***Introduction to Robotic Systems Course***

**LAB 02**

**Interrupt Lab Exercise:**

**Stack Use and Timing Behavior**

Contents

[1 Introduction 1](#_Toc31027271)

[1.1 Lab Overview 1](#_Toc31027272)

[2 Requirements 1](#_Toc31027273)

[3 Task: Software 1](#_Toc31027274)

[4 Task: Hardware Setup 1](#_Toc31027275)

[5 Task: Analysis 3](#_Toc31027276)

[5.1 CPU Behavior 3](#_Toc31027277)

[5.1.1 CPU State when Entering HANDLER 3](#_Toc31027278)

[5.1.2 CPU State after Entering HANDLER 3](#_Toc31027279)

[5.1.3 CPU State after Exiting Interrupt 4](#_Toc31027280)

[5.2 Timing 4](#_Toc31027281)

[5.2.1 Observe Overall CPU Timing Behavior 4](#_Toc31027282)

[5.2.2 Observe Detailed CPU Timing Behavior 4](#_Toc31027283)

[5.2.3 Observe Even More Detailed CPU Timing Behavior 5](#_Toc31027284)

[5.2.4 Pre-emption of Main Code 5](#_Toc31027285)

# Introduction

## Lab Overview

This lab exercise presents an example code to introduce the concept and application of interrupts in micro-controller operation using an interrupt service routine (ISR) to service interrupts received from a dual in-line package (DIP) switch. In the code supplied for this lab, the ISR increments a counter whenever an interrupt is received. LEDs connected to the micro-controller output the state of the counter. Also, in this lab, the states of the processor during context switching and interrupt execution is examined using the Keil debug tool.

# Requirements

The following hardware and software are required to complete this lab:

* **Hardware**
  + OpenCR1.0 Microcontroller Board and ULINK-ME debugger
  + Dip Switch, LEDs (x5), 270Ω Resistors (x3), Breadboard and jumper cables
  + Logic analyzer
* **Software:** Keil MDK. Version v5.28 (course was developed with this version)

# Task: Software

We have provided the complete source code for this lab.

# Task: Hardware Setup

Use Table 1, the schematic shown in Figure 1, and the OpenCR1.0 board layout in Figure 2 as a guide to connect the switch, LEDs, and resistors to the GPIO port on the MCU.

Connect the debug signals (**DBG\_Main** and **DBG\_ISR**) and the switch signal to a logic analyzer or oscilloscope. The signal **DBG\_ISR** will indicate execution of the handler, while the signal **DBG\_Main** will indicate the program is out of handler mode and is in a loop.

Table 1: Signals and connections

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Signal Name | Description | Direction | MCU | Arduino Pin on OpenCR1.0 |
| SW1 | Switch Input | Input to MCU | PB4 | 3 |
| DBG\_Main | Main Thread Debug Output | Output from MCU | PC1 | 7 |
| DBG\_ISR | ISR Debug Output | Output from MCU | PA2 | 6 |
| Red LED | Counter state | Output from MCU | PA8 | 5 |
| Green LED | Counter state | Output from MCU | PC6 | 1 |
| Blue LED | Counter state | Output from MCU | PC7 | 0 |

