Experiment 7

Digital Input/Output Interfacing and Programming

Objective

The objective of this lab is to give you a hands-on exposure to the programming of I/O, which when executed by the microcontroller (TI LM4F120, an ARM Cortex-M4 based microcontroller) simply blinks LED on the development board.

Introduction to GPIO

A microcontroller communicates with the outside world either by setting the voltage on the pin high (usually 5V) or low (usually 0V) or reading the voltage level of an input pin as being high (1) or low (0). We refer to these pins as general purpose input output (GPIO) pins. Any GPIO pin 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 and voltage outputs help your microcontroller exert control over the system in which it is embedded.

Configuring Peripherals

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

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

Structure of the Program

The overall structure of this program is illustrated below. The program begins by including the addresses relevant peripheral registers. Main routine follows the initialization steps described above and then enters an infinite loop which toggles an LED and waits for sometime.

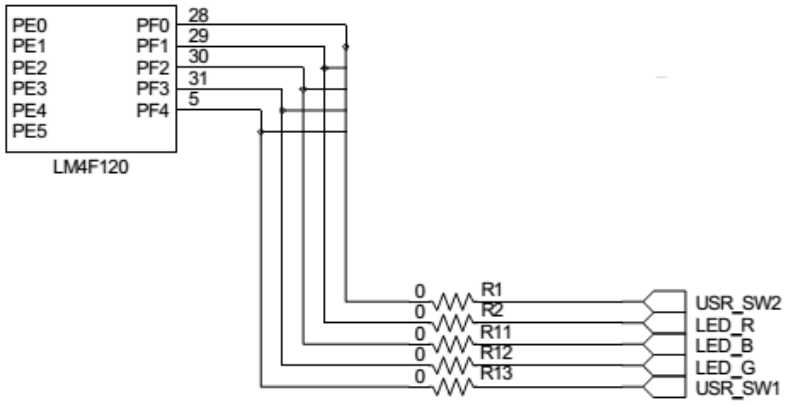
```
#define register_name    (*((volatile unsigned long *)register_address)

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

Pseudo code to blink LED

Where is LED?

The Stellaris LaunchPad has an RGB LED. This LED can be configured for any custom application. Table 7.1 shows how the LED is connected to the pins on the microcontroller. Figure 7.1 shows the physical connection of on-board LED.



LED schematic

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

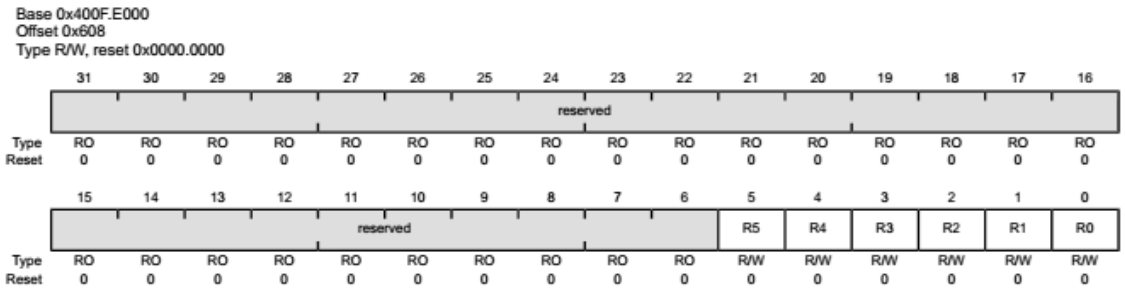
RGB LED Signals

LED Configuration

Now we will follow the above mentioned steps to configure the on-board LED.

Enabling the Clock

The RCGCGPIO register provides the capability to enable and disable GPIO modules in Run mode. When enabled, a module is provided a clock and access to module registers. When disabled, the clock is disabled to save power and accessing a module register generates a bus fault. This register is shown in Figure 7.2. The clock can be enabled for the GPIO port F by asserting the 6th bit of RCGGPIO register.



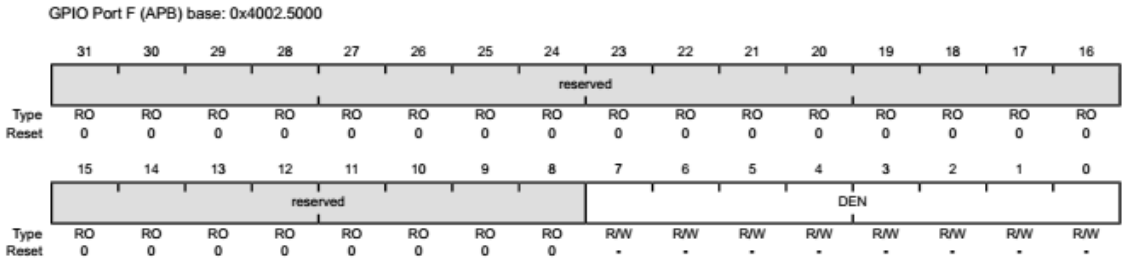
General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

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

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

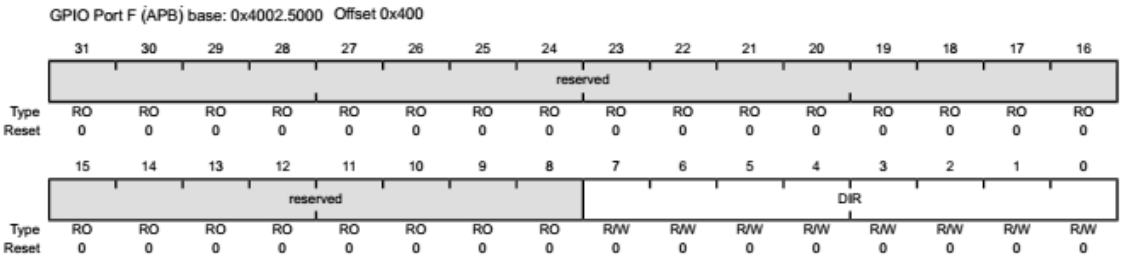
Configuring the Pin as Output

After enabling the clock, it is necessary to configure the 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 Digital Enable (GPIODEN)

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

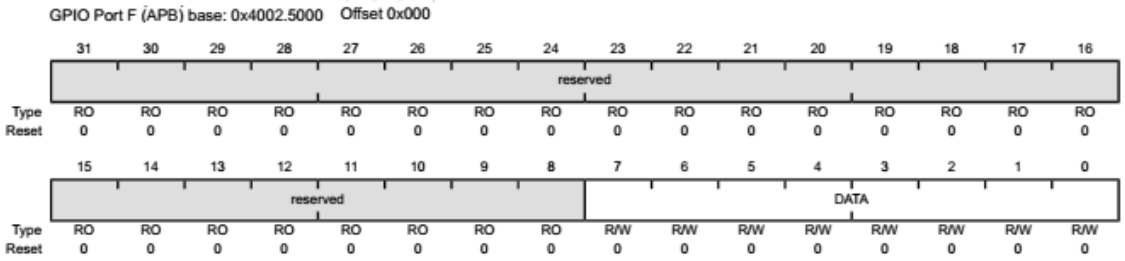


GPIO Direction (GPIODIR)

