



**Faculty of Engineering and Technology
Electrical and Computer Engineering Department**

**Computer Design Laboratory
ENCS4110**

**Report of Experiment No. 8
Timers Interrupts**

Student's Name: Maha Mali

Student's ID: 1200746

Sec: 1

Instructor: Dr. Abualseoud Hanani

Teacher assistant: Eng. Raha Zabadi

Date: 19/8/2022

Abstract

The main goal of this experiment is to know the internal timers/counters, also to learn how to setup nested vector interrupt controller. In addition to that we want to learn how to configure the timers to make internal interrupt.

Table of Contents	
Abstract.....	I
List Of Figures.....	IV
1.Theory	1
1.1 General Purpose Timer Interrupt	1
1.2 Applications	1
1.3 How to configure Timer Interrupt of TM4C123.....	1
2.Procedure and Discussion	3
2.1 Timer1A Interrupt with One Second Delay	3
2.2 Initialize Timer1A registers for one second Delay	4
2.3 Prescaler Configuration.....	5
2.4 Configure TM4C123 Timer Interrupt	7
2.5 Implementation of Timer Interrupt Handler function	8
2.6 Types of Interrupt and Exceptions in ARM Cortex-M	9
2.7 What Happens when Interrupt Occurs?.....	9
2.8 Interrupt Vector Table and Interrupt Processing	9
2.9 Relocating ISR Address from IVT	10
2.10 Examples	12
2.11 Steps Executed by ARM Cortex M processor During Interrupt Processing.....	13
2.12 ARM Cortex-M Context Switching	13
2.13 Interrupts Processing Steps.....	14
2.14 TM4C123 Timer Interrupt Example Code.....	16
2.15 Lab Work	17
2.15.1 Task1	17
2.15.2 Task 2	18
3. Conclusion	19
4.Feedback	20
5. References	21
6. Appendix.....	22
6.1 Example.....	22
6.2 Task 1	24

6.3 Task 2 26

List Of Figures

Figure1: TM4C123 microcontroller	2
Figure 2: Select TM4C123GH6PM Microcontroller.....	3
Figure 3: Device Header File	3
Figure 4: connect Blue LED	4
Figure 5:Enable Cock	4
Figure 6: Disables The Timer1 Output	4
Figure 7:Select 16-bit Configuration	4
Figure 8: Select The Periodic Mode Of Timer1	5
Figure 9: Equation Of Maximum Delay	5
Figure 10:TimerA Prescaler.....	6
Figure 11: TimerA counter Starting count Down Value	6
Figure 12: Clear the Timeout Flag.....	6
Figure 13: Enable the TimerA module	6
Figure 14: Enable the Interrupt	7
Figure 15:Enable IRQ21	7
Figure 16: Interrupt number.....	8
Figure 17: Implementation of Timer Interrupt Handler function	8
Figure 18: Interrupt Vector Table and Interrupt Processing	9
Figure 19: Relocating ISR Address from IVT	10
Figure 20: interrupt vector table of ARM Cortex M4.....	11
Figure 21:Example.....	12
Figure 22: Relocation Of Entry In IVT.....	12
Figure 23: Steps Executed by ARM Cortex M.....	13
Figure 24:Simulation Of Example	16
Figure 25:Equation For Task 1	17
Figure 26:Simulation for Task 1	17
Figure 27: Equation For Task 2	18
Figure 28: Simulation for Task 2	18

1.Theory

1.1 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 [1].

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].

1.2 Applications

- External event measurement (Example HC-SR04 with TM4C123)[1].
- Pulse width measurement [1] .
- Frequency measurement [1].
- Motor RPM Measurement TM4C123[1].

1.3 How to configure Timer Interrupt of TM4C123

TM4C123 microcontroller provides two timer blocks such as Timer A and Timer B as shown in figure 1. Each block has six 16/32 bits GPTM and six 32/64 bits GPTM. We will use the TimerA block in this experiment [1].

