Faculty of Engineering and Technology
Electrical and Computer Engineering Department

COMPUTER DESIGN LABORATORY
ENCS 4110

Experiment No. 8:

Software Interrupts (Timer Interrupts)

Prepared by:
Islam Zayed 1230007

Instructor: Dr. Abualseoud Hanani
Teaching Assistant: Eng. Mohammad Abu Shelbaia

Section : 1
Date: August 10, 2025

Abstract

This experiment investigated the configuration and operation of the TM4C123GH6PM microcontroller's General-Purpose Timer Module (GPTM) to produce hardware-based periodic interrupts for precise LED control. Through practical experimentation and applications, we may explore how the GPTM, capable of one-shot or periodic modes, was programmed in both 16-bit mode with prescalers and 32-bit mode without prescalers to achieve a range of blink intervals. We computed the reload values based on the 50 MHz system clock of the microcontroller, allowing timers to operate both independently and accurately without relying on inefficient software delays. The results demonstrated accurate and interference-free LED blinking, validating that hardware timers provide superior timing precision while reducing CPU overhead. After analyzing our obtained results and key findings, we concluded that the GPTM is indeed highly suitable for embedded systems requiring real-time responsiveness and multitasking capabilities. The experiment reinforced my understanding of timer register configuration, prescaler operation, and interrupt handling, while further developing and enforcing my practical skills essential for designing responsive, resource-efficient embedded applications where precise timing control is essential.

Table of Contents

List of Tables

List of Listings

1. Theory

1.1 GPTM (General-Purpose Timer Modules) :

"The TM4C123GH6PM microcontroller includes programmable General-Purpose Timer Modules (GPTM) that support various timing and counting functions. The module offers both 16/32-bit and 32/64-bit "Wide" timers with different capabilities and resolutions."[3]

Table 1 : Comparison of Standard vs. Wide GPTM Configurations in TM4C123 Microcontroller

Feature	**16/32-bit GPTM Blocks**	**32/64-bit Wide GPTM Blocks**
Timer Width	16-bit timers (Timer A & B) or combined 32-bit timer	32-bit timers (can be combined as 64-bit timer)
Prescaler Resolution	8-bit prescaler for 16-bit timers	16-bit prescaler for 32-bit timers
Timer Count Range	Up to 65,535 (16-bit)	Up to 4,294,967,295 (32-bit)

1.1.1 Operating Modes of GPTM :

- **One-Shot Mode**: Timer runs once and stops automatically.
- **Periodic Mode:** Timer repeats continuously, restarting after each cycle.
- **Other Modes:** Various modes for specific applications, including capture and PWM provided in the datasheet. [1]

1.2 Clock Source and System Clock Configuration :

The TM4C123 timer can use either the system clock or an external clock source. Typically, the MCU uses a 16 MHz external crystal, but the system clock is configured via the internal PLL and clock dividers to run at different speeds. In Keil uVision projects, the system clock commonly runs at 50 MHz using the 16 MHz crystal and PLL. [2]

1.3 Time Configuration :

1.3.1 Enable the Clocks to Peripherals :

Peripheral clocks must be enabled for both GPIO ports and timers:

```
SYSCTL->RCGCGPIO |= (1 << 5);    // Enable clock for GPIO Port F
SYSCTL->RCGCTIMER |= (1 << 1);   // Enable clock for Timer1
```

- `RCGCGPIO` controls clocks to GPIO ports; bit 5 corresponds to Port F.
- `RCGCTIMER` controls clocks to timers; bit 1 corresponds to Timer1.
- A simple delay loop follows, ensuring the clock to stabilize after enabling them.[3]

1.3.2 Configure GPIO Pins for Output :

Set the pins connected to LEDs as outputs and enable their digital functionality:

```
GPIOF->DIR |= 0x04;   // Set PF2 (Blue LED) as output
GPIOF->DEN |= 0x04;   // Enable digital function on PF2
```

1.3.3 Disable Timer Before Configuration:

Turn off the timer to prevent unwanted operation during setup:

```
TIMER1->CTL = 0;   // Disable Timer1A
```

1.3.4 Set Timer Configuration:

Choose between 16-bit and 32-bit modes and select the timer operation mode (refer to Table 2 for time register values) :

```
TIMER1->CFG = 0x4;    // 16-bit timer mode
TIMER1->TAMR = 0x2;   // Periodic mode
```

1.3.5 Configure Prescaler and Reload Value:

- The **prescaler** divides the input clock frequency to slow down timer counting.
- The **reload value** (`ILR`) sets the count duration.

Prescaler and reload values define the interrupt frequency as in the 50 MHz clocked system below:

```
TIMER1->TAPR = 255;       //Prescaler value (max for 8-bit)
TIMER1->TAILR = 65535;    //Reload value (max for 16-bit)
```

- Interrupt frequency formula for 16 bit timer : *Interrupt Frequency = System Clock/(Prescaler * Reload Value)*
- Use 32 bit timer for longer delays : *Interrupt Frequency = System Clock/Reload Value*

```
TIMER1->CFG = 0x0;           // 32-bit mode
TIMER1->TAMR = 0x2;          // Periodic mode
TIMER1->TAILR = 50000000 - 1; // 1-second interval
```

1.3.6 Clear Interrupt Flags and Enable Interrupts:

Before enabling interrupts, clear any existing interrupt flags to avoid immediate triggers.

```
TIMER1->ICR = 0x1;    // Clear timeout interrupt flag
TIMER1->IMR |= 0x1;   // Enable timeout interrupt
```

1.3.7 Enable Timer and NVIC Interrupt:

Turn the timer on and enable the interrupt in the microcontroller's interrupt controller.:

```
TIMER1->CTL |= 0x1;             // Enable Timer1A
NVIC->ISER[0] |= (1 << 21);   // Enable Timer1A interrupt in NVIC
```

1.4 Time Registers :

Table 2 : Timer Registers of TM4C123 and Their Functions

Register	Purpose	Example Values
`CFG`	Timer configuration: 16-bit (`0x4`) or 32-bit (`0x0`)	`0x4` for 16-bit, `0x0` for 32-bit
`TAMR`	Timer A Mode Register, sets the timer mode (`0x1`: One-Shot, `0x2`: Periodic).	`0x2` for Periodic mode
`TAPR`	Timer A Prescale Register, sets the prescaler value.	`255` for 8-bit prescaler max
`TAILR`	Timer A Interval Load Register, sets the reload value.	`65535` for 16-bit max or `50000000 - 1` for 1 sec at 50 MHz
`ICR`	Interrupt Clear Register, clears the interrupt	Write `1` to clear timeout flag
`IMR`	Interrupt Mask Register, enables/disables interrupts. (`0x1`: Timeout)	Set bit `0` to enable timeout interrupt
`MIS`	Masked Interrupt Status Register, shows which interrupts are active.	Read to check if interrupt occurred
`CTL`	Control register for enabling/disabling the timer. (`0x1`: Enable, `0x0`: Disable)	Set bit `0` to enable timer
`NVIC->ISER`	Interrupt Set-Enable Register, enables specific interrupts in the NVIC.	Set bit `21` for Timer1A interrupt

2. Procedure and Discussion :

2.1. Example Using Maximum 16 - bit Delay :

2.1.1 Code :

Listing 2.1: C program to toggle blue LED using Timer periodic interrupts for the first example

```
#include "TM4C123.h"

#define RED 0x02
#define BLUE 0x04
#define GREEN 0x08
#define YELLOW RED + GREEN
#define MAGENTA BLUE + RED
#define CYAN GREEN + BLUE
#define WHITE RED + GREEN + BLUE
#define SW1 0x10
#define SW2 0x01
#define DELAY 900000



const int sequence[] = {RED, BLUE, GREEN, YELLOW, MAGENTA, CYAN,
WHITE};
int index = 0;

void delay( volatile unsigned long ulLoop ){
      for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
            for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
            }
      }
}

int main(void)
{
      SYSCTL->RCGCGPIO |= (1<<5);
      SYSCTL->RCGCTIMER |= (1<<1);
      delay(0);
```

```
      GPIOF->DIR |= BLUE;
      GPIOF->DEN |= BLUE;

      TIMER1->CTL = 0;                  // Disable the timer
      TIMER1->CFG = 0x4;                // Choose 16-bit mode
      TIMER1->TAMR = 0x02;              // Periodic mode
      TIMER1->TAPR = 256  - 1;    // Prescaler
      TIMER1->TAILR = 65536 - 1;  // Initial Value
      TIMER1->ICR = 0x1;                // Clear Any Prior Interrupts
      TIMER1->IMR |=(1<<0);             // Enable Timeout Interrupt
      TIMER1->CTL |= 0x01;              // Enable the timer
      NVIC->ISER[0] |= (1<<21);
    while(1)
    {
    }
}


void TIMER1A_Handler()
{
      if(TIMER1->MIS & 0x1)
            GPIOF->DATA  ^= BLUE;
      TIMER1->ICR = 0x1;
}
```

