Henry O'Callaghan
Kevin Vander Heyden
EE2361 Lab 3 Deliverables

In the original test of our circuit, we checked to see if just the display was working and found that the circuit worked on the first try and we had flipped a couple bits for our seven-segment display and just had to adjust those to get it working properly. The next test we performed was the final circuit using both the keypad input and seeing if it correctly displayed on the seven-segment display. This test resulted in quite a bit of debugging. In doing our testing two of the wires for the display got flipped, which took a bit of debugging to determine was the issue. Also, our keypad values that were being output were off by one column. We started the debugging for this by setting the keypad input value to a specific number in the code to make sure that how we were taking the input and sending it to the display was correct. We then realized that two of the bits were flipped in our display and that the values that seemed like nonsense were actually just values that were one column off and with two display bits flipped. In doing our testing two of the wires for the display got flipped which accounted for the flipped bits and we were able to fix the column shift by adjusting how we indexed the keypad.

The display code was reasonably straightforward to make work. Initially, we use a predefined mask to set all of the LEDs connected to LATB off. Then we OR LATB with the pattern we want, based on the input to the function and a lookup table to make the code cleaner. We also use a digit enum to make the software more readable.

```
void showChar7seg(char myChar, enum DIGIT myDigit) {
        LATB &= CLEAR_ALL_DIG_AND_SEGS_BITS_MASK;
        LATB |= myDigit | patterns[myChar];
}
```

The keypad code, as described above, was trickier. The below code is the first part of our readKeyPad function. One at a time, it sets one of the columns low.

```
unsigned int readKeyPadRAW(void) {
  static unsigned short int pressed = 0;
  static unsigned int key = 16;
  int i;
  int num;
  for (i = 0; i < 4; i++) {
```

```
LATA &= 0b1111111111110000;
LATA += 0b1000 >> i;
int a;
for(a=0;a<10;a++)
{
asm("nop");
}
```

This lince checks to see if any rows are pressed by reading RB15 - RB12

```
num = (!_RB15 << 3) | (!_RB14 << 2) | (!_RB13 << 1) | !_RB12;
```

Lastly, below is a snippet of a series of switch case statements to return the pressed key. In hindsight, this could've been implemented cleaner by following the implementation showcased in discussion.

```
if (num > 0) {
    if (!pressed) {
    if (i == 1) {
    switch (num) {
    case 0b1000:
    key = 0x10;
    break;
    case 0b0100:
    key = 7;
    break;
    case 0b0010:
    key = 4;
    break;
    case 0b0001:
    key = 1;
    break;
```

```
}
```