***Efficient Embedded Course***

**LAB 3**

**INTERRUPT LAB EXERCISE:**

**STACK USE AND TIMING BEHAVIOUR**

Note. The figures shown in solutions may vary subject to different experimental environments

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc87594141)

[1.1 Lab overview 1](#_Toc87594142)

[2 Learning Outcomes 1](#_Toc87594143)

[3 Requirements 1](#_Toc87594144)

[4 Hardware Setup 2](#_Toc87594145)

[5 Analysis 2](#_Toc87594146)

[5.1 CPU Behaviour 3](#_Toc87594147)

[5.1.1 CPU state when entering handler 3](#_Toc87594148)

[5.1.2 Solution 3](#_Toc87594149)

[5.1.3 CPU State After Exiting Interrupt 4](#_Toc87594150)

[5.2 Timing 4](#_Toc87594151)

[5.2.1 Observe Overall CPU Timing Behaviour 4](#_Toc87594152)

[5.2.2 Solution 4](#_Toc87594153)

[5.2.3 Observe Detailed CPU Timing Behaviour 5](#_Toc87594154)

[5.2.4 Solution 5](#_Toc87594155)

[5.2.5 Observe Even More Detailed CPU Timing Behaviour 5](#_Toc87594156)

[5.2.6 Solution 6](#_Toc87594157)

[5.2.7 Pre-emption of Main Code 6](#_Toc87594158)

[5.2.8 Solution 6](#_Toc87594159)

# Introduction

## Lab overview

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

# 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.

# Requirements

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

* **KEIL µVision5 MDK IDE**
  + Please check the Getting Started with KEIL guide on how to download and install it.
* **STM32F407G-DISC1**
  + For more information, click [here](https://www.st.com/en/evaluation-tools/stm32f4discovery.html).
* **Logic Analyzer or Oscilloscope** 
  + Required to monitor the interrupt signals

# Hardware Setup

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.

Table . Signals and connections

|  |  |  |  |
| --- | --- | --- | --- |
| Signal Name | Description | Direction | MCU |
| SW1 | Switch Input | Input to MCU | PA0 |
| VSS | Ground |  |  |
| DBG\_Main | Main Thread Debug Output | Output from MCU | PB7 |
| DBG\_ISR | ISR Debug Output | Output from MCU | PB6 |

A picture containing text, electronics, circuit

Description automatically generated

Figure . DiscoveryF4 I/O connectors.

# Analysis

* Compile and load the code onto the board.
* Start the debugger.
* 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 Behaviour

### 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) |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### Solution

|  |
| --- |
| Saved R0 |
| Saved R1 |
| Saved R2 |
| Saved R3 |
| Saved R12 |
| Saved LR |
| Saved PC |
| Saved xPSR |

Graphical user interface, text, application

Description automatically generated

### 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 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#### Solution

Graphical user interface, text, application

Description automatically generated

Note that R4 and LR have been pushed onto the stack.

### CPU State After Exiting Interrupt

Single step out of the handler, verify return address.

## Timing

* Now connect the debug signals to a logic analyzer or oscilloscope.
* Disable the breakpoint in the handler function, and other breakpoints you may have added.
* Resume program execution.

### Observe Overall CPU Timing Behaviour

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 screen shot 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?

### Solution

**A picture containing diagram

Description automatically generated**

The switch was held down here for about 338 ms.

There is no delay visible.

### Observe Detailed CPU Timing Behaviour

Now zoom in so that the screen displays about 100 us, centred 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?

### Solution

A screenshot of a computer

Description automatically generated with medium confidence

DBG\_ISR is asserted for a little over 0.75 us.

There is a very slight delay visible.

Main does stop running briefly.

### Observe Even More Detailed CPU Timing Behaviour

Now zoom in so that the screen displays about 10 us centred 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.

### Solution

A picture containing diagram

Description automatically generated

DBG\_ISR is asserted for about 0.75 us.

There is a 4 us delay between the switch being pressed and the ISR running. We would expect 16 cycles latency from Switch until ISR execution begins, plus the overhead of the driver.

### 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 preempted by the ISR. The difference indicates the total preemption time.
2. How long is the total preemption in comparison with the duration of the DBG\_ISR signal? If the two times aren’t the same, explain why.

### Solution

A picture containing text, monitor, indoor, electronics

Description automatically generated

Normally toggles DBG\_Main every 4.5 us

When ISR runs, DBG\_Main’s pulse is 8.625 us wide

So main is preempted for 8.625 us - 4.5 us = 4.125 us

But the body of ISR only takes about 0.75 us

So there is an additional 3.375 us from interrupt response overhead – driver abstraction plus stacking and unstacking registers, etc.