 사용할 때도 경합 조건이 발생하지 않도록 주의해서 사용해야 한다.



$\text{test} = 3 * (\text{test} - 11)$

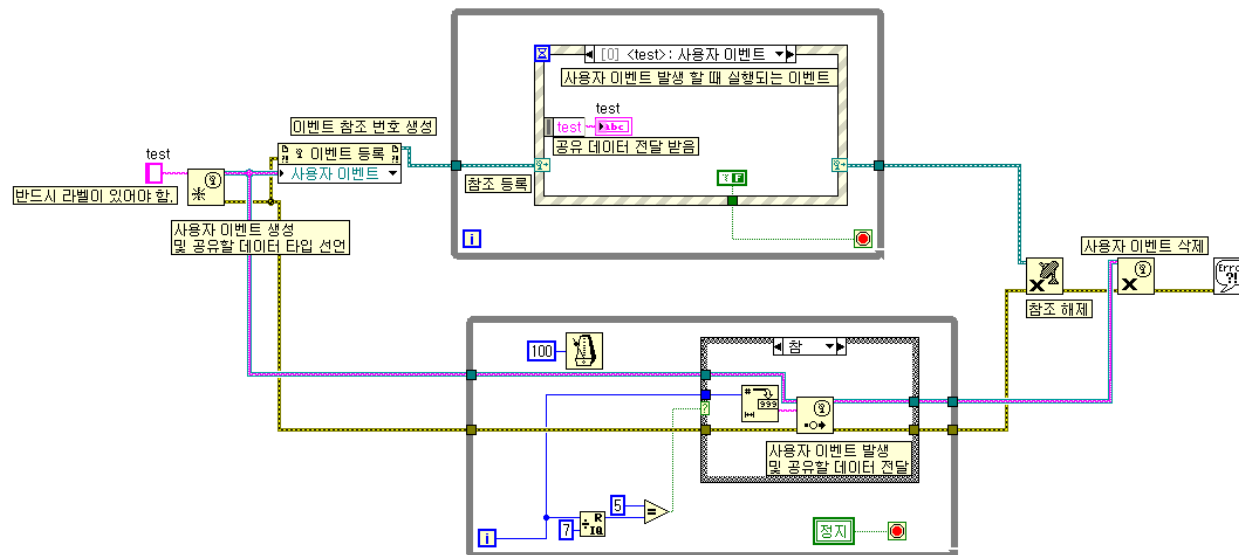
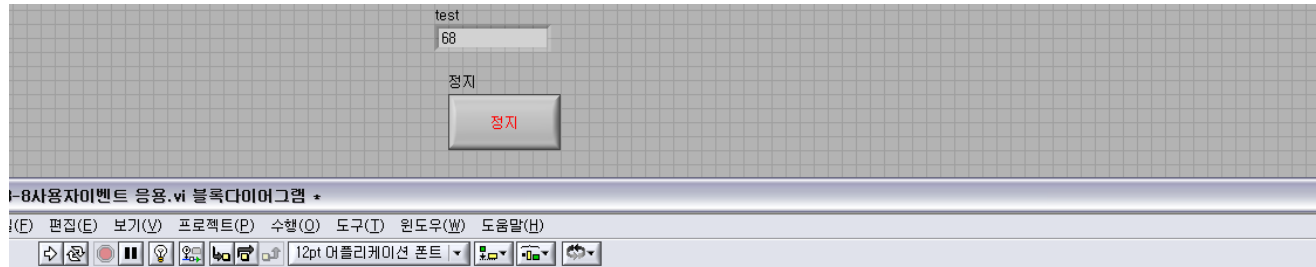
$\text{test} = \text{test} - 11$

$\text{test} = \text{test} * 3$

$\text{test} = 3 * \text{test} - 11$



다이나믹 이벤트(=사용자 이벤트)



•다이나믹 이벤트 : 사용자 정의 이벤트임.



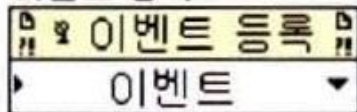
다이나믹 이벤트(사용자 이벤트)

- ❖ 다이나믹 이벤트(사용자 이벤트)는 이름처럼 사용자가 임의로 이벤트를 정의해서 사용할 수 있다.
- ❖ 부가적으로 다이나믹 이벤트(사용자 이벤트)는 루프들 간에 실시간으로 데이터를 전달하는 기능도 가지고 있다.

사용자 이벤트 생성



이벤트 등록



사용자 이벤트 발생



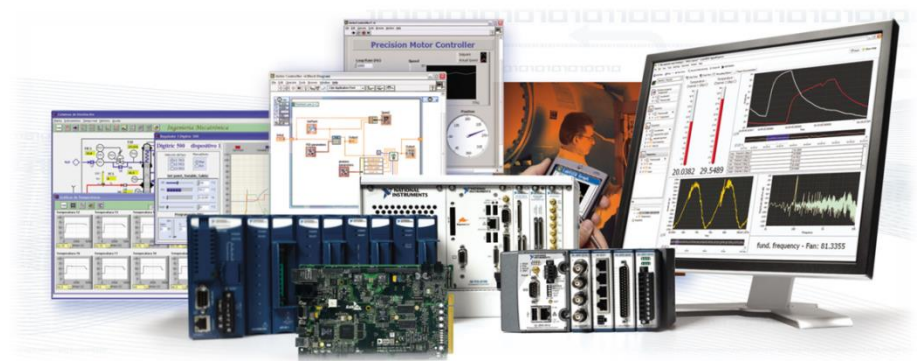
이벤트 등록 해제



사용자 이벤트 삭제



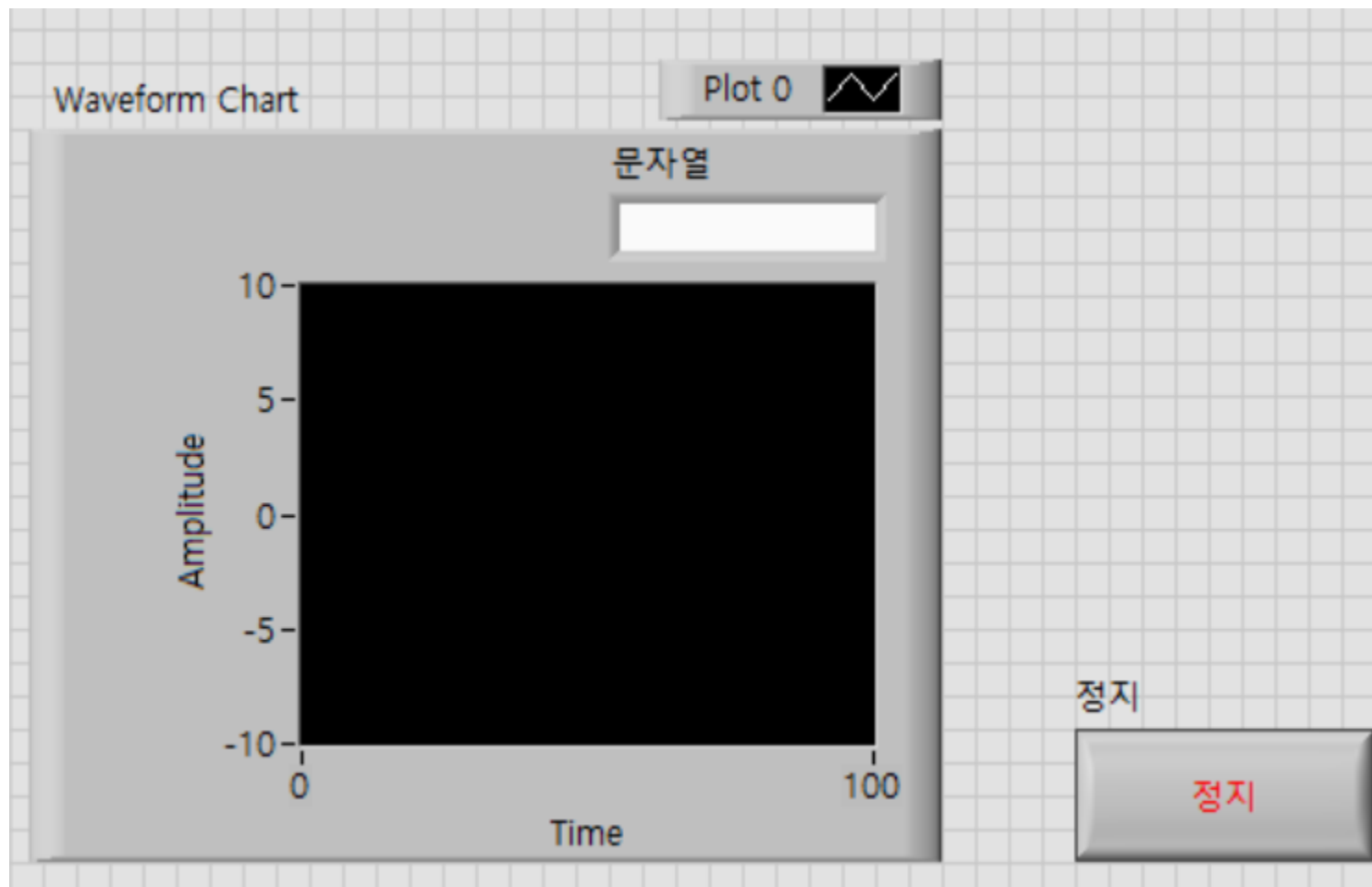
실습8-8-4) 다이나믹 이벤트 사용법





실습8-8-4) 다이나믹 이벤트 사용법

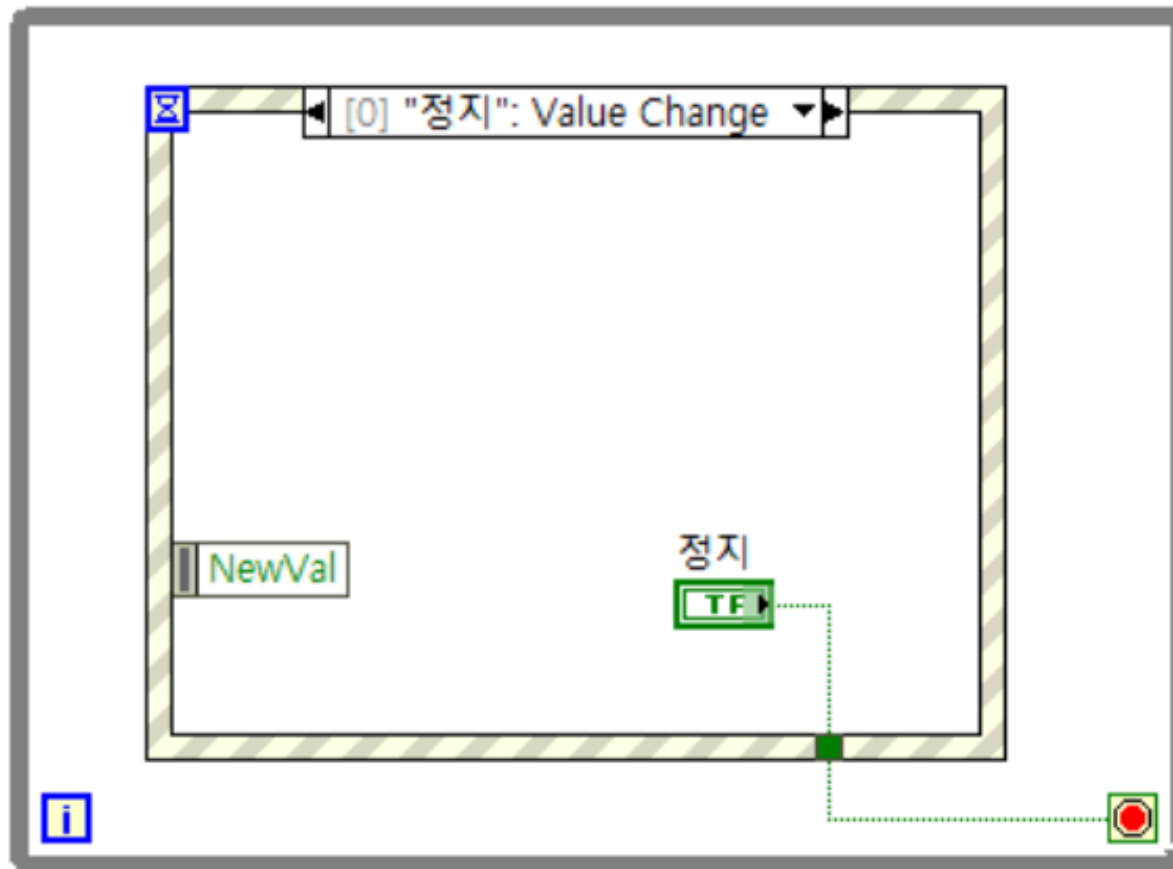
- ❖ 새 **VI**의 프런트패널에 웨이브폼 차트, 정지 버튼, 문자열 인디케이터를 배치





실습8-8-4) 다이나믹 이벤트 사용법

❖ 블록 다이어그램에서 그림과 같이 구성



문자열



Waveform Chart





실습8-8-4) 다이나믹 이벤트 사용법

❖ 블록 다이어그램에서 그림과 같이 구성

Visible Items ▶

Help

Examples

Description and Tip...

