CSE221
COURSE PROJECT
– Traffic Light –

Group 28:

Name	ID
Abdel-Rahman Ibrahim El Said Ahmed Megahed	18P7423
Eman Khaled Ahmed Ibrahuim	18P9713
Maryam Ahmed Galal Nouh	18P2824
Seif Ahmed ElSayed Elewa	18P5662
Youssef Mohamed Mostafa Mansi	18P5848

Code Link –

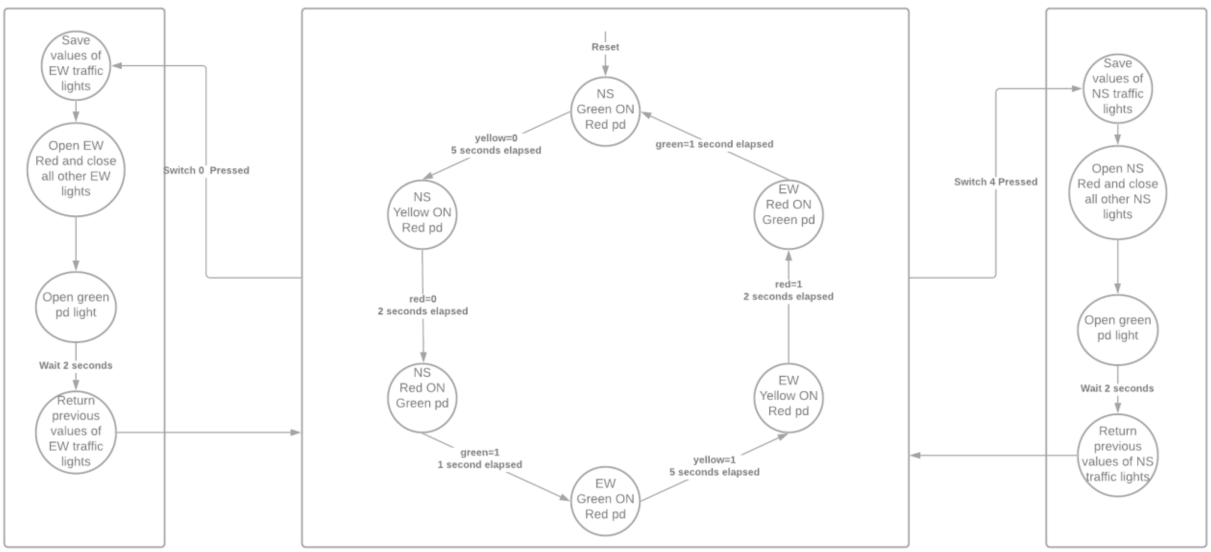
https://drive.google.com/drive/folders/1ZYpl6j4f3t4QrUmjtSF5L11G6PVdn8eN?usp=sharing

Video Link –

https://www.youtube.com/watch?v=PV--y3Qfuak

Project Idea –

Finite State Machine:



Assumptions:

- If the traffic light for cars is red on a street, then the pedestrian traffic light is green on that same street.
- If a pedestrian button is pressed it only affects its traffic light and pedestrian light. All other lights remain in their current states.
- If two pedestrians pushed the pedestrian button together, the pedestrian light that was already green is unaffected, while the other button fires an interrupt.
- If the same button was being pressed more than once in the same period for pedestrian crossing, the first time fires for an interrupt while all the other times are ignored.

Project Implementation –

Port Mapping:

Mapping	
Street 1 Red LED	Port B Pin 0
Street 1 Yellow LED	Port B Pin 1
Street 1 Green LED	Port B Pin 2
Street 1 Pd Red LED	Port B Pin 3
Street 1 Pd Green LED	Port B Pin 4
Street 2 Red LED	Port A Pin 2
Street 2 Yellow LED	Port A Pin 3
Street 2 Green LED	Port A Pin 4
Street 2 Pd Red LED	Port A Pin 5
Street 2 Pd Green LED	Port A Pin 6

