Middle East Technical University

Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development

THE1

Due: 09 April 2023, 23:55

Submission: via **ODTUClass**

1 Objectives

The purpose of this assignment is to familiarize you with **basic I/O operations on MPLAB X IDE simulation environment**.

This is not a group assignment, you must solve it **individually**. Any clarifications and revisions to the assignment will be posted to ODTUClass. You should ask your questions in the discussion forum dedicated for this assignment in ODTUClass.

2 Scenario

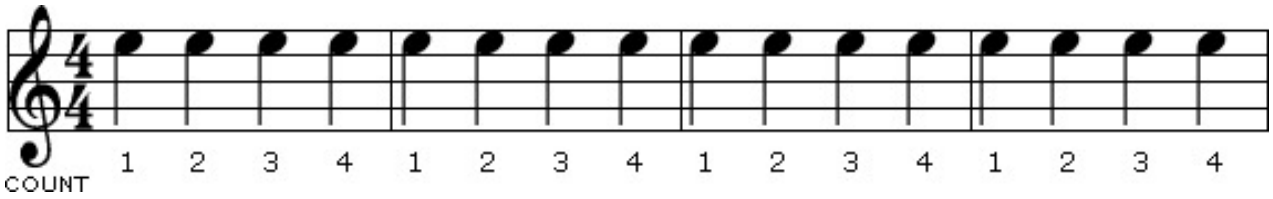

Figure 1: 4 bars of quarter notes with a bar length of 4.

A metronome is a tool that produces a consistent beat for musicians to practice rhythm. It emits a sound on each quarter note, and accentuates the first note of each bar. The bar length is typically 4 quarter notes. In Figure 1, 4 bars are shown, each consisting of 4 quarter notes.

In this assignment, you will make a visual metronome device. Instead of emitting sound, the device will demonstrate the beat using LED blinks. It will allow the user to pause & resume the metronome, change speed (1x speed and 2x speed options), and change the bar length. These features will be explained in detail in the specifications section.

3 Specifications

3.1 Configuration

S-1 The program shall be written for **PIC18F4620** running at **1 MHz instruction frequency**.

S-2 PORTA shall be used for output. RAi for $0 \leq i \leq 7$ denotes the ith least significant bit in PORTA. A single bit in a port is termed a "pin".

- RA0 and RA1 shall be used by the metronome.
- RA2 shall light up when the metronome is paused.
- The rest (RA3 to RA7) shall be unused.

S-3 PORTB shall be used for input. RBi for $0 \leq i \leq 7$ denotes the ith least significant bit in PORTB.

- RB0: Pause/Resume metronome.
- RB1: Switch between 2x and 1x speed.
- RB2: Reset bar length to 4.
- RB3: Increase bar length by 1.
- RB4: Decrease bar length by 1.
- The rest (RB5 to RB7) shall be unused.

3.2 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 the LEDs RA0, RA1, and RA2 for 1000 ± 50 ms at the startup. This is called the **initialization period**.

S-5.1 It 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 start the metronome.

3.3 Metronome Functionality

S-7 RA0 shall light up for 500 ± 50 ms and then turn off for 500 ± 50 ms (this is called a "blink" or a "note"), repeating this cycle as long as the metronome runs. [1]

S-8 RA1 shall light up with RA0 for the same duration at the start of each bar.

S-8.1 A bar is 4 notes long by default. This means that RA1 will light up with RA0 every 4th note.

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

[1] An error margin of 50 ms is too large for any musical application. However, such precise timing is not a learning outcome of this assignment.

3.4 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 After the initialization period (3.2), the program shall respond to all button clicks in 50 ms.

S-11 The program shall **NOT** implement button debouncing. Any change from 0 to 1 is a click, regardless of the duration or previous changes.

3.5 Pause/Resume

S-12 If the metronome is running, clicking RB0 shall pause the metronome.

S-13 As long as the metronome is paused, RA0 and RA1 shall be turned off and RA2 shall be lit.

S-14 If the metronome is already paused, clicking RB0 shall resume the metronome operation immediately. RA2 shall turn off, RA0 and RA1 shall continue where they left off.

3.6 Adjusting Speed

S-15 Clicking RB1 shall switch between 2x speed and 1x speed modes.

S-16 1x speed mode is the default.

S-17 In 1x speed mode, RA0 shall toggle (light up if turned off, turn off if lit) every 500 ± 50 ms.

S-18 In 2x speed mode, RA0 shall toggle every 250 ± 50 ms.

S-19 Regardless of speed, RA1 shall follow RA0 as described in subsection 3.3.

S-20 When RB1 is clicked, the speed change shall become effective after the next PORTA change. The duration of the current PORTA state is not important. See Table 3 for an example.

3.7 Changing the Bar Length

S-21 Clicking RB2 shall reset the bar length to its default value, which is 4.

S-22 Clicking RB3 shall decrease the bar length by 1.

S-23 Clicking RB4 shall increase the bar length by 1.

