H63ECH Coursework

Academic year 2017/2018

General considerations

- The coursework involves assembly and C language programming for a PIC microcontroller and debugging the programs using computer simulations and dedicated hardware. This development can take place using any computer with MPLAB X IDE installed (including university computers). The hardware (PICkit2 + 44 pin development board + USB A to mini B cable) is loaned to every student registered on the module for use either in the lab or at home.

- The coursework accounts for 30% of the module mark, thus you should plan to spend around 36 hours in total working on it. This is roughly equivalent to one week, working full time.

- The best way to approach this work is to complete the tasks outside of the lab sessions, then seek help on points you do not understand, and get the tasks ticked off when you are in the lab sessions. If you only attempt the tasks in the lab sessions you will not be able to complete all of the tasks.

- Do NOT complete all of the tasks and only then show your work to a demonstrator. Please have the tasks ticked off as soon as you have completed them. This means both that the marking of tasks is spread out over a longer period and it is easier for the demonstrators to manage, but importantly for you it also means that you get immediate feedback on the work that you are doing and hence can improve your work for future tasks.

- You may find it helpful to carry out the **44Pin Demo Board Lessons** before starting the coursework, but do not spend too much time on them.

- There are 7 tasks nominally worth 10 marks each, plus a final task worth 30 marks, for a total mark out of 100.

- The first three tasks 1-3 are introductory tasks – all the help for programming them will be provided from the module convener; you will not allowed to submit any other tasks if they not finish these. Completion of these tasks will make you familiar with the basics of both C and assembly embedded programming and their relative strengths and weaknesses.

- Tasks 4-5 are intermediate, and once completed in full you can use the most frequently used PIC16 ISA instructions.

- Task 6 involves programming peripheral devices of the microcontroller; task 7 involves programming the interrupts. These more advanced tasks are required to complete the coursework to the first class standard.

- Task 8 is an in depth assembly program that requires a reasonable amount of effort and will push you to produce a well-designed complex piece of code.

- The tasks are individualised via the student number. Out of 7 digits the last 5 digits are used.

- To achieve full marks for the code part of each task, **please show your working code to a demonstrator – they will take a record for the assessment.** If the code you present does not work, or needs some improvement then the demonstrator will ask you to make those changes before ticking off the task. The underlying meaning of this is that the aim of the exercises is not just to have you complete the tasks, but produce good quality code as well. There is no penalty for having the demonstrator look at the same task multiple times.

- If you did not manage to complete and have ticked off any of the tasks, you should state in your report what was achieved and what was not.. Some marks may be awarded for the submitted code based on the judgement of the module convener. If the code for any particular task does not work, the marks for the questions within this task will not be awarded.

- **All students have signed the undertaking regarding plagiarism and collusion. Within the context of this coursework ANY transfer of information electronically or printed on paper (snippets of code, diagrams, text, files etc) will be considered an offence, and both parties will be prosecuted for this. You may talk to each other about tasks, but do not directly share work nor allow another student to access your work. I have had cases most years where one student took work from their "friend" without the friend knowing. Please do not let this happen to you.**

- Important: Before you submit your code, please "clean" each of the project tasks from within MPLAB. This can be done by right clicking on the project in the Projects tab and choosing "Clean". This substantially reduces the size and number of files that I have to deal with when marking.

Deadlines and Submission

Tasks

There are effectively 8 weeks to work on the coursework in the lab (which gives 8 hours out of the above suggested 36). The final deadline for submission of the short report and completed code is given below. There are additional deadlines for completion of the tasks and having them ticked off. You will not be able to have your tasks ticked off after these dates, but partial marks may still be awarded for code submitted as part of your final zip file.

Deadline for Tasks 1-3:	30th October
Deadline for Tasks 4-5:	20th November
Deadline for Tasks 6-7:	4th December
Deadline for Task 8:	11th December

These are deadlines, not targets! Please work to complete the tasks well in advance.

Report and Code

The submission deadline for the completed coursework report is Monday 11th December 2017 at 3 pm. The report should be submitted online via moodle.

The archived directories of all fully completed and attempted code tasks should be submitted at the same time via moodle as a zip.

Notes

Even the simplest embedded program requires some common actions that can be implemented as subroutines. In your coursework you will be able to use the following subroutines:

Delay subroutines

Assembly (written by me)

`Del_ms` provides delay of precisely 1 ms
`DelWms` provides a delay approximately W ms
`DelWds` provides delay of approximately W×0.1s

C (provided by the XC8 compiler)

`__delay(x)` provides delay of precisely x Tcy
`__delay_us(x)` provides delay of precisely x µs
`__delay_ms(x)` provides delay of precisely x ms

Button Operation with Debounce

Assembly (written by me)

`call DeBounce` no parameters required, call it before checking the button
`skipPre` skip the following instruction if the button is pressed (no call required)
`skipRel` skip the following instruction if the buttonis released (no call required)

C (written by me)

`tmp=ReadSw();` call this function to obtain the button value (1 – not pressed, 0 – pressed)

Input of variables

These functions allow you to input a value using the potentiometer. They turn on LED4, then allow you to select one choice out of four by rotating the potentiometer. The currently selected value will be shown on the LEDs LD3...LD0 and LD4 will be on to show that you can now input a value. When you have selected the desired value, press the button to confirm it. After you have released the button, the selected value will blink on the LEDs, and then the program execution will continue. The subroutines keep LD7 and LD6 in the same state as they were before calling the subroutine.

Two subroutines are provided, you may wish to modify them or make your own version for some of the tasks.

`Select4` this subroutine returns the selection as a single bit in W (bit 0 is set for choice 0; bit 1 for choice 1 etc.) for assembly, or as the returned value when using C.
`SelectB` this subroutine returns the selection as a 2 bit binary value in W.

You can see examples of using these subroutines in the files Examples.INC and examples.c.

Introductory tasks (full help will be provided with these)

For these tasks you will need to use the last digit of your student ID number to substitute two parameters in the following calculation

`(<C> × A) <operation> B`

Where the 2 bit variables A and B are entered by the user according to the following table:

	0	1	2	3	4	5	6	7	8	9
<C>	6	5	3	7	6	5	3	7	6	5
<operation>	ADD	IOR	XOR	ADD	IOR	XOR	ADD	IOR	XOR	ADD

For example, the student with the ID number 1234567 (last digit 7) will use the following calculation

`(7×A)   |   B`

Task 1 (assembly)

Starting from the template **`Task1`**, modify the look-up-table provided and add the missing pieces to the code to implement the following actions

Switch on **`LD7`** (input **`A`**)
Call **`SelectB`** to input a 2 bit value for **`A`**
Store **`A`** in a temporary variable
`(the above code is provided in the template)`

Switch off **`LD7`**, switch on **`LD6`** (input **`B`**)
Call **`SelectB`** to input a 2 bit value for **`B`**
Using **`A`** and **`B`** as a 4 bit value, select the result out of a look up table
Display the result on **`LEDs`**, additionally switch on both **`LD7`** and **`LD6`**
Stop

In the report answer the following questions (5 marks for working code):

Q1.1 How would your code change if the variables A and B were 4 bit each (2 marks)?

Q1.2 How would your code change if the variables A and B were 8 bits each (3 marks)?

Task 2 (assembly)

Copy the project **`Template_ASM`** and rename it to **`Task2`**. Develop your code in the file **`PROGRAM.ASM`** to implement the actions below:

> Switch on **`LD7`** (input **`A`**)
> Call **`SelectB`** to input a 2 bit value for **`A`**
> Store A in a temporary variable
> (the above steps are the same as for task 1 and should be copied into the task 2 code)
>
> Switch off **`LD7`**, switch on **`LD6`** (input **`B`**)
> Call **`SelectB`** to input a 2 bit value for **`B`**
> Using **`A`** and **`B`** as 2 bit values, calculate the result using PIC instructions
> Display the result on **`LEDs`**, additionally switch on both **`LD7`** and **`LD6`**
> Stop

In the report answer the following questions (6 marks for working code):

Q2.1 How would you code change if the variables A and B were 4 bit each (2 marks)?

