



Faculty of Engineering & Technology
Electrical & Computer Engineering Department
COMPUTER DESIGN LABORATORY - ENCS 4110
Experiment#8: Timers Interrupts
Report#2

Prepared by: Rahaf Naser
ID: 1201319

Instructor: Dr. Abdelsalam sayyad
Teaching Assistant: Eng. Raha Zabadi

Section: 3
Date: 15/5/2024

Abstract

The aim of this experiment is to understand internal timers/counters and to learn how to setup nested vector interrupt controller (NVIC), in addition to understand how to configure the timers to make internal interrupt.

Table of contents

Abstract.....	I
List of tables.....	II
List of figures.....	III
1.Theory.....	1
1.1. General Purpose Timer Interrupt.....	1
1.2. Applications.....	1
1.3. How to configure Timer Interrupt of TM4C123.....	1
1.4. Types of Interrupt and Exceptions in ARM Cortex-M.....	2
1.6. Interrupt Vector Table and Interrupt Processing.....	2
1.7. Relocating ISR Address from IVT	3
1.8. Steps Executed by ARM Cortex M processor During Interrupt Processing.....	4
1.9. ARM Cortex-M Context Switching.....	4
1.10. Interrupts Processing Steps.....	4
2.Procedure and Results.....	6
2.1. Timer1A Interrupt with One Second Delay.....	7
2.2. Initialize Timer1A registers for one second Delay.....	7
2.4. Configure TM4C123 Timer Interrupt	9
2.5. Implementation of Timer Interrupt Handler function.....	10
2.7. TM4C123 Timer Interrupt Example Code.....	11
2.8. Lab Work	14
2.8.1. Task1.....	14
2.8.2. Task 2.....	15
2.8.3. Task3.....	16
3.Conclusion.....	17
4.References.....	18
5. Appendix.....	19

List of Figures

Figure1: TM4C123 microcontroller [1].....	1
Figure 2: Interrupt Vector Table and Interrupt Processing [1].....	2
Figure 3: Relocating ISR Address from IVT [1].....	3
Figure 4: interrupt vector table of ARM Cortex M4 [1].....	3
Figure 5: Steps Executed by ARM Cortex M [1].....	4
Figure 6: Select TM4C123GH6PM Microcontroller [1].....	6
Figure 7: Device Header File.....	6
Figure 8: connect Blue LED.....	7
Figure 9: Enable Cock.....	7
Figure 10: Disables The Timer1 Output.....	7
Figure 11: Select 16-bit Configuration.....	8
Figure 12: Select The Periodic Mode Of Timer1.....	8
Figure 13: Equation Of Maximum Dela.....	8
Figure 14: TimerA Prescaler.....	9
Figure 15: TimerA counter Starting count Down Value.....	9
Figure 16: Clear the Timeout Flag.....	9
Figure 17: Enable the TimerA module.....	9
Figure 18: Enable the Interrupt.....	10
Figure 19: Enable IRQ21.....	10
Figure 20: Interrupt number [1].....	10
Figure 21: Implementation of Timer Interrupt Handler function.....	11
Figure 22: Example.....	12
Figure 23: Relocation Of Entry In IVT.....	12
Figure 24: example1 result.....	13
Figure 25: task1 result.....	14
Figure 26: task2 result.....	15
Figure 27: task3 result.....	16

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. 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. [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].

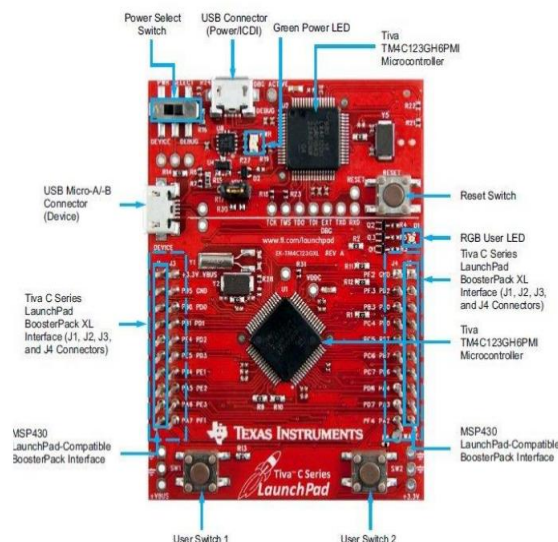


Figure1: TM4C123 microcontroller [1].

