

**EEET7050C & 7058C  
Microprocessor Applications**

**S3834951  
Mitchell Reynolds**

**Student Project Report**

## Project outline

This project involved building a board to suit our design, as well as programming both a microcontroller and PLD in ATMEL Studio and Quartus II(respectively). The board takes user input on a 4x4 keypad, and when a key is pressed it will display that number on the right-most of four seven-segment displays and shift the existing numbers to the left.

The two digital chips used on my board are:

- ATMEGA32L (microcontroller)- which provides master clock signal, sends data from EN and receives data at its MUX in.
- EPM7032AE (PLD) – which has our logic programming saved into it
- puts the 1-bit data input from ATMEGA32L into an 8-bit shift register (logically)
- decodes this 8 bit number and outputs it to a MUX, Walking Zero Decoder, and a 7-segment decoder (logically)
- outputs to the 7-segment display and digit selectors (physically)

Block diagram of the system:

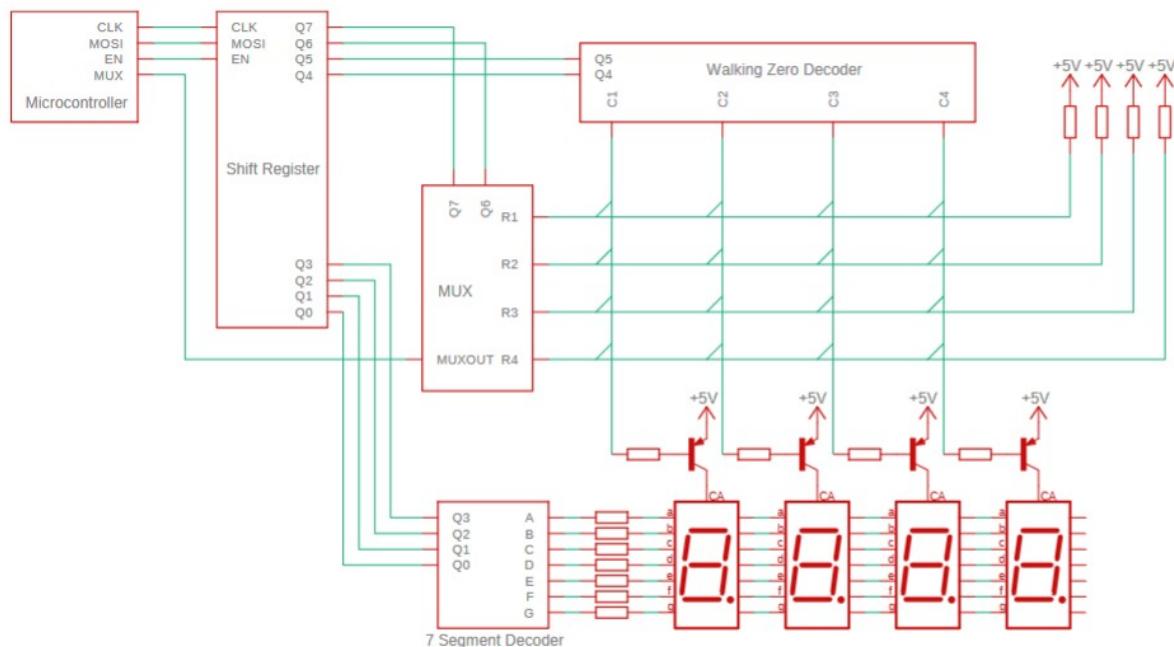
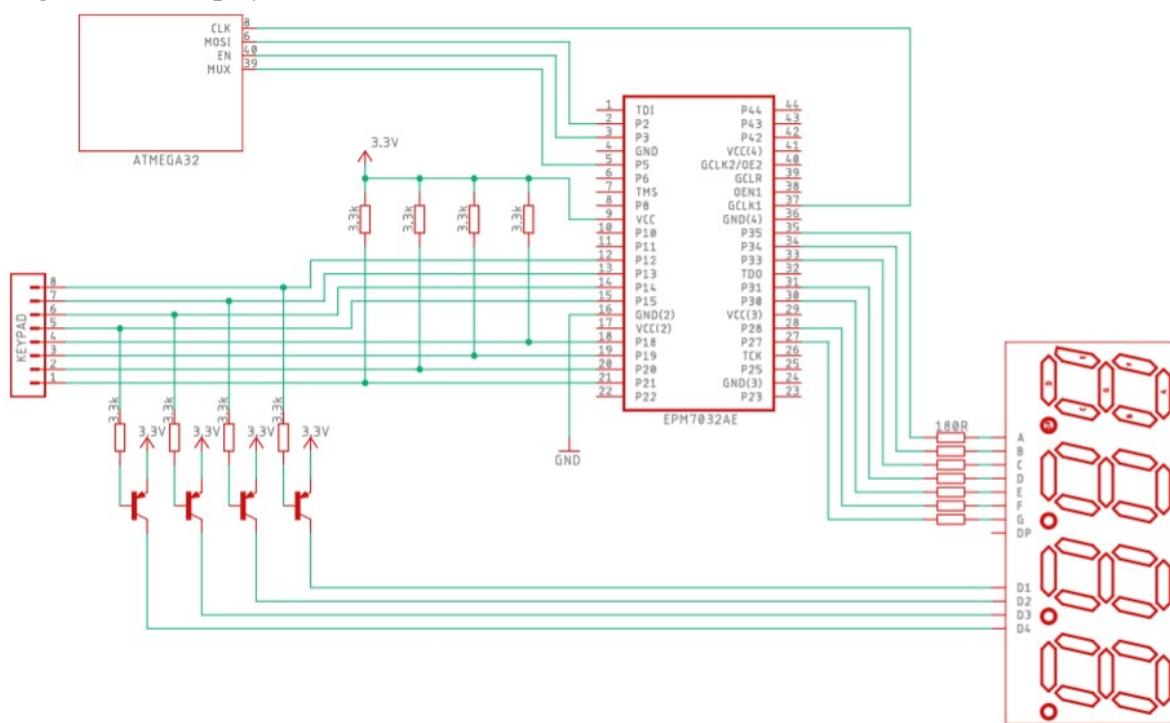


Diagram of the physical circuit connections:



## Notes re: logic operations for 7-segment display, truth tables

7-Segment display logic for common anode:

Common anode - 0 turns segment ON

Decimal	B <sub>4</sub> B <sub>2</sub> B <sub>1</sub> B <sub>0</sub>	a	b	c	d	e	f	g
0	0 0 0 0	0	0	0	0	0	0	1
1	0 0 0 1	1	0	0	1	1	1	1
2	0 0 1 0	0	0	1	0	0	1	0
3	0 0 1 1	0	0	0	0	1	1	0
4	0 1 0 0	1	0	0	1	1	0	0
5	0 1 0 1	0	1	0	0	1	0	0
6	0 1 1 0	0	1	0	0	0	0	0
7	0 1 1 1	0	0	0	1	1	1	1
8	1 0 0 0	0	0	0	0	0	0	0
9	1 0 0 1	0	0	0	0	1	0	0
10	1 0 1 0	1	1	1	1	1	1	1
11	1 0 1 1	1	1	1	1	1	1	1
12	1 1 0 0	1	1	1	0	0	0	0
13	1 1 0 1	1	1	1	1	1	0	0
14	1 1 1 0	1	1	1	1	1	1	0
15	1 1 1 1	1	1	1	1	1	1	1

K-maps for common anode 7-segment logic:

K-maps for common anode truth table					Project
	B <sub>3</sub>	B <sub>2</sub>	B <sub>1</sub>	B <sub>0</sub>	
a	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	
	00	0 0 0 0	00	0 0 0 0	
	01	1 0 0 0	01	1 0 0 0	
	11	1 1 1 1	11	1 1 1 1	
	10	0 0 1 1	10	0 0 1 1	
	$B_3B_1 + B_3B_2 + B_2\bar{B}_1\bar{B}_0 + B_3B_2\bar{B}_1B_0$				
b	B <sub>3</sub> B <sub>2</sub>	00 01 11 10	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	
	00	0 0 0 0	00	0 1 1 0	
	01	0 1 0 1	01	1 0 1 0	
	11	1 1 1 1	11	1 1 1 1	
	10	0 0 1 1	10	0 1 1 1	
	$B_3B_1 + B_3B_2 + B_2\bar{B}_1B_0 + B_2B_1\bar{B}_0$				
c	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	
	00	0 0 0 1	00	0 1 1 1	
	01	0 0 0 0	01	0 0 1 0	
	11	1 1 1 1	11	1 1 1 1	
	10	0 0 1 1	10	0 0 1 1	
	$B_3B_1 + B_3B_2 + \bar{B}_2B_1B_0$				
d	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	
	00	0 0 0 0	00	0 0 0 0	
	01	1 0 0 0	01	1 0 0 0	
	11	1 1 1 1	11	1 1 1 1	
	10	0 0 1 1	10	0 0 1 1	
	$B_3B_1 + B_3B_2 + B_2\bar{B}_1B_0 + B_3\bar{B}_2\bar{B}_1$				
e	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	
	00	0 0 0 0	00	0 1 1 0	
	01	0 1 0 1	01	1 0 1 0	
	11	1 1 1 1	11	1 1 1 1	
	10	0 0 1 1	10	0 1 1 1	
	$\bar{B}_3\bar{B}_2\bar{B}_1B_0$				
f	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	B <sub>3</sub> B <sub>2</sub> B <sub>0</sub>	00 01 11 10	
	00	0 0 0 1	00	0 1 1 1	
	01	0 0 0 0	01	0 0 1 0	
	11	1 1 1 1	11	1 1 1 1	
	10	0 0 1 1	10	0 0 1 1	
	$B_3B_1 + B_3B_2 + B_1B_0 + \bar{B}_3\bar{B}_2B_1$				
	$+ \bar{B}_3\bar{B}_2B_0$				

Walking zero decoder truth tables:

**C<sub>1</sub>**

<del>Q<sub>5</sub> Q<sub>4</sub></del>	0	1
0	0	1
1	1	1

$C_1 = Q_5 + Q_4$

**C<sub>2</sub>**

<del>Q<sub>5</sub> Q<sub>4</sub></del>	0	1
0	1	0
1	1	1

$C_2 = Q_5 + \overline{Q_4}$

**C<sub>3</sub>**

<del>Q<sub>5</sub> Q<sub>4</sub></del>	0	1
0	0	1
1	0	1

$C_3 = \overline{Q_5} + Q_4$

**C<sub>4</sub>**

<del>Q<sub>5</sub> Q<sub>4</sub></del>	0	1
0	1	1
1	1	0

$C_4 = \overline{Q_5} + \overline{Q_4}$

Walking zero decoder  
equations:

$$C_1 = Q_5 + Q_4$$

$$C_2 = Q_5 + \overline{Q_4}$$

$$C_3 = \overline{Q_5} + Q_4$$

$$C_4 = \overline{Q_5} + \overline{Q_4}$$

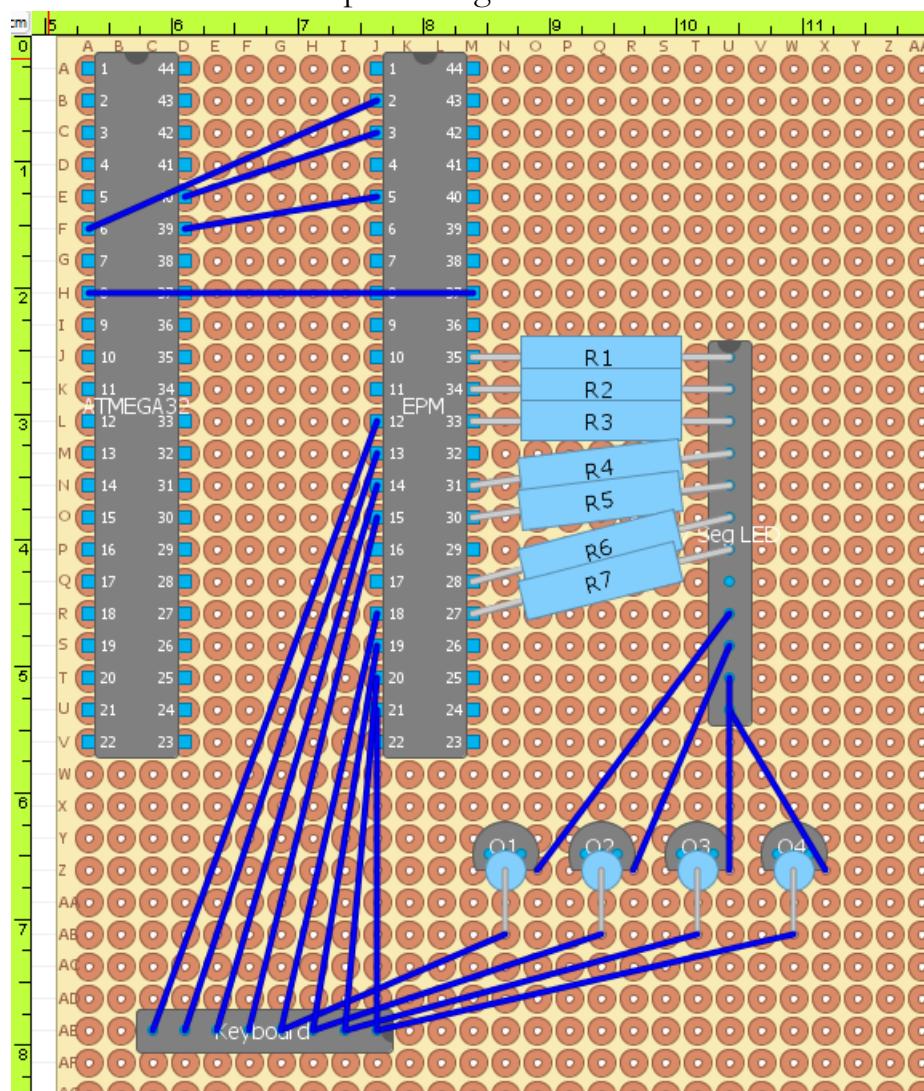
## Building the board

Building the board required careful planning and study of the diagrams. Planning was done using software, but this proved somewhat insufficient due to the complexity of the circuit and the rudimentary nature of the software.

More complex software like Eagle could perhaps been of use.

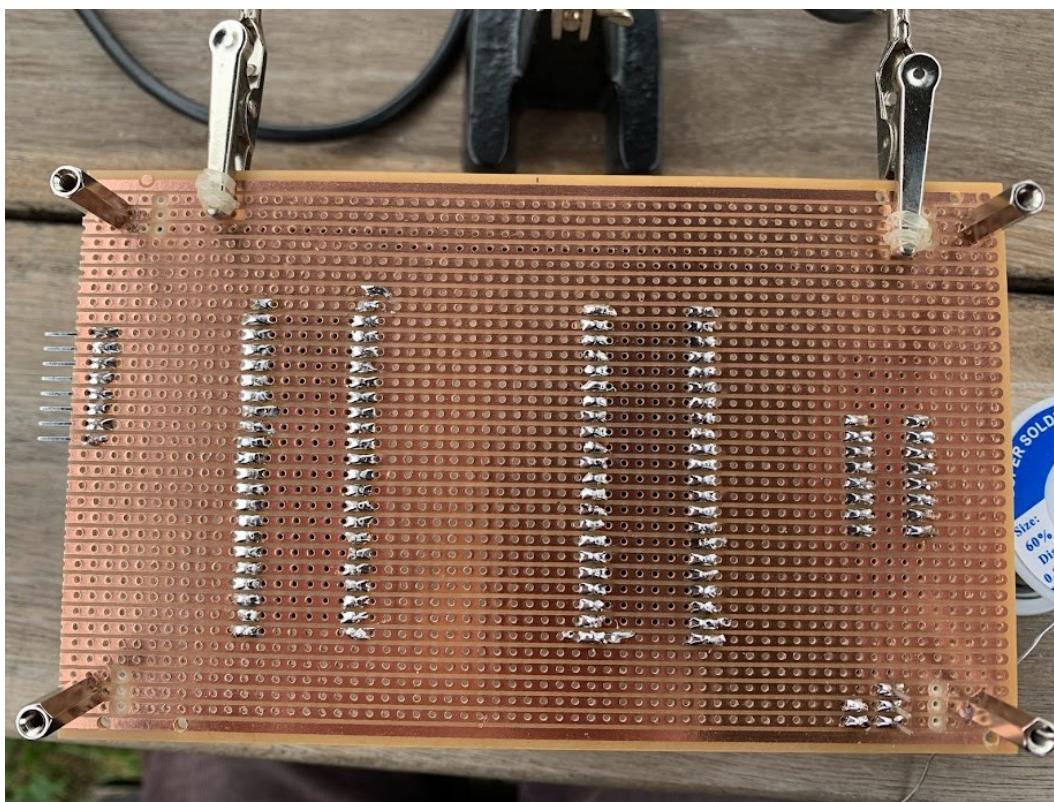
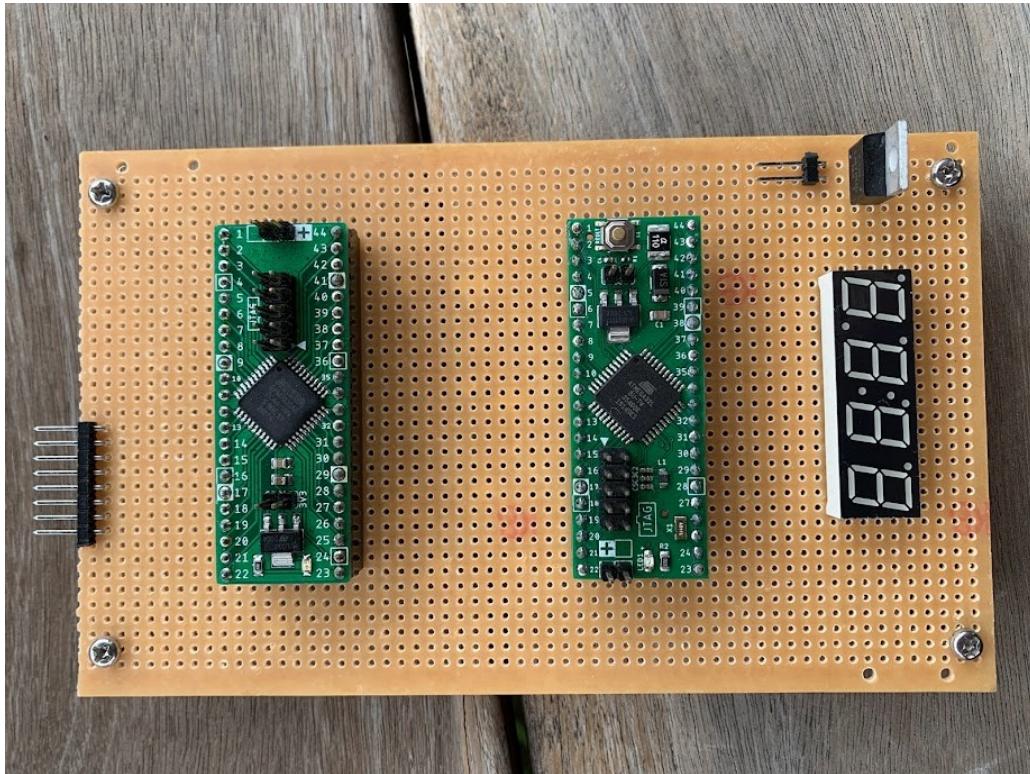
The planning software is for building guitar pedals, and while useful for a rough idea, I did not complete a plan on software, and instead meticulously planned the board physically before soldering (but the end result does resemble the plan below).