2.1.2 Results Discussion :

To generate a periodic LED blink without software delays, I configured the microcontroller's GPTM in 16-bit periodic mode with the maximum prescaler of 255 and reload value ,65535. This produced the longest delay possible in 16-bit mode,about 3.35 seconds,while keeping timing precise. The TIMER1A interrupt was enabled in the NVIC, and its ISR toggled the blue LED on each timeout, ensuring non-blocking operation. The LED blinked consistently at the expected interval, confirming correct prescaler and reload calculations. These results had demonstrated both the GPTM's accuracy and efficiency.

2.2 Lab Work Exercises :

2.2.1.1 Lab Work Exercise 1 :

Modify the code above (in Listing 2.1) to make the GREEN LED blinks every 100ms.

2.2.1.2 Code :

Listing 2.2: Solution to Exercise 1 in embedded C programming

```
#include "TM4C123.h"
#define RED 0x02
#define BLUE 0x04
#define GREEN 0x08
#define YELLOW RED + GREEN
#define MAGENTA BLUE + RED
#define CYAN GREEN + BLUE
#define WHITE RED + GREEN + BLUE
#define SW1 0x10
#define SW2 0x01
#define DELAY 900000



const int sequence[] = {RED, BLUE, GREEN, YELLOW, MAGENTA, CYAN,
WHITE};
int index = 0;

void delay( volatile unsigned long ulLoop ){
      for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
            for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
            }
      }
}

int main(void)
{
      SYSCTL->RCGCGPIO |= (1<<5);
      SYSCTL->RCGCTIMER |= (1<<1);
```

```
      delay(0);
      GPIOF->DIR |= GREEN;
      GPIOF->DEN |= GREEN;

      TIMER1->CTL = 0;                  // Disable the timer
      TIMER1->CFG = 0x4;                // Choose 16-bit mode
      TIMER1->TAMR = 0x02;              // Periodic mode
      TIMER1->TAPR = 256  - 1;    // Prescaler
      TIMER1->TAILR = 19531 - 1;  // Initial Value
      TIMER1->ICR = 0x1;                // Clear Any Prior Interrupts
      TIMER1->IMR |=(1<<0);             // Enable Timeout Interrupt
      TIMER1->CTL |= 0x01;              // Enable the timer
      NVIC->ISER[0] |= (1<<21);
    while(1)
    {
    }
}


void TIMER1A_Handler()
{
      if(TIMER1->MIS & 0x1)
            GPIOF->DATA  ^= GREEN;
      TIMER1->ICR = 0x1;
}
```

2.2.1.3 Results Discussion :

Solving the above lab work exercise to make the green LED blink every 100 ms, I configured the microcontroller's GPTM in 16-bit periodic mode. The prescaler was set to its maximum of 255 to slow the base timer clock, and calculated the reload value as 19531 for a 100 ms interval at a 50 MHz system clock. TIMER1A interrupts were enabled in the NVIC, and the ISR toggled the green LED at each timeout. The LED blinked at the expected frequency with consistent timing, confirming that my prescaler and reload calculations computed were correct. Our results verified the GPTM's reliability for short, precise timing tasks without using software delays.

2.2.2.1 Lab Work Exercise 2 :

Modify the code above (in Listing 2.1) to make the RED LED blinks every 4s.

