Traffic Light Controller with use of System Timer

Design a traffic controller system that uses the PIC. The system will have:

I) four RGB LEDs (called NS LIGHT, NSLT LIGHT, EW LIGHT, and EWLT LIGHT)

II) four input sensors:

a) NSLT_SW : to detect cars that want to make a left-turn on NS direction
b) EWLT_SW: for detection of cars wanting to make a left-turn on EW direction
c) NSPED_SW: switch pressed by pedestrians wanting to make a cross in the NS direction
d) EWPED_SW: switch pressed by pedestrians wanting to make a cross in the EW direction

III) two sets of 7-segment display to show the number of seconds left for pedestrian to cross

IV) A buzzer to make sound helping people with blindness to cross the streets

V) a photo sensor (photo resistor) to sense whether the traffic controller is operating in Day Mode or Night Mode.

Before we get into the design of the controller system we need to build some small routines that will help us in the final design.

PART 1)

Write the routine Wait_One_Second() that will wait like the name indicates for 1 second. Actually, this routine will call another routine Wait_Half_Second() twice. In between the two calls, asserts the setting and the resetting the LED called 'SEC_LED'. Refer to the schematics for the port bit that this LED is connected to.

```
void Wait_One_Second()
{
        SEC_LED = 0;                        // First, turn off the SEC LED
        Wait_Half_Second();                 // Wait for half second (or 500 msec)
        SEC_LED = 1;                        // then turn on the SEC LED
        Wait_Half_Second();                 // Wait for half second (or 500 msec)
}

void Wait_Half_Second()
{
        T0CON = 0x02                        // Timer 0, 16-bit mode, prescaler 1:8
        TMR0L = 0x??;                       // set the lower byte of TMR
        TMR0H = 0x??;                       // set the upper byte of TMR
        INTCONbits.TMR0IF = 0;              // clear the Timer 0 flag
        T0CONbits.TMR0ON = 1;               // Turn on the Timer 0

        while (INTCONbits.TMR0IF == 0);  // wait for the Timer Flag to be 1 for done
        T0CONbits.TMR0ON = 0;               // turn off the Timer 0
}
```

The 'Wait_Half_Second()' routine above uses the hardware timer to do the delay function. Through the Timer 0, a timer value is loaded and the timer will count from that value to the 0xFFFF. A Timer Flag (TMR0IF) will be set to '1' when 0xFFFF is reached. The software will wait until that flag is set and that will indicate that the wait is over. You need to specify the value of the TMR0L and TMR0H above so that the 'Wait_Half_Second()' lasts exactly one half of a second or 500 msec. Let us go over the calculation of the value to be loaded into the TMR0.

The system clock will run at 4 MHz (make sure to set OSCCON = 0x60). This implies that the timer clock will run a 1 MHz (1/4 of system clock). The period of the timer is then 1 usec. To achieve a wait time of 500 msec, it will take a count of 500 msec / 1 usec = 500, 000 count. The code of the Wait_Half_Second() routine specifies the use of a 1/8 prescaler. This will divide the 500,000 count by 8 giving a new count of 62500. The next step is to convert this number into a hex value and uses it to subtract from the value 0xFFFF. The resulting number is then the one to be loaded into the TMR0 register. The upper byte will go to TMR0H while the lower one goes to TMR0L.

Write the program with an infinite loop calling the Wait_One_Second() routine. Put a scope on the signal 'SEC_LED' and measure its width to be 500 msec or the period must be 1 sec. Make any kind of adjustment to the TMR0 register (especially the TMR0L) to get the right period.

Use the '#define' statement to assign the bit location for the variable SEC_LED. For example:

```
#define SEC_LED PORTEbits.RE2
```

PART 2)

After the 'Wait_One_Second()' routine is implemented, write the next support routine to be called 'Wait_N_Seconds(char seconds)' that will use the 'Wait_One_Second()' function to delay for N seconds:

```
void Wait_N_Seconds (char seconds)
{
char I;
        for (I = 0; I< seconds; I++)
        {
                Wait_One_Second();
        }
}
```

We will use this new routine for the main development.

PART 3)

Use the '#define' statement, create the following eight signal names: NS-RED, NS-GREEN, NSLT-RED, NSLT-GREEN, EW-RED, EW-GREEN, EWLT-RED, EWLT-GREEN. Use the schematics to determine the bit definitions. For example:

```
#define NS-RED       PORTAbits.RA5
#define NS-GREEN   PORTAbits.RA6
```

Do the same for the other three RGB LEDs. Refer to the schematics for pins assignments.

