Middle East Technical University
Department of Computer Engineering

CENG 336

Spring 2019

Take Home Exam 3

Regulations

Due: Sunday, May 31st 2020, 23:55. **Late submission is not allowed**

Submission: via ODTUClass

Any clarifications and revisions to the assignment will be posted to the ODTUClass discussion forum.

Introduction

The purpose of this assignment is to familiarize you with the **ADC module** of the PIC together with **TIMERs** and **INTERRUPTs**. In this assignment, you will essentially implement a guessing-game program, where the microcontroller is holding a special digit that one tries to guess. The user adjusts their guesses by turning the potentiometer on the board. The digit that the user would like to send is displayed in the 7-Segment display at this time. The user has to confirm their guesses by pressing on the RB4 button after they adjust them. After each confirmation, the microcontroller gives a hint to indicate that the number is greater or less.

This guessing phase ends if one of the following happens: the user fails to submit the correct guess in 5 seconds, or they manage to submit a correct guess in time. Regardless of what happens, the correct digit is blinked on the 7-Segment display for 2 seconds and the program restarts.

Specifications

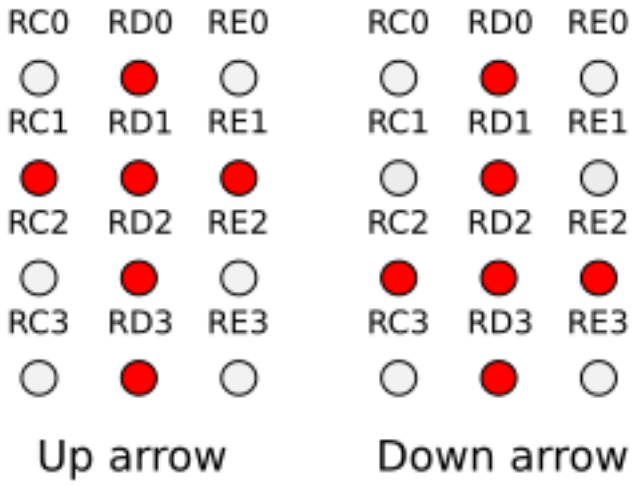
- The **special number** will be in range $[0, 9]$ and be assigned by the debug scripts you are provided.
- Once the simulation is started, the program should start the timers.
- The player is expected to find the number in 5 seconds. You should use **TIMER1** to keep time for 5 seconds.
- The **TIMER0** should start **ADC** conversion. You should use ADC interrupt feature to detect the end of conversion and read the converted value and sample ADC value at **50 ms** intervals using TIMER0 interrupt.
- While guessing the number, the player will be able to navigate through the values by adjusting it with ADC module.
- 10-bit adjusted ADC will be used. The ADC value should be 0 when you turn the ADC potentiometer clock-wise to its leftmost position (lowest voltage), and it should be 1023 when you turn the ADC potentiometer clock-wise to its rightmost position (highest voltage).

- After calculating the ADC value, you should map the values to $[0, 9]$ taking equal intervals. Lowest interval should map to the digit 0 and the highest interval should map to the digit 9. ADC values are mapped to the digits as shown in Table 1.

$0 <= X <= 102$	0
$102 < X <= 204$	1
$204 < X <= 306$	2
$306 < X <= 408$	3
$408 < X <= 510$	4
$510 < X <= 612$	5
$612 < X <= 714$	6
$714 < X <= 816$	7
$816 < X <= 918$	8
$918 < X <= 1023$	9

Table 1: ADC Value-Digit Mapping

- The digit that current ADC value is mapping to will be shown on the rightmost display (D0) of 7-Segment display. Each ADC read should update the 7-Segment display.
- The RB4 push button will be used for making a **guess**. The player can press RB4 button to assign the value on the 7-Segment display as the guess. The effect should happen on the press, not release.
- You will use lower 4 bits of PORTC, PORTD and PORTE to give hints to the player during the game.
- For each wrong guess, a hint will be given to the player by turning on the LEDs. If the guess is greater than the special number a down arrow should appear. If the guess is less than the special number an up arrow should appear. See below figure for details.



