Using hardware and software to make new stuff

Search

type, hit enter

Categories

- 3D Printing (1)
- Affinity Photo (1)
- Aide-memoir (1)
- Analog (1)
- Ansible (3)
- Dates to Remember (3)
- Docker (1)
- Electronics (144)
- ESP8266 (12)
- FPGA (4)
- Garden (1)
- General (8)
- Home Built CPU (6)
- Informatica (1)
- Internet of Things (8)
- iOS (2)
- KiCad (4)
- MSP430 (2)
- Netduino (49)
- NuttX (9)
- Photography (3)
- Pico (10)
- Raspberry Pi (17)
- Silverlight (6)
- Software Development (88)
- STM32 (5)
- STM8 (54)
- Tips (5)

Archives

- July 2024
- June 2024
- May 2024
- April 2024
- March 2024
- February 2024
- January 2024
- December 2023
- November 2023
- October 2023

- April 2023
- February 2020
- January 2020
- July 2019
- February 2018
- July 2017
- June 2017
- April 2017
- March 2017
- February 2017
- October 2016
- September 2016
- July 2016
- June 2016
- May 2016
- April 2016
- March 2016
- January 2016
- December 2015
- November 2015
- October 2015
- August 2015
- June 2015
- May 2015
- April 2015
- March 2015
- February 2015
- January 2015
- December 2014
- October 2014
- September 2014
- August 2014
- July 2014
- June 2014
- May 2014
- March 2014
- February 2014
- January 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- June 2013
- May 2013
- April 2013
- March 2013
- January 2013
- November 2012
- October 2012
- September 2012
- August 2012
- July 2012
- June 2012
- May 2012

- October 2011
- September 2011
- July 2011
- June 2011
- May 2011
- April 2011
- March 2011
- February 2011
- January 2011
- December 2010
- April 2010

External Interrupts on the STM8S

In a previous post we looked at the GPIO pins and how we could set the pins to either input or output but we concentrated on output. In this post we will look at using interrupts to detect input from the user and an output to indicate that the input has been detected. Our objective is:

1. Create a circuit which can register when a button has been pressed.
2. Toggle a LED when the button is pressed

Should be simple, so let's give it a go.

Hardware

For the hardware we will need three components:

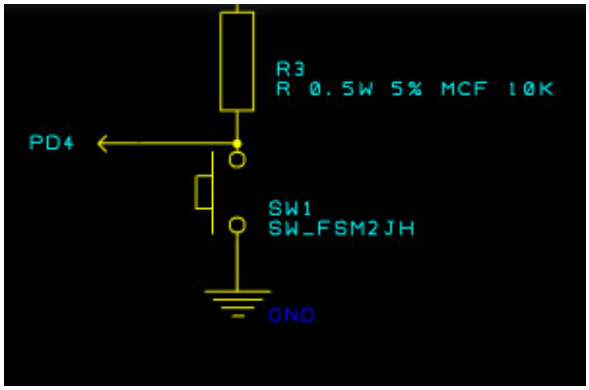
1. STM8S Circuit
2. Switch
3. LED

STM8S Circuit

For the STM8S we need the STM8S103F3 connected to 3.3V and ground. We will also need to add two ceramic capacitors, a 100nF between V_{ss} and V_{dd} (pins 7 and 9) and a 1uF between V_{ss} and V_{cap} (pins 7 and 8).

Switch Circuit

For the switch circuit we need a switch and a pull-up resistor. We will pull the input line to the STM8S high through the pull-up resistor. Pressing the switch will pull the input line low. This part of the circuit looks like this:

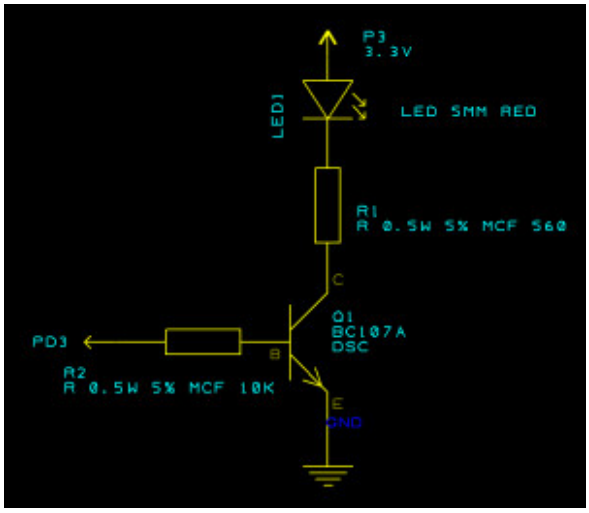


This circuit will have the pin held high until the switch is pressed. All we need to do is to detect the falling edge of the input to the STM8S and we have a way to detecting when the user presses the switch.

One thing to note here is that we will be ignoring switch bounce and so we are likely to get the odd spurious message. We will come back to this later.

LED Circuit

This circuit is a simple transistor circuit which is used to switch on a LED. By applying a low current to the base of a transistor we can switch on a larger current to drive the LED. A quick simple circuit looks like this:



By setting a GPIO pin connected to the base of the transistor to high we can turn on a LED.

