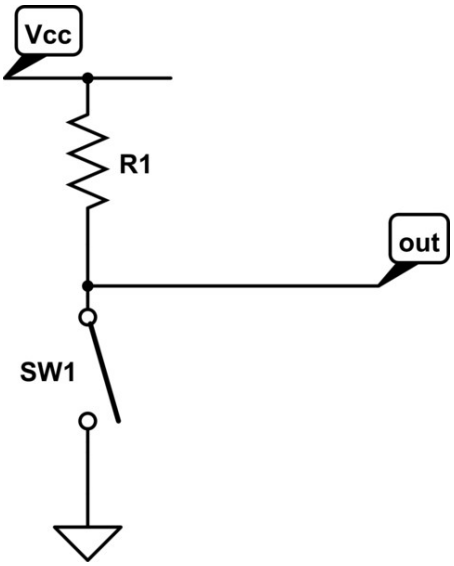
MIL: Introduction to GPIO

By Marquez Jones

What are General Purpose Input Outputs (GPIO)?:

One of the most basic uses for a microcontroller is to be able to turn on and off a device or be able to read a on/off signal. In our example code, we will look at how to turn and LED on and off first. We will then read in a button and output its state to an LED.

This will be accomplished via a system called GPIO which stands for General Purpose Input/Output. For the output case, the GPIO pin acts like a switch like the one below. You can either drive the pin high(to 3v3) or low(GND). Think of the GPIO pin as a switch that's controlled by your software.



In the input case, our microcontroller would be able to read a switch such as the one above and tell us where it's high or low.

Code Example: Blinking an LED

In this part, we're going to walk through blinking an LED in code. In this example, I'll show you how to initialize a GPIO pin to be an output and then how to toggle on and off the pin. For this part please open up the MIL_GPIO_Blink code in CCS.

DEFINES:

Define is a basically a label provided to some value or function. So when the compiler sees that label, it will place the function/value in its place. This increases readability. Here we make defines for the pin associated with the blue LED on the TIVA launchpad. In order to understand why I chose GPIO_PIN_2,you'll have to look at the launchpad schematic. PORTF_CLK_ENABLE is a macro which is like a define for functions. The Function it calls is the clock enable function for the GPIO Port; this is necessary for the GPIO to function.

```
/************************************ DEFINES/MACROS ****************************/

//defines
#define BLUE_LED_PIN GPIO_PIN_2     //check the TM4C123 Launchpad schematic

//macros
#define PORTF_CLK_ENABLE() SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
```

FUNCTION PROTOTYPES:

Function prototypes tells the compiler what functions you'll be using later in your code without have to define it at the top. This is also to increase readability as you don't necessary need to know how the function is fully defined to understand how it works or use it.

```
/************************************ FUNCTION PROTOTYPES ***********************/

/***************************************
 * Name: InitBlueLED
 * Desc: Set GPIO_PIN_2(BLUE LED) as output
 ***************************************/
void InitBlueLED(void);
```

FUNCTION DEFINITIONS:

Function definitions the definitions of functions previously prototyped. In our code, I wrote a function that initializes the blue LED on the launchpad. This uses a TivaWare function that tells your processor that the BLUE_LED_PIN(GPIO_PIN_2) on PORTF will be set as an output.

```
/**********************************FUNCTION DEFINITIONS*************************/

/***************************************
 * Name: InitBlueLED
 * Desc: Set GPIO_PIN_2(BLUE LED) as output
 ***************************************/
void InitBlueLED(void){

    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,BLUE_LED_PIN);

}
```

MAIN INIT SECTION:

Code Desc:

Here we initialize the clock for our system using a tailored clock init function made for MIL purposes. Whenever you write firmware, you need to call the MIL_ClkSetInt_16MHz function at the very beginning. We also enable our peripherals. Like our microcontroller, we must also enable the clock for each peripheral we deploy. The peripheral in question is a PORT F which contains our pins.

```
// initialize clock
MIL_ClkSetInt_16MHz();

//both the LED and switch are attached to port F
PORTF_CLK_ENABLE();

//gpio configurations
InitBlueLED();
```

MAIN WHILE LOOP:

NOTE: In embedded designs, most code will run in an infinite loop(while(1)). This is due to the embedded microprocessors will be operating while they have power.

Code Desc:

Here we call the GPIOPinWrite function in order to write a pin on the BLUE_LED_PIN then wait some amount of time before turning it off. We repeat this forever.

```
    while(1){

        //turn LED on
        GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED_PIN, BLUE_LED_PIN);

        //software delay(waste time)
        SysCtlDelay(1E6);

        //turn LED off
        GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED_PIN, 0x00);

        //software delay(waste time)
        SysCtlDelay(1E6);

    }
```

GPIOPinWrite:

This is a function provided by the TI TivaWare driver for writing data to our GPIO pins. I use it here to write a 1 to the LED and write a 0 to that pin.

Code Example: Reading a Button:

In this part, we'll go over how to read digital inputs from a button. We'll use the button to control the LED this time.

DEFINES:

Desc: In this code, I added defines for the switch we'll use. PUSH_SW_1 which is on GPIO_PIN_4(reference the launchpad schematic to understand why I chose GPIO PIN 4. The switch and LEDs are on the same port, so we can use the same CLK enable function from the previous example.

```
//defines
#define BLUE_LED_PIN GPIO_PIN_2

//both of these are the same value
//but for sake of clarity I made two separate
//defines
#define PUSH_SW_1_PIN GPIO_PIN_4
#define PUSH_SW_1_bm GPIO_PIN_4

//macros
#define PORTF_CLK_ENABLE() SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
```

FUNCTION PROTOTYPES:

Desc: Since we'll be adding a switch to our design, I added a function to initialize the Switch to an input pin.

```
42
43 //function prototypes
44 void InitSwitch(void);
45
46 void InitBlueLED(void);
47
```

FUNCTION DEFINITIONS:

We're using the same LED init function as above

```
76 /*************************************FUNCTION DEFINITIONS*************************/
77
78 /***************************************
79  * Name: InitBlueLED
80  * Desc: Set GPIO_PIN_2(BLUE LED) as output
81  ***************************************/
82 void InitBlueLED(void){
83
84     GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,BLUE_LED_PIN);
85
86 }
87
```

The additional function is with is InitSwitch is defined below. First the pin is set as an input pin since we'll be reading the value from the button. On top of this, there's an additional function that configures the pin for internal pull up. This is due to how the schematic was designed, it doesn't include a pull up resistor in hardware. For further explanation on why this is needed, check the appendix which has the switch schematic.

Refer here for info on pull ups: https://learn.sparkfun.com/tutorials/pull-up-resistors/all

```
/***************************************
 * Name: InitSwitch
 * Desc: Initializes GPIO_PIN_4 on port F as an
 *       input pin
 ***************************************/
void InitSwitch(void){

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, PUSH_SW_1_PIN);

    /*
     * Fundamental Rule of digital circuits,
     * all signals must be high or low
     * Check launchpad schematic to see why
     * the pull up in software is required
     */

    /*
     * Configure the push input to have an interal pull up resistor
     *
     * Notes:the gpio_strength parameter would set the current output to be
     * 2mA if the pin were configured as an output. In this case it has no effect
     * since the pin is an input pin
     */
    GPIOPadConfigSet(GPIO_PORTF_BASE,
                     PUSH_SW_1_PIN,
                     GPIO_STRENGTH_2MA,
                     GPIO_PIN_TYPE_STD_WPU);

}
```

MAIN INIT SECTION:

Code Desc:

The only change to the init section, is that we've also called out InitSwitch function which configures our input pin.

```
MIL_ClkSetInt_16MHz();

//both the LED and switch are attached to port F
PORTF_CLK_ENABLE();

//gpio configurations
InitBlueLED();

InitSwitch();
```

MAIN WHILE LOOP:

NOTE: In embedded designs, most code will run in an infinite loop(while(1)). This is due to the embedded microprocessors will be operating while they have power.

Code Desc:

Here we read the GPIO pin using GPIOPinRead which returns a bit field with the status of our switch. Note, there are 8 pins in each port, and the function allows you to read all 8 of them at once. The second parameter in the function tells you which pins, you want to look at so it'll set the other bits to 0 in the bit field. Our switch is in pin 4 which corresponds to the 5th bit (pin 0 is the first bit). The output of that function will look like 0001 0000 if the pin is high and 0000 0000 if the pin is low.

For more info on bitfield/bitmasking: https://en.wikipedia.org/wiki/Mask_(computing)

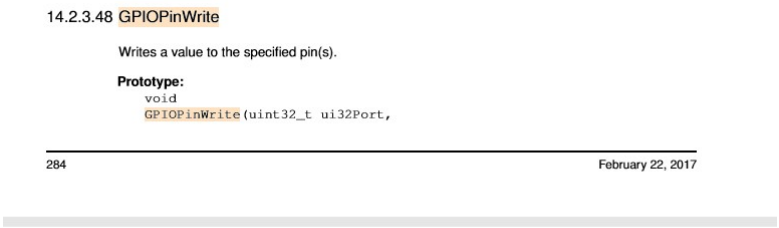
```
while(1){


    /*
     * Button note:
     * the buttons on the launchpad are active low
     */

    //check if button 1 has been pressed
    //if button is not pressed
    //if so turn LED off
    if(GPIOPinRead(GPIO_PORTF_BASE,PUSH_SW_1_PIN) & PUSH_SW_1_bm){

        STATE = 0x00;

    }
    //otherwise the button was pressed
    else{

        STATE = 0xFF;

    }
    GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED_PIN, STATE);

}
```

Code Exercise: Alternating LEDs:

In this part, you'll use some skeleton code to turn on LEDs alternating. In this, when the button is pressed, the blue LED will turn on. When the button is not pressed, the red LED will turn on and blue LED will turn off.

Appendix(Schematics and Functions):

GPIOPinWrite Function:

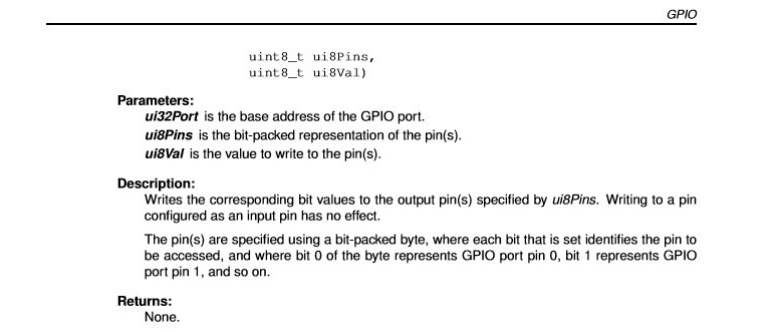


14.2.3.48 GPIOPinWrite

Writes a value to the specified pin(s).

Prototype:

```
void
GPIOPinWrite(uint32_t ui32Port,
```


```
             uint8_t ui8Pins,
             uint8_t ui8Val)
```

Parameters:

ui32Port is the base address of the GPIO port.
ui8Pins is the bit-packed representation of the pin(s).
ui8Val is the value to write to the pin(s).

Description:

Writes the corresponding bit values to the output pin(s) specified by *ui8Pins*. Writing to a pin configured as an input pin has no effect.

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

Returns:

None.

GPIOPinRead Function:

14.2.3.20 GPIOPinRead

Reads the values present of the specified pin(s).

Prototype:

```
int32_t
GPIOPinRead(uint32_t ui32Port,
            uint8_t ui8Pins)
```

Parameters:

ui32Port is the base address of the GPIO port.

ui8Pins is the bit-packed representation of the pin(s).

Description:

The values at the specified pin(s) are read, as specified by *ui8Pins*. Values are returned for both input and output pin(s), and the value for pin(s) that are not specified by *ui8Pins* are set to 0.

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

Returns:

Returns a bit-packed byte providing the state of the specified pin, where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on. Any bit that is not specified by *ui8Pins* is returned as a 0. Bits 31:8 should be ignored.

26.2.2.32 SysCtlPeripheralEnable

Enables a peripheral.

Prototype:

```
void
SysCtlPeripheralEnable(uint32_t ui32Peripheral)
```

Parameters:

ui32Peripheral is the peripheral to enable.

Description:

This function enables a peripheral. At power-up, all peripherals are disabled; they must be enabled in order to operate or respond to register reads/writes.

The *ui32Peripheral* parameter must be only one of the following values: **SYSCTL_PERIPH_ADC0**, **SYSCTL_PERIPH_ADC1**, **SYSCTL_PERIPH_CAN0**, **SYSCTL_PERIPH_CAN1**, **SYSCTL_PERIPH_CCM0**, **SYSCTL_PERIPH_COMP0**, **SYSCTL_PERIPH_EEPROM0**, **SYSCTL_PERIPH_EMAC**, **SYSCTL_PERIPH_EPHY**, **SYSCTL_PERIPH_EPI0**, **SYSCTL_PERIPH_GPIOA**, **SYSCTL_PERIPH_GPIOB**, **SYSCTL_PERIPH_GPIOC**, **SYSCTL_PERIPH_GPIOD**, **SYSCTL_PERIPH_GPIOE**, **SYSCTL_PERIPH_GPIOF**, **SYSCTL_PERIPH_GPIOG**, **SYSCTL_PERIPH_GPIOH**, **SYSCTL_PERIPH_GPIOJ**, **SYSCTL_PERIPH_GPIOK**, **SYSCTL_PERIPH_GPIOL**, **SYSCTL_PERIPH_GPIOM**, **SYSCTL_PERIPH_GPION**, **SYSCTL_PERIPH_GPIOP**, **SYSCTL_PERIPH_GPIOQ**, **SYSCTL_PERIPH_GPIOR**, **SYSCTL_PERIPH_GPIOS**, **SYSCTL_PERIPH_GPIOT**, **SYSCTL_PERIPH_HIBERNATE**, **SYSCTL_PERIPH_I2C0**, **SYSCTL_PERIPH_I2C1**, **SYSCTL_PERIPH_I2C2**, **SYSCTL_PERIPH_I2C3**, **SYSCTL_PERIPH_I2C4**, **SYSCTL_PERIPH_I2C5**, **SYSCTL_PERIPH_I2C6**, **SYSCTL_PERIPH_I2C7**, **SYSCTL_PERIPH_I2C8**, **SYSCTL_PERIPH_I2C9**, **SYSCTL_PERIPH_LCD0**, **SYSCTL_PERIPH_ONEWIRE0**, **SYSCTL_PERIPH_PWM0**, **SYSCTL_PERIPH_PWM1**, **SYSCTL_PERIPH_QEI0**, **SYSCTL_PERIPH_QEI1**, **SYSCTL_PERIPH_SSI0**, **SYSCTL_PERIPH_SSI1**, **SYSCTL_PERIPH_SSI2**, **SYSCTL_PERIPH_SSI3**, **SYSCTL_PERIPH_TIMER0**, **SYSCTL_PERIPH_TIMER1**, **SYSCTL_PERIPH_TIMER2**, **SYSCTL_PERIPH_TIMER3**, **SYSCTL_PERIPH_TIMER4**, **SYSCTL_PERIPH_TIMER5**, **SYSCTL_PERIPH_TIMER6**, **SYSCTL_PERIPH_TIMER7**, **SYSCTL_PERIPH_UART0**, **SYSCTL_PERIPH_UART1**, **SYSCTL_PERIPH_UART2**, **SYSCTL_PERIPH_UART3**, **SYSCTL_PERIPH_UART4**, **SYSCTL_PERIPH_UART5**, **SYSCTL_PERIPH_UART6**, **SYSCTL_PERIPH_UART7**, **SYSCTL_PERIPH_UDMA**, **SYSCTL_PERIPH_USB0**, **SYSCTL_PERIPH_WDOG0**, **SYSCTL_PERIPH_WDOG1**, **SYSCTL_PERIPH_WTIMER0**, **SYSCTL_PERIPH_WTIMER1**, **SYSCTL_PERIPH_WTIMER2**, **SYSCTL_PERIPH_WTIMER3**, **SYSCTL_PERIPH_WTIMER4**, or **SYSCTL_PERIPH_WTIMER5**

Note:

It takes five clock cycles after the write to enable a peripheral before the the peripheral is actually enabled. During this time, attempts to access the peripheral result in a bus fault. Care should be taken to ensure that the peripheral is not accessed during this brief time period.

Returns:

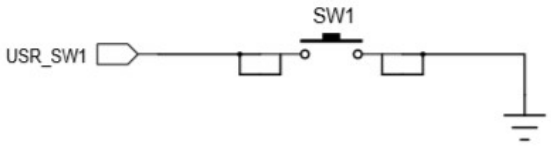
None.

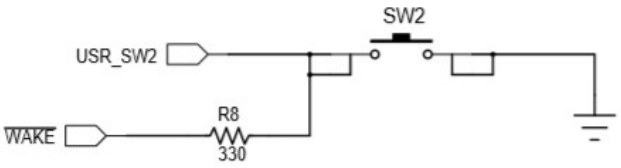
Schematics:

SWITCH CIRCUITS:

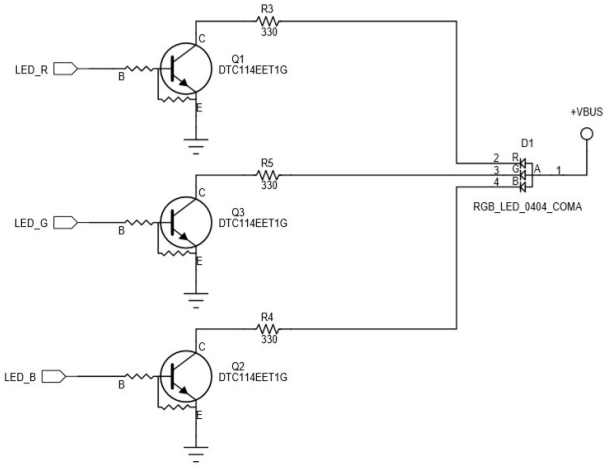
NOTE: The switches on the board do not have pull up resistors. In digital circuits, the voltage on an input must take on one of two states to be considered valid. The voltage must either be high (3.3V) or low (0V). In the current hardware configuration, the switch is either low (0V) when pressed or nothing if open since there's nothing driving the input. Open mean no connection. So in order to drive the input high when, the button is not pressed, we have to use a feature of the MCU which is internal pull ups meaning the MCU will use an internal pull up that'll drive the input high when the button is not pressed.

Refer here for info on pull ups: https://learn.sparkfun.com/tutorials/pull-up-resistors/all





LED CIRCUITS:



\

CONNECTIONS TO MCU:

NOTE: In this document we only care about the switch(USR_SW) and LED connections.

So

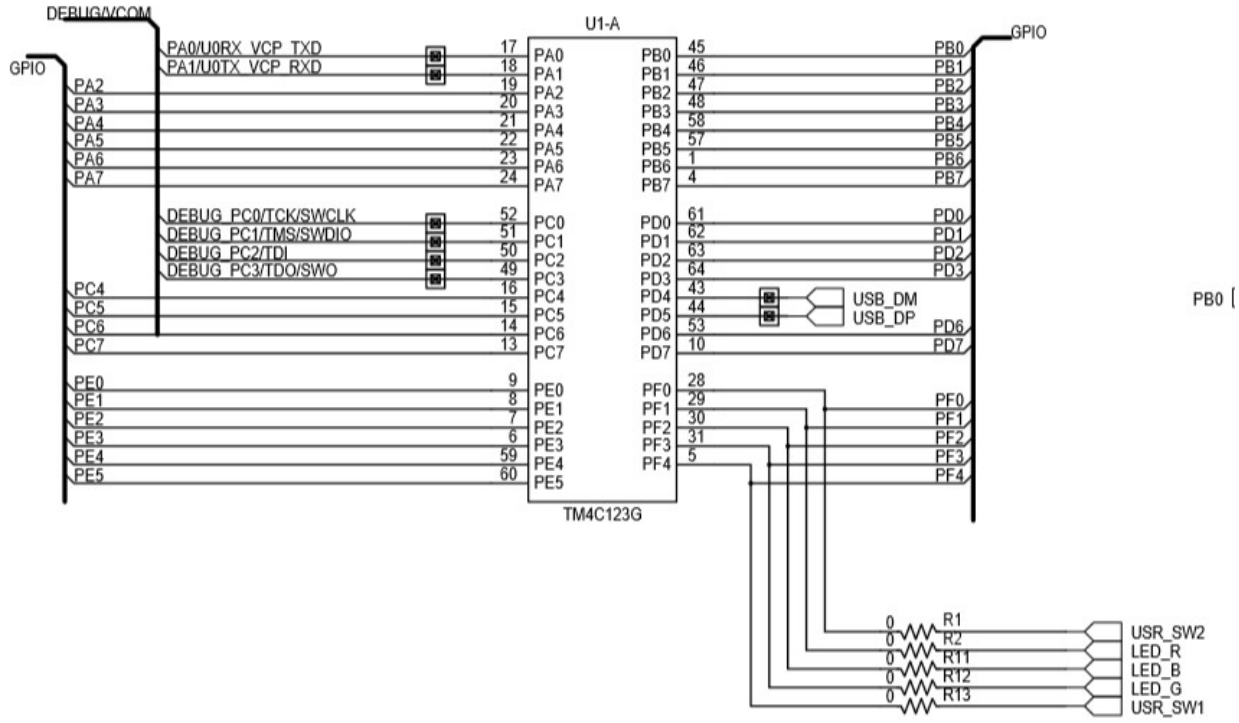
USR_SW1 => PIN 4 ON PORT F(PF4)

USR_SW2 => PIN 0 ON PORT F(PF0)

LED_R => PIN 1 ON PORT F(PF1)

LED_B => PIN 2 ON PORT F(PF2)

LED_G => PIN 3 ON PORT F(PF3)



Solutions to Exercise: