

HBE-COMBO II를 활용한

Traffic Lights

CMB2-APP-TL 사용자 매뉴얼



HANBACK ELECTRONICS CO., LTD.

차례

1. 외형 구성 및 설명	-----	3
2. 입출력 핀 구성표	-----	6
3. 회로도	-----	7

1. 외형 구성 및 설명

(가) 개요

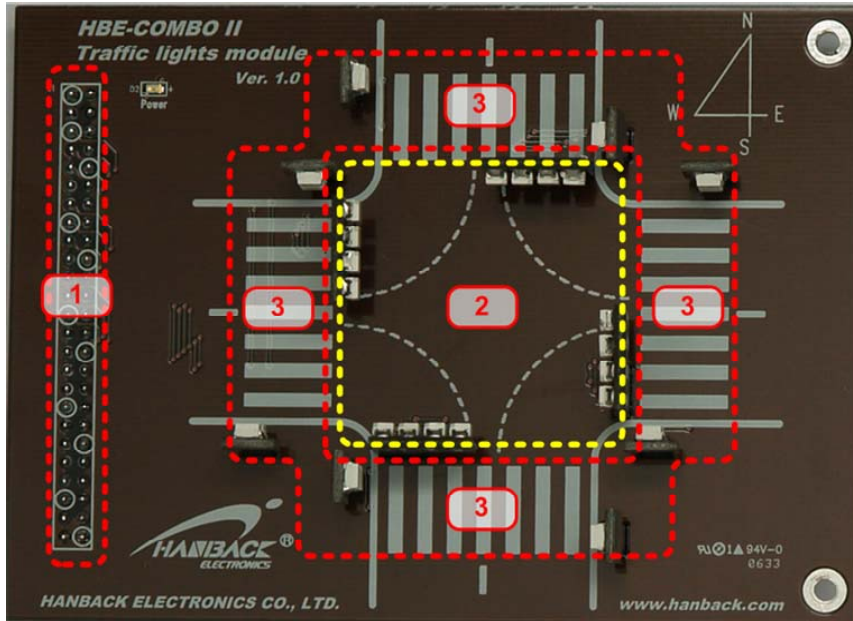
본 장비는 HBE-COMBO II의 확장 커넥터(EXT-3)로 장착하여 신호등 모듈을 구현하는 장비이다. 모듈에서는 사거리 신호등을 구현하기 위해 차량 신호등 4개와 보행자 신호등 4조를 가지고 구성하고 있다. 따라서 차량 신호등에서는 녹색, 황색, 적색 LED를 사용하여 좌회전, 직진, 정지등을 표시하도록 하고 있다. 또한 보행자 신호등에서 적색, 녹색 LED를 사용하여 보행 및 정지를 표시해 주도록 하고 있다. 따라서 본 모듈에서는 현 도로의 사거리 신호를 그대로 구성한 것으로 사거리 신호등의 원리를 이해하고 제어할 수 있도록 지원해 주고 있다.

신호등 모듈의 동작은 현 도로에 있는 사거리 신호등 모듈의 동작을 그대로 구현한 것으로, 차량 신호등 4개가 시계 방향으로 돌아가면 녹색과 좌회전 신호가 동시에 들어오게 된다. 또한 여기에 맞추어 보행자 신호등도 하나씩 녹색 LED가 켜지게 된다. 차량 신호등에서는 녹색과 좌회전 신호가 동시에 켜지게 되고, 적색 LED를 켜려고 하는 순간 황색 LED를 거쳐서 적색 LED를 켜게 된다. 또한 보행자 신호등에서는 녹색 LED가 켜지고 적색 LED로 넘어가려는 순간 녹색 LED가 깜빡 거리고, 얼마의 시간이 거치면 적색 LED가 켜지게 된다. 이러한 것은 10간의 간격을 두고 사거리 신호등이 계속 바뀌게 구성하고 있다. 또한 소스에서는 새벽 2시에서 5시 사이에서는 차량 신호등 및 보행자 신호등에서 적색 LED만 깜빡 거리게 소스를 설계하였다. 이러한 시간은 장비의 7-Segment를 통해 현재 시간을 확인할 수 있다.

(나) 사양

- Car(PLCC type LED)
 - Green(8EA)
 - Yellow(4EA)
 - Red(4EA)
- Foot passenger(PLCC type LED)
 - Green(8EA)
 - Red(8EA)

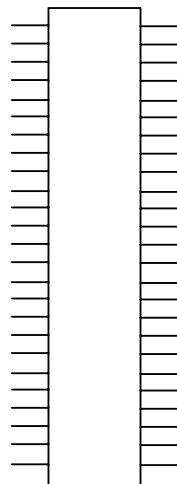
(다) 각 부분의 명칭 및 설명



[그림 1] 외형 모습 및 명칭

① Traffic Lights 모듈 연결 커넥터

모듈의 50핀 확장 커넥터를 통해 장비에서 전원을 받아서 모듈에서 사용하고, 또한 각 장치들에 대한 데이터 신호를 받아들여 동작을 하게 된다. 자세히 살펴보면 FPGA 모듈에서 전달되는 데이터 신호가 LED를 동작시키게 된다. 따라서 사거리 신호등의 각 LED마다 50핀 확장 커넥터를 통해 연결이 되어 있으므로 장비의 FPGA를 통해 각 LED를 제어할 수 있다. FPGA 모듈에서 '1'이라는 신호를 주어 각 LED를 제어하게 되는 것이다. 아래 그림에서 50핀 커넥터의 핀 정보를 확인해 볼 수 있다.



[그림2]. Vending Machine 모듈 확장 핀 구성

② 차량용 신호등

• 구성

차량용 신호등은 총 4개의 PLCC 타입의 LED를 4개(동, 서, 남, 북)로 구성하고 있다. 하나의 차량용 신호등에는 좌측부터 녹색(직진), 녹색(좌회전), 황색(경고), 적색(정지)의 4개의 LED로 구성되어 있고, 하나의 거리당 1개씩 총 4개로 구성이 되어 있다. 따라서 직진 및 좌회전의 신호등을 제어하여 차량을 움직이게 하고 황색 및 적색 신호를 이용하여 차량을 정지하게 하고 있다. 회로에서의 각 LED의 이름은 동(E), 서(W), 남(S), 북(N)이 커넥터 이름 앞에 오게 되고, 뒷 부분에 연결된 각 LED의 색깔이 오게 된다. 예를 들어 “E_LEFT” 동쪽의 좌회전 신호를 가리키게 된다.

