INTERFACING THE K60 MICROCONTROLLER

Course: ENGG*3640 Microcomputer Interfacing

Instructor: Radu Muresan

Student Names/Numbers:

Bilal Ayyache (0988616)

Emeka Madubuike (0948959)

Mohammed Al-Fakhri (0982745)

Ali Akhdar (1068542)

Date: 31/10/2018

Contents

1. Introduction

Lab four introduces the use of GPIO functionality system. This lab explains how the system is used to control components such as LEDS, timers such as PIT, and 7 segment displays. Through this lab, it is vey crucial to understand what GPIO defines as. K60 GPIO is a general-purpose input and output communication module to the processor core. For general purpose input, The logic state of each input pin is available via the Port Data Input registers, provided the pin is configured for a digital function and the corresponding Port Control and Interrupt module is enabled. For General purpose output, The logic state of each output pin can be controlled via the port data output registers and port data direction registers, provided the pin is configured for the GPIO function. The GPIO registers support 8-bit, 16-bit or 32-bit accesses. K60 GPIO has multiple features such as:

- Pin input data register visible in all digital pin- multiplexing modes
- Pin output data register with corresponding set/clear/toggle registers
- Pin data direction register
- Zero wait state access to GPIO registers

These multiple features can operate in different modes. The modes are run, wait, stop, and finally debug. GPIO modules supports 5x32 bit ports which consists of Port A, B, C, D and E.

1.1. Lab Description

In lab three, the concept of using the LPTMR (low power timer) was introduced to implement an interrupt system. In lab four, the PIT (Periodic interrupt timer) was used to signal interrupts. These interrupts called a handler function. These requests where later executed to alternate between two on board LEDs. Section 3.4.2 goes in deeper details on how the PIT Timer for the system was implemented. To prepare for such complicated implementation, the first objective of the lab was to understand how a simpler GPIO system work. To help understand how GPIO work, A blink project was implemented in which 3 LEDs blink at every second. The project file was supplied by the course advisor. More details about this project can be found in Section 3.4.1. After understanding how the GPIO system functions, the next objective was to implement a new GPIO application that blinks the LEDs on the board. PIT Timer was used to implement project. To further understand how GPIO functions, the next challenge was to implement a voltmeter device using a 3 digit 7 segment display, resistors, and transistors that was provided by the lab kit. More details about this project can be found in section 3.4.3.

In summary the main purpose of lab four was to further encourage the understanding of interrupt timers, controlling physical aspects of the freescale K60 board, and developing an interface between the microprocessor to a 7-segment display.

1.2. System Requirements

During this lab, the tools and equipment that were used are enumerated below:

- Kiel Uvision Program was used to create instructions to the K60 Microcontroller
- FreeScale TWR-K60D100M Microcontroller was used to implement instructions sent
- A Hi Speed USB 2.0 Connection cable between PC and board was used to connect the Microcontroller to the PC.
- Putty displayed the results by receiving commands through the COM port in which the USB 2.0 was connected to.
- To implement the circuit design, a 7 Segment display, transistors, resistors, and a breadboard was used. (More information of equipment can be found in section 2.1).

2. Background

2.1. Equipment

- **Kiel Uvision Program**: The Kiel Uvision program is an IDE that combines project management, source code editing, program debugging, and run-time environment into a single powerful environment. Using this environment, the user is able to easily and efficiently test, verify, debug, and optimize the code developed. During the third lab, the debug functionality helped in understanding how the code is acting.

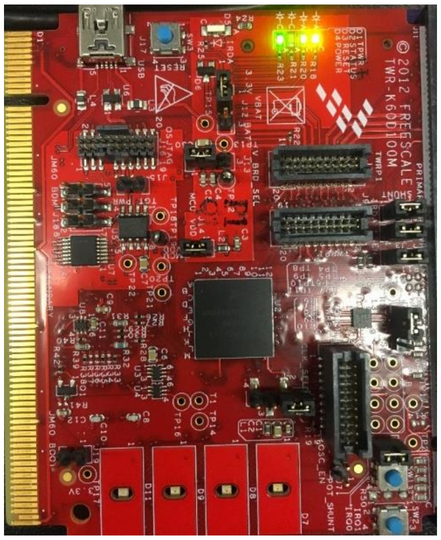


Figure 2.1 1K60 Microcontroller

- **K60 Microcontroller**: The K60 Microcontroller (Figure 2.1) from NXP contains a low power MCU core ARM Cortex-M4 that features an analog integration, serial communication, USB 2.0 full-speed OTG controller and 10/100 Mbps Ethernet MAC. These characteristics makes this Microcontroller suitable to preform task in a very efficient and fast manner. A USB connection was used to sync the microcontrollerwith the Keil uVision software that that was provided by the teaching assistant. Through Lab 3, The main feature that was used was the NVIC component.
- **BreadBoard:** A breadboard is a construction base for prototyping of electronics. In this lab we used a breadboard to implement the logic functions by connecting our IC Chips' pins to satisfy the objective of this lab. IC Chips were placed between the board's valley and pins were connected using jumper wires.
- **3 digit 7 segment display:** consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed. The displays common pin is generally used to identify which type of 7-segment display it is. As each LED has two connecting pins, one called the "Anode" and the other called the "Cathode", there are therefore two types of LED 7-segment display called: Common Cathode (CC) and Common Anode (CA).
- **Resistors** A resistor is a circuit element used to model the current resisting behavior of a material. For the purpose of constructing circuits, resistors are usually made from metallic alloys and carbon compounds
- **Transistors** a transistor is a "nonlinear" component that has three leads. Transistors can be used for switching and amplifying signals.

3. Implementation

As an implementation overview of this application, it was required to develop an application that shows the output on a 7-segment LED display screen. The implementation of this lab was divided to two parts. The first one is the software part that discusses how the code implementation was done including the configuration of all the registers. Many obstacles were encountered in this part and to overcome these, the code outline was written on a paper and debugged before typing into uVision Keil for execution. The second part is basically discussing the hardware components used in the implementation of this application like transistors, resistors, jumper wires and 7-segment LED.

3.1. Lab Implementation Overview

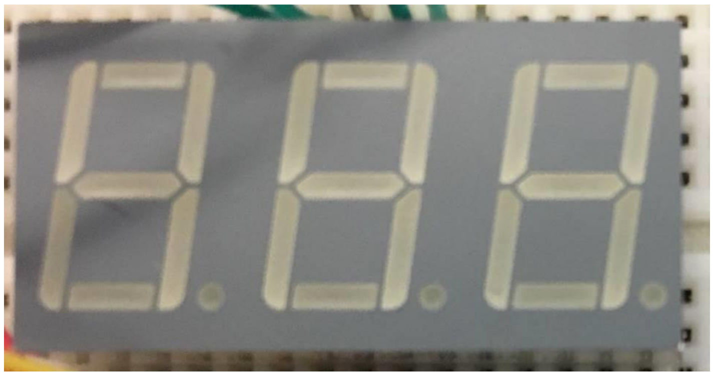


Figure 3.1 Running the Code

Figure 3.1 represents an overview of the implementation of this application without any input or output displayed, it shows the initial condition of this project before any changes happen, neither on the 7-segment LED display as an output nor in the code as an input.

