Middle East Technical University | Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development

THE2

Due: 03 May 2023, 23:59

Submission: via **ODTUClass**

1 Objectives

The purpose of this assignment is to familiarize you with **7-segment displays, timers and interrupts**. This is a group assignment; you will work in pairs.

2 Scenario

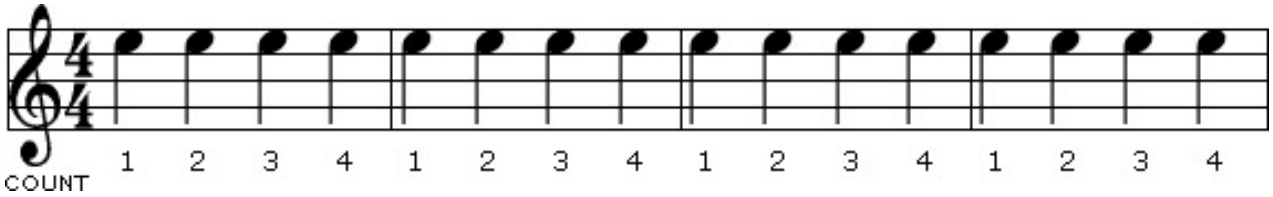

Figure 1: 4 bars of quarter notes with a bar length of 4.

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 create a metronome with 7-segment display, as seen in Figure 2. The features will be explained in detail in the specifications section.

- 9 different speed levels.
- Adjustable bar length between 2 and 8.
- Pause button.
- 7-segment display.
- Indicator LEDs denoting the start of a beat/bar.

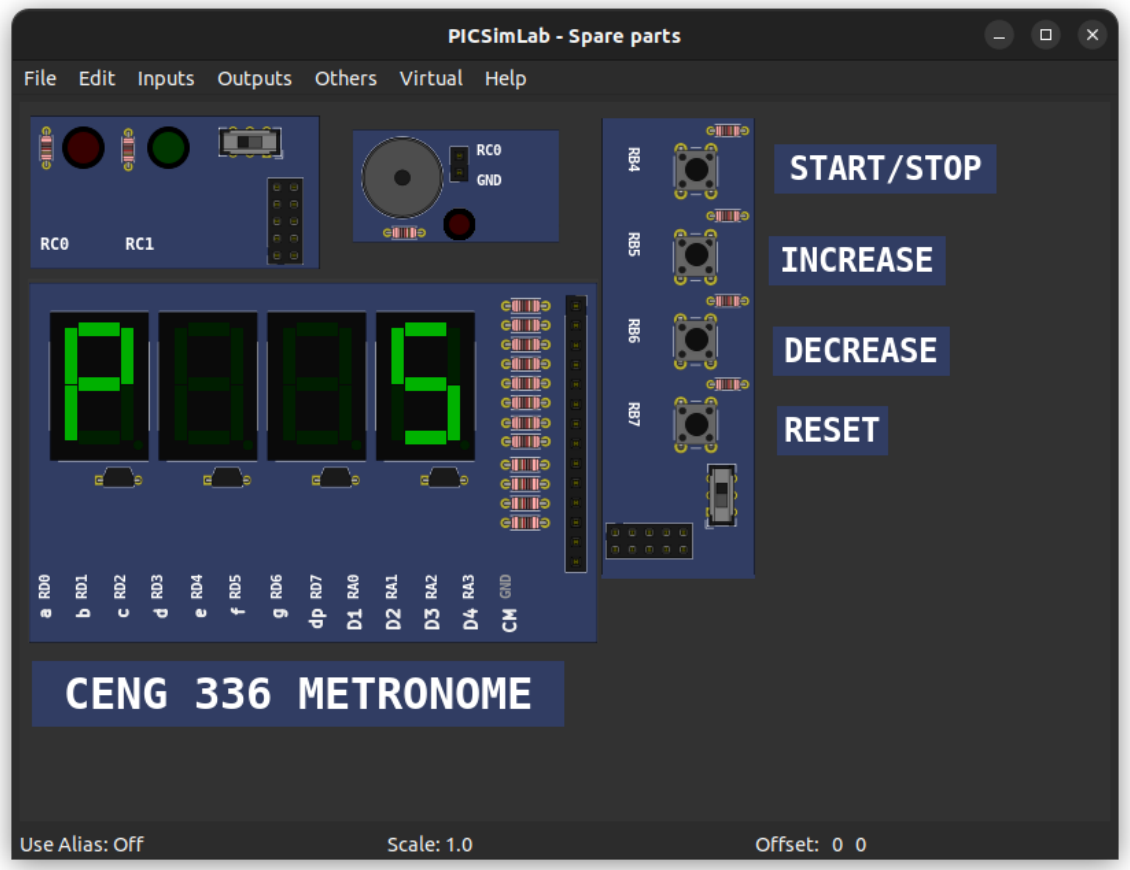

Figure 2: The screenshot of the metronome in paused state in PICSimLab. **Demonstration video:** https://youtu.be/4SBzmRkKeHw

3 Specifications

3.1 Configuration

S-1 The program shall be written for **PIC18F4620** running at **1 MHz instruction frequency**. Note that this is equivalent to **4 MHz clock rate in PICSimLab**.

S-2 Output ports are as follows:

- **RD0-RD7:** Segments a to g and point in 7-segment display.
- **RA0-RA3:** Multiplexer for 7-segment display, leftmost to rightmost. 7-segment display pinout is given in Figure 3.
- **RC0:** Beat LED. Also connected to a buzzer.
- **RC1:** LED indicating the start of a bar.

S-3 Input ports are as follows:

- **RB4:** Start/stop button.
- **RB5:** Increase button. Affects the speed level if paused, bar length if running.
- **RB6:** Decrease button. Affects the speed level if paused, bar length if running.

- **RB7:** Reset button. Affects the speed level if paused, bar length if running.

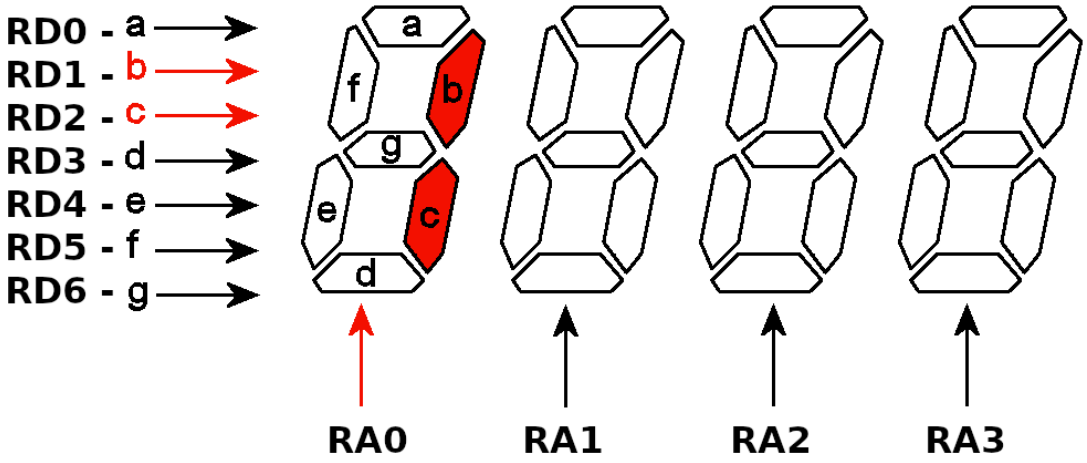

Figure 3: 7-segment display pinout. From left to right, the displays are named **DIS1** to **DIS4**. Note that RD7 (which controls the decimal point) is not shown in the diagram and not used in this assignment. Instructions on 7-segment display operation is provided in Appendix A.

