Middle East Technical University | Department of Computer Engineering

CENG 336

Intro. to Embedded Systems Development

Spring 2022-2023

LAB REVIEW EXAM

This examination has three independent questions that incrementally incorporate the use of LEDs, PORTB Interrupt-On-Change, Timers, and the ADC Module. You are required to implement and submit each task, separately.

1 Regulations

- You must code your program in C and compile it with MPLAB XC8 C compiler.
- Your program should be written for PIC18F4620 and working at 4 MHz clock frequency. Do not forget to configure it from the PICSimLab also.
- This lab exam consists of 3 tasks. Each task has its own rules. Please take attention to them.
- Your submissions will be graded using automated testing and code inspection. You **need to** call the functions provided to you in `breakpoints.c` in your source code for correct testing. See Testing section for details.
- **Using any library in any task is strictly forbidden.**
- You should choose **Breadboard** under the 'Board' tab on PICSimLab.
- You are provided a .pcf file to configure the spare parts. You can get access to the buttons, LEDs and the potentiometer by following tabs 'Modules' → 'Spare Parts' → 'Load Configuration' on PICSimLab.
- For the rest of this document a button press corresponds to the change from 0 to 1 on the corresponding port and a button release corresponds to the change from 1 to 0.
- **HAND OUT** : Along with this pdf, you will find `handout.zip`, PIC18F4620 Datasheet, XC8 C Compiler User Guide. In `handout.zip`, you are provided starter projects for each task and their testers. Also, you can check the solutions of each task with the provided .hex files.
- **SUBMISSION** : You should submit each task as a "separate" and "independently working file". At the top of each file ("task1.c", "task2.c" and "task3.c"), you should write your name, surname, and ID as a comment and submit through ODTUClass. Tasks that are not submitted will not be considered in your grade.

2 TASK-1 (20 Points)

- In this task, you are going to measure time by **polling** and show the result on LEDs.
- The time you are going to measure is the duration passing between two button presses.
- The buttons you are going to use are `RB4` and `RB5`.
- The user presses onto the `RB4` button first. After a while, s/he presses onto the `RB5` button. You should measure how much time it takes between `RB4` **button press** (not release) and `RB5` **button press** (not release).
- If the user successively clicks on `RB4` button until pressing onto `RB5`, only the first pressing onto `RB4` is taken into account, the remainings are ignored. Similarly, among the successive clicks on `RB5` button, only the first pressing should be reacted by your code.
- You should show the measured time in real-time at each multiple of 0.5 seconds. The measurement should be shown by turning on the `PORTC` and `PORTD` LEDs with a precision of 0.5. Examine the examples:
  - **EXAMPLE-1:** If the elapsed time between two button press is 3.728 seconds (3728 milliseconds), then the LEDs should be turned as follows:
    - when 0.5 second passed: No LED of `PORTC` and first five LEDs of `PORTD`
    - when 1 second passed: First LED of `PORTC` and no LED of `PORTD`
    - when 1.5 second passed: First LED of `PORTC` and first five LEDs of `PORTD`
    - when 2 seconds passed: First two LEDs of `PORTC` and no LED of `PORTD`
    - when 2.5 seconds passed: First two LEDs of `PORTC` and first five LEDs of `PORTD`
    - when 3 seconds passed: First three LEDs of `PORTC` and no LED of `PORTD`
    - when 3.5 seconds passed: First three LEDs of `PORTC` and first five LEDs of `PORTD`

    The final amount you should show on the LEDs is 3.5.
  - **EXAMPLE-2:** If the elapsed time between two button press is 6.439 seconds (6439 milliseconds), then the LEDs should be turned on as follows:
    - when 0.5 second passed: No LED of `PORTC` and first five LEDs of `PORTD`
    - when 1 second passed: First LED of `PORTC` and no LED of `PORTD`
    - when 1.5 second passed: First LED of `PORTC` and first five LEDs of `PORTD`
    - when 2 seconds passed: First two LEDs of `PORTC` and no LED of `PORTD`
    - when 2.5 seconds passed: First two LEDs of `PORTC` and first five LEDs of `PORTD`
    - when 3 seconds passed: First three LEDs of `PORTC` and no LED of `PORTD`
    - when 3.5 seconds passed: First three LEDs of `PORTC` and first five LEDs of `PORTD`
    - when 4 seconds passed: First four LEDs of `PORTC` and no LED of `PORTD`
    - when 4.5 seconds passed: First four LEDs of `PORTC` and first five LEDs of `PORTD`
    - when 5 seconds passed: First five LEDs of `PORTC` and no LED of `PORTD`
    - when 5.5 seconds passed: First five LEDs of `PORTC` and first five LEDs of `PORTD`
    - when 6 seconds passed: First six LEDs of `PORTC` and no LED of `PORTD`

    The final amount you should show on the LEDs is 6.
  - **EXAMPLE-3:** If the elapsed time between two button press is 0.821 seconds (821 milliseconds), then the LEDs should be turned on as follows:
    - when 0.5 second passed: No LED of `PORTC` and first five LEDs of `PORTD`

The final amount you should show on the LEDs is 0.5.

Note that you need to **round down** the values to their correspondences to 0.5-precision. Also note that there are only 8 pins in both `PORTC` and `PORTD`. Therefore, the maximum timing that your code will be tested with is 7.999 seconds whereas the minimum timing is 0.5 second.

