


Middle East Technical University Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development

THE1

Due: 24th of March 2024, 23:59

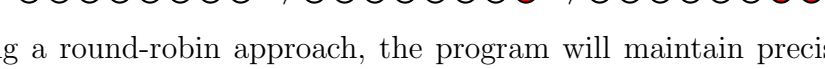
Submission: via **ODTUClass**

1 Introduction

The purpose of this individual assignment is to familiarize you with the basic I/O operations and round-robin approach on PIC18F8722 using MPLAB X IDE simulation environment. To this end, you will implement a device that is capable of rendering two kinds of progress bars using LEDs according to the user input.

The program will use an 8-bit output port connected to LEDs to display and update a progress bar. An example is shown below:

The second progress bar will operate in reverse order:



Using a round-robin approach, the program will maintain precise timing for both progress bars while handling the user input. The user will be able to enable/disable each of the two progress bars independently.

1.1 Notation

PORTX denotes an input/output port of PIC, where X is the letter of the port. LATX is the latch corresponding to PORTX. This assignment will use PORTB, PORTC, PORTD and PORTE. For a given PORTX, RXi for $0 \leq i \leq 7$ denotes the ith least significant bit in PORTX. For example, RB0 is the first (smallest) bit in PORTB. A single bit in a port is termed a "pin".

2 Specifications

2.1 Configuration

S-1 The program shall be written for **PIC18F8722** running at **1 MHz instruction frequency**.[1]

[1]Note that the instruction frequency of the boards given to you is 10 MHz. Consequently, the program will run 10 times faster on that hardware. In this assignment, we reduced the instruction frequency so that more time passes in the simulation for the same number of instructions. This makes tests run faster.

S-2 PORTB, PORTC and PORTD shall be used for output.

- PORTB and PORTC shall display separate progress bars.
- RD0 shall be a blinking LED.

S-3 PORTE shall be used for input.

- RE0: Toggle the progress bar on PORTC.
- RE1: Toggle the progress bar on PORTB.

2.2 Initialization

S-4 The program shall initialize all variables it uses, i.e., it shall not depend on variables being zero at the start.

S-5 The program shall light up all the LEDs in PORTB, PORTC and PORTD for 1000 ± 50 ms at the startup. This is called the **initialization period**.

S-5.1 The program does not need to respond to any inputs during this period, i.e., it can busy-wait. In this context, **busy-waiting** refers to a loop whose only purpose is to cause a time delay. During this loop, the program does not check for inputs and hence it's unresponsive.

S-6 After that, the program shall turn off all LEDs and begin operation.

2.3 Blinking LED

S-7 Immediately after the initialization period, RD0 shall turn off for 500 ± 50 ms and then light up for 500 ± 50 ms, repeating this cycle indefinitely.

S-8 The progress bars are disabled by default. All LEDs in PORTB and PORTC shall remain off until the user enables them.

An example run of the device is demonstrated in Table 1 when no input is given.

2.4 User Interaction

S-9 A "click" to a button is defined as a change from 1 to 0 in the corresponding pin.

S-9.1 Each pin is 0 by default, becomes 1 when pressed, and becomes 0 when released.

S-9.2 The program shall not busy-wait for the button release (1 to 0) after the button press (0 to 1). While the button is held (corresponding pin is 1), the program shall operate normally as if nothing happened. Only when a held button is released (1 to 0), a click is registered.

S-10 The program shall **NOT** implement button debouncing. Any change from 1 to 0 is a click, regardless of the duration or previous changes.

2.5 Progress Bars

S-11 PORTB and PORTC shall display two separate progress bars.

S-12 Progress bars shall update only when RD0 changes state (every 500 ± 50 ms).

S-13 The progress bar in PORTB shall advance from RB0 to RB7.

S-14 The progress bar in PORTC shall advance from RC7 to RC0 (reverse order).

S-15 If all bits in a progress bar are filled during update, all bits in the corresponding port shall be cleared for the next cycle (500 ± 50 ms period). Afterwards, the progress bar shall start again.

S-16 When a progress bar is disabled, its corresponding port shall be cleared in the next update (when RD0 changes state).

2.6 Implementation Details

S-17 The program shall use busy-waiting for the 1-second initialization period.

S-18 The program shall use a round-robin approach for the main operation and input handling.

S-19 Timers and interrupts are not allowed.

S-20 The program shall only write to the output ports when they need to change, i.e., once every 500 ± 50 ms. The program shall **NOT** perform unnecessary writes to the output ports in each iteration of the main loop. This is required for testing.

3 Example Runs

In this section, example runs of the program are given in tables.

- ○ represents that the LED is off, i.e., the corresponding pin is 0.
- ● represents that the LED is lit, i.e., the corresponding pin is 1.
- 8 LEDs denoted by ○○○○○○○○ represent an 8-bit output port. The rightmost LED is the least significant bit. For example, ○○○○○○●● represents the decimal value of 3 on the corresponding output port.

The duration represents how long the given LED configuration is maintained before advancing to the next row. Comments and inputs are provided as special rows in each table.