2.2.2.2 Code :

Listing 2.3: Solution to Exercise 2 in embedded C programming

```
#include "TM4C123.h"
#define RED 0x02
#define BLUE 0x04
#define GREEN 0x08
#define YELLOW RED + GREEN
#define MAGENTA BLUE + RED
#define CYAN GREEN + BLUE
#define WHITE RED + GREEN + BLUE
#define SW1 0x10
#define SW2 0x01
#define DELAY 900000



const int sequence[] = {RED, BLUE, GREEN, YELLOW, MAGENTA, CYAN,
WHITE};
int index = 0;

void delay( volatile unsigned long ulLoop ){
      for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
            for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
            }
      }
}

int main(void)
{
      SYSCTL->RCGCGPIO |= (1<<5);
      SYSCTL->RCGCTIMER |= (1<<1);
      delay(0);
      GPIOF->DIR |= RED;
      GPIOF->DEN |= RED;
```

```
     TIMER1->CTL = 0;                  // Disable the timer
     TIMER1->CFG = 0;                  // Choose 16-bit mode
     TIMER1->TAMR = 0x02;              // Periodic mode
     TIMER1->TAILR = 50000000*4 - 1;   // Initial Value
     TIMER1->ICR = 0x1;                // Clear Any Prior Interrupts
     TIMER1->IMR |=(1<<0);             // Enable Timeout Interrupt
     TIMER1->CTL |= 0x01;              // Enable the timer
     NVIC->ISER[0] |= (1<<21);
    while(1)
    {
    }
}

void TIMER1A_Handler()
{
     if(TIMER1->MIS & 0x1)
          GPIOF->DATA  ^= RED;
     TIMER1->ICR = 0x1;
}
```

2.2.2.3 Results Discussion :

Modifying the code in Listing 2.1 so that the red LED blinks every 4 seconds, I configured the microcontroller's GPTM in 32-bit periodic mode by setting the configuration register to 0x0. This was necessary as a 16 bit timer cannot provide time periods over 1 sec. Using a 50 MHz system clock, I computed the reload value as 200,000,000 ticks (50,000,000 × 4) , in alignment to the desired time period. The timer interrupt was enabled through the NVIC, and the ISR toggled the red LED at each timeout. The LED blinked precisely at the intended 4-second interval, confirming that the 32-bit timer mode provides accurate long delays without needing a prescaler. Our results demonstrated the suitability of GPTM for tasks requiring extended timing intervals while maintaining CPU efficiency and real-time responsiveness.

2.2.3.1 Lab Work Exercise 3 :

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.