```
GPIO_PORTF_DIR_R = 0x08;      // (2)
GPIO_PORTF_DEN_R = 0x08;
```

Toggle the LED

After configuring the LED (pin PF3) 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 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. So, 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(counter < 200000){   // (4)
          ++counter;
    }
```

Source Code

The complete source code for the program is given below

Example 7.1

```
#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 GPIO_PORTF_CLK_EN       0x20
#define GPIO_PORTF_PIN3_EN      0x08
#define LED_ON                  0x08
#define LED_OFF                 ~(0x08)
#define DELAY                   200000

int main(void)
{
    volatile unsigned long ulLoop;

 // Enable the GPIO port that is used for the on-board LED.
    SYSCTL_RCGCGPIO_R |= GPIO_PORTF_CLK_EN;

 // 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 |= GPIO_PORTF_PIN3_EN;
    GPIO_PORTF_DEN_R |= GPIO_PORTF_PIN3_EN;

 // Loop forever.
    while(1)
    {
     // Turn on the LED.
        GPIO_PORTF_DATA_R |= LED_ON;

     // Delay for a bit.
        for(ulLoop = 0; ulLoop < DELAY; ulLoop++);

     // Turn off the LED.
        GPIO_PORTF_DATA_R &= LED_OFF;

     // Delay for a bit.
        for(ulLoop = 0; ulLoop < DELAY; ulLoop++);
    }
}
```