Breakpoint ▶

Event Inspector Window

Structures Palette ▶

✓ Auto Grow

Exclude from Diagram Cleanup

Remove Event Structure

Edit Events Handled by This Case...

Add Event Case...

Duplicate Event Case...

Delete This Event Case

Show Dynamic Event Terminals

Find Control

Remove and Rewire

Properties

Edit Events

Event case

[0] "정지": Value Change

Event Specifiers

Event Source	Event
정지	Value Change

Event Sources

Filter

<Application>
<This VI>
Dynamic
Panes
Pane
Splitters
Controls
Waveform Chart
정지
문자열

Events

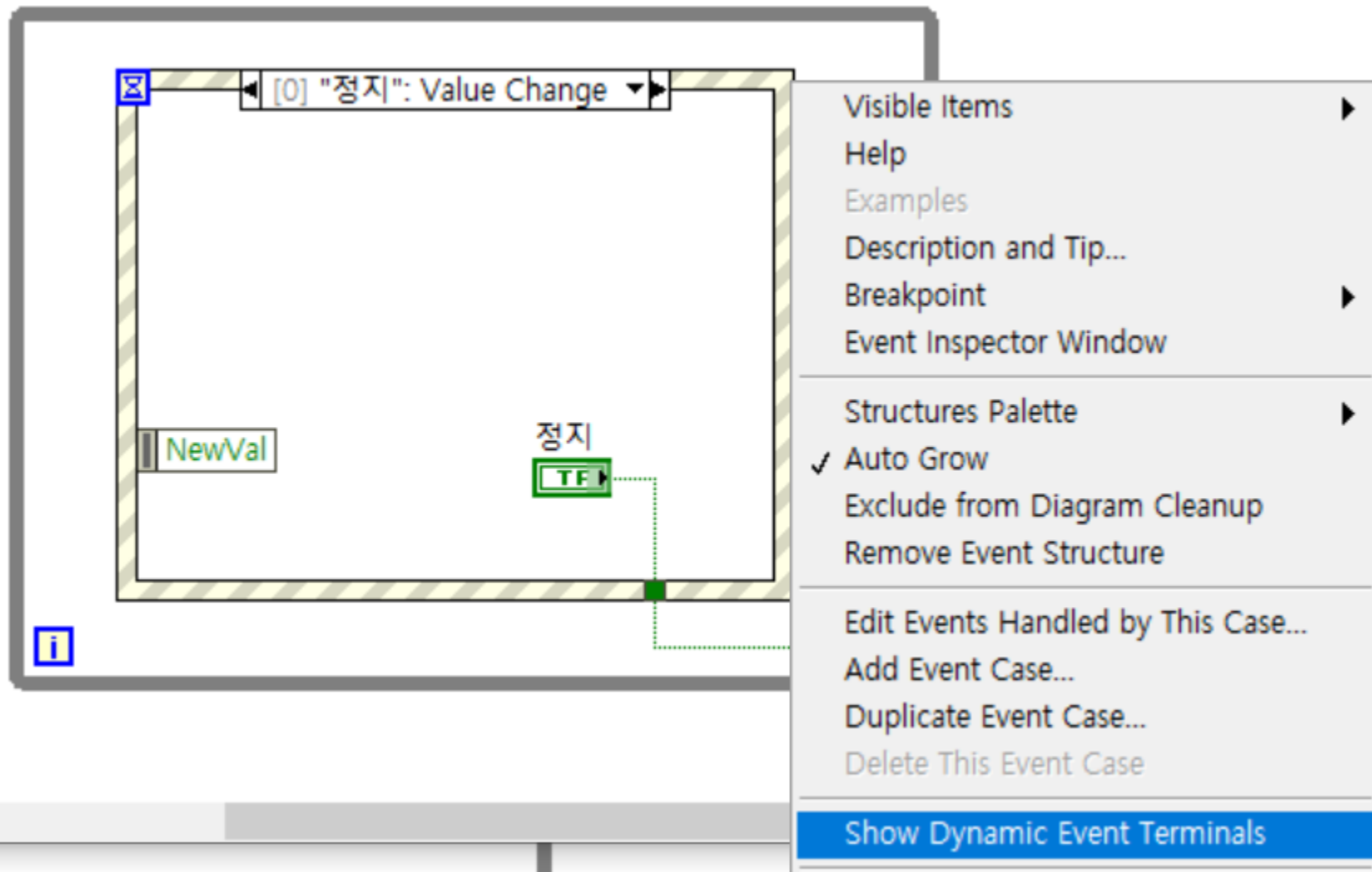
Filter

+	Key
+	Mouse
+	Drag
+	Shortcut Menu
→	Value Change



실습8-8-4) 다이나믹 이벤트 사용법

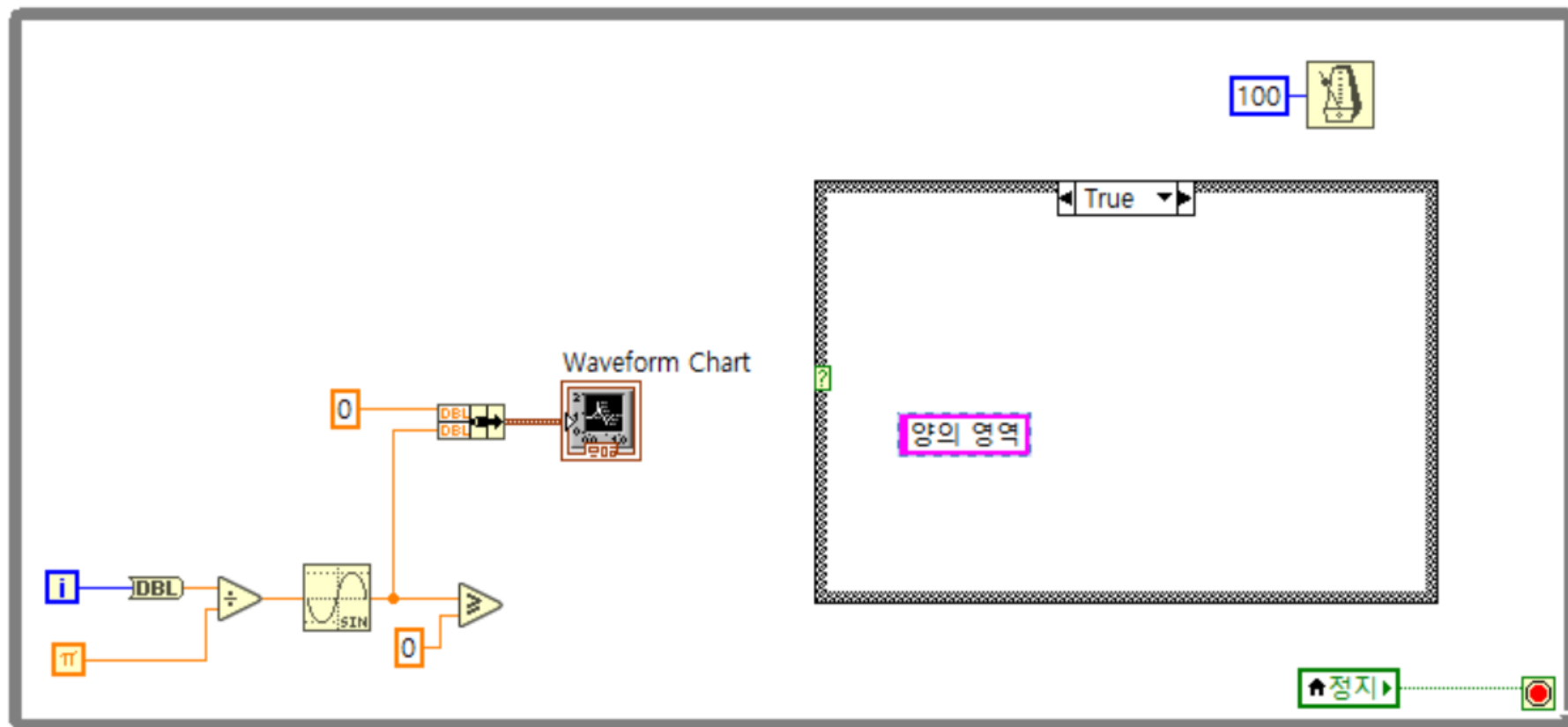
❖ 이벤트 구조에 그림과 같이 다이나믹 이벤트 터미널을 추가





실습8-8-4) 다이나믹 이벤트 사용법

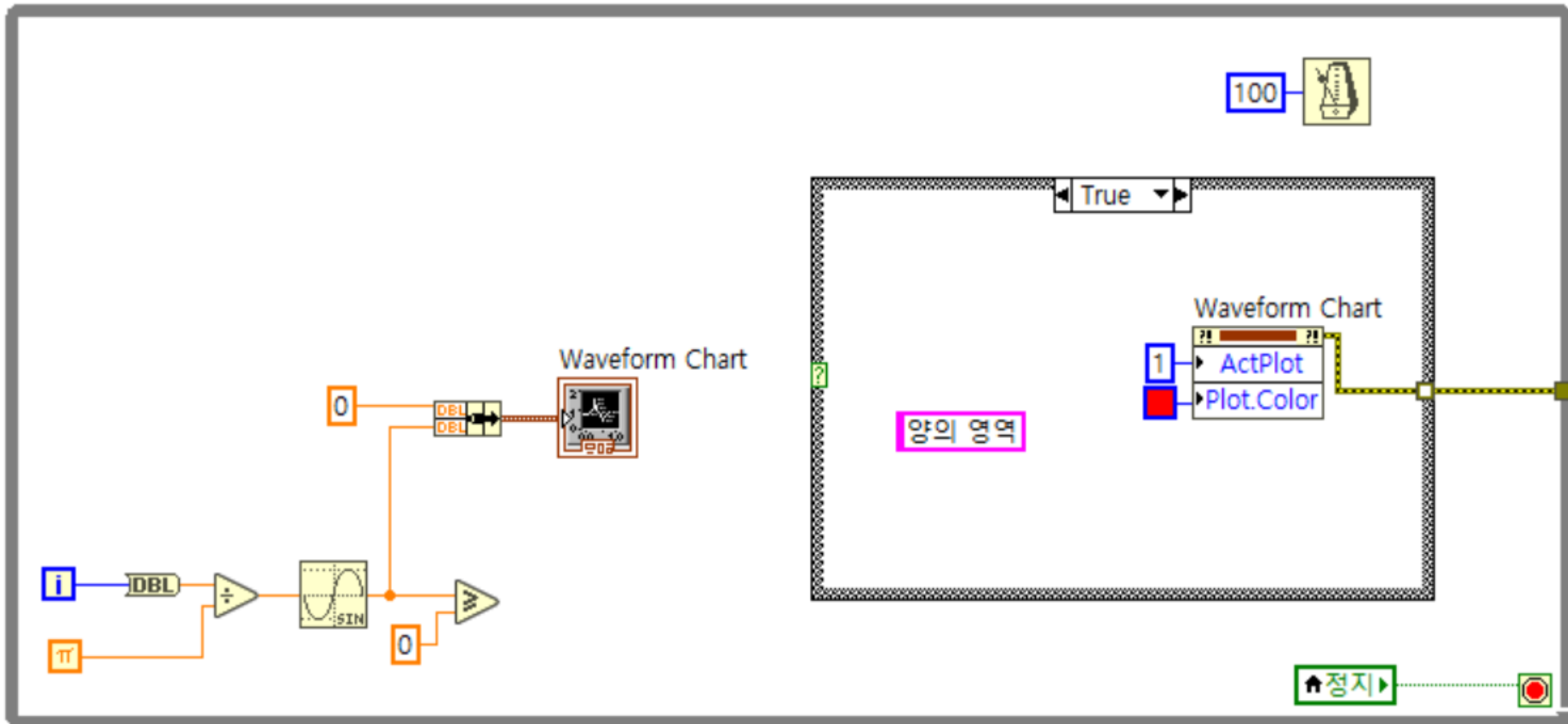
- ❖ 정지 버튼의 지역 변수를 생성하고 그림과 같이 블록 다이어그램을 추가
- ❖ 정지 버튼의 지역 변수를 읽기로 변환





실습8-8-4) 다이나믹 이벤트 사용법