3.2 User Interaction

S-4 A "click" to a button is defined as a change from 1 to 0 in the corresponding pin.

S-4.1 Each pin is 0 by default, becomes 1 when pressed, and becomes 0 when released.

S-5 The program shall use **PORTB on-change interrupt (RB interrupt)** to detect button clicks. (Note that this interrupt triggers on both 0 to 1 and 1 to 0 changes. The program shall respond only to the clicks, i.e., 1 to 0 changes.)

S-6 The program shall **NOT** implement button debouncing. Any change from 1 to 0 is a click, regardless of the duration or previous changes.

S-7 The program shall operate the 7-segment display at 60 FPS (frames per second) without any noticeable flicker. See A.1.

3.3 Initialization

S-8 The program shall initialize all variables it uses, i.e., it shall not depend on variables being zero at the start.

S-9 The program shall start in paused state.

3.4 Paused State

S-10 In paused state, **DIS1** shall display `P`, denoting that the device is in **p**aused state. **DIS4** shall display the speed level. The remaining displays shall display nothing.

S-11 The initial and the default value for the speed level shall be 6. The minimum speed level shall be 1, and the maximum speed level shall be 9.

S-12 In paused state, clicking RB5 and RB6 shall respectively increase and decrease the speed level. They shall not change the speed level if an increase/decrease would set the speed level outside its range.

S-13 In paused state, clicking RB7 shall reset the speed level to 6.

S-14 In paused state, clicking RB4 shall start the metronome.

S-15 When the device returns to paused state, the speed level shall be set to the last value set by the user.

3.5 Metronome Functionality

Table 1: Speed levels and the corresponding beat durations in milliseconds. An error margin of 1 ms is applied to each duration.

Speed Level	1	2	3	4	5	6	7	8	9
Beat Duration (ms)	1000	900	800	700	600	500	400	300	200

S-16 While the metronome is running, **DIS2** shall display the current beat, **DIS3** shall display a dash (only g segment is on), and **DIS4** shall display the bar length. **DIS1** shall display nothing.

S-17 The current beat shall start at 1.

S-18 Timer 0 shall be configured according to the speed level as shown in Table 1.

S-19 Timer 0 interrupt shall be used to advance the current beat by 1. If the current beat exceeds bar length after this increment, the current beat shall be reset to 1.

S-20 The initial and the default value for the bar length shall be 4. The minimum bar length shall be 2, and the maximum bar length shall be 8.

S-21 While the metronome is running, clicking RB5 and RB6 shall respectively increase and decrease the bar length. They shall not change the bar length if an increase/decrease would set the bar length outside its range.

S-22 While the metronome is running, clicking RB7 shall reset the bar length to 4.

S-23 While the metronome is running, clicking RB4 shall return the device to paused state.

S-24 At the start of each beat, RC0 shall be set to 1. After 50 ± 1 ms (regardless of the speed of the metronome), it shall be set to 0.

S-25 When a new bar begins (including the first bar when the metronome starts), RC1 shall be set to 1. After 50 ± 1 ms (regardless of the speed of the metronome), it shall be set to 0.

S-26 Timer 1 interrupt shall be used to clear RC0 and RC1 pins after 50 ms in each beat.

3.6 Implementation Details

S-27 The program shall write to the output ports only when they need to change. The program shall **NOT** perform unnecessary writes to the output registers. This is required for testing.

S-28 The program shall use high priority interrupts only. All interrupts used in this assignment (timer 0, timer 1, PORTB on-change) shall be in high priority mode.

S-29 The program shall have the following labels in the source code:

S-29.1 `main_loop`: The program shall hit this label when it reaches the main loop for the first time.

S-29.2 `timer0_interrupt`: When a timer 0 interrupt occurs, the program shall reach this label in the ISR (Interrupt Service Routine) before clearing the timer 0 interrupt bit.