• 동작

소스에서는 차량용 신호등이 하나씩 직진 및 좌회전이 켜지게 되고 나머지 3개의 신호등에서는 적색 LED를 표시하게 된다. 또한 황색 LED를 사용하여 경고를 표시하게 된다. 이 모든 것은 FPGA와 하나씩 연결되어 있어 각각의 제어가 가능하도록 구성하고 있다. FPGA 모듈에서 ‘1’의 신호를 주어 LED를 켜게 되는 것이다. 처음에 직진과 좌회전의 녹색 LED를 켜게 된다. 그리고 얼마의 시간이 지나 적색 LED로 넘어가기 전에 황색 LED를 거쳐 적색 LED로 넘어가게 된다.

③ 보행자 신호등

• 구성

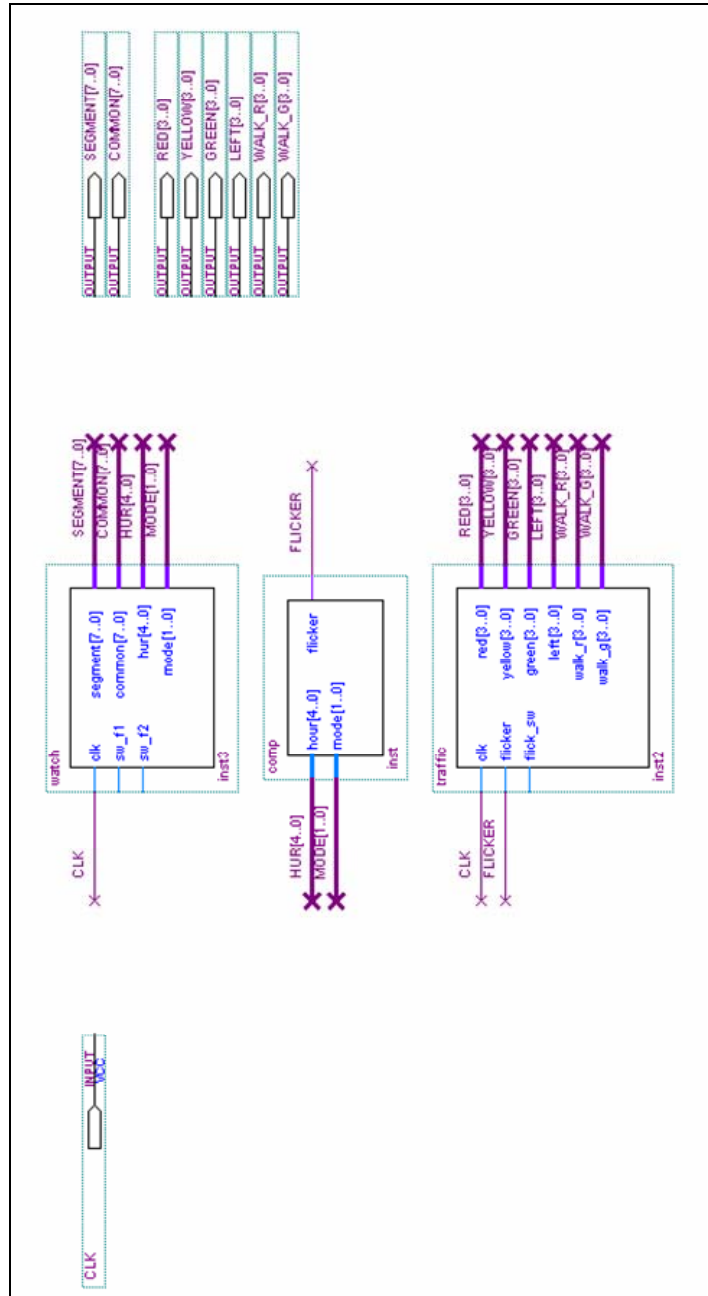
보행자 신호등은 차량 신호등과 마찬가지로 각 거리마다 하나씩 존재하여 총 4조의 보행자 신호등으로 구성하고 있다. 따라서 각 보행자 신호등마다 정지 및 보행을 나타내는 적색, 녹색의 LED로 구성을 하고 있다. 회로에서의 각 보행자 신호등의 LED의 이름은 앞에 보행자신호등을 나타내는 WALK가 붙고, 뒤에 동(E), 서(W), 남(S), 북(N)의 방향과 LED 색깔을 나타내는 RED, GREEN이 오게 된다.

• 동작

소스에서의 보행자 신호등의 동작은 차량용 신호등을 보고 동작하도록 구성하고 있다. 따라서 차량용 신호등에 녹색 및 좌회전 신호가 켜지면, 차량용 신호등의 오른쪽에 있는 보행자 신호등에서 녹색 LED가 켜지게 구성하고 있다. 따라서 녹색이 점등이 되면 얼마의 시간이 지나고 나면 LED가 깜빡 거리게 되고, 적색 LED를 켜게 된다.

2. SOURCE 설명

신호등 모듈의 설계 소스는 장비 내부에 있는 7-Segment와 연동하여 데모를 구동하고 있다. 따라서 7-Segment를 통해 시간을 표시해 주고, 신호등 모듈에서 차량 신호등과 보행자 신호등이 동작 된다. 7-Segment를 통해 하루 24시간을 표현한다. 또한 차량 및 보행자 신호등에서는 10단위로 신호가 바뀌게 된다. 또한 새벽 2시부터 5시 사이에는 황색 신호등이 점멸하는 형식으로 작동이 된다. 다음부터 소스에 대해 자세히 살펴보자.



위의 그림은 신호등 모듈의 top 파일을 보여주고 있다. 총 3개의 파일을 통해 신호등의 동작을 표현하고 있다. 다음부터 각 설계 파일을 살펴 보도록 하자.

1) WATCH.VHD

다음 블록은 시간을 표시하기 위한 설계 top 파일이다. 따라서 하위에 5개의 파일을 연동하여 사용하고 있다. sw_f1, sw_f2를 통해 내부 보드를 설정하고, 설정된 내부 모드의 값을 증가 시키는 입력을 가진다. 또한 SEGMENT와 COMMON을 통해 시계를 display해 준다. HUR과 MODE에서 신호등 점멸을 위한 출력을 한다.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY WATCH IS
PORT (
    CLK : IN STD_LOGIC;
    SW_F1 : IN STD_LOGIC;
    SW_F2 : IN STD_LOGIC;

    SEGMENT : OUT STD_LOGIC_VECTOR (7 DOWNT0 0);
    COMMON : OUT STD_LOGIC_VECTOR (7 DOWNT0 0);

