



Faculty of Engineering & Technology

Electrical & Computer Engineering Department

COMPUTER DESIGN LABORATORY ENCS 4110

Experiment No. 8: Timers Interrupts

Report 2

Prepared by:

Mohammad Abu Shams

1200549

Instructor: Dr. Abualseoud Hanani

Assistant: Eng. Raha Zabadi

Section: 1

Date: 20-8-2022

Birzeit

Abstract

This experiment's goals are to learn about internal timers and counters and how to put up a nested vector interrupt controller. We will discover how to set the timers to create internal interrupt. We will talk about ARM Cortex M4 TM4C123 Timer Interrupt Programming. microcontrollers. We will first talk about some uses for timer interrupts using one example. For demonstrations, we'll use a TM4C123 Tiva C launchpad. A TM4C123GH6PM microcontroller is included with it. To produce a 1s periodic interrupt, the general-purpose timer module TimerA module will be set up and initialized. In other words, the TimerA module's interrupt handler method will run once every second. The exception handler method allows us to carry out the tasks we need to.

Table of Contents

Abstract.....	I
List of figures	III
List of tables	IV
Theory.....	1
General Purpose Timer Interrupt	1
Applications	2
How to configure Timer Interrupt of TM4C123 TM4C123	3
Timer1A Interrupt with One Second Delay	4
Initialize Timer1A registers for one second Delay	5
Prescaler Configuration.....	6
Configure TM4C123 Timer Interrupt	7
Implementation of Timer Interrupt Handler function.....	8
Types of Interrupt and Exceptions in ARM Cortex-M	9
What Happends when Interrupt Occurs?	10
Interrupt Vector Table and Interrupt Processing	11
Relocating ISR Address from IVT	12
Examples	14
Steps Executed by ARM Cortex M processor During Interrupt Processing.....	15
ARM Cortex-M Context Switching	16
Interrupts Processing Steps.....	17
Stacking	17
Exception Entry	17
Exception Return	18
Procedure	19
TM4C123 Timer Interrupt Example Code.....	19
In-Lab Task	21
Task1	21
Task2.....	23
Conclusion.....	25
Feedback.....	26
References	27

List of figures

<i>Figure 1: System memory region [11].....</i>	<i>11</i>
<i>Figure 2: The interrupt vector table of the ARM Cortex-M4 microcontroller</i>	<i>12</i>
<i>Figure 3: The ARM Cortex-M microcontroller</i>	<i>15</i>
<i>Figure 4: TM4C123 Timer interrupt example code</i>	<i>19</i>
<i>Figure 5: Task1 code</i>	<i>21</i>
<i>Figure 6: Task2 code</i>	<i>23</i>

List of tables

<i>Table 1: The interrupt vector table of ARM Cortex M4 based TM4C123GH6PM microcontroller [12]</i>	<i>13</i>
---	-----------

Theory

General Purpose Timer Interrupt

Programmable general purpose timer modules (GPTM) of TM4C123 microcontroller can be used to count external events as a counter or as a timer. For example, we want to measure an analog signal with the ADC of TM4C123 microcontroller after every one second. By using GPTM, we can easily achieve this functionality. In order to do so, first configure the timer interrupt of TM4C123 to generate an interrupt after every one second. Second, inside the interrupt service routine of the timer, sample the analog signal value with ADC and turn off ADC sampling before returning from the interrupt service routine. Similarly, general purpose timer modules have many applications in embedded systems [1].

Applications

Few important applications are:

- External event measurement (Example HC-SR04 with TM4C123).
- Pulse width measurement - Frequency measurement.
- Motor RPM Measurement TM4C123 [2].

How to configure Timer Interrupt of TM4C123

microcontroller provides two timer blocks such as Timer A and Timer B. Each block has six 16/32 bits GPTM and six 32/64 bits GPTM. We will use the TimerA block in this experiment [3].

Timer1A Interrupt with One Second Delay

First, create a project in Keil uvision by selecting TM4C123GH6PM microcontroller Include the header file of TM4C123GH6PM microcontroller which contains a register definition file of all peripherals such as timers.

```
#include "TM4C123.h" // Device header file for Tiva Series Microcontroller.
```

In this section, we create a timer interrupt of 1Hz. That means the interrupt will occur after every one second. Because the time period is inverse of the frequency. Whenever an interrupt occurs, we will toggle an LED inside the corresponding interrupt service routine of TimerA module. In short, the interrupt service routine will execute every one second and toggle the onboard LED of TM4C123 Tiva C Launchpad. TM4C123 Tiva C Launchpad has an onboard RGB LED. Blue LED is connected with the PF3 pin of PORTF. We will toggle this LED inside the interrupt service routine of TimerA. First let's define a symbol name for this PF3 pin using #define preprocessor directive.

```
#define Blue (1<<2) // PF3 pin of TM4C123 Tiva Launchpad, Blue LED [4].
```

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.

```
SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
```

Before initialization, disables the Timer1 output by clearing GPTMCTL register.

```
TIMER1->CTL = 0; /* disable timer1 output */
```

First three bits of GPTMCFG register is 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. (check datasheet 726 of TM4C123GH6PM).

```
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 (refer to page 732 datasheet):

```
TIMER1->TMAR = 0x02; /*select periodic down counter mode of timer1, TMAR=0x01 for oneshot 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:

$$2^{16} = 65536 \text{ and } 16\text{MHz} = 16000000$$

$$\text{Maximum delay} = 65536 / 16000000 = 4.096 \text{ millisecond}$$

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 [5].