S-30 The code shall contain explanatory comments and descriptive variable names. Do not name your variables `var1`, `temp2`, etc. **10% of your grade will depend on your code quality.**

4 Testing

A template MPLAB project is provided to you for a quick start. Open this starting project in MPLAB X IDE. [1] In order to use the tester, put the "tests" folder inside the project directory.

Tester usage 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`

Your program needs to abide by the following rules for correct testing:

1. All labels given in S-29 must be present in your source code.
2. Do not use low-priority interrupts.
3. 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.
4. **DO NOT** perform unnecessary writes to output registers. Tests use the `watch` command in `mdb` to track changes in these registers. If you repeatedly rewrite the same data to them, the debugger will be overwhelmed and the test will fail.

[1] If you choose to create your own project, make sure that the project folder name ends with ".X" and your code is in a single assembly file, directly under the project directory.

5. 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.

6. 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 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. You will work in groups of 2.

4. **Submission:** Submit your code as a single `.s` file through ODTUClass.

5. **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.

6. **Grading:** Your submissions will be graded using automated testing, manual PICSimLab evaluation and code inspection.

   (a) Tester gives you a grade out of 90.

   (b) Tester grade will be adjusted after running and testing your code on PICSimLab.

   (c) Remaining 10% of the grade will be determined by your code quality: readabililty, comments, variable names, etc. Note that more than 10% of your grade may be affected if you have implementation errors that are not caught by the tester/PICSimLab. For example, if you use round-robin to handle inputs and fake PORTB on-change interrupts by triggering it manually, your grade will be reduced, even if it works perfectly on PICSimLab and the tester.

A 7-Segment Display Operation

To use the 7-segment display, select a single display using the PORTA pins, i.e., only one of RA0 to RA3 should be 1. Then, to draw the character you want, light the correct segments by setting the corresponding PORTD pins to 1. Notice that PORTD pins are connected to all displays, but it will only affect the display selected via PORTA.

In order to use all 4 displays at once, you should cycle through all displays in quick succession as shown in Figure 4. For example, if you want to render 1234 on the displays, you should select only **DIS1** by using PORTA, write the correct byte to PORTD to turn on the desired segments, and wait for a while. Then you should do the same for **DIS2**, **DIS3** and **DIS4**. This is illustrated in Figure 5.

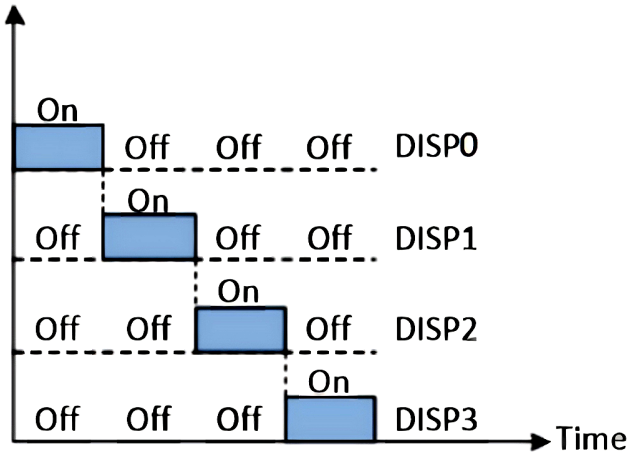

Figure 4: Time vs. display selection graph demonstrating how to operate all 4 displays at once. Only one display should be selected at a time, with PORTD set to the corresponding character. If these on-off cycles are repeated continuously with proper timing, all displays will be operated in a smooth manner without flicker.

A.1 Timing

The refresh rate must be greater than or equal to 60 FPS to create a continuous image without flicker. Therefore, the sum of delays must be smaller than 1/60 seconds, i.e., the total time for the events in Figure 4 must be 16.66 ms. Since we have 4 displays, this means that the delays between display switches must be at most 16.66 ms/4 $\approx$ 4.16 ms.