    HUR : OUT INTEGER RANGE 23 DOWNT0 0;
    MODE : OUT STD_LOGIC_VECTOR (1 DOWNT0 0)
);
END WATCH;
```

ARCHITECTURE A OF WATCH IS

다음 블록은 WATCH.VHD에서 총 5개의 파일을 연동하여 사용하기 위한 COMPONENT 부분이다. 하위 5개의 파일을 간단히 설명하면, TIME.VHD에서는 입력되는 클럭을 이용하여 시, 분, 초의 클럭을 생성하게 된다. 그리고 CLK_DIV.VHD에서는 신호등이 동작하기 위한 내부 클럭을 만들어 준다. 또한 DECTOSEG1.VHD와 DECTOSEG2.VHD는 생성된 값을 7-Segment에 표현하기 위한 decode부분이다. 마지막으로 SEG_MODULE.VHD는 스캐닝 방법을 통해 7-Segment를 구동시키는 설계 파일로써, 10KHz의 빠른 클럭으로 6개의 7-Segment를 하나씩 선택하면서 각각의 데이터를 display 하게 된다. 따라서 눈으로 볼 때 6개의 7-Segment가 동시에 켜져 있는 것 처럼 보이게 된다.

```
COMPONENT TIME
PORT (
    CLK : IN STD_LOGIC;
    SW_F1 : IN STD_LOGIC;
    SW_F2 : IN STD_LOGIC;

    HOUR : OUT INTEGER RANGE 23 DOWNT0 0;
    MINUTE : OUT INTEGER RANGE 59 DOWNT0 0;
    SECOND : OUT INTEGER RANGE 59 DOWNT0 0;
    SET_MODE : OUT STD_LOGIC_VECTOR (1 DOWNT0 0)
);
END COMPONENT;
```

```
COMPONENT CLK_DIV
PORT (
    CLK : IN STD_LOGIC;
```

```

    CLK_1H : OUT STD_LOGIC;
    CLK_100H : OUT STD_LOGIC
);
END COMPONENT;

COMPONENT DECTOSEG1
PORT(
    NUMBER : IN INTEGER RANGE 23 DOWNT0 0;
    SEG_TEN : OUT STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_ONE : OUT STD_LOGIC_VECTOR (6 DOWNT0 0)
);
END COMPONENT;

COMPONENT DECTOSEG2
PORT(
    NUMBER : IN INTEGER RANGE 59 DOWNT0 0;
    SEG_TEN : OUT STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_ONE : OUT STD_LOGIC_VECTOR (6 DOWNT0 0)
);
END COMPONENT;

COMPONENT SEG_MODULE
PORT (
    CLK : IN STD_LOGIC;
    SEG_EN : IN STD_LOGIC_VECTOR (7 DOWNT0 0);
    SEG_8 : IN STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_7 : IN STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_6 : IN STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_5 : IN STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_4 : IN STD_LOGIC_VECTOR (6 DOWNT0 0);
    SEG_3 : IN STD_LOGIC_VECTOR (6 DOWNT0 0);

    SEG_OUT : OUT STD_LOGIC_VECTOR (7 DOWNT0 0);
    COMMON : OUT STD_LOGIC_VECTOR (7 DOWNT0 0)
);
END COMPONENT;

SIGNAL CLK_1H, CLK_100H : STD_LOGIC;

SIGNAL HOUR : INTEGER RANGE 23 DOWNT0 0;
SIGNAL MINUTE : INTEGER RANGE 59 DOWNT0 0;
SIGNAL SECOND : INTEGER RANGE 59 DOWNT0 0;
SIGNAL SET_MODE : STD_LOGIC_VECTOR (1 DOWNT0 0);

SIGNAL SEG_8, SEG_7, SEG_6, SEG_5 : STD_LOGIC_VECTOR (6 DOWNT0 0);
SIGNAL SEG_4, SEG_3 : STD_LOGIC_VECTOR (6 DOWNT0 0);

SIGNAL SEG_EN : STD_LOGIC_VECTOR (7 DOWNT0 0);

SIGNAL DIGIT_ON : STD_LOGIC_VECTOR (2 DOWNT0 0);

BEGIN

U0 : CLK_DIV
PORT MAP (
    CLK => CLK,
    CLK_1H => CLK_1H,
    CLK_100H => CLK_100H
);

U1 : TIME
PORT MAP (

```

```

CLK => CLK_100H,
SW_F1 => SW_F1,
SW_F2 => SW_F2,

    HOUR => HOUR,
    MINUTE => MINUTE,
    SECOND => SECOND,
    SET_MODE => SET_MODE
);

U2 : DECTOSEG1
PORT MAP (
    NUMBER => HOUR,
    SEG_TEN => SEG_8,
    SEG_ONE => SEG_7
);

U3 : DECTOSEG2
PORT MAP (
    NUMBER => MINUTE,
    SEG_TEN => SEG_6,
    SEG_ONE => SEG_5
);

U4 : DECTOSEG2
PORT MAP (
    NUMBER => SECOND,
    SEG_TEN => SEG_4,
    SEG_ONE => SEG_3
);

U5 : SEG_MODULE
PORT MAP (
    CLK => CLK,
    SEG_EN => SEG_EN,
    SEG_8 => SEG_8,
    SEG_7 => SEG_7,
    SEG_6 => SEG_6,
    SEG_5 => SEG_5,
    SEG_4 => SEG_4,
    SEG_3 => SEG_3,

    SEG_OUT => SEGMENT,
    COMMON => COMMON
);

SEG_EN(7 DOWNTO 4) <= DIGIT_ON(2) & DIGIT_ON(2) & DIGIT_ON(1) & DIGIT_ON(1);
SEG_EN(3 DOWNTO 0) <= DIGIT_ON(0) & DIGIT_ON(0) & "00";

HUR <= HOUR;
MODE <= SET_MODE;

```

다음의 PROCESS문을 통해 신호등이 동작하는 과정에서 시간을 설정하기 위한 구문이다. 따라서 버튼 스위치를 통해 7-Segment를 점멸을 통해 초, 분, 시 순으로 점멸하게 된다.