S-24 Attempting to set the bar length to a value lower than 1 or greater than 255 is undefined behavior.

S-25 Any changes to bar length shall take effect immediately, i.e., affect the currently playing bar.

Table 2 demonstrates how the device reacts to changes to the bar length.

3.8 Implementation Details

S-26 The program shall use busy-waiting for the 1-second initialization period.

S-27 The program shall use a round-robin approach for metronome operation and input handling.

S-28 Timers and interrupts are not allowed.

S-29 The program shall only write to PORTA or LATA when it needs to change. The program shall **NOT** perform unnecessary writes to PORTA or LATA. This is required for testing.

4 Example Runs

In this section, example runs of the program are given in tables. Only used output LEDs (RA0, RA1, RA2) are shown, the rest (RA3 to RA7) are always off (0 in the corresponding pins).

- ○ 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.

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

- Table 1 shows the expected behavior when no input is given.
- Table 2 shows how the device is expected to handle bar length changes.
- Table 3 shows the expected behavior when a button is held and when the speed is changed.

Table 1: Example run with no input.

RA0	RA1	RA2	Duration (ms)	Description
●	●	●	1000	Step 0: Initialization.
●	●	○	500	Step 1: Start of the bar.
○	○	○	500	
●	○	○	500	Step 3: Second beat of the first bar.
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	○	○	500	
○	○	○	500	Step 8: End of the bar.
Steps 1-8 repeat indefinitely.				

Table 2: Changing the bar length.

RA0	RA1	RA2	Duration (ms)	Description
●	●	●	1000	
●	●	○	500	
○	○	○	500	
●	○	○	500	RB4 is clicked twice, bar length is set to 6.
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	○	○	500	This bar continues since the length is changed.
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	●	○	500	New bar starts.
○	○	○	500	RB2 is clicked, bar length is reset to 4.
●	○	○	500	
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	●	○	500	New bar starts.
○	○	○	500	
●	○	○	500	
○	○	○	500	RB3 is clicked, bar length is set to 3.
●	○	○	500	
○	○	○	500	
●	●	○	500	New bar starts.
○	○	○	500	
●	○	○	500	
○	○	○	500	
●	○	○	500	
○	○	○	500	
...				

Table 3: Changing the speed and correct button handling.

RA0	RA1	RA2	Duration (ms)	Description
●	●	●	1000	Step 0: Initialization.
●	●	○	500	
○	○	○	500	
●	○	○	500	RB1 becomes 1.
○	○	○	500	Nothing changes because 0 to 1 is not a “click”.
●	○	○	500	
○	○	○	500	
●	○	○	-[a]	RB1 becomes 0. This is now a complete “click”.
○	○	○	250	New speed takes effect now.
●	●	○	250	
○	○	○	250	
●	○	○	250	
○	○	○	250	
●	○	○	250	
○	○	○	250	
●	○	○	250	
○	○	○	250	
...				

[a] This duration is not important as long as it’s not exceedingly long (e.g. longer than 1 second). See specification item **S-20**.

5 Testing

A template MPLAB project is provided to you for a quick start. Open this starting project in MPLAB X IDE. [2] 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 PORTA or LATA. Tests use the `watch` command in `mdb` to track changes in PORTA. If you keep rewriting data to PORTA 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.

5.1 Grading Rubric (Tentative)

The grade reported by the Python script will be your grade. The grading criteria is given in Table 4. $\rightarrow$ denotes that the grading item is a subitem and depends on its parent, the closest previous grading item. If the program fails the parent grading item, all of its subitems are automatically failed. For example, you cannot get points for bar length reset button (RB2) if your increase/decrease bar length functions (RB4/RB3) don't work.

[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.

Table 4: Tentative Grading Rubric

Criterion	Points
Initialization period (3.2)	5.0
Metronome operation without input	15.0
Pause on RB0 click	5.0
→ Resume on RB0 click	5.0
→ Continue while RB0 is held	10.0
→ Pause when RB0 is released	5.0
Increase bar length on RB4 click	5.0
→ Reset bar length on RB2 click	2.5
→ Continue normally while RB4 is held	5.0
Decrease bar length on RB3 click	5.0
→ Reset bar length on RB2 click	2.5
→ Continue normally while RB3 is held	5.0
1x speed while RB1 is held	5.0
2x speed after RB1 is released	10.0
2x speed while RB1 is held	10.0
1x speed after RB1 is released	5.0

6 Regulations

1. Consult your lecture notes and datasheets first before asking any questions.
2. Ask your questions on ODTUClass discussion forum. Unless you have *really 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`.
3. **Submission:** You should submit your code as **a single file** named as `1234567.s` through ODTUClass, where `1234567` represents your seven-digit student number.
4. **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.
5. **Grading:** Your Submission will be evaluated using the testing script provided to you.

7 Hints

1. There are plenty of resources available to you in ODTUClass. To refer to instructions and specifications of PIC18F4620, you should use the PIC18F4620 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.