Lab Sheet 3: Timing

1 Aims

The aims of this lab are to:

- Understand how to measure three timing properties of the cyclic system.
- Understand how the MCU responds to interrupts.

1.1 Activities

Perform the following tasks:

1. Activity 0: Download the sample project and review the code carefully.
2. Activity 1: Modify the given code and use the oscilloscope to measure the cycle time and the proportion of the cycle spent executing code versus waiting.
3. Activity 2: Modify the given code and use the oscilloscope to measure how quickly a GPIO interrupt can handle an input button.
4. Activity 3: Estimate the timing by counting assembly code instructions.

Note that activities 1 & 2 do not depend on each other and can be done in either order. The code changes needed are minimal – you just need to understand what you are doing.

Assessment. If you wish, you can present work on this lab exercise for assessment. The arrangements differ from other labs as there is no code marking.

- **Answer the questions on the QMPlus quiz.** *You are recommended to look at the questions about the given code as you go along as answering them may help you complete the rest of the lab.*
- **Demonstration in the lab.** Ask one of the demonstrators to see your code working during the scheduled lab times. For this lab, the demonstration can be done in two parts and counts for 4 marks (double the usual 2 marks).
- **Code assessment.** There is no code assessment for this lab.

2 Activity 0: Download and Review Sample Project

Download the sample project. The project has the following features:

- The design is a 'cyclic system' with two tasks.
  a) One task flashes the Red LED, with on and off times of 1 sec.
  b) The second task checks for the variable set in the GPIO interrupt handler. When it is shows the button has been pressed, the blue LED changes
- Code is included to initialise two GPIOs, However, these GPIOs are not used in the given code. These GPIOs are:

Pin Number	Macro Name
PTE23	OUT1
PTE30	OUT2

- Code is included to configure two buttons using an interrupt
  a) Button B5. In the interrupt handler a signal (i.e. a variable) is set. This button is designed for use with an interrupt as it includes hardware components to debounce transitions.
  b) Button B1. This button is usually not used with an interrupt as without a hardware circuit to debounce, multiple interrupts might occur. In the provided code, the handler is empty.

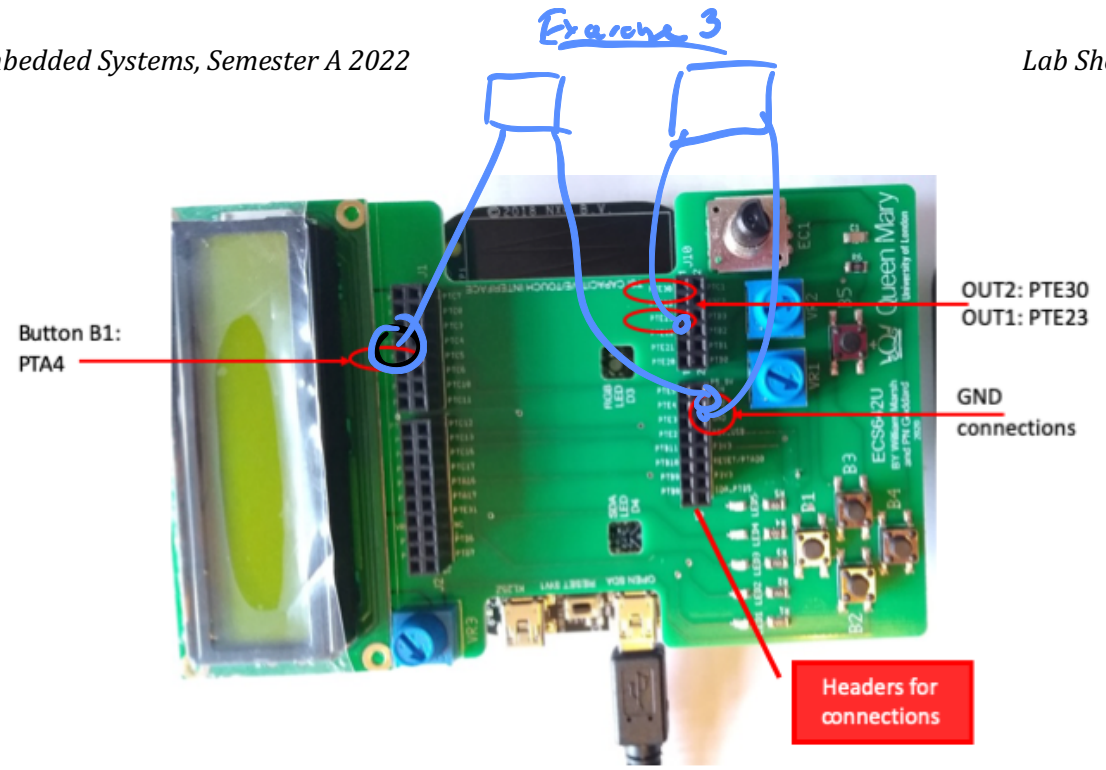

Figure 1: Location of the Connections on the Shield

3 Activity 1: Timing the Cyclic System

Make the following two measurements using an oscilloscope:

1. Measure the cycle time. Check whether the cycle time is exactly 10ms as measured by the oscilloscope.
2. Measure the proportion of the cycle time used to execute the code in the cyclic system.

The following notes explain how you can do this. See also Appendix 1 for help on the connecting and using the oscilloscope.

3.1 Measurement 1: Cycle Time

Use the GPIO output OUT1, which is on PTE23.

- Change the given program to toggle this output (use the 'toggle output register') in the main loop. See example in Task 2 of the given code.
- Connect this output to the oscilloscope to the red oscilloscope lead (see Figure 1)
- Connect the KL25Z GND to the black oscilloscope lead (see Figure 1).
- View the output on one channel of the oscilloscope and measure the cycle time.

You expect to see a periodic square wave of period 2 x 10ms, as in Figure 2 below. Check whether the cycle time is exactly 10ms as measured by the oscilloscope.

3.2 Measurement 2: Proportion Spent in Code Execution

Most of each 10ms cycle is spent waiting, representing 'unused' processing time. The proportion of the overall cycle time spent executing the code can be estimated by changing a GPIO at the start and end of the code section of the cycle.

1. Edit the code to set the GPIO OUT2 output high before task 1 and clear again after task 2. Make these changes directly in the main loop to minimise overheads.
2. Use the oscilloscope to measure the time spent executing code.
3. As before, connect OUT2 to the oscilloscope. Ensure the oscilloscope GND is connected to KL25Z GND (see Figure 1).

4. You should see a brief pulse occurring every 10ms. Measure the width (i.e. duration) of this pulse (remember the units).
5. Calculate the execution time as a proportion of the 10ms cycle.

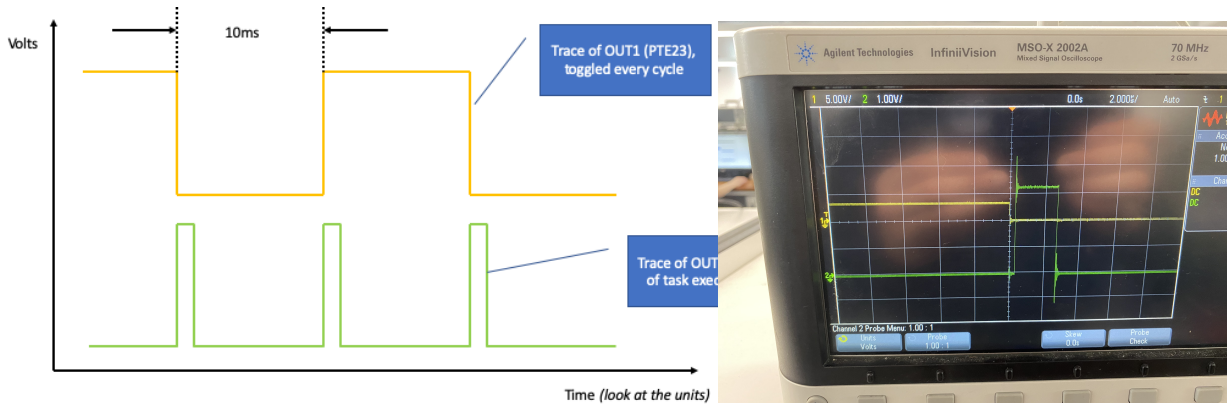

Figure 2: Oscilloscope Traces for Measurements 1 and 2 (not to scale)

You are expecting to see a short pulse, with width equal to the time spent executing code (other than the wait loop). Calculate the execution time as a proportion of the 10ms cycle. As the oscilloscope has two channels, it is possible to display both waveforms, as shown in the picture below. However, this diagram is not to scale and to measure the execution time of the code, you will need to zoom in more. Does the time vary? 2.3 microseconds No because there are no condition, same set of instruction are recorded.

Demonstrate this part of the work.

4 Activity 3: Measuring the Time to Handle an Interrupt

In this part of the lab, we press button B1 causes an interrupt. In the interrupt handler, we toggle OUT1. By observing both the button input and OUT1 on different channels of the oscilloscope, we can measure how quickly we are able to respond to an interrupt.

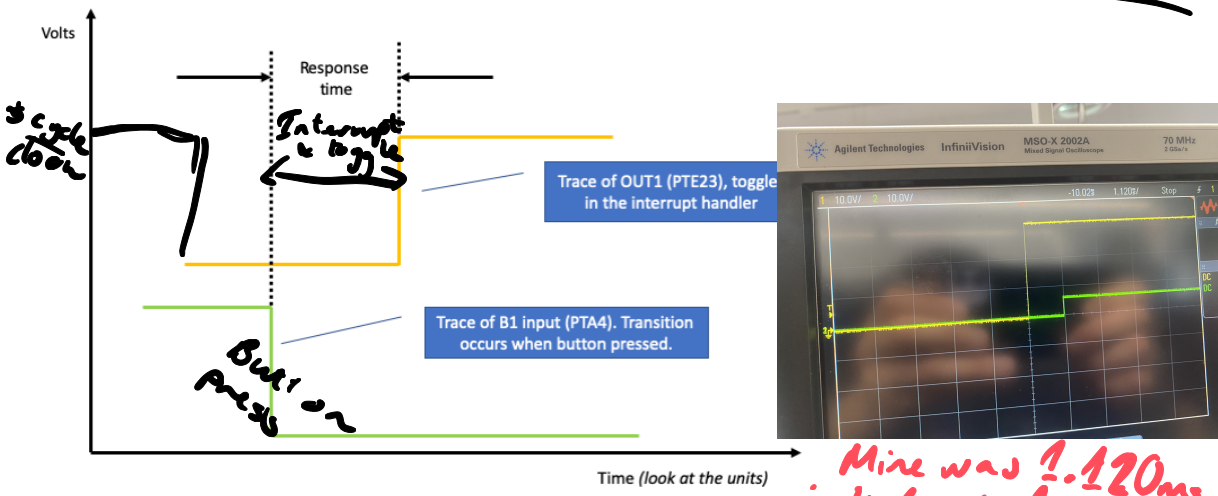

Figure 3: Measuring Interrupt Response Time

Mine was 1.120ms initially but it can go to 4ms.

- Modify the interrupt handler to toggle the GPIO OUT1 when button B1 is pressed. Button B5 continues to operate the blue LED as before.
- On one channel in the oscilloscope, connect OUT1 (this may already be connected). As before, ensure the oscilloscope GND is connected to the micro-controller.
- On the other channel of the oscilloscope, connect the PTA4 (which is the GPIO pin used for button B1) – see Figure 1.

- Trigger the oscilloscope on the falling edge of the button – the same edge that creates the interrupt – and display both the button and the GPIO as traces on the oscilloscope.
- Look at the delay between the button going low and the change in OUT1. This time includes both the time to switch to the interrupt and the time to set the GPIO output.

5 Activity 4: Counting Instructions

We can also estimate the time a program is expected to take by counting the instructions in assembly code. You can view this in the debugger.

- Most instructions take 1 CPU cycle
- Writing to a peripheral takes two cycles (the bus used is slower)

KL25Z Performance

Clock speed (default)	20.97 MHz (the exact figure is 5 x 2^{22} Hz)
Cycle time	47.7 ns
Instruction per µs	Approximately 21

5.1 Program Execution

Count the instructions in the two tasks called in the cyclic system. The number of instructions depends on the path take through the code, as more instructions are executed when state transitions occur.

- Assuming no transitions occur, we get the minimum number of instructions
- Assuming transitions occur in both tasks on the same cycle, gives the maximum number of instructions
- The measurement itself uses some instructions, with no variation.

Complete the following table

Task	Minimum Count			Maximum Count		
	Instructions	**Peripheral I/O**	**Total**	**Instructions**	**Peripheral I/O**	**Total**
Task1						
Task2						
Msrmnt				n/a	n/a	n/a

Record the calculated and measured times (include units):

- Measured =
- Calculated minimum = total count x 47.7ns =
- Calculated maximum = total count x 47.7ns =

5.2 Interrupt Time

Interrupt processing has some overhead before the instructions in the ISR are executed. We can estimate this overhead.

- ISR instruction count (remember peripheral I/O counts twice) =
- ISR time = 47.7ns x count =
- Overhead time = Measured time – ISR Time =
- Overhead cycles = Overhead time / 47.7ns =

6 Appendix: Notes on the Use of the Oscilloscopes

There are two models of oscilloscope in use in the lab. For some brief introduction, see the page https://qmplus.qmul.ac.uk/mod/page/view.php?id=1413151

The following picture shows how we connect the oscilloscope. You need:

- Oscilloscope leads, with 'crocodile' clips.
- Wire links that plug into the headers.