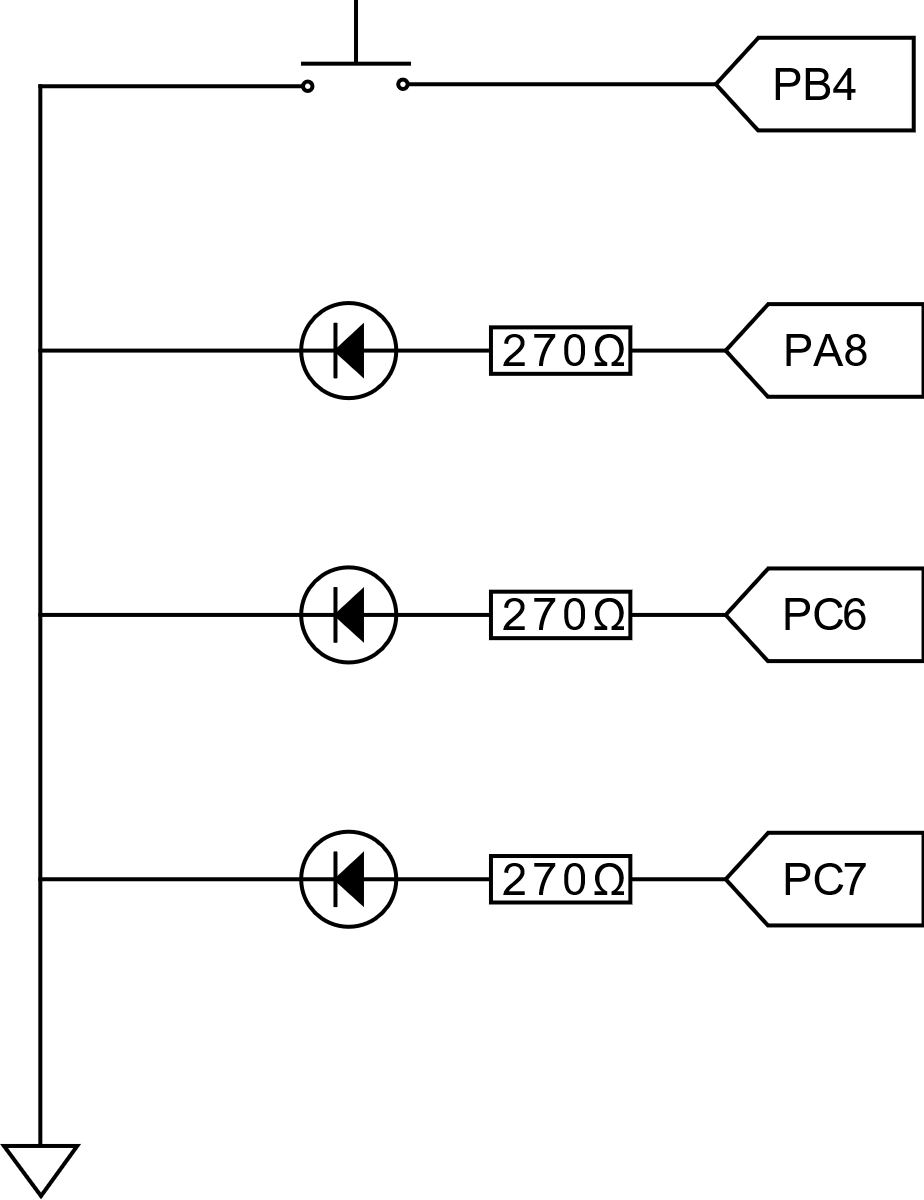


Figure 1: Schematic of LEDs and switch connected to the OpenCR1.0.



Figure 2: Layout of OpenCR1.0 Highlighting Arduino Pins.

# Task: Analysis

* Open the supplied project folder and run the Keil MDK project.
* Compile and load the code onto the board.
* Start the debugger session.
* Enable the disassembly window (**View**->**Disassembly Window**)
* Set a breakpoint at start of handler function.
* Run the program and then press the switch SW1

## CPU Behavior

### 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Register** | **Value** | **Register** | **Value** | **Register/State** | **Value** |
| R0 | – | R8 | – | xPSR | – |
| R1 | – | R9 | – | MSP | – |
| R2 | – | R10 | – | PSP | – |
| R3 | – | R11 | – | PRIMASK | – |
| R4 | – | R12 | – | CONTROL | – |
| R5 | – | R13 (SP) | – | Mode | – |
| R6 | – | R14 (LR) | – | Privilege | – |
| R7 | – | R15 (PC) | – | Stack | – |

1. Complete the table below to show what information is on the stack. Open a memory window (**View**->**Memory Windows**->**Memory 1**) and enter SP as the address. Right-click on the window and specify **Unsigned**->**Int** as the display format.

|  |  |  |
| --- | --- | --- |
| Address | Value | Description |
| (SP) | – | – |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |

### CPU State after Entering HANDLER

Step one line with F11, then 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Register** | **Value** | **Register** | **Value** | **Register/State** | **Value** |
| R0 | – | R8 | – | xPSR | – |
| R1 | – | R9 | – | MSP | – |
| R2 | – | R10 | – | PSP | – |
| R3 | – | R11 | – | PRIMASK | – |
| R4 | – | R12 | – | CONTROL | – |
| R5 | – | R13 (SP) | – | Mode | – |
| R6 | – | R14 (LR) | – | Privilege | – |
| R7 | – | R15 (PC) | – | Stack | – |

1. Complete the table below to show what information is on the stack. Open a memory window (**View**->**Memory Windows**->**Memory 1**) and enter SP as the address. Right-click on the window and specify **Unsigned**->**Int** as the display format.

|  |  |  |
| --- | --- | --- |
| Address | Value | Description |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |
| – | – | – |

### CPU State after Exiting Interrupt

Step out of the handler function (ctrl + F11) and verify the return address.

## Timing

* Now connect the debug signals to a logic analyzer or oscilloscope if you have not done so already.
* Disable the breakpoint in the handler function and any other breakpoints you may have added.
* Resume program execution.

### Observe Overall CPU Timing Behavior

Use the falling edge of the Switch input to trigger the data capture/sweep.

Set the time base of the logic analyzer (or oscilloscope) so that a switch press covers about one fourth of the screen. Capture a screenshot showing the switch signal, **DBG\_ISR**, and **DBG\_MAIN**.

1. How long was the switch pressed down?
2. Is there any noticeable delay between the switch being pressed and the ISR running?
3. Does the **DBG\_MAIN** indicate that main stops running at any time?

### Observe Detailed CPU Timing Behavior

Now zoom in so that the screen displays about 100 µs, centered on the ISR.

1. How long is the **DBG\_ISR** signal asserted?
2. Is there any noticeable delay between the switch being pressed and the ISR running?
3. Does the **DBG\_MAIN** signal indicate that main stops running at any time?

### Observe Even More Detailed CPU Timing Behavior

Now zoom in so that the screen displays about 10 µs centered on the ISR.

1. How long is the **DBG\_ISR** signal asserted?
2. Is there any noticeable delay between the switch being pressed and the ISR running? How many clock cycles does this correspond to? How does this compare to what you would expect?
3. Does the **DBG\_MAIN** signal indicate that main stops running at any time? If so, calculate for how long.

### Pre-emption of Main Code

Now zoom in so that the screen displays about 100 us centered on the ISR.

1. For how long is the main function delayed? First measure the pulse width of the **DBG\_MAIN** output signal before the switch is pressed. Then, measure the pulse width when main is pre-empted by the ISR. The difference indicates the total pre-emption time.
2. How long is the total pre-emption in comparison with the duration of the **DBG\_ISR** signal? If the two times aren’t the same, explain why.