```

PROCESS (CLK_100H)
BEGIN
    IF CLK_100H'EVENT AND CLK_100H = '1' THEN
        IF SET_MODE = "01" THEN

```

```

        DIGIT_ON(2 DOWNT0 1) <= "11";
        DIGIT_ON(0) <= CLK_1H;
    ELSIF SET_MODE = "10" THEN
        DIGIT_ON(2) <= '1';
        DIGIT_ON(1) <= CLK_1H;
        DIGIT_ON(0) <= '1';
    ELSIF SET_MODE = "11" THEN
        DIGIT_ON(2) <= CLK_1H;
        DIGIT_ON(1 DOWNT0 0) <= "11";
    ELSE
        DIGIT_ON <= "111";
    END IF;
END IF;
END PROCESS;
END A;

```

2) TIME.VHD

다음 설계 파일은 앞서 언급했듯이 7-Segment에 시계를 표시하기 위해 시간, 분, 초를 설정해 주는 구간이다. 따라서 입력 클럭에 따라 정해진 시간에 따라 시간이 동작하게 된다. 또한 스위치를 통해 시간 설정도 할 수 있도록 지원해 주고 있다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY TIME IS
PORT (
    CLK : IN STD_LOGIC;
        -- 시간 설정 모드에서 시, 분, 초를 선택하는 스위치
    SW_F1 : IN STD_LOGIC;
        -- 시간 설정 모드에서 값을 증가하는 스위치
    SW_F2 : IN STD_LOGIC;

        -- 시, 분, 초를 표시하기 위한 port
    HOUR : OUT INTEGER RANGE 23 DOWNT0 0;
    MINUTE : OUT INTEGER RANGE 59 DOWNT0 0;
    SECOND : OUT INTEGER RANGE 59 DOWNT0 0;
        -- 시간 셋팅 모드를 표시하기 위한 변수
    SET_MODE : OUT STD_LOGIC_VECTOR (1 DOWNT0 0)
);
END TIME;

ARCHITECTURE A OF TIME IS

    -- 시간 Setting State machine
    TYPE SET_MD IS (IDLE, SET_SEC, SET_MIN, SET_HUR);
    SIGNAL TIME_SET : SET_MD;

    SIGNAL S_SW_F1 : STD_LOGIC_VECTOR (1 DOWNT0 0);
    SIGNAL S_SW_F2 : STD_LOGIC_VECTOR (1 DOWNT0 0);

    SIGNAL HUR : INTEGER RANGE 23 DOWNT0 0;
    SIGNAL MIN : INTEGER RANGE 59 DOWNT0 0;
    SIGNAL SEC : INTEGER RANGE 59 DOWNT0 0;

```

```
SIGNAL CLK_SEC, CLK_MIN, CLK_HUR : STD_LOGIC;
```

```
BEGIN
```

다음의 PROCESS문은 SW_F1, SW_F2의 입력 시점을 감지하기 위한 구문으로 스위치의 값이 '01'이 되는 순간 다시 말해 상승 에지에서 스위치가 눌러진 것으로 인식하게 된다.

```
PROCESS (CLK)
BEGIN
  IF CLK'EVENT AND CLK = '1' THEN
    S_SW_F1(0) <= SW_F1;
    S_SW_F1(1) <= S_SW_F1(0);

    S_SW_F2(0) <= SW_F2;
    S_SW_F2(1) <= S_SW_F2(0);
  END IF;
END PROCESS;
```

다음 PROCESS문은 시간 Setting 모드의 상태를 정의하기 위한 구문이다.

```
PROCESS (CLK)
BEGIN
  IF CLK'EVENT AND CLK = '1' THEN
    IF S_SW_F1 = "01" THEN
      CASE TIME_SET IS
        WHEN IDLE =>
          SET_MODE <= "01";
          TIME_SET <= SET_SEC;
        WHEN SET_SEC =>
          SET_MODE <= "10";
          TIME_SET <= SET_MIN;
        WHEN SET_MIN =>
          SET_MODE <= "11";
          TIME_SET <= SET_HUR;
        WHEN SET_HUR =>
          SET_MODE <= "00";
          TIME_SET <= IDLE;
        WHEN OTHERS =>
          SET_MODE <= "00";
          TIME_SET <= IDLE;
      END CASE;
    END IF;
  END IF;
END PROCESS;
```

다음 PROCESS문은 100Hz의 클럭을 분주하여 1Hz의 클럭 enable signal을 만든다.

```
PROCESS (CLK)
VARIABLE CNT : INTEGER RANGE 99 DOWNT0 0;
BEGIN
  IF CLK'EVENT AND CLK = '1' THEN
    IF CNT = 98 THEN
      CLK_SEC <= '1';
      CNT := 0;
    ELSE
      CLK_SEC <= '0';
      CNT := CNT + 1;
    END IF;
  END IF;
END IF;
```

```
END PROCESS;
```

다음 PROCESS문에서는 모드가 초를 Setting 하는 모드일 경우 SW_F2가 눌러질 경우 '0'으로 클리어 한다.

```
PROCESS (CLK, TIME_SET, S_SW_F2)
BEGIN
  IF (TIME_SET = SET_SEC) AND (S_SW_F2 = "01") THEN
    SEC <= 0;
  ELSIF CLK'EVENT AND CLK = '1' THEN
    IF CLK_SEC = '1' THEN
      IF SEC = 58 THEN
        CLK_MIN <= '1';
      ELSE
        CLK_MIN <= '0';
      END IF;
      IF SEC = 59 THEN
        SEC <= 0;
      ELSE
        SEC <= SEC + 1;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

다음 PROCESS문에서는 모드가 분을 Setting하는 경우 SW_F2가 눌러질 경우 눌러진 만큼 숫자가 증가하게 된다.

```
PROCESS (CLK)
BEGIN
  IF CLK'EVENT AND CLK = '1' THEN
    IF (TIME_SET = SET_MIN) AND (S_SW_F2 = "01") THEN
      IF MIN = 59 THEN
        MIN <= 0;
      ELSE
        MIN <= MIN + 1;
      END IF;
    ELSIF TIME_SET = IDLE THEN
      IF CLK_SEC = '1' AND CLK_MIN = '1' THEN
        IF MIN = 58 THEN
          CLK_HUR <= '1';
        ELSE
          CLK_HUR <= '0';
        END IF;
        IF MIN = 59 THEN
          MIN <= 0;
        ELSE
          MIN <= MIN + 1;
        END IF;
      END IF;
    END IF;
  END IF;
END PROCESS;
```

다음 PROCESS문은 Setting 모드가 시간일 경우 SW_F2가 눌러질 경우 눌러진 만큼의 시간의 숫자가 증가한다.