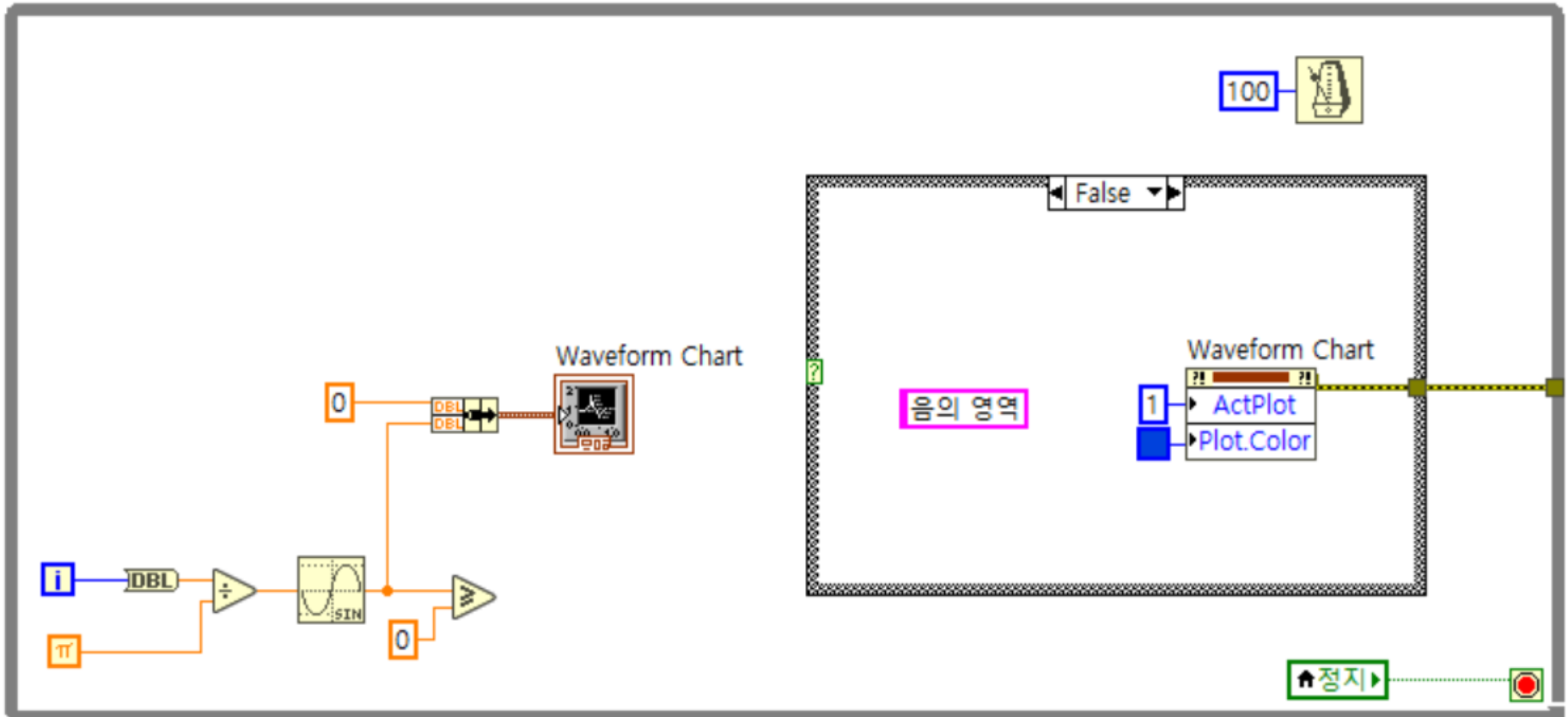
- ❖ 웨이브폼 차트의 프로퍼티 노드(**Active Plot**) 생성
- ❖ 속성 하나 추가 : **Plot Color**
- ❖ 프로퍼티 노드를 모두 쓰기로 변환





실습8-8-4) 다이나믹 이벤트 사용법

- ❖ 케이스 구조의 케이스를 **False**로 바꾸고 그림처럼 블록 다이어그램을 구성





실습8-8-4) 다이나믹 이벤트 사용법

- ❖ 블록 다이어그램에 그림과 같은 시퀀스 구조를 추가
- ❖ 정지 버튼의 지역변수를 하나 더 추가



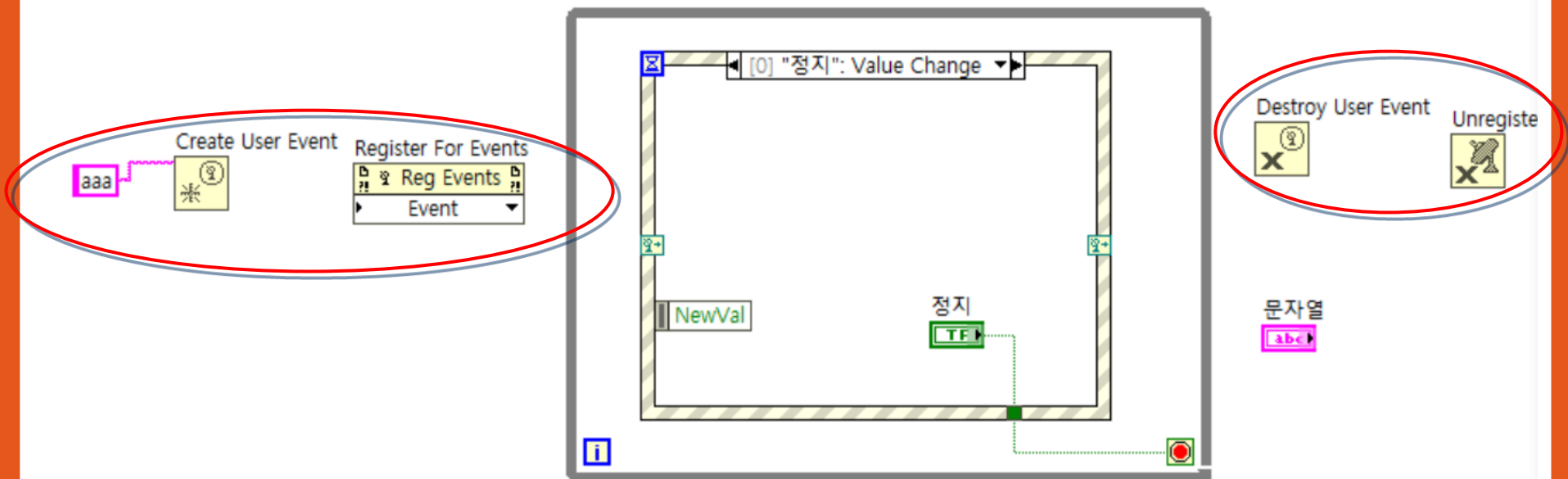
Simple Error Handler.vi





실습8-8-4) 다이나믹 이벤트 사용법

- ❖ 블록다이어그램을 그림과 같이 사용자 이벤트 생성.**VI**, 이벤트 등록.**VI**, 이벤트 등록 해제.**vi**, 사용자 이벤트 삭제.**VI**를 추가한다.
- ❖ 사용자 이벤트 생성.**vi**의 사용자 이벤트 데이터 타입 파라미터에 문자열 상수를 연결해 주고 라벨을 '**aaa**'라고 붙여준다.





실습8-8-4) 다이나믹 이벤트 사용법

- [illegible]



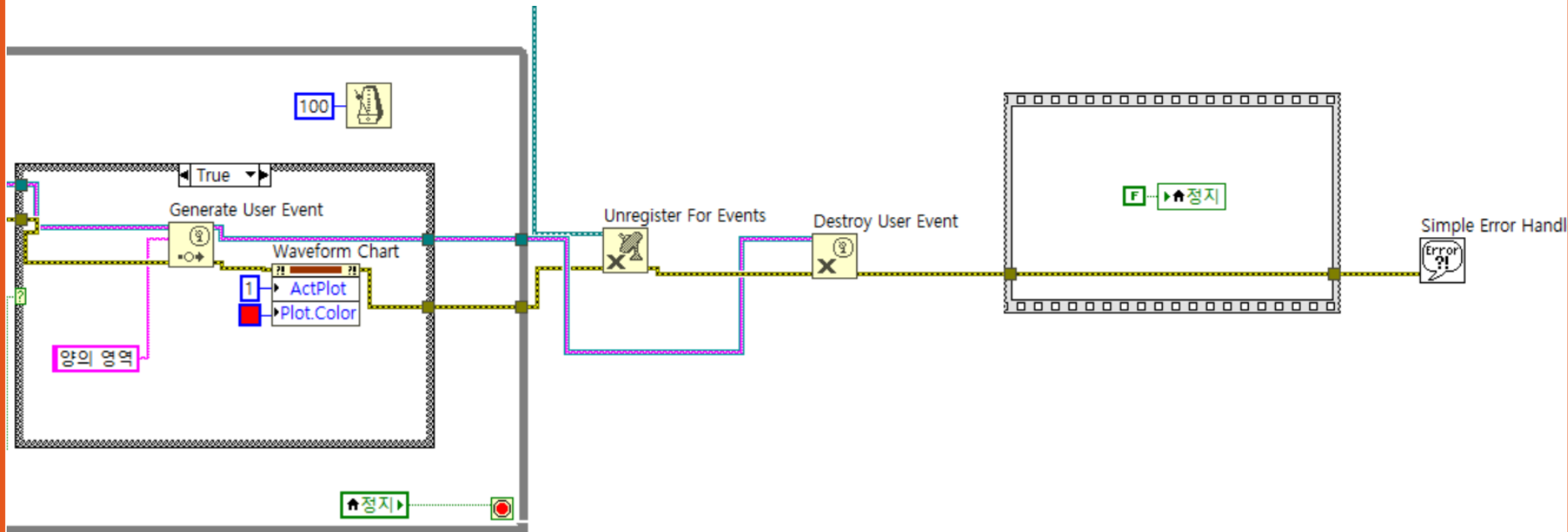
실습8-8-4) 다이나믹 이벤트 사용법

-



실습8-8-4) 다이나믹 이벤트 사용법


- ❖ 그림과 같이 와이어링한다.
- ❖ 참 케이스도 거짓 케이스와 같은 방법으로 와이어링한다





실습8-8-4) 다이나믹 이벤트 사용법

- ❖ 이벤트 구조의 바로가기메뉴 > 이벤트 케이스 추가 ... 에서 다이나믹의 (**aaa**): 사용자 이벤트를 선택한 후, 이벤트 데이터 노드를 '**aaa**'로 설정하고 그림과 같이 와이어링하고 수정한다.

 Edit Events

— □

Event case

[1] <aaa>: User Event

Event Specifiers

Event Source	Event
Dynamic: <aaa>	User Event

Event Sources

Filter

- <Application>
- <This VI>
- Dynamic
 - <aaa>: User Event
- Panes
 - Pane
- Splitters
- Controls
 - Waveform Chart
 - 정지
 - 문자열

Events

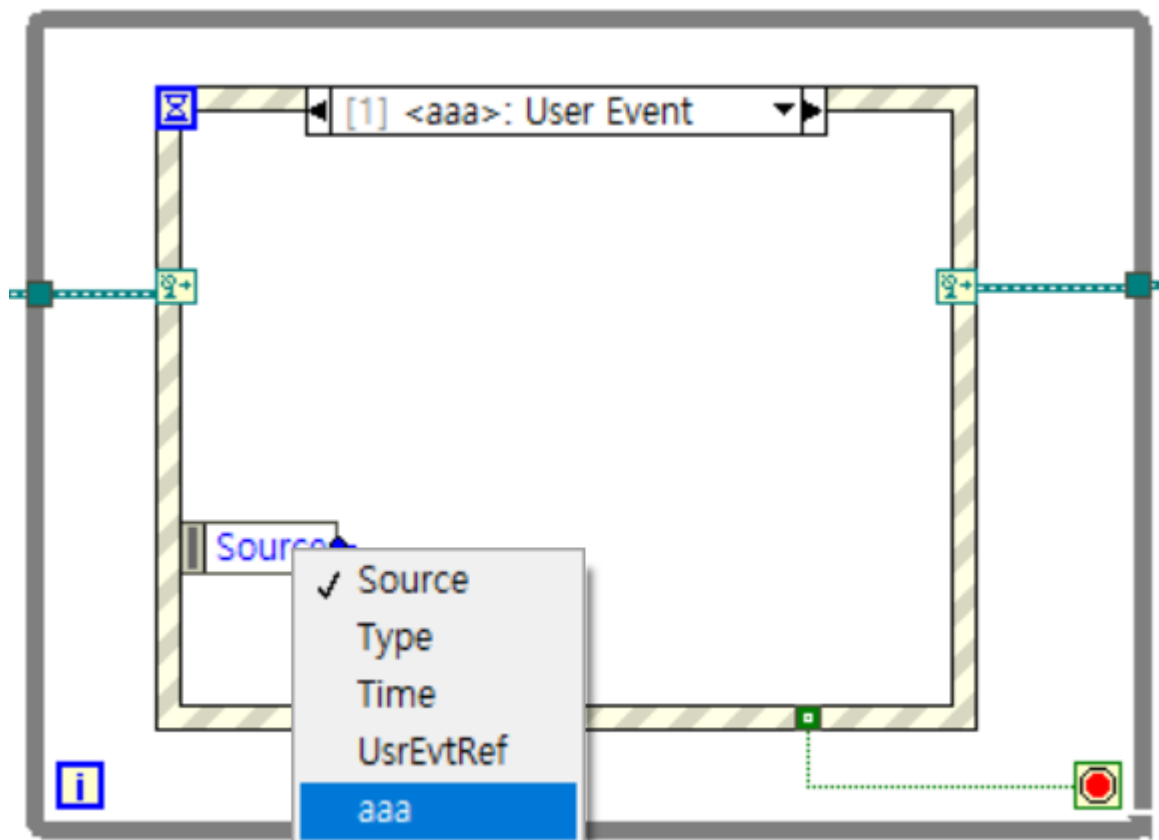
Filter

- User Event



실습8-8-4) 다이나믹 이벤트 사용법

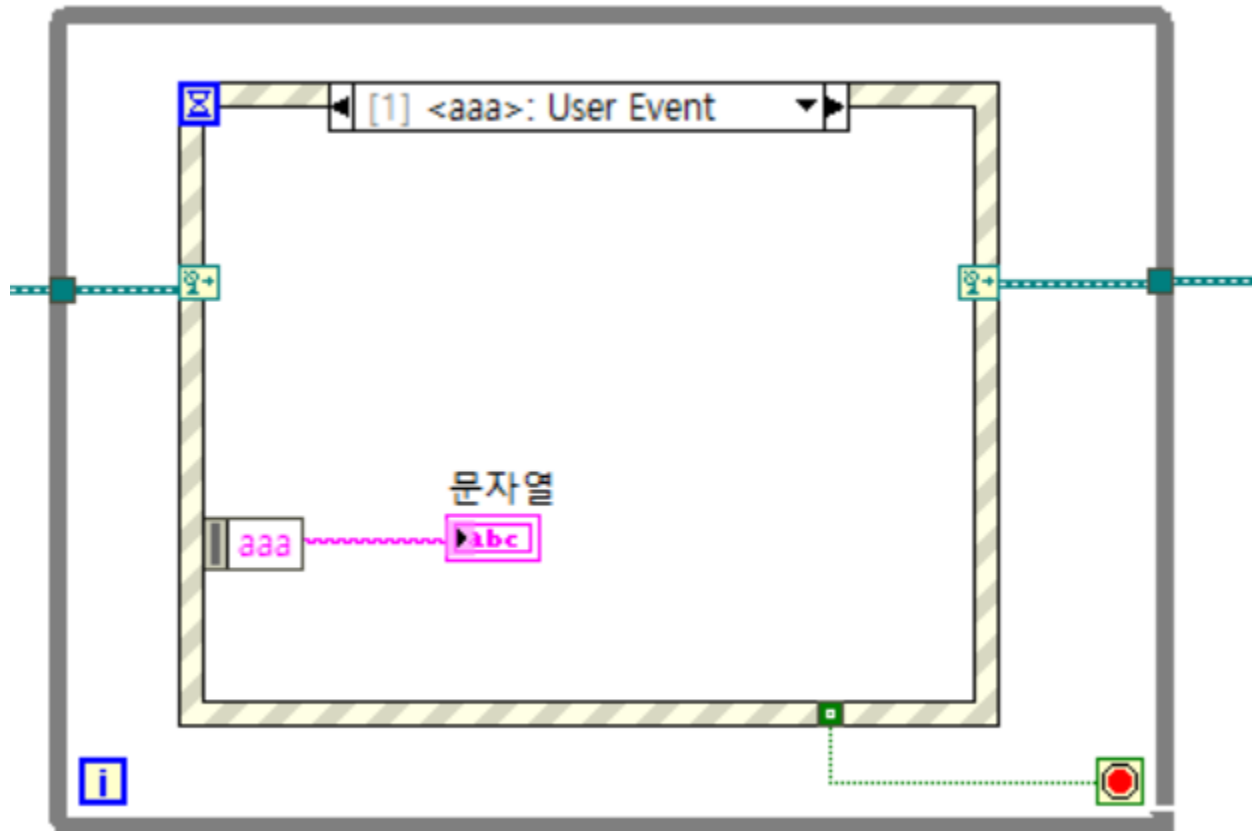
- ❖ 이벤트 구조의 바로가기메뉴 > 이벤트 케이스 추가 ... 에서 다이나믹의 (**aaa**): 사용자 이벤트를 선택한 후, 이벤트 데이터 노드를 '**aaa**'로 설정하고 그림과 같이 와이어링하고 수정한다.





실습8-8-4) 다이나믹 이벤트 사용법

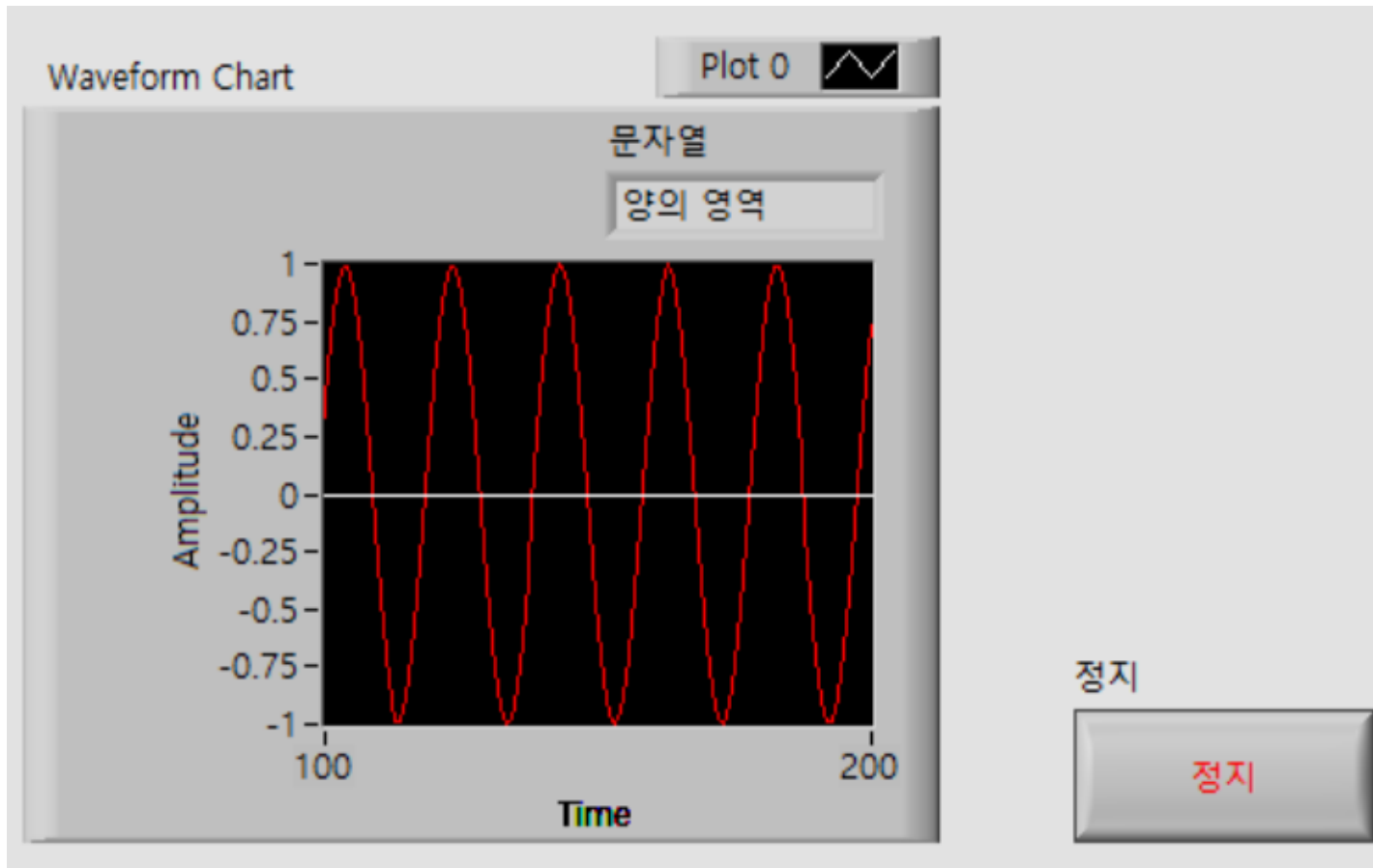
- ❖ 이벤트 데이터 노드를 그림과 같이 문자열 인디케이터와 와이어링한다.





