BIRZEIT UNIVERSITY

Electrical and Computer Engineering

Computer Design LAB – ENCS4110

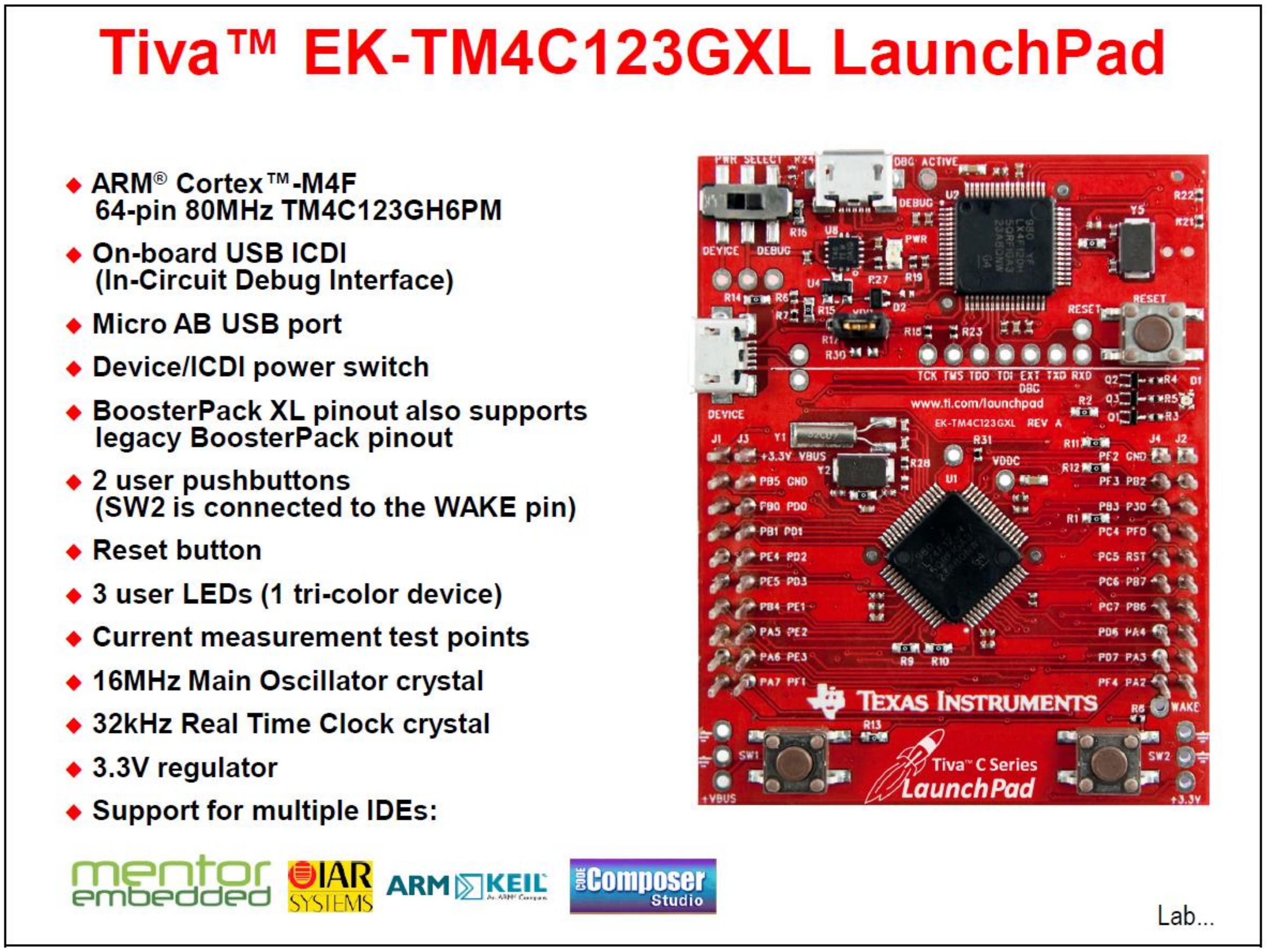
Experiment No. 6: GPIO (General Purpose Input/Output) Interface Using TM4C123G Boards

Objectives

The main objective of tis lab is to give students a first foot in the door exposure to the programming of I/O, which when executed by the microcontroller (TM4C123G, an ARM Cortex-M4) simply blinks LED located on the development board.

Tiva C Series TM4C123G LaunchPad Introduction

The TM4C123G is a member of the class of high performance 32-bit ARM cortex M4 microcontroller with a broad set of peripherals developed by Texas Instrumentals. The Tiva LaunchPad has a built-in processor clock frequency of up to 80MHz with a floating-point unit (FPU). The Cortex-M4F processor also supports the tail chaining functionality. It also includes a nested vector interrupt controller (NVIC). The debugging interface used is JTAG and SWD (serial wire debugger) for programming and debugging purposes**.**



Tiva™ EK-TM4C123GXL LaunchPad

- ARM® Cortex™-M4F
  64-pin 80MHz TM4C123GH6PM
- On-board USB ICDI
  (In-Circuit Debug Interface)
- Micro AB USB port
- Device/ICDI power switch
- BoosterPack XL pinout also supports legacy BoosterPack pinout
- 2 user pushbuttons
  (SW2 is connected to the WAKE pin)
- Reset button
- 3 user LEDs (1 tri-color device)
- Current measurement test points
- 16MHz Main Oscillator crystal
- 32kHz Real Time Clock crystal
- 3.3V regulator
- Support for multiple IDEs:

Lab...

TM4C123G LaunchPad Features

The TM4C123G has a vast variety of applications. It hosts a variety of communication peripherals, which can be used to connect all sorts of electronics devices; both sensors and actuators, for example, IR sensors, motors, etc. The TM4C123G is basically a Thumb2 16/32-bit code, which is 26% less memory and 25% faster than pure 32-bit. Moreover, it has a flexible clocking system, and can also access real time clock through hibernation module.

M4 Core and Floating-Point Unit

- 32-bit ARM® Cortex™-M4 core
- Thumb2 16/32-bit code: 26% less memory & 25 % faster than pure 32-bit
- System clock frequency up to 80 MHz
- 100 DMIPS @ 80MHz
- Flexible clocking system
  - Internal precision oscillator
  - External main oscillator with PLL support
  - Internal low frequency oscillator
  - Real-time-clock through Hibernation module
- Saturated math for signal processing
- Atomic bit manipulation. Read-Modify-Write using bit-banding
- Single Cycle multiply and hardware divider
- Unaligned data access for more efficient memory usage
- IEEE754 compliant single-precision floating-point unit
- JTW and Serial Wire Debug debugger access
  - ETM (Embedded Trace Macrocell) available through Keil and IAR emulators

TM4C123GH6PM Memory

256KB Flash memory
- Single-cycle to 40MHz
- Pre-fetch buffer and speculative branch improves performance above 40 MHz

32KB single-cycle SRAM with bit-banding

Internal ROM loaded with TivaWare software
- Peripheral Driver Library
- Boot Loader
- Advanced Encryption Standard (AES) cryptography tables
- Cyclic Redundancy Check (CRC) error detection functionality

2KB EEPROM (fast, saves board space)
- Wear-leveled 500K program/erase cycles
- Thirty-two 16-word blocks
- Can be bulk or block erased
- 10 year data retention
- 4 clock cycle read time

0x00000000	Flash
0x01000000	ROM
0x20000000	SRAM
0x22000000	Bit-banded SRAM
0x40000000	Peripherals & EEPROM
0x42000000	Bit-banded Peripherals
0xE0000000	Instrumentation, ETM, etc.

TM4C123GH6PM Peripherals

Battery-backed Hibernation Module

- Internal and external power control (through external voltage regulator)
- Separate real-time clock (RTC) and power source
- VDD3ON mode retains GPIO states and settings
- Wake on RTC or Wake pin
- Sixteen 32-bit words of battery backed memory
- 5 µA Hibernate current with GPIO retention. 1.7 µA without

Serial Connectivity

- USB 2.0 (OTG/Host/Device)
- 8 - UART with IrDA, 9-bit and ISO7816 support
- 6 - I^{2}C
- 4 - SPI, Microwire or TI synchronous serial interfaces
- 2 - CAN

TM4C123GH6PM Peripherals

Two 1MSPS 12-bit SAR ADCs

- Twelve shared inputs
- Single ended and differential measurement
- Internal temperature sensor
- 4 programmable sample sequencers
- Flexible trigger control: SW, Timers, Analog comparators, GPIO
- VDDA/GNDA voltage reference
- Optional hardware averaging
- 3 analog and 16 digital comparators
- µDMA enabled

0 - 43 GPIO

- Any GPIO can be an external edge or level triggered interrupt
- Can initiate an ADC sample sequence or µDMA transfer directly
- Toggle rate up to the CPU clock speed on the Advanced High-Performance Bus
- 5-V-tolerant in input configuration (except for PB0/1 and USB data pins when configured as GPIO)
- Programmable Drive Strength (2, 4, 8 mA or 8 mA with slew rate control)
- Programmable weak pull-up, pull-down, and open drain

TM4C123GH6PM Peripherals

Memory Protection Unit (MPU)

- Generates a Memory Management Fault on incorrect access to region

Timers

- 2 Watchdog timers with separate clocks
- SysTick timer. 24-bit high speed RTOS and other timer
- Six 32-bit and Six 64-bit general purpose timers
- PWM and CCP modes
- Daisy chaining
- User enabled stalling on CPU Halt flag from debugger for all timers

32 channel µDMA

- Basic, Ping-pong and scatter-gather modes
- Two priority levels
- 8,16 and 32-bit data sizes
- Interrupt enabled

TM4C123GH6PM Peripherals

Nested-Vectored Interrupt Controller (NVIC)