Screenshot of software plan using DIYLC software:

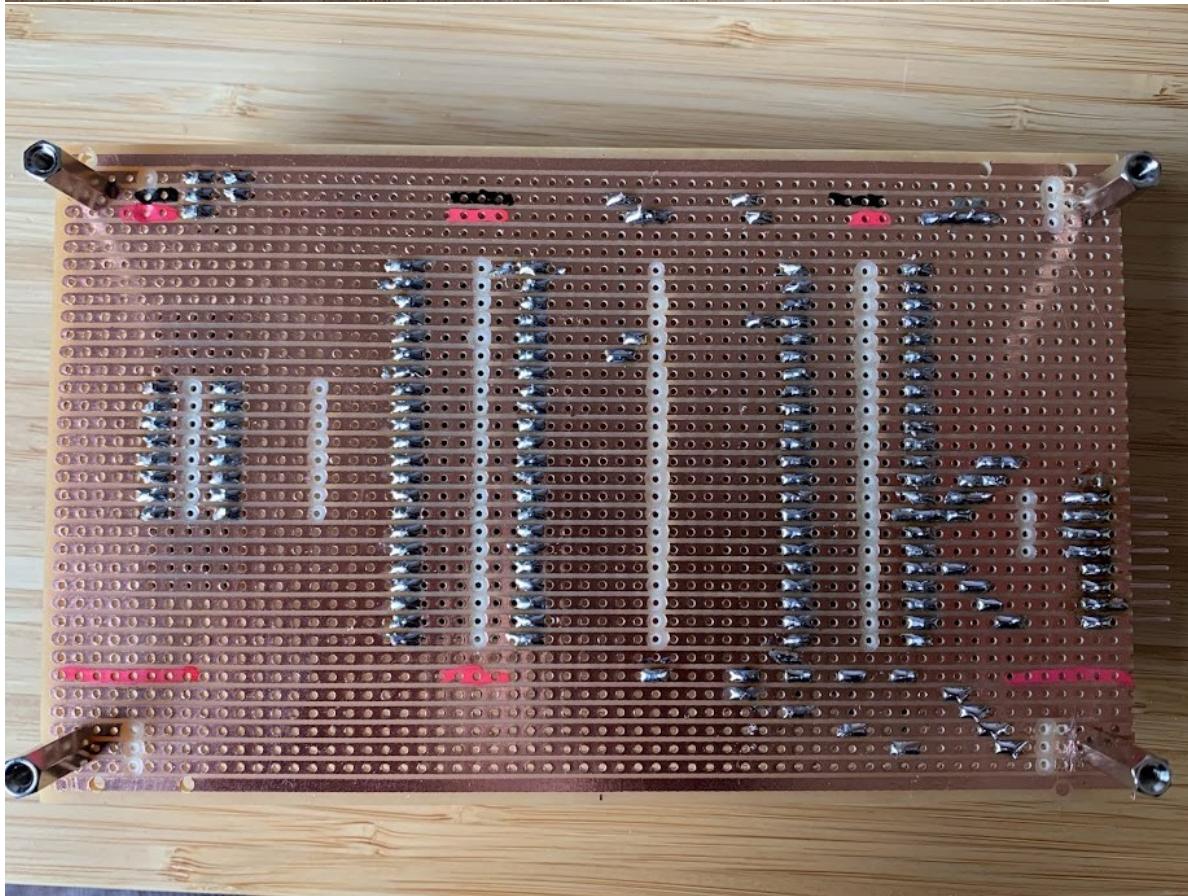
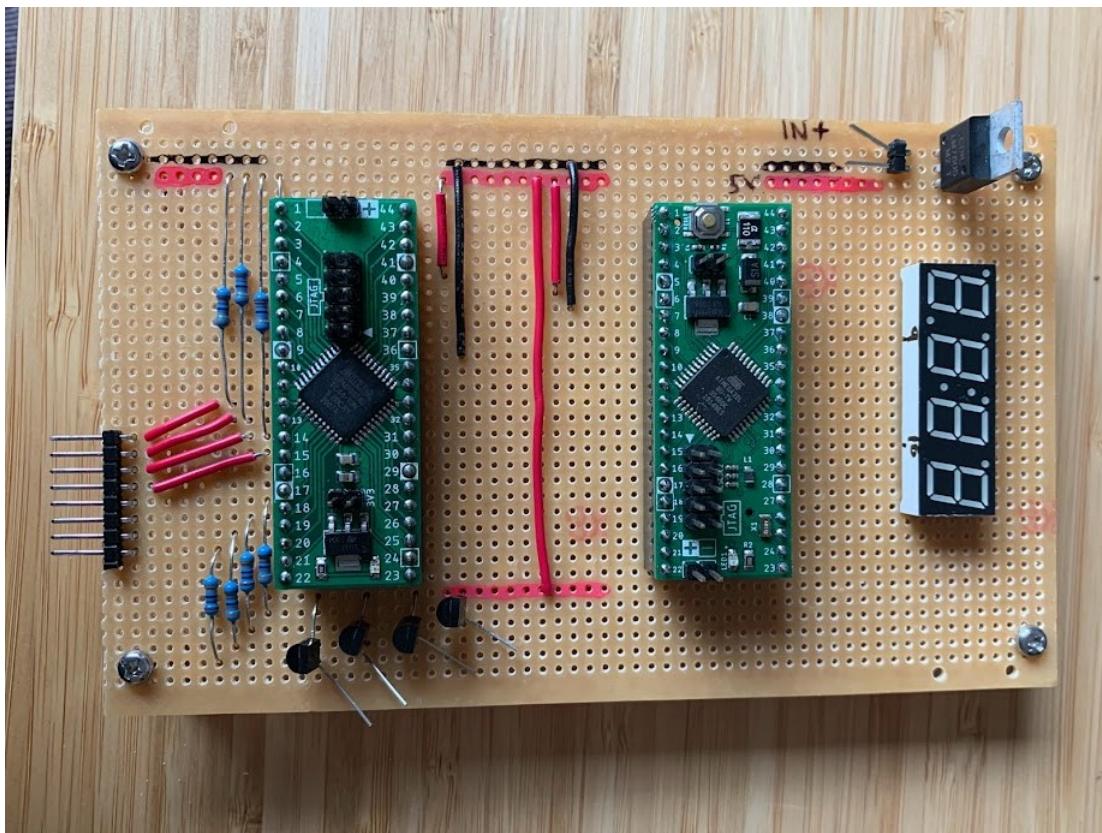


