Michael Savo, Sre Krishna

LAB 6

INTERRUPT LAB EXERCISE:

STACK USE AND TIMING BEHAVIOUR

Contents

1 Introduction

1.1 Lab overview

The interrupt demonstration uses an ISR to detect when a switch is pressed and increment a counter variable each time. The RGB LEDs are lit according to the three LSBs of the counter variable.

In this lab, you will evaluate the behaviour of a system with an interrupt. You will use the interrupt demonstration code.

2 Learning Outcomes

- Write an interrupt-driven program using an interrupt service routine (ISR).
- Analyse CPU timing behaviour via debug signals
- Use a debug tool to observe the state of the CPU when entering interrupt handling state.

3 Functions Used

- gpio_set(PIN pin, int value) - Sets the selected pin to the specified value
- leds_init() – Sets the 3 led pins to outputs and sets each led to 0 to start
- leds_set(int red_on, int green_on, int blue_on) – Turns on (with a 1) or off (with a 0) each led in sequence, from red to green to blue
- gpio_set_mode(PIN pin, PinMode mode) - Sets the output mode of a pin. Will primarily be either Input or Output modes for us
- gpio_set_trigger(PIN pin, TriggerMode trig, PinMode mode) - Sets the interrupt trigger for the specified pin. The TriggerMode can be Rising or Falling to check for either rising or falling edges respectively. You may also occasionally see None used to disable the interrupt. For PinMode, we will generally use PullUp to properly check for the interrupt.
- gpio_set_callback(PIN pin, void (*callback)(int status)) – Basically just associates the pin with a particular interrupt being used
- gpio_toggle(PIN pin) - Toggles a GPIO pin (aka a 0 becomes a 1 and a 1 becomes a 0)

4 Requirements

In this lab, we will be using the following hardware and software:

- **Keil µVision5 MDK IDE**
- **STM32 Nucleo-L552ZE-Q**
    - For more information, click here.
- **Oscilloscope**
    - Required to monitor the interrupt signals.

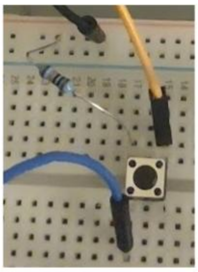
5 Hardware Setup

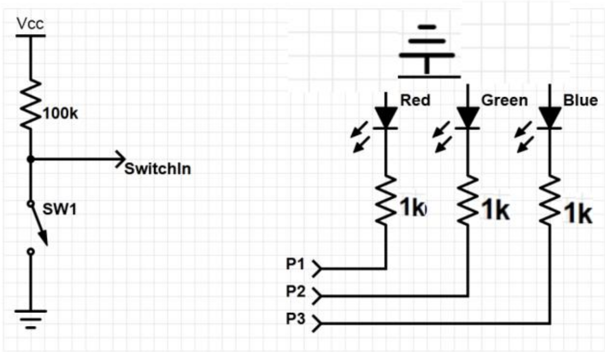
You should have a circuit built very similarly to the one shown on the slides:

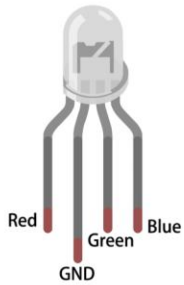
Orange wire to ground

Blue wire to a specified board input

Black wire to 5V

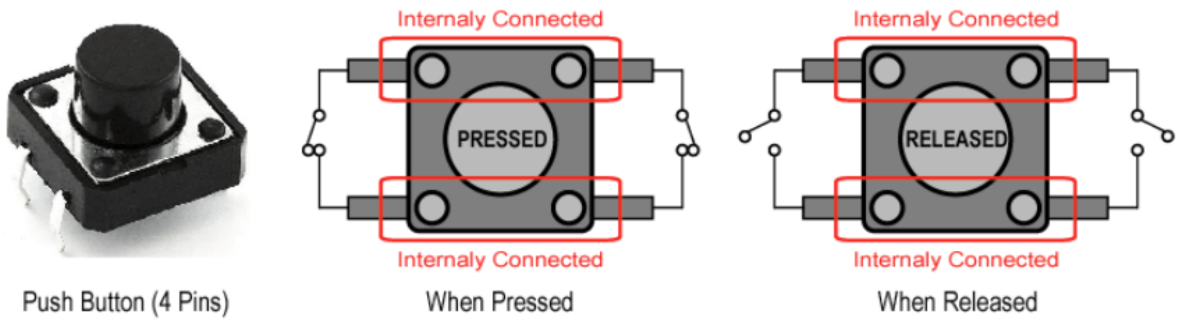






Pressing the button closes the circuit and connects us to ground!

Button wiring diagram:



Identify the 5V and various GND ports on your board. 5V should be the VCC for your switch/button in particular. For the switch/button, pay careful attention to which pins are connected by default – look this up for yourself if you are not familiar! Ultimately you want the 5V going through the 100k resistor and the switch input signal to be connected by default, and the connection to ground to only be made when you press the button.