Prescaler Configuration

Prescaler adds additional bits to the size of the timer. GPTM Timer A 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.

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. $16000000/250 = 64000\text{Hz} = 64\text{KHz}$. Hence, the time period for TimerA is $1/64000 = 15.625\text{ms}$. 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 */ [6]
```

Configure TM4C123 Timer Interrupt

There are two steps to enable the interrupt of peripherals in ARM Cortex-M microcontrollers. Firstly, enable the interrupt from the peripheral level (Timer module in this case) using their interrupt mask register. GPTM Interrupt Mask (GPTMIMR) register enables or disables the interrupt for the general purpose timer module of TM4C123 microcontroller. Bit0 of GPTMIMR enables the TimerA time-out interrupt mask.

```
TIMER1->IMR |= (1<<0); /*enables TimerA time-out interrupt mask */
```

Secondly, we must also enable the GPTM interrupt request to a nested vectored interrupt controller using their interrupt enable registers. Interrupt request number of Timer1A is 21 or IRQ21. Hence, it corresponds to NVIC interrupt enable register zero. This line enables the Timer1A to interrupt from NVIC level by setting bit21:

```
NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
```

You can also find the interrupt number of every exception handler or ISR inside the startup file of TM4C123 microcontroller [7].

Implementation of Timer Interrupt Handler function

ISR routines of all peripheral interrupts and exception routines are defined inside the startup file of TM4C123 Tiva microcontroller and the interrupt vector table is used to relocate the starting address of each interrupt function. In order to find the name of the handler function of Timer1 and sub-timer A, open the startup file and you will find that the name of the Timer1A handler routine is `TIMER1A_Handler()`. By using this name of the Timer1A interrupt handler, we can write a c function inside the main code like this:

```
TIMER1A_Handler()

{

// instructions you want to implement

} [8]
```

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. But 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 [9].

What Happends when Interrupt Occurs?

But whenever an exception or interrupt occurs, the processor uses an interrupt service routine for interrupts and an exception handler for system exceptions. In other words, whenever an interrupt occurs from a particular source, the processor executes a corresponding piece of code (function) or interrupt service routine [10].

Interrupt Vector Table and Interrupt Processing

But the question is how does the processor find the addresses of these interrupt service routines or exception handlers? In order to understand this, let's take a review of the memory map of ARM cortex M4 microcontrollers. The address space which 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:

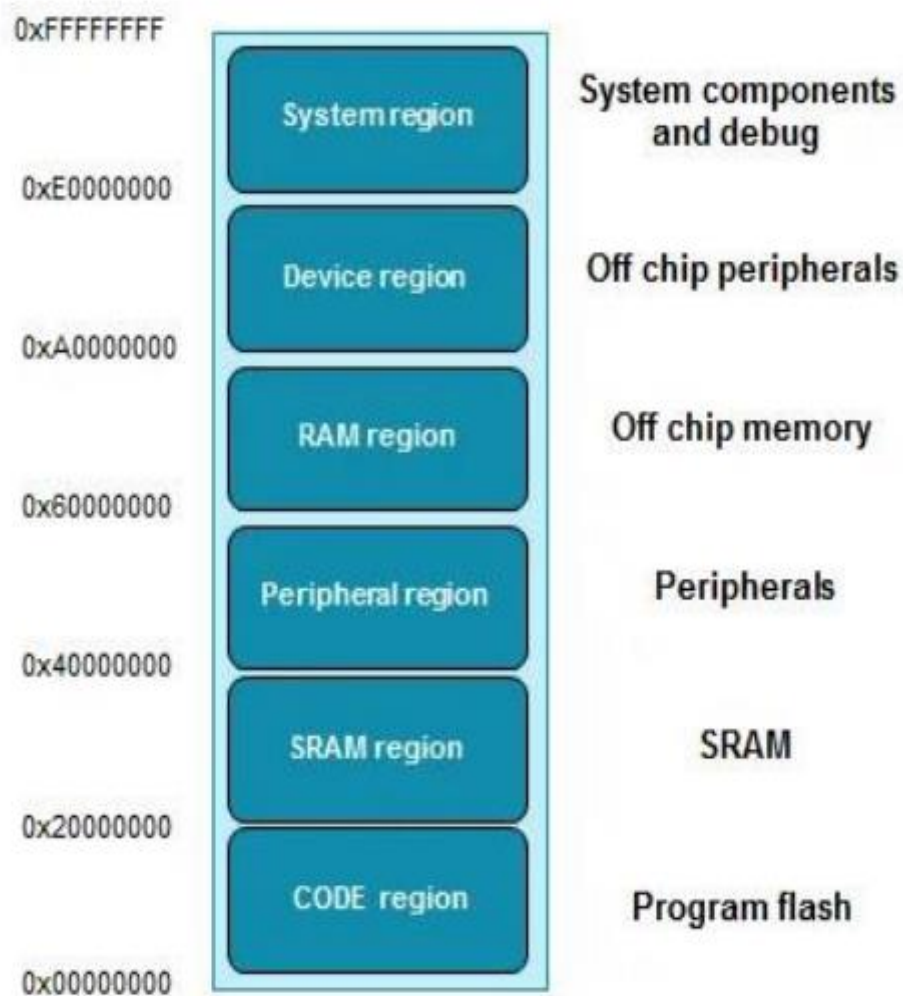


Figure 1: System memory region [11]

Relocating ISR Address from IVT

Coming back to the main discussion, the code memory region contains an interrupt vector table (IVT). This interrupt vectors 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 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.

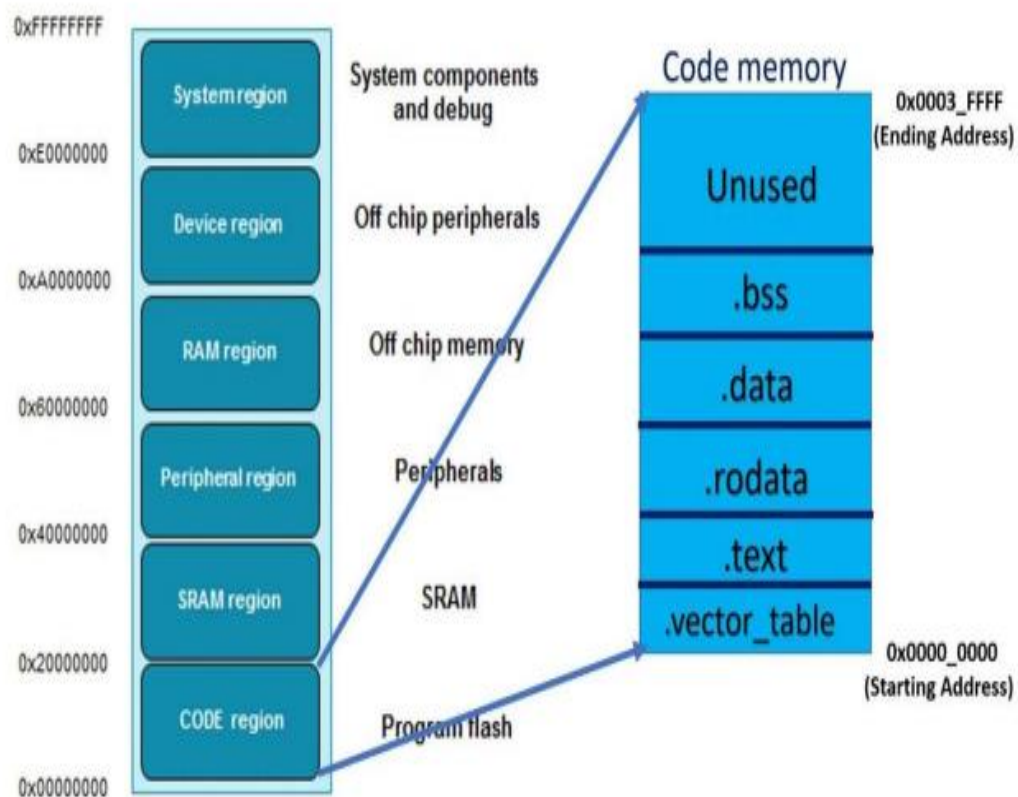


Figure 2: The interrupt vector table of the ARM Cortex-M4 microcontroller

The figure above shows the interrupt vector table of the ARM Cortex-M4 microcontroller. As you can see, 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.

Table 1: The interrupt vector table of ARM Cortex M4 based TM4C123GH6PM microcontroller [12]

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	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

Examples

For example, when an interrupt x occurs, the nested vectored interrupt controller uses this interrupt number to find the memory address of the interrupt service routine inside the IVT. After finding the starting address of the exception handler, NVIC sends the request to the ARM microcontroller to start executing the interrupt service routine.

Lets take an example, let's suppose, interrupt number 2 occurs. The NVIC used this formula to find the entry of crossponding IRQ x in IVT:

$$\text{Entry in IVT} = 64 + 4 * x ;$$

for $x=2$

$$\text{Entry in IVT} = 64 + 4 * 2 = 72$$

$$= 72 \text{ or } 0x0048$$

As you can see from the above interrupt vector table, the address of IRQ2 is 0X0048. NVIC will get the starting address of IRQ2 from the memory location 0x0048 and sets the program counter to the starting address of IRQ2. After that processor jumps to this locations to execute ISR.

One important point to note here is that the value of the interrupt number is negative also. The interrupt number or IRQ n of all system exceptions is in negative. This example shows the relocation of entry in IVT when bus fault exception occurs:

$$\text{Entry in IVT} = 64 + 4 * x ;$$

For bus fault exception $x = -11$

$$\text{Entry in IVT} = 64 + 4 * -11 = 20$$

$$= 20 \text{ or } 0x0014$$

[13].

Steps Executed by ARM Cortex M processor During Interrupt Processing

Till now we have discussed how ARM microcontroller finds the address of the first instruction of 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. But what happens during context switching requires more explanation.

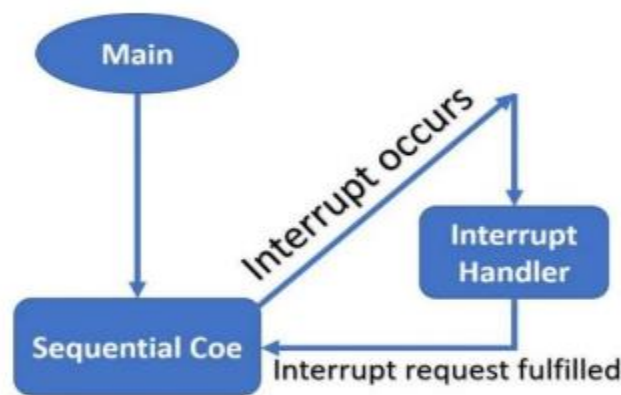


Figure 3: The ARM Cortex-M microcontroller

For example, the ARM cortex-M processor executing these instruction inside the main code:

1. LDR R2 [R0] // load value at address [R0] in register R2
2. LDR R1 [R0+4] // load value at address [R0+4] in register R1
3. ADD R3 R2 R1 // add R2+R1 and save the content in R3
4. STR R3 [R0] // Store R3 at address [R0]

The main application is executing the above instructions and the program counter is pointing to instruction 3. At the same time, an interrupt signal arrives [14].

ARM Cortex-M Context Switching

When an interrupt occurs and its request is approved by the NVIC and processor to execute, the contents of the CPU register are saved onto the stack. This way of CPU register preservation helps microcontrollers to start execution of the main application from 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.

But it takes time for the microcontroller to save the CPU register's content onto the stack. But 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[15].

Interrupts Processing Steps

The microcontroller will perform the following steps:

Stacking

1- First ARM Cortex-M finishes or terminates currently under execution like the instruction number 3 (ADD R3 R2 R1) in the above example. The condition of finishing or terminating current instruction depends on the value of the interrupt continuable instruction (ICI) field in the xPSR register.

2- After that suspend the current main application and save the context of the main application code into the stack. Microcontroller pushes registers R0, R1, R2, R3, R12, LR, Program counter and program status register (PSR) onto the stack.

The order in which stacking take place is PSR, PC, LR, R12, R3, R2, R1 and R0.

Exception Entry

After that, the ARM processor reads the interrupt number from the xPSR register. By using this interrupt number processor finds the entry of the exception handler in the interrupt vector table. Finally, it reads the starting address of the exception handler from the respective entry of IVT.

Now ARM processor updates the values of the stack pointer, linker register (LR), PC (program counter) with new values according to the interrupt service routine. The link register contains the type of interrupt return address.

After that interrupt services routine starts to execute and finish its execution.

Exception Return

The last step is to return to the main application code or to exist from the interrupt service routine. To return from ISR, the processor loads the link register (LR) with a special value. The most significant 24-bits of this value are set to 0xFFFFF and the least significant eight bits provide different ways to return from exception mode. For example, if the least significant 8-bits are F9. The processor will return from exception mode by popping all eight registers from the stack. In addition, it will return to thread mode using MSP as its stack pointer value.

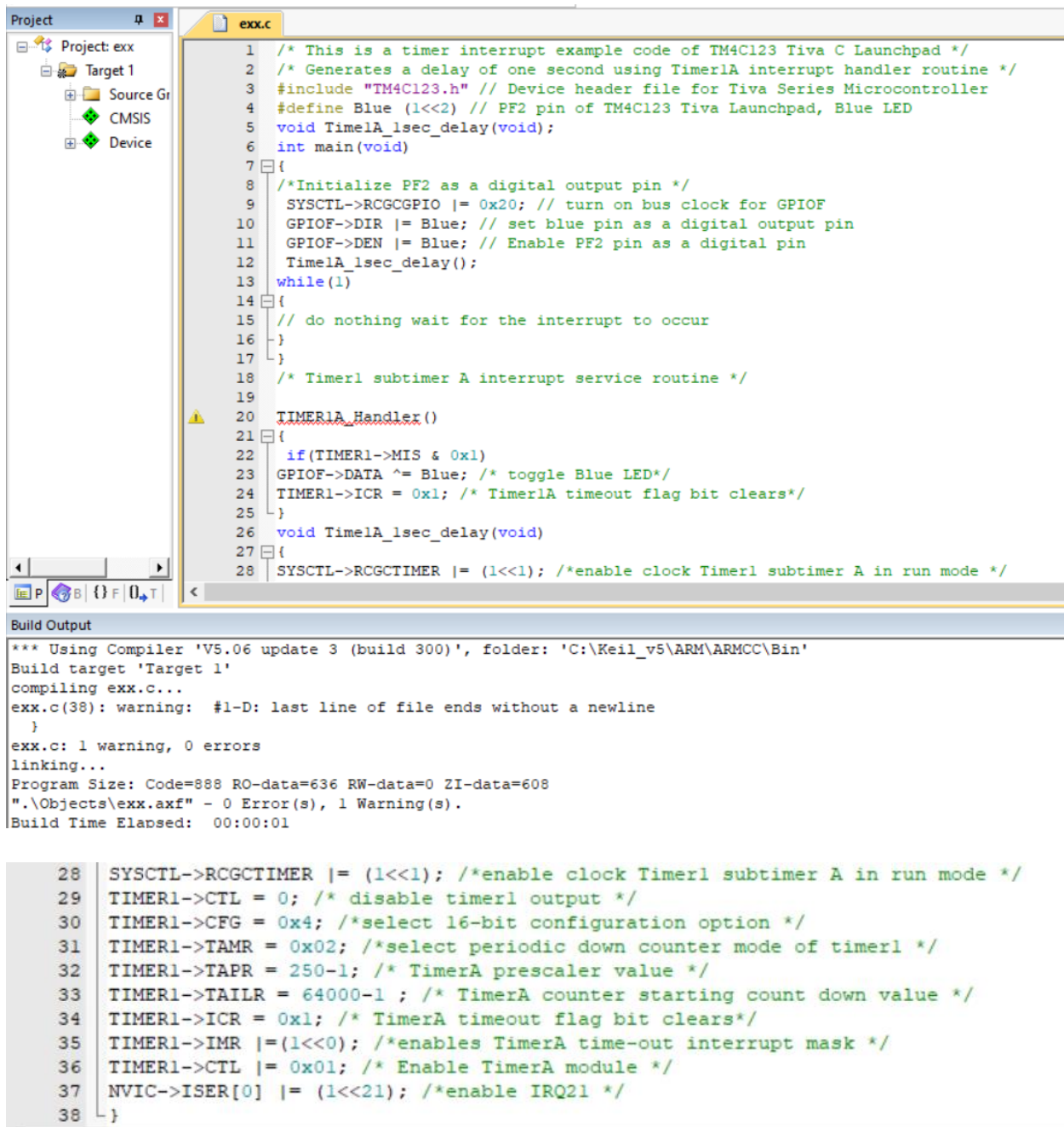
Least significant 8-bits of the value which is loaded to LR register during the exception return process determine which mode to return either remain in exception mode (in case of nested interrupts) or handler mode.

After that microcontroller starts to execute the main application from the same instruction where it is pre-empted by interrupt service routine [16].

Procedure

TM4C123 Timer Interrupt Example Code

This example code of TM4C123 Tiva C Launchpad generates a delay of one second using the Timer1A interrupt handler routine. Inside the main code, we initialize the PF2 pin as a digital output pin



The screenshot displays the Keil uVision IDE interface. The left pane shows the project structure for 'Project: exx', including 'Target 1', 'Source Gr', 'CMSIS', and 'Device'. The main editor window shows the source code for 'exx.c'. The code includes comments and C code for initializing the PF2 pin as a digital output and setting up a timer interrupt. The build output window at the bottom shows the compilation and linking process, including a warning about a missing newline at the end of the file.

```
1  /* This is a timer interrupt example code of TM4C123 Tiva C Launchpad */
2  /* Generates a delay of one second using Timer1A interrupt handler routine */
3  #include "TM4C123.h" // Device header file for Tiva Series Microcontroller
4  #define Blue (1<<2) // PF2 pin of TM4C123 Tiva Launchpad, Blue LED
5  void TimerA_lsec_delay(void);
6  int main(void)
7  {
8      /*Initialize PF2 as a digital output pin */
9      SYSTCL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
10     GPIOF->DIR |= Blue; // set blue pin as a digital output pin
11     GPIOF->DEN |= Blue; // Enable PF2 pin as a digital pin
12     TimerA_lsec_delay();
13     while(1)
14     {
15         // do nothing wait for the interrupt to occur
16     }
17 }
18 /* Timer1 subtimer A interrupt service routine */
19
20 TIMER1A_Handler()
21 {
22     if(TIMER1->MIS & 0x1)
23     {
24         GPIOF->DATA ^= Blue; /* toggle Blue LED*/
25         TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears*/
26     }
27 }
28 void TimerA_lsec_delay(void)
29 {
30     SYSTCL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
31     TIMER1->CTL = 0; /* disable timer1 output */
32     TIMER1->CFG = 0x4; /*select 16-bit configuration option */
33     TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
34     TIMER1->TAPR = 250-1; /* TimerA prescaler value */
35     TIMER1->TAIRL = 64000-1; /* TimerA counter starting count down value */
36     TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
37     TIMER1->IMR |= (1<<0); /*enables TimerA time-out interrupt mask */
38     TIMER1->CTL |= 0x01; /* Enable TimerA module */
39     NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
40 }
```

Build Output

```
*** Using Compiler 'V5.06 update 3 (build 300)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
compiling exx.c...
exx.c(38): warning: #1-D: last line of file ends without a newline
}
exx.c: 1 warning, 0 errors
linking...
Program Size: Code=888 RO-data=636 RW-data=0 ZI-data=608
".\Objects\exx.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed: 00:00:01
```

Figure 4: TM4C123 Timer interrupt example code

This program purpose is to turn on the blue LED with delay of one second. I defined Blue in a pin PF2 by shifting 1 by 2, 1 is in pin PF0 and after shifting it in by 2 it becomes in pin PF2 which enable the blue color, and then I turned on bus clock for GPIOF by put 1 in port F (0x20=00100000), then I complete wrote the code and I found the time counter by using this equation:

Time counter = time of delay / (1/ frequency).

And the frequency = 16MHz / prescaler.

now in this example, first I found the frequency, and the prescaler in this example equal 250, so the frequency = 16MHz/ 250

$$= (16*10^6)/250$$

$$=64000 \text{ Hz} = 64 \text{ Khz}$$

And 1/ frequency = 1/64000

$$=1.5625*10^{-5} \text{ second}$$

And the time counter = time of delay / 1.5625*10⁻⁵

$$= 1/1.5625*10^{-5}$$

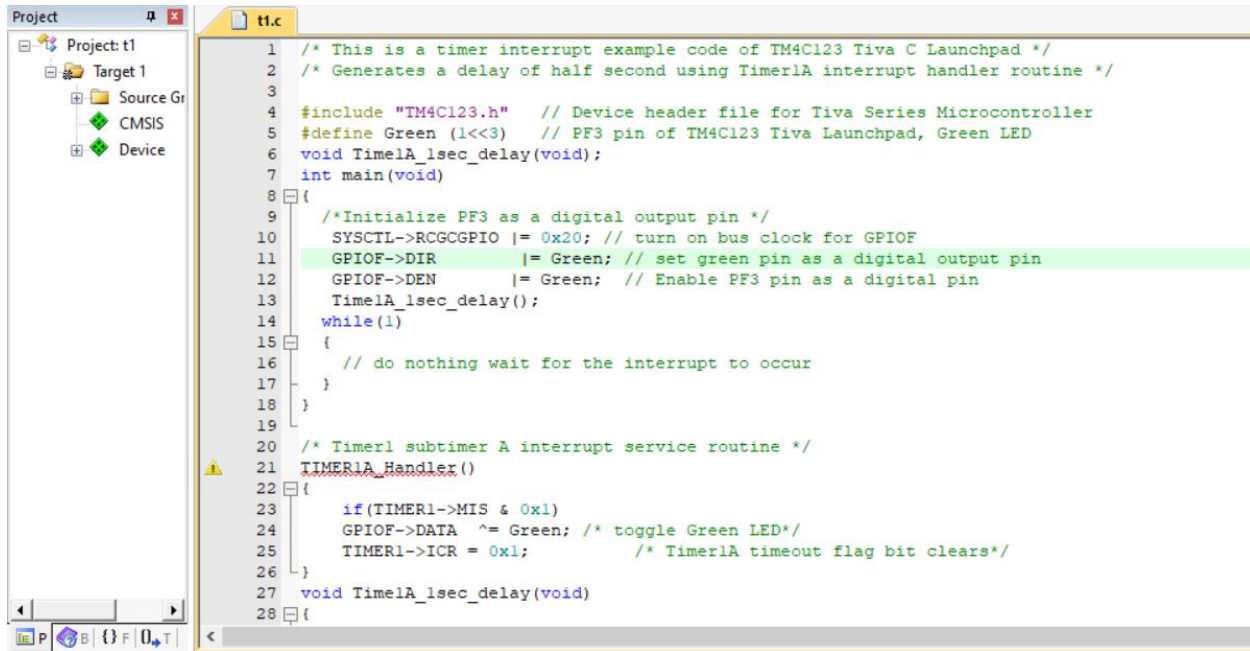
$$= 64000 \text{ second}$$

So the time counter equal 64000 second to turn on the blue LED every 1 second.

In-Lab Task

Task1

Modify the code above to make the GREEN Led blinks every **500ms**.



```
1  /* This is a timer interrupt example code of TM4C123 Tiva C Launchpad */
2  /* Generates a delay of half second using Timer1A interrupt handler routine */
3
4  #include "TM4C123.h" // Device header file for Tiva Series Microcontroller
5  #define Green (1<<3) // PF3 pin of TM4C123 Tiva Launchpad, Green LED
6  void TimerA_lsec_delay(void);
7  int main(void)
8  {
9      /*Initialize PF3 as a digital output pin */
10     SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
11     GPIOF->DIR |= Green; // set green pin as a digital output pin
12     GPIOF->DEN |= Green; // Enable PF3 pin as a digital pin
13     TimerA_lsec_delay();
14     while(1)
15     {
16         // do nothing wait for the interrupt to occur
17     }
18 }
19
20 /* Timer1 subtimer A interrupt service routine */
21 TIMER1_Handler()
22 {
23     if(TIMER1->MIS & 0x1)
24         GPIOF->DATA ^= Green; /* toggle Green LED*/
25     TIMER1->ICR = 0x1; // Timer1A timeout flag bit clears*/
26 }
27 void TimerA_lsec_delay(void)
28 {
```

Build Output

```
*** Using Compiler 'V5.06 update 3 (build 300)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
compiling t1.c...
linking...
Program Size: Code=888 RO-data=636 RW-data=0 ZI-data=608
".\Objects\t1.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

```
28 {
29     SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
30     TIMER1->CTL = 0; /* disable timer1 output */
31     TIMER1->CFG = 0x4; /*select 16-bit configuration option */
32     TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
33     TIMER1->TAPR = 250-1; /* TimerA prescaler value */
34     TIMER1->TAILR = 32000-1; /* TimerA counter starting count down value */
35     TIMER1->ICR = 0x1; // TimerA timeout flag bit clears*/
36     TIMER1->IMR |= (1<<0); /*enables TimerA time-out interrupt mask */
37     TIMER1->CTL |= 0x01; // Enable TimerA module */
38     NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
39 }
40
```

Figure 5: Task1 code

I modify the previous code to make the Green LED turn on every 500ms.

I defined Green in a pin PF3 by shifting 1 by 3, 1 is in pin PF0 and after shifting it in by 3 it becomes in pin PF3 which enable the green color, and then I turned on bus clock for GPIOF by put 1 in port F (0x20=00100000), then I complete wrote the code and I found the time counter by using this equation:

Time counter = time of delay / (1/ frequency).

And the frequency = 16MHz / prescaler.

now in this example, first I found the frequency, and the prescaler in this example equal 250, so the frequency = 16MHz/ 250

$$= (16*10^6)/250$$

$$=64000 \text{ Hz} = 64 \text{ Khz}$$

And 1/ frequency = 1/64000

$$=1.5625*10^{-5} \text{ second}$$

And the time counter = time of delay / 1.5625*10⁻⁵

$$= 0.5/1.5625*10^{-5}$$

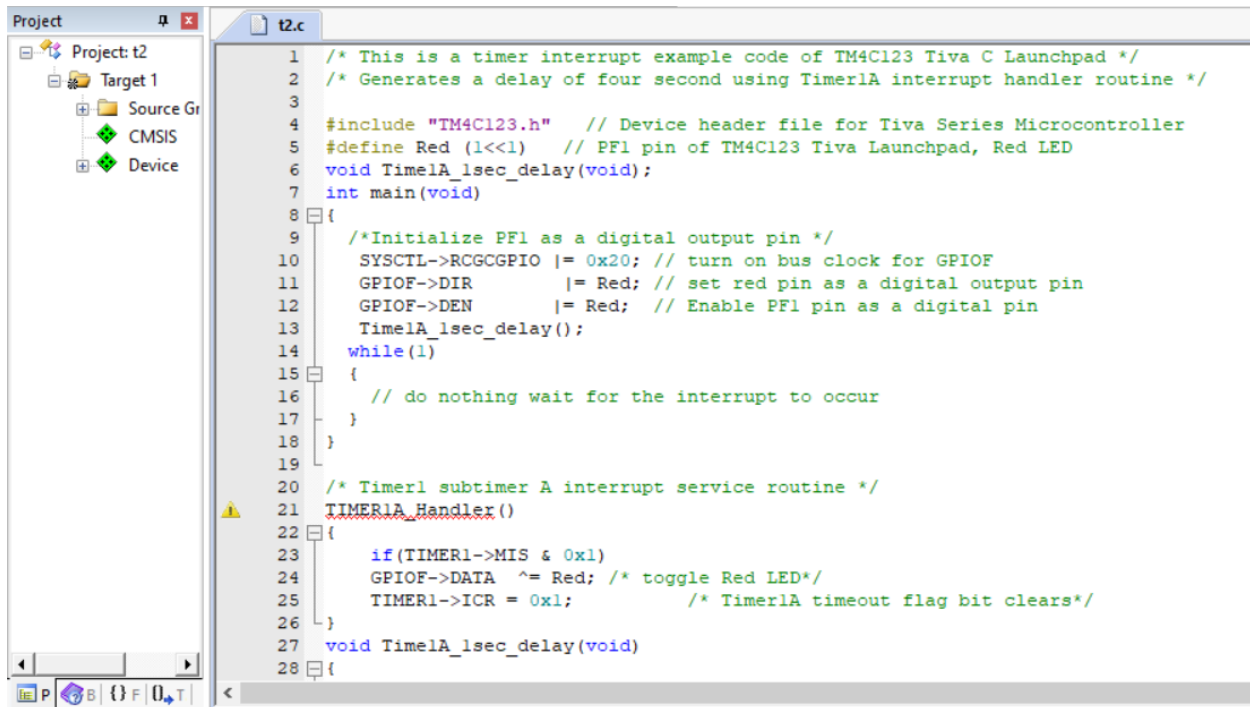
$$= 32000 \text{ second}$$

So the time counter equal 32000 second to turn on the green LED every 500ms.

Also I can keep the time counter equal 64000 as in the previous code, but in this case I will reduce time prescaler into a half and it will be 125 instead of 250, then the green LED is also blinks every 500ms.

Task2

Modify the code above to make the RED Led blinks every 4s.