1.4. Types of Interrupt and Exceptions in ARM Cortex-M

Throughout this experiment, we 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, 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].

1.5. 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].

1.6. 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, 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 [1].

As shown in the Figure 2.

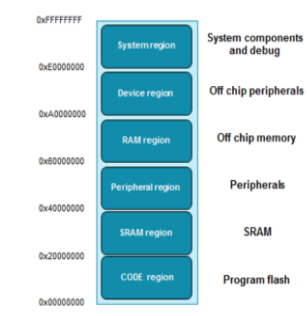


Figure 2: Interrupt Vector Table and Interrupt Processing [1].

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.

1.7. 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 3.

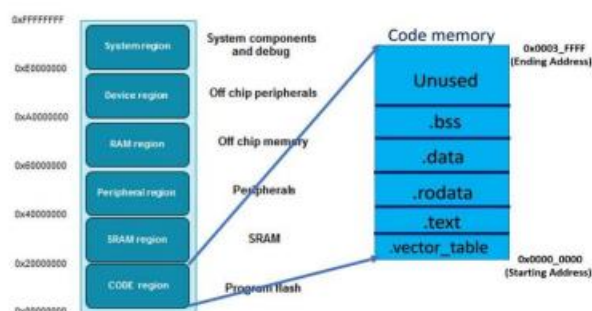


Figure 3: Relocating ISR Address from IVT [1].

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 4 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	-	-	-
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

Figure 4: interrupt vector table of ARM Cortex M4 [1].

1.8. Steps Executed by ARM Cortex M processor During Interrupt Processing

There are 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 Figure5, 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 [1].

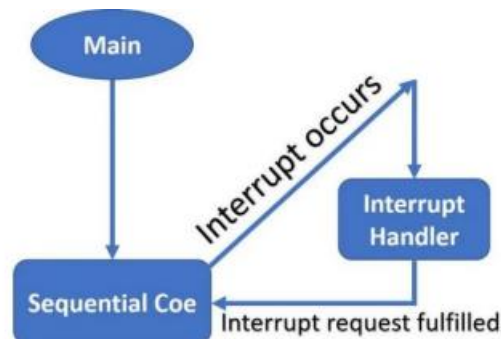


Figure 5: Steps Executed by ARM Cortex M [1].

1.9. 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 [1].

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 [1].

1.10. 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). 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 [1].

➤ 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 [1].

➤ 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 [1].

2.Procedure and Results

At First, we created a project in Keil uvision by selecting TM4C123GH6PM microcontroller as shown in Figure 6, include the header file of TM4C123GH6PM microcontroller which contains a register definition file of all peripherals such as timers [1].