If you use different delays for each display, the brightness levels will differ between the displays. This is not a desired effect; all displays should have the same brightness level.

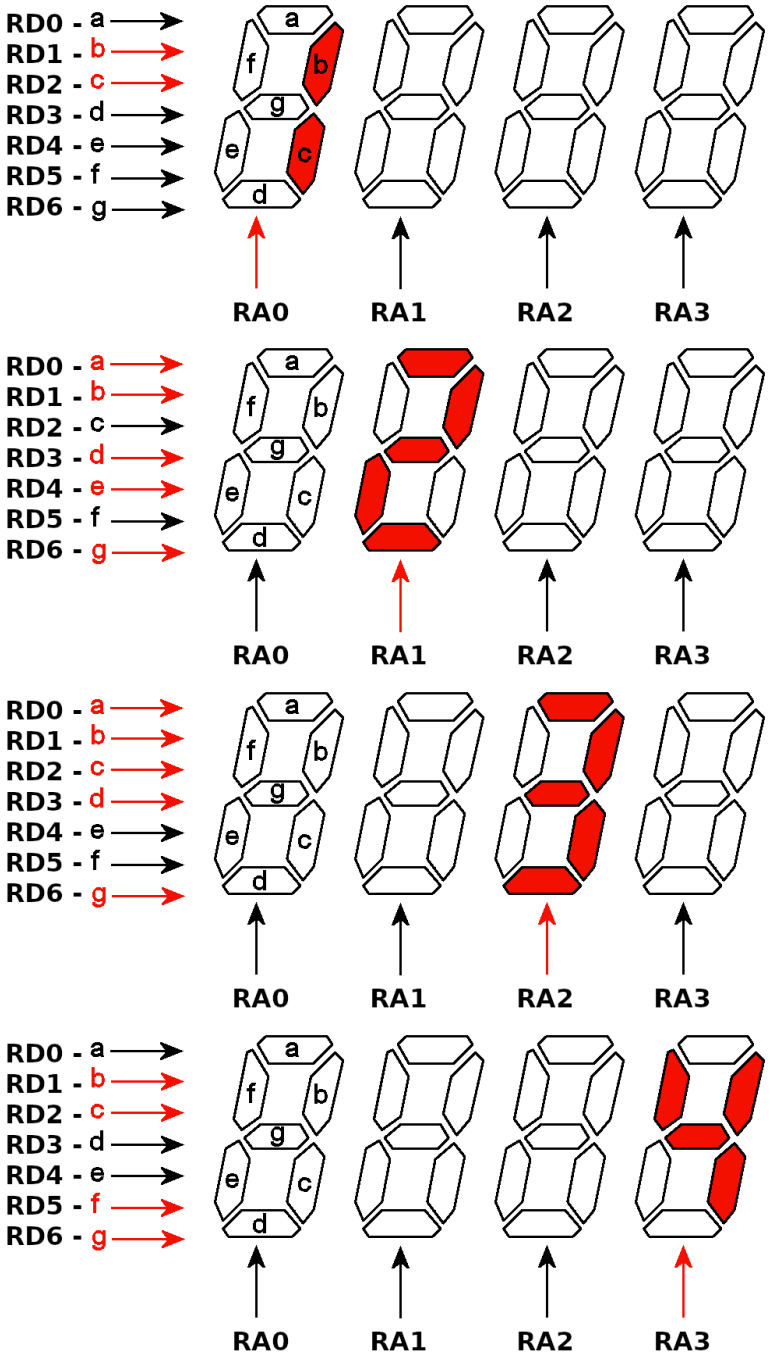

Figure 5: An example sequence demonstrating how to display 1234 on the 7-segment displays. Black and red arrows denote that the corresponding pin is 0 and 1 respectively. To use all 4 displays at the same time, cycle through this sequence continuously with a short amount of time between each display switch, as shown in Figure 4.