```
PROCESS (CLK)
BEGIN
  IF CLK'EVENT AND CLK = '1' THEN
    IF (TIME_SET = SET_HUR) AND (S_SW_F2 = "01") THEN
      IF HUR = 23 THEN
```

```

        HUR <= 0;
    ELSE
        HUR <= HUR + 1;
    END IF;
    ELSIF TIME_SET = IDLE THEN
        IF CLK_SEC = '1' AND CLK_MIN = '1' AND CLK_HUR = '1' THEN
            IF HUR = 23 THEN
                HUR <= 0;
            ELSE
                HUR <= HUR + 1;
            END IF;
        END IF;
    END IF;
END IF;
END IF;
END PROCESS;

SECOND <= SEC;
MINUTE <= MIN;
HOUR <= HUR;

END A;

```

3) CLK_DIV.VHD

이 설계 소스는 시간을 표현하기 위한 클럭과 내부 클럭을 만들어 주는 부분이다. 따라서 여기서 분주된 클럭을 이용하여 시간을 표현하고 신호등을 나타내는 LED를 점멸하는 기능을 하게 된다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY CLK_DIV IS
PORT (
    -- 10KHz의 클럭 입력
    CLK : IN STD_LOGIC;
    -- 분주된 100Hz의 클럭
    CLK_100H : OUT STD_LOGIC;
    -- 분주된 1Hz의 클럭
    CLK_1H : OUT STD_LOGIC
);
END CLK_DIV;

```

```

ARCHITECTURE A OF CLK_DIV IS

```

다음에 선언된 변수들은 10KHz를 100Hz로, 10KHz를 1Hz로 분주하기 위한 변수들과 100Hz와 1Hz클럭에 대응되는 변수들이다.

```

    SIGNAL CNT10 : INTEGER RANGE 49 DOWNT0 0;
    SIGNAL CNT1000 : INTEGER RANGE 4999 DOWNT0 0;
    SIGNAL S_CLK_10 : STD_LOGIC;
    SIGNAL S_CLK_1000 : STD_LOGIC;

    BEGIN

    PROCESS (CLK)
    BEGIN
        IF CLK'EVENT AND CLK = '0' THEN
            IF CNT10 = 49 THEN

```

```

        S_CLK_10 <= NOT S_CLK_10;
        CNT10 <= 0;
    ELSE
        CNT10 <= CNT10 + 1;
    END IF;

    IF CNT1000 = 4999 THEN
        S_CLK_1000 <= NOT S_CLK_1000;
        CNT1000 <= 0;
    ELSE
        CNT1000 <= CNT1000 + 1;
    END IF;
END IF;
END PROCESS;

CLK_1H <= S_CLK_1000;
CLK_100H <= NOT S_CLK_10;
END A;

```

4) DECTOSEG1.VHD

이 설계 소스는 10 자리로 카운트 되는 숫자를 7-Segment의 출력하기 위해 하나의 자리 숫자로 변화해 주는 작업을 한다. 예를 들어 현재 카운트 되는 숫자가 '24'일 때, '2'와 '4'의 숫자를 각 자리수로 분리해서 정의해 주는 작업을 한다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DECTOSEG1 IS
    PORT(
        -- 두 자리 십진수 입력
        NUMBER : IN INTEGER RANGE 23 DOWNT0 0;
        -- 10의 자리의 십진수를 7-Segment의 출력으로 변환한 값
        SEG_TEN : OUT STD_LOGIC_VECTOR (6 DOWNT0 0);
        -- 1의 자리의 십진수를 7-Segment의 출력으로 변환한 값
        SEG_ONE : OUT STD_LOGIC_VECTOR (6 DOWNT0 0)
    );
END DECTOSEG1;

ARCHITECTURE A OF DECTOSEG1 IS

    -- 십진수를 7-Segment 값으로 변환하기 위한 COMPONENT
    COMPONENT BCD2SEG
    PORT (
        BCD : IN INTEGER RANGE 9 DOWNT0 0;
        SEGMENT : OUT STD_LOGIC_VECTOR (6 DOWNT0 0)
    );
    END COMPONENT;

    SIGNAL DEC_TEN : INTEGER RANGE 9 DOWNT0 0;
    SIGNAL DEC_ONE : INTEGER RANGE 9 DOWNT0 0;

    BEGIN

        -- 십진수를 7-Segment 값으로 변환하는 블록
        U0 : BCD2SEG
        PORT MAP (

```

```

BCD => DEC_TEN,
SEGMENT => SEG_TEN
);

U1 : BCD2SEG
PORT MAP (
    BCD => DEC_ONE,
    SEGMENT => SEG_ONE
);

```

다음의 PROCESS문은 'NUMBER'의 크기에 따라 십의 자리와 일의 자리 값으로 변환해 주는 역할을 하고 있다.

```

PROCESS( NUMBER )
BEGIN
    IF NUMBER >= 20 THEN
        DEC_TEN <= 2;
        DEC_ONE <= NUMBER - 20;
    ELSIF NUMBER >= 10 THEN
        DEC_TEN <= 1;
        DEC_ONE <= NUMBER - 10;
    ELSE
        DEC_TEN <= 0;
        DEC_ONE <= NUMBER;
    END IF;
END PROCESS;
END A;

```

5) BCD2SEG.VHD

이 설계 파일에서는 DECTOSEG1.VHD에서 변환된 각 자리의 숫자 데이터를 가지고 최종적인 7-Segment에 출력하기 위한 decode 작업을 하게 된다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY BCD2SEG IS
PORT (
    -- 한 자리의 십진수 입력
    BCD : IN INTEGER RANGE 9 DOWNTO 0;
    -- 7-Segment로 변환된 값
    SEGMENT : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
);
END BCD2SEG;

ARCHITECTURE A OF BCD2SEG IS
BEGIN

PROCESS (BCD)
BEGIN
    IF BCD = 0 THEN
        SEGMENT <= "0111111";
    ELSIF BCD = 1 THEN
        SEGMENT <= "0000110";
    ELSIF BCD = 2 THEN
        SEGMENT <= "1011011";
    ELSIF BCD = 3 THEN
        SEGMENT <= "1001111";
    ELSIF BCD = 4 THEN
        SEGMENT <= "1100110";

```

```

ELSIF BCD = 5 THEN
    SEGMENT <= "1101101";
ELSIF BCD = 6 THEN
    SEGMENT <= "1111101";
ELSIF BCD = 7 THEN
    SEGMENT <= "0000111";
ELSIF BCD = 8 THEN
    SEGMENT <= "1111111";
ELSIF BCD = 9 THEN
    SEGMENT <= "1100111";
ELSE
    SEGMENT <= "0000000";
END IF;
END PROCESS;

END A;

```

6) DECTOSEG2.VHD

이 설계 파일은 DECTOSEG1.VHD와 비슷한 것으로 앞에서는 시간에 대한 각 자릿수 변환 작업만 정의한 것에 비해 이 부분에서는 분과, 초에 대한 각 자릿수 분리 작업을 하고 있다. 따라서 시간은 24까지 있는 것에 비해 분과 초는 60까지의 숫자를 숫자를 표현해 주고 있다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DECTOSEG2 IS
    PORT(
        NUMBER : IN INTEGER RANGE 59 DOWNT0 0;
        SEG_TEN : OUT STD_LOGIC_VECTOR (6 DOWNT0 0);
        SEG_ONE : OUT STD_LOGIC_VECTOR (6 DOWNT0 0)
    );
END DECTOSEG2;

ARCHITECTURE A OF DECTOSEG2 IS

    COMPONENT BCD2SEG
    PORT (
        BCD : IN INTEGER RANGE 9 DOWNT0 0;
        SEGMENT : OUT STD_LOGIC_VECTOR (6 DOWNT0 0)
    );
    END COMPONENT;

    SIGNAL DEC_TEN : INTEGER RANGE 9 DOWNT0 0;
    SIGNAL DEC_ONE : INTEGER RANGE 9 DOWNT0 0;

    BEGIN

        U0 : BCD2SEG
        PORT MAP (
            BCD => DEC_TEN,
            SEGMENT => SEG_TEN
        );

        U1 : BCD2SEG
        PORT MAP (
            BCD => DEC_ONE,
            SEGMENT => SEG_ONE
        );
    END A;

```

```

PROCESS( NUMBER )
BEGIN
  IF NUMBER >= 50 THEN
    DEC_TEN <= 5;
    DEC_ONE <= NUMBER - 50;
  ELSIF NUMBER >= 40 THEN
    DEC_TEN <= 4;
    DEC_ONE <= NUMBER - 40;
  ELSIF NUMBER >= 30 THEN
    DEC_TEN <= 3;
    DEC_ONE <= NUMBER - 30;
  ELSIF NUMBER >= 20 THEN
    DEC_TEN <= 2;
    DEC_ONE <= NUMBER - 20;
  ELSIF NUMBER >= 10 THEN
    DEC_TEN <= 1;
    DEC_ONE <= NUMBER - 10;
  ELSE
    DEC_TEN <= 0;
    DEC_ONE <= NUMBER;
  END IF;
END PROCESS;

END A;

```

7) SEG_MODULE.VHD

이 설계 파일은 이전에 decode 된 값을 7-Segment에 display하기 위한 부분이다. 일반적인 7-Segment에서는 하나당 8개의 핀을 통해 제어를 하게 되어 있지만 여기에서는 각 7-Segment 마다 데이터 핀을 공통으로 사용하고 있기 때문에 14개의 핀으로 6개의 7-Segment를 제어할 수 있도록 하드웨어가 구성되어 있다. 따라서 다음의 설계 파일에서 알 수 있듯이 스캐닝 방식을 통해 7-Segment를 구동하고 있다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY SEG_MODULE IS
PORT (
  CLK : IN STD_LOGIC;
  -- 7-Segment 선택 입력
  SEG_EN : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
  -- 7-Segment data 입력
  SEG_8 : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
  SEG_7 : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
  SEG_6 : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
  SEG_5 : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
  SEG_4 : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
  SEG_3 : IN STD_LOGIC_VECTOR (6 DOWNTO 0);

  -- 7-Segment 출력
  SEG_OUT : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
  -- 각 7-Segment 선택 핀
  COMMON : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END SEG_MODULE;

```

```

ARCHITECTURE A OF SEG_MODULE IS
SIGNAL CNT : INTEGER RANGE 7 DOWNTO 0;
BEGIN

```

```

PROCESS( CLK )

```

-- 8개의 7-Segment를 사용하므로 0 ~ 7까지 카운트 하기 위해 사용

```

VARIABLE CNT : INTEGER RANGE 7 DOWNTO 0;

```

```

BEGIN

```

```

  IF CLK'EVENT AND CLK = '1' THEN

```

```

    CNT := CNT + 1;

```

```

    CASE CNT IS

```

```

      WHEN 7 =>

```

```

        SEG_OUT <= '0' & SEG_8;

```

```

      WHEN 6 =>

```

```

        SEG_OUT <= '1' & SEG_7;

```

```

      WHEN 5 =>

```

```

        SEG_OUT <= '0' & SEG_6;

```

```

      WHEN 4 =>

```

```

        SEG_OUT <= '1' & SEG_5;

```

```

      WHEN 3 =>

```

```

        SEG_OUT <= '0' & SEG_4;

```

```

      WHEN 2 =>

```

```

        SEG_OUT <= '1' & SEG_3;

```

```

      WHEN 1 =>

```

```

        SEG_OUT <= "00000000";

```

```

      WHEN 0 =>

```

```

        SEG_OUT <= "00000000";

```

```

      WHEN OTHERS =>

```

```

        NULL;

```

```

    END CASE;

```

다음 조건문은 7-Segment의 common 값을 확인하여 common이 '1'이 되는 순간 7-Segment를 display 하고, '0'일 경우에는 display 하지 않게 된다. 그리고 SEG_EN 값이 '0'일 경우에 7-Segment값을 사용하지 않게 된다.

```

  IF CNT = 7 AND SEG_EN(7) = '1' THEN

```

```

    COMMON(7) <= '0';

```

```

  ELSE

```

```

    COMMON(7) <= '1';

```

```

  END IF;

```

```

  IF CNT = 6 AND SEG_EN(6) = '1' THEN

```

```

    COMMON(6) <= '0';

```

```

  ELSE

```

```

    COMMON(6) <= '1';

```

```

  END IF;

```

```

  IF CNT = 5 AND SEG_EN(5) = '1' THEN

```

```

    COMMON(5) <= '0';

```

```

  ELSE

```

```

    COMMON(5) <= '1';

```

```

  END IF;

```

```

  IF CNT = 4 AND SEG_EN(4) = '1' THEN

```

```

    COMMON(4) <= '0';

```

```

  ELSE

```

```

COMMON(4) <= '1';
END IF;

IF CNT = 3 AND SEG_EN(3) = '1' THEN
COMMON(3) <= '0';
ELSE
COMMON(3) <= '1';
END IF;

IF CNT = 2 AND SEG_EN(2) = '1' THEN
COMMON(2) <= '0';
ELSE
COMMON(2) <= '1';
END IF;

IF CNT = 1 AND SEG_EN(1) = '1' THEN
COMMON(1) <= '0';
ELSE
COMMON(1) <= '1';
END IF;

IF CNT = 0 AND SEG_EN(0) = '1' THEN
COMMON(0) <= '0';
ELSE
COMMON(0) <= '1';
END IF;
END IF;
END PROCESS;

END A;