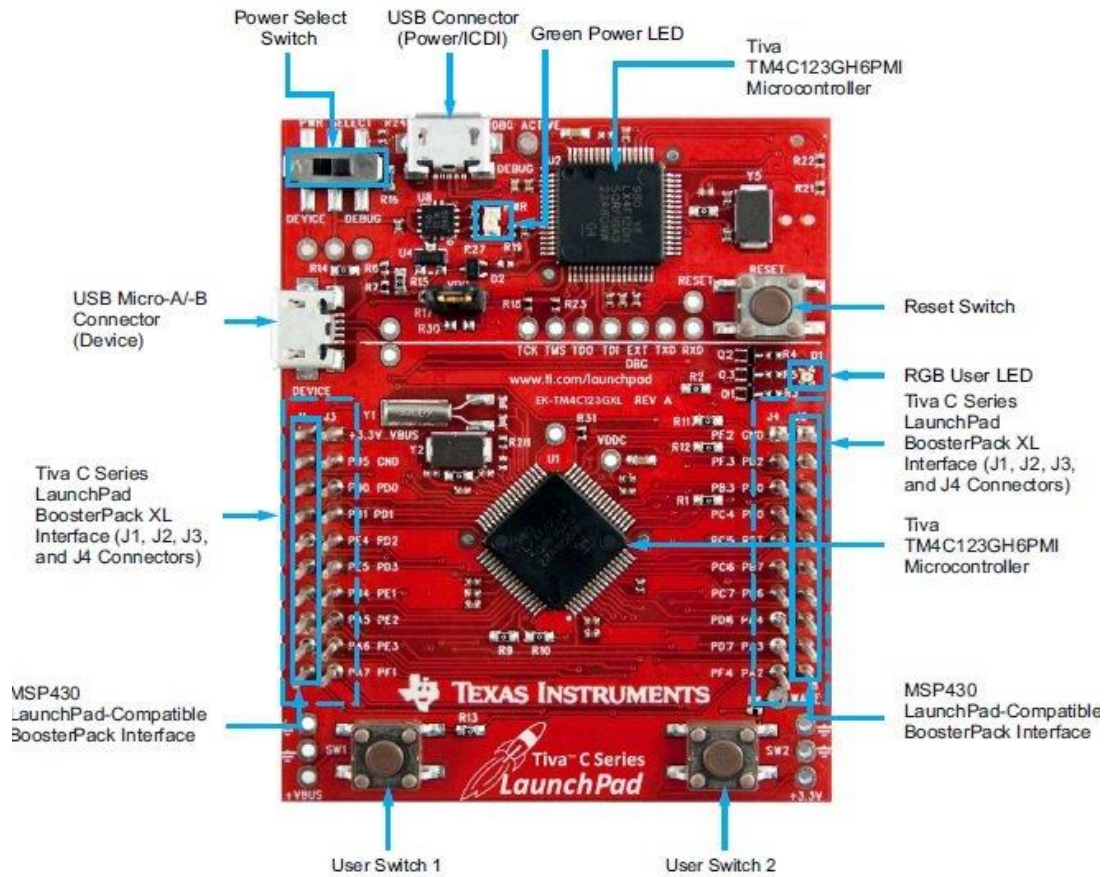


Figure1: TM4C123 microcontroller

2.Procedure and Discussion

At First, we created a project in Keil uvision by selecting TM4C123GH6PM microcontroller as shown in figure 2, include the header file of TM4C123GH6PM microcontroller which contains a register definition file of all peripherals such as timers [1].

