TRAFFIC LIGHT USING PIC16F877A

Name : Abanoub Adel Gerges

B.N : 2

2nd year communication and computer science department

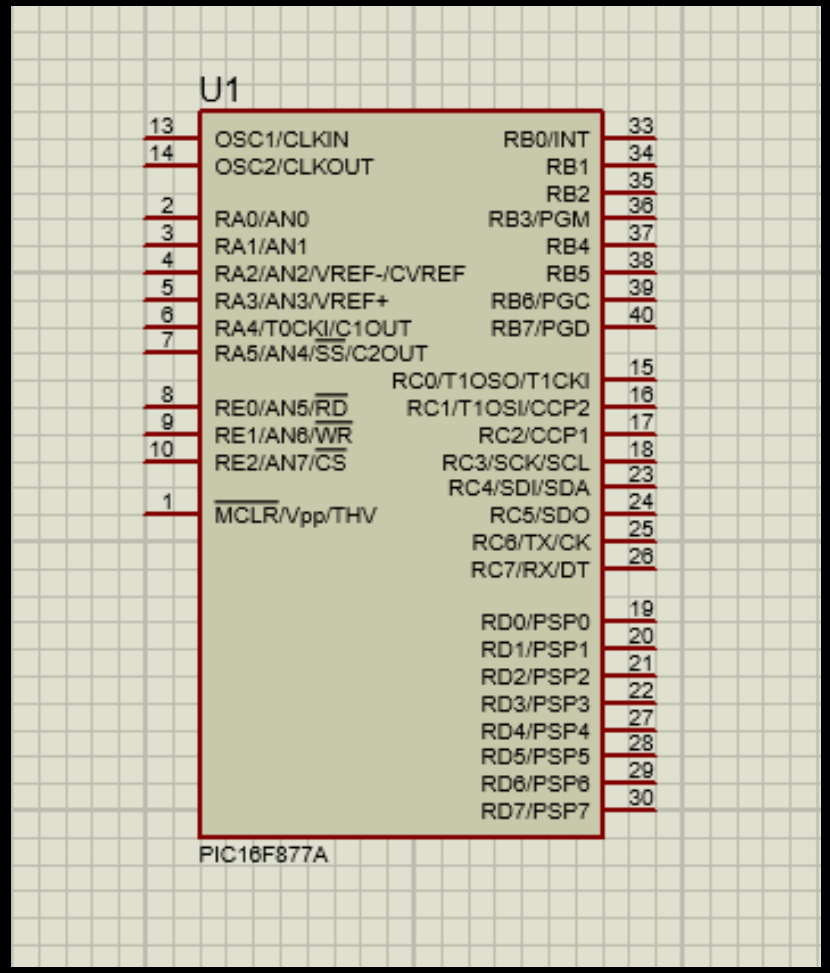
PINS

1- Power Supply Pins:

- VDD (pin 11) - Positive supply voltage
- VSS (pins 12, 31) - Ground reference

2- Oscillator Pins:

- OSC1/CLKIN (pin 13) - Input to the internal oscillator circuit
- OSC2/CLKOUT (pin 14) - Output from the internal oscillator circuit



PINS

3- Input/Output (I/O) Pins:

- Port A (RA0 to RA5) - 6 I/O pins
- Port B (RB0 to RB7) - 8 I/O pins
- Port C (RC0 to RC7) - 8 I/O pins
- Port D (RD0 to RD7) - 8 I/O pins
- Port E (RE0 to RE2) - 3 I/O pins

4- Reset Pin:

- MCLR/VPP (pin 1) - Master Clear (Reset) input or programming voltage input

PINS

- Analog Pins:
  - RA0/AN0 to RA5/AN5 - Analog input pins for the 10-bit Analog-to-Digital Converter (ADC)
- Communication Pins:
  - RC6/TX/CK - Asynchronous transmit pin or synchronous clock pin
  - RC7/RX/DT - Asynchronous receive pin or synchronous data pin
  - RB0/INT0 - External interrupt 0 input pin
  - RB1/INT1 - External interrupt 1 input pin
  - RB2/INT2 - External interrupt 2 input pin
  - RB3/PGM - Low-voltage programming pin
  - RB4/KBI0 - Keyboard interrupt 0 input pin
  - RB5/KBI1 - Keyboard interrupt 1 input pin
  - RB6/PGC - In-Circuit Debugger/Programmer Clock
  - RB7/PGD - In-Circuit Debugger/Programmer Data