3.1.1. Software Implementation

To implement the software, the program starts off by configuring the PIT0 Timer (Line 150-154). This is done by configuring the designed registers (PCR Within Port A, PDOR, PDDR, and the PSOR). After PIT0 have been configured, the NVIC_ENABLEDIRQ is configured to receive the interrupts generated (Line 133). The PIT Timer is then configured as seen in line 132-135. In line 150-153 the timer values are set by configuring the Pit channels. After timer values are set, port C is configured in order to send low signals to the 7 segment LED Selected display digits(Line 135/140). The display digits are selected manually and loaded using the PSOR register (Line 27-45). After loading is complete, all segments are cleared using PCOR register and a small delay is set. The 7 segment display is then loaded with the hardcoded digit (Line 155-175, active low for segments and high for display selection). All segments are later cleared using the PCOR register (Line 85-88). The third display is then loaded using the PSOR as seen in line 94-97 (active low for segments and high for display selection).

3.1.2. Hardware Implementation

As shown in the circuit, when the user resets the microcomputer the program runs. The 3 digit LED simply works when voltage is applied, so if the power source applies high on all segments and the microcomputer also applies high , the segments will not light up because the flow on both sides is the same which will cause the flow to stop and no voltage is going through the circuit. If the user inputs a number for example “1”, the board will set segments “f” and “e” low and the remaining segments to high this will cause segments “f” and “e” to light up.

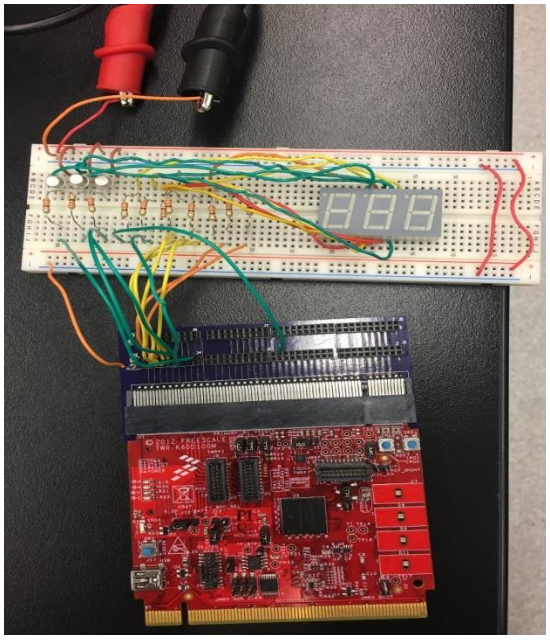


Figure 3.1. 1. Circuit Connection

3.2. Simulation Results

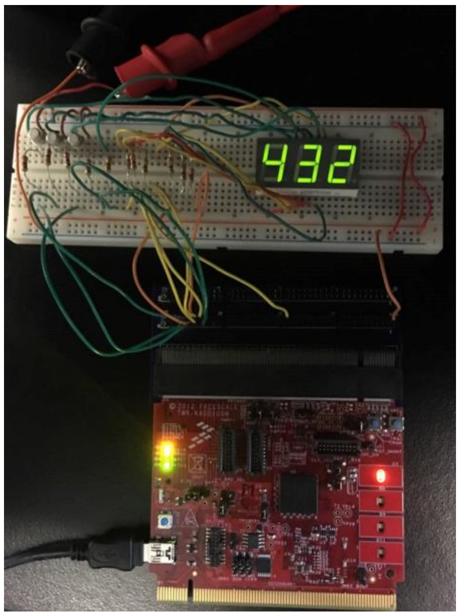


Figure 3.2. 1Results of running code with inputs

As shown in figure 3.2.1, after the code is uploaded on the board. The 3 digit LED will then display the number that has been coded by the user and keeps displaying it until the user changes it or the program gets terminated.

3.3. Block Diagram

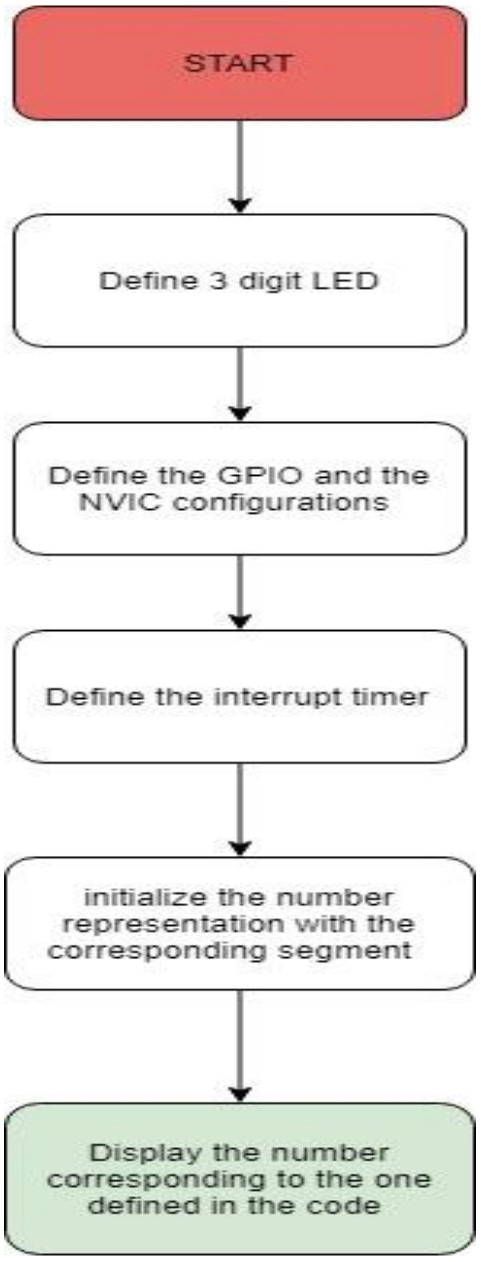

START
Define 3 digit LED
Define the GPIO and the NVIC configurations
Define the interrupt timer
initialize the number representation with the corresponding segment
Display the number corresponding to the one defined in the code

Figure 3.3. Block diagram

The first thing that the code does when it starts is defining 3 digit LED after that the program starts to define and initialize the GPIO and the (PIT) interrupt timer. Then the program initialize the numbers that will be used with their segment representation, then displaying it depending on the segment choosen by the user.

3.4. Lab Requirements

3.4.1. Lab Requirement 1

In part one of the lab the software implementation was provided by the teacher assistant. Using the provided program, the process of using the ports was explained and understood further with thorough debugging. The SIM port was used to enable the clock in port C. It also checks the SIM_Type structure in the MK60D10.h file. Referring to line 69 “SIM->SCGC5 |= (1UL << 9)” will move 1 in bit position 9 of the register SIM_SCGC5 and perform a logic Or to set bit 9 and then write back. In line 29 to 31 PORTA -> PCR[11] PORTA -> PCR[28] and PORTA -> PCR[29] control the registers that are associated with PORTA pins 11, 28, 29. These control registers are part of Port Control and Interrupt (PORT) module. In line 33 PTA -> PDOR register configures the logic levels that are driven on each general-purpose output. In line 36 PTA ->PDDR register configures the individual pins for input or output. In lines 46 PTA -> PCOR register configures whether to clear the fields of PDOR register. If the bit is set to 1 then the corresponding bit in PDOR will be clear. In lines 55 PTA -> PSOR register configures whether to set the fields of the PDOR register. If the bit is set to 1 then the corresponding bit in the PDOR will be set.

