**KEYPAD INTERFACING**

**Aayahna M. Herbert**

**Jacovia Cherry**

**Candice Sanders**

**October 3, 2017**

**ECE 3720 Section 007**

**Microcontroller Interfacing Laboratory**

**Abstract:** A circuit was built in order to send inputs to the keypad from the microcontroller. In order for the keypad to properly work, the circuit and code had to be set up in a very specific manner. When a button on the keypad was pushed, the 4 LEDs would have lit up by the MC corresponding to the number in binary. While the code and circuit was correct, the keypad did not respond to the MC when its buttons were pressed.

**INTRODUCTION**

The purpose of this lab was to design a circuit that sends eight inputs to the microcontroller from the keypad. The microcontroller would then output signals to four LEDs that represent binary numbers 0 through 15 when turned on. When done correctly, as the buttons on the keypad are pressed, the LEDs should light up the corresponding the binary number that was pressed and stay on until a new button is pressed.

**EXPERIMENTAL PROCEDURES**

The equipment used include one PIC32 MC, NI-ELVIS II board, and keypad. In order for the experiment to have positive results, the figure shown below. The first thing to be done, to avoid forgetting it later, is to set the PORTCs to inputs and outputs; C0 through C3 are outputs while C4 through C7 are inputs along with pins B0 through B3 are the four outputs. Pins B0 through B3 are used in order to avoid shifting bits down to 0 through 3. Afterwards, the correct keypad pins need to go their corresponding PORTC pins; key 0 to C1, key 1 to C2, key 2 to C4, key 3 to C3, key 4 to C5, key 5 to C6, key 6 to C7, and key 7 to C0. Next, the keypad inputs’ pull up registers should then be turned and set to one using the CNENx and CNPUx registers, respectively in order to use the pull up resistors. When finishing the rest of the code, the two unsigned character arrays holding 16 elements should be defined as “mask” and “key”; mask will hold the hex coordinates for each keypad value and key will hold the keypad values corresponding to its hex coordinates in mask. Before testing the board, the while loop section of the code should be set up in a way that will go through each element in the mask array, equate the value to LATC, set PORTC, and check to see if that mask value has been turned on and equal to the set PORTC. When the mask value is equal to the set value, the microcontroller will set LATB to the corresponding element in the key array corresponding, sending that value to the LED outputs of the MC to make the lights show the binary value of the selected keypad button.

**RESULTS**

After the circuit was hooked up and ready to be tested, the program detected no compiling errors when ran so the board could then be tested. While running tests, when the potentiometer was rotated, the LED lights lit up out of order. The correct results were never found due to running out of time and, therefore, not being able to the CPOL for each comparator.

**DISCUSSION**

Problems arose when the correct LEDs didn’t light up when keypad button is pressed. The LEDs themselves were tested individually to make sure none of the ones in use were broken and the potentiometer worked because it was changing lights, just not in correct order. The code and wiring were second-checked by the TA and the wires all came from a new pack recently purchased to avoid the possibility of dead wires being used and affecting the lab results. An error source was not able to be found other than the possibility of an internal issue with the circuit board itself.

**CONCLUSIONS**

In general, the take-away of this lab was to get hands-on experience with I/O interfacing with the keypad and understanding how it and the MC work together to output a value while using loops to scan and masks.

**FIGURES AND TABLES**

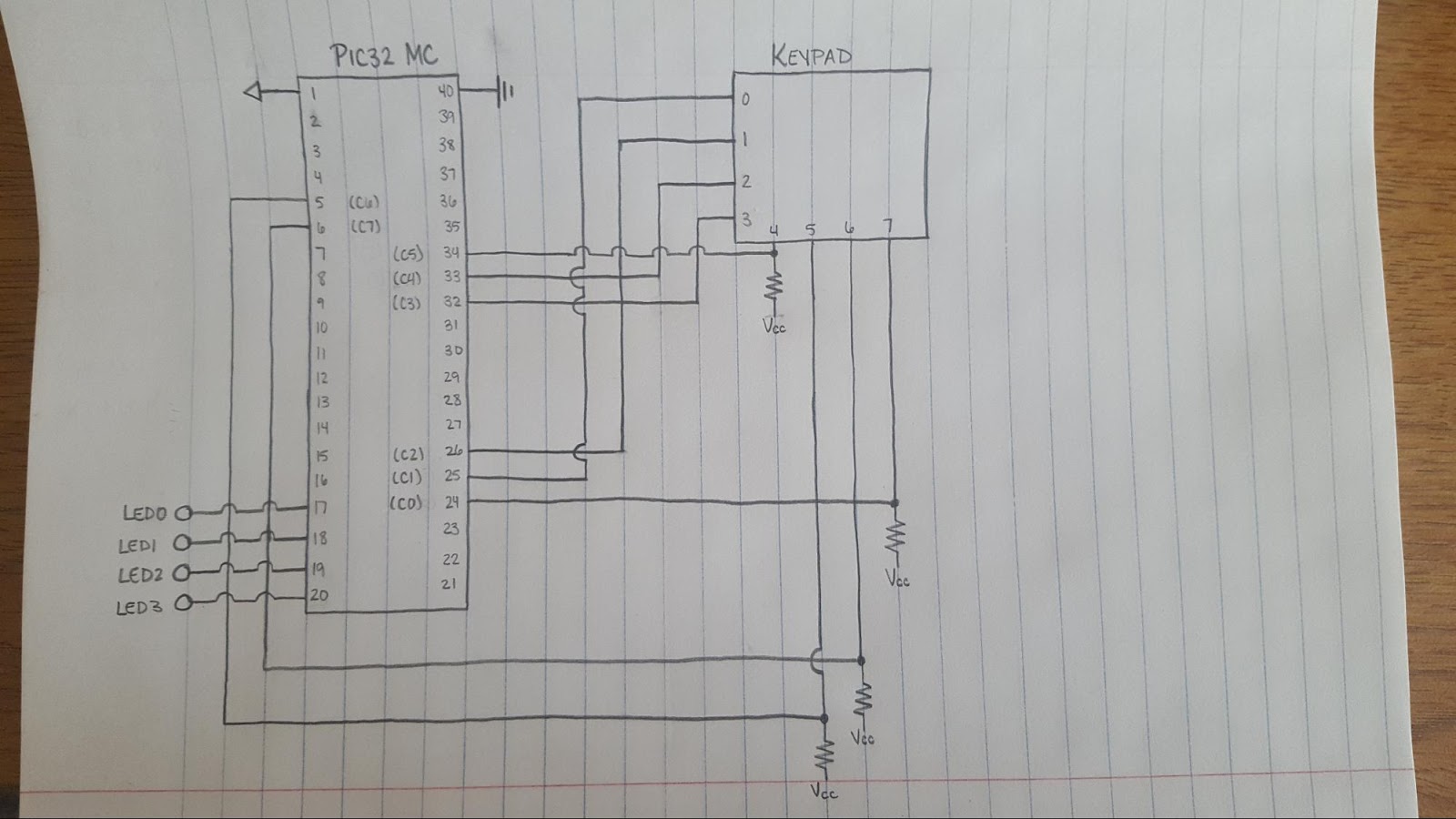


Figure 1: Circuit Schematic

**CODE**

#include<plib.h>

void main (void)

{

TRISCbits.TRISC0 = 0;

TRISCbits.TRISC1 = 0;

TRISCbits.TRISC2 = 0;

TRISCbits.TRISC3 = 0;

TRISCbits.TRISC4 = 1;

TRISCbits.TRISC5 = 1;

TRISCbits.TRISC6 = 1;

TRISCbits.TRISC7 = 1;

TRISBbits.TRISB0 = 0;

TRISBbits.TRISB1 = 0;

TRISBbits.TRISB2 = 0;

TRISBbits.TRISB3 = 0;

CNENCbits.CNIEC4 = 1;

CNENCbits.CNIEC5 = 1;

CNENCbits.CNIEC6 = 1;

CNENCbits.CNIEC7 = 1;

CNPUCbits.CNPUC4 = 1;

CNPUCbits.CNPUC5 = 1;

CNPUCbits.CNPUC6 = 1;

CNPUCbits.CNPUC7 = 1;

int i;

unsigned char mask[16] = {0xEE,0xDE,0xBE,0x7E,

0xED,0xDD,0xBD,0x7D,

0xEB,0xDB,0xBB,0x7B,

0xE7,0xD7,0xB7,0x77};

unsigned char key[16] = {1,2,3,0xA,

4,5,6,0xB,

7,8,9,0xC,

14,0,15,0xD};

while(1)

{

for (i=0; i<16; i++)

{

LATC = mask[i];

if ((PORTC & 0xFF) == mask[i])

LATB = key[i];

}

}

}