## Build development:

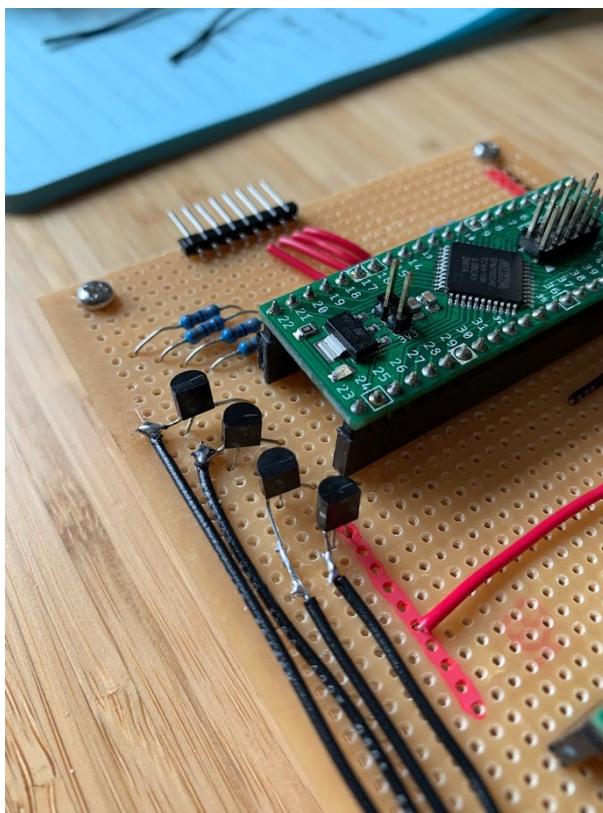
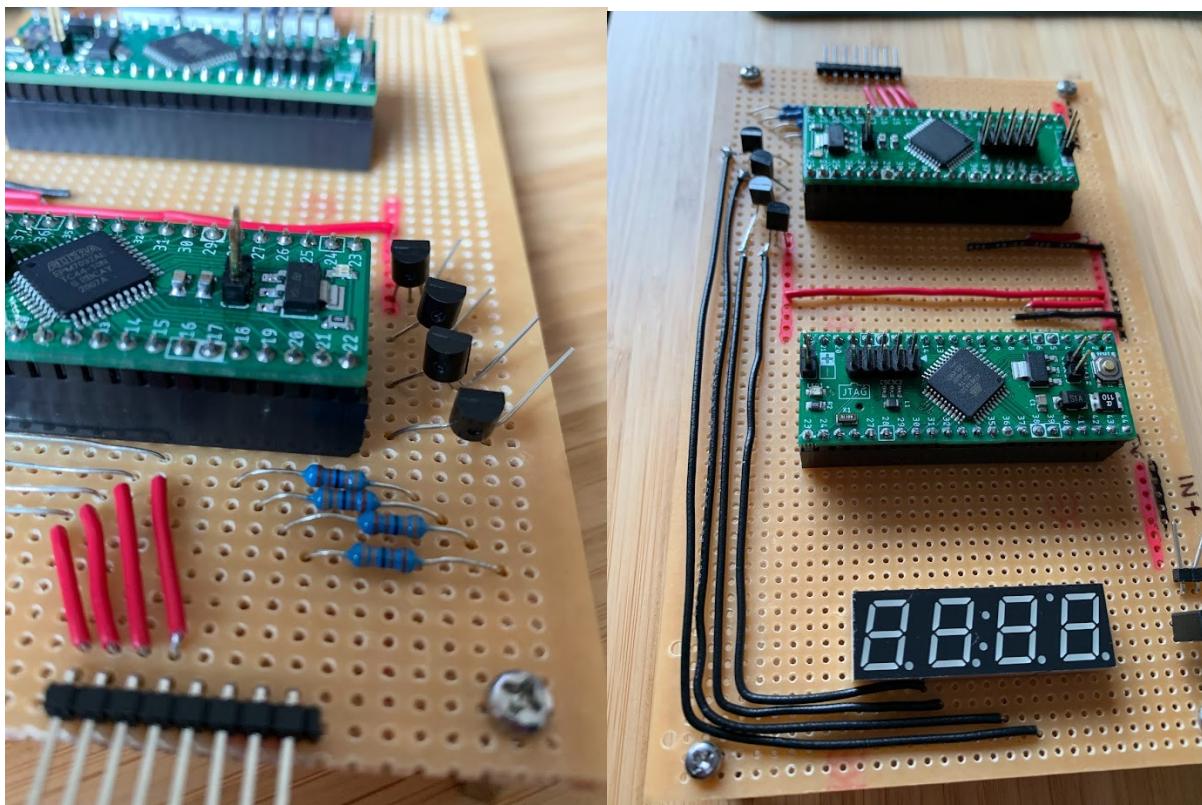
Soldering only the boards, 7-segment display, power, and keypad connector to start.