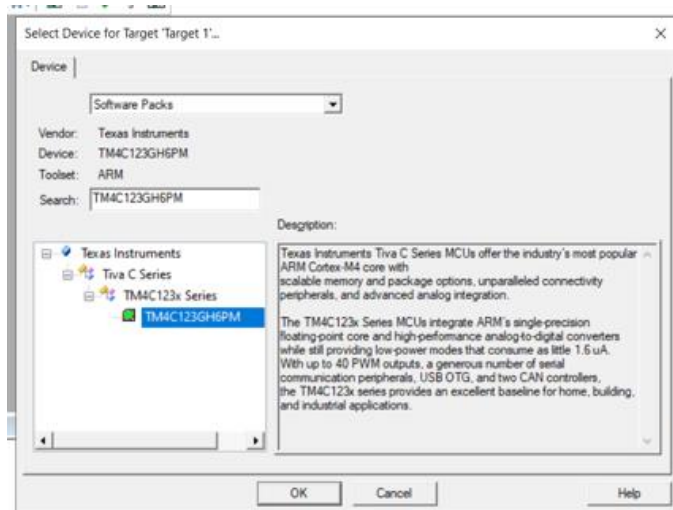


Figure 2: Select TM4C123GH6PM Microcontroller

Then we wrote (#include "TM4C123.h") which is mean the Device header file for Tiva Series Microcontroller [1]. As shown in figure 3.

```
1 #include "TM4C123.h" // Device header file for Tiva Series Microcontroller
2
```

Figure 3: Device Header File

2.1 Timer1A Interrupt with One Second Delay

In this part, we make a timer interrupt of 1Hz. That shows 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 [1].

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 [1]. As shown in figure 4.

```
// Name:Maha Maher Mali
//ID:1200746
#include "TM4C123.h" // Device header file for Tiva Series Microcontroller
#define Blue (1<<2) // PF3 pin of TM4C123 Tiva Launchpad, Blue LED
```

Figure 4: connect Blue LED

2.2 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. The line 5 as shown in figure 5 sets the bit1 of the RCGTIMER register to 1[1].

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

Figure 5:Enable Cock

Before initialization, disables the Timer1 output by clearing GPTMCTL register [1].As shown in figure 6.

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

Figure 6: Disables The Timer1 Output

The 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 [1], as shown in figure 6.

```
TIMER1->CFG = 0x4; /*select 16-bit configuration option */
```

Figure 7:Select 16-bit Configuration

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 as shown in figure 8 , we do this according to datasheet [1].

```
TIMER1->TMAR = 0x02; /*select periodic down counter mode of timer1 */
```

Figure 8: Select The Periodic Mode Of Timer1

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 in figure 9 [1]:

$$\begin{aligned} 2^{16} &= 65536 \\ 16\text{MHz} &= 16000000 \\ \text{Maximum delay} &= 65536/16000000 = 4.096 \text{ millisecond} \end{aligned}$$

Figure 9: Equation Of Maximum Delay

Then 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 [1].

2.3 Prescaler Configuration

A prescaler is an electronic counting circuit used to reduce a high frequency electrical signal to a lower frequency by integer division. The purpose of the prescaler is to allow the timer to be clocked at the rate a user desires [3].

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[1] .

For example, if we want to generate a one-second delay, using a Prescaler value of 250 scales down the operating frequency for TimerA block to 64000Hz then $16000000/250 = 64000\text{Hz} = 64\text{KHz}$. So, the time period for TimerA is $1/64000 = 15.625\text{ms}$. That mean, load the Prescaler register with a value of 250[1].As shown in figure 10 .

```
TIMER1->TAPR = 250-1; /* TimerA prescaler value */
```

Figure 10:TimerA Prescaler

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[1].As shown in figure 11.

```
TIMER1->TAILR = 64000 - 1 /* TimerA counter starting count down value */
```

Figure 11: TimerA counter Starting count Down Value

GPTMICR register is used to clear the timeout flag bit of timer[1]. As shown in figure 12.

```
TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears*/
```

Figure 12: Clear the Timeout Flag

After initialization and configuration, enables the TimerA module which we disabled earlier[1]. As shown in figure 13.

```
TIMER1->CTL |= (1<<0); /* Enable TimerA module */
```

Figure 13: Enable the TimerA module

2.4 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 [1]. As shown in figure 14.

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

Figure 14: Enable the Interrupt

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 bit 21. As shown in figure 15.

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

Figure 15:Enable IRQ21

You can also find the interrupt number of every exception handler or ISR inside the startup file of TM4C123 microcontroller as shown in figure 16.