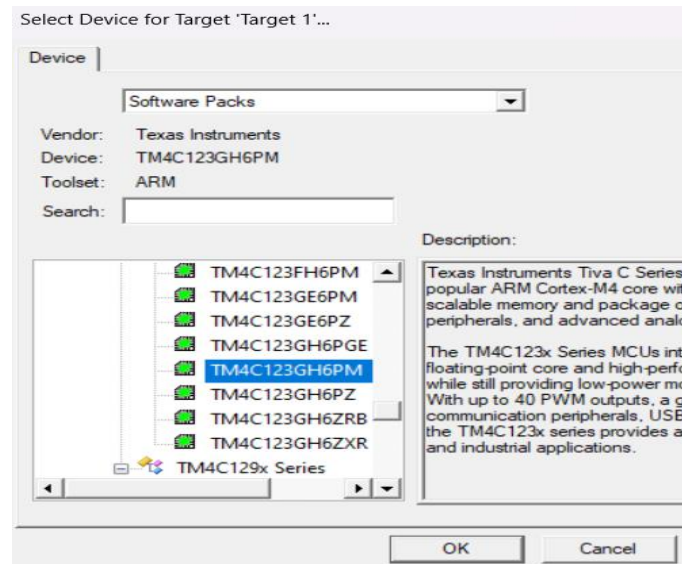


Figure 6: Select TM4C123GH6PM Microcontroller [1].

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

As shown in Figure 7 below.

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

Figure 7: Device Header File[1].

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 8 below.

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

Figure 8: connect Blue LED[1].

2.2. Initialize Timer1A registers for one second Delay

First, we 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 9 sets the bit1 of the RCGTIMER register to 1 [1].

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

Figure 9: Enable Cock[1].

Before initialization, disables the Timer1 output by clearing GPTMCTL register [1].

As shown in Figure 10.

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

Figure 10: Disables The Timer1 Output [1].

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 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 11.

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

Figure 11: Select 16-bit Configuration [1].

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, we set the GPTMTAMR register to 0x02 as shown in Figure 12, we do this according to datasheet [1].

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

Figure 12: Select The Periodic Mode Of Timer1 [1].

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 13 [1]:

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

Figure 13: Equation Of Maximum Dela [1].

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 14.

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

Figure 14: TimerA Prescaler [1].

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 15.

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

Figure 15: TimerA counter Starting count Down Value [1].

GPTMICR register is used to clear the timeout flag bit of timer[1]. As shown in Figure 16.

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

Figure 16: Clear the Timeout Flag [1].

After initialization and configuration, we enable the TimerA module which we disabled earlier[1]. As shown in Figure 17.

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

Figure 17: Enable the TimerA module [1].

2.4. Configure TM4C123 Timer Interrupt

There are two steps to enable the interrupt of peripherals in ARM Cortex-M microcontrollers. Firstly, enabling 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 18.

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

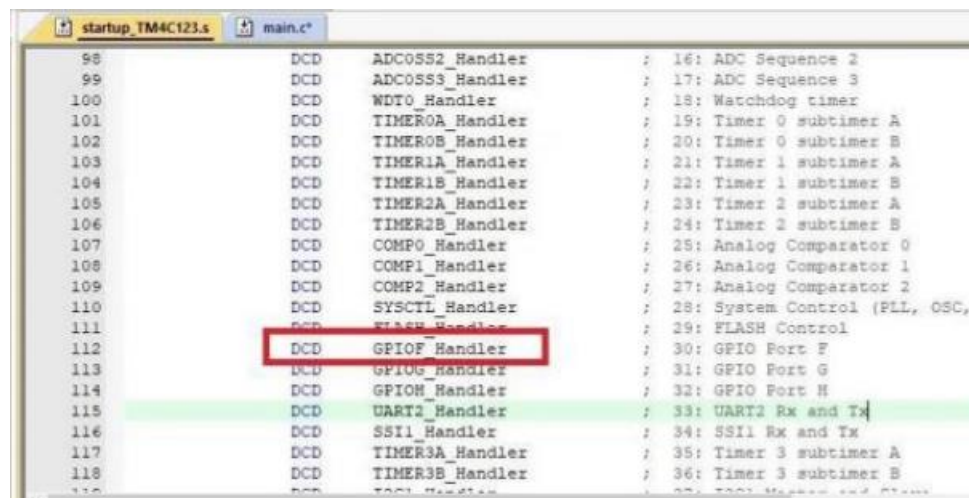
Figure 18: Enable the Interrupt [1].

Secondly, enabling 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 19.

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

Figure 19: Enable IRQ21 [1].