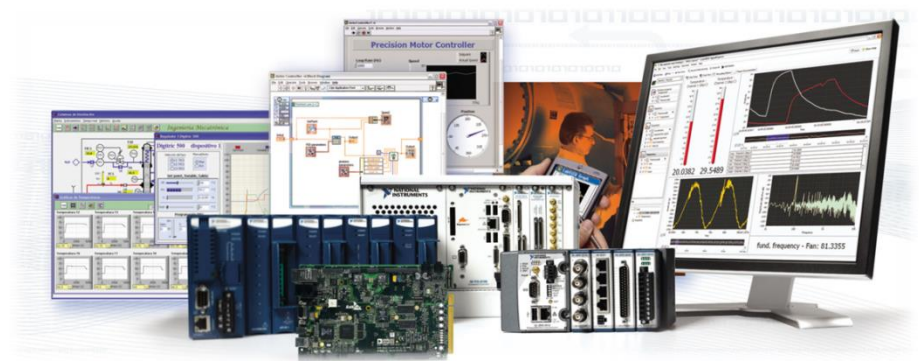
실습8-8-4) 다이나믹 이벤트 사용법

- ❖ 프런트패널에서 실행하면 그림처럼 사인파가 **0**보다 크거나 같으면 문자열 컨트롤에 '양의 영역'이, **0**보다 작으면 '음의 영역'이 사용자 이벤트를 통해 전달되는 것을 볼 수 있다





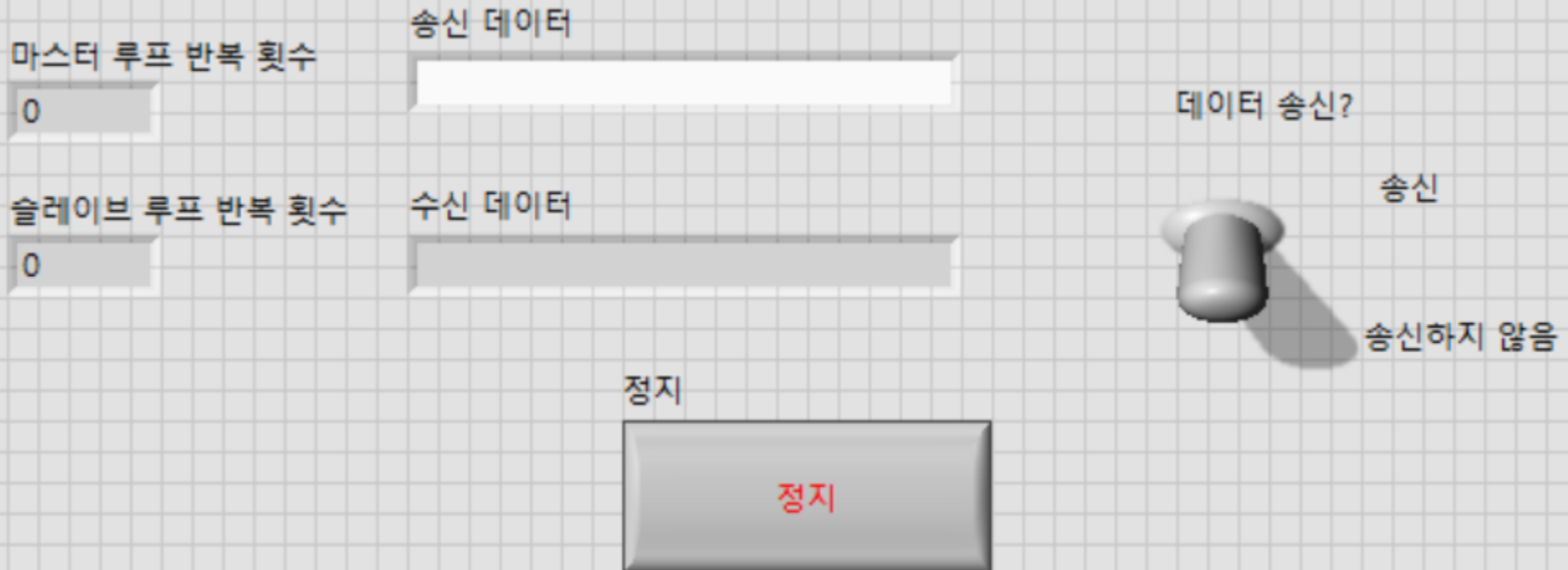
실습8-9-1) 마스터/슬레이브 디자인 패턴 사용법





실습8-9-1) 마스터/슬레이브 디자인 패턴 사용법

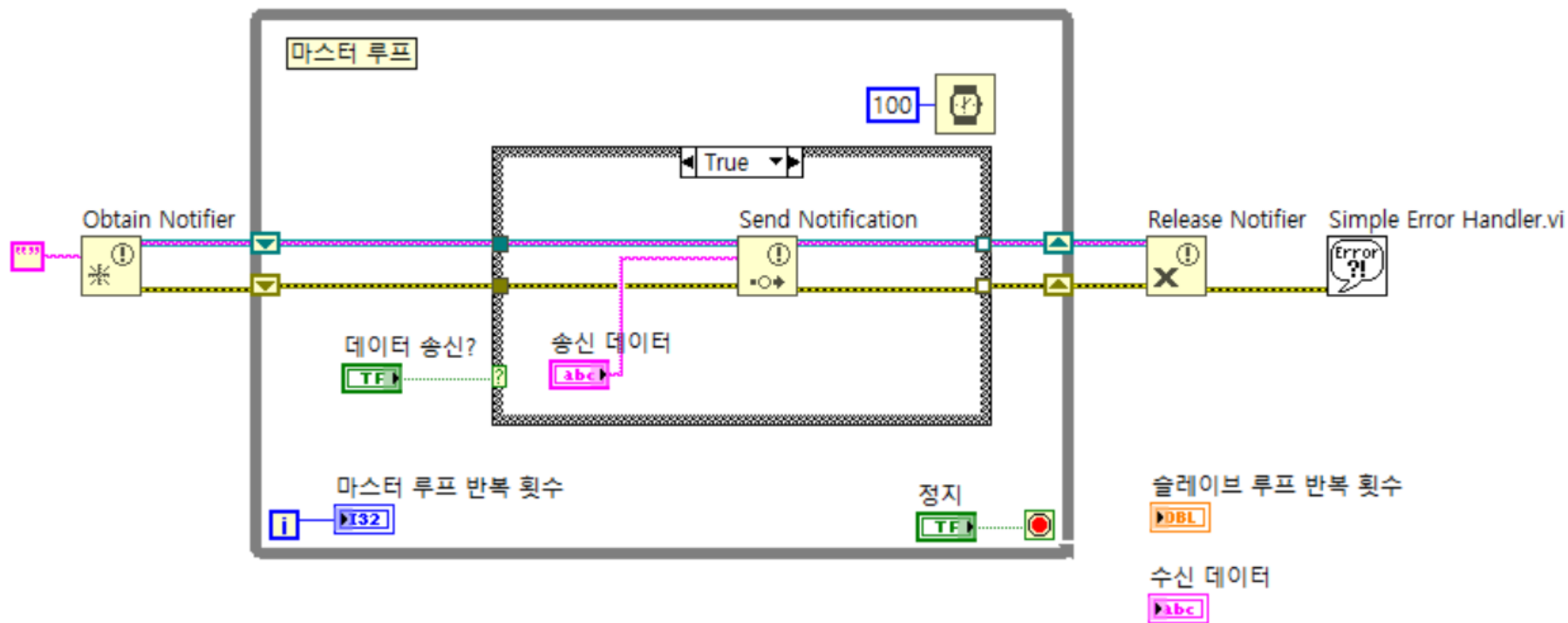
❖ 그림과 같이 프런트 패널을 구성한다.





실습8-9-1) 마스터/슬레이브 디자인 패턴 사용법

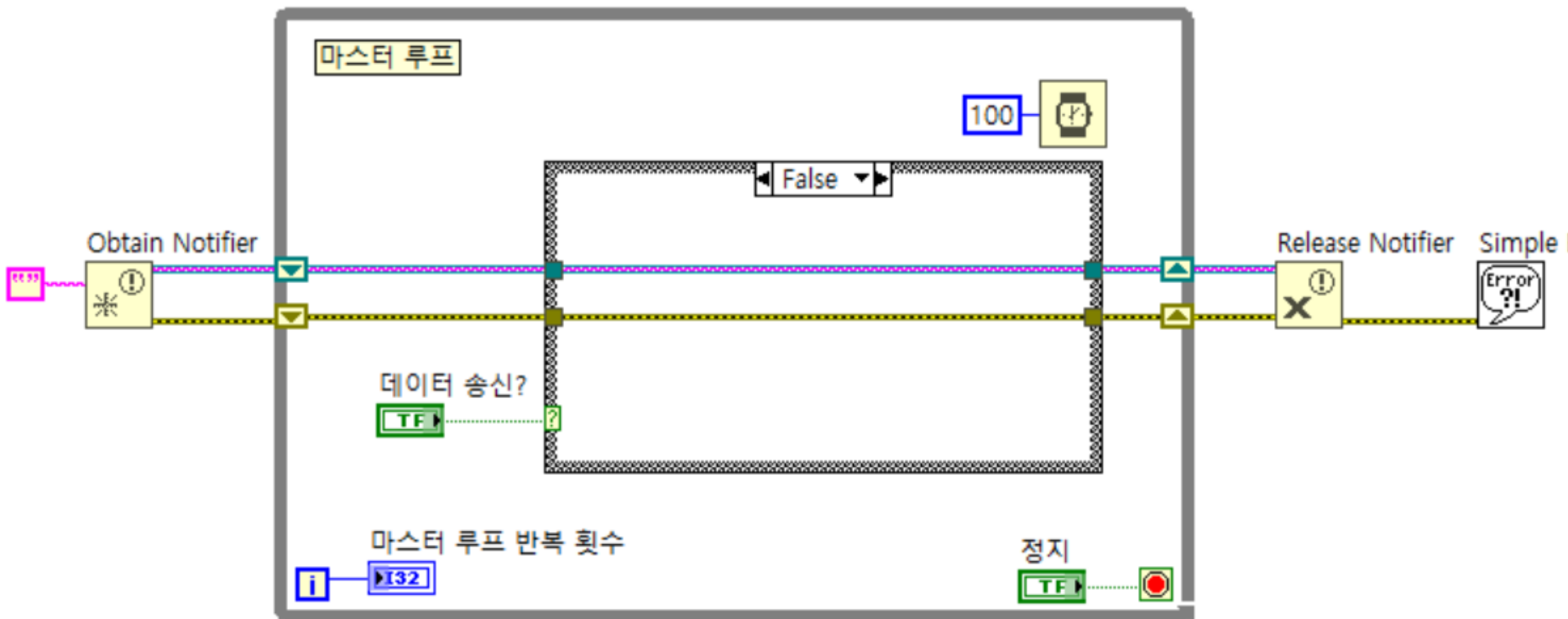
❖ 그림과 같이 블록 다이어그램을 구성한다.