The screenshot shows the Keil uVision IDE with a project named 't2'. The source file 't2.c' is open, displaying the following code:

```
1  /* This is a timer interrupt example code of TM4C123 Tiva C Launchpad */
2  /* Generates a delay of four second using Timer1A interrupt handler routine */
3
4  #include "TM4C123.h" // Device header file for Tiva Series Microcontroller
5  #define Red (1<<1) // PF1 pin of TM4C123 Tiva Launchpad, Red LED
6  void TimerA_lsec_delay(void);
7  int main(void)
8  {
9      /*Initialize PF1 as a digital output pin */
10     SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
11     GPIOF->DIR      |= Red; // set red pin as a digital output pin
12     GPIOF->DEN      |= Red; // Enable PF1 pin as a digital pin
13     TimerA_lsec_delay();
14     while(1)
15     {
16         // do nothing wait for the interrupt to occur
17     }
18 }
19
20 /* Timer1 subtimer A interrupt service routine */
21 TIMER1A_Handler()
22 {
23     if(TIMER1->MIS & 0x1)
24     {
25         GPIOF->DATA ^= Red; /* toggle Red LED*/
26         TIMER1->ICR = 0x1; // Timer1A timeout flag bit clears*/
27     }
28 }
29 void TimerA_lsec_delay(void)
30 {
31     SYSCTL->RCGCTIMER |= (1<<1); /*enable clock Timer1 subtimer A in run mode */
32     TIMER1->CTL = 0; /* disable timer1 output */
33     TIMER1->CFG = 0x0; /*select 32-bit configuration option */
34     TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */
35     //TIMER1->TAPR = 250-1; /* TimerA prescaler value */
36     TIMER1->TAILR = 64000000-1; /* TimerA counter starting count down value */
37     TIMER1->ICR = 0x1; // TimerA timeout flag bit clears*/
38     TIMER1->IMR |= (1<<0); /*enables TimerA time-out interrupt mask */
39     TIMER1->CTL |= 0x01; // Enable TimerA module */
40     NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
41 }
```

The Build Output window shows the following text:

```
*** Using Compiler 'V5.06 update 3 (build 300)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
assembling startup_TM4C123.s...
compiling t2.c...
compiling system_TM4C123.c...
linking...
Program Size: Code=884 RO-data=636 RW-data=0 ZI-data=608
".\Objects\t2.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
```

The code in the image is a modified version of the original code, where the delay is set to 4 seconds (64000000-1) and the timer is configured for periodic down counter mode. The code is shown in a separate block below the screenshot.

Figure 6: Task2 code

I modify the first code to make the Red LED turn on every 4s.

I defined Red in a pin PF1 by shifting 1 by 1, 1 is in pin PF0 and after shifting it in by 1 it becomes in pin PF1 which enable the red color, and then I turned on bus clock for GPIOF by put 1 in port F (0x20=00100000), then I complete wrote the code and I used 32-bit configuration option instead of 16-bit, put it 0x0, and then the time prescaler canceled and put 1 instead of it, and I found the time counter by using this equation:

Time counter = time of delay / (1/ frequency).

And the frequency = 16MHz / prescaler.

now in this example, first I found the frequency, and the prescaler in this example equal 1, so the frequency = 16MHz/1

$$= (16*10^6) \text{ Hz}$$

And 1/ frequency = 1/(16*10⁶)

$$=6.25*10^{-8} \text{ second}$$

And the time counter = time of delay / 6.25*10⁻⁸

$$= 4/6.25*10^{-8}$$

$$= 64*10^6 \text{ second}$$

$$= 64000000$$

So the time counter equal 64000000 second to turn on the red LED every 4s.

Conclusion

In this experiment, I learnt about timers interrupt and I wrote some codes on this topics including : generates a delay of one second using Timer1A interrupt handler routine to make the blue LED blinks every one second and then modify this code to make the green LED blinks every 500ms, also to make the red LED blinks every 4s. Now I can deal with timers and I understood them and writing codes using them.

Feedback

This experiment is a nice one, I learnt some new ideas about timers interrupt and the time of this experiment was perfect and I finished my tasks before ending time.

References

- [1] *ENCS4110 Lab Manual Page 1*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [2] *ENCS4110 Lab Manual Page 2*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [3] *ENCS4110 Lab Manual Page 2*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [4] *ENCS4110 Lab Manual Page 2*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [5] *ENCS4110 Lab Manual Page 3*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [6] *ENCS4110 Lab Manual Page 3&4*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [7] *ENCS4110 Lab Manual Page 4*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [8] *ENCS4110 Lab Manual Page 4&5*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf
- [9] *ENCS4110 Lab Manual Page 5*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4110/Exp8_Timer_Interrupt.pdf

- [10] *ENCS4110 Lab Manual Page 5*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf
- [11] *ENCS4110 Lab Manual Page 6*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf
- [12] *ENCS4110 Lab Manual Page 7*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf
- [13] *ENCS4110 Lab Manual Page 7&8*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf
- [14] *ENCS4110 Lab Manual Page 8&9*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf
- [15] *ENCS4110 Lab Manual Page 9* . (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf
- [16] *ENCS4110 Lab Manual Page 9&10*. (2022, 8 20). Retrieved from
file:///C:/Users/lenovo/OneDrive/Desktop/semesters/The%20second%20summer/ENCS4
110/Exp8_Timer_Interrupt.pdf