Q2.2 How would your code change if the variables A and B were 8 bits each (2 marks)?

Task 3.1 (C)

Copy **`Template_C_ECH`** to the same directory and rename it to **`Task3.1`**, calculate the result using a look up table in C.

Switch on **`LD7`** (input **`A`**)
Call **`SelectB`** to input a 2 bit value for **`A`**
Store **`A`** in a temporary variable, switch off **`LD7`**, switch on **`LD6`** (input **`B`**)
Call **`SelectB`** to input a 2 bit value for **`B`**
Using **`A`** and **`B`** as a 4 bit index value, select the result out of an array (LUT in C)
Display the value on the LEDs and switch on both **`LD6`** and **`LD7`**

(2 marks for working code)

Task 3.2 (C)

Copy **`Template_C_ECH`** to the same directory and rename it to **`Task3.2`**, calculate the result using instructions in C.

Switch on **`LD7`** (input **`A`**)
Call **`SelectB`** to input a 2 bit value for **`A`**
Store **`A`** in a temporary variable, switch off **`LD7`**, switch on **`LD6`** (input **`B`**)
Call **`SelectB`** to input a 2 bit value for **`B`**
Using **`A`** and **`B`** as 2 bit values, calculate the result using PIC instructions
Display the result on the **`LEDs`** and switch on both **`LD6`** and **`LD7`**

In the report answer the following questions (2 marks for working code):

Q3.1 Fill in the following table (4 marks):

For "Execution time", you should calculate the time taken for each task to complete the code from immediately after the value for B has been obtained to just after the LEDs and LD6 and LD7 have been turned on.

	Task1	Task 3.1	Task2	Task3.2
Execution time, T_{cy}				
Code size, Instructions				

Q3.2 Based on this result and your knowledge from the module, comment on the relative advantages of C and assembler (2 marks).

Intermediate tasks (help will be provided for assembly only).

These tasks are relevant to embedded programming that uses the superloop to repeat the same actions over and over again. Please complete them in **both** assembly and C, naming your directories appropriately, e.g. Task4A and Task4C. You should be making appropriate use of functions/subroutines for this work.

Task 4 – Digit display

Display last four digits of your student number

(for the student number 1234567 the student will need to display "7" for an input of 0, "6" for an input of, "5" for an input of 2, and "4" for an input of 3. If the value in your student ID is 8 or 9, use a value of 7 instead. This task is about using conditional execution statements of the PIC16 ISA) The following code is to be placed inside the superloop

> Call **`Select4`** to select the digit to display
> Display the selected digit on **`LD7..LD5`** if the selection is in position 1, 2 or 3
> If the selected digit is in position 0, display the value using the Morse code on **`LD7`**
> (please use 0.3s for the duration of the single dot: **`movlw 3;  call DelWds`**)
> Delay 3 s

In the report answer the following questions (5 marks for the working assembly code, 3 marks for the working C code):

Q4.1 State size of your assembly code (the lower the better, up to 2 marks are awarded).

Task 5 – Bit banging PWM

Control brightness of an LED using bit banged PWM, with the brightness based on your student number.

Use the largest and smallest values from the last four digits of your student number to choose some of the relative brightness levels that you will create. For the student number 123**4567** you would select digits 4 and 7.

Produce code to implement the following tasks:

When the program starts

A. Starting with the button released, use Select4 to select one out of the four values 'X' that relate to the duty factor of the bit banged PWM, using the potentiometer;

B. When you press the button to confirm the selection, as long as you keep the button pressed the bit-banged PWM routine should run, setting the brightness of LD0 based on the chosen value. LED LD1 should be set to be on to allow the brightnesses to be compared.

C. To choose another brightness, release the button and return to A above.

To implement this user scenario, the code should perform the following actions in the superloop:

```
Obtain value from Select4
    if the user selected choice 3, set X=D'12';
    if the user selected choice 2, set X to the largest value (in the above example X=7); if the user selected choice 1, set X to the smallest digit (in the above example X=4);
    if the user selected choice 0, set X=D'1'.

Turn LD0 off, LD1 on
Provide a loop of 28 iterations {
    If the iteration number equals X, switch LD0 on
    (hint: use SUBLW or XORFW between X and the loop counter, then test Z bit of STATUS to detect their equality)
    Delay 100 us ( call Del_100us )
}
```

In the report answer the following questions (4 marks for the working assembly code, 2 marks for the working C code):

Q5.1 By observing the brightness of the LED at different duty factors note that it depends on the duty factor non-linearly. Suggest why. (1 mark for every reasonable suggestion)

Q5.2 Estimate from your code the resultant PWM frequency and duty factor for choice 2. Show your working. (2 marks)

Advanced tasks (no specific help will be provided with these).

These tasks are relevant to embedded programming that uses peripherals and interrupts. Please complete them in either assembly or C as requested and give appropriate directory names.

Task 6 (either assembly or C) – Peripheral programming

Task 6.1 – Read the ADC

Using the built in ADC, read the output code for the built in fixed voltage reference, and display it on the LEDs. Do nothing in the superloop. Do not use any of the subroutines provided by the module convenor. Works – 1 mark.

Q6.1 What voltage does the fixed voltage reference provide (multiply the code read above by the ADC resolution found according to your applied ADC settings)? Show your working. If the result does not match what you expect, analyse why. (1 mark).

Task 6.2 - Control brightness of an LED using hardware PWM.

In this task use pin **`P1D`** to control the brightness of the LED connected to it. Switch on LD3...LD0 to have a brightness reference on board. Do nothing in the superloop.

Using the forth digit of your student ID number (4 for the student 123**4**567) and the following table, find the required PWM frequency:

4th digit	0	1	2	3	4
	5	6	7	8	9
Frequency, kHz	50	10	5	2	1

Using the 5th digit of your student ID number Y (5 for the student 1234**5**67) find the required PWM duty cycle as close as it is possible to the value calculated from the following formula:

Duty cycle = 10 + Y*2 (%)

The student 1234567 will thus use PWM of 20% duty cycle at 1 kHz. Works – 1 mark.

Q6.2 What is the advantage of using the peripheral comparing to the use of software bit bang (task 5, 1 mark)?

Q6.3 What duty cycle value was it actually possible to set and why? Show your working (1 mark).

Task 6.3 – Comparator

Using the third digit of your student ID number as a four bit binary number (if the value is 0, use 1 instead), set the variable low range reference voltage to the positive input of the comparator 1.

(The student 1234567 with the third digit 3 will set VR<3:0> to 0011.)

The comparator's negative input should be connected to the **`RA0`** (on board potentiometer). The code should display the comparator's output using LD6.

Rotate the potentiometer slowly. When the comparator trips the code should measure the `RA0` input voltage using the ADC; display the measured code using LEDs and stop. Works – 3 marks.

Q6.4 What voltage was set using the internal variable reference voltage? Show your working (1 mark).

Q6.5 What was the voltage when the comparator tripped? Show your working (1 mark)

Task 7 - Interrupt programming

Task 7.1 (assembly only)

Enable external interrupt on the falling edge; switch on `LD0` from the ISR when it happens.

Do nothing in the superloop. Works – 2 marks.

Task 7.2 (choose C or assembly)

Enable interrupt on PORTB change and count the number of changes on pin RB0 in the ISR. Superloop displays this value using LEDs. Works – 3 marks.

Task 7.3 (C only)

Set Timer 1 to overflow approximately twice a second. Enable interrupts for the ADC, for Timer1 and for the external interrupt, using the rising edge.

In your ISR count how many times each of these interrupts occurred, using 8 bit counters.

In your superloop:

Call `ReadADC()`

If the button is not currently pressed, you should call Select4 and display one of the above counters depending on the result. If choice 0 was selected, display the ADC interrupt counter; if choice 1 was selected, display the Timer1 interrupt counter; for the two remaining choices display the external interrupt counter. Continuously display the selected value until the button is pressed again. Works – 5 marks.

Task 8

This is the final coursework task and is the development of a more complex assembly program than you have previously attempted. This task is marked a bit differently to the other tasks in that as well as being assessed on the correct functionality of your code, you will also be assessed on the quality of your code and appropriate use of subroutines and peripherals. I want to see well designed and thought out code that is logical and easy to follow. You should also be aiming to achieve as low power usage as possible (this will not be directly measured) by ensuring that any peripherals or features you are using are only turned on when needed. You should already be producing good quality code based on the feedback you will have been given with the previous tasks.

This is not a trivial task, please do not underestimate it.

If you need further clarification of what is involved with the task you are free to ask a demonstrator, but direct feedback on your actual code will not be given for this task.

The task is to write a program in assembly that acts as a programmable display using the 8 LEDs. The display has three different display modes:

- Variable PWM
- Side to side strobing display
- Linear feedback shift register

When the board is first turned on, one of the above modes should be active and running on the LEDs. The decision about which mode is active is determined by the value of the potentiometer. You should split the range of the potentiometer into three ranges to specify which of the modes is currently active. You can use either the ADC or a comparator connected to the potentiometer to determine which mode to display, but whatever method you use should not significantly affect the display. When the potentiometer is rotated, the new mode should be made active with only a very short delay, and should continue where that mode last left off. You may need to reconfigure peripherals when moving from one mode to another. It may be worth considering making subroutines to help with this.

By pressing the button with a single press and release, your program should enter a configuration mode for the currently active display mode. When your program enters configuration mode, you should use LD7 and LD6 to indicate which display mode is currently being configured and clear the remaining LEDs. Your program should then be ready to receive new configuration values using the ADC and potentiometer, with the configuration options for the different modes as described below. The configuration option is hence selected using the ADC and potentiometer, and input using the button. When the configuration is complete your program should return to display mode after checking which mode should be active – note that because the user has been using the potentiometer this may be a different mode.

The display modes and their configuration options are described below.

Variable PWM

When this mode is active, one LED must be controlled with PWM. You may choose which LED to control as part of your design. The LED should be changing brightness continuously, i.e. changing from between two different brightness values. This mode has three configuration options which must be presented to the user in this order:

1. The speed of the variable PWM (one of four, choose appropriately), i.e. how quickly the display changes between the two brightness values.

2. The brightness (one of four, please choose appropriately) of the first brightness value

3. The brightness (one of four) of the second brightness value

Side to side strobe

When this mode is active, the display initially consists of a single LED “moving” from side to side from LED 0-7 and back to 0. This mode has two configuration options, which must be presented to the user in this order:

1. The speed of the motion (one of four, choose appropriately), i.e. the amount of time between one LED turning on and the next LED turning on

2. Choose between the LED moving back and forth, or just in one direction only, i.e. when the LED gets to LD7 it should next move to LD0 in this mode

Linear feedback shift register

A LFSR is a means of generating a pseudo random number. When this mode is active, the current value of the 8-bit LFSR you have created will be displayed on the LEDs. If you are not familiar with LFSRs, I suggest you do some searching on how to implement them. The Wikipedia page has good information – you should be producing an 8-bit maximal period Galois LFSR. Note that it is important for the initial value of the LFSR to be chosen with some care – setting it to 0 will not work.

This mode has one configuration option, which is the speed at which new values are displayed (one of four, choose appropriately).

Assessment

The assessment for this task is broken down as follows:

Completion: 16 marks (4 marks per mode, plus 4 marks for everything else)
Use of peripherals: 4 marks
Use of subroutines: 4 marks
Overall code quality and structure, including good quality comments where needed only: 6 marks

Please show your working code to a demonstrator to have the “completion” part of the assessment ticked off. The other elements of the assessment will be marked based on your submitted code, which should be included as a project along with your other tasks. Please note that the demonstrator sheets only have completion tick boxes for the three display modes, this is because in practice you must complete the “everything else” part before you are able to demonstrate any of the display modes. Partial completion is possible for this task, you can get the different display modes ticked off separately.