3.4.2. Lab Requirement 2

The PIT timer is initialized by setting the necessary control registers in the PIT module. This implementation was done in C so the registers were set through the provided functions. This can be seen through the code example below. The PIT Timer is enabled by writing to the PIT_MCR register. Then the value of the number of clock cycles in a period that creates a two second interrupt is calculated and loaded into the LDVAL register. Finally the timer is enabled, the interrupt is generated and the flag is cleared.

```
PIT->MCR = (0UL << 1);                      //Enable PIT timer
PIT->CHANNEL[0].LDVAL = 50000000;           //Load value for PIT timer for 2 second interrupts
PIT->CHANNEL[0].TCTRL =(3UL << 0);          //Turn on interrupt and timer enable
PIT->CHANNEL[0].TFLG = (1UL << 0);          //Clear flag
```

Timer Calculation

To fully utilize the PIT timer, the value to load for one second delay must be calculated. Since the M4 is inherently running at 50MHz, this gives us ~50 million instructions per second (Time = 1/Frequency).

Hence the LDVAL register for PIT0 was loaded with 50,000,000. To change the interrupt timer, we can simply change the loaded LDVAL register with whatever value we wish.

3.4.3. Lab Requirement 3

3-Digits Voltmeter Device Control

Each pin corresponds to a single LED for the 7-segment display. According to the figure below, each line for a full digit requires a connection (including the decimal point). If we were to label each LED location as pins. Pin 0 will be LED a, Pin 1 will be LED b, Pin 3 will be LED c, and son on with pins 8, 9, and 10 are used to power which digit position will be displayed. For this lab it can either display the left, middle or right digit. The 8 pins used for digit display are active low, therefore they are required to be 1 in the case they needed to be turned on. We used this property in our code, making sure that whatever number needed to be outputted had the required pins set (turns off / 1) or cleared (turns on / 0). In order to implement the placement of the LEDS to make digits we had variables for every hex number that could be displayed ranging from 1 to f.

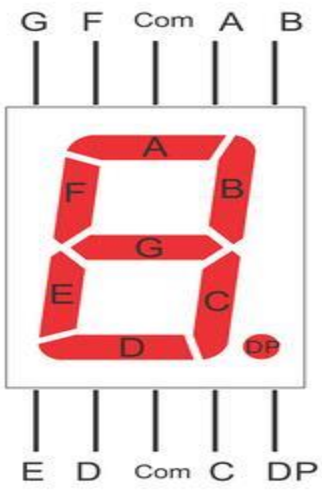


Figure 3.4.3. 7 Segment Display Labelled

Circuit Design

Using the schematic found in the lab manual we designed our 7-segment circuit accordingly. All connections run from the eight resistors without a transistor connected are used to power each individual LED on all digits (including the decimal point). For example, for an ‘8’ all the connections will be supplied with a current to light up certain LEDs to make a digital ‘8’. On the other hand, for a ‘1’ only two connections are supplied to create a digital ‘1’. The other three connections (with the transistors) are used to activate the individual digits. There are a total of three digits for the 7 segment display used and each transistor corresponds to which digit placement will be displayed (either left, middle, right). Voltage will be supplied to the corresponding transistor causing current to flow from collector to emitter which ends up lighting the LEDs required to make a digital number appear on the display. While one transistor is being supplied a voltage the others are required to be off or else the digits are displayed twice. This is because the circuit shares a common connection to display numbers, therefore it is required to turn the other transistors off.

4. **Conclusion**

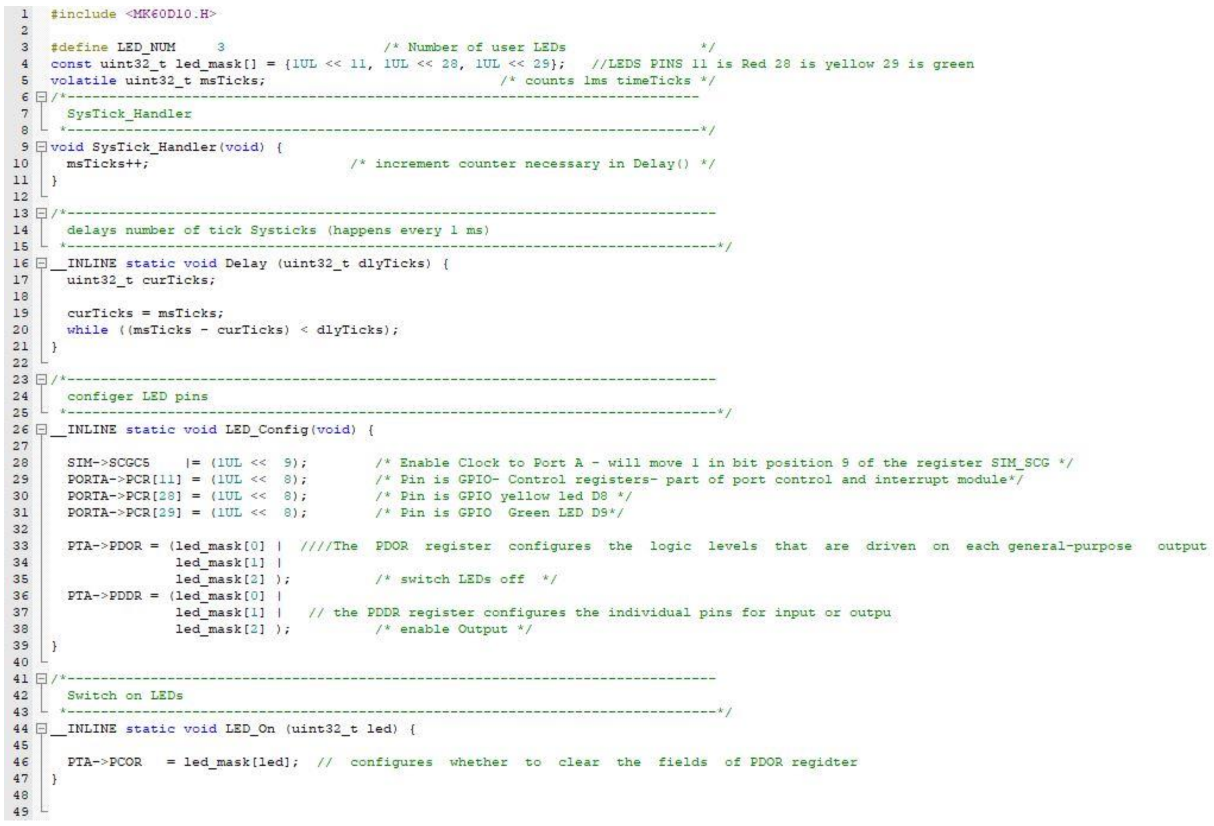
As a conclusion, the requirements of this lab were accomplished successfully. Requirement one was uploading and reviewing the code provided in the lab manual to control LEDs. Requirement two which is a GPIO application that blinks the LEDs on the board where the delay is controlled by using the Periodic Interrupt Timer (PIT) in this case it blinks every 2 seconds. As can be seen in the figures in appendix this and all other sections of the lab was written in C. As for the third requirement of this Lab, two parts were involved; part one was a software implementation where it was required to develop an application that runs a code in C and displays the results on a 7-segment LED> Four our implementation the results displayed on the 7 segment LED were hard coded. It was essential to make sure that NVIC, SIM (SCGCN, to configure the timer), PORTC (send low signals to the 7-segment display), PIT (to set timer values), PTA and PTC control registers are configured as they should be to allow for proper execution. After all the control registers had received the suitable values, the 7 segment display was loaded with a value and displayed the number. The hardware part was implemented following the instructions laid out in the lab manual. The two boards were always disconnected until the accuracy of our circuit was determined and no damages would occur.

References

[1] Radu Muresan, "ENGG3640: Microcomputer Interfacing Laboratory Manual, Version 2", University of Guelph, July 2016.

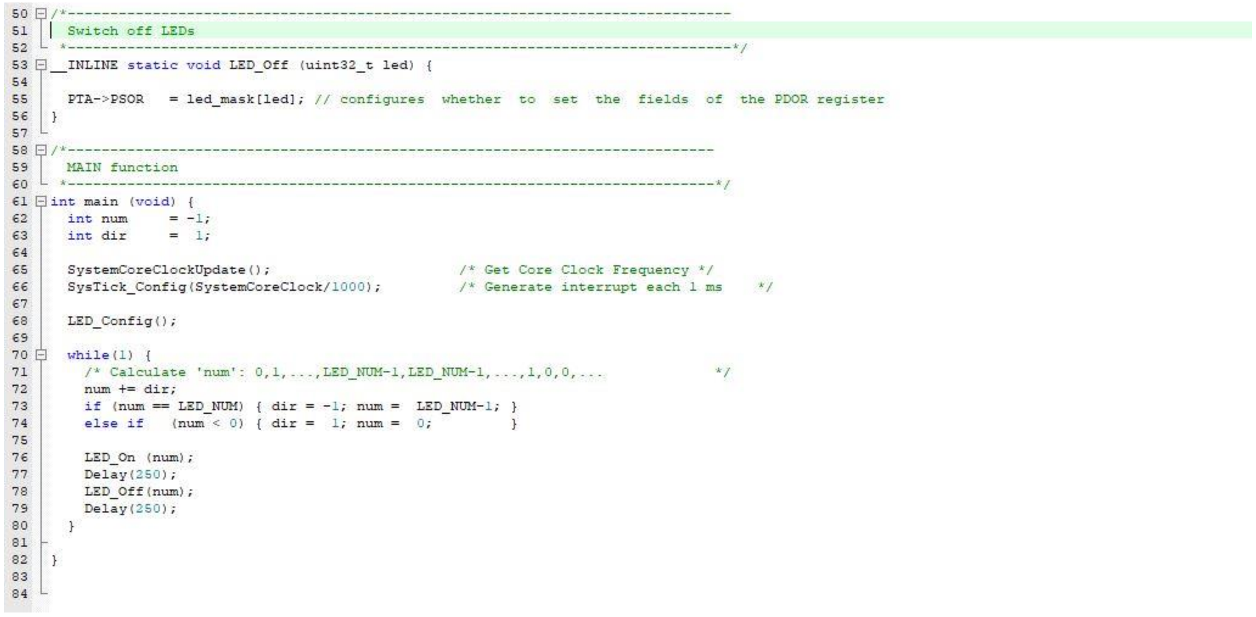
Appendices

A. Main.c



```c
#include <MK60D10.H>

#define LED_NUM     3                   /* Number of user LEDs                */
const uint32_t led_mask[] = {1UL << 11, 1UL << 28, 1UL << 29};   //LEDS PINS 11 is Red 28 is yellow 29 is green
volatile uint32_t msTicks;                            /* counts 1ms timeTicks */
/*----------------------------------------------------------------------------
  SysTick_Handler
 *----------------------------------------------------------------------------*/
void SysTick_Handler(void) {
  msTicks++;                        /* increment counter necessary in Delay() */
}

/*------------------------------------------------------------------------------
  delays number of tick Systicks (happens every 1 ms)
 *------------------------------------------------------------------------------*/
__INLINE static void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < dlyTicks);
}

/*------------------------------------------------------------------------------
  configer LED pins
 *------------------------------------------------------------------------------*/
__INLINE static void LED_Config(void) {

  SIM->SCGC5    |= (1UL <<  9);        /* Enable Clock to Port A - will move 1 in bit position 9 of the register SIM_SCG */
  PORTA->PCR[11] = (1UL <<  8);        /* Pin is GPIO- Control registers- part of port control and interrupt module*/
  PORTA->PCR[28] = (1UL <<  8);        /* Pin is GPIO yellow led D8 */
  PORTA->PCR[29] = (1UL <<  8);        /* Pin is GPIO  Green LED D9*/

  PTA->PDOR = (led_mask[0] |  ////The  PDOR  register  configures  the  logic  levels  that  are  driven  on  each general-purpose   output
               led_mask[1] |
               led_mask[2] );          /* switch LEDs off  */
  PTA->PDDR = (led_mask[0] |
               led_mask[1] |   // the PDDR register configures the individual pins for input or outpu
               led_mask[2] );          /* enable Output */
}

/*------------------------------------------------------------------------------
  Switch on LEDs
 *------------------------------------------------------------------------------*/
__INLINE static void LED_On (uint32_t led) {

  PTA->PCOR   = led_mask[led];  //  configures  whether  to  clear  the  fields  of PDOR regidter
}

```

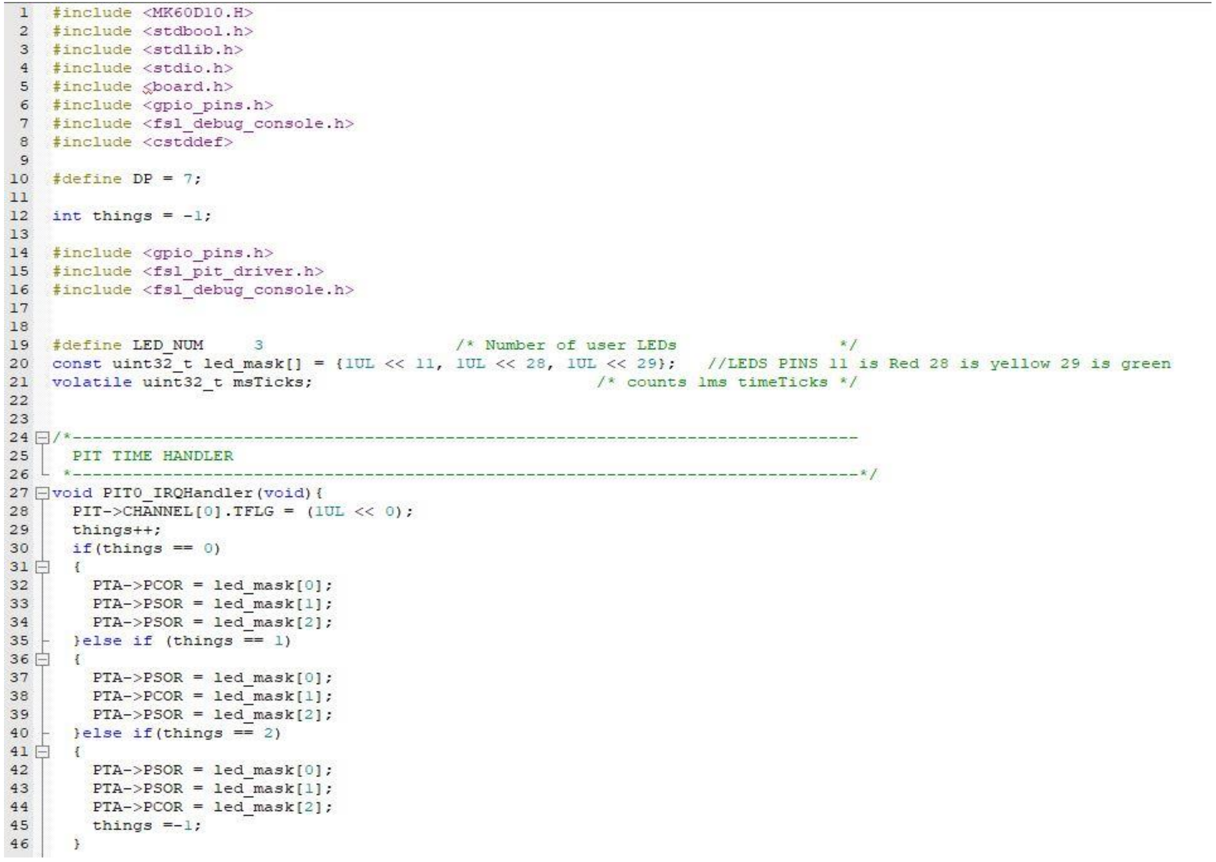
Figure A. 1 Main.c for Requirement 1



```c
/*----------------------------------------------------------------------------
  Switch off LEDs
 *----------------------------------------------------------------------------*/
__INLINE static void LED_Off (uint32_t led) {

  PTA->PSOR   = led_mask[led]; // configures  whether  to  set  the  fields  of  the PDOR register
}

/*----------------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------------*/
int main (void) {
  int num     = -1;
  int dir     =  1;

  SystemCoreClockUpdate();                      /* Get Core Clock Frequency */
  SysTick_Config(SystemCoreClock/1000);         /* Generate interrupt each 1 ms    */

  LED_Config();

  while(1) {
    /* Calculate 'num': 0,1,...,LED_NUM-1,LED_NUM-1,...,1,0,0,...             */
    num += dir;
    if (num == LED_NUM) { dir = -1; num =  LED_NUM-1; }
    else if   (num < 0) { dir =  1; num =  0;         }

    LED_On (num);
    Delay(250);
    LED_Off(num);
    Delay(250);
  }

}

```

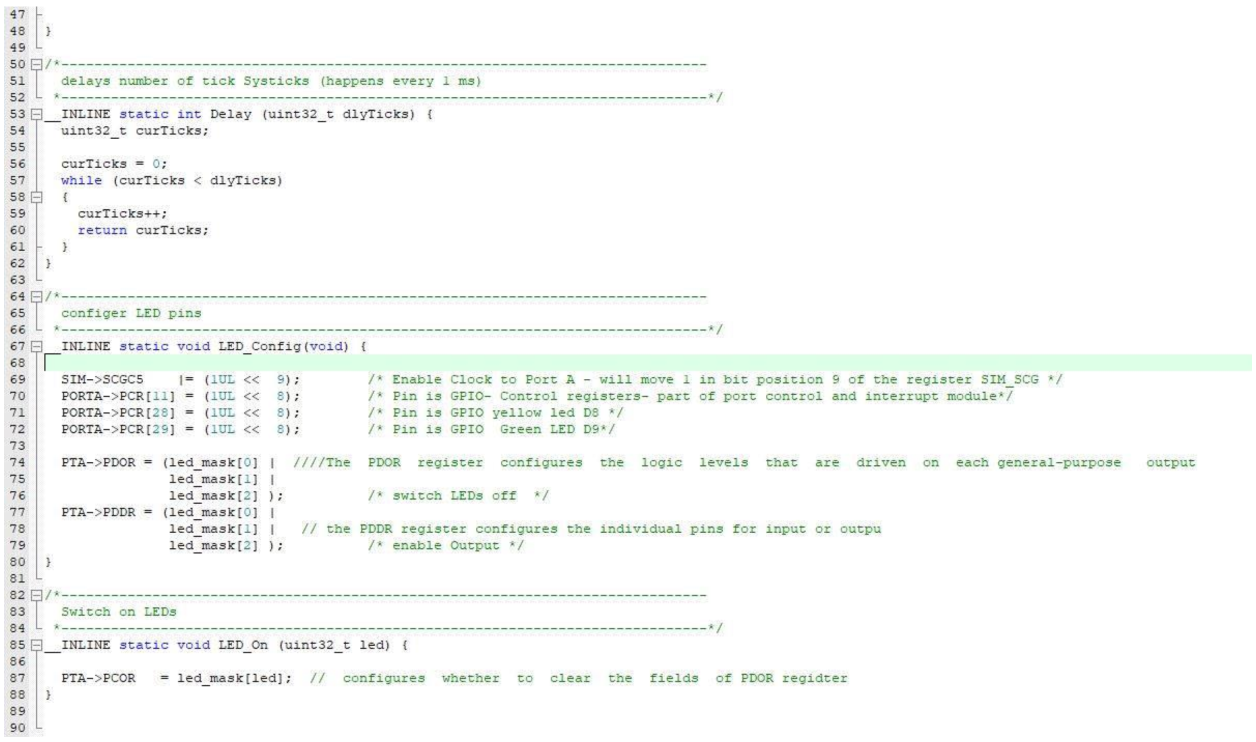
Figure A. 2 Main.c for Requirementt 1

B. Main.c



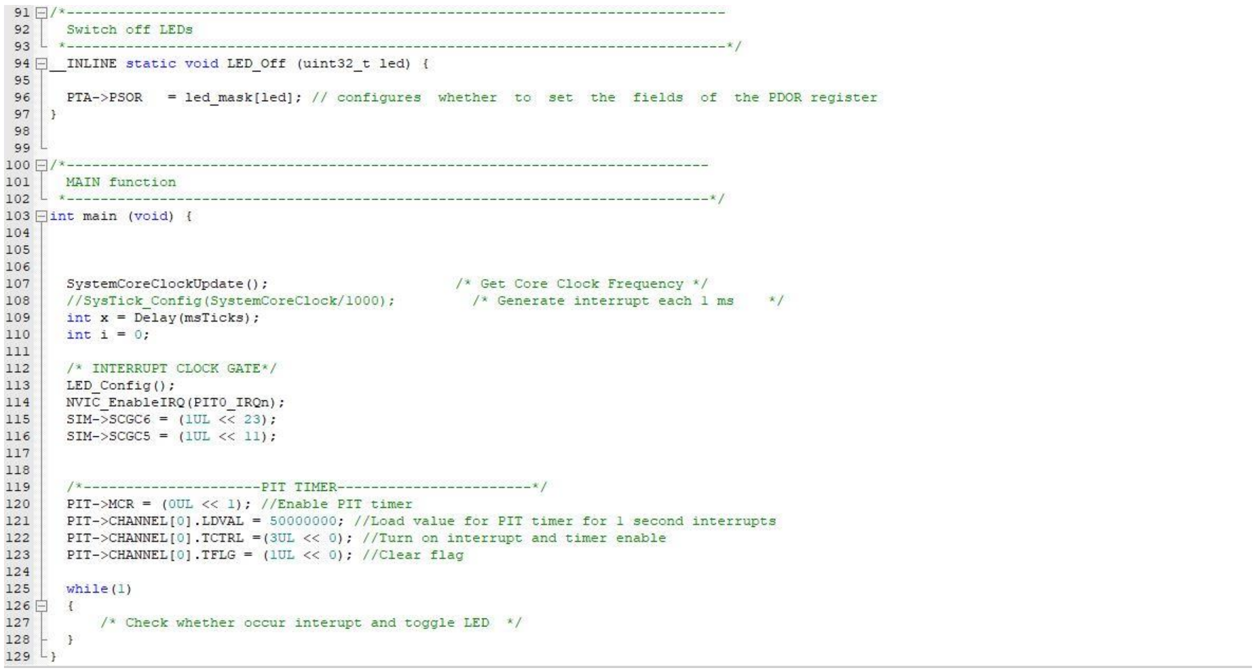
```c
#include <MK60D10.H>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <board.h>
#include <gpio_pins.h>
#include <fsl_debug_console.h>
#include <cstddef>

#define DP = 7;

int things = -1;

#include <gpio_pins.h>
#include <fsl_pit_driver.h>
#include <fsl_debug_console.h>


#define LED_NUM     3                     /* Number of user LEDs                */
const uint32_t led_mask[] = {1UL << 11, 1UL << 28, 1UL << 29};   //LEDS PINS 11 is Red 28 is yellow 29 is green
volatile uint32_t msTicks;                              /* counts 1ms timeTicks */


/*----------------------------------------------------------------------------
  PIT TIME HANDLER
 *----------------------------------------------------------------------------*/
void PIT0_IRQHandler(void){
  PIT->CHANNEL[0].TFLG = (1UL << 0);
  things++;
  if(things == 0)
  {
    PTA->PCOR = led_mask[0];
    PTA->PSOR = led_mask[1];
    PTA->PSOR = led_mask[2];
  }else if (things == 1)
  {
    PTA->PSOR = led_mask[0];
    PTA->PCOR = led_mask[1];
    PTA->PSOR = led_mask[2];
  }else if(things == 2)
  {
    PTA->PSOR = led_mask[0];
    PTA->PSOR = led_mask[1];
    PTA->PCOR = led_mask[2];
    things =-1;
  }
```

Figure B. 1. Main.c for Requirement 2



```

}

/*------------------------------------------------------------------------------
  delays number of tick Systicks (happens every 1 ms)
 *------------------------------------------------------------------------------*/
__INLINE static int Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = 0;
  while (curTicks < dlyTicks)
  {
    curTicks++;
    return curTicks;
  }
}

/*------------------------------------------------------------------------------
  configer LED pins
 *------------------------------------------------------------------------------*/
__INLINE static void LED_Config(void) {

  SIM->SCGC5    |= (1UL <<  9);        /* Enable Clock to Port A - will move 1 in bit position 9 of the register SIM_SCG */
  PORTA->PCR[11] = (1UL <<  8);        /* Pin is GPIO- Control registers- part of port control and interrupt module*/
  PORTA->PCR[28] = (1UL <<  8);        /* Pin is GPIO yellow led D8 */
  PORTA->PCR[29] = (1UL <<  8);        /* Pin is GPIO  Green LED D9*/

  PTA->PDOR = (led_mask[0] |  ////The  PDOR  register  configures  the  logic  levels  that  are  driven  on  each general-purpose   output
               led_mask[1] |
               led_mask[2] );          /* switch LEDs off  */
  PTA->PDDR = (led_mask[0] |
               led_mask[1] |   // the PDDR register configures the individual pins for input or outpu
               led_mask[2] );          /* enable Output */
}

/*------------------------------------------------------------------------------
  Switch on LEDs
 *------------------------------------------------------------------------------*/
__INLINE static void LED_On (uint32_t led) {

  PTA->PCOR   = led_mask[led];  //  configures  whether  to  clear  the  fields  of PDOR regidter
}


```

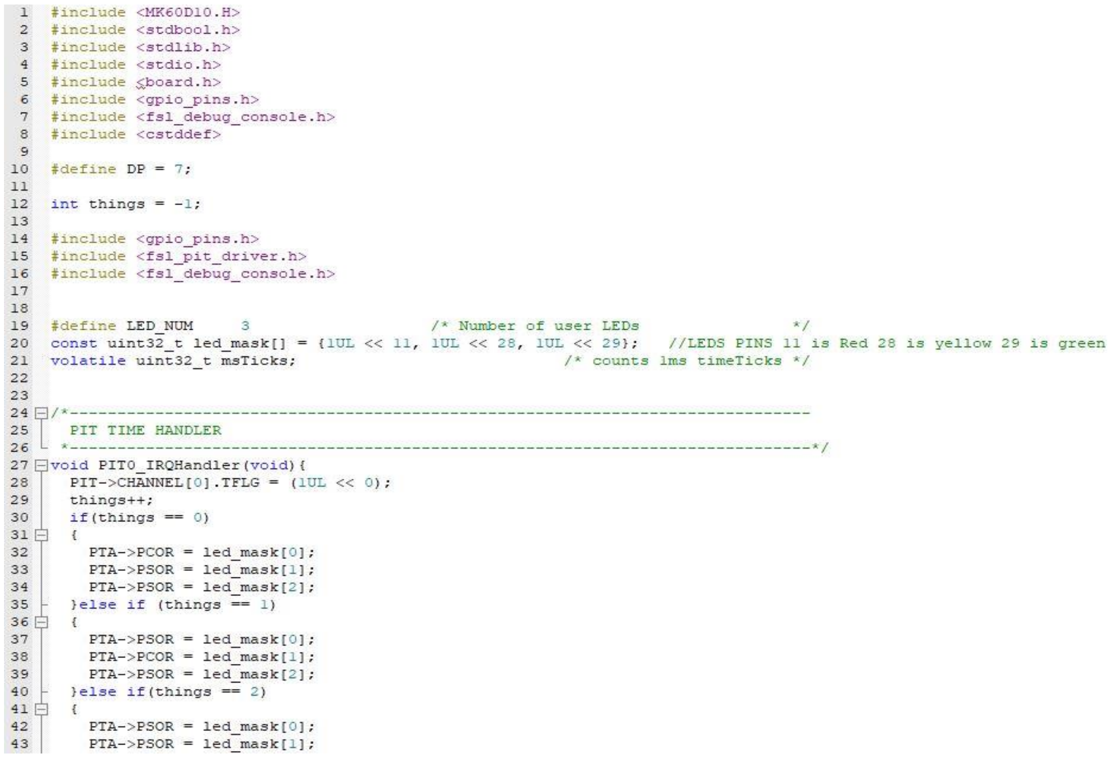
Figure B. 2. Main.c for Requirement 2



```
/*------------------------------------------------------------------------------
  Switch off LEDs
 *------------------------------------------------------------------------------*/
__INLINE static void LED_Off (uint32_t led) {

  PTA->PSOR   = led_mask[led]; // configures  whether  to  set  the  fields  of  the PDOR register
}


/*------------------------------------------------------------------------------
  MAIN function
 *------------------------------------------------------------------------------*/
int main (void) {



  SystemCoreClockUpdate();                      /* Get Core Clock Frequency */
  //SysTick_Config(SystemCoreClock/1000);         /* Generate interrupt each 1 ms    */
  int x = Delay(msTicks);
  int i = 0;

  /* INTERRUPT CLOCK GATE*/
  LED_Config();
  NVIC_EnableIRQ(PIT0_IRQn);
  SIM->SCGC6 = (1UL << 23);
  SIM->SCGC5 = (1UL << 11);


  /*---------------------PIT TIMER-----------------------*/
  PIT->MCR = (0UL << 1); //Enable PIT timer
  PIT->CHANNEL[0].LDVAL = 50000000; //Load value for PIT timer for 1 second interrupts
  PIT->CHANNEL[0].TCTRL =(3UL << 0); //Turn on interrupt and timer enable
  PIT->CHANNEL[0].TFLG = (1UL << 0); //Clear flag

  while(1)
  {
      /* Check whether occur interupt and toggle LED  */
  }
}
```

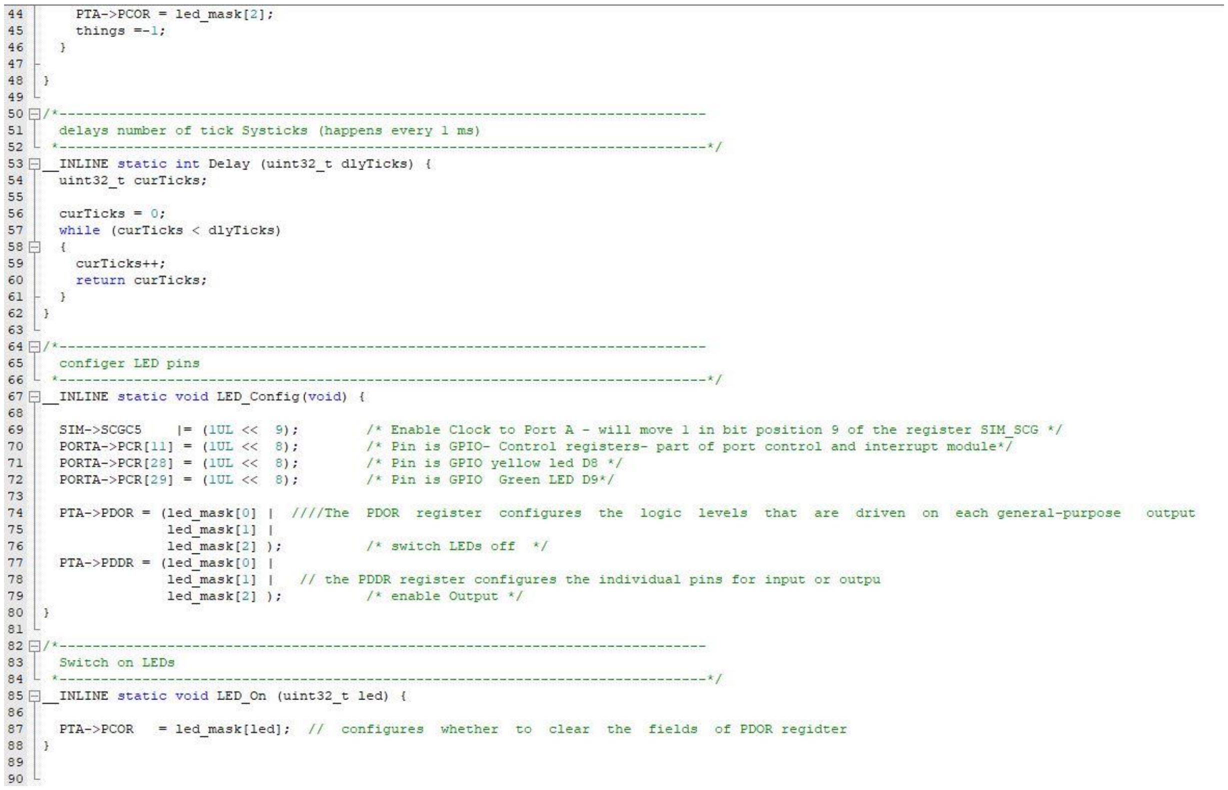
Figure B. 3. Main.c for Requirement 2

C. Main.c



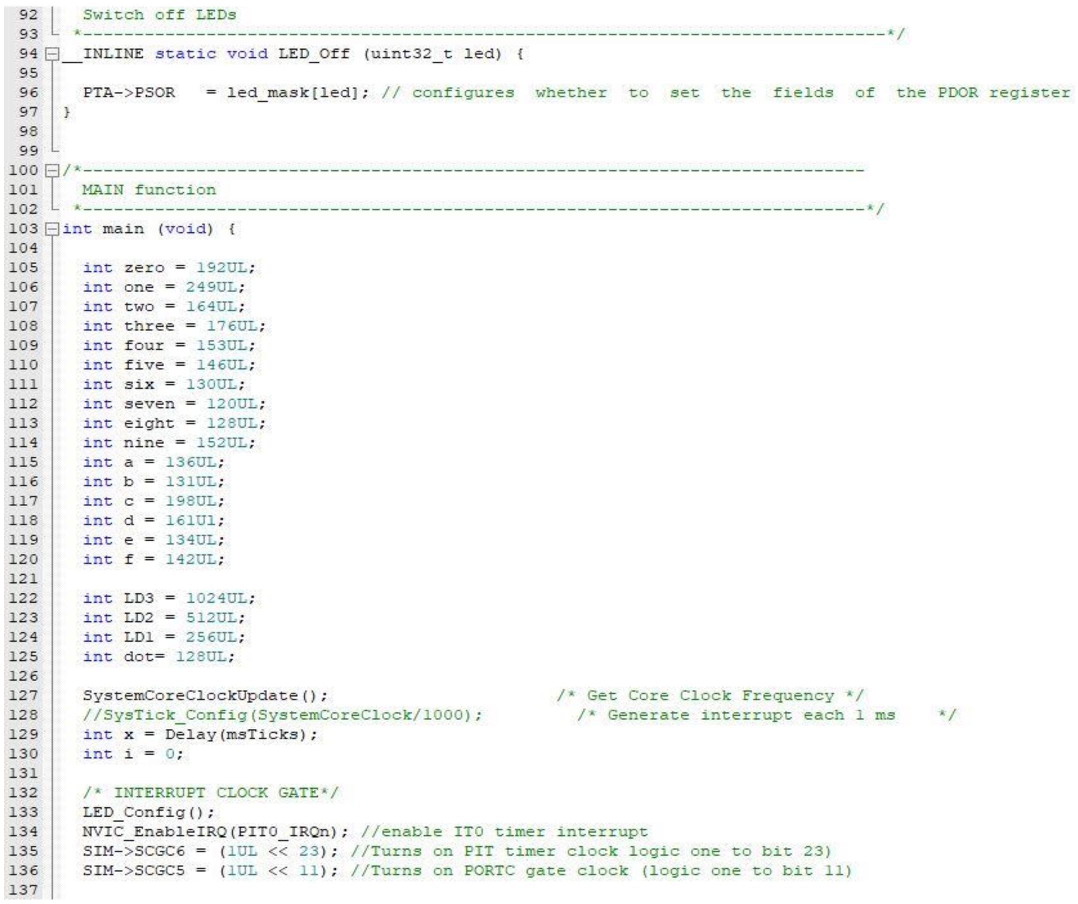
```c
#include <MK60D10.H>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <board.h>
#include <gpio_pins.h>
#include <fsl_debug_console.h>
#include <cstddef>

#define DP = 7;

int things = -1;

#include <gpio_pins.h>
#include <fsl_pit_driver.h>
#include <fsl_debug_console.h>


#define LED_NUM     3                    /* Number of user LEDs                */
const uint32_t led_mask[] = {1UL << 11, 1UL << 28, 1UL << 29};    //LEDS PINS 11 is Red 28 is yellow 29 is green
volatile uint32_t msTicks;                             /* counts 1ms timeTicks */


/*----------------------------------------------------------------------------------
  PIT TIME HANDLER
 *----------------------------------------------------------------------------------*/
void PIT0_IRQHandler(void){
  PIT->CHANNEL[0].TFLG = (1UL << 0);
  things++;
  if(things == 0)
  {
    PTA->PCOR = led_mask[0];
    PTA->PSOR = led_mask[1];
    PTA->PSOR = led_mask[2];
  }else if (things == 1)
  {
    PTA->PSOR = led_mask[0];
    PTA->PCOR = led_mask[1];
    PTA->PSOR = led_mask[2];
  }else if(things == 2)
  {
    PTA->PSOR = led_mask[0];
    PTA->PSOR = led_mask[1];
```

Figure C. 1. Main.c for Requirement 3



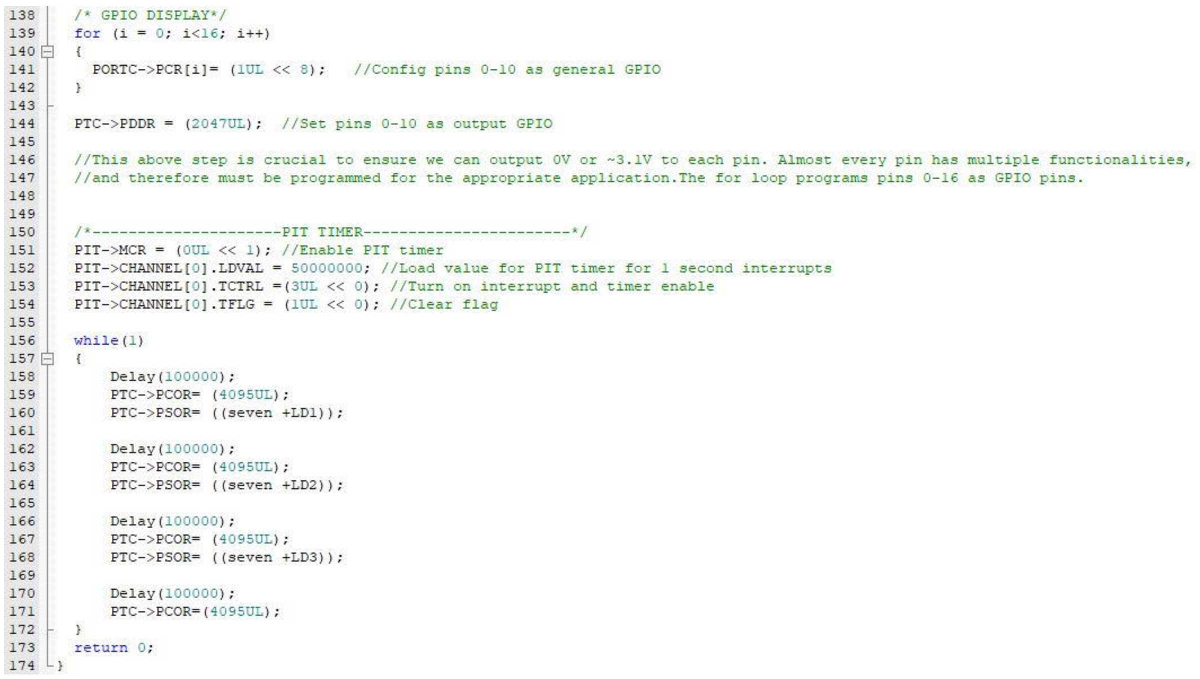
```c
    PTA->PCOR = led_mask[2];
    things =-1;
  }

}

/*----------------------------------------------------------------------------------
  delays number of tick Systicks (happens every 1 ms)
 *----------------------------------------------------------------------------------*/
__INLINE static int Delay (uint32_t dlyTicks) {
  uint32_t curTicks;

  curTicks = 0;
  while (curTicks < dlyTicks)
  {
    curTicks++;
    return curTicks;
  }
}

/*----------------------------------------------------------------------------------
  configer LED pins
 *----------------------------------------------------------------------------------*/
__INLINE static void LED_Config(void) {

  SIM->SCGC5    |= (1UL <<  9);        /* Enable Clock to Port A - will move 1 in bit position 9 of the register SIM_SCG */
  PORTA->PCR[11] = (1UL <<  8);        /* Pin is GPIO- Control registers- part of port control and interrupt module*/
  PORTA->PCR[28] = (1UL <<  8);        /* Pin is GPIO yellow led D8 */
  PORTA->PCR[29] = (1UL <<  8);        /* Pin is GPIO  Green LED D9*/

  PTA->PDOR = (led_mask[0] |  ////The  PDOR  register  configures  the  logic  levels  that  are  driven  on  each general-purpose   output
               led_mask[1] |
               led_mask[2] );          /* switch LEDs off  */
  PTA->PDDR = (led_mask[0] |
               led_mask[1] |   // the PDDR register configures the individual pins for input or outpu
               led_mask[2] );          /* enable Output */
}

/*----------------------------------------------------------------------------------
  Switch on LEDs
 *----------------------------------------------------------------------------------*/
__INLINE static void LED_On (uint32_t led) {

  PTA->PCOR   = led_mask[led];  //  configures  whether  to  clear  the  fields  of PDOR regidter
}

```

Figure C. 2. Main.c for Requirement 3



```
 Switch off LEDs
 *----------------------------------------------------------------------------*/
__INLINE static void LED_Off (uint32_t led) {

  PTA->PSOR   = led_mask[led]; // configures  whether  to  set  the  fields  of  the PDOR register
}


/*----------------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------------*/
int main (void) {

  int zero = 192UL;
  int one = 249UL;
  int two = 164UL;
  int three = 176UL;
  int four = 153UL;
  int five = 146UL;
  int six = 130UL;
  int seven = 120UL;
  int eight = 128UL;
  int nine = 152UL;
  int a = 136UL;
  int b = 131UL;
  int c = 198UL;
  int d = 161Ul;
  int e = 134UL;
  int f = 142UL;

  int LD3 = 1024UL;
  int LD2 = 512UL;
  int LD1 = 256UL;
  int dot= 128UL;

  SystemCoreClockUpdate();                       /* Get Core Clock Frequency */
  //SysTick_Config(SystemCoreClock/1000);          /* Generate interrupt each 1 ms    */
  int x = Delay(msTicks);
  int i = 0;

  /* INTERRUPT CLOCK GATE*/
  LED_Config();
  NVIC_EnableIRQ(PIT0_IRQn); //enable IT0 timer interrupt
  SIM->SCGC6 = (1UL << 23); //Turns on PIT timer clock logic one to bit 23)
  SIM->SCGC5 = (1UL << 11); //Turns on PORTC gate clock (logic one to bit 11)

```

Figure C. 3. Main.c for Requirement 3



```
  /* GPIO DISPLAY*/
  for (i = 0; i<16; i++)
  {
    PORTC->PCR[i]= (1UL << 8);   //Config pins 0-10 as general GPIO
  }

  PTC->PDDR = (2047UL);  //Set pins 0-10 as output GPIO

  //This above step is crucial to ensure we can output 0V or ~3.1V to each pin. Almost every pin has multiple functionalities,
  //and therefore must be programmed for the appropriate application.The for loop programs pins 0-16 as GPIO pins.


  /*---------------------PIT TIMER-----------------------*/
  PIT->MCR = (0UL << 1); //Enable PIT timer
  PIT->CHANNEL[0].LDVAL = 50000000; //Load value for PIT timer for 1 second interrupts
  PIT->CHANNEL[0].TCTRL =(3UL << 0); //Turn on interrupt and timer enable
  PIT->CHANNEL[0].TFLG = (1UL << 0); //Clear flag

  while(1)
  {
      Delay(100000);
      PTC->PCOR= (4095UL);
      PTC->PSOR= ((seven +LD1));

      Delay(100000);
      PTC->PCOR= (4095UL);
      PTC->PSOR= ((seven +LD2));

      Delay(100000);
      PTC->PCOR= (4095UL);
      PTC->PSOR= ((seven +LD3));

      Delay(100000);
      PTC->PCOR=(4095UL);
  }
  return 0;
}
```

Figure C. 4. Main.c for Requirement 3