MAIN BLOCKS

1- Arithmetic Logic Unit (ALU):

The ALU is the central processing unit of the microcontroller. It performs arithmetic and logical operations on data, such as addition, subtraction, and logical operations like AND, OR, and NOT. The ALU is responsible for executing the instructions fetched from the program memory.

2- Status and Control:

This block manages the status flags that indicate the outcome of operations performed by the ALU, such as the Zero (Z) flag, Carry (C) flag, and Digital Carry (DC) flag. The control unit also manages the flow of instructions and data within the microcontroller.

MAIN BLOCKS

3-Program Counter:

The program counter is a register that holds the address of the next instruction to be fetched from the program memory. It is incremented after each instruction fetch to maintain the correct program flow.

4- Flash Program Memory:

The PIC16F877A has 8KB of Flash program memory, which is used to store the program code. The program memory is non-volatile, meaning the code is retained even when the power is turned off.

5- Instruction Register:

The instruction register holds the current instruction being executed by the microcontroller. The instruction is fetched from the program memory and loaded into the instruction register.

MAIN BLOCKS

- 6- Instruction Decoder:
  - The instruction decoder interprets the instruction stored in the instruction register and generates the appropriate control signals to execute the instruction. It coordinates the operation of the various blocks within the microcontroller to perform the desired task.

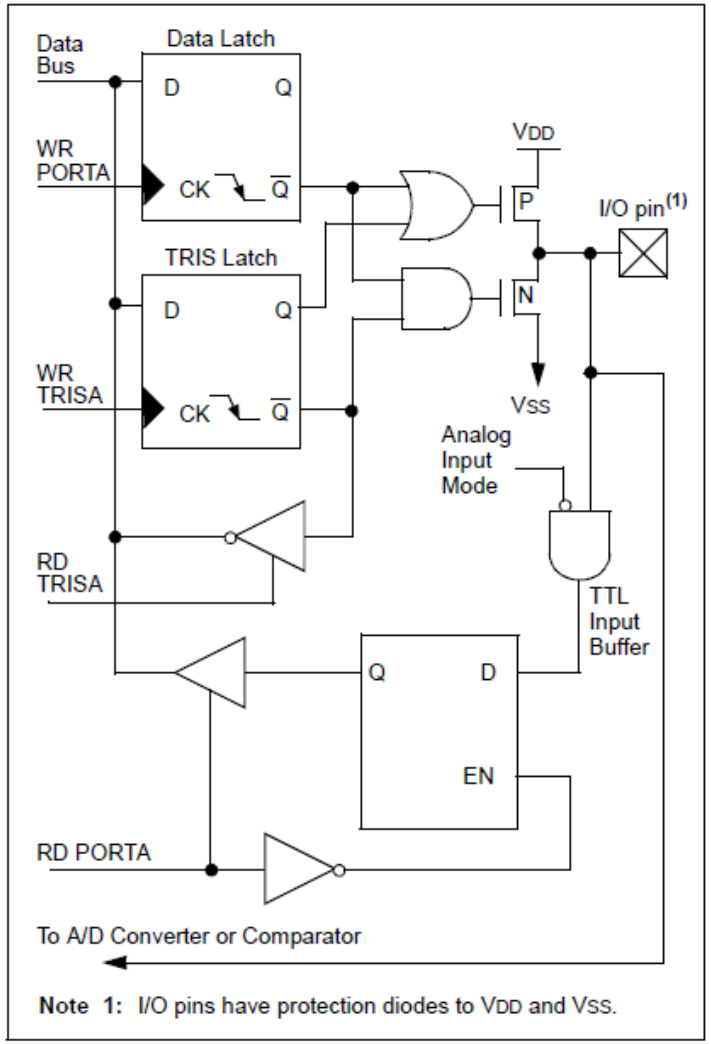
RA4

In pins RA3:RA0 there is 2 transistors PMOS && NMOS (data sheet page 41)