실습8-9-1) 마스터/슬레이브 디자인 패턴 사용법

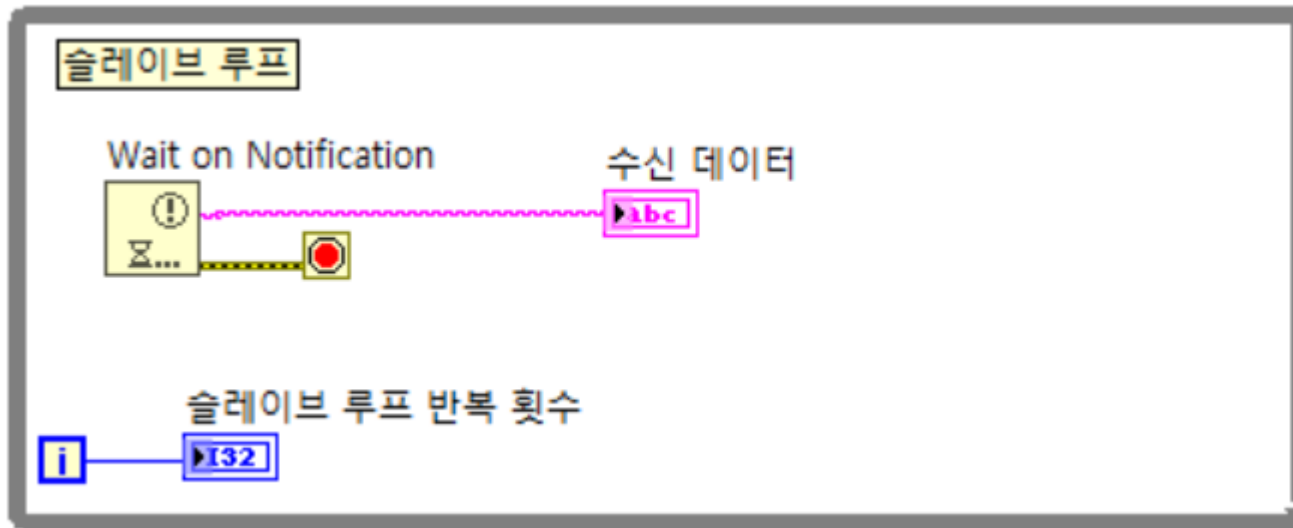
❖ 그림과 같이 블록 다이어그램을 구성한다.





실습8-9-1) 마스터/슬레이브 디자인 패턴 사용법

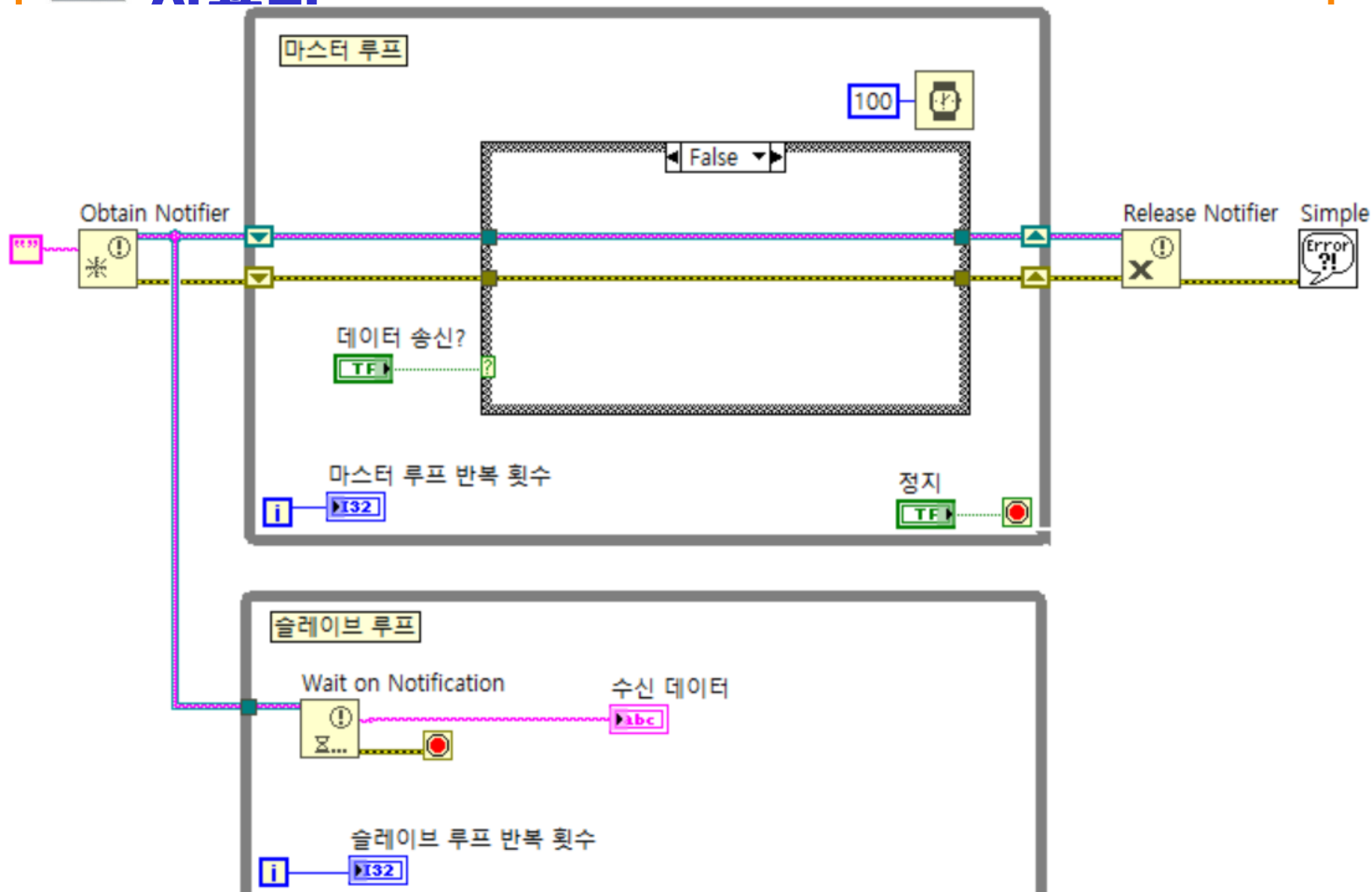
❖ 그림과 같이 블록 다이어그램을 구성한다.





실습8-9-1) 마스터/슬레이브 디자인 패턴

사요버



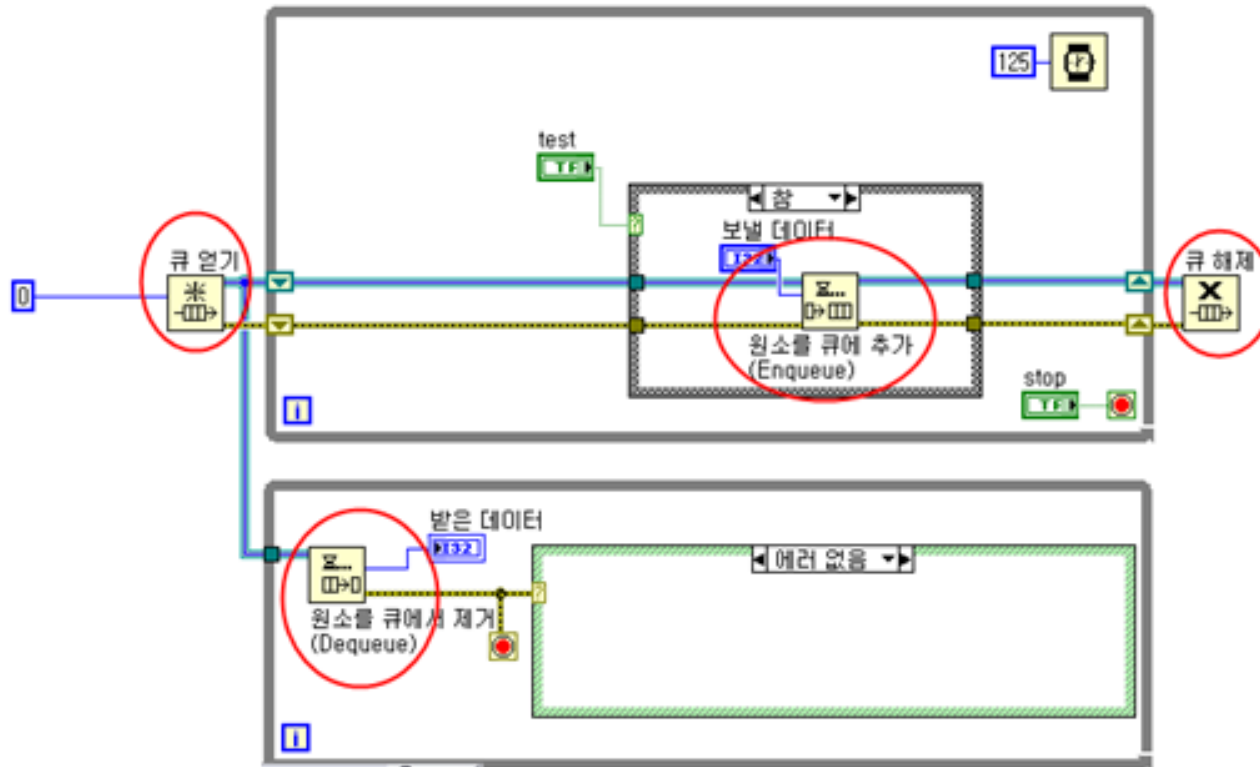


실습8-9-1) 마스터/슬레이브 디자인 패턴 사용법

- ❖ 그림과 같이 ‘데이터 송신?’을 ‘송신하지 않음’ 위치에 놓고 실행.
- ❖ 그러면 ‘마스터 루프 반복 횟수’는 증가할 것이고 데이터는 전달되지 않을 것
- ❖ 이를 확인한 다음 ‘데이터 송신?’을 ‘송신’으로 변경하게 되면 두 **While** 루프의 반복 횟수가 증가될 것이며, ‘송신 데이터’에서 입력한 값이 ‘수신 데이터’에 전달이 되고, ‘송신 데이터’의 입력 값을 바꾸면 실시간으로 ‘수신 데이터’에 전달



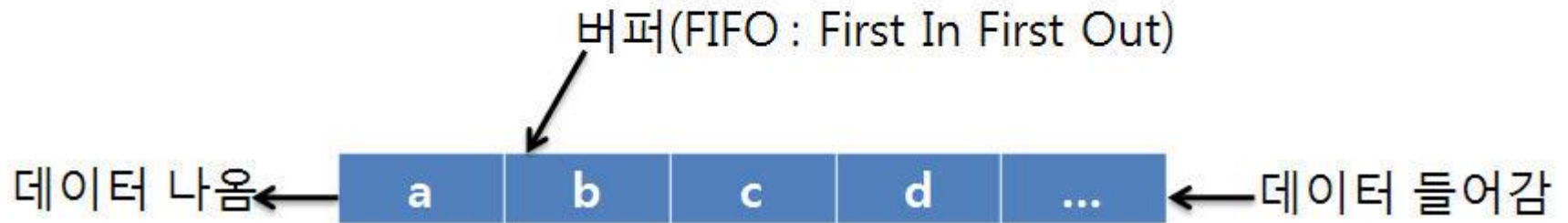
생산/소비 데이터(큐)



• '원소를 큐에 추가'노드에서 '원소를 큐에서 제거'노드로 데이터 실시간 전달

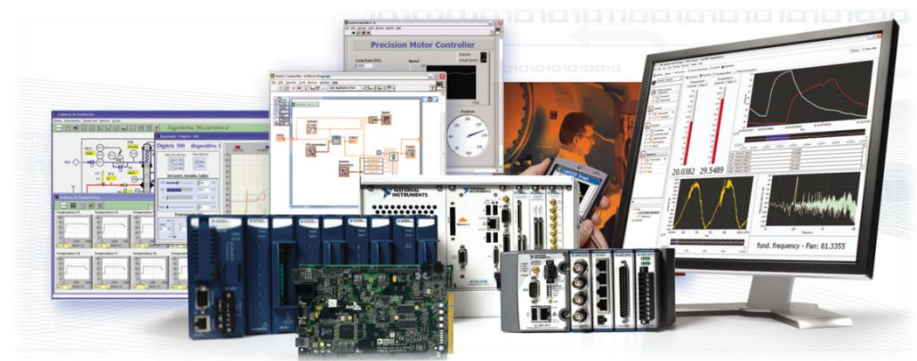


큐의 버퍼 구조



- 알림자와 달리 버퍼 구조를 가짐으로써 데이터 손실을 방지.