We also find the interrupt number of every exception handler or ISR inside the startup file of TM4C123 microcontroller as shown in Figure 20.



98	DCD	ADC0SS2_Handler	: 16: ADC Sequence 2
99	DCD	ADC0SS3_Handler	: 17: ADC Sequence 3
100	DCD	WDIO_Handler	: 18: Watchdog timer
101	DCD	TIMER0A_Handler	: 19: Timer 0 subtimer A
102	DCD	TIMER0B_Handler	: 20: Timer 0 subtimer B
103	DCD	TIMER1A_Handler	: 21: Timer 1 subtimer A
104	DCD	TIMER1B_Handler	: 22: Timer 1 subtimer B
105	DCD	TIMER2A_Handler	: 23: Timer 2 subtimer A
106	DCD	TIMER2B_Handler	: 24: Timer 2 subtimer B
107	DCD	COMP0_Handler	: 25: Analog Comparator 0
108	DCD	COMP1_Handler	: 26: Analog Comparator 1
109	DCD	COMP2_Handler	: 27: Analog Comparator 2
110	DCD	SYSCTL_Handler	: 28: System Control (PLL, OSC,
111	DCD	FLASH_Handler	: 29: FLASH Control
112	DCD	GPIOF_Handler	: 30: GPIO Port F
113	DCD	GPIOG_Handler	: 31: GPIO Port G
114	DCD	GPIOH_Handler	: 32: GPIO Port H
115	DCD	UART2_Handler	: 33: UART2 Rx and Tx
116	DCD	SSI1_Handler	: 34: SSI1 Rx and Tx
117	DCD	TIMER3A_Handler	: 35: Timer 3 subtimer A
118	DCD	TIMER3B_Handler	: 36: Timer 3 subtimer B
119	DCD	TIMER3C_Handler	: 37: Timer 3 subtimer C

Figure 20: Interrupt number [1].

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 21.

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

Figure 21: Implementation of Timer Interrupt Handler function [1].

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.

2.6. 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 22 [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 22: Example [1].

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 23 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 23: Relocation Of Entry In IVT [1].

2.7. TM4C123 Timer Interrupt Example Code

This example code sets up a timer interrupt on a TM4C123 Tiva C Launchpad microcontroller to generate a one-second delay. It configures Timer1 subtimer A to operate in periodic down counter mode, triggering an interrupt every second. When the interrupt occurs, it toggles the blue LED connected to pin PF3. The main function initializes PF3 as a digital output pin and then calls the function to set up the timer delay. Once initialized, the program enters an infinite loop, waiting for interrupts to occur.

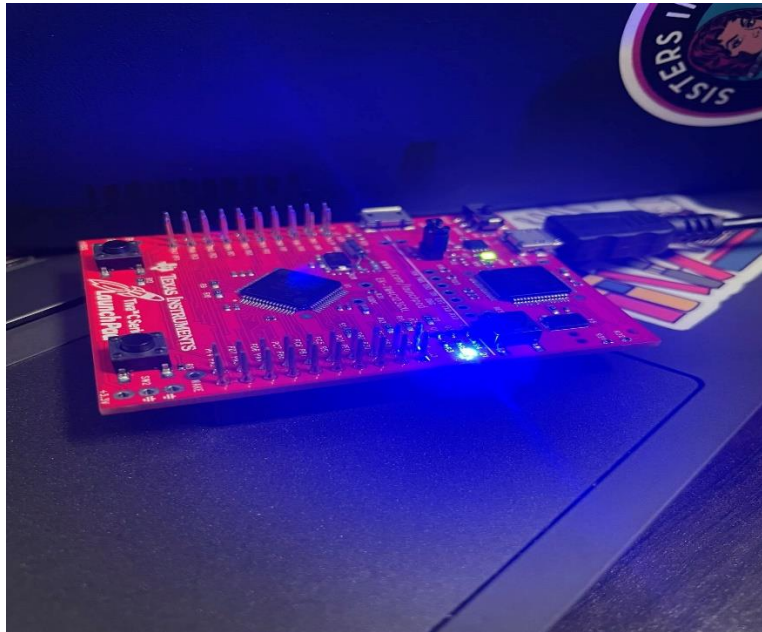


Figure 24: example1 result

Overall, this part of experiment demonstrates how to utilize hardware interrupts and timers to create precise timing intervals, showcasing a fundamental aspect of embedded systems programming.