```

8) COMP.VHD

이 설계 파일은 시간을 설정하는 작업이나 현재 시간이 2 ~ 5시 사이일 때 신호등이 점멸 되도록 만들어 주는 구문이다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY COMP IS
PORT ( HOUR : IN INTEGER RANGE 23 DOWNT0 0;
-- 설정 모드일 경우 'MODE'는 '00'의 값을 가진다.
MODE : IN STD_LOGIC_VECTOR (1 DOWNT0 0);
-- 신호 점멸을 위한 signal
FLICKER : OUT STD_LOGIC
);
END COMP;

ARCHITECTURE A OF COMP IS
BEGIN

PROCESS ( HOUR )
BEGIN
-- 시간이 2 ~ 5시일 경우
IF HOUR >= 2 AND HOUR <= 4 THEN
FLICKER <= '1';
-- 시간 설정 모드일 경우
ELSIF MODE /= "00" THEN
FLICKER <= '1';

```

```

ELSE
    FLICKER <= '0';
END IF;
END PROCESS;

END A;

```

9) TRAFFIC.VHD

이 설계 소스는 앞서 설계한 회로에서 나오는 제어 신호를 통해 교통 신호등과 보행자 신호등의 LED를 제어하기 위한 부분이다. 따라서 동, 서, 남, 북으로 바뀌어서 신호등의 불이 바뀌게 된다. 이렇게 시간에 따라 신호등의 불이 바뀌고 점멸하는 기능을 하고, 시간이 새벽 시간일 경우에는 차량 신호등에는 황색등만, 보행자 신호등에는 붉은색 등만 점등하도록 구성하고 있다.

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;

ENTITY TRAFFIC IS
PORT(
    -- 10KHz의 클럭입력
    CLK      : IN STD_LOGIC;
    -- 시간 조건에 의해 발생하는 점멸신호
    FLICKER  : IN STD_LOGIC;
    -- 강제 점멸 입력
    FLICK_SW : IN STD_LOGIC;
    -- 다음은 교통 신호등에서 동, 서, 남, 북에 있는 각각을 제어하기 위한 port 선언
    RED      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    YELLOW   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    GREEN    : BUFFER STD_LOGIC_VECTOR(3 DOWNTO 0);
    LEFT     : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    -- 보행자 신호등에서 동, 서, 남, 북에 있는 각각을 제어하기 위한 port 선언
    WALK_R   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    WALK_G   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
);
END TRAFFIC;

ARCHITECTURE ARC OF TRAFFIC IS

    -- 10KHz의 클럭을 1Hz로 분주하기 위한 변수
    SIGNAL CNT      : INTEGER RANGE 0 TO 4999;
    -- 신호등 바뀌는 시간을 카운트하기 위한 변수
    SIGNAL SCNT     : INTEGER RANGE 0 TO 12;
    -- 신호 방향의 변수(0:남, 1:서, 2:동, 3:북)
    SIGNAL DIRECTION : INTEGER RANGE 0 TO 3;
    -- SCNT 변수의 카운트용 클럭
    SIGNAL SCLK     : STD_LOGIC;
    -- DIRECTION 분주의 카운트용 클럭
    SIGNAL MCLK     : STD_LOGIC;

```

```
-- 점멸 신호용 변수
```

```
SIGNAL S_FLICKER : STD_LOGIC;
```

```
BEGIN
```

다음 PROCESS문에서는 점멸 버튼이 눌러지면 점멸 신호를 토글한다.

```
PROCESS (FLICKER, FLICK_SW)
```

```
BEGIN
```

```
IF FLICK_SW = '1' AND FLICK_SW'EVENT THEN
```

```
    S_FLICKER <= NOT S_FLICKER;
```

```
END IF;
```

```
END PROCESS;
```

다음 PROCESS문은 10KHz 클럭을 분주하여 1Hz의 클럭을 생성하는 구문이다.

```
PROCESS (CLK)
```

```
BEGIN
```

```
IF CLK = '1' AND CLK'EVENT THEN
```

```
    IF CNT = 4999 THEN
```

```
        CNT <= 0;
```

```
        SCLK <= NOT SCLK;
```

```
    ELSE
```

```
        CNT <= CNT + 1;
```

```
    END IF;
```

```
END IF;
```

```
END PROCESS;
```

다음은 PROCESS 문은 신호등의 한 방향이 진행되는 시간을 카운트하는 구문이다.

```
PROCESS (S_FLICKER, SCLK)
```

```
BEGIN
```

```
-- 점멸 버튼이나 시간에 의한 점멸이 발생할 경우 카운트 값을 클리어 한다.
```

```
IF S_FLICKER = '1' OR FLICKER = '1' THEN
```

```
    SCNT <= 0;
```

```
    MCLK <= '0';
```

```
ELSIF SCLK = '1' AND SCLK'EVENT THEN
```

```
    IF SCNT = 12 THEN
```

```
        SCNT <= 0;
```

```
        MCLK <= '1';
```

```
    ELSE
```

```
        SCNT <= SCNT + 1;
```

```
        MCLK <= '0';
```

```
    END IF;
```

```
END IF;
```

```
END PROCESS;
```

다음 PROCESS문은 신호의 방향을 전환하기 위한 구문이다.

```
PROCESS (S_FLICKER, MCLK)
```

```
BEGIN
```

```
IF S_FLICKER = '1' OR FLICKER = '1' THEN
```

```
    DIRECTION <= 0;
```

```
ELSIF MCLK = '1' AND MCLK'EVENT THEN
```

```
    IF DIRECTION = 3 THEN
```

```
        DIRECTION <= 0;
```

```
    ELSE
```

```
        DIRECTION <= DIRECTION + 1;
```

```
    END IF;
```

```
END IF;
```

```
END PROCESS;
```