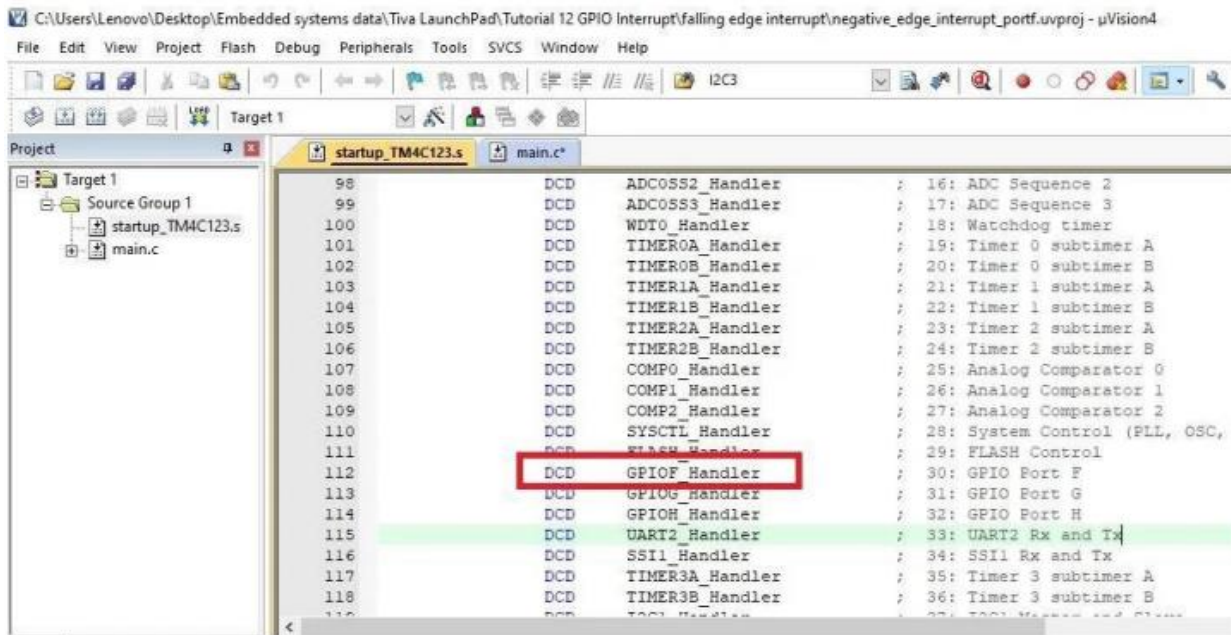


Figure 16: Interrupt number

2.5 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 [1]. We can write a c function inside the main code as shown in figure 17.

```
TIMER1A_Handler()
{
// instructions you want to implement
}
```

Figure 17: Implementation of Timer Interrupt Handler function

2.6 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 [4].

2.7 What Happens 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 [4].

2.8 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 18.

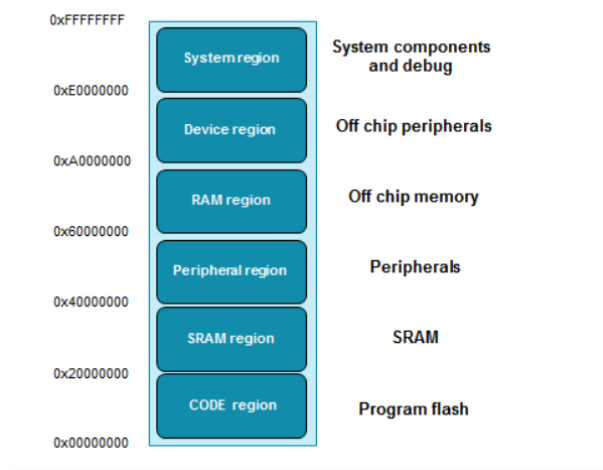


Figure 18: Interrupt Vector Table and Interrupt Processing

The flash or code memory region is used to store the microcontroller's application code and it is further divided into different segments such as vector table, text, read-only data, read-write data, and bss segments. These sections are used for predefined functions.

2.9 Relocating ISR Address from IVT

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 [4]. As shown in the figure 19.

