Middle East Technical University Department of Computer Engineering

CENG 336

Introduction to Embedded Systems Development

Spring 2020-2021

THE1 - LED Etch a Sketch

Due date: 20.04.2021, Tuesday, 23:55

The purpose of this assignment is to make you familiar with the PIC Assembly and assess your ability to implement button debouncing/timing/other tasks in a round-robin fashion.

1 Overview

In this assignment, you will implement an application that you draw shapes on a 3x8 LED grid by pressing buttons connected to RB[0,1,2,3] pins. The LEDs on the rows are connected to the pins of PORT[A,C,D]. A video is also attached in which you can view the application in action.

2 Specifications

2.1 Application Behavior

The application has 3 phases. For ease of understanding , the buttons will be associated with the following names:

Pin	Button Name
RB2	UP/RIGHT
RB1	DOWN/LEFT
RB0	TOGGLE
RB3	CONFIRM

Table 1: Button Names for Etch a Sketch.

The application should perform the designated action for a button only when that particular button is **released**. In other words, pressing/holding a button should not cause any change, only releasing that button should modify the application behavior.

A line refers to subsequent LEDs in a row being ON. This term does not specify anything regarding other LEDs in that row, they could be ON or OFF. A shape refers to the collection of lines etched in a row since the beginning of the application.

2.1.1 Start-Up

In this phase, initialize your application and then turn all of the LEDs on all of rows ON for 1000±50 msecs, and then turn them OFF. Your application should not be responsive to button presses/releases etc. during this period. Immediately move on to row selection phase after turning off the LEDs with the topmost row selected and no row has any shape drawn on (meaning they are all OFF).

2.1.2 Row Selection Phase

In this phase, the user selects which row to draw a line on. The LEDs on the selected row should blink with intervals of 200±50 msecs. The blinking should override any previously drawn shape on that row.

The user can use UP/DOWN buttons to make the row above/below the selected row the new selected row. If moving up/down is not possible (e.g. pressing UP when the selected row is the topmost row) the action should have no effect.

If a selected row has a previously drawn shape, switching from this row to another row should reveal this shape (which was overriden by blinking when the row was selected).

The TOGGLE button inverts the colors (toggles each LED) in the underlying shape of the selected row. Again, this should be only apparent when the selected row changes as the blinking overrides the shape seen on a row.

The CONFIRM button finalizes the decision on the selected row and transitions the application to drawing phase.

2.1.3 Drawing Phase

In this phase, a candidate line is added to the previously drawn shape in the selected row. This addition is similar to "regular" drawing, the ON LEDs on the candidate line should turn previously OFF LEDs ON. The candidate line is always of length 1 and starts at the left edge of the row when drawing phase is entered, and should be visible in the row along with the previously drawn shape throughout the entire phase.

The TOGGLE button switches between the left or right edge of the line as the selected edge. The right edge is always selected by default when drawing phase is entered.

The RIGHT/LEFT buttons move the selected edge right or left in the row. However, the line length can never be shorter than 1 and the right edge can not move past the left edge / vice versa. As the line is modified, the change should be visible in the row.

The CONFIRM button permanently etches the line into the row shape and should transition the application back to row selection phase with the current selected row as the candidate selected row again (meaning the row should start blinking).

2.2 Button Debouncing

Contact Bouncing is an issue that happens in real circuitry due to physical nature of the buttons. To guard against bouncing, implement features in your button handlers such that they do not respond to irregular pulses from bouncing (one way to do it is measuring how much time passed and eliminating very short pulses of period shorter than 10 msecs).

2.3 Debug Labels

Your application behavior will be blackbox checked by debugger scripts along with light glassbox checking for any errors that debugger scripts might miss. Some of these scripts are provided to you. Since the debugger scripts cannot know when to stop execution and check for values by themselves, you will need to add labels to certain locations in your program:

- `init_complete`: Add this label right after your last initialization instruction. You should hit this label exactly once.
- `sec_passed`: Add this label right after the start-up phase ends. You should hit this label exactly once.

- `msec200_passed:` You should start hitting this label every 200±50 msecs after you hit `sec_passed`, regardless of the phase of your application/button states. This label should also drive your application in a way that if you are in row selection phase, the selected row leds should be ON after every 1st,3rd,5th... hit of this label (including hits in drawing phase), and they should be OFF after every 2nd,4th,6th... hit (including hits in drawing phase).
- `rb[0,1,2,3]_released:` You should hit these labels when you detect a proper release that is not a bouncing noise for each button.
- `main:` This should be the label of your main loop in your application.

3 Setup

You should configure your board by opening PICSimlab and then:

- Click File > Load Workspace and then pick `the1_2021.pzw`.
- Click Modules > Spare Parts , in the new window, File > Load Configuration, and then pick `the1_2021.pcf`.

Create a project to be compiled with C18 and create a single source file `the1.asm`. You can fetch the config directives and other boilerplate from the recitation code.

To run the debug tests in Linux, you will need `Python>=3.6`. `mdb` and `make` should already be in your `PATH` environment variable. Simply copy the `tests` directory under your project and within `tests`, run:

```
> ./build_and_run_all_tests.sh
```

To run the debug tests in Windows, you will need `Python>=3.6` (also add it to your `PATH` environment variable). You will also need to add `make` and the debugger `mdb` to `PATH`, which come installed with your MPLAB X IDE, check the documentation to learn where they reside. After these steps, simply copy the `tests` directory under your project and within `tests`, run with `cmd`:

```
> ./build_and_run_all_tests.bat 2>nul
```

It is highly encouraged that you take apart debugger tests and write your own. You can check `mdb` documentation regarding the commands (you can skip on stimulus control language however, it is not necesarry at this stage). Two of the debugger tests are programmatically generated and since they are quite hard to read, the python scripts that use the generator are also provided for you to follow easily. It would simplify your job a lot if you try to programmatically generate your own tests as well (the generator itself is not shared with you as it spoils some implementation details for you to figure out).

4 Other Regulations and Submission

- Your application should be implemented in **PIC Assembly for MPASM** (Install C18 compiler and pick it when you create a project to use it) for **PIC18F4620 with a 40 MHz clock**. You can slow the clock down in PicSimLab to make it work in real time as much as possible (clock text being red indicates PicSimLab is not able to run real time at the indicated frequency) but remember that all of the specifications in this text are made for a 40 MHz clock.
- Use of interrupts and timers are not allowed.
- Your application should be implemented in a way that tasks are visited in a round-robin fashion. No task is allowed to block your application in row selection or drawing phases, your application should be always responsive to all the button releases and a button action should also not freeze it (this includes holding a button).

- Pressing multiple buttons at the same time is not a use case for this application and will not be checked in the blackbox/glassbox tests.
- Your application behavior will be blackbox checked by debugger scripts, along with light glassbox checking for any errors that debugger scripts might miss.
- When you are done, submit `the1.asm` to ODTUCLASS.
- **This is an individual assignment. Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homeworks, or the Internet. The violators will be punished according to the department regulations.**
- Please submit your questions publicly on the course forum if your question does not include parts of your solution. If your question does contain some of the solution, send an email to onem@ceng.metu.edu.tr.