Demo 2: GPIO Interrupts

ECE3056 Fall 2016

October 3, 2016

1 Overview

This program flashes the LEDs of the STM32 Value Line Discovery board in response to button presses, demonstrating the setting and use of an interrupt handler. Each time the button is pressed, a counter in memory is incremented, and the number of flashes of the LEDs corresponds to the number of button presses that have been counted.

2 Operation

2.1 Interrupts

In Demo 1, the state of the GPIO pins corresponding to the pushbutton switches is read repeatedly in a loop, in a process known as *polling*. This is quite simple, but leads to a trade-off: in order to achieve quick responses to inputs, polling must be performed more frequently, but the more frequently polling is performed, the more processor time is spent performing the polling istead of doing other work. In battery-powered or other energy constrained systems, this problem accompanied by a need for the processor to always be executing code in order to respond to external inputs.

Interrupts solve both of these types of problems by providing a table of *vector*s, or pointers to code, at which the processor will begin executing immediately as a consequence of certain external events. Instead of repeatedly polling the state of each input device, the processor can simply set up a table of interrupts, and then selectively enable them whenever it is ready to receive input from a device of a given type. That input then causes the code pointed to by the vector to be executed. Once that code has finished, it executes an ordinary subroutine return and the processor resumes execution at its pre-interrupt program counter.

2.2 Interrupt Vector Table

The vector table at the beginning of the microcontroller's program flash memory, previously used to hold the initial program counter and stack pointer, is also

used to hold the interrupt vectors. [1] In addition to several special purpose vectors, there is a set of 87 general-purpose hardware IRQ (interrupt request) vectors, starting 64 bytes (16 entries) into the table.

2.3 Setting up the Interrupt

There are three pieces of hardware involved in setting up an interrupt on a GPIO pin:

- The AFIO (alternative function I/O) unit; selects which pins cause interrupts. [2]
- The EXTI (external interrupt) controller; selects which edge of input signals (rising or falling) trigger an interrupt. [3]
- the NVIC (nested vectored interrupt controller); the main interrupt controller for the ARM core. [4]

Once these have all been set up the `cpsie` instruction (change processor state: interrupt enable) is used to enable interrupts. [5] The program then enters an infinite loop. An interrupt handler responds to the rising edge of signals on Port A bit 0 (the "down" part of a button press) with an increment of a counter and a call to the `blink` subroutine from Demo 0.

3 Further Reading

The source is thoroughly annotated with comments, pointing to the pages in the ST Microelectronics manuals where further information can be found on each system used.

4 Exercises

The following exercises can be completed by modifying the demo code.

1. Instead of incrementing the counter once, update the counter repeatedly until the button is released.
2. Instead of an empty infinite loop, make the main loop repeated flash a number of times representing the number of times a button has been pressed. Remove the call to the `blink` subroutine from the interrupt handler.

[1] PM0056 p. 36
[2] RM0041 p. 124
[3] RM0041 p. 134
[4] PM0056 p. 118
[5] PM0056 p. 98

3. Make the interrupt happen when the button is released instead of when it is pressed. The code that must be changed is in the section where the EXTI registers are loaded.