Radiocrafts
Embedded Wireless Solutions

SPR Tutorial 1: Button, LED and Timer

v1.0, August 2, 2018

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

No table of figures entries found.

Abbreviations

Abbreviation	Description
SPR	Radiocrafts Software-Programmable RF Module
RIIoT	Radiocrafts Industrial Internet of Things

1 Introduction

In this tutorial, you will develop a simple application that blinks a LED. Pressing a button starts or stops the blinking of the LED.

You will learn:

- How to connect a button to a GPIO input.
- How to connect a LED to a GPIO output
- How to configure pullup or pulldown of a GPIO
- How to handle events based on GPIO input (e.g. rising or falling edge)
- How to use a timer
- How to use UART print statements to debug your application

This short tutorial introduces you to the basics of interfacing GPIO and using timers. When you finish this, you can go on to other tutorials that show how to connect to a network and other interfaces like I2C, SPI, ADC.

2 Setup

For this tutorial, you will need:

- RC1880 Sensor Board
- a button switch

In addition, you need the compiler and flashing tool as introduced in the **Getting Started Guide**. So if you have not read it, please start there first.

2.1 Hardware Setup

Connect one end of the button to GPIO_8 (module pin 34). Connect the other end to ground.

When the button is pressed, it will short to ground and pull the signal on the pin low. Later when we configure the GPIO pin, we will configure it with an internal pull-up so that the signal is normally high until the button is pressed.

For the LED, the application will use the green LED on the sensor board that is connected to GPIO_0 on the module.

For all the GPIO pins available for the module, read the section on GPIO in SPR Platform User's Guide

3 Coding

The tutorial will build up the application code step by step, but you can skip to see to the full source code in 3.7. You can also grab the full source code in the `example` folder, in the file `tutorial1_button_blink_led.c`.

3.1 Include the SPR header file

In your SPR SDK project folder, start a new source file `app.c`.

You can name your source file however you like, but then remember to modify the `SOURCE_FILES` *option in the file* `user_build_options.ini` *with the name of your file so the compiler knows which file to compile.*

The top of every SPR application must include the header file `spr_app.h` where all the APIs are declared.

```
#include "spr_app.h"
```

3.2 Map the Button and LED to GPIOs

Next we create constant macros to map the button and LED to the correct GPIO pins.

```
/********* Constants ***********/
#define BUTTON_0 GPIO_8
#define LED_0 GPIO_0
```

With the macros defined, you can now reference the button and LED in the rest of the code without remembering which GPIO pin they are connected to.

3.3 Declare the timer

We will declare a variable for our timer. We will later use this variable when creating, starting and stopping the timer.

```
/********* File-scope Variables ****************/
static TimerId ledBlinkTimer;
```

`static` *just means that we can access this timer variable anywhere within this file, but not outside your file.*

3.4 The Setup()

Next, you write the function `Setup()`, which is called whenever the module starts. Here you can configure the GPIOs for the button and the LED, and also register a handler function for button press. Let's take a look.

```
void Setup()
{
   Debug.printf("My Setup\r\n"); (1)

   // setup the button GPIO and event detection
   GPIO.setDirection(BUTTON_0, INPUT);
   GPIO.setPull(BUTTON_0, PULL_UP); (2)
   GPIO.setHandler(BUTTON_0, FALLING_EDGE, buttonHandler); (3)

   // setup the LED
   // no pull up is set. it uses output push-pull
   GPIO.setDirection(LED_0, OUTPUT);
   // initialize to off
   GPIO.setValue(LED_0, LOW);

   // create the timer
   ledBlinkTimer = Timer.create(PERIODIC, 500*MILLISECOND, ledBlinkTimerHandler); (4)
}
```

(1) This will be printed on your UART COM port to show whenever the application starts.
(2) Configure the GPIO with an internal pull up, so it is normally high. When the button is pressed, it shorts the signal to low
(3) Configures the button GPIO to call the function `buttonHandler` when the pin triggers on any falling edge (from high to low). We will implement the handler in the next section
(4) This creates a periodic timer that calls the function `ledBlinkTimerHandler` every 500 milliseconds.

3.5 The Button Handler

Next we implement the button handler. Pressing the button toggles between starting and stopping the blinking timer.