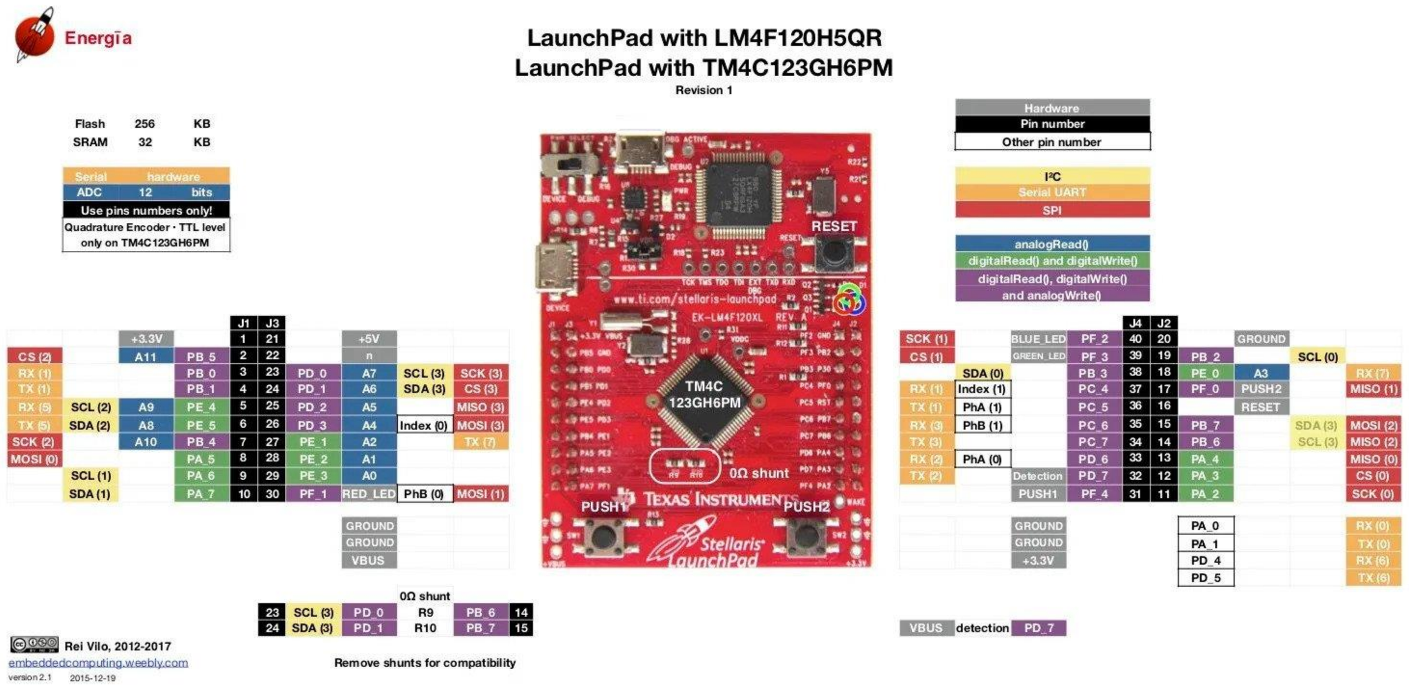
- 7 exceptions and 71 interrupts with 8 programmable priority levels
- Tail-chaining and other low-latency features
- Deterministic: always 12 cycles or 6 with tail-chaining
- Automatic system save and restore

Two Motion Control modules. Each with:

- 8 high-resolution PWM outputs (4 pairs)
- H-bridge dead-band generators and hardware polarity control
- Fault input for low-latency shutdown
- Quadrature Encoder Inputs (QEI)
- Synchronization in and between the modules