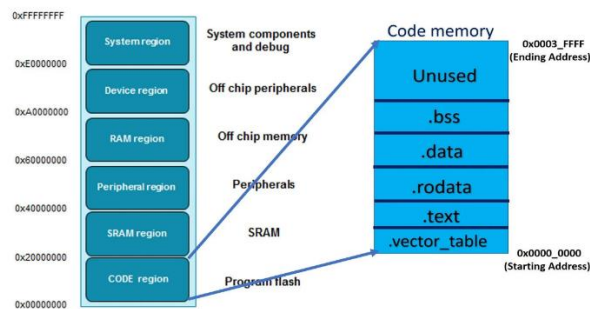


Figure 19: Relocating ISR Address from IVT

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 figure 20 shows the interrupt vector table of ARM Cortex M4 based TM4C123GH6PM microcontroller [4].

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		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 20: interrupt vector table of ARM Cortex M4

2.10 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 [4].

Let's take an example, suppose interrupt number 2 occurs. The NVIC used this formula to find the entry of corresponding IRQ x in IVT as shown in figure 21 [4].

$$\begin{aligned}\text{Entry in IVT} &= 64 + 4 * x ; \\ \text{for } x &= 2 \\ \text{Entry in IVT} &= 64 + 4 * 2 = 72 \\ &= 72 \text{ or } 0x0048\end{aligned}$$

Figure 21: Example

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 these locations to execute ISR [4].

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 in figure 22 shows the relocation of entry in IVT when bus fault exception occurs.

$$\begin{aligned}\text{Entry in IVT} &= 64 + 4 * x ; \\ \text{For bus fault exception } x &= -11 \\ \text{Entry in IVT} &= 64 + 4 * -11 = 20 \\ &= 20 \text{ or } 0x0014\end{aligned}$$

Figure 22: Relocation Of Entry In IVT

2.11 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 23, 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

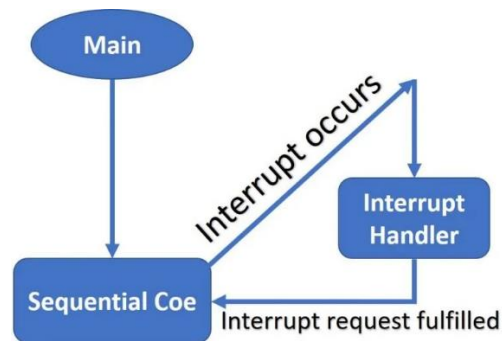


Figure 23: Steps Executed by ARM Cortex M

For example, the ARM cortex-M processor executing this 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.

2.12 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.

2.13 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.

2.14 TM4C123 Timer Interrupt Example Code¹

This example code of TM4C123 Tiva C Launchpad generates a delay of one second using the Timer1A interrupt handler routine.

```
*** Using Compiler 'V5.06 update 3 (build 300)', folder: 'C:\Keil_v5\ARM\ARMCC\Bin'
Build target 'Target 1'
compiling EXP888.c...
linking...
Program Size: Code=888 RO-data=636 RW-data=0 ZI-data=608
".\Objects\EXP888.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
```

Figure 24:Simulation Of Example

First, we include the device header file for Tiva Series Microcontroller, this is done by using **#include "TM4C123.h"**. Second, the blue LED is connected with the PF3 pin of PORT F, and this is done by using **#define Blue (1<<2)**.

In addition to that, in the main we turn on bus clock for GPIOF by using **SYSCTL->RCGCGPIO |= 0x20**. Then, we set blue pin as a digital output pin via this instruction **GPIOF->DIR |= Blue**. Also, we enable PF2 pin as a digital pin by using **GPIOF->DEN |= Blue**.

Then in the while loop do nothing just wait for the interrupt to occur. In the function **Time1A_1sec_delay** we do this:

- Enable clock Timer1 subtimer A in run mode by using **SYSCTL->RCGCTIMER |= (1<<1)**.
- Disable timer1 output by using **TIMER1->CTL = 0**.
- Select 16-bit configuration option by using **TIMER1->CFG = 0x4**.
- Select periodic down counter mode of timer1 by using **TIMER1->TAMR = 0x02**.
- We give TimerA prescaler value by using **TIMER1->TAPR = 250-1**.
- TimerA counter starting count down value by using **TIMER1->TAILR = 64000 - 1**.
- TimerA timeout flag bit clears by using **TIMER1->ICR = 0x1**.
- Enables TimerA time-out interrupt mask by using **TIMER1->IMR |= (1<<0)**.
- Enable TimerA module by using **TIMER1->CTL |= 0x01**.
- Enable IRQ21 by using **NVIC->ISER[0] |= (1<<21)**.