2.8. Lab Work

2.8.1. Task1

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 the **main()** function, the green LED pin (connected to pin PF3) is initialized as a digital output. Then, the **Time1A_500ms_delay()** function is called to set up the timer interrupt for a 500ms delay. The interrupt service routine **TIMER1A_Handler()** toggles the green LED and clears the timer interrupt flag. Within the **Time1A_500ms_delay()** function, Timer1A is configured for a 500ms periodic down counter mode, with appropriate values for prescaler and counter starting count down.



Figure 25: task1 result

Overall, this code creates a precise 500ms blinking interval for the green LED, demonstrating how to utilize hardware interrupts for timing purposes in embedded systems.

2.8.2. Task 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 the **main()** function, the red LED pin (connected to pin PF1) is initialized as a digital output. Then, the **Time1A_4s_delay()** function is called to set up the timer interrupt for a 4-second delay.

The interrupt service routine **TIMER1A_Handler()** toggles the red LED and clears the timer interrupt flag.

Within the **Time1A_4s_delay()** function, Timer1A is configured for a 4-second periodic down counter mode, with appropriate values for prescaler and counter starting count down.

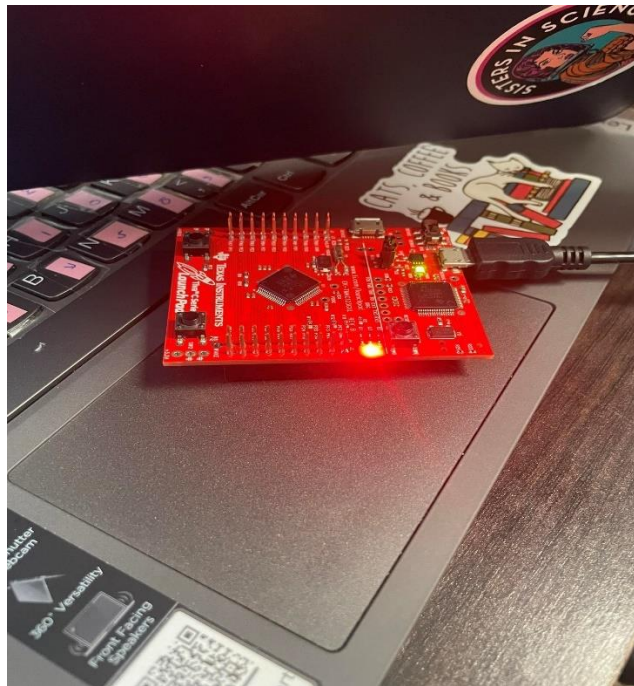


Figure 26 : task2 result

Overall, this code creates a precise 4-second blinking interval for the red LED, demonstrating how to utilize hardware interrupts for timing purposes in embedded systems.

2.8.3. Task3

The code in this task is to use the onboard LED and another two external LEDs with the TM4C123G board to make one LED flashes every 10 seconds, one flashes every 5 seconds, and one flashes every one second.

The code configures three different timers to control three LEDs on a TM4C123G board. The red LED connected to pin PF1 flashes every 10 seconds, the blue LED connected to pin PF2 flashes every 5 seconds, and the green LED connected to pin PF3 flashes every 1 second.

Each timer is set up with a periodic down counter mode to achieve the desired flashing intervals. Timer1 controls the red LED, Timer2 controls the blue LED, and Timer3 controls the green LED.

The interrupt service routines for each timer toggle the respective LED and clear the timer interrupt flag to maintain the timing accuracy.

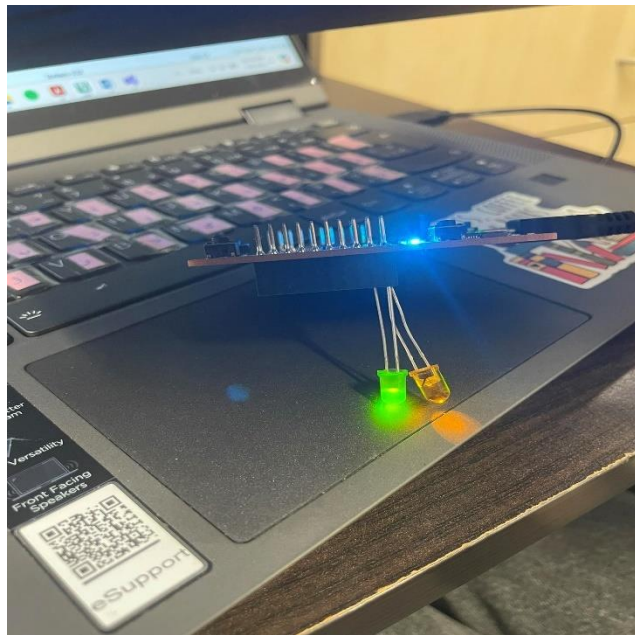


Figure 27: task3 result

Overall, this code effectively demonstrates how to utilize multiple timers to control different LED flashing intervals, showcasing the versatility and capability of the TM4C123G microcontroller.

3.Conclusion

In this experiment, we learned and understand internal timers/counters, also we learn how to setup nested vector interrupt controller (NVIC), in addition we understand how to configure the timers to make internal interrupt.

4. References

- [1] <https://microcontrollerslab.com/timer-interrupt-tm4c123-generate-delay-with-timer-interrupt-service-routine/> . [Accessed on 14 May 2024]
- [2] http://shukra.cedt.iisc.ernet.in/edwiki/Em/s:TM4C123_Timer_Programming .
[Accessed on 14 May 2024]
- [3] <https://en.wikipedia.org/wiki/Prescaler> . [Accessed on 15 May 2024]
- [4] <https://microcontrollerslab.com/interrupt-processing-arm-cortex-m-microcontrollers/> .
[Accessed on 15 May 2024]

5. Appendix

5.1. code of example

```
/* 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 */
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 */
}

```

5.2. code of labwork1

```

#include "TM4C123.h" // Device header file for Tiva Series Microcontroller
#define Green (1 << 3) // PF3 pin of TM4C123 Tiva Launchpad, Green LED
void Time1A_500ms_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_500ms_delay();

    while (1)
    {
        // Do nothing, wait for the interrupt to occur
    }
}

```



```

}

/* Timer1 subtimer A interrupt service routine */
void TIMER1A_Handler(void)
{
    if (TIMER1->MIS & 0x1)
        GPIOF->DATA ^= Green; /* Toggle Green LED */
    TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears */
}

void Time1A_500ms_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 for 500ms */
    TIMER1->TAILR = 8000000 - 1; /* TimerA counter starting count down value for 500ms */
    TIMER1->ICR = 0x1; /* TimerA timeout flag bit clears */
    TIMER1->IMR |= (1 << 0); /* Enable TimerA time-out interrupt mask */
    TIMER1->CTL |= 0x01; /* Enable TimerA module */
    NVIC->ISER[0] |= (1 << 21); /* Enable IRQ21 */
}

```

5.3. code of labwork2

```

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

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

void Time1A_4s_delay(void);

