H63ECH Coursework

Academic year 2019/2020

NANDI GUO-14311495

Task 1 (assembly)

Starting from the template **Task1**, add the missing pieces to the code to implement the following actions. You will need to modify the look-up-table (LUT) provided accordingly.

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**

Combining **A** and **B** to 4 bit value, select the correct *calculation* result from the LUT 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)?

The code should be same.
The position for A should be shifted 2 bits left by using RLF instruction,
Then using ADDWF add the shifted A with B to get the LUT order.

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

The code should be same.
The position for A should be shifted 2 bits left by using RLF instruction,
Then using ADDWF add the shifted A with B to get the LUT order.

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, perform the *calculation* 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)?

The code should be same.
The position for A should be shifted 2 bits left by using RLF instruction, which means A*4, then use ADDWF add the shifted A and Original A, thus A*5 = A*4 + A
Then use XORWF to calculate the A^B.

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

The code is still same.
The position for A should be shifted 2 bits left by using RLF instruction, which means A*4, then use ADDWF add the shifted A and Original A, thus A*5 = A*4 + A
Then use XORWF to calculate the A^B.

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}	1.35s	1.38s	1.31s	1.33s
Code size, Instructions	2.41 KB	1.37 KB	1.62 KB	1.04 KB

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

Advantages of C

- **C is easier to use for writing programs and C allows users to write programs faster.**

- **It is easier to understand the C code, and it is easier to learn C language.**
- **C has smaller file.**

Advantages of assembler:

- **Assembler is a lower level programming language than C, so this makes it a good for programming directly to hardware.**
- **It provides a faster execution time.**

Task 4 – Digit display

Display last four digits of your student number

(for the student number 11234567 the student will need to display "7" for an input of 0, "6" for an input of 1, "5" for an input of 2, and "4" for an input of 3. If a 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).

The size is 2.25 KB.

There are repeat codes for controlling the LED, this part should be optimized by using loops.

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 1123**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 (counting **DOWN**) {
If the loop counter equals X, switch **`LD0`** on
(***hint***: use **`SUBLW`** or **`XORWF`** 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)

- **The frequency can not be controlled precisely.**
- **Human eyes also have limited observation ability.**

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

The Led is lighting when loop counter = 5,4,3,2,1.
The loop has 28 iterations.
Thus, the duty factor is 5/28 = 17.9 %
PWM frequency = 1/T = 1/0.179=5.6 Hz

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. Think carefully about the value you would expect to measure and justify the output of the ADC accordingly so that all the information can be displayed in one byte (rather than sequentially displaying two bytes). 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).

The LED shown a binary number value 1111110, which equal 126 in Decimal.

Then the voltage4rt4rt is 126. 5*(126/1023) = 0.615V

Thus, it meets the requirement (0.6V).

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 fifth digit of your student ID number (4 for the student 1123**4**567) and the following table, find the required PWM frequency:

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

Using the 6^{th} digit of your student ID number Y (5 for the student 11234**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 11234567 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)?

- **Clear algorithm and simple code implementation**
- **Faster execution speed**

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

The resolution determines the number of available duty cycles for a given period.

$$Resolution = \frac{log[4(PR2+1)]}{log(2)} \ bits$$

It is possible to set the duty cycle value to 50%, the voltage signal can be 1,0 only, it can be ON 50% of the time and should be OFF 50% of the time, it is easier to calculate and achieve.

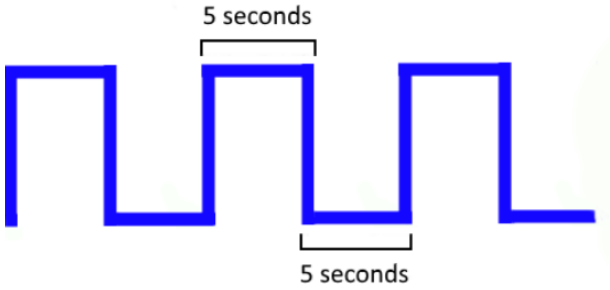

Figure 1: 50% duty cycle example

Task 6.3 – Comparator

Using the fourth 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 11234567 with the fourth digit 3 will set VR<3:0> to 0011.)

The comparator's negative input should be connected to the **RA0** (on board potentiometer). Add code so that when the comparator trips the code should measures the **RA0** input voltage using the ADC; display the measured code using LEDs and **stop**. Works – 3 marks.

Start by rotating the potentiometer fully clockwise, then starting the program. Slowly rotate the potentiometer until the ADC value is displayed on the screen.

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

VR<3: 0> : CVref Value Selection 0≤VR<3:0>≤15

When VRR=1 CVref = (VR<3:0>/24)*VDD

When VRR = 0 CVref = Vdd/4+(VR<3:0>/32)*VDD

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

The voltage should be 208mv.

w