- To sum up, you should update the LEDs at each 0.5 seconds and keep the current LED configuration turned on until the next update comes.
- When `RB5` button is pressed, you should **clear all the LEDs**.
- After the first pressing onto `RB5` button (note that the successive clicks are ignored), your code should be ready for another time measurement. That is, it should not be terminated. Instead, it should wait for a next `RB4` button press.
- The button actions should not block the execution of the program.
- You should show the correct LED configuration within an error margin of ± 50 milliseconds.
- You can update `PORTC` and `PORTD` as many times as you want. In other words, updating them is not just limited to the moment of reflecting changes in the LED configuration.
- Don't worry, there is no bouncing effect on the given buttons.
- **Using interrupts (timer, button, etc.) and the built-in delay functions are strictly forbidden.**

3 TASK-2 (35 Points)

- In this task, you are going to implement the same task given in "TASK-1" by using **interrupts** this time.
- You need to measure the elapsed time by using TIMER-0 interrupt.
- For the button presses you are going to use `PORTB ON-CHANGE` interrupt. You should start the time measurement with the **RB4** on-change interrupt occurring at **rising edge** and end with the **RB5** on-change interrupt occurring at **rising edge**.
- Among the successive clicks on the same button, only the first pressing is taken into account which is also the case in "TASK-1".
- Precision and maximum timing limit are the same as the given in "TASK-1", which are 0.5 and 7.999 seconds. You can examine the examples given in "TASK-1".
- Similar to "TASK-1", you should clear all the LEDs when `RB5` button is pressed and your code should wait for a next possible `RB4` button press.
- You should show the correct LED configuration within an error margin of ± 50 milliseconds.
- You can update `PORTC` and `PORTD` as many times as you want. In other words, updating them is not just limited to the moment of reflecting changes in the LED configuration.
- Don't worry, there is no bouncing effect on the given buttons.
- **Using polling and the built-in delay functions are strictly forbidden.**

4 TASK-3 (45 Points)

- In this task, you are going to adjust the precision used in "TASK-2" by using **ADC (analog to digital converter) interrupt**.
- You need to measure the elapsed time by using TIMER-0 interrupt.
- By using the AN0 dial on PICSimLab, you are going to change the precision amount shown on the LEDs. The scale of 1024 should be separated into the following intervals:
  - $[0, 341]$: The precision is 1.0. That is, the values shown on the LEDs can be integer multiples of 1.0. If the elapsed time is measured as 7.9, you should show it as 7 on the LEDs.
  - $[342, 681]$: The precision is 0.5. That is, the values shown on the LEDs can be integer multiples of 0.5. If the elapsed time is measured as 7.9, you should show it as 7.5 on the LEDs.
  - $[682, 1023]$: The precision is 0.2. That is, the values shown on the LEDs can be integer multiples of 0.2. if the elapsed time is measured as 7.9, you should show it as 7.8 on the LEDs.

  Note that you need to **round down** the values to their correspondences of the required precision. Also note that there are only 8 pins in both `PORTC` and `PORTD`. Therefore, the maximum timing that your code will be tested with is 7.999 seconds whereas the minimum timing is 1.0, 0.5 and 0.2 second for each interval, respectively.
- **The precision could be changed at any moment. However, it affects the next time measurement which starts with next pressing onto `RB4` after `RB5`.** You can check how the program behaves with the task3.hex file.
- **You should do ADC sampling at every 100 milliseconds within an error margin of ± 10 milliseconds. You should use TIMER-1 for this purpose.**
- You should show the correct LED configuration within an error margin of ± 50 milliseconds.
- You can update `PORTC` and `PORTD` as many times as you want. In other words, updating them is not just limited to the moment of reflecting changes in the LED configuration.
- Don't worry, there is no bouncing effect on the given buttons.
- **Using polling and the built-in delay functions are strictly forbidden.**

5 Testing

The "`tests`" folders are ready under each task's project directory. Tester usage instructions:

1. Do not use low-priority interrupts.
2. All `PORTB` pull-ups must be disabled. (See `INTCON2` in the datasheet.) Otherwise, it works fine in PICSimLab but it flips all PORTB pins in MPLAB simulator.
3. When the simulator first runs, all variables will be set to 0. Your program MUST NOT depend on this behavior. The tester works by performing a hardware reset after each test case. After a hardware reset, the variables will NOT be set to 0. If you assume that all variables are set to 0 at the start, it's likely that your program will exhibit peculiar behavior after the first test case.
4. Make sure that there are no non-ASCII characters (e.g. special Turkish characters) in your source code. Otherwise, you may get `UnicodeDecodeError` while running the tester. Even if you don't encounter any errors, avoid non-ASCII characters because they may cause issues during grading.

5. You need to call the provided empty functions in your source code:

    - `init_complete()` should be called as soon as the program finishes initializing necessary SFRs, variables, etc.
    - `measurement_start()` should be called as soon as RB4 is pressed.
    - `measurement_end()` should be called as soon as RB5 is pressed.
    - `adc_done()` should be called per 100ms using TIMER1 as soon as a successful conversion of AN0 is made. You will need this function only for the last task.

6. In the project directory, run: `make`

    - Note that your code won't be compiled this way when you debug the program in MPLAB IDE. You need to run this separately.

7. `cd` into `tests` and run: `python3 test.py`

Important Note: The grades you will receive with the provided tests are NOT your final grades.