실습8-10-1) 생산/소비 데이터 디자인 패턴





실습8-10-1) 생산/소비 데이터 디자인 패턴

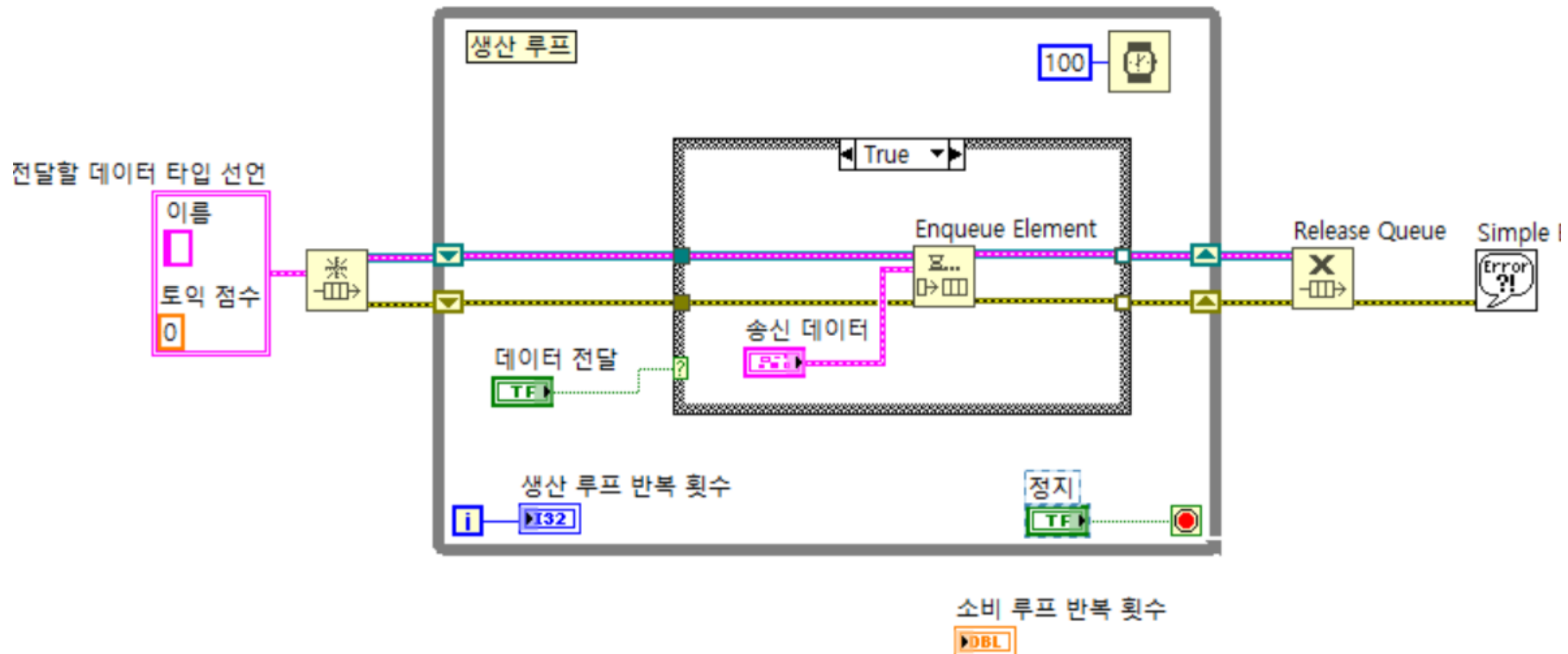
❖ 그림과 같이 프론트 패널을 구성

생산 루프 반복 횟수		송신 데이터		데이터 전달	
99		이름	홍길동		전달
		토익 점수	70		
소비 루프 반복 횟수		수신 데이터		정지	
30		이름			
		토익 점수	0		
		:전달하지 않음:			