For the LED, we'll be using a special RGB LED to allow us to configure the singular color that is visible. Once again, do a little digging (and see above) on the intended connections of this, which may also help you with the wiring for that part of the circuit. It should end up fairly similar to the one shown above.

Connect the switch signal to the GPIO port input on the MCU as shown in table below. Connect the debug signals and the switch signal to a logic analyzer or oscilloscope. This matches the pins used in the supplied code.

Signal /Connection Name	Description	Direction	MCU
SW1	Switch Input	Input to MCU	PD_15
DBG_Main	Main Thread Debug Output	Output from MCU	PD_11
DBG_ISR	ISR Debug Output	Output from MCU	PC_6
P_LED_R (P1)	Output to Red LED	Output from MCU	PA_5
P_LED_G (P2)	Output to Green LED	Output from MCU	PA_6
P_LED_B (P3)	Output to Blue LED	Output from MCU	PA_7

Please see the included Nucleo-L552ZE-Q pins legend (NUCLEO_L552ZE_pins.docx) for the pinout of the Arduino-included Zio connectors for CN7, CN8, CN9 and CN10.

6 Analysis

- Compile and load the provided code onto the board.
- Start the debugger.
- First, test and verify that your code is working as intended
  - The button is “bouncy”, so it may not always follow the exact pattern listed here, but the goal should be to have the following colors displayed in order every time you press the button a total of 8 times
    - Red
    - Green
    - Yellow
    - Blue
    - Pink
    - Teal
    - White
    - Off

After completing the above:

- Enable the disassembly window if it is not already visible (View->Disassembly Window)
- Set a breakpoint at start of handler function (button_press_isr).
- Run the program, and then press the switch SW1.

6.1 CPU Behaviour

6.1.1 CPU state when entering handler

Examine the stack and CPU registers with the debugger.

1. Complete the table below to show the values of the CPU registers and state information. You can also just take a screenshot/screenshots!

Register	**Value**	**Register**	**Value**	**Register/State**	**Value**
R0	0x0000000F	R8	0x00000000	xPSR	0x6900002A
R1	0x080007CD	R9	0xFFFFFFFF	MSP	0x200005F8
R2	0x00000000	R10	0x08000BF8	PSP	0x00000000
R3	0x00000000	R11	0x00000000	PRIMASK	0
R4	0x00000000	R12	0xE000ED0C	CONTROL	0x00
R5	0x00000000	R13 (SP)	0x200005F8	Mode	Handler
R6	0x42020000	R14 (LR)	0x080007CC	Privilege	Privileged
R7	0x00000000	R15 (PC)	0x6900002A	Stack	MSP

2. Open a memory window (View->Memory Windows->Memory 1) and enter the current value of SP as the address.
3. Place a breakpoint on the while(1) instruction in main. Continue program execution until that breakpoint occurs. Complete the table below to show what information is on the stack. Essentially, using the memory window, you should examine the stack in the region between the original SP and the new one that is currently showing in R13. Look for values that match either the old values of registers that you captured in step 1, or the new values of registers that you are currently able to see. Keep in mind that the endianness of the system (as previously discussed) impacts the order that the register's bytes are actually stored in memory!

Address	Value	Description
(SP) 0x200005F8	00 00 02 42 40 00 00 00 00 00	SP at earlier breakpoint
0x20000608	02 42 00 0C 02 42 0B 00 03 00	Matches R4
0x20000628	42020C00 42020C00 0003000B 0003000F	Matches R4
0x20000638	00000000 00000000 FFFFFFFF 08000BF8	Matches R9
0x20000658	0000001A 08000929 42020C00 0003000B	Matches R5
0x20000670	03 00 00 00 8F 02 00 08 84 D7	SP at while(1) in main

6.2 Timing

- Now connect the switch and debug signals (DBG_ISR and DBG_MAIN) to a logic analyzer or oscilloscope.
    - You will need 3 probes for this, one per signal.

   - Make sure each probe is connected to ground as well
- Disable the breakpoint in the handler function, and other breakpoints you may have added.
- Resume program execution.

6.2.1 Observe Overall CPU Timing Behaviour

Set the time base (horizontal scaling) and voltage (vertical scaling) of the oscilloscope so that you can clearly see the oscillating behavior of the DBG_MAIN signal and the (initially) still behavior of the DBG_ISR. Press the switch and capture a screenshot showing the switch signal, DBG_ISR, and DBG_MAIN – use the trigger functionality we discussed to aid you. It may take some tinkering to be able to accurately visualize each signal!

1. Is there any noticeable delay between the switch being pressed and the ISR running?
   a. There is a noticeable delay between the switch being pressed and the ISR running. Using the cursors, we found that it takes 24.80us.
2. Does the DBG_MAIN indicate that main stops running at any time?
   a. Yes, you can see that the clock goes from 1 to zero after the ISR ran. This means main has stopped running.