Faculty of Engineering & Technology Electrical & Computer Engineering Department

COMPUTER DESIGN LABORATORY

ENCS4110

Report #2
"Timers Interrupts"

Prepared by:
Student Name: Mohammad Salem
Student ID: 1200651

Instructor: Dr. Abualseoud Hanani
Assistant: Tareq Zidan

Section: 1
Date: 3/5/2024

1. Abstract

Utilizing the TM4C123 microcontroller, this experiment exposes students to how to set up of timers and interrupts. Students will also be prepared to build up a nested vector interrupt controller (NVIC) and work with internal timers and counters. Additionally, to attempt to program certain activities to be paused when a specific action is taken. And to test the codes in experiment in specific piece of hardware so they can be understood. After worked to clarify these steps: Calculate the desired delay, choose a prescaler value, Calculate the scaled-down frequency, Calculate the time period, Load the prescaler register, Load the starting count value, Enable TimerA module, Enable TimerA timeout interrupt , Enable Timer1A interrupt in NVIC, Implement Timer1A interrupt handler.

Table of Contents

Table of Figures

2. Theory

2.1: General Purpose Timer Interrupt

The TM4C123 microcontroller's programmable general-purpose timer modules (GPTM) offer versatile functionality, allowing them to function as counters or timers to track external events. One practical application is measuring an analog signal using the TM4C123's ADC at regular one-second intervals. This desired functionality can be easily achieved through the use of GPTM [2].

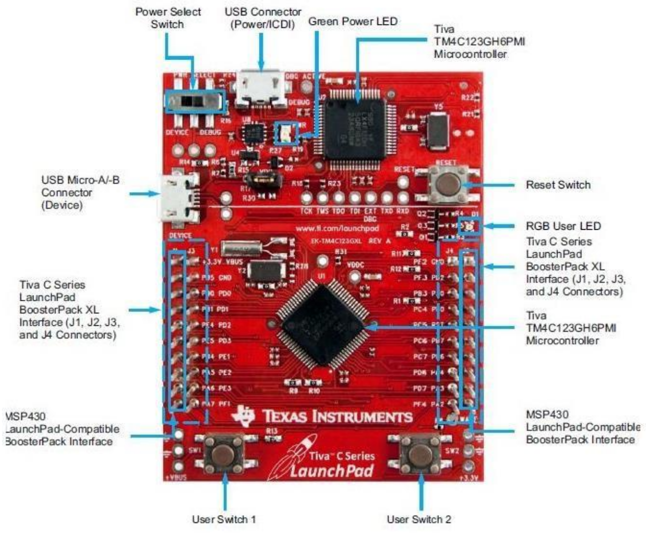

Power Select Switch
USB Connector (Power/ICDI)
Green Power LED
Tiva TM4C123GH6PMI Microcontroller
USB Micro-A/-B Connector (Device)
Reset Switch
RGB User LED
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva C Series LaunchPad BoosterPack XL Interface (J1, J2, J3, and J4 Connectors)
Tiva TM4C123GH6PMI Microcontroller
MSP430 LaunchPad-Compatible BoosterPack Interface
MSP430 LaunchPad-Compatible BoosterPack Interface
User Switch 1
User Switch 2

Figure 1:TM4C123 microcontrollers [1]

To implement this functionality, follow these steps:

.Configure the timer interrupt of the TM4C123 microcontroller to generate an interrupt signal every one second.

.Within the interrupt service routine (ISR) associated with the timer interrupt, sample the analog signal value using the ADC.

.Before exiting the ISR, remember to turn off ADC sampling.
In embedded systems, general-purpose timer modules find extensive application due to their flexibility and usefulness in various tasks.

Types of interrupts:
There are two main types of interrupts: hardware interrupts and software interrupts. Hardware interrupts are triggered by an interrupt request signal generated by peripheral circuits. Conversely, software interrupts are initiated by the execution of a specific instruction designed for this purpose [2].

2.2: Timer1A Interrupt with One Second Delay

We generate a one Hertz timer interrupt. That implies that the interruption will happen once per second. thus the frequency and time duration are inversely proportional. In the TimerA module's related interrupt service function, we flip an LED whenever an interrupt occurs. In essence, the TM4C123 Tiva C LaunchPad's onboard LED will be toggled every one second by the interrupt service procedure[2].

The RGB LED on the TM4C123 Tiva C LaunchPad is built-in. The PF3 pin of the PORTF is linked to the blue LED. Within TimerA's interrupt service function, we shall turn this LED on and off. Let's first use the #define preprocessor command to define the symbol name for this PF3 pin.

2.3: Initialize Timer1A registers for one second delay

First, enable the clock to timer block 1 using RCGTIMER register. Setting 1st bit of RCGTIMER register enables the clock to 16/32 bit general purpose timer module 1 in run mode. This line sets the bit1 of the RCGTIMER register to 1 [2].
SYSCTL->RCGCTIMER |= (1<<1); /* disable timer1 output */

The first three bits of the GPTMCFG register are used to select the mode of operation of timer blocks either in concatenated mode (32 or 64 bits mode) or individual or split timers (16 or 32 bit mode). We will be using a 16/32-bit timer in 16-bits configuration mode. Hence, writing 0x4 to this register selects the 16-bit configuration mode.
TIMER1->CFG = 0x4; /*select 16-bit configuration option, CFG=0x00 for 32-bit option */

Timer modules can be used in three modes such as one-shot, periodic, and capture mode. We want the timer interrupt to occur periodically after a specific time. Therefore, we will use the periodic mode. In order to select the periodic mode of timer1, set the GPTMTAMR register to 0x02 like this:
TIMER1->TMAR = 0x02; /*select periodic down counter mode of timer1, TMAR=0x01 for one shot mode */

We are using timer1 in 16-bit configuration. The maximum delay a 16-bit timer can generate according to 16MHz operating frequency of TM4C123 microcontroller is given by this equation:

2^16 = 65536
16MHz = 16000000
Maximum delay = 65536/16000000 = 4.096 millisecond

Hence, the maximum delay that we can generate with timer1 in 16-bit configuration is 4.096 millisecond. Now the question is how to increase delay size? There are two ways to increase the timer delay size: either increase the size of timer (32-bit or 64-bit) or use a prescaler value. Because we have already selected the 16-bit timer1A configuration. Therefore, we can use the prescaler option.

2.4: Prescaler Configuration

Prescaler adds additional bits to the size of the timer. GPTM TimerA Prescale (GPTMTAPR) register is used to add the required Prescaler value to the timer. TimerA in 16-bit has an 8-bit Prescaler value. Prescaler basically scales down the frequency to the timer module. Hence, an 8- bit Prescaler can reduce the frequency (16MHz) by 1-255 [2].

For example, we want to generate a one-second delay, using a Prescaler value of 250 scales down the operating frequency for TimerA block to 64000Hz .i.e. 16000000/250 = 64000Hz = 64KHz. Hence, the time period for TimerA is 1/64000 = 15.625ms. Hence, load the Prescaler register with a value of 250.
TIMER1->TAPR = 250-1; /* TimerA prescaler value */

When the timer is used in periodic countdown mode, the GPTM TimerA Interval Load (GPTMTAILR) register is used to load the starting value of the counter to the timer.
TIMER1->TAILR = 64000 - 1 /* TimerA counter starting count down value */

GPTMICR register is used to clear the timeout flag bit of timer.
TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/

After initialization and configuration, enables the Timera module, which we disabled earlier.
TIMER1->CTL |= (1<<0); /* Enable TimerA module */

2.5: Types of Interrupt and Exceptions in ARM Cortex-M

Throughout this experiment, we will use exception and interrupt terms interchangeably. Because, in ARM Cortex-M literature both terms are used to refer to interrupts and exceptions. Although there is a minor difference between interrupt and exception. Interrupts are special types of exceptions which are caused by peripherals or external interrupts,such as, Timers, GPIO, UART, I2C, etc. On the contrary, exceptions are generated by processor or system. For example, In ARM Cortex-M4, the exceptions numbered from 0-15 are known as system exceptions and the peripheral interrupts can be between 1 to 240. However, the available number of peripheral interrupts can be different for different ARM chip manufacturers. For instance, ARM Cortex-M4 based TM4C123 microcontroller supports 16 system exceptions and 78 peripheral interrupts [2].

2.6: Interrupt Vector Table and Interrupt Processing

However, the question is how does the processor find the addresses of these interrupt service routines or exception handlers? In order to understand this, let us take a review of the memory map of ARM cortex M4 microcontrollers. The address space that ARM MCU supports is 4GB. This memory map is divided into different memory sections, such as, code, RAM, peripheral, external memory, system memory region, as shown in the figure below:

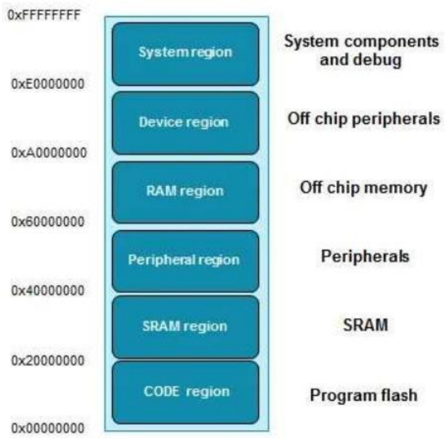


Figure 2: Interrupt Vector Table [2]

2.7: Relocating ISR Address from IVT

Coming back to the main discussion, the code memory region contains an interrupt vector table (IVT). This interrupt vector table consists of a reset address of stack pointer and starting addresses of all exception handlers. In other words, it contains the starting address that points to the respective interrupt and exception routines, such as, reset handler. Hence, the microcontroller uses this starting address to find the code of the interrupt service routine that needs to be executed in response to a particular interrupt [2].