- Table 1 shows the expected behavior when no input is given.
- Table 2 demonstrates the operation of PORTB progress bar.
- Table 3 demonstrates the operation of PORTC progress bar.
- Table 4 demonstrates both progress bars in parallel.

Table 1: Example run with no input.

PORTB	PORTC	PORTD	Duration (ms)
●●●●●●●●	●●●●●●●●	●●●●●●●●	1000
Initialization period ends, main operation begins.			
○○○○○○○○	○○○○○○○○	○○○○○○○○	500
○○○○○○○○	○○○○○○○○	○○○○○○○●	500
Last 2 steps repeat indefinitely.			

Table 2: Example run demonstrating the progress bar on PORTB.

PORTB	PORTC	PORTD	Duration (ms)
●●●●●●●●	●●●●●●●●	●●●●●●●●	1000
○○○○○○○○	○○○○○○○○	○○○○○○○○	500
○○○○○○○○	○○○○○○○○	○○○○○○○●	500
○○○○○○○○	○○○○○○○○	○○○○○○○○	200
RE1 is clicked, the progress bar on PORTB is enabled.			
○○○○○○○○	○○○○○○○○	○○○○○○○○	300
○○○○○○○●	○○○○○○○○	○○○○○○○●	500
○○○○○○●●	○○○○○○○○	○○○○○○○○	500
○○○○○●●●	○○○○○○○○	○○○○○○○●	500
○○○○●●●●	○○○○○○○○	○○○○○○○○	500
○○○●●●●●	○○○○○○○○	○○○○○○○●	500
○○●●●●●●	○○○○○○○○	○○○○○○○○	500
○●●●●●●●	○○○○○○○○	○○○○○○○●	500
●●●●●●●●	○○○○○○○○	○○○○○○○○	500
○○○○○○○○	○○○○○○○○	○○○○○○○●	500
○○○○○○○●	○○○○○○○○	○○○○○○○○	500
○○○○○○●●	○○○○○○○○	○○○○○○○●	500
○○○○○●●●	○○○○○○○○	○○○○○○○○	350
RE1 is clicked, the progress bar on PORTB is disabled.			
○○○○○●●●	○○○○○○○○	○○○○○○○○	150
○○○○○○○○	○○○○○○○○	○○○○○○○●	500
○○○○○○○○	○○○○○○○○	○○○○○○○○	500
○○○○○○○○	○○○○○○○○	○○○○○○○●	300
RE1 is clicked, the progress bar on PORTB is enabled again.			
○○○○○○○○	○○○○○○○○	○○○○○○○●	200
○○○○○○○●	○○○○○○○○	○○○○○○○○	500
○○○○○○●●	○○○○○○○○	○○○○○○○●	500
○○○○○●●●	○○○○○○○○	○○○○○○○○	500
○○○○●●●●	○○○○○○○○	○○○○○○○●	500

Table 3: Example run demonstrating the progress bar on PORTC.

PORTB	PORTC	PORTD	Duration (ms)
●●●●●●●●	●●●●●●●●	●●●●●●●●	1000
○○○○○○○○	○○○○○○○○	○○○○○○○○	200
RE0 is clicked, the progress bar on PORTC is enabled.			
○○○○○○○○	○○○○○○○○	○○○○○○○○	300
○○○○○○○○	●○○○○○○○	○○○○○○○●	500
○○○○○○○○	●●○○○○○○	○○○○○○○○	500
○○○○○○○○	●●●○○○○○	○○○○○○○●	500
○○○○○○○○	●●●●○○○○	○○○○○○○○	500
○○○○○○○○	●●●●●○○○	○○○○○○○●	500
○○○○○○○○	●●●●●●○○	○○○○○○○○	500
○○○○○○○○	●●●●●●●○	○○○○○○○●	500
○○○○○○○○	●●●●●●●●	○○○○○○○○	500
○○○○○○○○	○○○○○○○○	○○○○○○○●	500
○○○○○○○○	●○○○○○○○	○○○○○○○○	500
Stopping and restarting behavior is similar to PORTB.			

Table 4: Example run demonstrating both progress bars.

PORTB	PORTC	PORTD	Duration (ms)
●●●●●●●●	●●●●●●●●	●●●●●●●●	1000
○○○○○○○○	○○○○○○○○	○○○○○○○○	200
RE1 is clicked, the progress bar on PORTB is enabled.			
○○○○○○○○	○○○○○○○○	○○○○○○○○	300
○○○○○○○●	○○○○○○○○	○○○○○○○●	500
○○○○○●●●	○○○○○○○○	○○○○○○○○	500
○○○○●●●●	○○○○○○○○	○○○○○○○●	500
○○○●●●●●	○○○○○○○○	○○○○○○○○	500
○○○●●●●●	○○○○○○○○	○○○○○○○●	250
RE0 is clicked, the progress bar on PORTC is enabled.			
○○○●●●●●	○○○○○○○○	○○○○○○○●	250
○○●●●●●●	●○○○○○○○	○○○○○○○○	500
○●●●●●●●	●●○○○○○○	○○○○○○○●	500
●●●●●●●●	●●●○○○○○	○○○○○○○○	500
○○○○○○○○	●●●●○○○○	○○○○○○○●	500
○○○○○○○●	●●●●●○○○	○○○○○○○○	500
○○○○○●●●	●●●●●●○○	○○○○○○○●	500
○○○○●●●●	●●●●●●●○	○○○○○○○○	500
○○○●●●●●	●●●●●●●●	○○○○○○○●	500
○○●●●●●●	○○○○○○○○	○○○○○○○○	500
○●●●●●●●	●○○○○○○○	○○○○○○○●	100
RE1 is clicked, the progress bar on PORTB is disabled.			
○○●●●●●●	●○○○○○○○	○○○○○○○●	400
○○○○○○○○	●●○○○○○○	○○○○○○○○	500
○○○○○○○○	●●●○○○○○	○○○○○○○●	500