When you write in code that a pin is 0 both transistors will have 0 input so PMOS will be open circuit and NMOS will be short circuit and the final output from the pin will be 0V this will work as a sink

When you write in code that a pin is 1 both transistors will have 1 input so PMOS will be short circuit and NMOS will be open circuit and the final output from the pin will be 5V this will work as a source

FIGURE 4-1: BLOCK DIAGRAM OF RA3:RA0 PINS



Note 1: I/O pins have protection diodes to VDD and VSS.

RA4

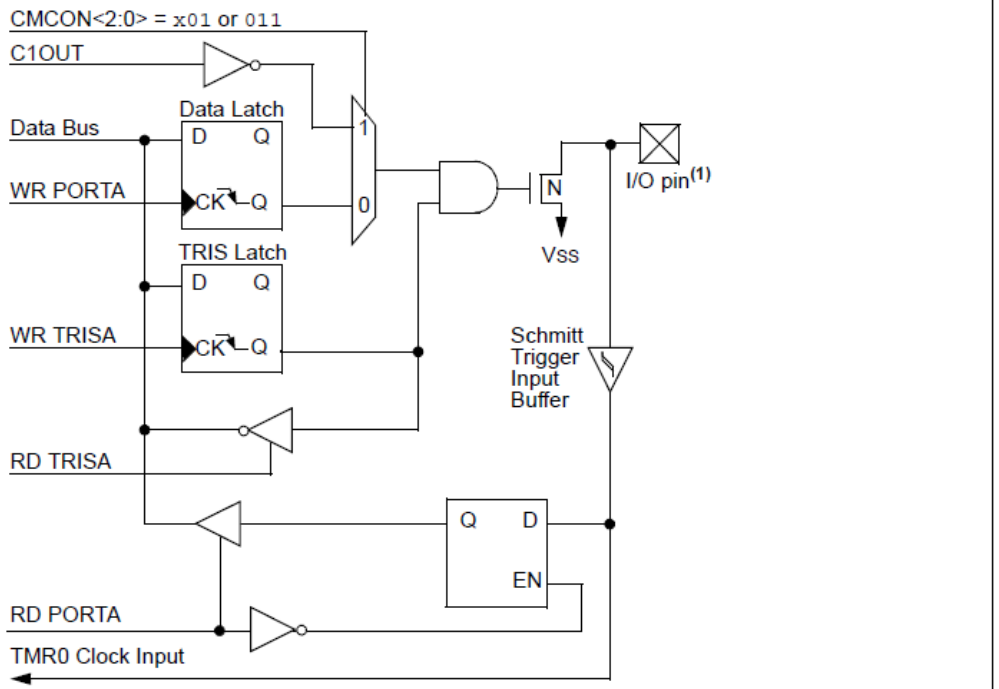
In pin RA0 there is only NOMS (open drain)

The following figure from data sheet page 42

When you write in code that a pin is 0 NMOS will be short circuit and the final output from the pin will be 0V this will work as a sink

When you write in code that a pin is 1 NMOS will be open circuit and the final output from the pin will be unknown

FIGURE 4-2: BLOCK DIAGRAM OF RA4/T0CKI PIN



Note 1: I/O pin has protection diodes to Vss only.

RA4

- Pin RA4 can be connected to 8.5V as long as current don’t exceed 25 mA (data sheet page 173)

- RA4 is open drain high voltage (data sheet page 179)

ATMEGA328P VS PIC16F877A

- Memory Size:
  - ATMega328P: 32KB of Flash program memory, 2KB of SRAM, and 1KB of EEPROM.
  - PIC16F877A: 8KB of Flash program memory, 368 bytes of SRAM, and 256 bytes of EEPROM
- Power Consumption:
  - ATMega328P: Operates at a wide range of voltages (1.8V to 5.5V) and has low power modes for power savings.
  - PIC16F877A: Operates at 2V to 5.5V and has a sleep mode for power savings, but the power consumption is generally higher than the ATMega328P.