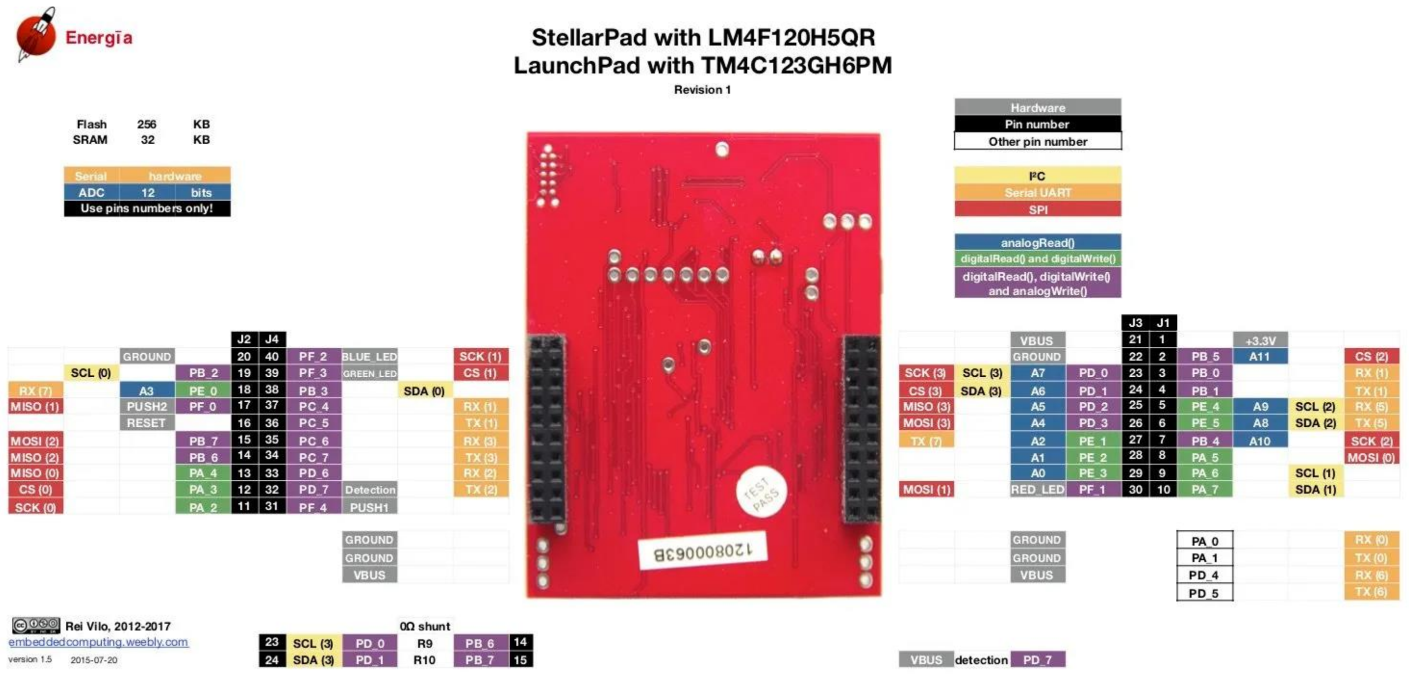
TM4C123G LaunchPad Pinout Diagram

The figure below shows the front end pinout diagram:



Energia
LaunchPad with LM4F120H5QR
LaunchPad with TM4C123GH6PM
Revision 1
Flash 256 KB
SRAM 32 KB
Serial hardware
ADC 12 bits
Use pins numbers only!
Quadrature Encoder · TTL level only on TM4C123GH6PM
Hardware
Pin number
Other pin number
I²C
Serial UART
SPI
analogRead()
digitalRead() and digitalWrite()
digitalRead(), digitalWrite() and analogWrite()
RESET
PUSH1
PUSH2
0Ω shunt
R9
R10
Remove shunts for compatibility
Rei Vilo, 2012-2017
embeddedcomputing.weebly.com
version 2.1 2015-12-19

The figure below shows the pin configuration for back end connector:



GPIO (General Purpose Input/Output)

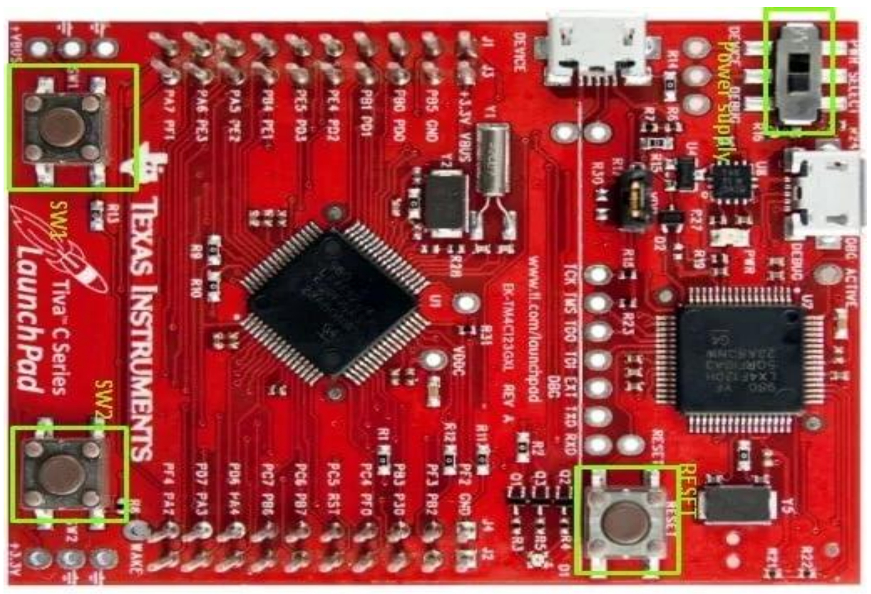
It has 0-43 general purpose input-output pins. Any GPIO can be used as an external edge or level-triggered interrupt, it can initiate ADC sampling, can change toggle rate to up to CPU clock speed on the advanced high-performance bus. Each input pin has a tolerance voltage of 5V in the input configuration. Every GPIO pin also has a weak pull-up, pull-down, and open drain.

There are many registers associated with each of the above I/O ports, and they have designated addresses in the memory map. The above addresses are the base addresses meaning that within that base address we have the registers associated with that port.

Port Name	Lower Address	Upper Address
GPIO port A	0x40004000	0x40004FFF
GPIO port B	0x40005000	0x40005FFF
GPIO port C	0x40006000	0x40006FFF
GPIO port D	0x40007000	0x40007FFF
GPIO port E	0x40024000	0x40024FFF
GPIO port F	0x40025000	0x40025FFF

On Board Push Buttons

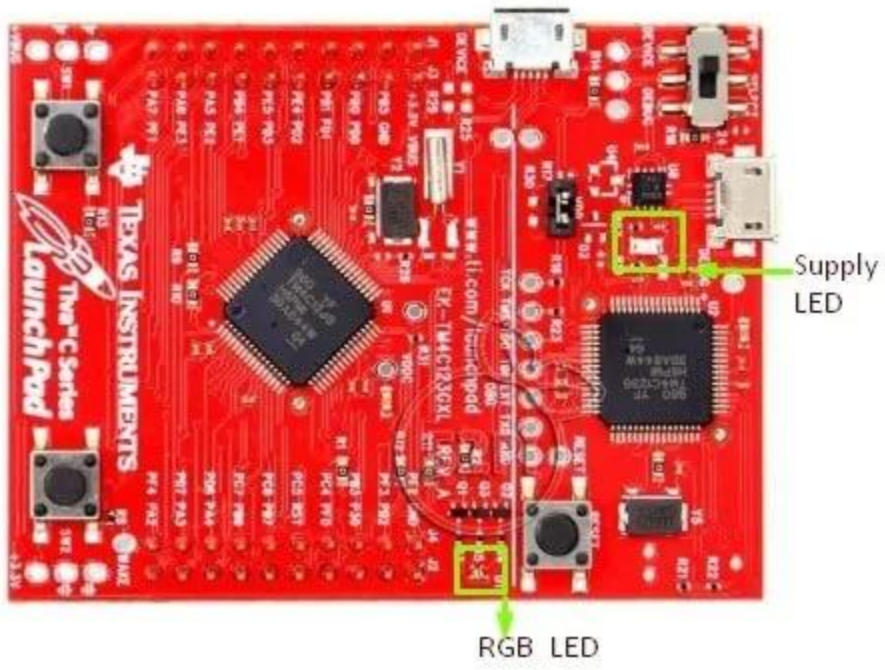
There are two onboard switches (push button) on the LaunchPad that are connected internally with the GPIO pins, a toggle switch that is used as a power switch, and another push button that is used to reset or restart the program execution that is already loaded on the board. As shown in the figure below:



SW1 push-button switch is connected to the **PF0** GPIO pin and the SW2 push-button switch is connected to the **PF4** GPIO pin.

Onboard LEDs

TIVA LaunchPad has one tri-color (RGB) LED onboard. It is internally connected to the F port of the GPIO pins, and when enabled, the LED shows the color of the enabled pin. Moreover, there is a power LED of color green onboard, when it is "on", it tells the user that the board is turned on. Both LEDs are highlighted in the figure below.



LED Flashing

A GPIO pin is a pin that can be configured through software to be either a digital input or a digital output. GPIO outputs let you translate logical values within your program to voltage values on output pins. These are the voltage outputs that help your microcontroller exert control over the system into which it is embedded.

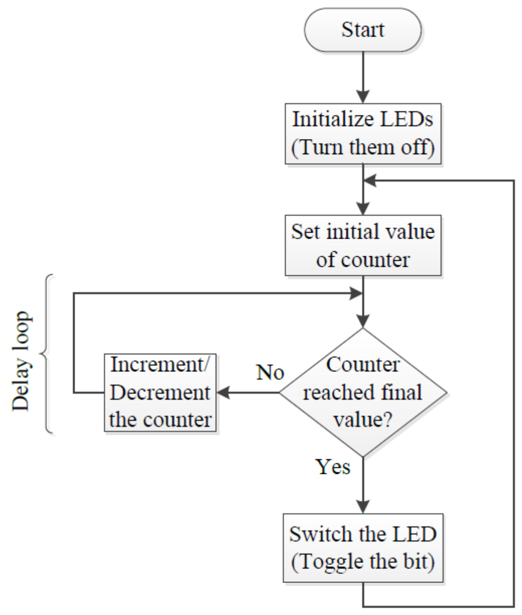
Configuring Peripherals

The fundamental initialization steps required to utilize any of the peripherals are:

1. Enable clocks to the peripheral
2. Configure pins required by the peripheral
3. Configure peripheral hardware

LED flashing code is implemented using an infinite loop which toggles bit 28 of the address 0x2009C020 after certain delay. The delay is implemented using the loop that just increments the loop counter until the condition is satisfied, as shown in the following flow-chart.

The overall structure of the program is shown in listing below. The program begins by defining macros for the relevant register addresses. The main routine follows the initialization steps described above then enters a loop in which it toggles an LED and waits for sometime.

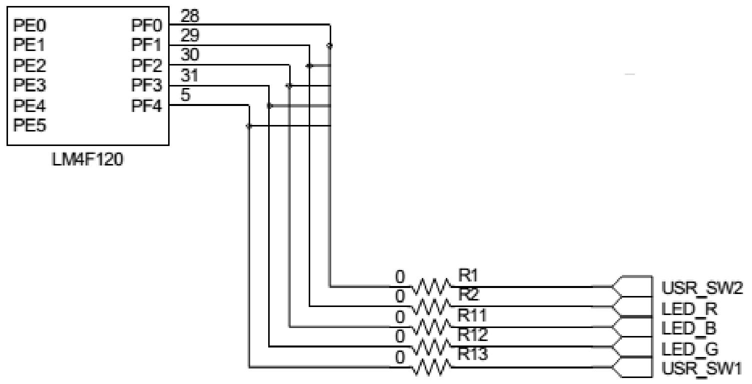

Start
Initialize LEDs (Turn them off)
Set initial value of counter
Counter reached final value?
No
Increment/ Decrement the counter
Yes
Switch the LED (Toggle the bit)
Delay loop

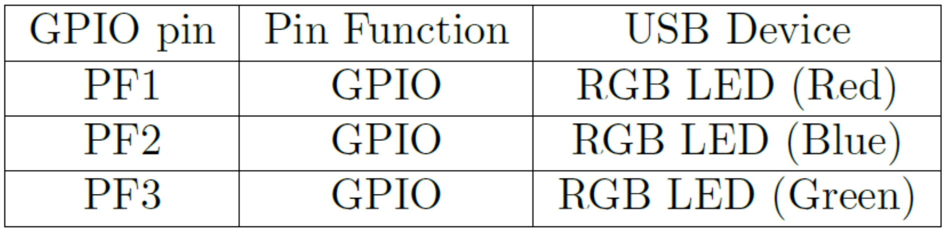
```
#include " lm4f120h5qr.h"

int main ( void )
 {
//Enable peripheral
. . . ( 1 ) . . .
// Conf igure pins
. . . ( 2 ) . . .
while ( 1 ) {
//Turn ON LED
. . . ( 3 ) . . .
//Delay for a bit
. . . ( 4 ) . . .
//Turn OFF LED
. . . ( 5 ) . . .
}
}
```

Where is LED?

The Stellaris LaunchPad comes with an RGB LED. The LED can be configured for use in any custom application. The following table shows how the LED is connected to the pins on the microcontroller. The following figure shows the physical connection of the LED.





GPIO pin	Pin Function	USB Device
PF1	GPIO	RGB LED (Red)
PF2	GPIO	RGB LED (Blue)
PF3	GPIO	RGB LED (Green)

LED Configuration

We will follow the steps stated above to configure the on-board LED.

Enabling the Clock

The **RCGCGPIO** register provides the software with the capability to enable and disable GPIO modules in Run mode. When enabled, a module is provided with a clock and access to the module. When disabled, the clock is disabled to save power and access attempts to module registers generate a bus fault. This register is shown in the figure below. The clock can be enabled for the GPIO port F by asserting the 6th bit of **RCGGPIO** register.

Now, to set any bit (i.e., make it 1) in a given register, we can do it in three different ways. For example, to set the 6th bit of RCGGPIO register, we can use:

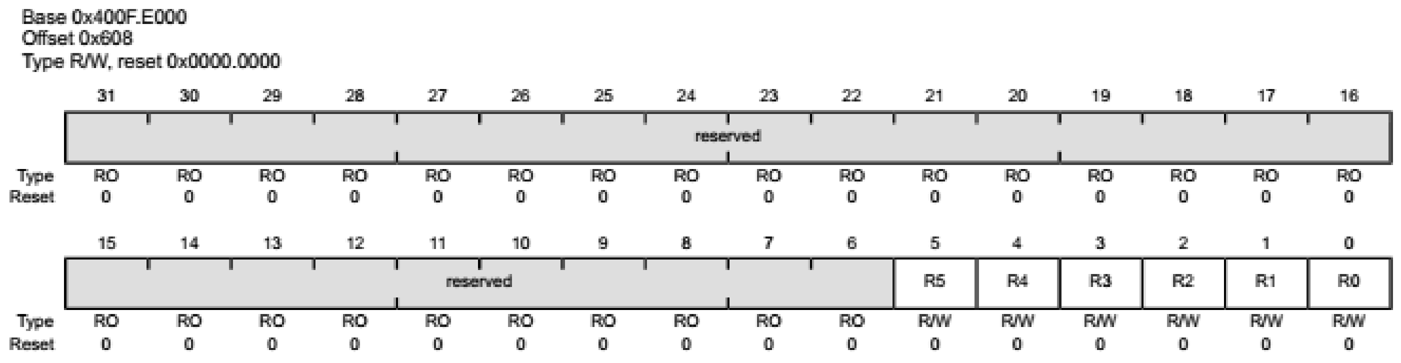
Case 1: RCGGPIO = (1<<6); // direct assign: other pins set to 0.

Case 2: RCGGPIO |= 0x20; // direct assign: other pins not affected

Case 3: RCGGPIO |= (1<<6); //binary – OR and assign: other pins not affected.

The following command can be used to enable clock signal for GPIO port F

```
SYSCTL_RCGCGPIO_R = 0x20 ;                // ( 1 )
```



Base 0x400F.E000
Offset 0x608
Type R/W, reset 0x0000.0000

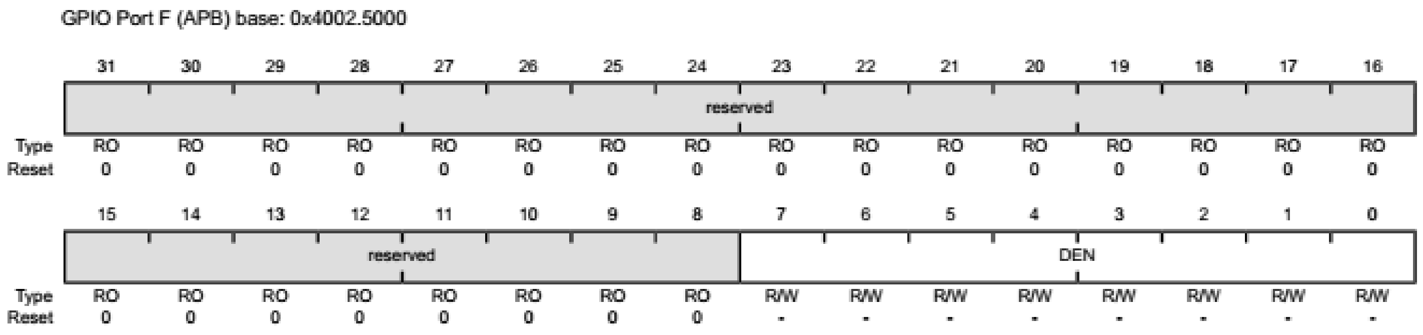
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved										R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

*General-Purpose Input/output Run Mode Clock Gating Control (**RCGCGPIO**)*

Configuring the Pin as Output

After enabling the clocks, it is necessary to configure any required pins. In this case, a single pin (PF3) must be configured as an output. To use the pin as a digital input or output, the corresponding bit in the GPIODEN register must be set, and then setting a bit in the GPIODIR register configures the corresponding pin to be an output

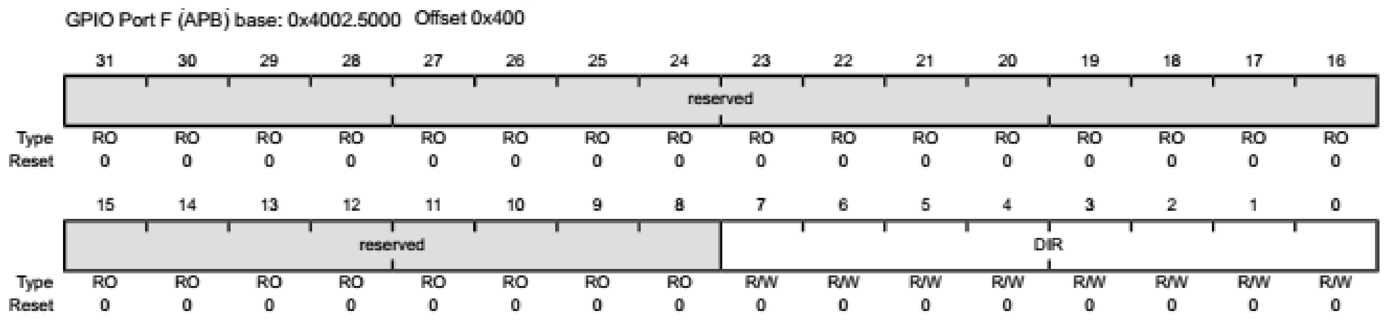


GPIO Port F (APB) base: 0x4002.5000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DEN							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-

GPIO Digital Enable (GPIODEN)



GPIO Port F (APB) base: 0x4002.5000 Offset 0x400

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DIR							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

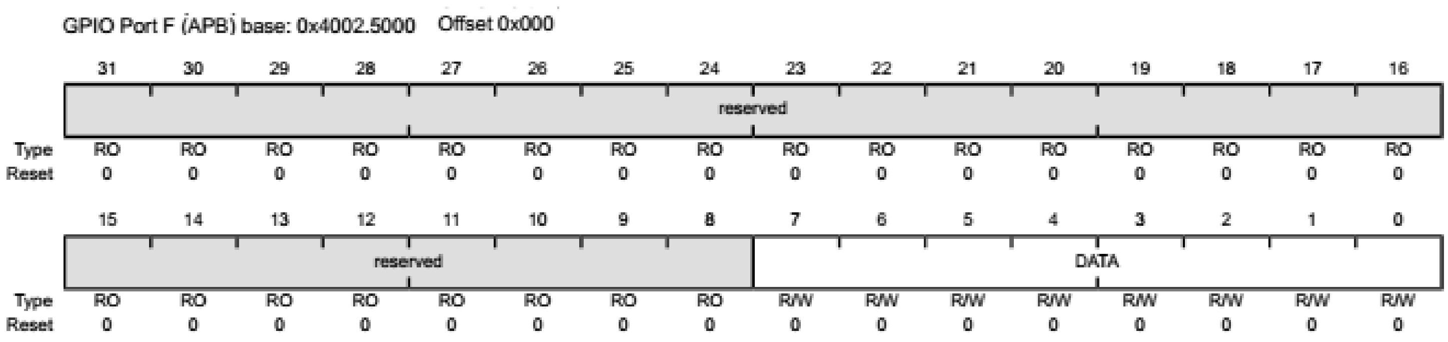
GPIO Direction (GPIODIR)

The commands used to set the corresponding bits in GPIODEN and GPIODIR registers are given as follows

```
GPIO PORTF DIR R = 0x08;            // ( 2 )
GPIO PORTF DEN R = 0x08 ;
```

Toggle the LED

After configuring the LED as an output, we want to toggle it after regular intervals. LED can be turned ON and OFF by setting and resetting the corresponding bits in the GPIODATA register.