¹ The written code with comments can be found in the appendix.

2.15Lab Work

2.15.1 ²Task1

The code in this task is very similar to the previous example, but instead of using Blue LED we use GREEN Led blinks every **500ms**.

In this task TimerA counter starting count down value by using **TIMER1->TAILR = 32000-1** ,we use 3200 according to this equation in figure 25 .

The image shows a handwritten equation on lined paper:
$$\text{Timer Counter} = \frac{\text{Time}}{\frac{1}{\text{frequency}}} = \frac{500\text{ms}}{\frac{1}{64\text{M}}} = 32,000$$
 Below the equation, there are red annotations. An arrow points from the word "frequency" in the denominator to "16MHz" and "prescaler". Another arrow points from the "64M" in the denominator to "32bit" and "16MHz".

Figure 25:Equation For Task 1

```
Build Output
compiling EXP888.c...
EXP888.c(39): warning: #1-D: last line of file ends without a newline
}
EXP888.c: 1 warning, 0 errors
linking...
Program Size: Code=888 RO-data=636 RW-data=0 ZI-data=608
".\Objects\EXP888.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed: 00:00:01
```

Figure 26:Simulation for Task 1

² The written code with comments can be found in the appendix.

2.15.2 ³Task 2

The code in this task is very similar to the previous example, but instead of using Blue LED we use RED Led blinks every 4s.

In this task TimerA counter starting count down value by using **TIMER1->TAILR = 64000000-1**, we use 64000000 according to this equation in figure 26.

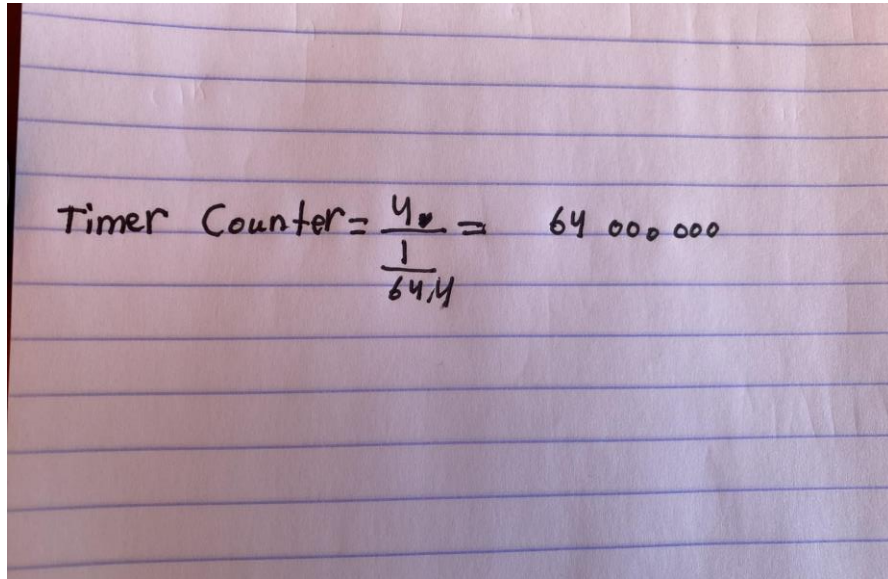

$$\text{Timer Counter} = \frac{40}{\frac{1}{64,4}} = 64\ 000\ 000$$

Figure 27: Equation For Task 2

```
Build Output
compiling EXP888.c...
EXP888.c(39): warning: #1-D: last line of file ends without a newline
}
EXP888.c: 1 warning, 0 errors
linking...
Program Size: Code=888 RO-data=636 RW-data=0 ZI-data=608
".\Objects\EXP888.axf" - 0 Error(s), 1 Warning(s).
Build Time Elapsed: 00:00:01
```

Figure 28: Simulation for Task 2

³ The written code with comments can be found in the appendix.