ATMEGA328P VS PIC16F877A

- Pin Count:
  - ATMega328P: 28 pins
  - PIC16F877A: 40 pins

ATMEGA328P VS PIC16F877A

- Examples of Embedded Systems:

1- Battery-Powered Devices:

- The ATMega328P would be a better choice for battery-powered embedded systems due to its lower power consumption and wider range of operating voltages. This makes it suitable for devices like wearables, IoT sensors, and remote monitoring applications where power efficiency is crucial.

2- Small-Scale Projects:

The ATMega328P may be a better choice for small-scale embedded projects that don't require a large number of I/O pins. The ATMega328P's 28-pin package and lower pin count can simplify the hardware design and PCB layout, making it a suitable option for projects like Arduino-based prototypes, simple control systems, and hobby electronics.

ATMEGA328P VS PIC16F877A

- In contrast, the PIC16F877A may be more suitable for embedded systems that require a larger number of I/O pins, more program memory, or better performance in certain specific applications. Examples could include industrial control systems, automotive electronics, or embedded systems with more complex peripherals and interfaces.

- When choosing between the ATMega328P and the PIC16F877A, factors like power consumption, memory requirements and the specific needs of the embedded system should be carefully evaluated to determine the most appropriate microcontroller for the application.

TRAFFIC LIGHT PROJECT

In this project we will use PIC16f877A and connect it with crystal and button for MCLR to restart the program if you want

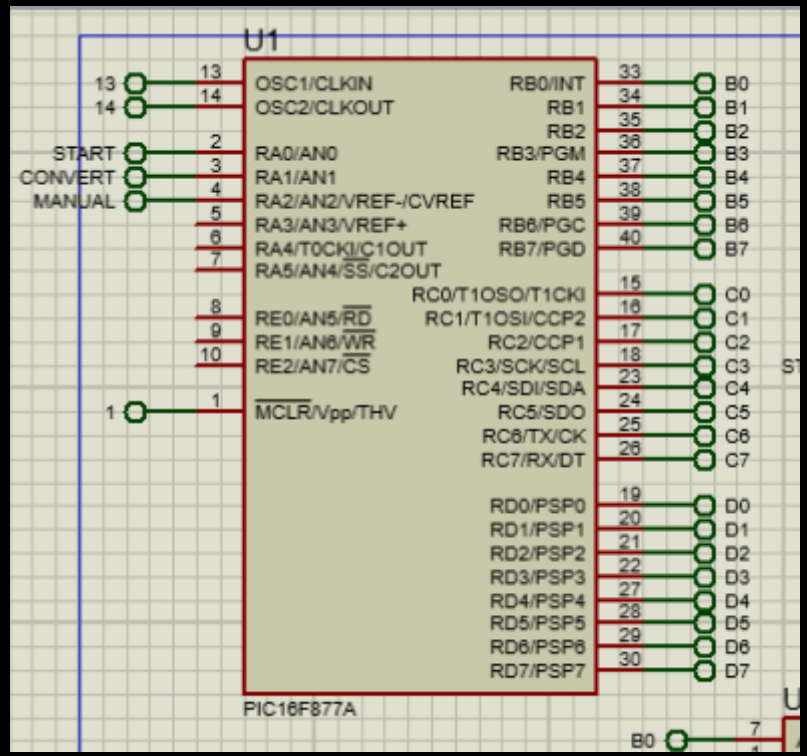
Let's made another three buttons

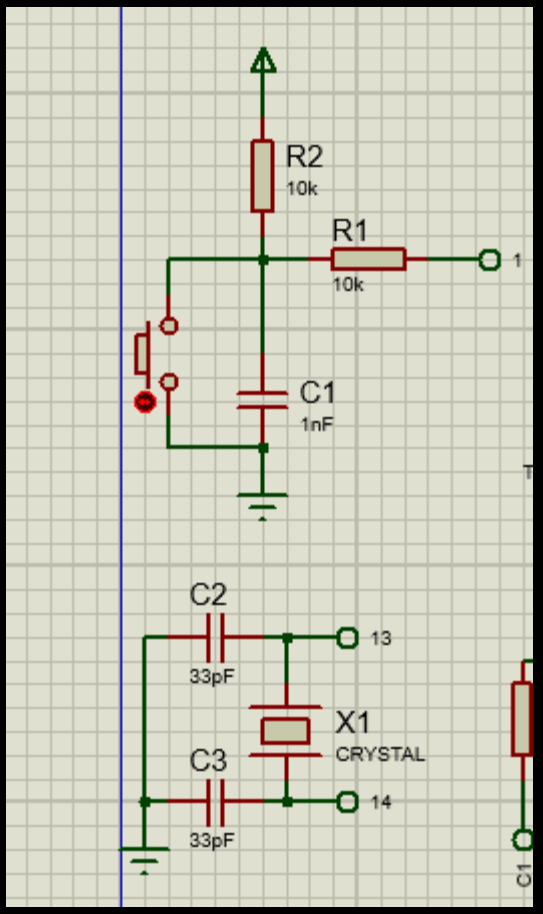
RA0 => start

RA1 => convert

RA2 => manual

Focus in their names because we will use them again in the code we will discuss that later





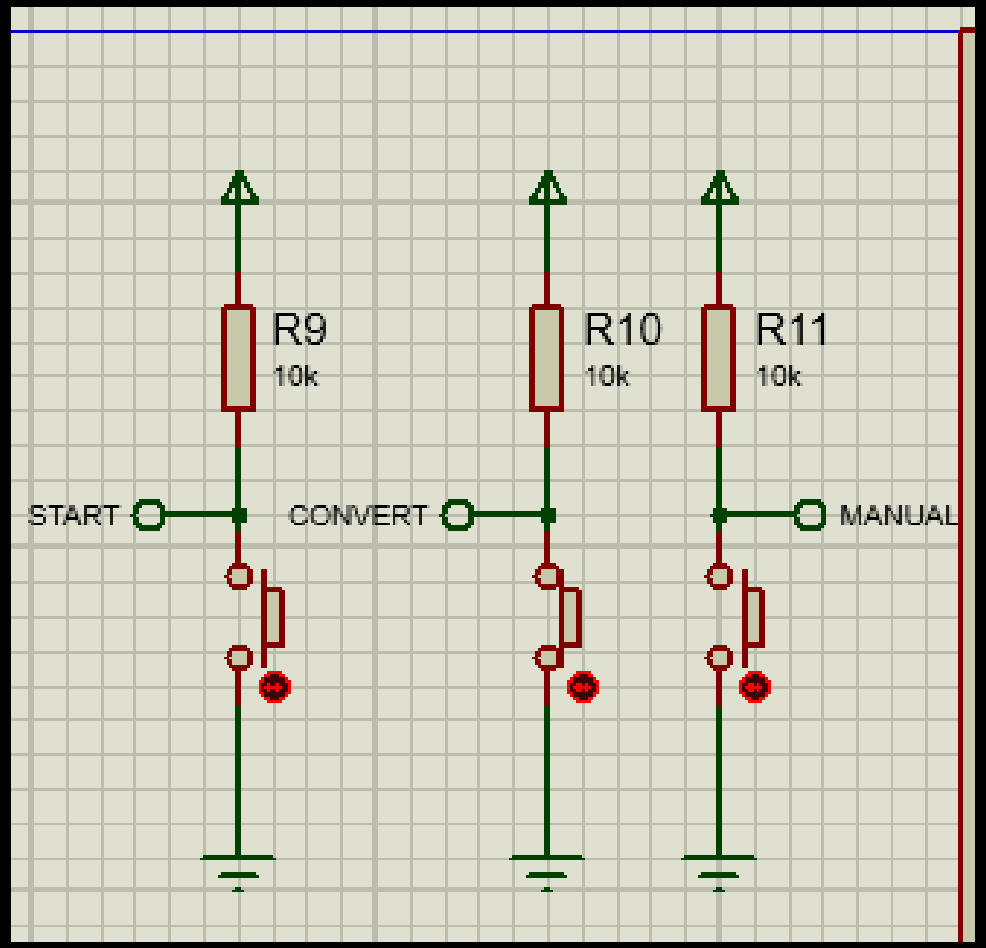
TRAFFIC LIGHT PROJECT

When you start the program the counter and traffic light won't start until you click start (counter will display 00)

If you want to switch between manual and automatic mode you should click a long click (as an figure) in convert button, while you are pressing that button counter won't change (00)

When convert button is up you are in the automatic mode, otherwise you are in the manual mode (figure is in the manual mode)

When you click in manual (in manual mode) the counter will count for 3 seconds and then switch southern and western roads

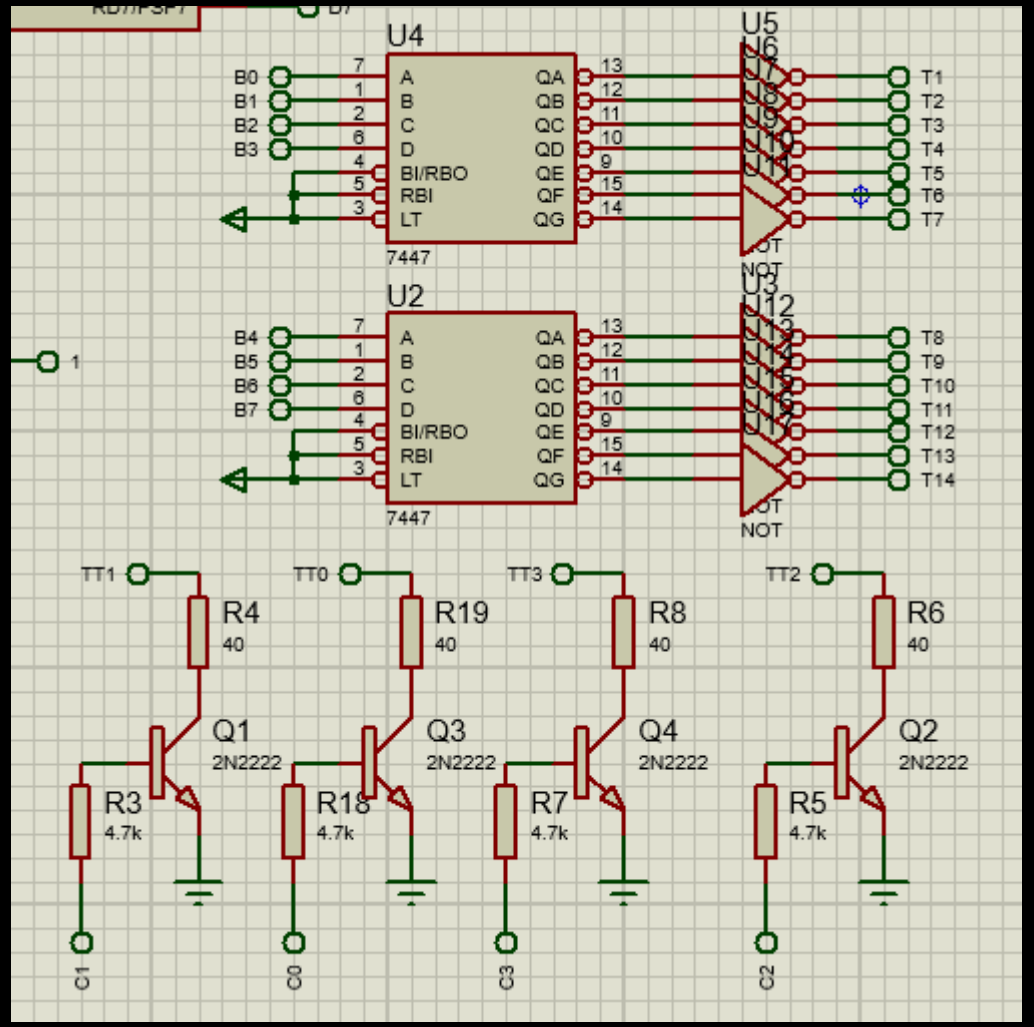


TRAFFIC LIGHT PROJECT

We will use common cathode so we need 7448 but in project description you prefer to use 7447 so we will just use 7447 and 7 NOT gates after it to display correct number in 7 segment That's it, quite simply

Instead of using 8 (330 ohm) resistors we will use a (40 ohm) resistor connected to 7 segment to reduce number of resistors

We use port B for the counter and pins c0:c3 as enables to the (NPN) transistors as it shown in project description



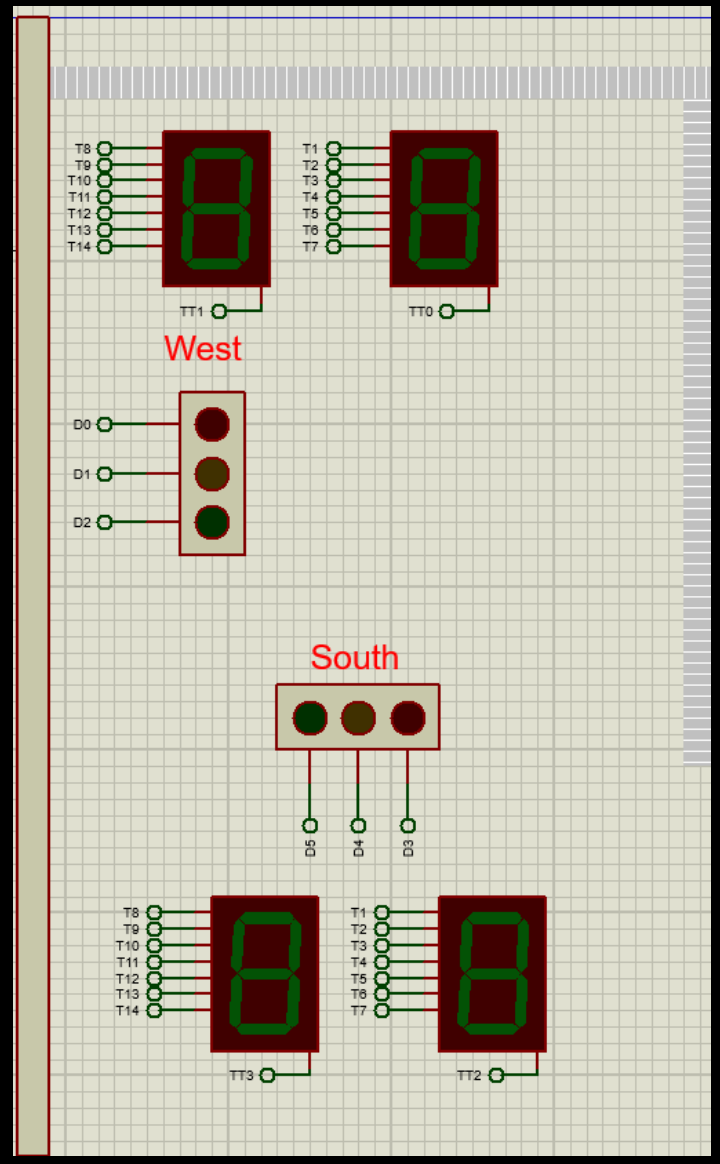
TRAFFIC LIGHT PROJECT

The final project will look like this

There is 7 tunnels for each (7-seg) and one enable tunnel

We will use “TRAFFIC LIGHTS” component and will take input from port D

Focus in tunnels names because we will use them in the code



TRAFFIC LIGHT PROJECT

We will use macros (define) to make the code simple so instead of writing porta.b0 every time we will write start and so on

We will declare global int counter and global char flags we will use them later

To BCD function display counter on port B , but how ?

If you have decimal number for ex 23 we will display 3 in the first four pins using (counter%10) and the 2 in the last 4 pins (counter/10) returns 2 but we want to shift it four bits so multiply it by (1<<4) which equals (10000) in binary

```
#define wait delay_ms(1000)
#define start porta.b0
#define convert porta.b1
#define manual porta.b2
#define red_west portd.b0
#define yellow_west portd.b1
#define green_west portd.b2
#define red_south portd.b3
#define yellow_south portd.b4
#define green_south portd.b5

int counter = 0;
char flags = 0;
void to_BCD(){
    // convert two digits of counter to BCD
    // Then assignment them in portb
    portb = (counter % 10) + (counter / 10) * (1<<4);
    // (1<<4) = 16
}
```

TRAFFIC LIGHT PROJECT

From it's name manual mode function works till you stop pressing convert button , If you wasn't pressing it the function will do nothing

Every time you press manual button one press (not long press) it will convert roads by reverse flags.B0

For three seconds the red traffic light will be the same and the green one will be yellow after the three seconds we convert roads

We know the roads order from flags.B0

```
void manual_mode(){
    while(!convert){
        portb = 0;
        if(!manual){
            flags.B0 = ! flags.B0;
            if(red_west){
              red_west = 1, green_west = 0, yellow_west = 0;
              red_south = 0, green_south = 0, yellow_south = 1;
            }else{
                red_west = 0, green_west = 0, yellow_west = 1;
                red_south = 1, green_south = 0, yellow_south = 0;
            }
            for(counter = 3 ; counter > 0 ; counter--){
                to_BCD();
                wait;
            }
        }
        if(flags.B0){
            red_west = 1, green_west = 0, yellow_west = 0;
            red_south = 0, green_south = 1, yellow_south = 0;
        }else{
            red_west = 0, green_west = 1, yellow_west = 0;
            red_south = 1, green_south = 0, yellow_south = 0;
        }
    }
}
```

TRAFFIC LIGHT PROJECT

In the function stop west the west traffic light will always be red

So as you can see all the leds are 0 instead red west led and green south red

Counter will be in the beginning 15 and decrease 1 every second

We check every time if you are pressing convert button we will go out of this function because it is an automatic mode function

To BCD display the counter in the 4 (7 seg)

When counter is 3 there is just 3 second last till green be red so we make it yellow

```
void stop_west(){
    red_west = 1, green_west = 0, yellow_west = 0;
    red_south = 0, green_south = 1, yellow_south = 0;
    for(counter = 15 ; counter > 0 ; counter--){
        if(!convert){
            return;
        }
        to_BCD();
        if(counter == 3){
            yellow_south = 1, green_south = 0;
        }
        wait;
    }
}
```

TRAFFIC LIGHT PROJECT

The (stop south) function is like (stop west) function they both in automatic mode every thing is the same but south will be the red this time and west will be green then yellow in the last 3 seconds

In main function we made adcon1=7 to convert port a to digital

Port A is the input (remember our three buttons)

Port B, C and D are the outputs but we made port C equals 255 because it is control our four (7 seg) and of course we want them to work

```
void stop_south(){
    red_west = 0, green_west = 1, yellow_west = 0;
    red_south = 1, green_south = 0, yellow_south = 0;
    for(counter = 23 ; counter > 0 ; counter--) {
        if(!convert){
            return;
        }
        to_BCD();
        if(counter == 3){
            green_west = 0, yellow_west = 1;
        }
        wait;
    }
}
void main() {
    adcon1 = 7;
    // Input
    trisa = 1;
    porta = 0b111111;
    ////////////////////
    // Outputs
    trisb = 0;
    trisc = 0;
    trisd = 0;
    portb = 0;
    portc = 0xff;
    portd = 0;
    ////////////////////
```

TRAFFIC LIGHT PROJECT

We made flags.B0 1 at first so we will begin with stop west function

While start is not pressed the program won't begin

We start with manual mode function and then flags.B0 will tell us if we will start with stop west or stop south function

You might ask why we enter manual mode function every time ?

Because in the function if you wasn't pressing convert button it will go out of the function

If you press convert button while you are in (stop south) or (stop west) it will go out of it and enter manual mode function

```
    portc = 0xff;
    portd = 0;
    ////////////////////
    flags.B0 = 1;
    while(start){;}
    while(1){
        manual_mode();
        if(flags.B0){
            stop_west();
            stop_south();
        }else{
            stop_south();
            stop_west();
        }
    }
}
```


Thank
You