4 Testing

A template MPLAB project is provided to you for a quick start. Open this starting project[2] in MPLAB X IDE. A testing script written in Python is also included in it. Instructions:

1. The following programs must be available in your `PATH`: `python3`, `make`, and `mdb` (Microchip debugger command line tool, it should be present if you have installed MPLAB and XC8 correctly.)

2. 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.

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

All tests are black-box tests; they don't assume anything about the internal structure of your program. But your program needs to abide by the following rules for correct testing:

1. **DO NOT** perform unnecessary writes to the output ports in each iteration of the main loop. Tests use the `watch` command in `mdb` to track changes in ports. If you keep rewriting data to the output ports in every iteration, the debugger will be overwhelmed and the test will fail.

2. When the simulator first runs, all variables will be set to 0. You 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.

3. 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.

4.1 Grading

Upon completion, the Python testing script will print the grading rubric and your grade for each criterion. The total grade reported by the Python script will be your grade, excluding the late submission penalty (if applicable) and other external penalties that may incur due to cheating or using forbidden features (e.g. timers and interrupts). Since the testing script determines your grade, please note that not abiding by the rules given in Testing section (Section 4) may result in **0 grade** due to complete testing failure, or a significant grade reduction.

[2] If you choose to create your own project (not recommended), make sure that the project folder name ends with ".X" and your code is in a single assembly file, directly under the project directory. After that, put the "tests" folder (from the starting project) inside your own project directory.

5 Regulations

1. This is not a group assignment, you must solve it **individually**.
2. Any clarifications and revisions to the assignment will be posted to ODTUClass. You are expected to follow the announcements so that you are up to date.
3. Ask your questions in the discussion forum dedicated for this assignment in ODTUClass. Unless you have *specific* question about *your code*, use the forum. If you think that your question is too specific to ask on the forum, you can ask your questions via email to İlker: `ilker@ceng.metu.edu.tr`
4. Please consult your lecture notes and datasheets first before asking any questions.
5. **Submission:** Upload a single `*.s` file to the ODTUClass assignment. The file name does not matter as long as the extension is correct.
6. **Late Policy:** You can extend the deadline by 3 days at maximum. For each day you extend the deadline, you will receive -10 grade penalty.
7. **Grading:** Your Submission will be evaluated using the testing script provided to you.
8. **Using the Board:** Optionally, you can deploy your solution to the board given to your group. Your solution is expected to work correctly on the board, albeit faster because of the difference in the instruction frequency (see the footnote of **S-1**). The board configuration is the same as in THE0. However, keep in mind that this is an individual assignment. You are not allowed to reveal your solution to other group members in any way, shape, or form, even partially.

6 Hints

1. There are plenty of resources available to you in ODTUClass. To refer to instructions and specifications of PIC18F8722, you should use the PIC18F8722 Datasheet. Since you will be using MPLAB X IDE simulation environment for this assignment, Recitation 1 documents can also be a great set up guide. If you set up a local environment, you should use MPLAB X IDE version 5.45 and XC8 Compiler version 2.30, both are available in the downloads archive. If you cannot set up a local environment, you can use the department labs.
2. **Stopwatch** tool of the simulator can be used to measure time spent by a code segment, by adding **breakpoints** to the start and end of that segment. You can find this window from **Window → Debugging** menu. It is important to configure your instruction frequency to 1 MHz from **Project properties → Simulator → Instruction Frequency** before starting the simulator for the time values to be correct. (1 MHz is the default but double-check to make sure.) **Project Properties** window can be reached by right clicking the project name in **Projects** panel. You can refer to Recitation 1 documents for detailed explanations and screenshots.
3. You can also see the states of your ports, variables and pins by using the Logical Analyzer, SFR and Variables windows. They can be found under the **Window** menu in **Simulator, Debugging, Target Memory Views** sections respectively.

6.1 Troubleshooting

1. **Cannot fire stimulus:** When you fire an event in the stimulus window, you should see a notification like "Asynchronous stimulus RB0 fired" at the bottom of the window. If it doesn't appear and the stimulus doesn't happen (note that the stimulus becomes effective only after you step/continue the debugger), this issue might be caused by **non-ASCII characters in your source code** or the encoding. Your code must NOT contain any non-ASCII characters. If the issue persists after removing all non-ASCII characters, try replacing your `.s` file with either `main.s` provided in the starting project or `recitation1.s` in order to ensure that the file has the right encoding.