3. Conclusion

In this experiment, we learned about internal timers/counters, also how to configure the timers to make internal interrupt. In addition to that we manage to Implementation of Timer Interrupt Handler function then we write many codes to apply what we learned.

4.Feedback

The experiment was good and interesting , also the time was enough, so we finished the experiment early.

5. References

- [1] <https://microcontrollerslab.com/timer-interrupt-tm4c123-generate-delay-with-timer-interrupt-service-routine/> . Accessed on 19-08-2022 at 10:27AM.
- [2] http://shukra.cedt.iisc.ernet.in/edwiki/EmSys:TM4C123_Timer_Programming . Accessed on 19-08-2022 at 11:35AM.
- [3] <https://en.wikipedia.org/wiki/Prescaler> . Accessed on 19-08-2022 at 11:35AM.
- [4] <https://microcontrollerslab.com/interrupt-processing-arm-cortex-m-microcontrollers/> . Accessed on 19-08-2022 at 12:11PM.

6. Appendix

6.1 Example

// Name:Maha Maher Mali

//ID:1200746

/* 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 Blue (1<<2) // PF3 pin of TM4C123 Tiva Launchpad, Blue LED

void Time1A_1sec_delay(void);

int main(void)

{

 /*Initialize PF3 as a digital output pin */

 SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF

 GPIOF->DIR |= Blue; // set blue pin as a digital output pin

 GPIOF->DEN |= Blue; // Enable PF2 pin as a digital pin

 Time1A_1sec_delay();

 while(1)

 {

 // do nothing wait for the interrupt to occur

 }

}

```

/* Timer1 subtimer A interrupt service routine */

void TIMER1A_Handler()

{

    if(TIMER1->MIS & 0x1)

        GPIOF->DATA ^= Blue; /* 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 */

}

```

.....

6.2 Task 1

// ame:Maha Maher Mali

//ID:1200746

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

```
#define Green (1<<3) // PF3 pin of TM4C123 Tiva Launchpad, Green LED
```

```
void Time1A_1sec_delay(void);
```

```
int main(void)
```

```
{
```

```
    /*Initialize PF3 as a digital output pin */
```

```
    SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
```

```
    GPIOF->DIR      |= Green; // set green pin as a digital output pin
```

```
    GPIOF->DEN      |= Green; // Enable PF3 pin as a digital pin
```

```
    Time1A_1sec_delay();
```

```
    while(1)
```

```
    {
```

```
        // do nothing wait for the interrupt to occur
```

```
    }
```

```
}
```

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

```
void TIMER1A_Handler()
```

```
{
```

```

    if(TIMER1->MIS & 0x1)

        GPIOF->DATA ^= Green; /* toggle Green 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 = 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 */

}

```

.....

6.3 Task 2

// name:Maha Maher Mali

//ID:1200746

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

```
#define Red (1<<1) // PF1 pin of TM4C123 Tiva Launchpad, Red LED
```

```
void Time1A_1sec_delay(void);
```

```
int main(void)
```

```
{
```

```
    /*Initialize PF1 as a digital output pin */
```

```
    SYSCTL->RCGCGPIO |= 0x20; // turn on bus clock for GPIOF
```

```
    GPIOF->DIR      |= Red; // set red pin as a digital output pin
```

```
    GPIOF->DEN      |= Red; // Enable PF1 pin as a digital pin
```

```
    Time1A_1sec_delay();
```

```
    while(1)
```

```
    {
```

```
        // do nothing wait for the interrupt to occur
```

```
    }
```

```
}
```

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

```
TIMER1A_Handler()
```

```
{
```

```

    if(TIMER1->MIS & 0x1)

        GPIOF->DATA ^= Red; /* toggle Red 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 = 0x0; /*select 32-bit configuration option */

TIMER1->TAMR = 0x02; /*select periodic down counter mode of timer1 */

//TIMER1->TAPR = 250-1; /* TimerA prescaler value */

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 */

}

```

.....