Next, write a routine to set the proper color for one RGB LED. For example, let us write a routine to set the color for the NS RGB:

```
void Set_NS(char color)
{
        switch (color):
                case OFF: NS-RED =0;NS-GREEN=0;break;
                case RED: NS-RED =1;NS-GREEN=0;break;
                case GREEN: NS-RED =0;NS-GREEN=1;break;
                case YELLOW: NS-RED =1;NS-GREEN=1;break;
}
```

where 'color' can have four values:

0: OFF
1: RED
2: GREEN
3: YELLOW

(use #define to declare the value of those colors)

Once completed, you can do the same for the remaining three RGB LEDs and call the three additional routines as:

1) Set_NSLT(char color)
2) Set_EW(char color)
3) Set_EWLT(char color)

For testing purpose, once these four routines are written, generate a program that has an infinite loop to call all the four routines each displaying all the three colors. Do space each color with a call to the Wait_N_Seconds() (defined in Part 2).

For example:

```
while (1)
{
        for (int i=0, i<4;i++)
        {
                SET_NS(i);              // Set color for North-South direction
                SET_NSLT(i);            // Set color for North-South Left-Turn direction
                SET_EW(i);              // Set color for East-West direction
                SET_EWLT(i);            // Set color for East-West Left-Turn direction
                Wait_N_Second(1);   // call Wait-N-Second routine to wait for 1 second
        }
}
```

This exercise will allow your team to check the connections of the four sets of RGB LEDs.

PART 4)

Write a routine that will control the operation of the only 7-segment display. The name of the routine should be called PED_Control() and it should have two arguments 'Direction' and 'Num_Sec'. Here is the definition of the routine:

Void PED_Control(char Direction, char Num_Sec)

The variable 'Direction' will select which direction the pedestrian counter is addressed for.

If 'Direction' is set to '0', it will apply for the North-South direction. The lower digit of the 2-digit 7-segment will be turned off while the upper digit will start with the number indicated by the second parameter 'Num_Sec'. Take the value from that variable subtract 1 from it and display the result on to the 7-segment display. At the start, the display will show the counter to be at 'Num_Sec' – 1. Then, on every second, the count will start to decrement by 1 and it will continue to do so until the count reaches 1. On the next second, the display will be completely turned off. The number of seconds elapsed between the number 'Num_Sec' – 1 and when the display is turned off is actually 'Num_Sec'. For example, if 'Num_Sec' is 5, then the display will start from 4, 3, 2, 1 and off. The total is then 5 seconds.

If the 'Direction' is set to '1', then this will apply to the East-West direction. This time the upper digit will be turned off while the lower digit will start counting from 'Num_Sec' – 1 to 1 and then get turned off 1 second later.

While counting in either direction, an audible sound through the buzzer must be generated to indicate that the pedestrian counter is active. To achieve this task while waiting for a second, generate the new 'Wait_One_Second_With_Beep()' below and use it for the wait.

```
void Wait_One_Second_With_Beep()
{
        SEC_LED = 1;                    // First, turn on the SEC LED
        Activate_Buzzer()               // Activate the buzzer
        Wait_Half_Second();             // Wait for half second (or 500 msec)
        SEC_LED = 0;                    // then turn off the SEC LED
        Deactivate_ Buzzer ();          // Deactivate the buzzer
        Wait_Half_Second();             // Wait for half second (or 500 msec)
}
```

Note: when one digit is counting in one direction, the other digit must be turned off. When the counting is done, both digits must be off.

After that routine is properly written, write a test program to check its operation. For example, do an infinite loop to check all the possible combinations:

```
while (1)
{
        PED_Control (0, 8)              // Set direction 0 and do for 8 seconds
        PED_Control(1, 6)               // Set direction 1 for 6 seconds
}
```

PART 5)

This is the main design for the traffic controller.

We have a total of four DIP-Switch inputs: NSLT SW, NSPED SW, EWLT SW, and EWPED SW (see schematics for pin assignments) and a light sensor input MODE.

The EWLT SW and NSLT SW inputs are the sensors (switches) to indicate that a car is requesting to make a left-turn on the direction indicated by the switch's name: EW is for the East-West direction while NS is for the North-South. When a sensor is 1, there is a car waiting to make a left turn.

The EWPED SW and NSPED SW inputs are the sensors used to detect the presence of pedestrians wanting to cross. When the input is '0', no pedestrian is present to cross. When the input is '1', there is at least a pedestrian wishing to cross.

The input 'MODE', as mentioned before, is a light sensor used to determine the mode of operation. It is implemented through the use of the photo resistor PR1. When the voltage at PR1 is below 2.5V, the MODE is for the Day Time Mode. When it is above 2.5V, the MODE is Night Time Mode. The 'MODE_LED' is turned on when the day mode is on and it will be turned off for the night mode.

A while infinite loop will scan the logic state of the light sensor and based on that logic state the program will call either the routine 'void Day_Mode()' or 'void Night_Mode()'

Traffic Light Night Time Mode:

This is the Mode that operates when at night whereas no Pedestrian Counter is used. **The two 7-segment displays must be off during the entire process.** This is the simplest mode:

Implement the following steps:

1) Next, set the NS RGB to GREEN while the other RGB LEDs remain in RED.

2) Wait for 7 seconds with NS RGB still in GREEN. Then go to step 3). Use the 'Wait_N_Seconds' routine to do the delay.

3) Set the NS RGB to YELLOW and wait for 3 seconds. Go to step 4)

4) Next, change NS RGB to RED and go to step 5)

5) Check the logic state of EWLT switch:

    a. If 0, go directly to step 9)

b. If 1, there is a left-turn request. Go to step 6)

6) Turn on EWLT RGB to GREEN and leave it for 7 seconds. Afterwards, go to step 7)

7) Change EWLT RGB to YELLOW. Wait for 3 seconds and go to step 8)

8) Turn off EWLT RGB to RED. Go to step 9)

9) Turn EW RGB to GREEN and wait for 6 seconds. Next, go to step 10)

10) The EW RGB will be changed to YELLOW and it stays on for 3 seconds. Go to step 11)

11) Change EW RGB to RED. Go to step 12)

12) Check the logic state of the NSLT switch.

    a. If 0, go back directly to step 16)
    b. If 1, go to step 13)

13) Turn on NSLT RGB to GREEN and leave it on for 8 seconds. Afterwards, go to step 14)

14) Change NSLT RGB to YELLOW. Wait for 3 seconds. Go to step 15)

15) Turn NS-LT RGB to RED. Go to step 16)

16) This completes the sequence.

Traffic Light Day Time Mode:

In this mode, the use of the Pedestrian Counter is added as well as the two inputs EWPED and NSPED.

The design will incorporate the following implementation:

1. Next, set the NS RGB to GREEN while the other LEDs remain in RED. Read in the logic state of the NSPED switch:

    a. If '1', then do the Pedestrian countdown for the NS direction and wait for 9 seconds (use the PED_Control() routine for this task). When done, go to step 2)

    b. If the input is '0', then go directly to step 2)

2. Wait for 8 seconds with NS RGB still in GREEN. Then go to step 3)

3. Set the NS RGB to YELLOW and wait for 3 seconds. Go to step 4)

4. Next, change NS RGB to RED and go to step 5)

5. Check the logic state of EWLT switch:

    a. If 0, go directly to step 9)
    b. If 1, there is a left-turn request. Go to step 6)

6. Turn on EWLT RGB to GREEN and leave it for 7 seconds. Afterwards, go to step 7)

7. Change EWLT RGB to YELLOW. Wait for 3 seconds and go to step 8)

8. Turn off EWLT RGB to RED. Go to step 9)

9. Turn on EW RGB to GREEN. Read in the logic of EWPED switch.

    a. If '1', then do the Pedestrian Countdown for the EW direction and wait for 8 seconds. When completed, go to step 10)

    b. If '0', then go to step 10) directly.

10. The EW RGB is GREEN and stays for 7 seconds. Afterwards, go to step 11)

11. The EW RGB will be changed to YELLOW and it stays on for 3 seconds. Go to step 12)

12. Change EW RGB to RED. Go to step 13)

13. Check the logic state of the NSLT switch.

    a. If 0, go back directly to step 1)
    b. If 1, there is a left-turn request. Go to step 14)

14. Turn on NSLT RGB to GREEN and leave it on for 8 seconds. Afterwards, go to step 15)

15. Change NSLT RGB to YELLOW. Wait for 3 seconds. Go to step 16)

16. Turn NSLT RGB to RED. Go to step 17)

17. The traffic cycle is complete. This is the end of the sequence.