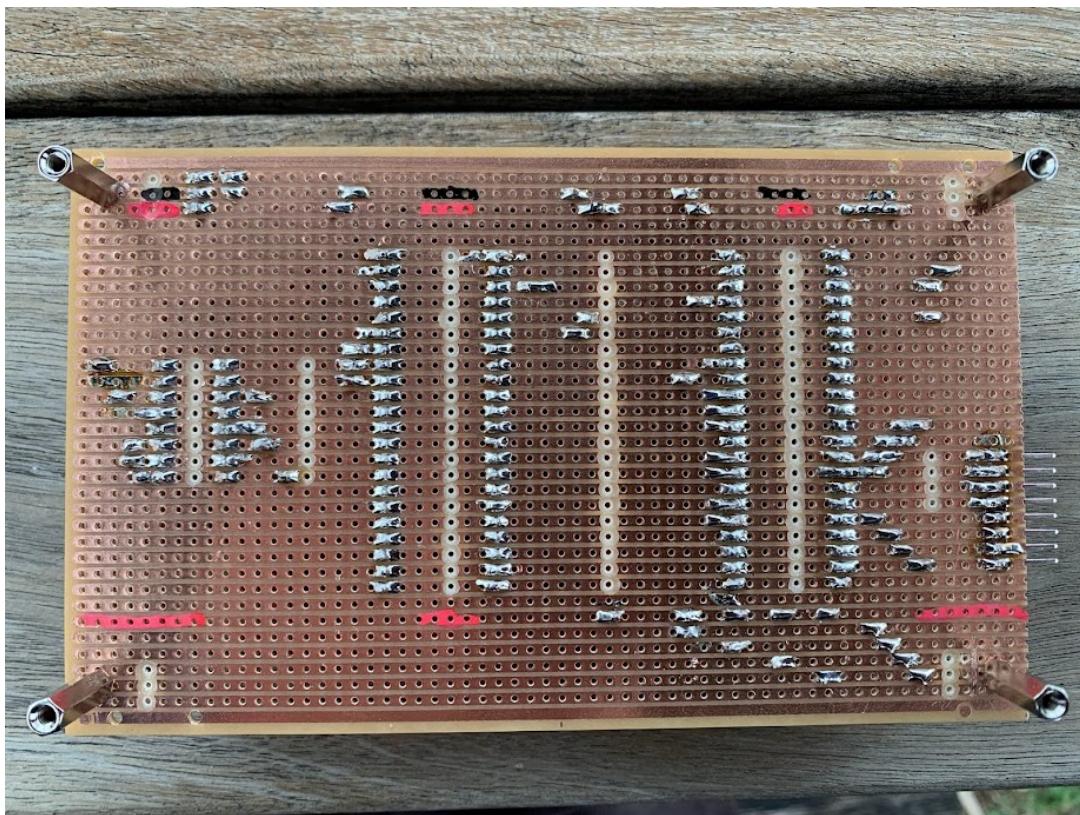
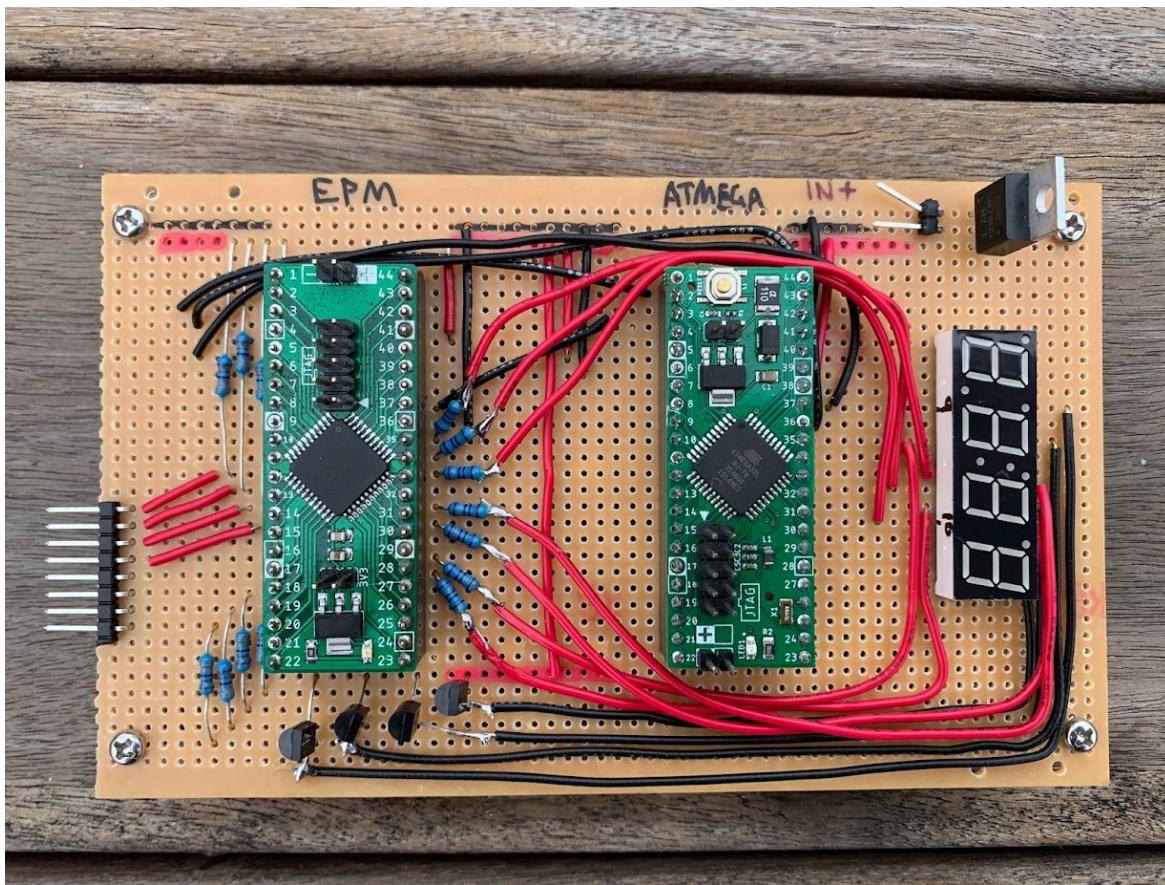
Adding resistors, wires, transistors.



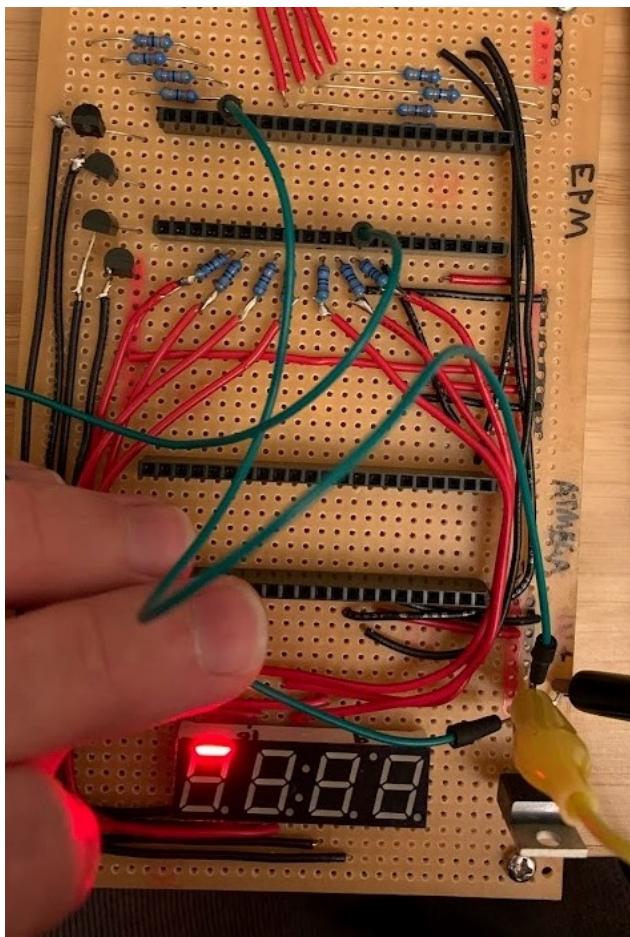
Soldering and wiring the transistors,



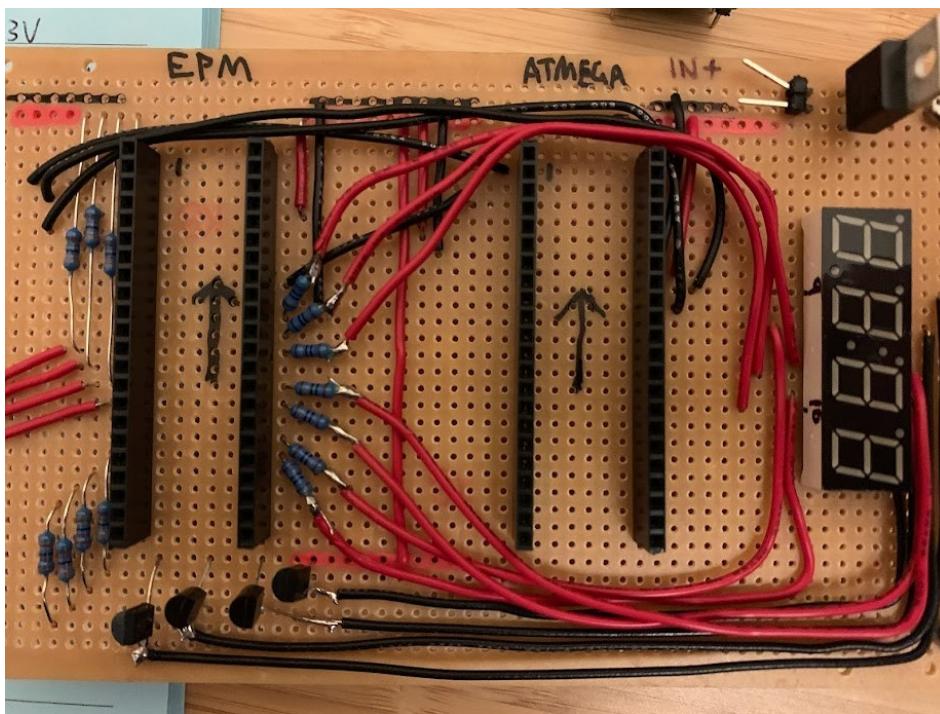
“Final” board with everything soldered.



Testing the 7-segment display by grounding the connections to each segment.

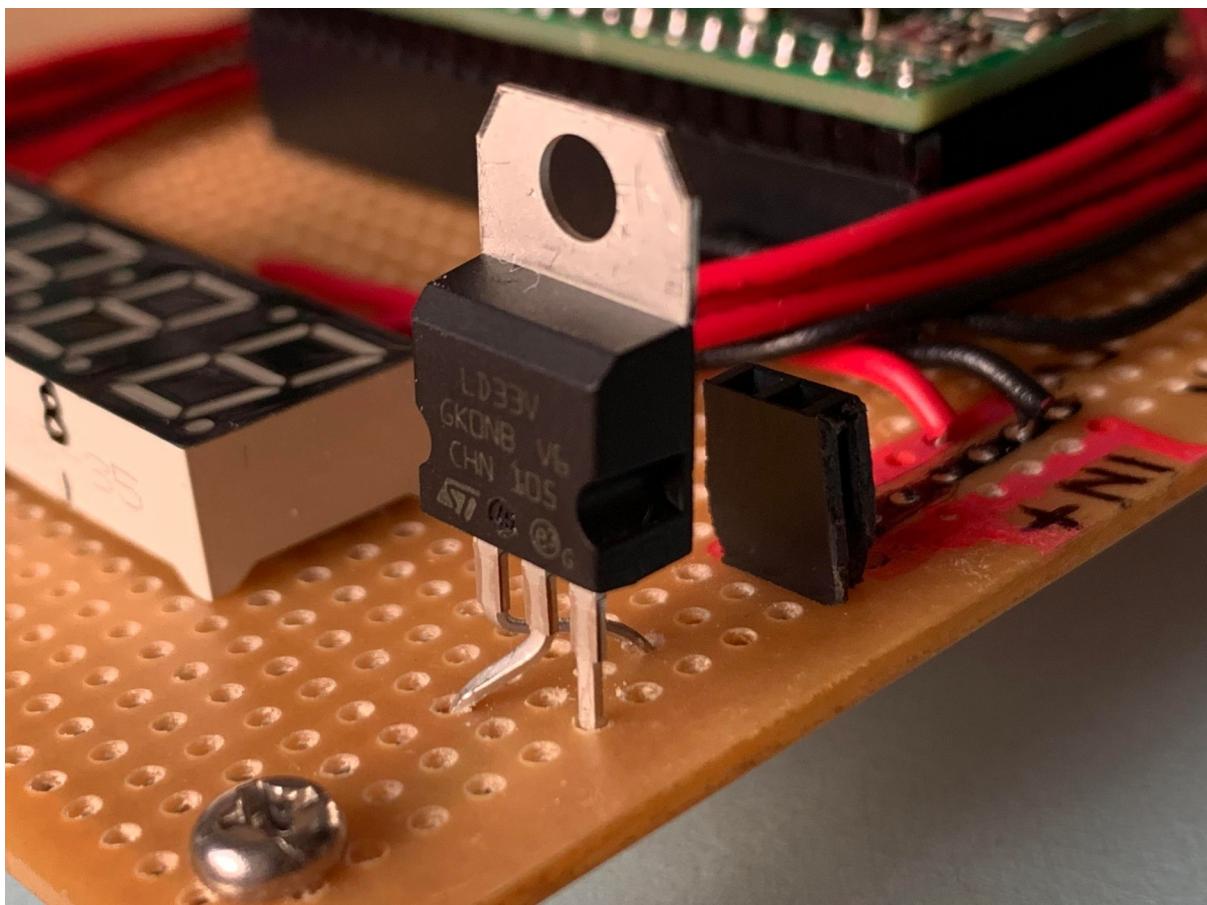


Labeled the underside so that chips can be removed and placed back in correctly to reduce the likelihood of faults later (pin1 top left).



Later the 5V regulator needed to be removed and replaced with a 3.3V regulator. The 3.3V regulator did not have the same leg assignments, but because the whole board was built I bend the legs into place rather than altering anything on the board.

Initially I also connected power to the input and ground pins with alligator clips, but I replaced these with female pin headers for tidiness.



