15-348: Embedded Systems, Fall 2022
Homework 3: *State* of the Art Texting
Due: Sunday, September 11th, 2022 by 10:00pm

Part	Quantity
Launchpad	1
16-Button Keypad	1
10KΩ Resistor	4

Introduction

This is a programming homework. You must submit your code to Gradescope by **Sunday, September 11th, 2022 by 10:00pm**. Only submit the files indicated below. You should also demo your code to the one of the course staff members on or before Thursday, September 15th during one of the available slots. You can only demo once. More instructions about the demos will be announced during class and on Piazza.

Learning Objectives:

Design and implement a finite state machine.

Getting Started:

1. For each task, create a new project in CCS, as explained in the tutorial on the website.
2. Submit only the source code files that you modified (i.e., not the entire project, binaries, etc).

Some Important Notes:

1. You should read the entire assignment before starting. This will help you predict what your final circuit will look like, what sort of functionality your software will need, etc.
2. We expect that your implementation exactly matches the specifications indicated below.
3. Although we might be lenient with unspecified behaviors, you should justify your implementation choices.
4. If you are unsure about the specifications, you should ask questions on Piazza or directly to the course staff.
5. Reading, understanding, and eliciting the requirements and specifications is also part of the assignment.

Starter Code:

The starter code for both parts of this homework can be found at
https://github.com/CMUQ-15-348/hw3-startercode

Task 1: Keypad Input (30 pts)

In this task, you will program the controller so that it can receive input from a 4x4 keypad. A 4x4 keypad consists of sixteen keys arranged in a matrix of four rows and four columns as shown in the figure below:

In an input device like this keypad, the buttons are not connected individually to input pins on a controller since that would require many input pins (consider what would happen on a keyboard with 101 keys). Instead, the interface to the keypad consists of Input/Output divided into rows and columns. The details of how this interface works can be found at:
http://pcbheaven.com/wikipages/How_Key_Matrices_Works/
Read the above reference and get yourself familiar with the basics of this interface.

The keypad provided to you is divided into rows and columns as given below:

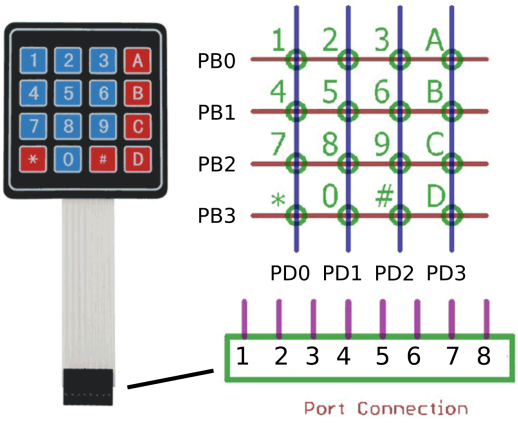

Figure 1: Keypad pinout

First, make a circuit as shown in the figure below:

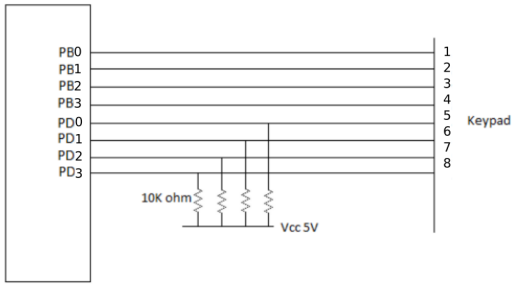

Figure 2: Connection between LaunchPad and Keypad

To determine which key is pressed, follow the procedure described below:

1. `PB0-PB3` are configured as output ports
2. `PD0-PD3` are configured as input ports
3. You will enable one row at a time and then read the columns to see which key was pressed.
4. Writing 1 to a row disables it and writing a 0 enables it. For example, if we set `PB3`, `PB2`, and `PB1` to 1 and `PB0` to 0, this will enable row1. Once a row is enabled, you need to check if any key in that row is pressed. If a key is pressed, its corresponding bit at the input pins will become 0, otherwise, the pins would be 1. For example, if row1 is enabled and if Button A is pressed, then the value at `PD3` would be 0, and the values at `PD0`, `PD1`,

and `PD1` would be 1. You need to cycle through each row and check if any key in that row is pressed.

5. You will need to debounce. Remember our new debouncing approach discussed in class.

6. You are given a file called `serial.c` and `serial.h` in the starter code. Use the functions in these files to write the key pressed on the keypad to the serial port so you can view it on your PC.

7. Check the resources page on the website for instructions on how to use the serial port from your computer.

Task 2: Going Old School (20 pts)

In this task, you will design and implement a state machine that will allow you to enter text using the 4x4 keypad, similar to the techniques used in old phones:

Figure 3: Old phone keypad

If you look at the old phone keypads, each key had a sequence of letters printed next to the main key number: Number 2 contains ABC, Number 3 contains DEF, and so on. To enter B, I can press key 2 twice, in quick succession. If I need A, I can press 2 once and then pause for a little bit for the system to register that I wanted A.

Your task is to design a state machine that replicates the behavior of text entry on old phones following the following principles.

- If a key is pressed n times, followed by a pause of 1 second, the system will treat that as the end of the sequence for that button and decode the entered character.
- If a key is pressed n times, and then a different key is entered, this will indicate the end of the previous entry and the start of a new entry. The machine should register the output of the previous entry.

This problem is ideally setup for a state machine, and you **must** solve it using a state machine. Along with your code submission, **you must also submit a diagram of your state machine that shows all of the states and transitions related to the letters on key 2**. The other letters/keys will be very similar, so you are not required to completely write-out the entire state machine for all keys, as it will require a lot of states.

Implement the entire state machine on the controller using a state table as demonstrated in class. You can reuse the code of Task 1 as the input source. As each character is typed, it should be displayed on the terminal on your computer through the serial port.

Additional Guidance

Drawing Your State Machine

You are *must* draw and submit the required partial state machine for Part 2. You should do this before you start coding. The entire problem is much, much easier once you have a properly designed state machine.

Dealing with Time

For Task 2 of this assignment, the amount of time passed will be part of the input to your state machine. There are multiple ways you can handle tracking the amount of time passed, including keeping a simple counter inside your main event loop or setting up an interrupt-driven timer. You can decide for yourself how you want to handle it.

Debouncing

You will need to debounce your inputs from the keypad. It is now required that your debouncing not "stall out" your program while it is running, so remember our new debouncing approach based on state machines discussed in class.

General Tips

- Work incrementally, testing each small piece of functionality as you go. For example, your first step might be simply seeing if you can detect when a button is pushed on the keypad.
- If you spend too long debugging something, and can't find the problem, start over. Rewire your circuit and rewrite the code from scratch. It shouldn't take you that long the second time, and you may be surprised to find that it works.