Bert Zhao
EE 474

Abstract: This lab report provides an overview of lab 2 of EE/CSE 474. There is an introduction to the two primary objectives of this lab. Procedure specifics are included along with the relevant results and diagrams. The code for the lab can be found in the appendix.

Introduction: The two new concepts introduced in this lab are timers and interrupts. In section A of this lab, we learn to implement the blinking LED from lab 1 using the on-board timer rather than a busy loop. We also apply these timers the the traffic light controller that we created in lab 1. Section B of this lab introduces interrupts. We work with the startup file to implement our interrupt functions. We once again make updates to the traffic light controller, this time implementing our interrupts.

Procedure: In section A, we implement timers in the blinking light problem. Much like in lab 1, we use GPIO port F for the LEDs. There are eight different combinations of lights that can be made from the red, green, and blue. For the timer, we want it to control the LEDs to blink at a one second rate. Since the timer counts in units per clock cycle, and since the TIVA has a clock frequency of 16MHz, we need to set the timer to count down from 16,000,000 to trigger a time out at exactly once per second. We implement the timers to the traffic light controller in the second part of this section. We want the buttons of the traffic light controller to become active only after holding them down for two seconds, and we want the lights to stay on for five seconds before moving on to the next light in the sequence. To do this, two timers are used, one for the buttons and one for the light. The light timer counts down from 80,000,000 and the button timer counts down from 32,000,000 to get their respective five and two second delays.

In section B, we learn how implement interrupts for both our timers and GPIO ports. This also involves modifying the cstartup_M.c file for the implementation of our own handler functions. The first task is to use the on-board buttons to control a timer. Switch 1 is used to turn off the timer and leave on a solid red LED. Switch 2 is used to turn on the timer and have a blue LED blink at a one second rate with the timer. The second part of this lab involves implementing interrupts to the traffic light controller. The button timers and light timers use interrupts while the actual button detection is done through polling. The traffic light controller is functionally identical to section 1.

Results: The results of this lab were very similar to lab 1. The main focus of this lab was to achieve the same result with a more efficient implementation. More specifically, usings timers and interrupts. A description for the traffic light controller and important diagrams from the lab 1 report are included below.

For the traffic controller, we created the circuit of two switches and three LEDs on a breadboard. The circuit diagram and picture of the complete circuit are shown below. I used GPIO port A and pins PA2 and PA3 were used for the start/stop and pedestrian crossing buttons respectively. PA5, PA6, and PA7 were used for the green, yellow, and red LEDs. To solve potential debouncing issues when using the push buttons, a delay was used following the detection of each potential button press.

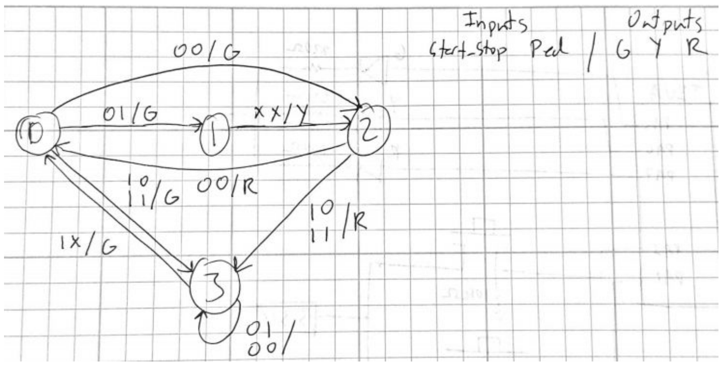


Figure 1: Traffic light state diagram

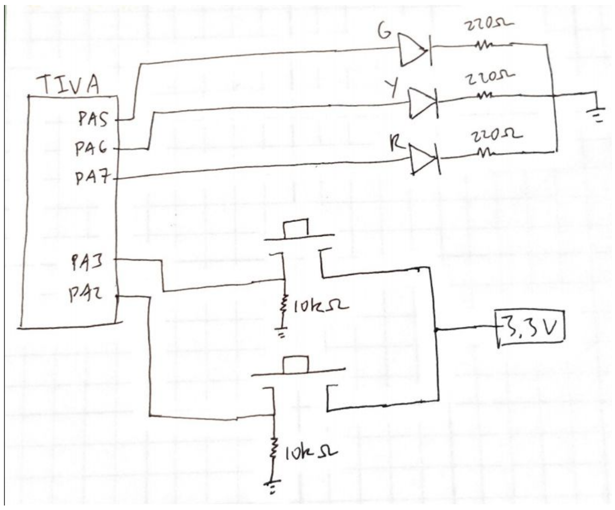


Figure 2: Traffic light circuit diagram

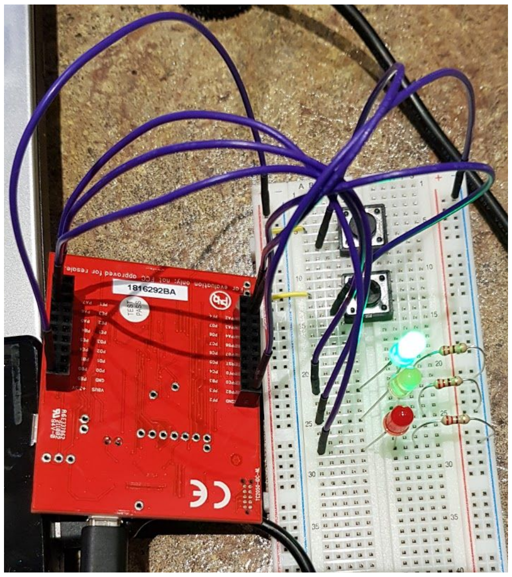
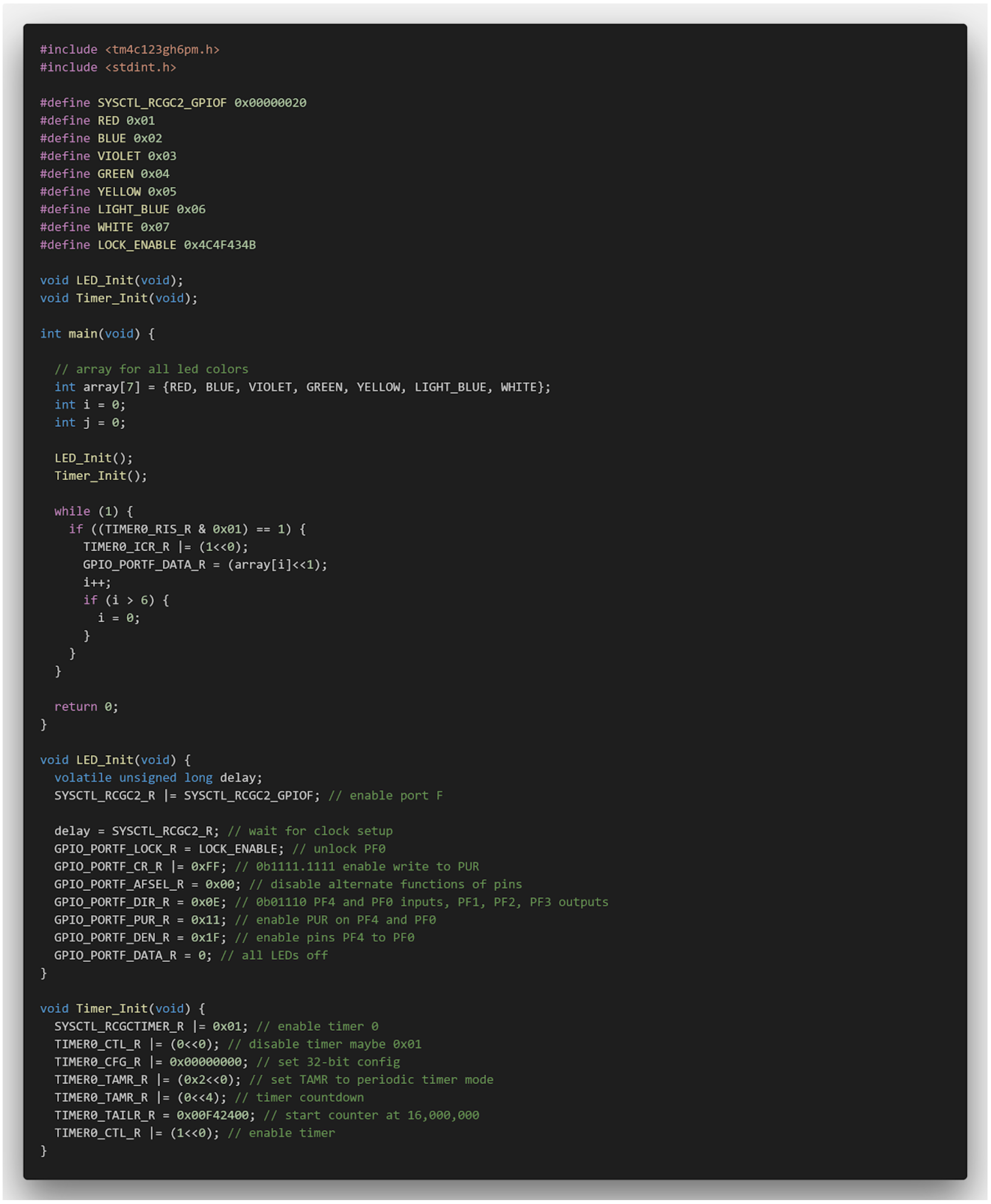


Figure 3: Traffic light circuit picture

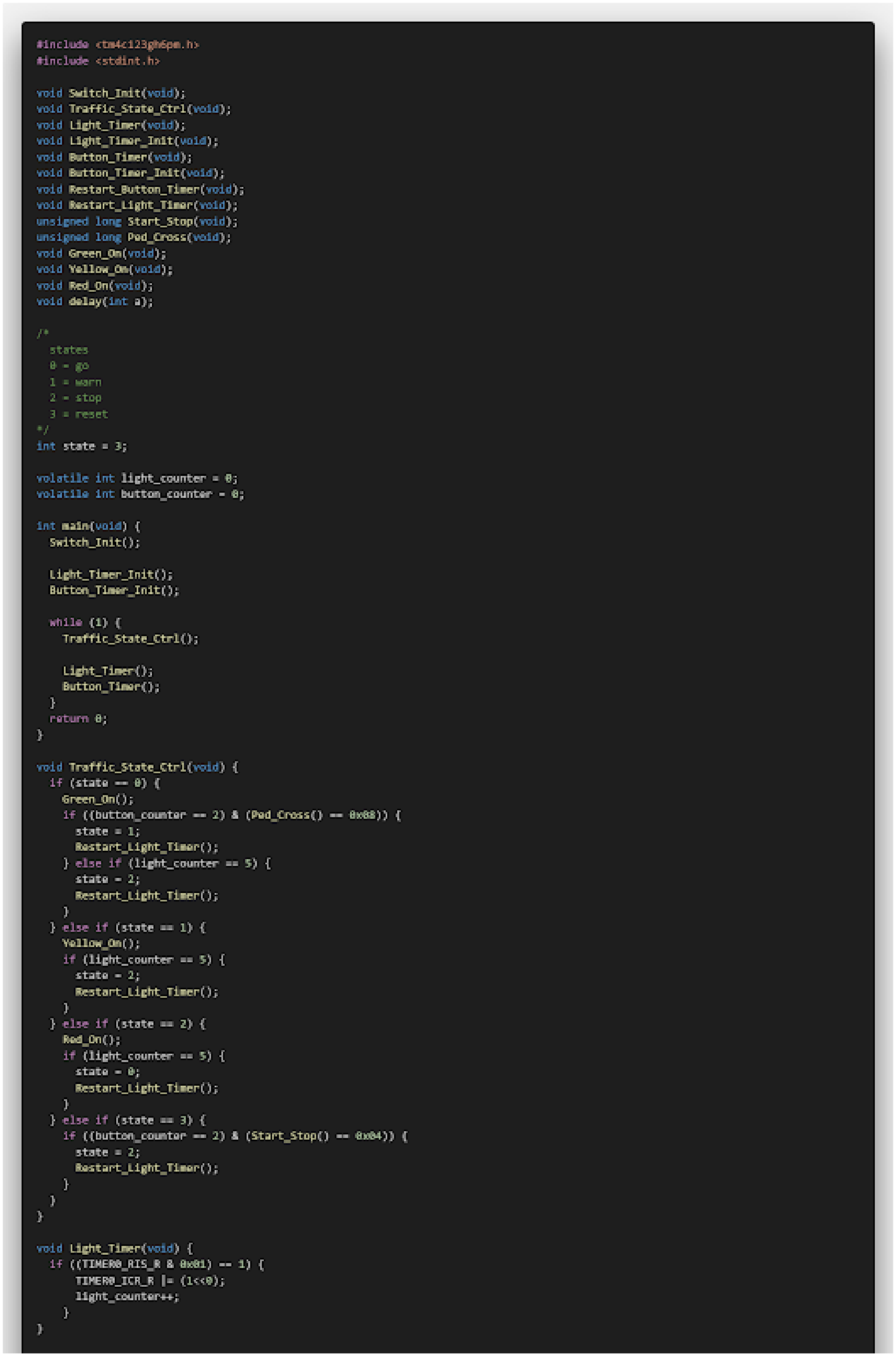
Conclusion: This lab introduced timers and interrupts. Interrupts are an important fundamental concept for embedded programming and it will be important to be familiar with interrupts moving forward into future labs and projects. Another important component to this lab is learning how to successfully manipulate the cstartup_M.c file to add our own interrupt handler functions. It is important to be very careful and to double check your work during this process.

Appendix:

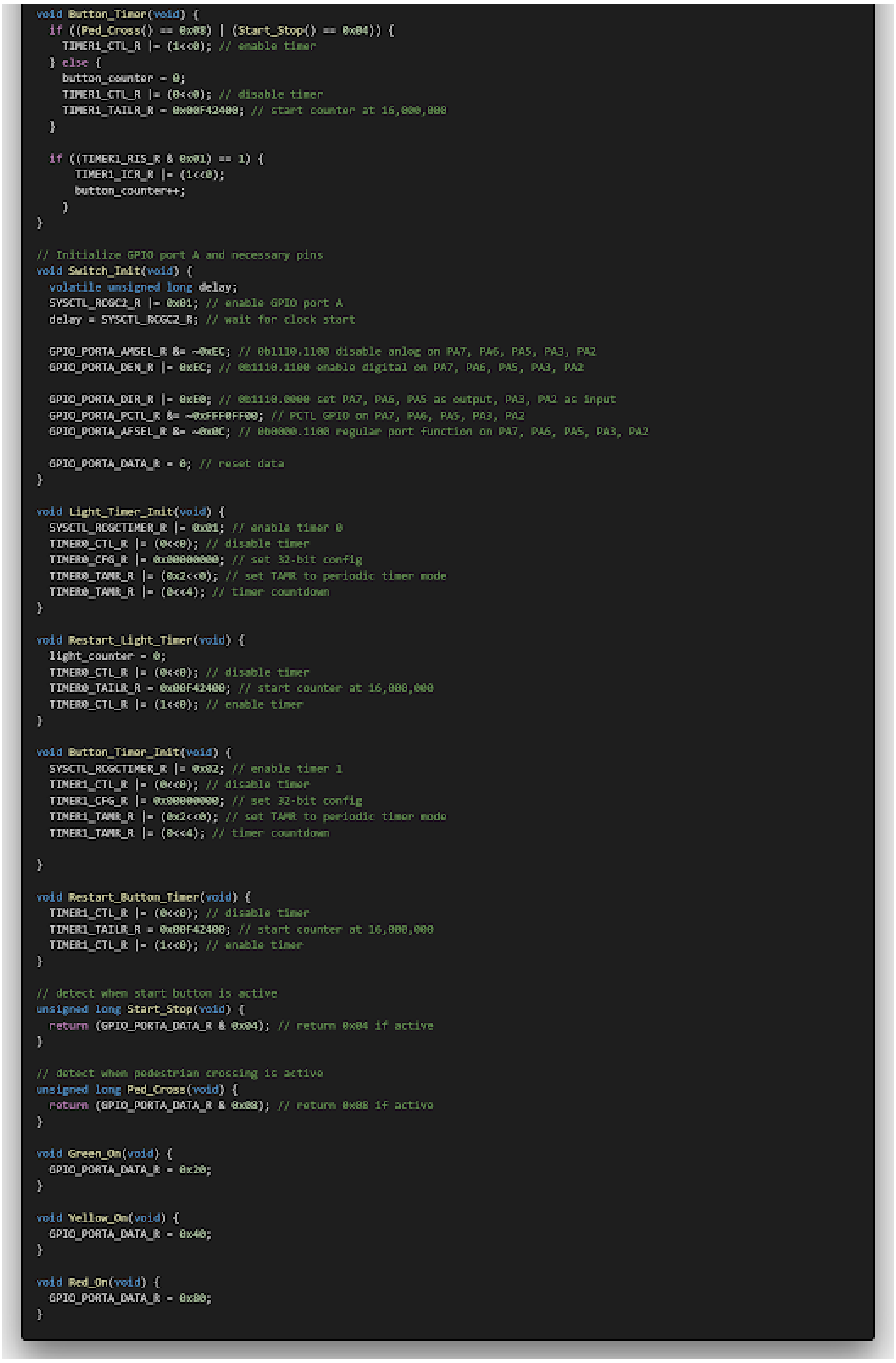


```
#include <tm4c123gh6pm.h>
#include <stdint.h>

#define SYSCTL_RCGC2_GPIOF 0x00000020
#define RED 0x01
#define BLUE 0x02
#define VIOLET 0x03
#define GREEN 0x04
#define YELLOW 0x05
#define LIGHT_BLUE 0x06
#define WHITE 0x07
#define LOCK_ENABLE 0x4C4F434B

void LED_Init(void);
void Timer_Init(void);

int main(void) {

  // array for all led colors
  int array[7] = {RED, BLUE, VIOLET, GREEN, YELLOW, LIGHT_BLUE, WHITE};
  int i = 0;
  int j = 0;

  LED_Init();
  Timer_Init();

  while (1) {
    if ((TIMER0_RIS_R & 0x01) == 1) {
      TIMER0_ICR_R |= (1<<0);
      GPIO_PORTF_DATA_R = (array[i]<<1);
      i++;
      if (i > 6) {
        i = 0;
      }
    }
  }

  return 0;
}

void LED_Init(void) {
  volatile unsigned long delay;
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF; // enable port F

  delay = SYSCTL_RCGC2_R; // wait for clock setup
  GPIO_PORTF_LOCK_R = LOCK_ENABLE; // unlock PF0
  GPIO_PORTF_CR_R |= 0xFF; // 0b1111.1111 enable write to PUR
  GPIO_PORTF_AFSEL_R = 0x00; // disable alternate functions of pins
  GPIO_PORTF_DIR_R = 0x0E; // 0b01110 PF4 and PF0 inputs, PF1, PF2, PF3 outputs
  GPIO_PORTF_PUR_R = 0x11; // enable PUR on PF4 and PF0
  GPIO_PORTF_DEN_R = 0x1F; // enable pins PF4 to PF0
  GPIO_PORTF_DATA_R = 0; // all LEDs off
}

void Timer_Init(void) {
  SYSCTL_RCGCTIMER_R |= 0x01; // enable timer 0
  TIMER0_CTL_R |= (0<<0); // disable timer maybe 0x01
  TIMER0_CFG_R |= 0x00000000; // set 32-bit config
  TIMER0_TAMR_R |= (0x2<<0); // set TAMR to periodic timer mode
  TIMER0_TAMR_R |= (0<<4); // timer countdown
  TIMER0_TAILR_R = 0x00F42400; // start counter at 16,000,000
  TIMER0_CTL_R |= (1<<0); // enable timer
}
```

Code 1: Section A Blinking Light

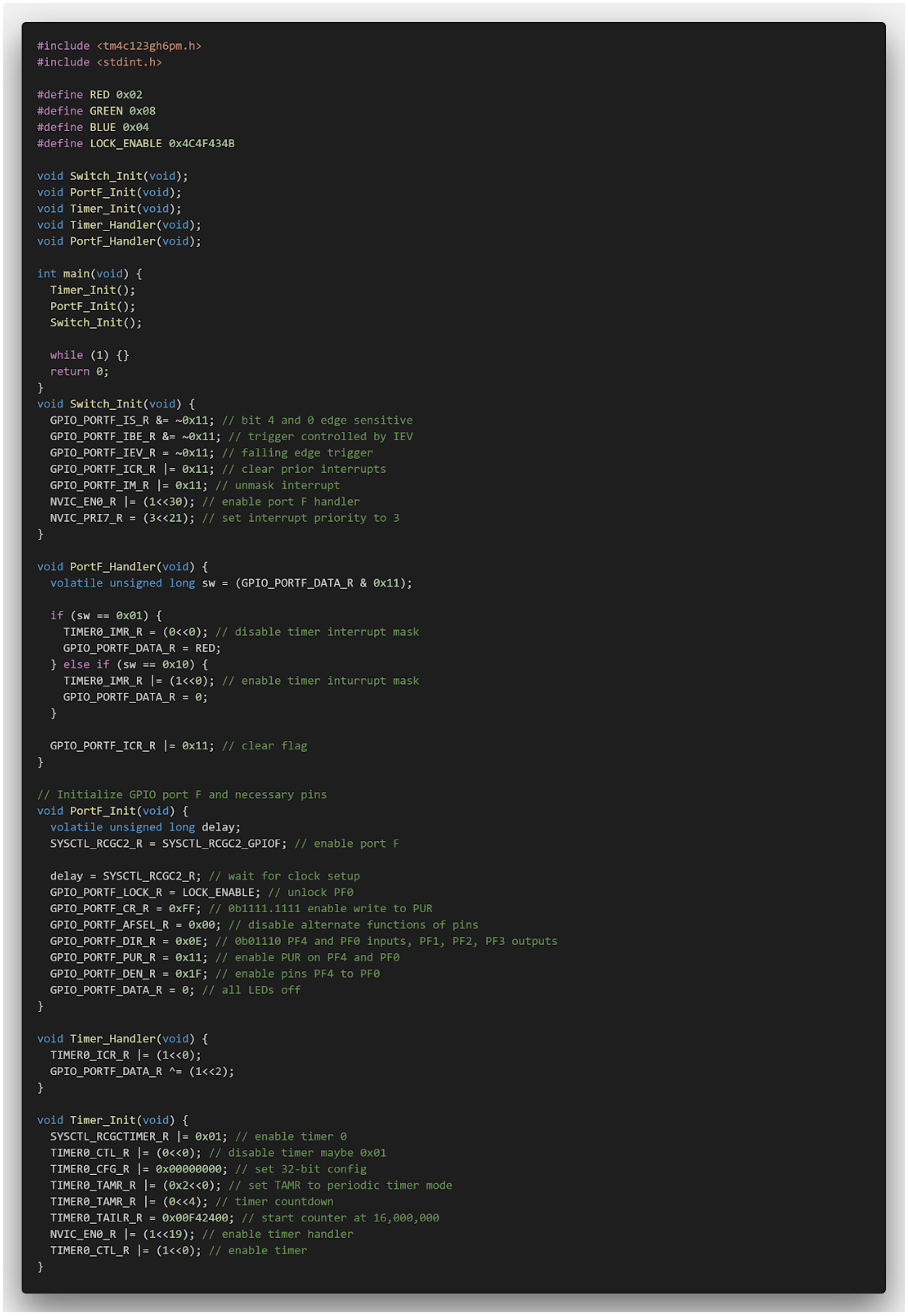


```c
#include <tm4c123gh6pm.h>
#include <stdint.h>

void Switch_Init(void);
void Traffic_State_Ctrl(void);
void Light_Timer(void);
void Light_Timer_Init(void);
void Button_Timer(void);
void Button_Timer_Init(void);
void Restart_Button_Timer(void);
void Restart_Light_Timer(void);
unsigned long Start_Stop(void);
unsigned long Ped_Cross(void);
void Green_On(void);
void Yellow_On(void);
void Red_On(void);
void delay(int a);

/*
  states
  0 = go
  1 = warn
  2 = stop
  3 = reset
*/
int state = 3;

volatile int light_counter = 0;
volatile int button_counter = 0;

int main(void) {
  Switch_Init();

  Light_Timer_Init();
  Button_Timer_Init();

  while (1) {
    Traffic_State_Ctrl();

    Light_Timer();
    Button_Timer();
  }
  return 0;
}

void Traffic_State_Ctrl(void) {
  if (state == 0) {
    Green_On();
    if ((button_counter == 2) & (Ped_Cross() == 0x08)) {
      state = 1;
      Restart_Light_Timer();
    } else if (light_counter == 5) {
      state = 2;
      Restart_Light_Timer();
    }
  } else if (state == 1) {
    Yellow_On();
    if (light_counter == 5) {
      state = 2;
      Restart_Light_Timer();
    }
  } else if (state == 2) {
    Red_On();
    if (light_counter == 5) {
      state = 0;
      Restart_Light_Timer();
    }
  } else if (state == 3) {
    if ((button_counter == 2) & (Start_Stop() == 0x04)) {
      state = 2;
      Restart_Light_Timer();
    }
  }
}

void Light_Timer(void) {
  if ((TIMER0_RIS_R & 0x01) == 1) {
      TIMER0_ICR_R |= (1<<0);
      light_counter++;
    }
}
```



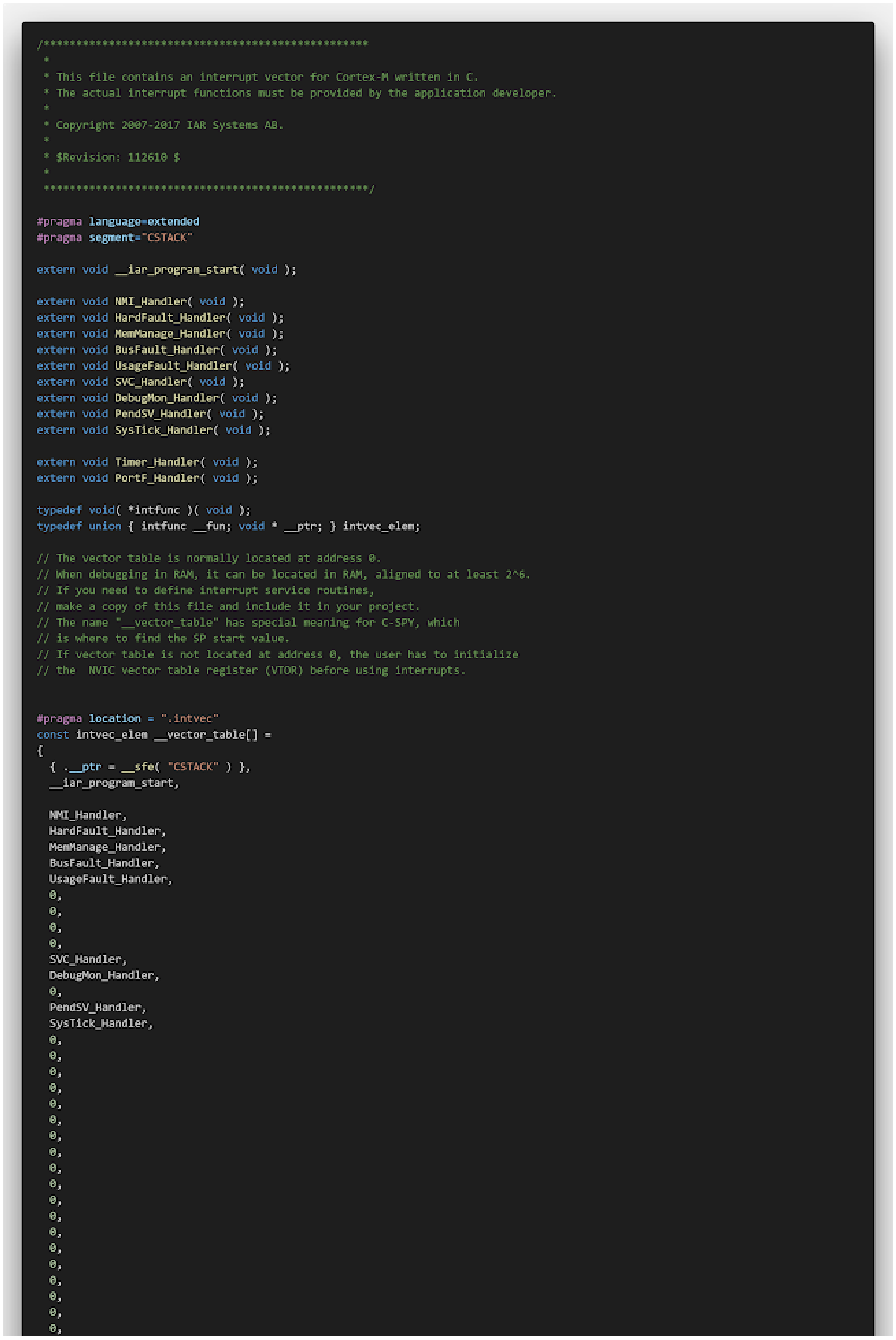
```c
void Button_Timer(void) {
  if ((Ped_Cross() == 0x08) | (Start_Stop() == 0x04)) {
    TIMER1_CTL_R |= (1<<0); // enable timer
  } else {
    button_counter = 0;
    TIMER1_CTL_R |= (0<<0); // disable timer
    TIMER1_TAILR_R = 0x00F42400; // start counter at 16,000,000
  }

  if ((TIMER1_RIS_R & 0x01) == 1) {
      TIMER1_ICR_R |= (1<<0);
      button_counter++;
    }
}

// Initialize GPIO port A and necessary pins
void Switch_Init(void) {
  volatile unsigned long delay;
  SYSCTL_RCGC2_R |= 0x01; // enable GPIO port A
  delay = SYSCTL_RCGC2_R; // wait for clock start

  GPIO_PORTA_AMSEL_R &= ~0xEC; // 0b1110.1100 disable anlog on PA7, PA6, PA5, PA3, PA2
  GPIO_PORTA_DEN_R |= 0xEC; // 0b1110.1100 enable digital on PA7, PA6, PA5, PA3, PA2

  GPIO_PORTA_DIR_R |= 0xE0; // 0b1110.0000 set PA7, PA6, PA5 as output, PA3, PA2 as input
  GPIO_PORTA_PCTL_R &= ~0xFFF0FF00; // PCTL GPIO on PA7, PA6, PA5, PA3, PA2
  GPIO_PORTA_AFSEL_R &= ~0x0C; // 0b0000.1100 regular port function on PA7, PA6, PA5, PA3, PA2

  GPIO_PORTA_DATA_R = 0; // reset data
}

void Light_Timer_Init(void) {
  SYSCTL_RCGCTIMER_R |= 0x01; // enable timer 0
  TIMER0_CTL_R |= (0<<0); // disable timer
  TIMER0_CFG_R |= 0x00000000; // set 32-bit config
  TIMER0_TAMR_R |= (0x2<<0); // set TAMR to periodic timer mode
  TIMER0_TAMR_R |= (0<<4); // timer countdown
}

void Restart_Light_Timer(void) {
  light_counter = 0;
  TIMER0_CTL_R |= (0<<0); // disable timer
  TIMER0_TAILR_R = 0x00F42400; // start counter at 16,000,000
  TIMER0_CTL_R |= (1<<0); // enable timer
}

void Button_Timer_Init(void) {
  SYSCTL_RCGCTIMER_R |= 0x02; // enable timer 1
  TIMER1_CTL_R |= (0<<0); // disable timer
  TIMER1_CFG_R |= 0x00000000; // set 32-bit config
  TIMER1_TAMR_R |= (0x2<<0); // set TAMR to periodic timer mode
  TIMER1_TAMR_R |= (0<<4); // timer countdown

}

void Restart_Button_Timer(void) {
  TIMER1_CTL_R |= (0<<0); // disable timer
  TIMER1_TAILR_R = 0x00F42400; // start counter at 16,000,000
  TIMER1_CTL_R |= (1<<0); // enable timer
}

// detect when start button is active
unsigned long Start_Stop(void) {
  return (GPIO_PORTA_DATA_R & 0x04); // return 0x04 if active
}

// detect when pedestrian crossing is active
unsigned long Ped_Cross(void) {
  return (GPIO_PORTA_DATA_R & 0x08); // return 0x08 if active
}

void Green_On(void) {
  GPIO_PORTA_DATA_R = 0x20;
}

void Yellow_On(void) {
  GPIO_PORTA_DATA_R = 0x40;
}

void Red_On(void) {
  GPIO_PORTA_DATA_R = 0x80;
}
```

Code 2: Section A Traffic Light w/Timers

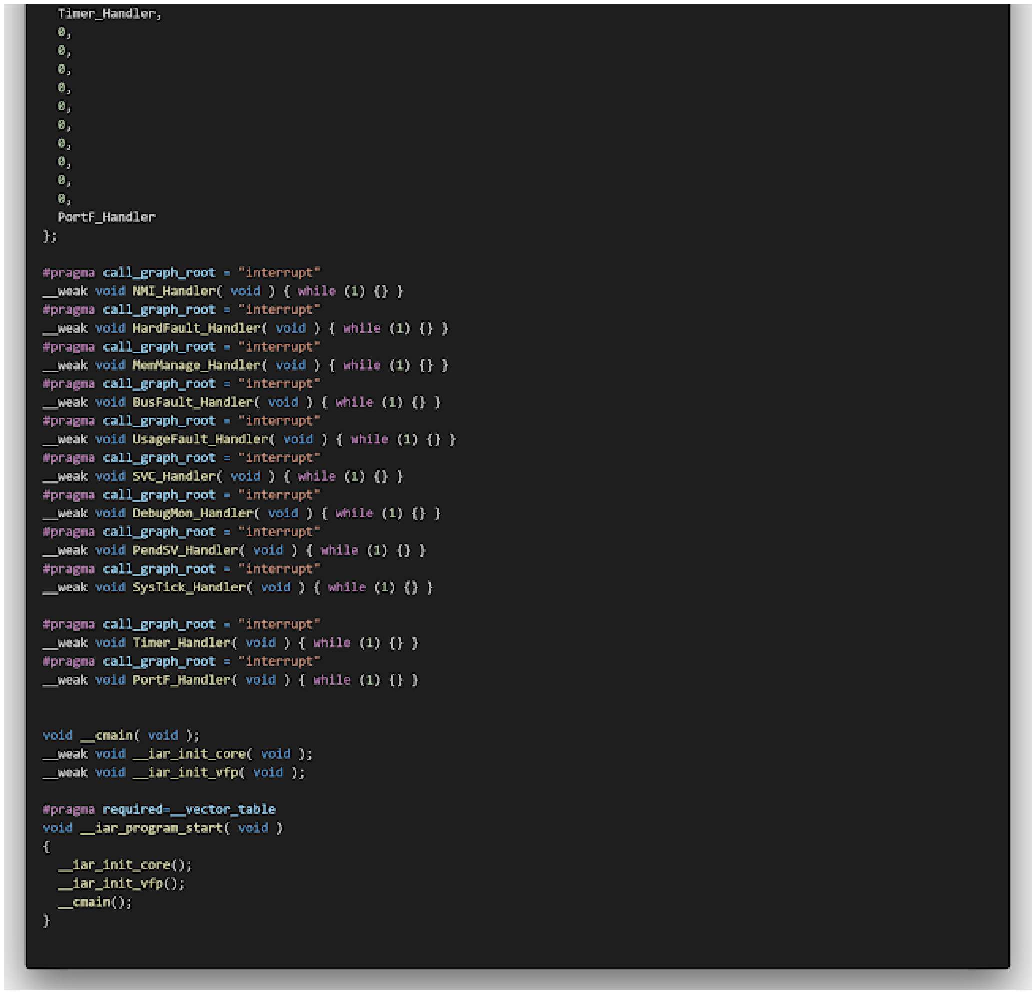


```c
#include <tm4c123gh6pm.h>
#include <stdint.h>

#define RED 0x02
#define GREEN 0x08
#define BLUE 0x04
#define LOCK_ENABLE 0x4C4F434B

void Switch_Init(void);
void PortF_Init(void);
void Timer_Init(void);
void Timer_Handler(void);
void PortF_Handler(void);

int main(void) {
  Timer_Init();
  PortF_Init();
  Switch_Init();

  while (1) {}
  return 0;
}
void Switch_Init(void) {
  GPIO_PORTF_IS_R &= ~0x11; // bit 4 and 0 edge sensitive
  GPIO_PORTF_IBE_R &= ~0x11; // trigger controlled by IEV
  GPIO_PORTF_IEV_R = ~0x11; // falling edge trigger
  GPIO_PORTF_ICR_R |= 0x11; // clear prior interrupts
  GPIO_PORTF_IM_R |= 0x11; // unmask interrupt
  NVIC_EN0_R |= (1<<30); // enable port F handler
  NVIC_PRI7_R = (3<<21); // set interrupt priority to 3
}

void PortF_Handler(void) {
  volatile unsigned long sw = (GPIO_PORTF_DATA_R & 0x11);

  if (sw == 0x01) {
    TIMER0_IMR_R = (0<<0); // disable timer interrupt mask
    GPIO_PORTF_DATA_R = RED;
  } else if (sw == 0x10) {
    TIMER0_IMR_R |= (1<<0); // enable timer inturrupt mask
    GPIO_PORTF_DATA_R = 0;
  }

  GPIO_PORTF_ICR_R |= 0x11; // clear flag
}

// Initialize GPIO port F and necessary pins
void PortF_Init(void) {
  volatile unsigned long delay;
  SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF; // enable port F

  delay = SYSCTL_RCGC2_R; // wait for clock setup
  GPIO_PORTF_LOCK_R = LOCK_ENABLE; // unlock PF0
  GPIO_PORTF_CR_R = 0xFF; // 0b1111.1111 enable write to PUR
  GPIO_PORTF_AFSEL_R = 0x00; // disable alternate functions of pins
  GPIO_PORTF_DIR_R = 0x0E; // 0b01110 PF4 and PF0 inputs, PF1, PF2, PF3 outputs
  GPIO_PORTF_PUR_R = 0x11; // enable PUR on PF4 and PF0
  GPIO_PORTF_DEN_R = 0x1F; // enable pins PF4 to PF0
  GPIO_PORTF_DATA_R = 0; // all LEDs off
}

void Timer_Handler(void) {
  TIMER0_ICR_R |= (1<<0);
  GPIO_PORTF_DATA_R ^= (1<<2);
}

void Timer_Init(void) {
  SYSCTL_RCGCTIMER_R |= 0x01; // enable timer 0
  TIMER0_CTL_R |= (0<<0); // disable timer maybe 0x01
  TIMER0_CFG_R |= 0x00000000; // set 32-bit config
  TIMER0_TAMR_R |= (0x2<<0); // set TAMR to periodic timer mode
  TIMER0_TAMR_R |= (0<<4); // timer countdown
  TIMER0_TAILR_R = 0x00F42400; // start counter at 16,000,000
  NVIC_EN0_R |= (1<<19); // enable timer handler
  TIMER0_CTL_R |= (1<<0); // enable timer
}
```

Code 3: Section B Interrupts



```c
/**************************************************
 *
 * This file contains an interrupt vector for Cortex-M written in C.
 * The actual interrupt functions must be provided by the application developer.
 *
 * Copyright 2007-2017 IAR Systems AB.
 *
 * $Revision: 112610 $
 *
 **************************************************/

#pragma language=extended
#pragma segment="CSTACK"

extern void __iar_program_start( void );

extern void NMI_Handler( void );
extern void HardFault_Handler( void );
extern void MemManage_Handler( void );
extern void BusFault_Handler( void );
extern void UsageFault_Handler( void );
extern void SVC_Handler( void );
extern void DebugMon_Handler( void );
extern void PendSV_Handler( void );
extern void SysTick_Handler( void );

extern void Timer_Handler( void );
extern void PortF_Handler( void );

typedef void( *intfunc )( void );
typedef union { intfunc __fun; void * __ptr; } intvec_elem;

// The vector table is normally located at address 0.
// When debugging in RAM, it can be located in RAM, aligned to at least 2^6.
// If you need to define interrupt service routines,
// make a copy of this file and include it in your project.
// The name "__vector_table" has special meaning for C-SPY, which
// is where to find the SP start value.
// If vector table is not located at address 0, the user has to initialize
// the  NVIC vector table register (VTOR) before using interrupts.


#pragma location = ".intvec"
const intvec_elem __vector_table[] =
{
  { .__ptr = __sfe( "CSTACK" ) },
  __iar_program_start,

  NMI_Handler,
  HardFault_Handler,
  MemManage_Handler,
  BusFault_Handler,
  UsageFault_Handler,
  0,
  0,
  0,
  0,
  SVC_Handler,
  DebugMon_Handler,
  0,
  PendSV_Handler,
  SysTick_Handler,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
```



```
  Timer_Handler,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  PortF_Handler
};

#pragma call_graph_root = "interrupt"
__weak void NMI_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void HardFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void MemManage_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void BusFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void UsageFault_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SVC_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void DebugMon_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PendSV_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void SysTick_Handler( void ) { while (1) {} }

#pragma call_graph_root = "interrupt"
__weak void Timer_Handler( void ) { while (1) {} }
#pragma call_graph_root = "interrupt"
__weak void PortF_Handler( void ) { while (1) {} }


void __cmain( void );
__weak void __iar_init_core( void );
__weak void __iar_init_vfp( void );

#pragma required=__vector_table
void __iar_program_start( void )
{
  __iar_init_core();
  __iar_init_vfp();
  __cmain();
}
```

Code 4: cstartup_M for Task 3