- There is no limit to the number of guessing attempts. The player can guess as much as they want.

- The game ends when 5 seconds are up or when the player guesses the special number correctly. At the end of the game, the special number is shown to the player on 7-Segment display and all the LEDs are turned off. For 2 seconds, 7-Segment display should blink the special number with half-second intervals. In other words, 7-Segment display should first show the special number for 500 ms then should dim out for 500 ms and repeat this again. **This interval should be ensured with TIMER1.**

- After displaying the number, the game restarts. This cycle continues until the system is powered off or the simulator is stopped.

Simulation Environment and Specifications

Due to the COVID-19 precautions, most of you are unable to access the MCDEV development kits assigned to you. For the evaluations to be fair, your code will be black-box tested with the simulator and the debugger along with white-box testing when needed. For your code to comply with the debug script tests, you are required to label certain points of your code and write to some variables according to the specifications given below. Some of these scripts are also provided to you along with the homework so that you can quickly evaluate your code.

You are encouraged to learn more about debugging commands (which are similar to gdb) from `https://microchipdeveloper.com/mplabx:mdb` so you can write your own to speed up your tests in the simulator. You may also want to learn about SCL (`https://microchipdeveloper.com/mplabx:scl`) which can be used to simulate various inputs on the pins, but you do not need too much of it.

Debouncing

In real life, most switches bounce (`https://en.wikipedia.org/wiki/Switch#Contact_bounce`) when pressed and released. Since the usual simulation does not incorporate that, some of the debug scripts excite the pins with erratic inputs that stabilize over time. **Your code should be robust against these such that multiple responses do not emerge after a "real life like" pulse is sent to RB4**. One way to do this is to accept the change in the input only if it persists for a while (For example, a pulse for 1 ms is probably a bounce noise. However, if that pulse is more than approximately 10 ms, it is a genuine one).

Important Variables

- `volatile char special`: Your program is supposed to be comparing the digit the ADC reading represents with this variable. The debug script tests may write a value to this variable at the beginning of your program. **Do not access this variable directly, use the `special_number()` function instead. Do not write to this variable.**

- `volatile int adc_value`: This variable should represent the most recent value read from the ADC. **Do not write to this variable any value other than the the most recent value read from the ADC.**

Label Functions

The MPLAB X debugger (mdb) unfortunately does not support labels for program addresses. The `breakpoints.c` file defines several empty functions. The only purpose of these functions are

to simulate labels that should inform mdb to stop running the code when PC hits the addresses of these functions.

- `init_complete:` should be called as soon as the program finishes initializing necessary SFRs, variables, etc. Your program is expected to have started TIMER0 and TIMER1 countdowns, PORTB on change interrupts and the ADC readings (per 50 ms) after this call.
- `adc_complete:` should be called per 50ms using TIMER0 as soon as a successful conversion of AN12 is made and `adc_value` is updated.
- `hs_passed:` should be called per 500ms using TIMER1.
- `rb4_handled:` should be called as soon as RB4 is pressed and the signal is properly debounced. It should not be called for any other PORTB change, RB4 release, or simulated bouncing noise on the signal (discussed above).
- `latjh_update_complete:` should be called as soon as the outputs of PORT[JH] are updated. The function may be called even if their values do not change. Your program is expected to call this function after each call to `adc_complete` in the guessing phase and after each call to `hs_passed`,`correct_guess` and `game_over` functions in the end-game phase.
- `latcde_update_complete:` should be called as soon as the outputs of PORT[CDE] are updated. The function may be called even if their values do not change. Your program is expected to call this function after each call to `rb4_handled` in the guessing phase and after the call to `correct_guess` and `game_over` functions.
- `correct_guess:` should be called after a correct guess (meaning the program execution will hit `rb4_handled` first, and then this function). The 7-Segment display should stop reflecting the ADC readings and start blinking the special digit in the rest of the program execution.
- `game_over:` should be called after the 5 second timeout without any correct guess. The 7-Segment display should stop reflecting the ADC readings and start blinking the special digit in the rest of the program execution.
- `restart:` should be called after the 7-Segment display has blinked 2 times (for 2 seconds), just before the program restarts.