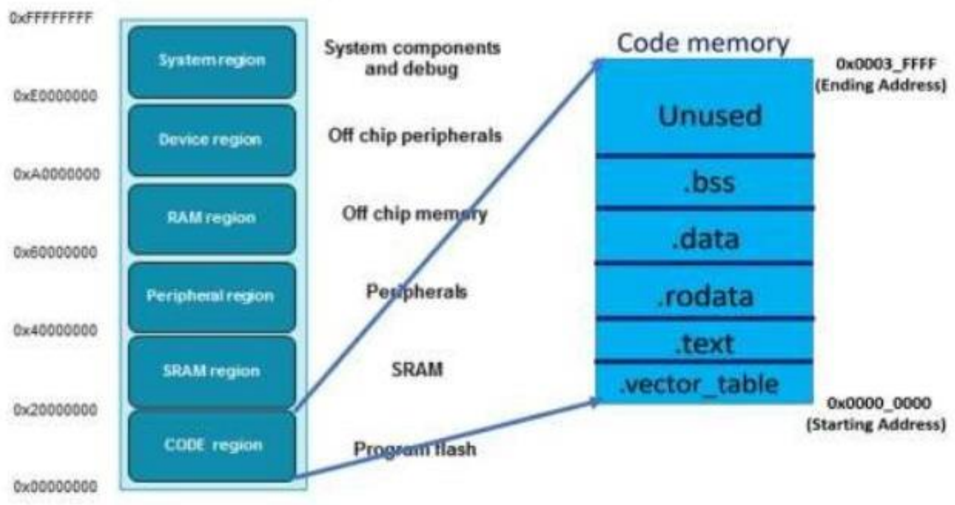
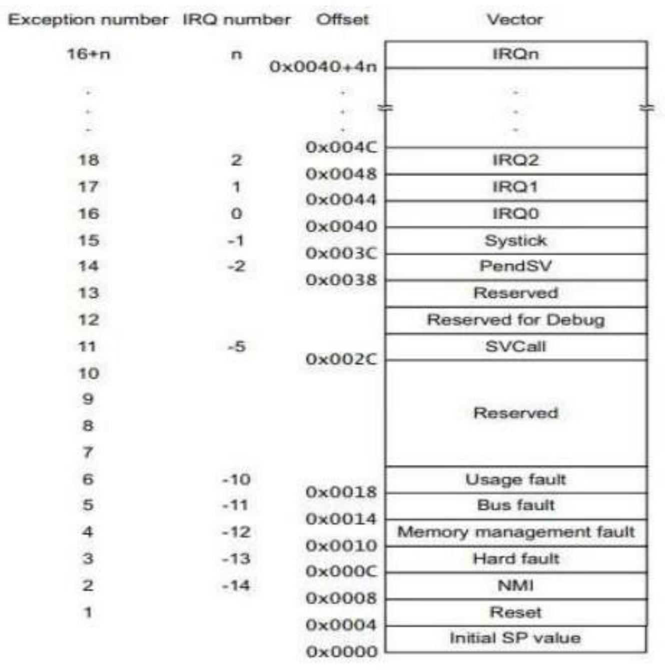


Figure 3: Relocating ISR Address from IVT [2]

the interrupt vector table is an array of memory addresses. It contains the starting address of all exception handlers. Furthermore, each interrupt/exception also has a unique interrupt number assigned to it. The microcontroller uses interrupt number to find the entry inside the interrupt vector table. The following table shows the interrupt vector table of ARM Cortex M4 based TM4C123GH6PM microcontroller.



Exception number	IRQ number	Offset	Vector
16+n	n	0x0040+4n	IRQn
.		.	.
.		.	.
.		.	.
18	2	0x004C	IRQ2
17	1	0x0048	IRQ1
16	0	0x0044	IRQ0
15	-1	0x0040	Systick
14	-2	0x003C	PendSV
13		0x0038	Reserved
12			Reserved for Debug
11	-5		SVCall
10		0x002C	Reserved
9			
8			
7			
6	-10		Usage fault
5	-11	0x0018	Bus fault
4	-12	0x0014	Memory management fault
3	-13	0x0010	Hard fault
2	-14	0x000C	NMI
1		0x0008	Reset
		0x0004	Initial SP value
		0x0000	

Figure 4: the interrupt vector table of ARM Cortex M4 based TM4C123GH6PM microcontroller [2]

2.8: Steps Executed by ARM Cortex M processor During Interrupt Processing

Until now, we have discussed how ARM microcontroller finds the address of the first instruction of the corresponding interrupt service routine. In this section, we will see the sequence of steps that occurs during interrupt processing, such as, context switching, context saving, registers stacking and unstacking.

Whenever an interrupt occurs, the context switch happens. That means the processor moves from thread mode to the handler mode. As shown in the figure below, the ARM Cortex-M microcontroller keeps executing the main application, but when an interrupt occurs, the processor switches to interrupt service routine. After executing ISR, it switches back to the main application again. Nevertheless, what happens during context switching requires more explanation [2].

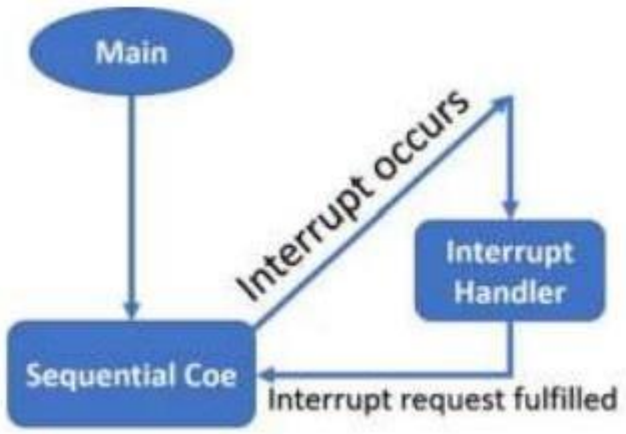


Figure 5: interrupt processing [2]

2.9: ARM Cortex-M Context Switching

When an interrupt occurs and its request is approved by the NVIC and processor, the contents of the CPU registers are saved onto the stack. This way of CPU registers preservation helps microcontrollers to resume the interrupted program form the same instruction where it is suspended due to an interrupt service routine. The process of saving the main application registers content onto the stack is known as context switching.

However, it takes time for the microcontroller to save the CPU register's content onto the stack. However, to Harvard architecture of ARM processors (separate instruction and data buses), the processor performs context saving and fetching of starting address of interrupt service routine in parallel [2].