GPIO Port F (APB) base: 0x4002.5000 Offset 0x000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								DATA							
Type	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

GPIO Data (GPIODATA)

The commands for toggling LED are as follows

```
GPIO PORTF DATA R = 0x08;        // ( 3 )
GPIO PORTF DATA R = 0x00 ;       // ( 5 )
```

Introducing a Delay

We cannot observe the toggling of LED because of very high frequency. We introduce a delay loop in order to observe the toggle sequence of the LED. The syntax for the loop is shown in the following figure

```
int counter = 0 ;
while ( count e r < 200000) {        // ( 4 )
++counter ;
}
```

Source Code

The complete assembly and C language code for the program is given as follows

Assembly Language Code

```
; Directives
        PRESERVE8
        THUMB ; Marks the THUMB mode of operation
        ;Data variables are declared in DATA AREA;
        AREA const_data , DATA, READONLY
        ; Initialing some constants
SYSCTL_RCGCGPIO_R     EQU    0x400FE608
GPIO_PORTF_AFSEL_R    EQU    0x40025420
```

```
GPIO_PORTF_DIR_R        EQU    0x40025400
GPIO_PORTF_DEN_R        EQU    0x4002551C
GPIO_PORTF_DATA_R       EQU    0x400253FC

DELAY EQU 200000

;The user code ( program) is placed in CODE AREA;
        AREA |.text| , CODE, READONLY, ALIGN=2
        ENTRY ; ENTRY marks the starting point of the code execution
        EXPORT __main
__main
; User Code starts from the next line
; Enable clock for PORT F
        LDR R1 , =SYSCTL_RCGCGPIO_R
        LDR R0 , [R1]
        ORR R0 ,R0, #0x20
        STR R0 , [R1]
        NOP ; No operations for 3 cycles
        NOP
        NOP
; Set the direction for PORT F
        LDR R1 , =GPIO_PORTF_DIR_R
        LDR R0 , [R1]
        ORR R0 , #0x08
        STR R0 , [R1]
    ; Digital enable for PORT F
        LDR R1 , =GPIO_PORTF_DEN_R
        LDR R0 , [R1]
        ORR R0 , #0x08
        STR R0 , [R1]
; Infinite loop LED flash
LED_flash
; Set the data for PORT F to turn LED on
        LDR R1 , =GPIO_PORTF_DATA_R
        LDR R0 , [R1]
        ORR R0 , R0 , #0x08
        STR R0 , [R1]

; Delay loop
        LDR R5 , =DELAY
delay1
        SUBS R5,#1
        BNE delay1

; Set the data for PORT F to turn LED off
        LDR R1 , =GPIO_PORTF_DATA_R
        LDR R0 , [R1]
        AND R0 , R0 , #0xF7
        STR R0 , [R1]
; Delay loop
        LDR R5 , =DELAY
delay2
        SUBS R5,#1
        BNE delay2
```

```
          B LED_flash

          ALIGN
          END ; End of the program , matched with ENTRY keyword
```

C language Code

```
 #define SYSCTL_RCGCGPIO_R       (*((volatile unsigned long *)0x400FE608))

 #define GPIO_PORTF_DATA_R       (*((volatile unsigned long *)0x400253FC))
 #define GPIO_PORTF_DIR_R       (*((volatile unsigned long *)0x40025400))
 #define GPIO_PORTF_DEN_R       (*((volatile unsigned long *)0x4002551C))

 #define DELAY 200000


int main ( void )
{
 volatile unsigned long ulLoop ;

// Enable the GPIO port that is used for the onboard LED.

          SYSCTL_RCGCGPIO_R = 0x20;

// Do a dummy read to insert a few cycles after enabling the peripheral.

          ulLoop = SYSCTL_RCGCGPIO_R;

//_ Enable the GPIO pin for the LED (PF3). Set the direction as output and enable the GPIO pin for digital //function. _/

          GPIO_PORTF_DIR_R = 0x08;
          GPIO_PORTF_DEN_R = 0x08;

 // Loop for ever.

while (1) {
 // Toggle  the LED.

          GPIO_PORTF_DATA_R ^= 0x08;  // ^ means XOR in c

// Delay for a bit.

for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
 {

 for (ulLoop = 0; ulLoop < DELAY; ulLoop++)
          {
   }

}
    }
}
```

<u>In-Lab Exercises</u>:

1- Modify the assembly program so that the Blue Led is flashing instead of the Green one.

-Increase delay amount to 2000000 and observe the response.

-Decrease the delay amount to 2000 and observe the response. Explain?

2- Modify the c code so the three Leds light in this sequence R->B->G. Choose a reasonable delay so the three led’s can be observed.