Putting it all together, we connect the switch to PD4 and the LED to PD3.

Software

The software is a simple extension of the previous post on GPIO pins. Here we are going to use the same port (port D) for both input and output. The full code is:

```
1  #include <intrinsics.h>
2  #include <iostm8s103f3.h>
3
4  //
5  //  Process the interrupt generated by the pressing of the button on PD4.
6  //
7  #pragma vector = 8
```

```
//
//  Main program loop.
//
void main()
{
    //
    //  Initialise the system.
    //
    __disable_interrupt();
    PD_ODR = 0;             //  All pins are turned off.
    PD_DDR = 0xff;          //  All pins are outputs.
    PD_CR1 = 0xff;          //  Push-Pull outputs.
    PD_CR2 = 0xff;          //  Output speeds up to 10 MHz.
    //
    //  Now configure the input pin.
    //
    PD_DDR_DDR4 = 0;        //  PD4 is input.
    PD_CR1_C14 = 0;         //  PD4 is floating input.
    //
    //  Set up the interrupt.
    //
    EXTI_CR1_PDIS = 2;      //  Interrupt on falling edge.
    EXTI_CR2_TLIS = 0;      //  Falling edge only.
    __enable_interrupt();

    while (1)
    {
        __wait_for_interrupt();
    }
}
```

So let's start and break the program down. The first thing you will notice is that we initially configure all of the pins on port D as outputs as we have previously seen:

```
PD_ODR = 0;             //  All pins are turned off.
PD_DDR = 0xff;          //  All pins are outputs.
PD_CR1 = 0xff;          //  Push-Pull outputs.
PD_CR2 = 0xff;          //  Output speeds up to 10 MHz.
```

The next thing we do is to set up the system to allow PD4 as an input pin:

```
PD_DDR_DDR4 = 0;        //  PD4 is input.
PD_CR1_C14 = 0;         //  PD4 is floating input.
```

The final step of the configuration is to set the external interrupt for the port to detect the falling edge of the signal:

```
EXTI_CR1_PDIS = 2;      //  Interrupt on falling edge.
EXTI_CR2_TLIS = 0;      //  Falling edge only.
```

The final part of the main program is to wait for an interrupt. This is repeated infinitely:

```
while (1)
```

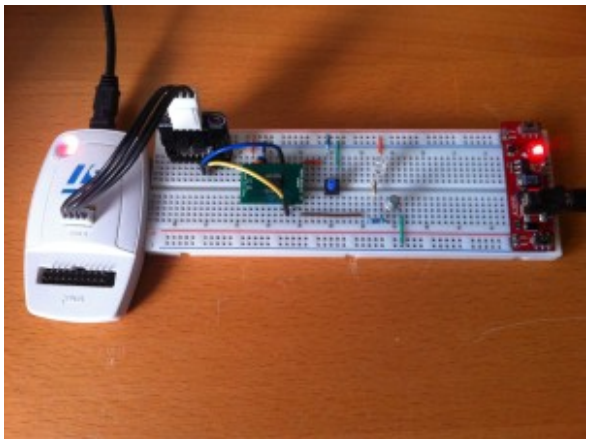
The interesting part of the problem is the interrupt routine as this is where the real work is actually done. The code is simple enough you just have to make sure that you declare the method correctly:

```
1  #pragma vector = 8
2  __interrupt void EXTI_PORTD_IRQHandler(void)
3  {
4      PD_ODR_ODR3 = !PD_ODR_ODR3;     //  Toggle Port D, pin 3.
5  }
```

The first two lines declare the method as an Interrupt Service Routine (ISR). The *#pragma vector = 8* tells the compiler which interrupt this method will be servicing. In this case, this method will be called to process the interrupts for Port D. We will look into ISRs a little more in a later post.

Putting it all Together

If we wire all of the hardware together on a breadboard we end up with something like the following:



and a quick video of it working:

Switch Bounce

up to a switch you see something like the following:

In this case the switch is held down and then released. Notice the spike; this is caused by the switch bounce. If the signal is large enough then the microcontroller will think that the switch has been pressed twice rather than once – in fact that is what happened in the above video. There are a few ways to solve this but I'll leave that as an exercise to the reader.

You can find the full source for the above project here. This application has been tested on tmy reference platform, the Variable Labs Protomodule and the STM8S Discovery board.

Compatibility

System	Compatible?
STM8S103F3 (Breadboard)	
Variable Lab Protomodule	
STM8S Discovery	

Tags: Electronics, LED, Software Development, STM8, The Way of the Register

Thursday, August 16th, 2012 at 8:36 pm • Electronics, STM8 • RSS 2.0 feed Both comments and pings are currently closed.

Comments are closed.

Pages

- About
- Making a Netduino GO! Module
- The Way of the Register
- Making an IR Remote Control
- Weather Station Project
- NuttX and Raspberry Pi PicoW

Support This Site

Buy me a coffee

Currently Reading