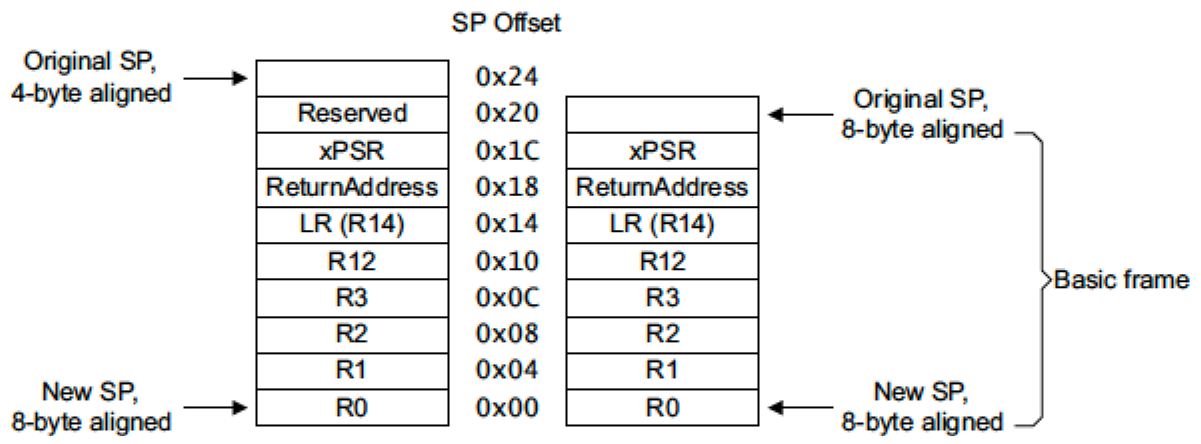


Figure 6: ARM Cortex-M Context Switching

3. Procedure

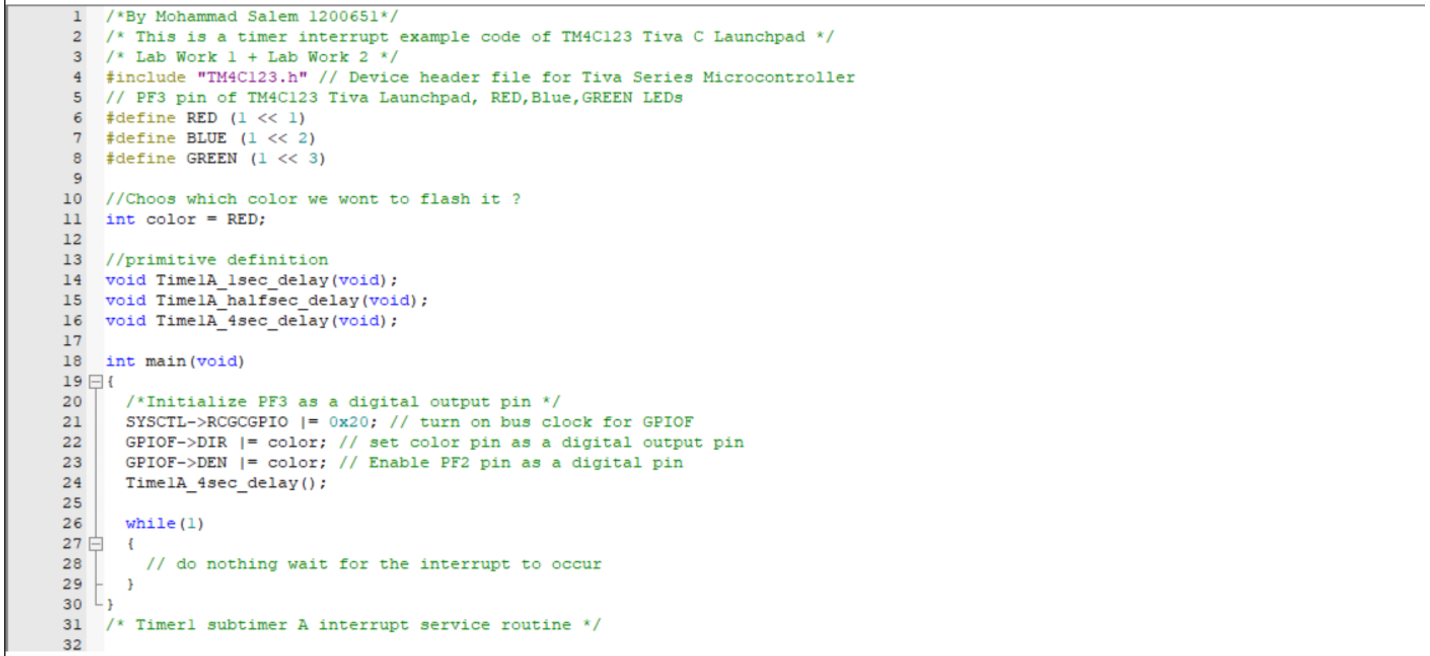
To begin, you can create a project in Keil uVision by following these steps:

1) Launch Keil uVision software on your computer.
2) From the main menu, go to "Project" and select "New µVision Project" (or press Ctrl + N).
3) Choose a suitable location to save your project and provide a name for it.
4) Select the TM4C123GH6PM microcontroller as the target device. You can do this by navigating to "Device" in the "Create New Project" window, expanding the "Texas Instruments" folder, and choosing the "TM4C123GH6PM" microcontroller.
5) Click on the "Save" button to create the project.
6) Once the project is created, you will be prompted to select a startup file. You can choose the appropriate startup file for your project or leave it as the default one.
7) Now, you have an empty project open in Keil uVision. You can add source files, configure settings, and write your code within this project.

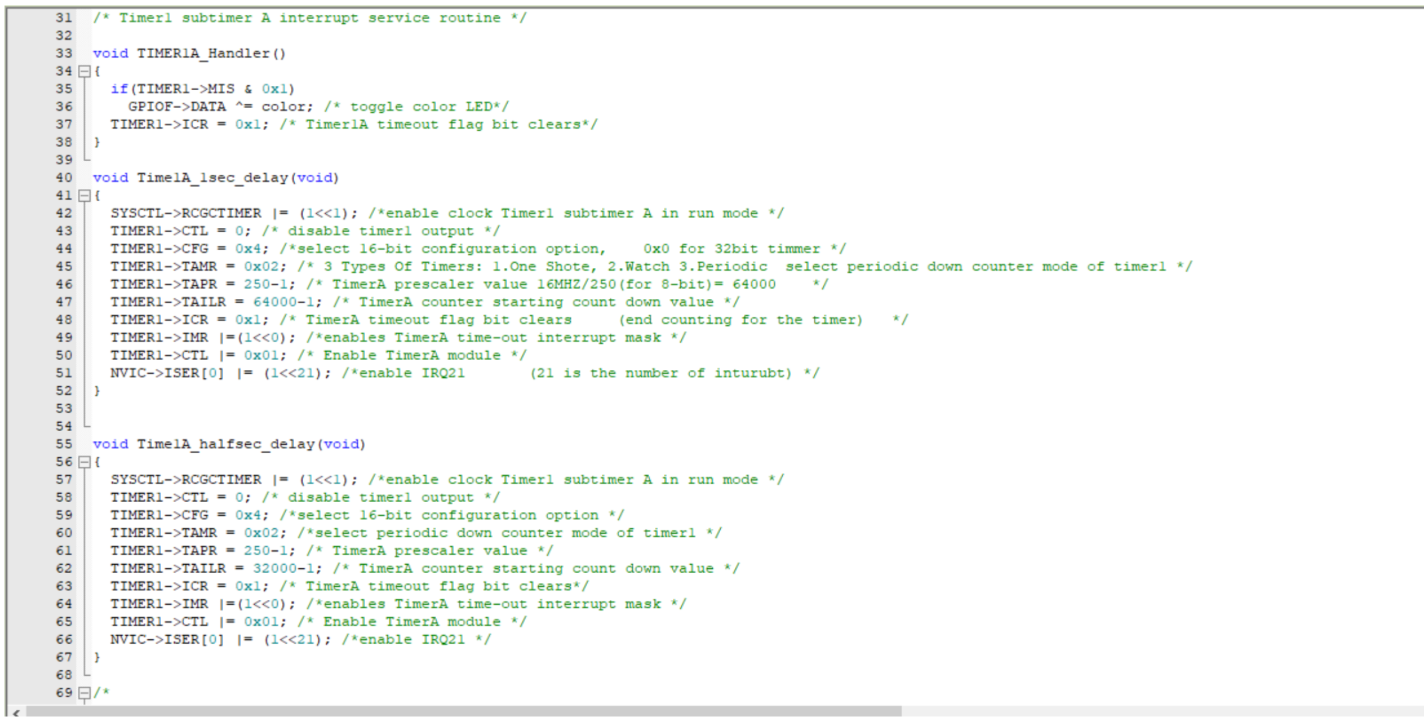
Remember to consult the Keil uVision documentation or specific tutorials for detailed instructions on working with the software and the TM4C123GH6PM microcontroller.

TM4C123 Timer Interrupt Example Code:

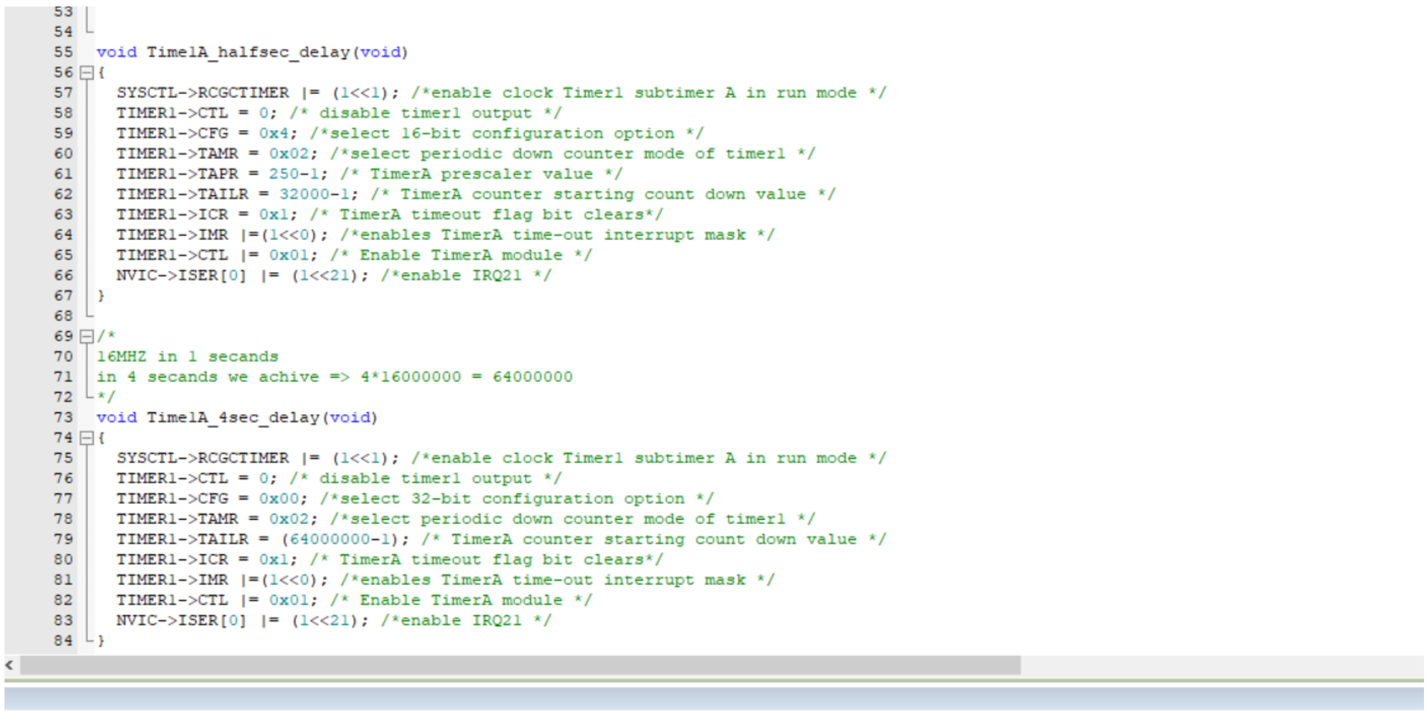
Using the Timer1A interrupt handler technique, this TM4C123 Tiva C Launchpad sample code generates a one-second delay, half second's delay, and 4 seconds delay in functions. However, all details for Lab Work 1 and Lab Work 2 as shown in Code and Comments bellow.



```c
/*By Mohammad Salem 1200651*/
/* This is a timer interrupt example code of TM4C123 Tiva C Launchpad */
/* Lab Work 1 + Lab Work 2 */
#include "TM4C123.h" // Device header file for Tiva Series Microcontroller
// PF3 pin of TM4C123 Tiva Launchpad, RED,Blue,GREEN LEDs
#define RED (1 << 1)
#define BLUE (1 << 2)
#define GREEN (1 << 3)

//Choos which color we wont to flash it ?
int color = RED;

//primitive definition
void Time1A_1sec_delay(void);
void Time1A_halfsec_delay(void);
void Time1A_4sec_delay(void);

int main(void)
{
  /*Initialize PF3 as a digital output pin */
  SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
  GPIOF->DIR |= color; // set color pin as a digital output pin
  GPIOF->DEN |= color; // Enable PF2 pin as a digital pin
  Time1A_4sec_delay();

  while(1)
  {
    // do nothing wait for the interrupt to occur
  }
}
/* Timer1 subtimer A interrupt service routine */

```



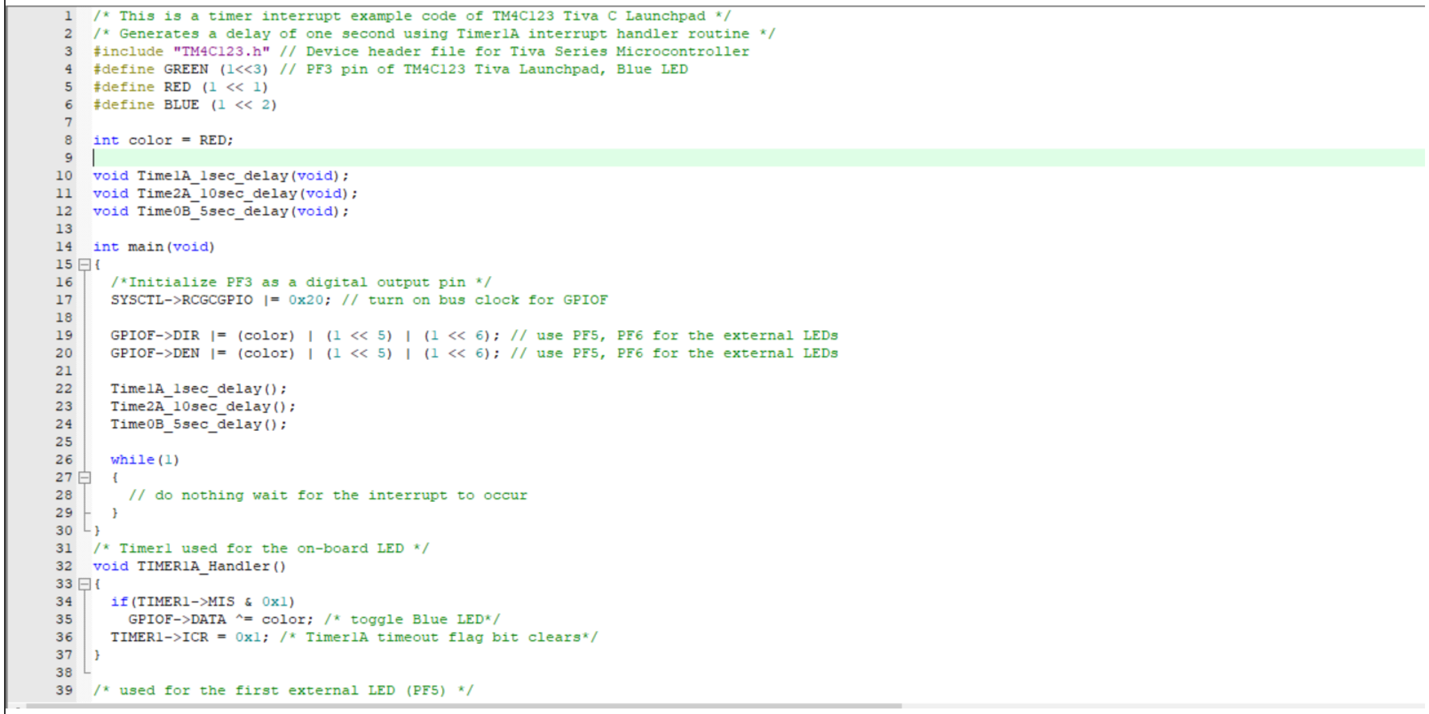
```c
/* Timer1 subtimer A interrupt service routine */

void TIMER1A_Handler()
{
  if(TIMER1->MIS & 0x1)
    GPIOF->DATA ^= color; /* toggle color LED*/
  TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears*/
}

void Time1A_1sec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x4; /*select 16-bit configuration option,    0x0 for 32bit timmer */
  TIMER1->TAMR = 0x02; /* 3 Types Of Timers: 1.One Shote, 2.Watch 3.Periodic  select periodic down counter mode of timer1 */
  TIMER1->TAPR = 250-1; /* TimerA prescaler value 16MHZ/250(for 8-bit)= 64000    */
  TIMER1->TAILR = 64000-1; /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears     (end counting for the timer)   */
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<21); /*enable IRQ21       (21 is the number of inturubt) */
}


void Time1A_halfsec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x4; /*select 16-bit configuration option */
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAPR = 250-1; /* TimerA prescaler value */
  TIMER1->TAILR = 32000-1; /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
}

/*
```