## Programming

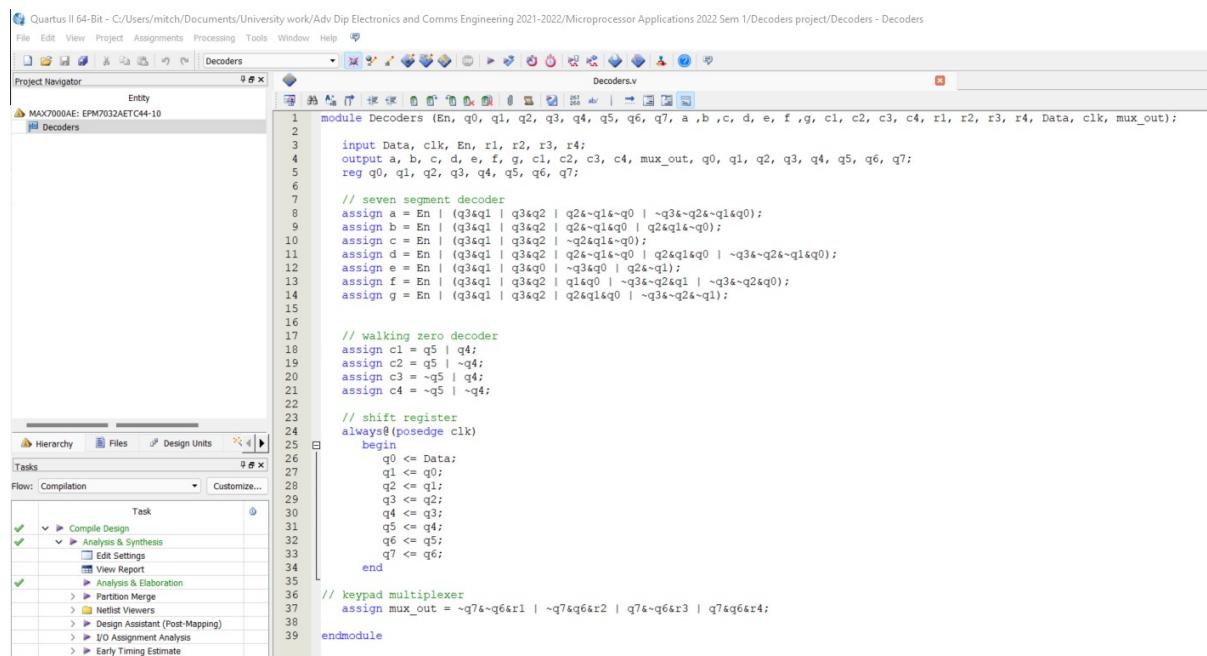
### Quartus II and EPM7032AE (PLD):

In Quartus we created different blocks of logic for the 7-segment decoder, walking zero decoder, shift register, and the keypad multiplexer.

I have attached above in the report the truth tables for some of these, and the same formulas have been written in the program in Quartus to be used in the PLD.

The shift register is self-explanatory: it's first bit is the Data in, and each successive bit will equal the previous bit upon each clock cycle.

The walking zero decoder only has 4 states, signifying which output is having voltage applied to it (and is being read).



```

Quartus II 64-Bit - C:/Users/mitch/Documents/University work/Adv Dip Electronics and Comms Engineering 2021-2022/Microporcessor Applications 2022 Sem 1/Decoders project/Decoders - Decoders
File Edit View Project Assignments Processing Tools Window Help
Decoders.v
MAX7000AE: EPM7032AETC44-10
Decoders

1 module Decoders (En, q0, q1, q2, q3, q4, q5, q6, q7, a, b, c, d, e, f, g, c1, c2, c3, c4, r1, r2, r3, r4, Data, clk, mux_out);
2
3   input Data, clk, En, r1, r2, r3, r4;
4   output a, b, c, d, e, f, g, c1, c2, c3, c4, mux_out, q0, q1, q2, q3, q4, q5, q6, q7;
5   reg q0, q1, q2, q3, q4, q5, q6, q7;
6
7   // seven segment decoder
8   assign a = En | (q3&q1 | q3&q2 | q2&~q1&~q0 | ~q3&~q2&~q1&q0);
9   assign b = En | (q3&q1 | q3&q2 | q2&~q1&q0 | q2&q1&~q0);
10  assign c = En | (q3&q1 | q3&q2 | ~q2&q1&~q0);
11  assign d = En | (q3&q1 | q3&q2 | q2&~q1&q0 | q2&q1&q0 | ~q3&~q2&~q1&q0);
12  assign e = En | (q3&q1 | q3&q0 | ~q1&q0 | q2&~q1);
13  assign f = En | (q3&q1 | q3&q2 | q1&q0 | ~q3&~q2&q1 | ~q3&~q2&~q1);
14  assign g = En | (q3&q1 | q3&q2 | q2&q1&q0 | ~q3&~q2&~q1);
15
16
17   // walking zero decoder
18   assign c1 = q5 | q4;
19   assign c2 = q5 | ~q4;
20   assign c3 = ~q5 | q4;
21   assign c4 = ~q5 | ~q4;
22
23   // shift register
24   always@(posedge clk)
25     begin
26       q0 <= Data;
27       q1 <= q0;
28       q2 <= q1;
29       q3 <= q2;
30       q4 <= q3;
31       q5 <= q4;
32       q6 <= q5;
33       q7 <= q6;
34     end
35
36   // keypad multiplexer
37   assign mux_out = ~q7&~q6&r1 | ~q7&q6&r2 | q7&~q6&r3 | q7&q6&r4;
38
39 endmodule

```

## Programming

### ATMEL Studio and ATMEGA32L (microcontroller):

This code is to read the input from the MUX out of the EPM chip, which is sent to an input on the ATMEGA, and retrieve a number from the array.

```

/*f-
 * main.c
 *
 * Created: 4/29/2022 9:21:56 AM
 * Author: mitch
 */

#include <avr/io.h>
void keypad(void);
unsigned char display[4]={1,9,6,5};
unsigned char key_data[16]={0xD,0xC,0xB,0xA,0xF,9,6,3,0,8,5,2,0xE,7,4,1}; // new array, numbers assigned to correct values
unsigned char i, key_pressed, key_held;
int main(void)
{
    unsigned char x,y;          // setting up global characters
    DDRA=0x01;
    DDRB=0xB0;
    SPCR=0x50; // set SPI control register with 01010000, turn on SPE enable, set as master
    while(1)
    {
        for(i=0;i<16;i++)
        {
            PORTA=PORTA | 0x01;
            x= i<<4 | display[i&3]; // was 1&3 // shifts and calls on array
            SPDR=x;                  // send data to left seven segment display
            while((SPSR & 0x80)==0x0);
            y=SPDR;                 // send data to digit
            PORTA=PORTA & 0xFE;      // make enable pin of PORTA = 0
            for(j=0;j<100;j++);     // delay
            keypad();
        }
    }
}

void keypad (void)
{
    if((PIN_A & 0x02)==0 && key_held==0)
    {
        key_held=1;
        key_pressed=key_data[i];
        display[0]=display[1];
        display[1]=display[2];
        display[2]=display[3];
        display[3]=key_pressed;
    }
    if((PIN_A & 0x02)==2 && (key_data[i]==key_pressed) && (key_held==1)) // if previous key is released
        key_held=0;
}

```

The array had to be set up with a customised order, because when I originally tested the code each button was calling the incorrect number.

I had to note the exact position of the number that was being called by each button, and then place the correct number at this location. This took quite a bit of time, but after debugging this the program works perfectly.

**Link to video of testing:**

<https://photos.app.goo.gl/DNU4LvqWMNEeoZxR7>