AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

INTRODUCTION TO EMBEDDED SYSTEMS

CSE_211

TEAM_14

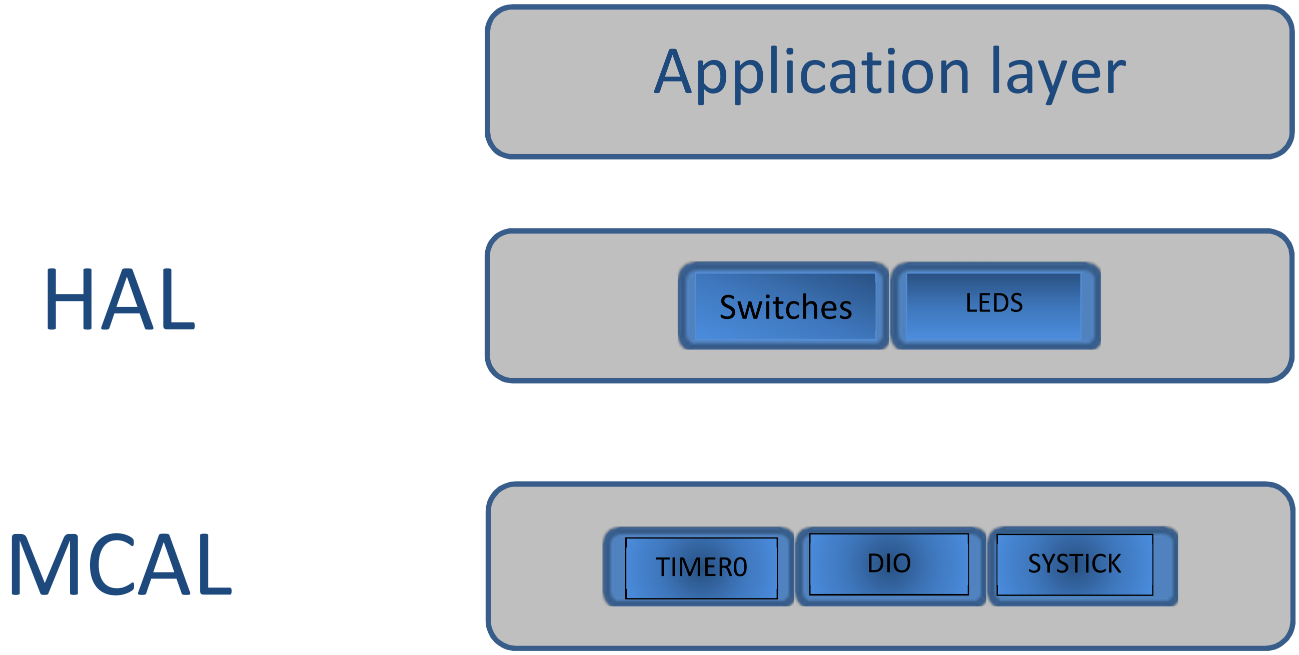
NAME	ID	Sec
Abdelrahman Wael Abdelrahman	1804781	1
Elsayed Ayman Elsayed Ali Habib	1804765	1
Abdelrahman Mahmoud Ehab leithy	1809161	1
Mahmoud Ayman Abdelaziz El Said	1802936	1
Anas Ahmed Talaat khaled	1806766	1

Contents

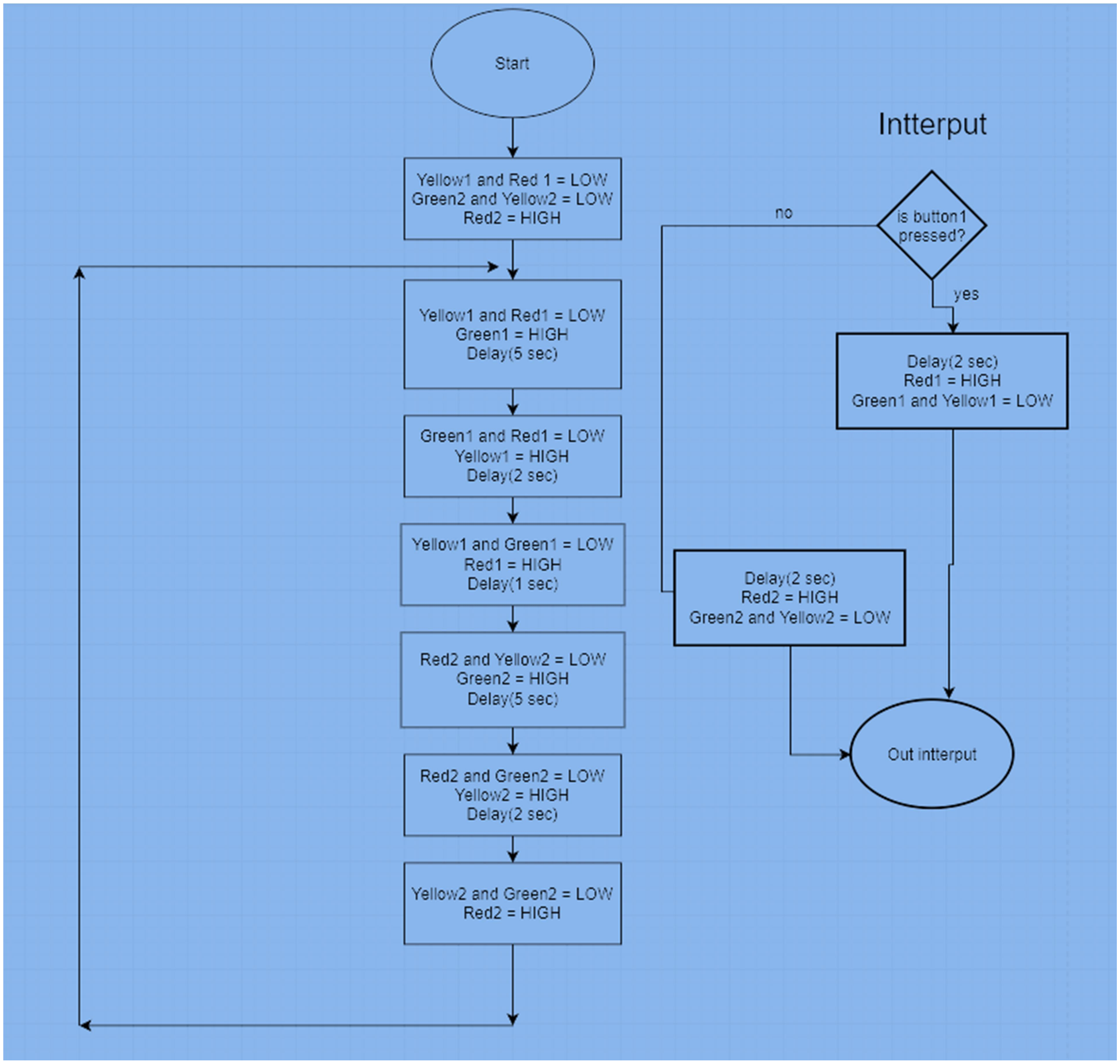
Abstract

The request idea of the project was 2 traffic lights working corresponding to each other as it does in 2 perpendicular streets, the green light of street 1 will work for 5 seconds then the yellow light will work for 2 seconds then the red light will work for 1 second and hold then the green light of street 2 will work and the same sequence of street 1 will happen again here. When the user press one of 2 buttons it will wait 2 seconds then make the traffic light red.

The project is designed and implemented based on the layered architecture model as follow:



Flowchart


Start
Yellow1 and Red 1 = LOW
Green2 and Yellow2 = LOW
Red2 = HIGH
Yellow1 and Red1 = LOW
Green1 = HIGH
Delay(5 sec)
Green1 and Red1 = LOW
Yellow1 = HIGH
Delay(2 sec)
Yellow1 and Green1 = LOW
Red1 = HIGH
Delay(1 sec)
Red2 and Yellow2 = LOW
Green2 = HIGH
Delay(5 sec)
Red2 and Green2 = LOW
Yellow2 = HIGH
Delay(2 sec)
Yellow2 and Green2 = LOW
Red2 = HIGH
Intterput
is button1 pressed?
no
yes
Delay(2 sec)
Red1 = HIGH
Green1 and Yellow1 = LOW
Delay(2 sec)
Red2 = HIGH
Green2 and Yellow2 = LOW
Out intterput

The Video and Src code link

https://drive.google.com/drive/folders/1FtdF8JhA0-0CLr_TflFWyXYXZKMdBxlh?usp=sharing

Code

Main.c

```
#include "tm4c123gh6pm.h"
#include "DIO.h"
#include "keypad.h"
#include <stdio.h>
#include <stdlib.h>
#include "systickk.h"
#include "Timer0.h"
#include "std_types.h"
#include "LEDS_TRAFFIC_INIT.h"
#define INTERRUPTSW2       1
#define INTERRUPTSW1       3
#define SYSTICKSW2        2
#define SYSTICKSW1        4
/******************************** PROJECT TEAM 14
**************************************************************************/
/* Enable Exceptions ... This Macro enable IRQ interrupts, Programmble Systems Exceptions and Faults
by clearing the I-bit in the PRIMASK. */
#define Enable_Exceptions()    __asm("CPSIE I")
/* Disable Exceptions ... This Macro disable IRQ interrupts, Programmble Systems Exceptions and Faults
by clearing the I-bit in the PRIMASK. */
#define Disable_Exceptions()   __asm("CPSID I")
/* Go to low power mode while waiting for the next interrupt */
#define Wait_For_Interrupt()   __asm("WFI")
/* Global variable indicates interrupt triggered/not */
volatile unsigned char interrupt_flag = 0;
 void GPIOPortF_Handler(void)
{
  if(GPIO_PORTF_MIS_R & 0x01 ) {
    interrupt_flag = INTERRUPTSW2 ; /* set the variable to indicate that the interrupt is triggered */
    GPIO_PORTF_ICR_R   |= (1<<0);      /* Clear Trigger flag for PF0 (Interupt Flag) */
  }
  else if (GPIO_PORTF_MIS_R & 0x10 )  {
    interrupt_flag = INTERRUPTSW1  ;  /* set the variable to indicate that the interrupt is triggered */
    GPIO_PORTF_ICR_R   |= (1<<4);       /*  Clear Trigger flag for PF4 (Interupt Flag) */

}
}
uint16 RESERVE_VALUE = 0 ;   // GLOBAL VARIABLE TO SAVE VALUE OF TICKS BEFORE DISABLE TIMER0
uint16 gtick = 0 ;         // GLOBAL GTICK VARIABLE TO COUNT NUMBERS OF SECONDS OF TIMER 0
uint16 sys_gtick = 0 ;   // GLOBAL SYS_GTICK VARIABLE TO COUNT NUMBERS OF SECONDS OF SYSTICK
 int main()
{
  Wait_For_Interrupt() ;
  TRAFFIC_ONE_INIT ();  // INIT THE TRAFFIC LIGHT 1
  TRAFFIC_TWO_INIT () ;  // INIT THE TRAFFIC LIGHT  2
  PEDESTRIAN_TWO_INIT () ; // INIT THE PEDESTRIAN LIGHT  1
  PEDESTRIAN_ONE_INIT () ; // INIT THE PEDESTRIAN LIGHT LIGHT  2
  SysTick_Init() ;     // INIT OF SYSTICK
  TIMER0_INIT () ;   // INIT OF TIEMR0
  WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_HIGH )  ; // INITIALIZE GREEN LIGHT OF TRAFFIC 1
  WRITE_PIN (PORT_E ,PIN_1 ,LOGIC_HIGH )  ;   // INITIALIZE RED LIGHT OF PEDESTRIAN 1
  WRITE_PIN (PORT_A ,PIN_6 ,LOGIC_HIGH )  ; // INITIALIZE RED LIGHT OF  PEDESTRIAN 2
  SW2_Init()  ;      //INITIALIZE SWITCH  2
  SW1_Init();        //INITIALIZE SWITCH   1
 while(1)
   {
Enable_Exceptions() ;       // ENABLE INTERRUPTS
   if( interrupt_flag == INTERRUPTSW2  ) {
       RESERVE_VALUE = gtick  ;
        DISABLE_TIMER0();
       sys_gtick = TIME_OUT () ;
     WRITE_PIN (PORT_E ,PIN_1 ,LOGIC_LOW )  ;
     WRITE_PIN (PORT_E ,PIN_2 ,LOGIC_HIGH )  ;
       if (sys_gtick == 2 )  {
      interrupt_flag = SYSTICKSW2 ;

     }
   }
   else if ( interrupt_flag == INTERRUPTSW1 ) {
      RESERVE_VALUE = gtick  ;
       DISABLE_TIMER0();
      sys_gtick = TIME_OUT () ;
     WRITE_PIN (PORT_A ,PIN_6 ,LOGIC_LOW )  ;
     WRITE_PIN (PORT_A ,PIN_5 ,LOGIC_HIGH )  ;
     if (sys_gtick == 2 )  {
      interrupt_flag = SYSTICKSW1 ;

     }
   }
   if(interrupt_flag == SYSTICKSW2){
     WRITE_PIN (PORT_E ,PIN_1 ,LOGIC_HIGH )  ;
     WRITE_PIN (PORT_E ,PIN_2 ,LOGIC_LOW )  ;
    systick_counter =  0 ;
    interrupt_flag  = 0 ;
    counter = RESERVE_VALUE ;
    TIMER0_INIT () ;
   }
   else if (interrupt_flag == SYSTICKSW1) {
     WRITE_PIN (PORT_A ,PIN_6 ,LOGIC_HIGH )  ;
     WRITE_PIN (PORT_A ,PIN_5 ,LOGIC_LOW )  ;
    systick_counter =  0 ;
   interrupt_flag  = 0 ;
```

```
counter = RESERVE_VALUE ;
  TIMER0_INIT () ;

}

 gtick = CHECK_FLAG()  ;
if(gtick == 0 ) {
WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_HIGH )  ;
WRITE_PIN (PORT_B ,PIN_2 , LOGIC_LOW  ) ;
WRITE_PIN (PORT_B ,PIN_3 , LOGIC_LOW  ) ;
WRITE_PIN (PORT_D,PIN_1 , LOGIC_LOW)  ;
WRITE_PIN (PORT_D ,PIN_2 , LOGIC_LOW ) ;
WRITE_PIN (PORT_D ,PIN_3 , LOGIC_LOW  ) ;

}
if(gtick == 5 ) {
 WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_LOW )  ;
WRITE_PIN (PORT_B ,PIN_2 , LOGIC_HIGH ) ;
WRITE_PIN (PORT_B ,PIN_3 , LOGIC_LOW  ) ;
WRITE_PIN (PORT_D,PIN_1 , LOGIC_LOW)  ;
WRITE_PIN (PORT_D ,PIN_2 , LOGIC_LOW ) ;
WRITE_PIN (PORT_D ,PIN_3 , LOGIC_LOW  ) ;
 }
  if(gtick == 7) {
  WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_LOW )  ;
WRITE_PIN (PORT_B ,PIN_2 , LOGIC_LOW  ) ;
WRITE_PIN (PORT_B ,PIN_3 , LOGIC_HIGH  ) ;
WRITE_PIN (PORT_D,PIN_1 , LOGIC_LOW)  ;
WRITE_PIN (PORT_D ,PIN_2 , LOGIC_LOW ) ;
WRITE_PIN (PORT_D ,PIN_3 , LOGIC_LOW  ) ;
  WRITE_PIN (PORT_E ,PIN_2 , LOGIC_HIGH  ) ;
  WRITE_PIN (PORT_E ,PIN_1 , LOGIC_LOW ) ;
 }
  if(gtick == 8) {
    WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_LOW )  ;
WRITE_PIN (PORT_B ,PIN_2 , LOGIC_LOW  ) ;
WRITE_PIN (PORT_B ,PIN_3 , LOGIC_LOW   ) ;
WRITE_PIN (PORT_D,PIN_1 , LOGIC_HIGH)  ;
WRITE_PIN (PORT_D ,PIN_2 , LOGIC_LOW ) ;
WRITE_PIN (PORT_D ,PIN_3 , LOGIC_LOW  ) ;
 WRITE_PIN (PORT_B ,PIN_3 , LOGIC_HIGH  ) ;
 }
  if(gtick == 13 ) {
  WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_LOW )  ;
WRITE_PIN (PORT_B ,PIN_2 , LOGIC_LOW  ) ;
WRITE_PIN (PORT_B ,PIN_3 , LOGIC_LOW   ) ;
WRITE_PIN (PORT_D,PIN_1 , LOGIC_LOW )  ;
WRITE_PIN (PORT_D ,PIN_2 , LOGIC_HIGH ) ;
WRITE_PIN (PORT_D ,PIN_3 , LOGIC_LOW  ) ;
```

```
     WRITE_PIN (PORT_B ,PIN_3 , LOGIC_HIGH  ) ;
     }
       if(gtick == 14 ) {  // RED LIGHT OF TRAFFIC 2
      WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_LOW )  ;
    WRITE_PIN (PORT_B ,PIN_2 , LOGIC_LOW  ) ;
    WRITE_PIN (PORT_B ,PIN_3 , LOGIC_LOW   ) ;
    WRITE_PIN (PORT_D,PIN_1 , LOGIC_LOW )  ;
    WRITE_PIN (PORT_D ,PIN_2 ,  LOGIC_LOW) ;
    WRITE_PIN (PORT_D,PIN_3 , LOGIC_HIGH  ) ;
     WRITE_PIN (PORT_B ,PIN_3 , LOGIC_HIGH  ) ;
     WRITE_PIN (PORT_A ,PIN_5 ,LOGIC_HIGH )  ;
     WRITE_PIN (PORT_A ,PIN_6 ,LOGIC_LOW )  ;

     }
    if(gtick == 15 ) {    // wait one second all leds off
       WRITE_PIN (PORT_B ,PIN_1 ,LOGIC_LOW )  ;
     WRITE_PIN (PORT_B ,PIN_2 , LOGIC_LOW ) ;
     WRITE_PIN (PORT_B ,PIN_3 , LOGIC_LOW  ) ;
     WRITE_PIN (PORT_D,PIN_1 , LOGIC_LOW)  ;
     WRITE_PIN (PORT_D ,PIN_2 , LOGIC_LOW ) ;
     WRITE_PIN (PORT_D ,PIN_3 , LOGIC_LOW  ) ;
     WRITE_PIN (PORT_E ,PIN_1 ,LOGIC_HIGH )  ;
     WRITE_PIN (PORT_A ,PIN_6 ,LOGIC_HIGH )  ;
     WRITE_PIN (PORT_E ,PIN_2 ,LOGIC_LOW )  ;
     WRITE_PIN (PORT_A ,PIN_5 ,LOGIC_LOW )  ;

    }
    if(gtick == 16 ) {
    counter = 0 ; // extern value of counter in another file
    }
}
}
```

LEDS_TRAFFIC_INIT.h

```
void TRAFFIC_ONE_INIT ();
void TRAFFIC_TWO_INIT ();
void PEDESTRIAN_TWO_INIT ();
void PEDESTRIAN_ONE_INIT () ;
void SW2_Init(void) ;
void SW1_Init(void) ;
```

LEDS_TRAFFIC_INIT.c

```
#include "tm4c123gh6pm.h"
#include "DIO.h"
#define GPIO_PORTF_PRIORITY_MASK      0xFF1FFFFF
#define GPIO_PORTF_PRIORITY_BITS_POS  21
```

```
#define GPIO_PORTF_INTERRUPT_PRIORITY 2
#define NUMBER_OF_ITERATIONS_PER_ONE_MILI_SECOND 762
void TRAFFIC_ONE_INIT (){
DIO_init ( PORT_B ,PIN_1 ,PIN_OUTPUT ) ; // LED GREEN TRAFFIC 1
 DIO_init ( PORT_B ,PIN_2 ,PIN_OUTPUT ) ;  // LED YELLOW TRAFFIC 1
 DIO_init ( PORT_B ,PIN_3 ,PIN_OUTPUT ) ;  // LED TRAFFIC 1
   GPIO_PORTB_AMSEL_R &= 0xF1;        /* Disable Analog on PB1, PB2 and PB3 */
   GPIO_PORTB_PCTL_R  &= 0xFFFF000F;   /* Clear PMCx bits for PB1, PB2 and PB3 to use it as GPIO pin
*/
   GPIO_PORTB_DIR_R   |= 0x0E;        /* Configure PB1, PB2 and PB3 as output pin */
   GPIO_PORTB_AFSEL_R &= 0xF1;        /* Disable alternative function on PB1, PB2 and PB3 */
   GPIO_PORTB_DEN_R   |= 0x0E;        /* Enable Digital I/O on PB1, PB2 and PB3 */
   GPIO_PORTB_DATA_R  &=  0xF1;        /* Clear bit 3 , 1 and 2 in Data regsiter to turn off the leds */
}
void TRAFFIC_TWO_INIT (){

DIO_init ( PORT_D ,PIN_1 ,PIN_OUTPUT ) ; // LED GREEN TRAFFIC 2
 DIO_init ( PORT_D ,PIN_2 ,PIN_OUTPUT ) ;  // LED YELLOW TRAFFIC 2
 DIO_init ( PORT_D ,PIN_3 ,PIN_OUTPUT ) ;  // LED TRAFFIC 2
   GPIO_PORTD_AMSEL_R &= 0xF1;        /* Disable Analog on PD1, PD2 and PD3 */
   GPIO_PORTD_PCTL_R  &= 0xFFFF000F;   /* Clear PMCx bits for PD1, PD2 and PE3 to use it as GPIO pin
*/
   GPIO_PORTD_DIR_R   |= 0x0E;        /* Configure PD1, PD2 and PD3 as output pin */
   GPIO_PORTD_AFSEL_R &= 0xF1;        /* Disable alternative function on PD1, PD2 and PD3 */
   GPIO_PORTD_DEN_R   |= 0x0E;        /* Enable Digital I/O on PD1, PD2 and PD3 */
   GPIO_PORTD_DATA_R  &=  0xF1;        /* Clear bit 3 , 1 and 2 in Data regsiter to turn off the leds */
}
void PEDESTRIAN_ONE_INIT (){

DIO_init ( PORT_E ,PIN_1 ,PIN_OUTPUT ) ; // LED red PEDESTRIAN 1
 DIO_init ( PORT_E ,PIN_2 ,PIN_OUTPUT ) ;  // LED green PEDESTRIAN 1
  GPIO_PORTD_AMSEL_R &= 0xF9;        /* Disable Analog on PD1, PD2 and PD3 */
   GPIO_PORTE_PCTL_R  &= 0xFFFFF00F;   /* Clear PMCx bits for PD1, PD2 and PE3 to use it as GPIO pin
*/
   GPIO_PORTE_DIR_R   |= 0x06;        /* Configure PD1, PD2 and PD3 as output pin */
   GPIO_PORTE_AFSEL_R &= 0xF9;        /* Disable alternative function on PD1, PD2 and PD3 */
   GPIO_PORTE_DEN_R   |= 0x06;        /* Enable Digital I/O on PD1, PD2 and PD3 */
   GPIO_PORTE_DATA_R  &=  0xF9;        /* Clear bit 3 , 1 and 2 in Data regsiter to turn off the leds */
}

void PEDESTRIAN_TWO_INIT (){

DIO_init ( PORT_A ,PIN_6 ,PIN_OUTPUT ) ; // LED red PEDESTRIAN 1
DIO_init ( PORT_A ,PIN_5 ,PIN_OUTPUT ) ;  // LED green PEDESTRIAN 1
   GPIO_PORTA_AMSEL_R &= 0x9F;        /* Disable Analog on PD1, PD2 and PD3 */
   GPIO_PORTA_PCTL_R  &= 0xF00FFFFF;   /* Clear PMCx bits for PD1, PD2 and PE3 to use it as GPIO pin
*/
   GPIO_PORTA_DIR_R   |= 0x60;        /* Configure PD1, PD2 and PD3 as output pin */
   GPIO_PORTA_AFSEL_R &= 0x9F;        /* Disable alternative function on PD1, PD2 and PD3 */
```

```
    GPIO_PORTA_DEN_R   |= 0x60;         /* Enable Digital I/O on PD1, PD2 and PD3 */
    GPIO_PORTA_DATA_R  &=  0x9F;        /* Clear bit 3 , 1 and 2 in Data regsiter to turn off the leds */
}
void SW2_Init(void)
{
    GPIO_PORTF_LOCK_R  = 0x4C4F434B;   /* Unlock the GPIO_PORTF_CR_REG */
    GPIO_PORTF_CR_R    |= (1<<0);       /* Enable changes on PF0 */
    GPIO_PORTF_AMSEL_R &= ~(1<<0);      /* Disable Analog on PF0 */
    GPIO_PORTF_PCTL_R  &= 0xFFFFFFF0;   /* Clear PMCx bits for PF0 to use it as GPIO pin */
    GPIO_PORTF_DIR_R   &= ~(1<<0);      /* Configure PF0 as input pin */
    GPIO_PORTF_AFSEL_R &= ~(1<<0);      /* Disable alternative function on PF0 */
    GPIO_PORTF_PUR_R   |= (1<<0);       /* Enable pull-up on PF0 */
    GPIO_PORTF_DEN_R   |= (1<<0);       /* Enable Digital I/O on PF0 */
    GPIO_PORTF_IS_R    &= ~(1<<0);      /* PF0 detect edges */
    GPIO_PORTF_IBE_R   &= ~(1<<0);      /* PF0 will detect a certain edge */
    GPIO_PORTF_IEV_R   &= ~(1<<0);      /* PF0 will detect a falling edge */
    GPIO_PORTF_ICR_R   |= (1<<0);       /* Clear Trigger flag for PF0 (Interupt Flag) */
    GPIO_PORTF_IM_R    |= (1<<0);       /* Enable Interrupt on PF0 pin */
    /* Set GPIO PORTF priotiy as 2 by set Bit number 21, 22 and 23 with value 2 */
    NVIC_PRI7_R = (NVIC_PRI7_R & GPIO_PORTF_PRIORITY_MASK) |
(GPIO_PORTF_INTERRUPT_PRIORITY<<GPIO_PORTF_PRIORITY_BITS_POS);
    NVIC_EN0_R        |= 0x40000000;   /* Enable NVIC Interrupt for GPIO PORTF by set bit number 30 in
EN0 Register */
}
 void SW1_Init(void)
{
    GPIO_PORTF_AMSEL_R &= ~(1<<4);      /* Disable Analog on PF4 */
    GPIO_PORTF_PCTL_R  &= 0xFFF0FFFF;   /* Clear PMCx bits for PF4 to use it as GPIO pin */
    GPIO_PORTF_DIR_R   &= ~(1<<4);      /* Configure PF4 as input pin */
    GPIO_PORTF_AFSEL_R &= ~(1<<4);      /* Disable alternative function on PF4 */
    GPIO_PORTF_PUR_R   |= (1<<4);       /* Enable pull-up on PF4 */
    GPIO_PORTF_DEN_R   |= (1<<4);       /* Enable Digital I/O on PF4 */
    GPIO_PORTF_IS_R    &= ~(1<<4);      /* PF4 detect edges */
    GPIO_PORTF_IBE_R   &= ~(1<<4);      /* PF4 will detect a certain edge */
    GPIO_PORTF_IEV_R   &= ~(1<<4);      /* PF4 will detect a falling edge */
    GPIO_PORTF_ICR_R   |= (1<<4);       /* Clear Trigger flag for PF4 (Interupt Flag) */
    GPIO_PORTF_IM_R    |= (1<<4);       /* Enable Interrupt on PF4 pin */
    /* Set GPIO PORTF priotiy as 2 by set Bit number 21, 22 and 23 with value 2 */
    NVIC_PRI7_R = (NVIC_PRI7_R & GPIO_PORTF_PRIORITY_MASK) |
(GPIO_PORTF_INTERRUPT_PRIORITY<<GPIO_PORTF_PRIORITY_BITS_POS);
    NVIC_EN0_R        |= 0x40000000;   /* Enable NVIC Interrupt for GPIO PORTF by set bit number 30 in
EN0 Register */
}
```

Timer0.h

```
#include "std_types.h"
extern uint16 counter ;
void TIMER0_INIT (void) ;
uint16 CHECK_FLAG() ;
void DISABLE_TIMER0();
void ENABLE_TIMER0() ;
```

Timer0.c

```
#include "tm4c123gh6pm.h"
#include "DIO.h"
#include "Timer0.h"
#include "common_macros.h"
#include "std_types.h"
uint16 counter = 0 ;
void TIMER0_INIT (void) {
 SET_BIT(SYSCTL_RCGCTIMER_R ,0) ;  //START THE CLOCK OF TIMER 0
 CLEAR_BIT(TIMER0_CTL_R, 0) ;        // DISABLE TIMER BEFORE CONFIGURATION
 TIMER0_CFG_R    =  TIMER_CFG_32_BIT_TIMER   ; //32_BIT TIMER
 TIMER0_TAMR_R  |= (0X2<<0) ;   // PERIODIC COUNTER
  CLEAR_BIT(TIMER0_TAMR_R  , 4 ) ;     // DOWN COUNTER
  TIMER0_TAILR_R  = 0X00F42400 ;  //16000000 COUNTSA TO COUNT ONE SECOND
  SET_BIT(TIMER0_CTL_R, 0) ;  // ENABLE TIMER0
}

uint16 CHECK_FLAG(){
  if(TIMER0_RIS_R & 0x00000001 ) {
    SET_BIT(TIMER0_ICR_R, 0); // CLEAR FLAG
    counter ++ ;  // INCREASES THE COUNTER EVERY ENTIRY FOR THE FUNCTION
  }
return counter ;
}
void DISABLE_TIMER0(){
CLEAR_BIT(TIMER0_CTL_R, 0) ;        // DISABLE TIMER BEFORE CONFIGURATION
}
void ENABLE_TIMER0(){
SET_BIT(TIMER0_CTL_R, 0) ;  // ENABLE TIMER0
}
```

systickk.h

```
#include "tm4c123gh6pm.h"
#include "std_types.h"
/******************************REFISTERS DEFINITIONS ************************/
#define   CURRENT_REGISTER   NVIC_ST_CURRENT_R
#define   SYSTICK_CONTROL_REG    NVIC_ST_CTRL_R
#define   RELOAD_REG             NVIC_ST_RELOAD_R
#define   RELOAD_FLAG             (SYSTICK_CONTROL_REG & 1<<16)
extern uint16 systick_counter ;
/************************FUNCTIONS PROTOTYPES *****************************/
void SysTick_Init(void) ;
uint16  TIME_OUT (void) ;
void DISABLE_SYSTICK () ;
/**********************************************************/
```

systickk.c

```
#include "systickk.h"
#include "tm4c123gh6pm.h"
#include "std_types.h"
#include "DIO.h"
#define TIME_IN_MILIS    50000
#define SYSTEM_FREQ_TH  16000
uint16  systick_counter = 0 ;

void SysTick_Init(void)
{
   SYSTICK_CONTROL_REG    = 0;             /* Disable the SysTick Timer by Clear the ENABLE Bit */
    RELOAD_REG  = 15999999;       /* Set the Reload value with 15999999 to count ONE Second */
    CURRENT_REGISTER = 0;             /* Clear the Current Register value */
    /* Configure the SysTick Control Register
     * Enable the SysTick Timer (ENABLE = 1)
     * Disable SysTick Interrupt (INTEN = 0)
     * Choose the clock source to be System Clock (CLK_SRC = 1) */
      SYSTICK_CONTROL_REG |= 0x05;
}
uint16 TIME_OUT (void){
  if(RELOAD_FLAG ==0) {
  }
  else {
    systick_counter ++ ;
     return  systick_counter  ;
}
}
void DISABLE_SYSTICK () {

 SYSTICK_CONTROL_REG &= 0x00;
}
```

DIO.h

```
#include "tm4c123gh6pm.h"
#ifndef DIO_H
#define DIO_H
#define NUM_TIMERS 762
#define ENABLE_DIGITAL 0XFF
# define PORT_A 0
# define PORT_B 1
# define PORT_C 2
# define PORT_D 3
# define PORT_E 4
# define PORT_F 5
# define PIN_0 0
# define PIN_1 1
# define PIN_2 2
# define PIN_3 3
# define PIN_4 4
# define PIN_5 5
# define PIN_6 6
# define PIN_7 7
# define PIN_8 8
# define DIR_LOW  0
# define DIR_HIGH 1
#define PORT_LOW  0X00
#define PORT_HIGH 0XFF
typedef enum {
PIN_INPUT , PIN_OUTPUT
} PIN_DIRECTION ;
typedef enum {
 LOGIC_LOW , LOGIC_HIGH
} VALUE_PIN  ;
int DIO_ReadPort (char Port_num) ;
 char DIO_ReadPin(char Port_num, char Pin_num);
void DIO_init ( char port_num , char pin_num , PIN_DIRECTION direction );
void WRITE_PIN (char port_num,char pin_num , VALUE_PIN  value  ) ;
void DIO_WritePort ( char port_num  , char value ) ;


#endif
```

DIO.c

```
#include "tm4c123gh6pm.h"
#include "DIO.h"
#include "common_macros.h"
int DIO_ReadPort (char Port_num)
{
  volatile unsigned int val;
  switch(Port_num)
  {
  case 0:
    val= GPIO_PORTA_DATA_R;
    break;

  case 1:
    val= GPIO_PORTB_DATA_R;
    break;

  case 2:
    val= GPIO_PORTC_DATA_R;
    break;

  case 3:
    val= GPIO_PORTD_DATA_R;
    break;

  case 4:
    val= GPIO_PORTE_DATA_R;
    break;

  case 5:
    val= GPIO_PORTF_DATA_R;
    break;

  }
  return val;
}
char DIO_ReadPin(char Port_num, char Pin_num)
{
    unsigned int val;
  switch(Port_num)
  {
     case  0 :
   switch(Pin_num)
   {
   case 0:
     val = GET_BIT (GPIO_PORTA_DATA_R, 0);
  break;
case 1:
  val = GET_BIT(GPIO_PORTA_DATA_R, 1);
  break;
case 2:
  val = GET_BIT(GPIO_PORTA_DATA_R, 2);
  break;
case 3:
  val = GET_BIT(GPIO_PORTA_DATA_R, 3);
  break;
 case 4:
  val = GET_BIT(GPIO_PORTA_DATA_R, 4);
  break;
  case 5:
  val = GET_BIT(GPIO_PORTA_DATA_R, 5);
  break;
  case 6:
  val = GET_BIT(GPIO_PORTA_DATA_R, 6);
  break;
  case 7:
  val = GET_BIT(GPIO_PORTA_DATA_R, 7);
  break;
}
break;
  case  1 :
switch(Pin_num)
{
case 0:
  val = GET_BIT(GPIO_PORTB_DATA_R, 0);
  break;
case 1:
  val = GET_BIT(GPIO_PORTB_DATA_R, 1);
  break;
case 2:
  val = GET_BIT(GPIO_PORTB_DATA_R, 2);
  break;
case 3:
  val = GET_BIT(GPIO_PORTB_DATA_R, 3);
  break;
 case 4:
  val = GET_BIT(GPIO_PORTB_DATA_R, 4);
  break;
  case 5:
  val = GET_BIT(GPIO_PORTB_DATA_R, 5);
  break;
  case 6:
  val = GET_BIT(GPIO_PORTB_DATA_R, 6);
  break;
  case 7:
```

```
  val = GET_BIT(GPIO_PORTB_DATA_R, 7);
  break;
}
break;
  case  2 :
switch(Pin_num)
{
case 0:
  val = GET_BIT(GPIO_PORTC_DATA_R, 0);
  break;
case 1:
  val = GET_BIT(GPIO_PORTC_DATA_R, 1);
  break;
case 2:
  val = GET_BIT(GPIO_PORTC_DATA_R, 2);
  break;
case 3:
  val = GET_BIT(GPIO_PORTC_DATA_R, 3);
  break;
 case 4:
  val = GET_BIT(GPIO_PORTC_DATA_R, 4);
  break;
  case 5:
  val = GET_BIT(GPIO_PORTC_DATA_R, 5);
  break;
  case 6:
  val = GET_BIT(GPIO_PORTC_DATA_R, 6);
  break;
  case 7:
  val = GET_BIT(GPIO_PORTC_DATA_R, 7);
  break;
}
break;
 case  3 :
switch(Pin_num)
{
case 0:
  val = GET_BIT(GPIO_PORTD_DATA_R, 0);
  break;
case 1:
  val = GET_BIT(GPIO_PORTD_DATA_R, 1);
  break;
case 2:
  val = GET_BIT(GPIO_PORTD_DATA_R, 2);
  break;
case 3:
  val = GET_BIT(GPIO_PORTD_DATA_R, 3);
  break;
 case 4:
   break;
  case 5:
  val = GET_BIT(GPIO_PORTD_DATA_R, 5);
  break;
  case 6:
  val = GET_BIT(GPIO_PORTD_DATA_R, 6);
  break;
  case 7:
  val = GET_BIT(GPIO_PORTD_DATA_R, 7);
  break;
 }
 break;
  case  4 :

 switch(Pin_num)
 {
 case 0:
  val = GET_BIT (GPIO_PORTE_DATA_R, 0);
  break;
 case 1:
  val = GET_BIT(GPIO_PORTE_DATA_R, 1);
  break;
 case 2:
  val = GET_BIT(GPIO_PORTE_DATA_R, 2);
  break;
 case 3:
  val = GET_BIT(GPIO_PORTE_DATA_R, 3);
  break;
  case 4:
  val = GET_BIT(GPIO_PORTE_DATA_R, 4);
  break;
 }
 break;
case  5 :

 switch(Pin_num)
 {
 case 0:
  val = GET_BIT(GPIO_PORTF_DATA_R, 0);
  break;
 case 1:
  val = GET_BIT(GPIO_PORTF_DATA_R, 1);
  break;
 case 2:
  val = GET_BIT(GPIO_PORTF_DATA_R, 2);
  break;
 case 3:
  val = GET_BIT(GPIO_PORTF_DATA_R, 3);
  break;
```

```
    case 4:
     val = GET_BIT(GPIO_PORTF_DATA_R, 4);
     break;
   }
   break;
  }
  return val;
}
void DIO_init ( char port_num,char pin_num ,PIN_DIRECTION direction )
{
  switch( port_num  )
  {
  case 0 :
    SYSCTL_RCGCGPIO_R |= 0x00000001;
    switch (direction)
    {
    case 0 :
      CLEAR_BIT(GPIO_PORTA_DIR_R,pin_num);
      SET_BIT(GPIO_PORTA_PUR_R,pin_num);
      break;
      case 1 :
      SET_BIT(GPIO_PORTA_DIR_R,pin_num);
      break;
    }
  case 1 :
    SYSCTL_RCGCGPIO_R |= 0x00000002;
    switch (direction)
    {
     case 0 :
      CLEAR_BIT(GPIO_PORTB_DIR_R,pin_num);
      SET_BIT(GPIO_PORTB_PUR_R,pin_num);
      break;
      case 1 :
      SET_BIT(GPIO_PORTB_DIR_R,pin_num);
      break;
    }
  case 2 :
    SYSCTL_RCGCGPIO_R |= 0x00000004;
    switch (direction)
    {
    case 0 :
      CLEAR_BIT(GPIO_PORTC_DIR_R,pin_num);
      SET_BIT(GPIO_PORTC_PUR_R,pin_num);
      break;
      case 1 :
      SET_BIT(GPIO_PORTC_DIR_R,pin_num);
      break;
    }
  case 3 :
    SYSCTL_RCGCGPIO_R |= 0x00000008;
    switch (direction)
    {
    case 0 :
      CLEAR_BIT(GPIO_PORTD_DIR_R,pin_num);
      SET_BIT(GPIO_PORTD_PUR_R,pin_num);
      break;
      case 1 :
      SET_BIT(GPIO_PORTD_DIR_R,pin_num);
      break;
    }
  case 4 :
    SYSCTL_RCGCGPIO_R |= 0x00000010;
     GPIO_PORTE_LOCK_R = 0x4C4F434B;
    switch (direction)
    {
     case 0 :
      CLEAR_BIT(GPIO_PORTE_DIR_R,pin_num);
      SET_BIT(GPIO_PORTE_PUR_R,pin_num);
      break;
      case 1 :
      SET_BIT(GPIO_PORTE_DIR_R,pin_num);
      break;
    }
  case 5 :
    SYSCTL_RCGCGPIO_R |= 0x00000020;
     GPIO_PORTF_LOCK_R = 0x4C4F434B;
      GPIO_PORTF_DEN_R = ENABLE_DIGITAL;
    switch (direction)
    {
    case 0 :
      CLEAR_BIT(GPIO_PORTF_DIR_R,pin_num);
      SET_BIT(GPIO_PORTF_PUR_R,pin_num);
      break;
      case 1 :
      SET_BIT(GPIO_PORTF_DIR_R,pin_num);
      break;
    }
  }
}
void WRITE_PIN (char port_num,char pin_num , VALUE_PIN  value  ){
  switch( port_num  ) {
     case 0 :
    switch (value) {
    case 0 :
    CLEAR_BIT(GPIO_PORTA_DATA_R ,pin_num);
    break;
    case 1 :
      SET_BIT(GPIO_PORTA_DATA_R ,pin_num);
      break;
  }
  break;
     case 1 :
    switch (value) {
    case 0 :
    CLEAR_BIT(GPIO_PORTB_DATA_R ,pin_num);
    break;
    case 1 :
      SET_BIT(GPIO_PORTB_DATA_R ,pin_num);
      break;
  }
  break;
     case 2 :
    switch (value) {
    case 0 :
    CLEAR_BIT(GPIO_PORTC_DATA_R ,pin_num);
    break;
    case 1 :
      SET_BIT(GPIO_PORTC_DATA_R ,pin_num);
      break;
  }
  break;
  case 3 :
    switch (value) {
    case 0 :
    CLEAR_BIT(GPIO_PORTD_DATA_R ,pin_num);
    break;
    case 1 :
      SET_BIT(GPIO_PORTD_DATA_R ,pin_num);
      break;
  }
  break;
   case 4 :
    switch (value) {
    case 0 :
    CLEAR_BIT(GPIO_PORTE_DATA_R ,pin_num);
    break;
    case 1 :
      SET_BIT(GPIO_PORTE_DATA_R ,pin_num);
      break;
  }
  break;
   case 5 :
    switch (value) {
    case 0 :
    CLEAR_BIT(GPIO_PORTF_DATA_R ,pin_num);
    break;
    case 1 :
      SET_BIT(GPIO_PORTF_DATA_R ,pin_num);
      break;
  }
  break;
  }
}
void DIO_WritePort ( char port_num , char value ){
     switch( port_num)
  {
  case 0 :
    SYSCTL_RCGCGPIO_R |= 0x00000001;
    GPIO_PORTA_DATA_R  = value ;
    break;
     case 1 :
    SYSCTL_RCGCGPIO_R |= 0x00000002;
    GPIO_PORTB_DATA_R = value ;
    break;
     case 2 :
    SYSCTL_RCGCGPIO_R |= 0x00000004;
    GPIO_PORTC_DATA_R = value ;
    break;
     case 3 :
    SYSCTL_RCGCGPIO_R |= 0x00000008;
    GPIO_PORTD_DATA_R = value ;
    break;
     case 4 :
    SYSCTL_RCGCGPIO_R |= 0x000000010;
    GPIO_PORTE_DATA_R = value ;
    break;
     case 5 :
    SYSCTL_RCGCGPIO_R |= 0x00000020;
    GPIO_PORTF_DATA_R = value ;
    break;
  }
}
```

common_macros.h

```
#ifndef COMMON_MACROS
#define COMMON_MACROS

#define SET_BIT(REG,BIT) (REG|=(1<<BIT))
#define CLEAR_BIT(REG,BIT) (REG&=(~(1<<BIT)))
#define GET_BIT(REG,BIT) (REG & (1<<BIT) )
#define TOGGLE_BIT(REG,BIT) (REG^=(1<<BIT))
#define GET_BIT(REG,BIT) (REG & (1<<BIT) )
#define ROR(REG,num) ( REG= (REG>>num) | (REG<<(8-num)) )


#define ROL(REG,num) ( REG= (REG<<num) | (REG>>(8-num)) )


#define BIT_IS_SET(REG,BIT) ( REG & (1<<BIT) )

#define BIT_IS_CLEAR(REG,BIT) ( !(REG & (1<<BIT)) )

#endif
```

Std_types.h

```
#ifndef STD_TYPES_H_
#define STD_TYPES_H_

/* Boolean Data Type */
typedef unsigned char boolean;

/* Boolean Values */
#ifndef FALSE
#define FALSE      (0u)
#endif
#ifndef TRUE
#define TRUE       (1u)
#endif

/*#define LOGIC_HIGH      (1u)
#define LOGIC_LOW       (0u)*/

#define NULL_PTR    ((void*)0)

typedef unsigned char       uint8;        /*          0 .. 255              */
typedef signed char         sint8;        /*       -128 .. +127             */
typedef unsigned short      uint16;       /*          0 .. 65535            */
typedef signed short        sint16;     /*      -32768 .. +32767          */
typedef unsigned long       uint32;     /*           0 .. 4294967295      */
typedef signed long         sint32;     /* -2147483648 .. +2147483647     */
typedef unsigned long long  uint64;     /*      0 .. 18446744073709551615  */
typedef signed long long    sint64;     /* -9223372036854775808 .. 9223372036854775807 */
typedef float               float32;
typedef double              float64;

#endif /* STD_TYPE_H_ */
```