```

```

int main(void)
{
    /* Initialize PF1 as a digital output pin */
    SYSCTL->RCGCGPIO |= (1 << 5); // 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_4s_delay();

    while (1)
    {
        // Do nothing, wait for the interrupt to occur
    }
}

/* Timer1 subtimer A interrupt service routine */
void TIMER1A_Handler(void)
{
    if (TIMER1->MIS & 0x1)
        GPIOF->DATA ^= Red; /* Toggle Red LED */
    TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears */
}

void Time1A_4s_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 for 4s */
TIMER1->TAILR = 64000000 - 1; /* TimerA counter starting count down value for 4s */
TIMER1->ICR = 0x1;           /* TimerA timeout flag bit clears */
TIMER1->IMR |= (1 << 0);     /* Enable TimerA time-out interrupt mask */
TIMER1->CTL |= 0x01;         /* Enable TimerA module */

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

```

5.4. code for labwork3

```

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

#define Red (1<<1) // PF3 pin of TM4C123 Tiva Launchpad, Blue LED
#define Blue (1<<2) // PF3 pin of TM4C123 Tiva Launchpad, Blue LED
#define Green (1<<3) // 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 |= Red; // set blue pin as a digital output pin
    GPIOF->DEN |= Red; // Enable PF2 pin as a digital pin
    GPIOF->DIR |= Blue; // set blue pin as a digital output pin
    GPIOF->DEN |= Blue; // Enable PF2 pin as a digital pin
    GPIOF->DIR |= Green; // set blue pin as a digital output pin
    GPIOF->DEN |= Green; // 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 */
TIMER1A_Handler() {
    if(TIMER1->MIS & 0x1)
        GPIOF->DATA ^= Red; /* toggle Blue LED*/
        TIMER1->ICR = 0x1; /* Timer1A timeout flag bit clears*/
}

TIMER2A_Handler() {
    if(TIMER2->MIS & 0x1)
        GPIOF->DATA ^= Blue; /* toggle Blue LED*/
        TIMER2->ICR = 0x1; /* Timer1A timeout flag bit clears*/
}

TIMER3A_Handler() {
    if(TIMER3->MIS & 0x1)
        GPIOF->DATA ^= Green; /* toggle Blue LED*/
        TIMER3->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 */
    SYSCTL->RCGCTIMER |= (1<<2); /*enable clock Timer1 subtimer A in run mode */
    TIMER2->CTL = 0; /* disable timer1 output */
    TIMER2->CFG = 0x0; /*select 32-bit configuration option */
    TIMER2->TAMR = 0x02; /*select periodic down counter mode of timer1 */
    SYSCTL->RCGCTIMER |= (1<<3); /*enable clock Timer1 subtimer A in run mode */
    TIMER3->CTL = 0; /* disable timer1 output */
}

```

```

TIMER3->CFG = 0x0; /*select 32-bit configuration option */
TIMER3->TAMR = 0x02; /*select periodic down counter mode of timer1 */
//TIMER1->TAPR = 250-1; /* TimerA prescaler value */
TIMER1->TAILR = 16000000-1 ; /* 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 */
TIMER2->TAILR = 80000000-1 ; /* TimerA counter starting count down value */
TIMER2->ICR = 0x1; /* TimerA timeout flag bit clears*/
TIMER2->IMR |= (1<<0); /*enables TimerA time-out interrupt mask */
TIMER2->CTL |= 0x01; /* Enable TimerA module */
TIMER3->TAILR = 160000000-1 ; /* TimerA counter starting count down value */
TIMER3->ICR = 0x1; /* TimerA timeout flag bit clears*/
TIMER3->IMR |= (1<<0); /*enables TimerA time-out interrupt mask */
TIMER3->CTL |= 0x01; /* Enable TimerA module */
NVIC->ISER[0] |= (1<<21); /*enable IRQ21 */
NVIC->ISER[0] |= (1<<23); /*enable IRQ21 */
NVIC->ISER[1] |= (1<<3); /*enable IRQ21 */
}

```