다음 PROCESS문은 동, 서, 남, 북의 신호등을 실제적으로 제어하는 구문이다.

```
PROCESS (S_FLICKER, DIRECTION, SCNT, SCLK)
BEGIN
-- 점멸 상태가 될 경우 황색등가 보행자 적색등을 점멸한다.
IF S_FLICKER = '1' OR FLICKER = '1' THEN
  RED <= "0000";
  GREEN <= "0000";
  YELLOW <= SCLK & SCLK & SCLK & SCLK;
  WALK_R <= SCLK & SCLK & SCLK & SCLK;
  WALK_G <= "0000";
ELSE
CASE DIRECTION IS
  -- 남쪽 신호등 제어
  WHEN 0 =>
  -- 10초 동안 녹색등을 점등한다.
    IF SCNT <= 10 THEN
      RED <= "0111";
      GREEN <= "1000";
      YELLOW <= "0000";

  -- 2초 동안 황색등을 점등한다.
    ELSIF SCNT <= 12 THEN
      RED <= "0111";
      GREEN <= "0000";
      YELLOW <= "1000";
  -- 예외의 조건이 발생할 경우 황색등이 점멸한다.
    ELSE
      RED <= "0000";
      GREEN <= "0000";
      YELLOW <= SCLK & SCLK & SCLK & SCLK;
    END IF;

  -- 3초 동안 보행자 녹색등을 점등한다.
    IF SCNT <= 3 THEN
      WALK_R <= "1011";
      WALK_G <= "0100";
  -- 7초 동안 보행자 녹색등을 점멸한다.
    ELSIF SCNT <= 10 THEN
      WALK_R <= "1011";
      WALK_G <= '0' & (NOT SCLK) & "00";
  -- 보행자 적색등을 점등한다.
    ELSIF SCNT <= 12 THEN
      WALK_R <= "1111";
      WALK_G <= "0000";
  -- 예외의 조건이 발생할 경우 보행자 적색등을 점멸한다.
    ELSE
      WALK_R <= "0000";
      WALK_G <= SCLK & SCLK & SCLK & SCLK;
    END IF;

  -- 서쪽 신호 제어
  WHEN 1 =>
    IF SCNT <= 10 THEN
      RED <= "1011";
      GREEN <= "0100";
```

```

        YELLOW <= "0000";
    ELSIF SCNT <= 12 THEN
        RED <= "1011";
        GREEN <= "0000";
        YELLOW <= "0100";
    ELSE
        RED <= "0000";
        GREEN <= "0000";
        YELLOW <= SCLK & SCLK & SCLK & SCLK;
    END IF;

    IF SCNT <= 3 THEN
        WALK_R <= "1101";
        WALK_G <= "0010";
    ELSIF SCNT <= 10 THEN
        WALK_R <= "1101";
        WALK_G <= "00" & (NOT SCLK) & '0';
    ELSIF SCNT <= 12 THEN
        WALK_R <= "1111";
        WALK_G <= "0000";
    ELSE
        WALK_R <= "0000";
        WALK_G <= SCLK & SCLK & SCLK & SCLK;
    END IF;

```

-- 동쪽 신호 제어

WHEN 2 =>

```

    IF SCNT <= 10 THEN
        RED <= "1101";
        GREEN <= "0010";
        YELLOW <= "0000";
    ELSIF SCNT <= 12 THEN
        RED <= "1101";
        GREEN <= "0000";
        YELLOW <= "0010";
    ELSE
        RED <= "0000";
        GREEN <= "0000";
        YELLOW <= SCLK & SCLK & SCLK & SCLK;
    END IF;

    IF SCNT <= 3 THEN
        WALK_R <= "1110";
        WALK_G <= "0001";
    ELSIF SCNT <= 10 THEN
        WALK_R <= "1110";
        WALK_G <= "000" & (NOT SCLK);
    ELSIF SCNT <= 12 THEN
        WALK_R <= "1111";
        WALK_G <= "0000";
    ELSE
        WALK_R <= "0000";
        WALK_G <= SCLK & SCLK & SCLK & SCLK;
    END IF;

```

-- 북쪽 신호 제어

WHEN 3 =>

```

    IF SCNT <= 10 THEN
        RED <= "1110";

```

```

        GREEN <= "0001";
        YELLOW <= "0000";
    ELSIF SCNT <= 12 THEN
        RED <= "1110";
        GREEN <= "0000";
        YELLOW <= "0001";
    ELSE
        RED <= "0000";
        GREEN <= "0000";
        YELLOW <= SCLK & SCLK & SCLK & SCLK;
    END IF;

    IF SCNT <= 3 THEN
        WALK_R <= "0111";
        WALK_G <= "1000";
    ELSIF SCNT <= 10 THEN
        WALK_R <= "0111";
        WALK_G <= (NOT SCLK) & "000";
    ELSIF SCNT <= 12 THEN
        WALK_R <= "1111";
        WALK_G <= "0000";
    ELSE
        WALK_R <= "0000";
        WALK_G <= SCLK & SCLK & SCLK & SCLK;
    END IF;
    END CASE;
END IF;
END PROCESS;

-- 차량의 좌회전 신호는 직진 신호와 같이 동작 된다.
LEFT <= GREEN;

END ARC;

```

3. 입출력핀 구성표

Traffic Lights 모듈 연결 커넥터의 핀 구성

확장 모듈 PCB 실제 이름 및 핀 번호		HBE-COMBO II 보드로 연결할 때의 포트 및 핀 번호			
핀 이름	핀 번호	EP2C35	EP2C50~7 0	XC3S1000 ~4000	포트 및 핀 번호
VCC (+ 5V)	JP-1	-	-	-	EXT3-1
VCC (+ 5V)	JP-2	-	-	-	EXT3-2
VCC (+ 12V)	JP-3	-	-	-	EXT3-3
VCC (+ 12V)	JP-4	-	-	-	EXT3-4
N_YELLOW	JP-5	H21	H21	L20	EXT3-5
WALK_M_GREEN	JP-6	H19	AA20	L19	EXT3-6
N_RED	JP-7	G24	G24	K22	EXT3-7
WALK_N_RED	JP-8	G23	G23	K21	EXT3-8
WALK_W_GREEN	JP-9	G26	G26	K24	EXT3-9
N_GREEN	JP-10	G25	G25	K23	EXT3-10
WALK_W_RED	JP-11	F26	F26	J24	EXT3-11
N_LEFT	JP-12	F25	F25	J23	EXT3-12
E_GREEN	JP-13	G22	G22	K20	EXT3-13
W_GREEN	JP-14	G21	G21	J25	EXT3-14
E_LEFT	JP-15	F21	F21	J20	EXT3-15
W_LEFT	JP-16	F20	F20	H26	EXT3-16
E_YELLOW	JP-17	F24	F24	J22	EXT3-17
W_YELLOW	JP-18	F23	F23	J21	EXT3-18
E_RED	JP-19	E24	E24	H23	EXT3-19
W_RED	JP-20	E23	E23	H22	EXT3-20
WALK_E_GREEN	JP-21	E26	E26	H25	EXT3-21
S_GREEN	JP-22	E25	E25	H24	EXT3-22
WALK_E_RED	JP-23	D25	D25	E24	EXT3-23
S_LEFT	JP-24	D23	D23	E23	EXT3-24
WALK_S_GREEN	JP-25	E22	E22	H21	EXT3-25
S_YELLOW	JP-26	D26	D26	H20	EXT3-26
WALK_S_RED	JP-27	B25	B25	D26	EXT3-27
S_RED	JP-28	B24	B24	D25	EXT3-28
Not Connector		JP42 ~ JP48			
GND	JP-49				EXT3-49
GND	JP-50				EXT3-50

4. 회로도