실습8-10-1) 생산/소비 데이터 디자인 패턴

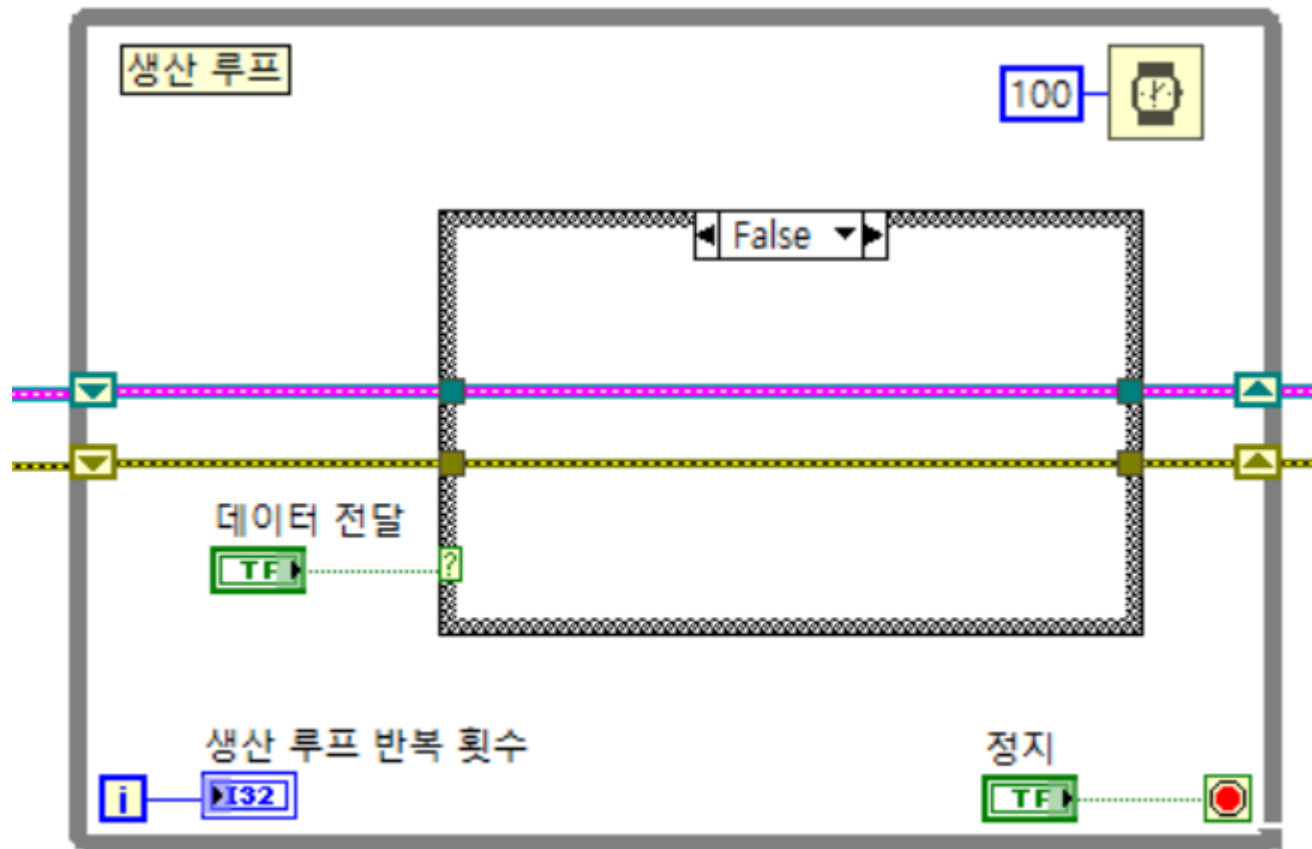
❖ 그림과 같이 블록 다이어그램을 구성





실습8-10-1) 생산/소비 데이터 디자인 패턴

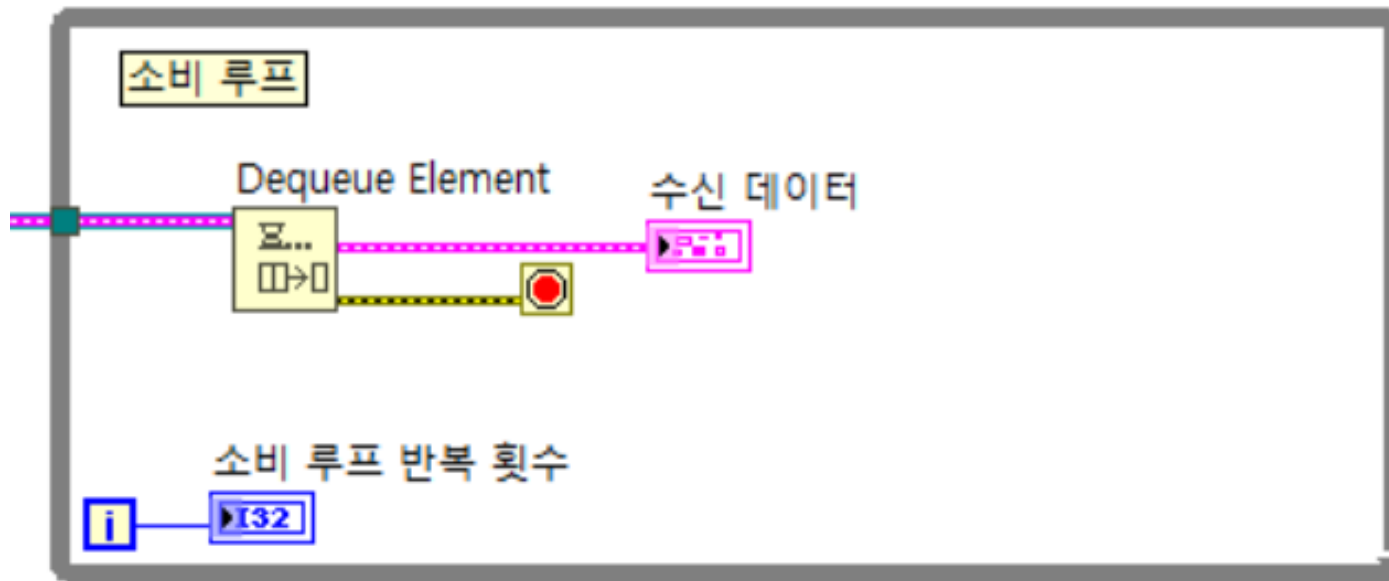
❖ 그림과 같이 블록 다이어그램을 구성





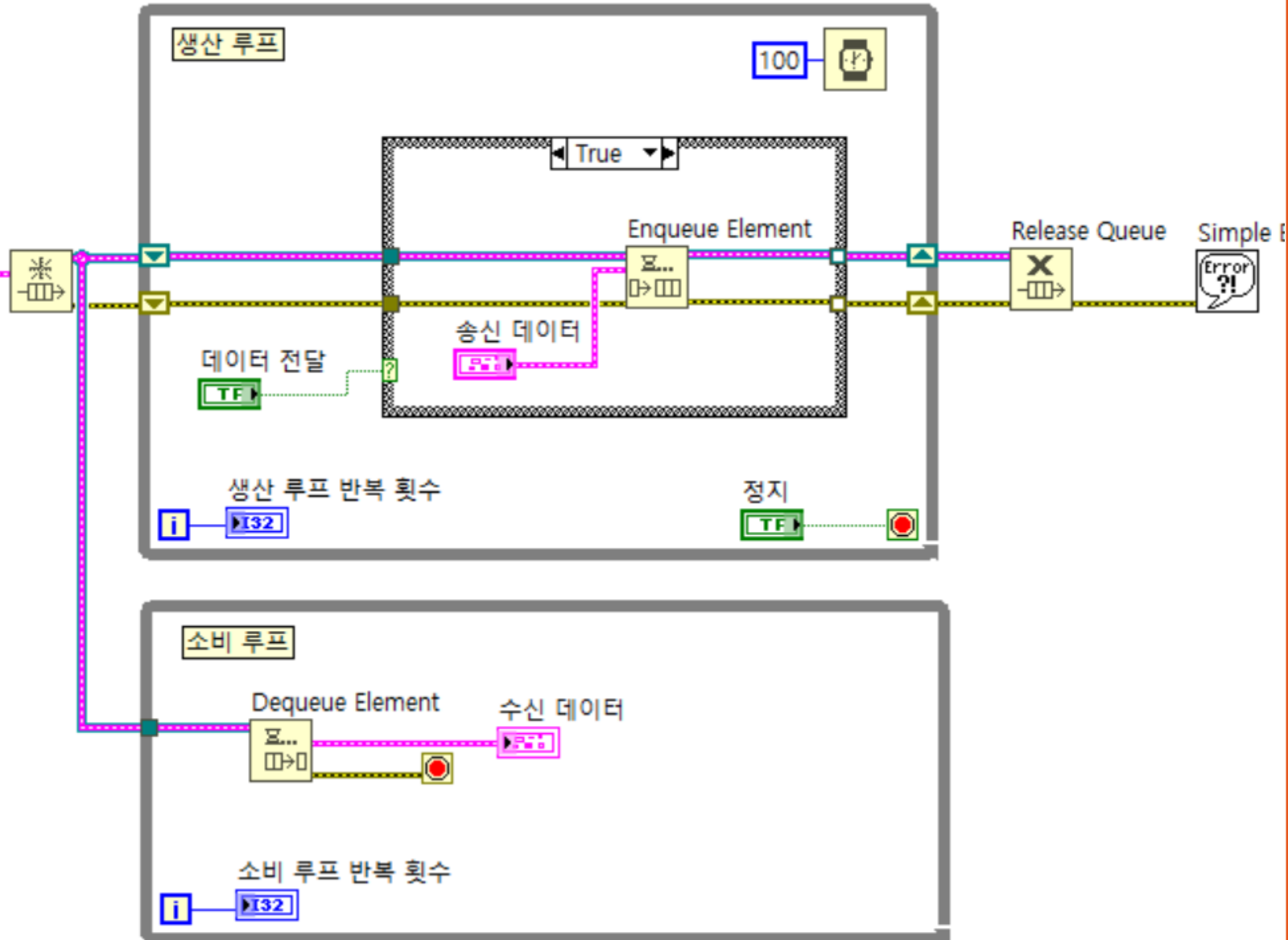
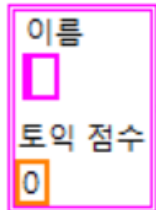
실습8-10-1) 생산/소비 데이터 디자인 패턴

❖ 그림과 같이 블록 다이어그램을 구성



실습8-10-1) 생산/소비 데이터 디자인

데이터 타입 선언





실습8-10-1) 생산/소비 데이터 디자인 패턴

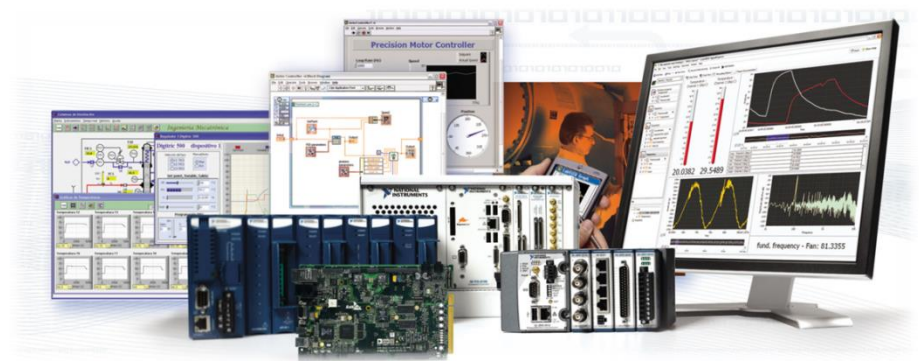
- ❖ 그림과 같이 ‘데이터 전달’을 ‘전달하지 않음’ 위치에 놓고 실행.
- ❖ 그러면 ‘생산 루프 반복 횟수’는 증가할 것이고, 데이터는 전달되지 않음.
- ❖ 이를 확인한 다음 ‘데이터 전달’을 ‘전달’로 변경하게 되면 두 **While** 루프의 반복 횟수가 증가될 것이며, ‘송신 데이터’에서 입력한 값이 ‘수신 데이터’에 전달이 되고, ‘송신 데이터’의 입력 값을 바꾸어도 실시간으로 ‘수신 데이터’에 전달



The diagram illustrates a queue system's state transitions. It starts with an initial state '이 루프는 생산 루프입니다.' (This loop is the production loop). A transition labeled '[0] "Queue Event": 값 변경' leads to a sub-state 'Queue Event'. This sub-state contains a 'TF' (True/False) condition. From the 'Queue Event' sub-state, a transition labeled 'element' leads to a state 'Σ...' (Sigma...). This state then transitions to a state 'status'. A transition labeled 'v' leads to a state 'v' (v). A transition labeled 'X' leads to a state 'X' (X). A transition labeled 'data (can be any type)' leads to a state 'data' (data). A transition labeled '이 루프는 소비 루프입니다.' (This loop is the consumption loop) leads to a state '이 루프는 소비 루프입니다.' (This loop is the consumption loop). This state contains a sub-state '에러 없음' (No error) and a transition labeled 'Σ...' (Sigma...). A transition labeled 'v' leads to a state 'v' (v). A transition labeled 'X' leads to a state 'X' (X). A transition labeled 'data (can be any type)' leads to a state 'data' (data). A transition labeled '이 루프는 생산 루프입니다.' (This loop is the production loop) leads back to the initial state.

- 

실습8-11-1) 생산/소비 이벤트 디자인 패턴





실습8-11-1) 생산/소비 이벤트 디자인 패턴

❖ 그림과 같이 프론트 패널을 구성

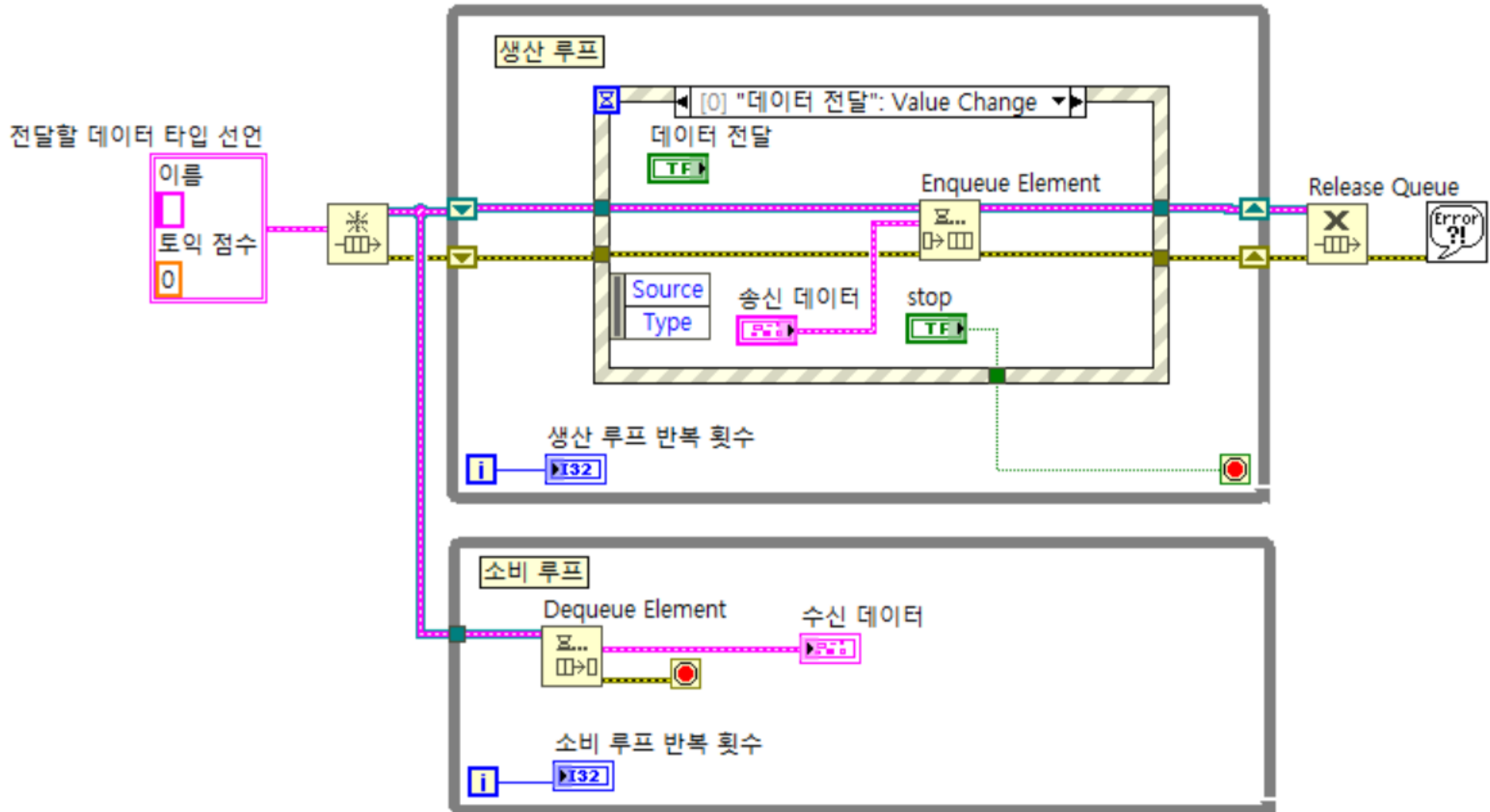
송신 데이터	
생산 루프 반복 횟수	이름
0	홍길동
	토익 점수
	550

수신 데이터	
소비 루프 반복 횟수	이름
0	
	토익 점수
	0

데이터 전달	
데이터 전달	
정지	



❖ 그림과 같이 블록 다이어그램을 구성





실습8-11-1) 생산/소비 이벤트 디자인 패턴

- ❖ 실행하여 ‘데이터 전달’ 버튼의 값을 변경하면 ‘송신 데이터’가 ‘수신 데이터’로 전달이 되고 ‘생산 루프 반복 횟수’와 ‘소비 루프 반복 횟수’가 하나씩 동시에 증가할 것
- ❖ ‘송신 데이터’의 입력을 변경하고 이를 전달하려면 역시 ‘데이터 전달’ 버튼의 값을 변경하면 된다.

생산 루프 반복 횟수 10	송신 데이터 이름 홍길동 토익 점수 550	데이터 전달 데이터 전달
소비 루프 반복 횟수 10	수신 데이터 이름 홍길동 토익 점수 550	정지



8장 요약

- 단일 루프 디자인 패턴 / 멀티 루프 디자인 패턴
- 멀티 루프간의 실시간 데이터 전송법
- 다양한 변수들 - 로컬, 글로벌, 기능적 글로벌, 공유
- 동기화 - 알림자, 큐