Preparing your Project for Debug Tests

The debug tests require some modifications to your project so that the above function and variable symbols can be seen in the mdb session. Your MPLAB X IDE version should be >= 5.15 and XC8 Compiler version should be >= 2.00.

1. Type `mplab_ide` to open the MPLAB X IDE on terminal.
2. Select File -> New -> Standalone Project. Create new project with the following configurations.
    - Select device -> PIC18F8722
    - Select tool -> Simulator
    - Select Compiler -> XC8

- Project Name: the3
- "Project Location" should be a directory called `the3`. (The path should end with `"/the3"`).
- Check "Use project location as the project folder" option. The "Project Location" and the "Project Folder" paths should be the same.

3. Select File -> Project Properties. Under "XC8 Compiler" item, pick "Option Categories" as "Optimizations". **Uncheck the "Assembler files" option and Check the "Debug" option.** Click Apply and OK.

4. You may also want to adjust the settings under "Simulator" tab as you did in THE1. Keep in mind that the options under this tab will only apply to the simulator in the GUI. The debug tests have their own simulator settings set.

5. Right click on "Source Files", located in the Projects window on the left. Select "Add Existing Item", and then add the `breakpoints.c` and `main.c` files. Similarly, right click on the "Header Files", Select "Add Existing Item", and then pick `breakpoints.h` file.

6. Try compiling in the GUI to see if there were any problems until now.

7. Outside the GUI create a new directory under the project directory called `debug_tests` and extract all the supplementary files here except `main.c, breakpoints.c` and `breakpoints.h`.

8. Try running `./build.sh` from the terminal with `debug_tests` as your current directory, it recompiles with `make clean` and `make all` using the Makefile that MPLAB generated. The project should be successfully recompiled. Now try running `mdb initconfig.dbg` from the terminal with `debug_tests` as your current directory, which does some initial configurations for the debugger and puts it into interactive mode. Try typing `break init_complete`, enter, `run`, enter to see if the function labels are visible and the execution breaks at the correct place. If they are, you are all set (Type `quit` to exit `mdb`). Delete the inside of the main function and start working on your homework.

Coding Rules

- **You MUST properly comment your code, including a descriptive and informative comment at the beginning of your code explaining your design, choice of tasks and their functionality.**
- You will code your program using PIC C language. You should use **XC8 C compiler** for MPLAB X. You can find the related documents on ODTUClass.
- Your program should be written for PIC18F8722 working at **40 MHz**.
- You should obey the specifications above about TIMER0 and TIMER1 interrupt implementations.

Resources

- Code stubs and MDB scripts provided with homework
- PIC18F8722 Datasheet

- PIC Development Tool User and Programming Manual
- Recitation Documents
- ODTUClass Discussions

Hand In Instructions

- You should submit your code as a single file named as the3.zip through ODTUClass. This file should only include "main.c". Make a single submission per group.
- At the top of "main.c" file, you should write your group number followed by name, surname, and ID of participants as a comment text.

Cheating

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations.

Cheating Policy: Students/Groups may discuss the concepts among themselves or with the instructor or the assistants. However, when it comes to doing the actual work, it must be done by the student/group alone. As soon as you start to write your solution or type it, you should work alone. In other words, if you are copying text directly from someone else - whether copying files or typing from someone else's notes or typing while they dictate - then you are cheating (committing plagiarism, to be more exact). This is true regardless of whether the source is a classmate, a former student, a website, a program listing found in the trash, or whatever. Furthermore, plagiarism even on a small part of the program is cheating. Also, starting out with code that you did not write, and modifying it to look like your own is cheating. Aiding someone else's cheating also constitutes cheating. Leaving your program in plain sight or leaving a computer without logging out, thereby leaving your programs open to copying, may constitute cheating depending upon the circumstances. Consequently, you should always take care to prevent others from copying your programs, as it certainly leaves you open to accusations of cheating. We have automated tools to determine cheating. Both parties involved in cheating will be subject to disciplinary action. [Adapted from http://www.seas.upenn.edu/ cis330/main.html]