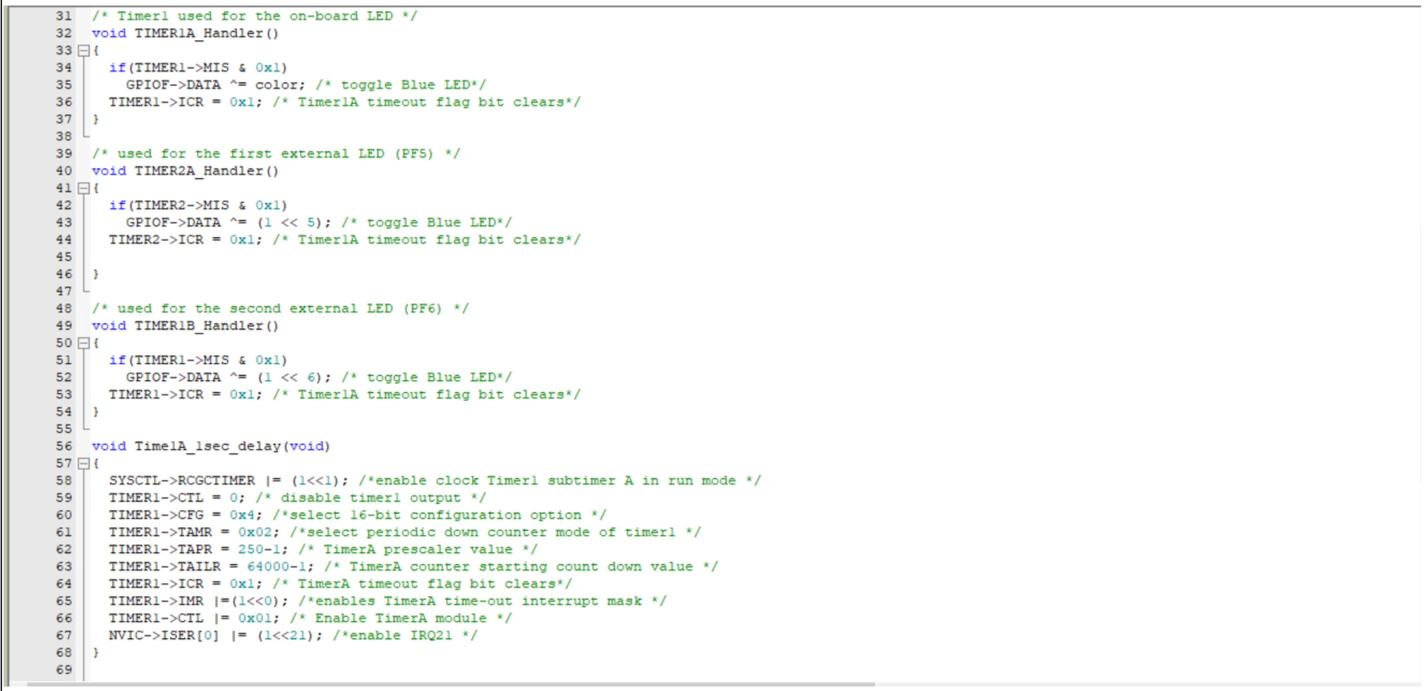
```c


void Time1A_halfsec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x4; /*select 16-bit configuration option */
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAPR = 250-1; /* TimerA prescaler value */
  TIMER1->TAILR = 32000-1; /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
}

```

```c
/*
16MHZ in 1 secands
in 4 secands we achive => 4*16000000 = 64000000
*/
void Time1A_4sec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x00; /*select 32-bit configuration option */
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAILR = (64000000-1); /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
}
```

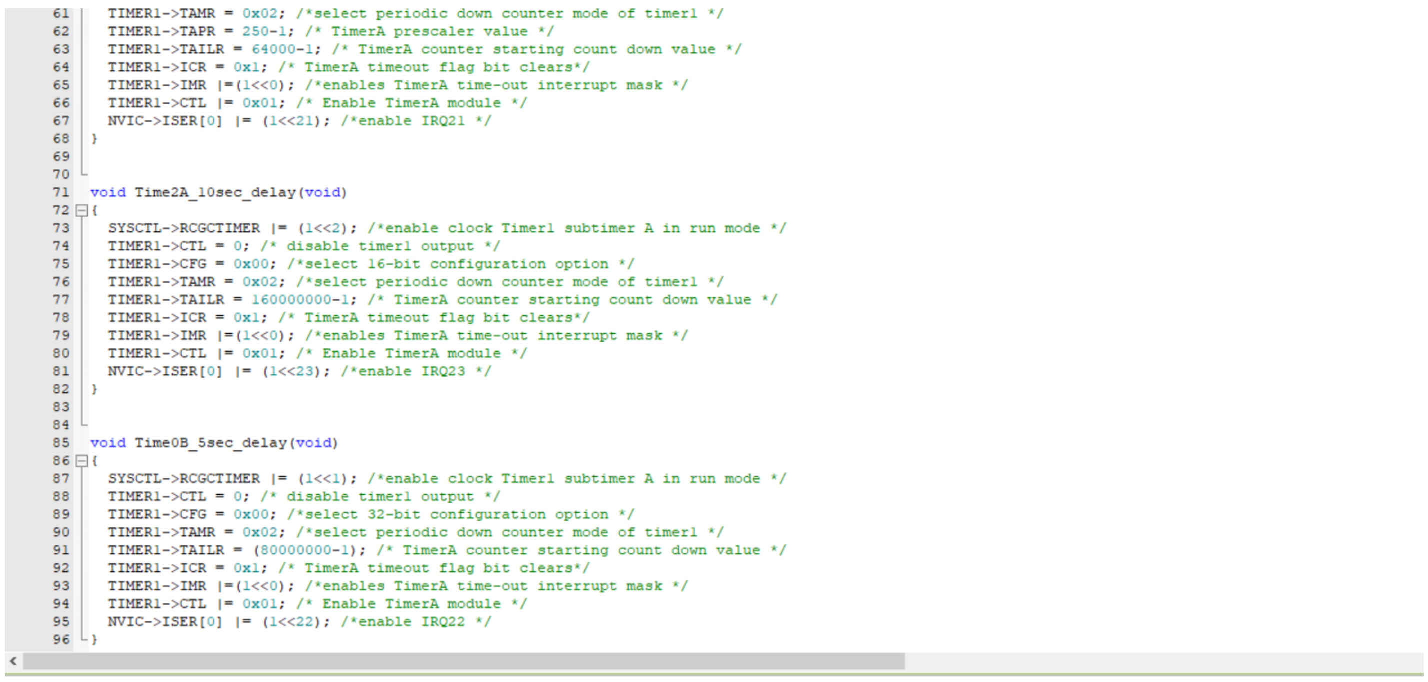
Lab Work 3:



```c
/* This is a timer interrupt example code of TM4C123 Tiva C Launchpad */
/* Generates a delay of one second using Timer1A interrupt handler routine */
#include "TM4C123.h" // Device header file for Tiva Series Microcontroller
#define GREEN (1<<3) // PF3 pin of TM4C123 Tiva Launchpad, Blue LED
#define RED (1 << 1)
#define BLUE (1 << 2)

int color = RED;

void Time1A_1sec_delay(void);
void Time2A_10sec_delay(void);
void Time0B_5sec_delay(void);

int main(void)
{
  /*Initialize PF3 as a digital output pin */
  SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF

  GPIOF->DIR |= (color) | (1 << 5) | (1 << 6); // use PF5, PF6 for the external LEDs
  GPIOF->DEN |= (color) | (1 << 5) | (1 << 6); // use PF5, PF6 for the external LEDs

  Time1A_1sec_delay();
  Time2A_10sec_delay();
  Time0B_5sec_delay();

  while(1)
  {
    // do nothing wait for the interrupt to occur
  }
}
/* Timer1 used for the on-board LED */
void TIMER1A_Handler()
{
  if(TIMER1->MIS & 0x1)
    GPIOF->DATA ^= color; /* toggle Blue LED*/
  TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears*/
}

/* used for the first external LED (PF5) */
```



```c
/* Timer1 used for the on-board LED */
void TIMER1A_Handler()
{
  if(TIMER1->MIS & 0x1)
    GPIOF->DATA ^= color; /* toggle Blue LED*/
  TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears*/
}

/* used for the first external LED (PF5) */
void TIMER2A_Handler()
{
  if(TIMER2->MIS & 0x1)
    GPIOF->DATA ^= (1 << 5); /* toggle Blue LED*/
  TIMER2->ICR = 0x1; /* Timer1A timeout flag bit clears*/

}

/* used for the second external LED (PF6) */
void TIMER1B_Handler()
{
  if(TIMER1->MIS & 0x1)
    GPIOF->DATA ^= (1 << 6); /* toggle Blue LED*/
  TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears*/
}

void Time1A_1sec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x4; /*select 16-bit configuration option */
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAPR = 250-1; /* TimerA prescaler value */
  TIMER1->TAILR = 64000-1; /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
}

```



```c
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAPR = 250-1; /* TimerA prescaler value */
  TIMER1->TAILR = 64000-1; /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
}


void Time2A_10sec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<2); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x00; /*select 16-bit configuration option */
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAILR = 160000000-1; /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<23); /*enable IRQ23 */
}


void Time0B_5sec_delay(void)
{
  SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
  TIMER1->CTL = 0; /* disable timer1 output */
  TIMER1->CFG = 0x00; /*select 32-bit configuration option */
  TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
  TIMER1->TAILR = (80000000-1); /* TimerA counter starting count down value */
  TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
  TIMER1->IMR |=(1<<0); /*enables TimerA time-out interrupt mask */
  TIMER1->CTL |= 0x01; /* Enable TimerA module */
  NVIC->ISER[0] |= (1<<22); /*enable IRQ22 */
}
```

4. Conclusion

The objective of this experiment was to utilize TM4C123 microcontrollers and program Timer interrupts. By employing ARM Cortex M4 microcontrollers, students gained knowledge on configuring TimerA to generate a 1-second interrupt. Additionally, they explored various applications of general-purpose timers. The experiment focused on tasks such as initializing TimerA registers, configuring interrupts, and implementing the Timer Interrupt Handler function. Through this process, valuable insights into interrupt processing on ARM Cortex-M CPUs were obtained.

5. References

[1] from https://microcontrollerslab.com/timer-interrupt-tm4c123-generate-delay-with-timer-interrupt-service-routine/
[2] Digital Lab Manual, BZU.