Code:

```
1   #include "stdint.h"
2   #include "stdbool.h"
3   #include "driverlib/sysctl.h"
4   #include "driverlib/systick.h"
5   #include "driverlib/interrupt.h"
6   #include "inc/hw_memmap.h"
7   #include "inc/hw_gpio.h"
8   #include "inc/hw_ints.h"
9   #include "inc/hw_types.h"
10  #include "driverlib/gpio.h"
11  #include "stdio.h"
12  #include "stdlib.h"
13  #include "tm4c123gh6pm.h"
14  #include "driverlib/timer.h"
15  #include "string.h"
16  #include "time.h"
17  #include "driverlib/sysctl.h"
18  #include "inc/hw_memmap.h"
19  #include "driverlib/gpio.h"
20  #include "tm4c123gh6pm.h"
21
22  //Interrupt Handler Declarations
23  void GPIOFHandler(void); //Handler for Port F Interrupts
24  void Timer0AHandler(void); // Handler for Timer 0A Interrupts
25  void Timer1AHandler(void); // Handler for Timer 1A Interrupts
26  void Timer2AHandler(void); // Handler for Timer 2A Interrupts
27
28  //Global Variables for FSM
29  static int t=0; // Indicates which timer is counting
30  static int red = 0; // For State Traversal
31  static int green = 1; // For State Traversal
32  static int yellow = 0; // For State Traversal
33  static int paused = 0; // Indicates if a timer is counting or paused
34
```

We first import all the libraries used in our project (lines 1-20).

Then, we declare the functions that are defined in the end of the code. There are four functions, `GPIOHandler()` if the interrupt service routine (ISR) for the ports used. Meanwhile, `Timer0Ahandler()`, `Timer1Ahandler()`, and `Timer2Ahandler()` are three ISRs for three of the timers used in the project: timers 0,1 and 2.

We used five variables to navigate the finite state machine, `t` is a variable that can take one of three values {0,1,2}, representing the timer currently counting. As for `red`, `green` and `yellow`, as seen in the FSM diagram, they are the three flags that move us from one state to another. They take two values, the value 0 means the traffic light North-South (NS) and the value 1 means the traffic light East-West (EW), while `yellow` is the transition from state green=on to yellow=on, `red` is the transition from yellow=on to red=on and green is the transition from red=on to green=on. The last variable, `paused` is a flag to indicate whether or not a timer has been paused, value 0 means it is still counting, while 1 means It is paused.

```c
35 int main(){
36   // Port F Initialization
37   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); // To enable clock to Port F
38   while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOF));// Wait until clock is enabled for Port F
39   GPIO_PORTF_LOCK_R = 0x4C4F434B;// To unclock Port F
40   GPIO_PORTF_CR_R = 0x11;// To enable changes to pins 0 & 4
41   GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);// Set Pin 0 as input
42   GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);// Set Pin 4 as input
43   GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU); // Set Pin 0 to pull up
44   GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);// Set Pin 4 to pull up
45
46
47   // Port A Initialization
48   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);// To enable clock to Port A
49   while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOA));// Wait until clock is enabled for Port A
50   GPIO_PORTA_LOCK_R = 0x4C4F434B;// To unclock Port A
51   GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_2);// Set Pin 2 as output Red EW
52   GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3);// Set Pin 3 as output Yellow EW
53   GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_4);// Set Pin 4 as output Green EW
54   GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_5);// Set Pin 5 as output Red pd EW
55   GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6);// Set Pin 6 as output Green pd EW
56
57
58   //Port B Initialization
59   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);// To enable clock to Port B
60   while (!SysCtlPeripheralReady(SYSCTL_PERIPH_GPIOB));// Wait until clock is enabled for Port B
61   GPIO_PORTB_LOCK_R = 0x4C4F434B; // To unclock Port B
62   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_0);// Set Pin 0 as output Red NS
63   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1);// Set Pin 1 as output Yellow NS
64   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2);// Set Pin 2 as output Green NS
65   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_3);// Set Pin 3 as output Red pd NS
66   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_4);// Set Pin 4 as output Green pd NS
67
```

This is the beginning of the main.

Here we first initialize Port F, used for the switches, which are pin 0 and pin 4. Line 37 enables the clock to Port F, and then we enter a while loop in line 38 to check if the clock is enabled or not, if it is not enabled, busy-wait, otherwise, break out of the loop. Then, unlock Port F and enable changed to pins 0 and 1. Finally, we set pin 0 and pin 4 as inputs, read the switch when it is pressed, and set to pull up.

Then, initialize Port A, used for the east-west car and pedestrian traffic lights, via pins 2-6 (red, yellow and green for cars then red and green for pedestrians, respectively). We enable the clock to Port A, and then we enter a while loop to check if the clock is enabled or not, if it is not enabled, busy-wait, otherwise, break out of the loop. Then, unlock Port A. Lastly, we set the 5 pins as outputs.