2.2.3.2 Code :

Listing 2.4: Solution to Exercise 3 in embedded C programming

```
#include "TM4C123.h"
#define RED 0x02
#define BLUE 0x04
#define GREEN 0x08
#define YELLOW RED + GREEN
#define MAGENTA BLUE + RED
#define CYAN GREEN + BLUE
#define WHITE RED + GREEN + BLUE
#define SW1 0x10
#define SW2 0x01
#define DELAY 900000
#define SRC_CLK 50000000

const int sequence[] = {RED, BLUE, GREEN, YELLOW, MAGENTA, CYAN,
WHITE};
int index = 0;
void delay( volatile unsigned long ulLoop ){
    for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
        for (ulLoop = 0; ulLoop < DELAY; ulLoop++) {
          }
     }
}
int main(void)
{
        SYSCTL->RCGCGPIO |= (1<<5);
        SYSCTL->RCGCTIMER |= (1<<1) | (1<<0) | (1<<2);
        delay(0);
        GPIOF->DIR |= WHITE;
        GPIOF->DEN |= WHITE;
```

```
        TIMER0->CTL = 0;                // Disable the timer
        TIMER0->CFG = 0x0;               // Choose 32-bit mode
        TIMER0->TAMR = 0x02;               // Periodic mode
        TIMER0->TAILR = 1*SRC_CLK - 1;  // Initial Value
        TIMER0->ICR = 0x1;               // Clear Any Prior
Interrupts
        TIMER0->IMR |=(1<<0);            // Enable Timeout Interrupt
        TIMER0->CTL |= 0x01;              // Enable the timer


        TIMER1->CTL = 0;                // Disable the timer
        TIMER1->CFG = 0x0;               // Choose 32-bit mode
        TIMER1->TAMR = 0x02;               // Periodic mode
        TIMER1->TAILR = 5*SRC_CLK - 1;  // Initial Value
        TIMER1->ICR = 0x1;               // Clear Any Prior
Interrupts
        TIMER1->IMR |=(1<<0);            // Enable Timeout Interrupt
        TIMER1->CTL |= 0x01;              // Enable the timer

        TIMER2->CTL = 0;                // Disable the timer
        TIMER2->CFG = 0x0;               // Choose 32-bit mode
        TIMER2->TAMR = 0x02;               // Periodic mode
        TIMER2->TAILR = 10*SRC_CLK - 1;  // Initial Value
        TIMER2->ICR = 0x1;               // Clear Any Prior
Interrupts
        TIMER2->IMR |=(1<<0);            // Enable Timeout Interrupt
        TIMER2->CTL |= 0x01;              // Enable the timer

        NVIC->ISER[0] |= (1<<TIMER0A_IRQn);
        NVIC->ISER[0] |= (1<<TIMER1A_IRQn);
        NVIC->ISER[0] |= (1<<TIMER2A_IRQn);
    while(1)
    {
    }
}
void TIMER0A_Handler()
{
```

```
    if(TIMER0->MIS & 0x1)
        GPIOF->DATA  ^= BLUE;
     TIMER0->ICR = 0x1;
}
void TIMER1A_Handler()
{
    if(TIMER1->MIS & 0x1)
        GPIOF->DATA  ^= GREEN;
     TIMER1->ICR = 0x1;
}
void TIMER2A_Handler()
{
    if(TIMER2->MIS & 0x1)
        GPIOF->DATA  ^= RED;
     TIMER2->ICR = 0x1;
}
```

2.2.3.3 Results Discussion :

To achieve simultaneous control of three LEDs at different blink rates (1 s, 5 s, and 10 s), the microcontroller's GPTM was configured with three independent 32-bit timers (TIMER0, TIMER1, TIMER2) in periodic mode. Using the 50 MHz system clock, reload values were calculated as *(Period × Clock Frequency)* for each timer to match the corresponding intervals precisely. Each timer was assigned its own interrupt service routine to toggle a specific LED color, blue for 1 s, green for 5 s, and red for 10 s,while ensuring no interference between tasks. The experiment confirmed that multiple hardware timers can run in parallel, each maintaining accurate timing without introducing software delays. This method significantly reduced CPU overhead, improved responsiveness, and demonstrated the GPTM's effectiveness for multitasking scenarios in real-time embedded systems.

2.3 Lab Task (To-Do) :

2.3.1 To-Do :

Implement an LED blinking program that cycles through multiple colors using two switches, where one switch changes the LED color and the other switch changes the blink speed.

2.3.2 Code :

Listing 2.5: Solution to the To-Do in embedded C programming

```
#include "TM4C123.h"
#include "utils.h"

#define SRC_CLK 50000000

volatile int index = 0;

int seq[] = {RED, BLUE, GREEN, YELLOW, MAGENTA, CYAN, WHITE};
int main(void)
{
      SYSCTL->RCGCGPIO |= (1<<5);
      SYSCTL->RCGCTIMER |= 0x7; // 111 -> 0X7 1110 -> 0XE
      delayMs(30);

          GPIOF->LOCK = 0x4C4F434B;   /* unlock commit register */
    GPIOF->CR = 0x01;           /* make PORTF0 configurable */
    GPIOF->LOCK = 0;            /* lock commit register */

      GPIOF->DIR |= WHITE;
      GPIOF->DEN |= WHITE + SWITCHES;
      GPIOF->PUR |= SWITCHES;

          GPIOF->IS  &= ~(1<<4)|~(1<<0);        /* make bit 4, 0 edge
sensitive */
    GPIOF->IBE &=~(1<<4)|~(1<<0);         /* trigger is controlled by
IEV */
    GPIOF->IEV &= ~(1<<4)|~(1<<0);        /* falling edge trigger */
    GPIOF->ICR |= (1<<4)|(1<<0);          /* clear any prior
```

```
interrupt */
    GPIOF->IM  |= (1<<4)|(1<<0);          /* unmask interrupt */

     TIMER0->CTL = 0;                  // Disable the timer
     TIMER0->CFG = 0x0;                // Choose 16-bit mode
     TIMER0->TAMR = 0x02;              // Periodic mode
     TIMER0->TAILR = 0.5f * 50000000;
     TIMER0->ICR = 0x1;                // Clear Any Prior Interrupts
     TIMER0->IMR |=(1<<0);             // Enable Timeout Interrupt
     TIMER0->CTL |= 0x01;              // Enable the timer
     NVIC->ISER[0] = 1<<TIMER0A_IRQn;
     NVIC->ISER[0] = 1<<GPIOF_IRQn;
    while(1)
    {
    }
}

void TIMER0A_Handler()
{
     static int flag = 0;
     if(TIMER0->MIS & 0x1){
          if (flag){
               GPIOF->DATA  = seq[index];
          } else {
               GPIOF->DATA  &= ~WHITE;
          }
          flag = 1 -  flag;
     }
     TIMER0->ICR = 0x1;
}

void GPIOF_Handler(void) {
     static int flag = 0;
          delayMs(30);
    if (GPIOF->MIS & 0x10) { /* check if interrupt causes by PF4
SW1*/
                        index = (index + 1) %7;
                GPIOF->DATA  = seq[index];
        GPIOF->ICR |= 0x10; /* clear the interrupt flag */
```

```
    }
    else if (GPIOF->MIS & 0x01) {  /* check if interrupt causes by
PF0/SW2 */
                            if (flag == 0) {
                                  TIMER0->TAILR = 0.5f * 50000000;
                            }else {
                                  TIMER0->TAILR = 2 * 50000000;
                            }
                            flag = 1 - flag;
        GPIOF->ICR |= 0x01; /* clear the interrupt flag */
    }
}
```

2.3.3 Results Discussion :

The program provided above in Listing 2.5 initially solves the problem by configuring TIMER0 in periodic mode to blink LEDs at a set interval, initially 0.5 s, we had calculated from the 50 MHz clock. The seq [ ] array stores predefined LED color patterns, and a flag in the TIMER0A_Handler toggles between LED ON and OFF states to create the blinking effect. Two pushbuttons ,SW1 and SW2,were enabled through GPIO edge-triggered interrupts,where SW1 cycles through the LED color sequence, and SW2 toggles the blink period between 0.5 s and 2 s by updating the timer reload value in real time. Our interrupt-driven approach had removed the need for blocking delays, allowing responsive user input handling while maintaining accurate timing. The experiment confirmed that integrating GPTM with GPIO interrupts enables both precise periodic events and responsive control logic.

Conclusion:

After completing the experiment on configuring the TM4C123GH6PM's GPTM, General-Purpose Timer Module for hardware-based periodic interrupts, I am confident in setting up timers in both 16-bit and 32-bit modes to achieve precise timing control. By adjusting prescalers and reload values, I was able to successfully program multiple LEDs to blink at intervals of different durations without relying on inefficient software delays. I had observed that smaller time periods, typically in milliseconds, should utilize a 16-bit timer, whereas those over one second require a 32-bit register. Thirty - two bit timers had indeed provided longer,more precise delays than 16-bit timers. This practical implementation of timer registers demonstrated how interrupt-driven design minimizes CPU overhead while ensuring consistent timing regardless of other tasks running in the system. The experiment deepened my understanding of timer register configurations and reinforced the advantages of using hardware timers and interrupts over other methods such as polling. All in all, I developed stronger practical skills and greater intuition for designing responsive, real-time embedded systems where both accurate event scheduling and resource efficiency are crucial.

References :

[1]:Manual for Computer Design Lab, 2025, Birzeit University.

[2]:https://microcontrollerslab.com/timer-interrupt-tm4c123-generate-delay-with-timer-interrupt-service-routine/ [Accessed on August 7 , 2025]

[3]: https://mabushelbaia.com/encs4110/, [Accessed on August 8 ,2025]

Appendices: