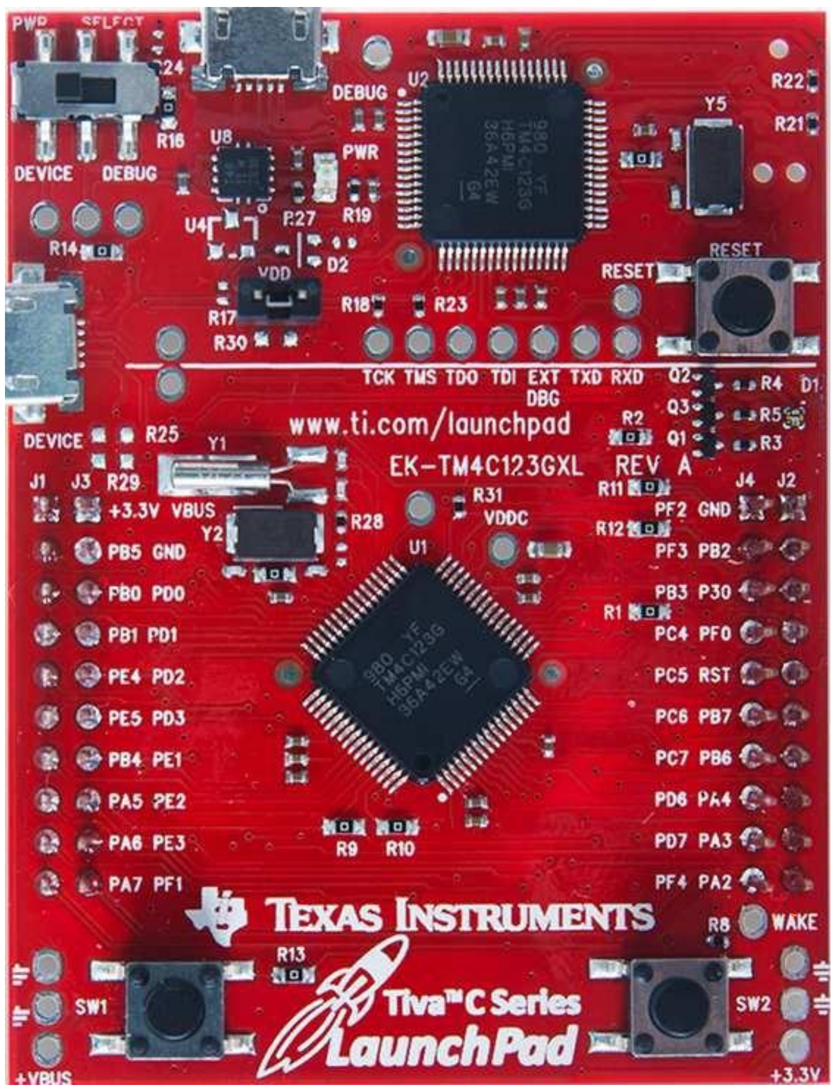
Embedded System Design using TM4C LaunchPad™ Development Kit



TEXAS INSTRUMENTS

Experiment 6 Interrupt Programming with GPIO

Topics **Page**

6.1 Objective

The main objective of this experiment is to toggle an LED by configuring the timer interrupt of the TM4C123GH6PM microcontroller. This experiment will help to understand the Timer Interrupt and its configuration.

6.2 Introduction

In this experiment, the timer peripheral of the TM4C123GH6PM processor is configured in periodic mode. In periodic mode, the timer load register is loaded with a preset value and the timer count is decremented for each clock cycle.When the timer count reaches zero, an interrupt is generated. On each interrupt, the processor reads the current status of the LED connected to a GPIO port and toggles it. Consequently, the GPIO output is programmed by the timer interrupt. The functional block diagram as shown in **Figure 2-1** illustrates the working principle of the experiment.

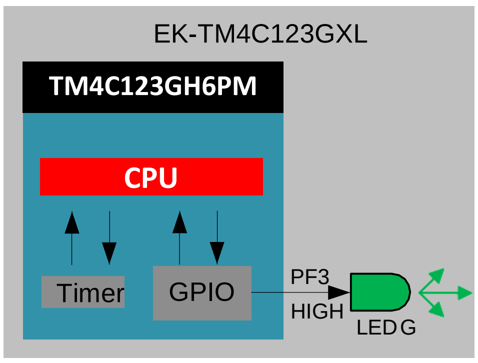


Figure 2-1 Functional Block Diagram

6.2.1 Timer

The programmable timer peripheral is used to count or time external events that drive the timer input pins. The GPT (General Purpose Timer) Module is one of the timing resource available on the Tiva™ C Series microcontrollers. Other timer resources include the System Timer and the PWM timer in the PWM modules. The features of the general purpose timer peripheral of the TM4C123GH6PM MCU are:

- Six 16/32-bit general purpose timers and six 32/64-bit wide general purpose timers
- Twelve 16/32-bit and twelve 32/64-bit capture/compare/PWM pins
- Timer modes supported:
  - One-shot
  - Periodic
  - Input edge count or time capture with 16-bit prescaler
  - PWM generation (separated only)
  - Real-Time Clock (concatenated only)
- Count up or down
- Simple PWM Support for timer synchronization, daisy-chains, and stalling during debug
- Can trigger ADC samples or DMA transfers

In this experiment, the timer is used in periodic mode. The capabilities of the general purpose timer in periodic mode are given in the **Table 2-1**.

Table 2-1: Capabilities of General Purpose Timer in Periodic Mode

Timer Use	Count Direction	Counter Size		Prescaler Size		Prescaler Behaviour (Count Diection)
		16/32-bit GPTM	32/64-bit Wide GPTM	16/32-bit GPTM	32/64-bit Wide GPTM	
Individual Up or Down 16	bit 32	bit 8	bit 16	bit	Individual Up or Down 16	Timer Extension (Up), Prescaler (Down
Concate-nated	Up or Down	32-bit	64-bit	-	-	N/A

6.2.2 GPIO Module

Tiva TM4C123GH6PM MCUs feature a GPIO (General Purpose Input Output) module that is composed of six physical GPIO blocks. Each block corresponds to an individual GPIO port (Port A, Port B, Port C, Port D, Port E, Port F). The GPIO pins are connected to the on-board LEDs in the LaunchPad as shown in the LaunchPad schematics in **Figure 2-2**.

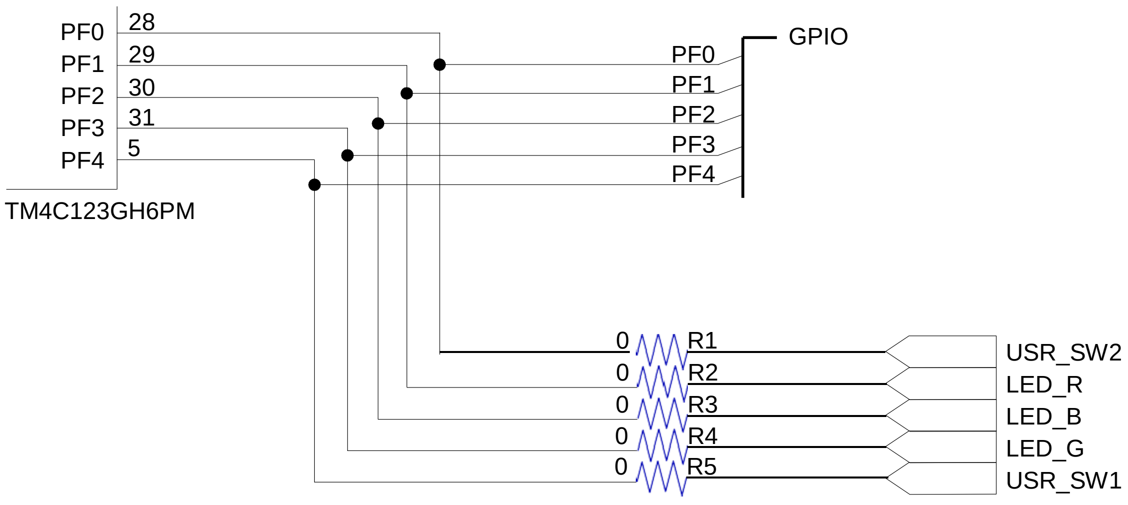


Figure 2-2 LaunchPad Schematics for GPIO Connection with LEDs

The features of GPIO are:

- Availability of up to 43 GPIOs, depending on configuration
- Highly flexible pin multiplexing allows usage of pins as GPIO or one of the several peripheral functions available
- Bit masking in both read and write operations through address lines
- Programmable control for GPIO interrupts:
  - Interrupt generation masking

- Edge-triggered on rising, falling, or both

- Level-sensitive on High or Low values

- Programmable weak pull-up, pull-down and open drain
- Programmable drive strength and slew rate control

The GPIO pins are configured with pin functions as shown in **Table 6-2** in the EK-TM4C123GXL.

Table 6-2: GPIO Pin Functions and Connected USB Devices

GPIO Pin	Pin Function	USB Device
PF4	GPIO	SW1
PF0	GPIO	SW2
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGB LED (Green)

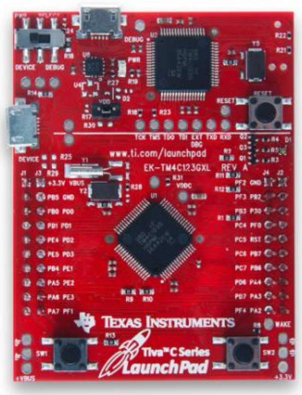
6.3 Component Requirements

6.3.1 Software Requirement

1. Code Composer Studio
2. TivaWare_C_Series

6.3.2 Hardware Requirement

Table 2-3: Components Required for the Experiment

S.No	Components	Specification	Image
1.	Tiva LaunchPad	EK-TM4C123GXL LaunchPad	
2.	USB Cable		

6.4 Software

The software for the experiment is written in C language and developed using the CCS Integrated Development Environment (IDE). Refer to "**Project Creation and Build**" in the Getting Started section of this manual for project build and debug using CCS. The software is programmed into the target device TM4C123GH6PM on the EK-TM4C123GXL using the USB interface.

6.4.1 Flowchart

The flowchart for the code is shown in **Figure 6-3**. The software written in C configures and enables the system clock to 40MHz. It then enables GPIO Port F of EK-TM4C123GXL and configures pin 3 (PF3) connected to the green LED as output. The timer is configured in periodic mode and the timer interrupt is enabled. On interrupt in periodic mode, the Timer Interrupt service Routine (ISR) reads the GPIO pin connected to the LED. If the current status of the LED is HIGH, then the processor sends a LOW to the LED and vice-versa. Thus, the green LED is toggled for each timer interrupt whose frequency is based on the timer load register value variable ui32Period in the code.

Calculation of timer period ui32Period

The timer counts for every cycle of the system clock frequency. The number of timer counts required to obtain a given frequency is calculated by

Number of clock cycles = System Clock Frequency / Desired Frequency

In the program, to toggle the GPIO at 10Hz and 50% duty cycle,

ui32Period = Number of clock cycles * Duty cycle = (40 MHz /10 Hz) * (50/100) =2 * 10^6 counts

DRAW YOUR
FLOWCHAT !

Figure 6-3 Flowchart for Interrupt Programming with GPIO

6.4.2 C Program Code for Interrupt Programming with GPIO

```
#include <stdint.h>

#include <stdbool.h>

#include "inc/tm4c123gh6pm.h"

#include "inc/hw_memmap.h"

#include "inc/hw_types.h"

#include "driverlib/sysctl.h"

#include "driverlib/interrupt.h"

#include "driverlib/gpio.h"

#include "driverlib/timer.h"


int main(void)

{

uint32_t ui32Period;

SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
SYSCTL_OSC_MAIN);

SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
GPIO_PIN_3);

SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);

ui32Period = (SysCtlClockGet() / 10) / 2;

TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);

IntEnable(INT_TIMER0A);

TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

IntMasterEnable();

TimerEnable(TIMER0_BASE, TIMER_A);

while(1)

{

}

}

void Timer0IntHandler(void)

{

// Clear the timer interrupt

TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

// Read the current state of the GPIO pin and+
// write back the opposite state
if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
}
else
{
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
}
```

Table 2-4: API Functions used in the Application Program

API Function	Parameters	Description
TimerConfigure(uint32_t ui32Base, uint32_t ui32Config)	• **ui32Base** is the base address of the timer module • **ui32Config** is the configuration for the timer	This function configures the operating mode of the timer(s).
SysCtlClockGet(void)		This function determines the clock rate of the processor clock
TimerLoadSet(uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)	• **ui32Base** is the base address of the timer module • **ui32Timer** specifies the timer(s) to adjust; must be one of **T**IMER_A, TIMER_B, or TIMER_BOTH. Only TIMER_A should be used when the timer is configured for full-width operation • **ui32Value** is the load value	This function configures the timer load value
TimerIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags)	• **ui32Base** is the base address of the timer module • **ui32IntFlags** is the bit mask of the interrupt sources to be enabled	This function enables the indicated timer interrupt sources
TimerEnable(uint32_t ui32Base, uint32_t ui32Timer)	• **ui32Base** is the base address of the timer module • **ui32Timer** specifies the timer(s) to enable; must be one of TIMER_A, TIMER_B, or TIMER_BOTH	This function enables operation of the timer module

6.5 Procedure

1. Connect the EK-TM4C123GXLto the PC using the USB cable supplied.

2. Build, program and debug the code into the EK-TM4C123GXL using CCS to view the status of the green LED.

6.6 Observation

The green LED blinks on successful execution of the program. The rate of blinking can be altered by changing the ui32Period variable in the program.

6.7 Summary

In this experiment, we have successfully configured the Timer interrupt and written an ISR for the general purpose timer to change the GPIO pin status.