Then, initialize Port B, used for the north-south car and pedestrian traffic lights, via pins 0-4 (red, yellow and green for cars then red and green for pedestrians, respectively). We enable the clock to Port B, and then we enter a while loop to check if the clock is enabled or not, if it is not enabled, busy-wait, otherwise, break out of the loop. Then, unlock Port B. Lastly, we set the 5 pins as outputs.

```
// Timer 0 A Initialization
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);// To enable clock to Timer 0
while (!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER0));// Wait until clock is enabled for Timer 0
TimerDisable(TIMER0_BASE,TIMER_A); // Disable Timer 0 A
TimerConfigure(TIMER0_BASE, TIMER_CFG_A_ONE_SHOT); // Make Timer 0 A oneshot
TimerClockSourceSet(TIMER0_BASE, TIMER_CLOCK_SYSTEM); // Make Timer 0 use the system clock to count
TimerLoadSet(TIMER0_BASE, TIMER_A,15999999); // Add a load value of 1 second to Timer 0 A
TIMER0_CTL_R |=0x2; // Make Timer 0 A Stall for debugging
TimerIntRegister(TIMER0_BASE, TIMER_A, Timer0AHandler); // Set the Interrupt function for Timer 0 A
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Enable the Interrupt on Timer 0 A Timeout

// Timer 1 A Initialization
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1); // To enable clock to Timer 1
while (!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER1));// Wait until clock is enabled for Timer 1
TimerDisable(TIMER1_BASE,TIMER_A); // Disable Timer 1 A
TimerConfigure(TIMER1_BASE, TIMER_CFG_A_ONE_SHOT ); // Make Timer 1 A oneshot
TimerClockSourceSet(TIMER1_BASE, TIMER_CLOCK_SYSTEM); // Make Timer 1 use the system clock to count
TimerLoadSet(TIMER1_BASE, TIMER_A,15999999); // Add a load value of 1 second to Timer 1 A
TIMER1_CTL_R |=0x2; // Make Timer 1 A Stall for debugging
TimerIntRegister(TIMER1_BASE, TIMER_A, Timer1AHandler); // Set the Interrupt function for Timer 1 A
TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);// Enable the Interrupt on Timer 1 A Timeout


// Timer 2 A Initialization
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);// To enable clock to Timer 2
while (!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER2));// Wait until clock is enabled for Timer 2
TimerDisable(TIMER2_BASE,TIMER_A);// Disable Timer 2 A
TimerConfigure(TIMER2_BASE, TIMER_CFG_A_ONE_SHOT );// Make Timer 2 A oneshot
TimerClockSourceSet(TIMER2_BASE, TIMER_CLOCK_SYSTEM);// Make Timer 2 use the system clock to count
TimerLoadSet(TIMER2_BASE, TIMER_A,15999999);// Add a load value of 1 second to Timer 2 A
TIMER2_CTL_R |=0x2;// Make Timer 2 A Stall for debugging
TimerIntRegister(TIMER2_BASE, TIMER_A, Timer2AHandler);// Set the Interrupt function for Timer 2 A
TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);// Enable the Interrupt on Timer 2 A Timeout
```

Following that, we initialize the timers.

First, we initialize timer 0 A to count 5 seconds for the green=on to yellow=on transition. To do so, we first need to enable its clock, and enter a while loop to check if the clock is enabled or not, if it is not enabled, busy-wait, otherwise, break out of the loop. We then disable the timer, just in case it was already counting. Set the timer's mode to be oneshot, using half the timer, "A", and to use the system clock. Then we add a load value of 15999999 equivalent to one second. Lastly, we give the timer its ISR, here, `Timer0Ahandler()` and enable interrupts at timeout.

Second, to initialize timer 1 A to count 2 seconds for the yellow=on to red=on transition, we follow the same steps for timer 0 A. The only difference is in the arguments passed to the functions and the ISR, which is `Timer1Ahandler()`.

Last, to initialize timer 2 A to count 1 second for the red=on to green=on transition, we follow the same steps for timer 0 A, modifying the arguments passed to the functions and the ISR, which is `Timer2Ahandler()`.

```
// Timer 3 A Initialization
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER3);// To enable clock to Timer 3
while (!SysCtlPeripheralReady(SYSCTL_PERIPH_TIMER3));// Wait until clock is enabled for Timer 3
TimerDisable(TIMER3_BASE,TIMER_A);// Disable Timer 3 A
TimerConfigure(TIMER3_BASE, TIMER_CFG_A_ONE_SHOT );// Make Timer 3 A oneshot
TimerClockSourceSet(TIMER3_BASE, TIMER_CLOCK_SYSTEM);// Make Timer 3 use the system clock to count
TimerLoadSet(TIMER3_BASE, TIMER_A,15999999);// Add a load value of 1 second to Timer 3 A
TIMER3_CTL_R |=0x2;// Make Timer 3 A Stall for debugging
```

Here, we initialize timer 3, which, unlike the other timers, is not for the traffic lights. We enable its clock, and enter a while loop to check if the clock is enabled or not, if it is not enabled, busy-wait, otherwise, break out of the loop. We then disable the timer, just in case it was already counting. Set the timer's mode to be oneshot, using half the timer, "A", and to use the system clock. Then we add a load value of 15999999 equivalent to one second.

```

//Set Interrupt Priority
IntPrioritySet(INT_GPIOF, 0x10); //Highest Priority
IntPrioritySet(INT_TIMER0A, 0x20);
IntPrioritySet(INT_TIMER1A, 0x20);
IntPrioritySet(INT_TIMER2A, 0x20);


// Set Initial State
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0,0); // Red NS OFF
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1,0); // Yellow NS OFF
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2,GPIO_PIN_2); // Green NS ON
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3,GPIO_PIN_3); // Red pd NS ON
GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4,0); // Green pd NS OFF

GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2,GPIO_PIN_2); // Red EW ON
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3,0); // Yellow EW OFF
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4,0); // Green EW OFF
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5,0); // Red EW pd OFF
GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6,GPIO_PIN_6); // Green EW pd ON


GPIOIntRegister(GPIO_PORTF_BASE, GPIOFHandler); // Set the interrupt handler for Port F
GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_4); // Enable Interrupt for Port F Pin 4
GPIOIntEnable(GPIO_PORTF_BASE, GPIO_INT_PIN_0);// Enable Interrupt for Port F Pin 0
TimerEnable(TIMER0_BASE, TIMER_A); // Start Timer 0

while(1){

  __asm("    wfi\n"); // Allow for the processor to sleep until interrupt
}

return 0;
}
```

We start off here by setting the priorities of the ports and the timers, giving the highest prioroty to GPIO interrupts, and the least priority to the timer interrupts. Timer 0 A, Timer 1 A, Timer 2 A are given the same priotity.

Then we inialzite the state of each LED:

- Cars North-South - initial state of Green is on, while Red and Yellow are off.
- Pedestrians North-South - Red is on and Green is off.
- Cars East-West - initial state of Red is on, while Green and Yellow are off.

- Pedestrians East-West - Red is off and Green is on

Then we set the interrupt handler (ISR) on Port F, which is `GPIOFHandler()`, enabling interrupts from pin 0 and pin 4. Lastly, we enable the first timer, timer 0 A, to start counting, starting from the reset state of the FSM.

The main function contains a while loop that is infinite, that repeatedly sleeps the interrupt until an interrupt is fired.

```c
148 //Timer 0A Handler
149 void Timer0AHandler(void){
150   paused=1; // Set paused to 1 indicating that no timer is counting
151   static int count=1; //counts the number of times the interrupt is called
152   if(count%5==0){ // Makes sure the function works once every 5 times it is called (5 seconds)
153     if (yellow==0){ //Checks state variable yellow
154 
155       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2,0); //Close Green NS
156       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, GPIO_PIN_1); // Open yellow NS
157       yellow=1; // Sets yellow to 1 so the next time the function is called it goes to the other street
158 
159     }
160     else{ //if yellow = 1
161 
162       GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4,0); //Close Green EW
163       GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, GPIO_PIN_3); // Open yellow EW
164       yellow=0; // Sets yellow to 0 so the next time the function is called it goes to the other street
165 
166     }
167 
168     count++; //Increments count
169     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //Clears interrupt for Timer 0A
170     t=1; //Indicates Timer 1A is counting
171     TimerEnable(TIMER1_BASE, TIMER_A); //Enables Timer 1A
172     paused=0; // Indicates that there is a timer counting
173 
174   }
175   else{ // if count is not divisible by 5
176 
177     count++; //increments count
178     TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); // Clears interrupt for Timer 0A
179     TimerEnable(TIMER0_BASE, TIMER_A); //Enables Timer 0A again
180     paused=0;// Indicates that there is a timer counting
181   }
182 
183 }
```

Above is the timer 0 A handler function, which is the function that handles the green LED that should be on for 5 seconds. First, once inside the function, we set `paused = 1`, since timer 0 A has finished counting and no other timer has been enabled, meaning that no timer is currently couting. In line 151, we are initializing a new static variable, `count`, to count the number of times this function is called and make sure that it doesn't fire an interrupt if it is not a multiple of 5, which represents 5 seconds.

If we have reached a multiple of 5, meaning that 5 seconds have passed, then we check for the state of `yellow`, if it is 0, then we close the green traffic light for cars on the north-south street and open yellow for that same street, finishing by setting `yellow` to 1, toggling its value to modify the other street next time. Otherwise, we do the same on the East-West street and set the `yellow` flag to 0. Aftwerwards, we change the value of t to 1 meaning a transition in the FSM to count 2 seconds via timer 1 A. Then we enable timer 1 A to start counting and evidently, set paused to 0.

If we have not reached a multiple of 5, we re-enable timer 0 A and also, set `paused` to 0.

At each function call, the count variable increments and clears the interrupt flag for the timer to exit the ISR.

```c
//Timer 1A Handler
void Timer1AHandler(void){
  paused=1;// Set paused to 1 indicating that no timer is counting
  static int count=1;//counts the number of times the interrupt is called
  if(count%2==0){// Makes sure the function works once every 2 times it is called (2 seconds)

    if(red==0){//Checks state variable red
      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1,0); //Close Yellow NS
      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0, GPIO_PIN_0); // Open Red NS
      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, 0); //Close Red Pd NS
      GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, GPIO_PIN_4); //Open Green Pd NS
      red=1;// Sets red to 1 so the next time the function is called it goes to the other street

    }
    else{
      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3,0); //Close Yellow EW
      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2, GPIO_PIN_2); // Open Red EW
      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5, 0); //Close Red Pd EW
      GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, GPIO_PIN_6); //Open Green Pd EW
      red=0; // Sets red to 0 so the next time the function is called it goes to the other street

    }
    count++; //Increments count
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); //Clear Interrupt for TImer 1A
    t=2; //Indicates Timer 2A is counting
    TimerEnable(TIMER2_BASE, TIMER_A); //Enables Timer 2A
    paused=0; // Indicates that there is a timer counting
  }
  else{
    count++; //Increments count
    TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); //Clear Interrupt for TImer 1A
    TimerEnable(TIMER1_BASE, TIMER_A); //Enables Timer 1A again
    paused=0; // Indicates that there is a timer counting
  }

}
```

Above is the timer 1 A handler function, which is the function that handles the yellow LED that should be on for 2 seconds. First, once inside the function, we set `paused = 1`, since timer 1 A has finished counting and no other timer has been enabled, meaning that no timer is currently couting. We then initialize a new static variable, `count`, to count the number of times this function is called and make sure that it doesn't fire an interrupt if it is not a multiple of 2, which represents 2 seconds.

If we have reached a multiple of 2, meaning that 2 seconds have passed, then we check for the state of `red`, if it is 0, then we close the yellow traffic light for cars on the north-south street and open red for that same street. We also close the pedestrian red light on that street and open the green one, finishing by setting `red` to 1, toggling its value to modify the other street next time. Otherwise, we do the same on the East-West street and set the `red` flag to 0. Aftwerwards, we change the value of t to 2 meaning a transition in the FSM to count 1 second via timer 2 A. Then we enable timer 2 A to start counting and evidently, set paused to 0.

If we have not reached a multiple of 2, we re-enable timer 1 A and also, set `paused` to 0.

At each function call, the count variable increments, and clears the interrupt flag for the timer to exit the ISR.

```
// Timer 2A Handler
void Timer2AHandler(void){
  paused=1; // Set paused to 1 indicating that no timer is counting
  if(green==0){ //Checks state variable green
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0,0); //Close Red NS
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2,GPIO_PIN_2);// Open Green NS
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4, 0); //Close Green Pd NS
    GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3, GPIO_PIN_3); //Open Red Pd NS

    green=1; // Sets green to 1 so the next time the function is called it goes to the other street
  }
  else{
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2, 0); //Close Red EW
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4, GPIO_PIN_4); // Open Green EW
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, 0); //Close Green Pd EW
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5, GPIO_PIN_5); //Open Red Pd EW
    green=0; // Sets green to 0 so the next time the function is called it goes to the other street
  }
  TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT); // Clears Timer 2A interrupt
  t=0; // Indicates Timer 0A is counting
  TimerEnable(TIMER0_BASE, TIMER_A); // Enables Timer 0A
  paused=0; // Indicates that there is a timer counting
}
```

Above is the Timer 2 A handler function. First we set `paused = 1` indicating no timer is running, then we check the value of the state variable `green`; 0 indicates we change the values of NS street at Port B and 1 indicates we change the values of EW street at Port A.

We check for the state of `green`, if it is 0, then we close the red traffic light for cars on the North-South street and open red for that same street. We also close the pedestrian green light on that street and open the red one, finishing by setting `green` to 1, toggling its value to modify the other street next time. Otherwise, we do the same on the east-west street and set the `green` flag to 0. Aftwerwards, we clear the interrupt flag for the timer to exit the ISR and change the value of t to 0 meaning a transition in the FSM to count 5 seconds via timer 0 A. Then we enable timer 0 A to start counting and evidently, set paused to 0.

```
// Port F Handler
void GPIOFHandler(void){
  if(!paused){ // Checks to see if there is a timer counting
    if(t==0){ //Identifies which timer is counting
      TimerDisable(TIMER0_BASE,TIMER_A); //Pause Timer 0A
    } else if (t==1){
      TimerDisable(TIMER1_BASE,TIMER_A);//Pause Timer 1A
    } else if (t==2){
      TimerDisable(TIMER2_BASE,TIMER_A);//Pause Timer 2A
    }
    paused=1; // Set paused to 1 indicating that no timer is counting
  }

  //Bonus 1
  if((GPIOIntStatus(GPIO_PORTF_BASE,true)&0x11)!=0){ // Interrupt flags for both switches
    int32_t rns=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_0); //Get the Red NS value
    int32_t rew=GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_2); // Get the Red EW value
    if(rew>rns){ //Check to see which one is On
      GPIOIntClear(GPIO_PORTF_BASE,GPIO_PIN_4); //Red EW is on so clear the interrupt for EW
    }
    else{
      GPIOIntClear(GPIO_PORTF_BASE,GPIO_PIN_0);//Red NS is on so clear the interrupt for NS
    }
  }
```

This is the function that is called by the ISR if any interrupt flag for Port F is present, which handles if any pedestrian button is pressed. The function starts by checking if there is a timer that is still counting. If there is, then it pauses that timer. The variable `t` is used to identify which timer was currently active.

Then, we implemented the bonus scenario; if two switches were pressed at the same time. We handled this scenario by seeing if both interrupt flags for each switch are active or not. If they both were found to be active, then the processor checks to see which traffic light is red. If the traffic light is red on the East-West street, this means that the pedestrian light for that street is already green, then the interrupt for that street is cleared (as the lights do not need to be changed). This allows us to treat this case as if a single button was pressed.

```
// Pin 0 (Right Switch) is the switch for NS
if((GPIOIntStatus(GPIO_PORTF_BASE,true)&0x01)!=0){
  // Save current State
  int32_t rns=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_0);  //Red NS
  int32_t yns=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_1); //Yellow NS
  int32_t gns=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_2); //Green NS
  int32_t rpns=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_3); //Red pd NS
  int32_t gpns=GPIOPinRead(GPIO_PORTB_BASE,GPIO_PIN_4); //Green pd NS

  // Turn pedestrian lights green and street lights red
  GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_1,0); //Yellow NS OFF
  GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_2,0); //Green NS OFF
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3,0); //Red pd NS OFF
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0,GPIO_PIN_0); //Red NS ON
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4,GPIO_PIN_4); //Green pd NS ON

  //Delay for 2 seconds
  TimerEnable(TIMER3_BASE,TIMER_A); //Enable Timer 3A
  while((TIMER3_RIS_R & TIMER_TIMA_TIMEOUT) == 0){}; //Wait until Timer 3A Timeout
  TIMER3_ICR_R=0x1; //Clear timeout
  TimerEnable(TIMER3_BASE,TIMER_A); //Enable Timer 3A again
  while((TIMER3_RIS_R & TIMER_TIMA_TIMEOUT) == 0){}; //Wait until Timer 3A Timeout
  TIMER3_ICR_R=0x1; //Clear timeout

  //Return values of lights to its previous state
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_0,rns); //Red NS
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1,yns); //Yellow NS
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2,gns); //Green NS
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_3,rpns); //Red pd NS
  GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_4,gpns); //Green pd NS


}
```

Here we check to see if there is an interrupt flag for Pin 0. If there is, that means that Pin 0 was pressed, so we save the values of the traffic lights (the current state) before changing any value to be able to return the lights to that state once finished. The variables `rns`, `yns`, `gns`, `rpns`, and `gpns` store the values of the red, yellow, green, red pedestrian, and green pedestrian lights for the North-South street respectively. Then, we turn off all the lights and only turn on the green pedestrian and red traffic lights for the North-South street. After that, we enable Timer 3A to start counting and wait for it to timeout then clear it twice. This gives us the effect of waiting for 2 seconds which enables the pedestrian lights to stay green for 2 seconds. After that, the previous values for the lights that were stored are written again for each pin; indicating that the period of pedestrian crossing is over for the North-South street.

```c
// Pin 4 is the switch for EW
else if((GPIOIntStatus(GPIO_PORTF_BASE,true)&0x10)!=0){
  // Save current State
  int32_t rew=GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_2); //Red EW
  int32_t yew=GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_3); //Yellow EW
  int32_t gew=GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_4); //Green EW
  int32_t rpew=GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_5); //Red pd EW
  int32_t gpew=GPIOPinRead(GPIO_PORTA_BASE,GPIO_PIN_6); //Green pd EW

  // Turn pedestrian lights green and street lights red
  GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_3,0); //Yellow EW OFF
  GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_4,0); //Green EW OFF
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5,0); //Red pd EW OFF
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2,GPIO_PIN_2); //Red EW ON
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6,GPIO_PIN_6); //Green pd EW ON

  //Delay for 2 seconds
  TimerEnable(TIMER3_BASE,TIMER_A); //Enable Timer 3A
  while((TIMER3_RIS_R & TIMER_TIMA_TIMEOUT) == 0){}; //Wait until Timer 3A Timeout
  TIMER3_ICR_R=0x1; //Clear timeout
  TimerEnable(TIMER3_BASE,TIMER_A); //Enable Timer 3A again
  while((TIMER3_RIS_R & TIMER_TIMA_TIMEOUT) == 0){}; //Wait until Timer 3A Timeout
  TIMER3_ICR_R=0x1; //Clear timeout

  //Return values of lights to its previous state
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_2,rew); //Red EW
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3,yew); //Yellow EW
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_4,gew); //Green EW
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_5,rpew); //Red pd EW
  GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6,gpew); //Green pd EW

}
```

Here we check to see if there is an interrupt flag for Pin 4. If there is, that means that Pin 4 was pressed, so we save the values of the traffic lights (the current state) before changing any value to be able to return the lights to that state once finished. The variables `rew`, `yew`, `gew`, `rpew`, and `gew` store the values of the red, yellow, green, red pedestrian, and green pedestrian lights for the East-West street respectively. Then, we turn off all the lights and only turn on the green pedestrian and red traffic lights for the East-West street. After that, we enable Timer 3A to start counting and wait for it to timeout then clear it twice. This gives us the effect of waiting for 2 seconds which enables the pedestrian lights to stay green for 2 seconds. After that, the previous values for the lights that were stored are written again for each pin; indicating that the period of pedestrian crossing is over for the East-West street.

```
//Bonus 2
GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_0); //Clear Interrupt for Pin 0
GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4); // Clear Interrupt for Pin 4



if (paused==1){ // Checks to see if a timer paused
  if(t==0){ //Identifies which timer is paused
    TimerEnable(TIMER0_BASE,TIMER_A); //Enables Timer 0A
  }
  else if (t==1){
    TimerEnable(TIMER1_BASE,TIMER_A); //Enables Timer 1A
  }
  else if (t==2){
    TimerEnable(TIMER2_BASE,TIMER_A); //Enables Timer 2A
  }
}
}
```

After returning the traffic lights to the previous state, we need to clear the interrupt flags for both pins for the processor to be able to continue the task that it was doing before the interrupt. Clearing both pins here makes us handle the situation where there was a button pressed more than one time during the pedestrian corssing period. This allows us to neglect the button being pressed again during the pedestrian crossing period. Then, we make sure that `paused==1`; meaning that no timer is active, and by using the variable `t`, we find the timer that should be enabled and enable it.