Add the following code after `Setup()`.

```
static void buttonHandler(enum GPIO_Pin pin, enum GPIO_InterruptEdge edge)
{
  Debug.printline("Button value=%d", GPIO.getValue(BUTTON_0));

  if (Timer.isActive(ledBlinkTimer)) {
     Timer.stop(ledBlinkTimer);
```

```
        Debug.println("Start blinking");
    } else {
        Timer.start(ledBlinkTimer);
        Debug.println("Stop blinking");
    }
}
```

A function declared as `static` just means that the function can only be called within this file. It is the C language's way to make a function private.

`Debug.println` follows the same format specifiers as the standard `printf` in C. In this code, the `%d` specifier in `value=%d` means that `GPIO.getValue(BUTTON_0)` will be inserted here and printed as a decimal integer. Other specifiers include %s for string, %s for hex.

3.5.1 Declaring the Handler

Because the `buttonHandler()` comes after `Setup()`, it must be declared before `Setup()` so the code within `Setup()` knows how to reference to it.

Add the following line before `Setup()`

```
/********* Private Function Declarations ****************/
static void buttonHandler(enum GPIO_Pin pin, enum GPIO_InterruptEdge edge);
```

Don't forget the semi-colon `;` at the end of the line!

3.6 Timer Handler

The timer handler is called whenever the timer triggers (every 500ms). It toggles the LED.

Add the following code after the button handler:

```
static void ledBlinkTimerHandler()
{
    GPIO.toggle(LED_0);
}
```

Then add the declaration of the handler before Setup(). Your list of declared functions should now look like this:

```
/********* Private Function Declarations ****************/
static void buttonHandler(enum GPIO_Pin pin, enum GPIO_InterruptEdge edge);
static void ledBlinkTimerHandler();
```

3.7 Full Source Code

The full source code should look like below. You can also grab the full source code in the `example` folder, in the file `tutorial1_button_led.c`.

```
#include "spr_app.h"

/********* Constants ***********/
#define BUTTON_0 GPIO_8
#define LED_0 GPIO_0

/********* Private Variables ****************/
// These are variables that can be shared between functions in this file.
// static int counter = 0;
static TimerId ledBlinkTimer;

/********* Private Function Declarations ****************/
// These are event handlers that will be implemented later in the file, but
// are declared here so Setup() can reference them when registering events.
static void button0Handler(enum GPIO_Pin pin, enum GPIO_InterruptEdge edge);
static void ledBlinkTimerHandler();

/********* Public Functions ****************/

/**
* Setup() is called by the framework on startup
*/
void Setup()
{
   Debug.println("My Setup");

   // Configure GPIO for Button 0
   GPIO.setDirection(BUTTON_0, INPUT);
   GPIO.setPull(BUTTON_0, PULL_UP);
   GPIO.setHandler(BUTTON_0, FALLING_EDGE, button0Handler);

   // Configure GPIO for LED 0
   GPIO.setDirection(LED_0, OUTPUT);
   GPIO.setValue(LED_0, LOW);

   // create timer
   ledBlinkTimer = Timer.create(PERIODIC, 500*MILLISECOND, ledBlinkTimerHandler);
}

/********* Private Functions ****************/
static void button0Handler(enum GPIO_Pin pin, enum GPIO_InterruptEdge edge)
{
   Debug.println("Button value=%d", GPIO.getValue(BUTTON_0));
```

```
    if (Timer.isActive(ledBlinkTimer)) {
       Timer.stop(ledBlinkTimer);
       Debug.printline("Timer stop");
    } else {
       Timer.start(ledBlinkTimer);
       Debug.printline("Timer start");
    }
}

static void ledBlinkTimerHandler()
{
    Debug.printline("Blink");
    GPIO.toggle(LED_0);
}
```

4 Compile and Test

Compile and flash the application using the same steps introduced in the Getting Started Guide.

Open a serial terminal so you can see the debugging statements.

Try it out. Press the button. See if the LED blinks!

5 What's Next

Congratulations on completing this tutorial!

You can read the SPR Platform User's Guide for more details on the API for GPIO.

The next tutorial will show you how to connect the module to a network.

6 Revision History

Revision	Date	Changes
V1.0	August 2, 2018	For release

Disclaimer

Trademarks

Life Support Policy

Radiocrafts Support:

Knowledge base: https://radiocrafts.com/knowledge-base/
Application notes library: https://radiocrafts.com/resources/application-notes/
Whitepapers: https://radiocrafts.com/resources/articles-white-papers/
Technology overview: https://radiocrafts.com/technologies/
RF Wireless Expert Training: https://radiocrafts.com/resources/rf-wireless-expert-training/

Contact Radiocrafts

Sales requests: https://radiocrafts.com/contact/