31/07/23

July 31, 2023 8:11 AM

Start 8:00

I went back to looking to the paper on the capacitive charging method. But this is really not making sense to me. They mention that their proposed method is a closed loop system meaning it should take feedback from the battery voltage to control the MOSFETs, but in their experimental test they just show a PWM signal where it cycles between state 1 and state 2. I understand that would probably work, cause you have just cycling power back and forth between cells until everything is balanced, but that is not a closed loop system, and it also does not use any of the 6 modes that they outlined in the paper.

B1 B2
B1 B3
B1 B4

B2 B1
B3
B4

B3 B1
B2
B4

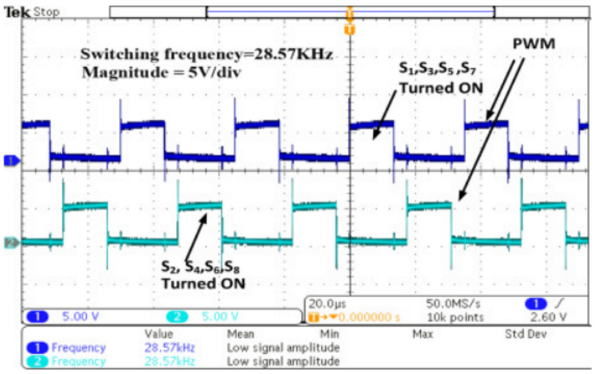
B4 B1
B2
B3

→ 12 options

$2^4 = 16$

4 pin minimum

> Fig. 12 shows the PWM pulses for the MOSFET switches at 28.57kHz switching frequency with 5V/Div. Fig. 13 shows the imbalance in the four cells voltages connected in series before cell voltage equalisation with an initial unequal voltage as B_1=3.96V, B_2=2.67V, B_3=2.75V, B_4=3.86V. In this proposed cell voltage equalisation method, the cell voltage equalization is done during the load-connected condition. So, the cell voltage balancing and discharging of cells to load happens at the same time. For accurate cell voltage balancing, voltage differences between cells should not exceed 0.1V.



Like this does not look like a closed loop system. It looks like they are just switching between state 1 and state 2. Also I am not even sure what happens in state 1 and state 2. The paper does not go into detail about these two, following the pattern from the modes described, Cell 1, 2, 3, 4 would discharge into the GND line. Which doesn't really make a lot of sense. State 2 would have cells 1, 2, 3, 4 discharge into the positive line, which again does not make much sense. This circuit is really just MOSFETS, cells and capacitors. There are definitely cells and MOSFETS here in the lab, and im sure Megan would have 6 capacitors.

https://mdpi-res.com/d_attachment/wevj/wevj-05-00385/article_deploy/wevj-05-00385.pdf

I have been reading this paper. It talks about a single switched capacitor which seems much simpler to understand. The control is a bit more complicated, however it shouldn't be too bad. It would require 1 more pin.

This paper also talks about switching and using a PWM signal. I again don't really see why you wouldn't just charge a cell until it is evened out, then disconnect it and connect another one. Reading the paper, I think it might have to do with controlling the current. I think that they are using the switching to generate a frequency to cause some impedance with the capacitor. This way they can limit the current without needing a component like a resistor which will consume power. But shouldn't the power still be consumed if there is no inductor balancing it out.

I assume that the other paper is using a PWM signal for the same reason, to control the current when balancing the cells. I still do not really understand why they just cycle between State 1 and State 2.

And switching from state 1 to state 2 makes sense cause it would drive the circuit the over direction, which would generate AC in the capacitors, I think. I am not sure if the switching frequency would cause any impedance when the capacitor is being charged the same way.

So do I need to run an enable and the PWM so that I switch the directions back and forth with the PWM, then enable the specific switches I want to connect a given path during that process?

I guess I don't actually need both, I could do that in software if that is what I need. Like only connect the PWM signal to the pins that I want. But I am not sure that I can connect multiple output pins to the same PWM signal, which I would need for this.

But if I take the time to reassign the PWM pins I may mess with the timing.

The proposed active cell balancing scheme is useful for multiple cell modules by connecting the modules in series or parallel combinations as per the voltage and current rating requirements of the battery pack. With the proposed control structure, all optimal voltage equalization paths are discovered from one cell to other cells in the battery pack with the least number of capacitors, as shown in Fig. 3 and 4. Therefore, the proposed method is robust and reliable for the unbalanced states of the battery pack. The summary of the proposed method switching sequence is given in Table 1

Table 1. Switching sequence of the proposed cell balancing strategy.

	Target cells	Status of switches
State 1	Odd number	S_1, S_3, S_5, S_7 are turned ON
	Even number	S_2, S_4, S_6, S_8 are turned OFF
State 2	Odd number	S_1, S_3, S_5, S_7 are turned OFF
	Even number	S_2, S_4, S_6, S_8 are turned ON

This is also in the paper, which is a summary of the switching sequence. I am not sure what it means my target cells. Especially because each option only covers half the switches. Like 'Odd number' doesn't fully define the state of the switches.

The single switched capacitor circuit also seems viable. But it seems like this other system is better, I just cant understand how to use it properly. I think I might need Nick to read through it and see if he is able to understand it. He definitely has a better foundation on all this stuff than I do.

and turned on or off by near-zero current switches for minimising the switching loss. In the proposed balancing circuit, the switching gate pulse is 50% of the duty cycle and the switching frequency is equal to the resonant frequency so that all MOSFETs switched are achieved soft switching. Also, this soft-switching minimised the impedance and allow flowing the maximum current and reduced the voltage balancing time. When the cell voltage variation occurred in the EESS string between two cells then associate switches of the highest energy capacitive cell are turn on in the fast phase MOSFET gate pulse and energy temporally store in single switches-capacitor and series LC tank. In the second phase MOSFET gate pulse, all associate switches of the lowest energy capacitive cell are turned on and the stored energy is transferred. This process is repeated until the voltage balancing achieves between two cells.

https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/el.2020.1417

This paper is saying that the duty cycle is because you have to charge the capacitor, and then discharge the capacitor, it is not a steady state thing that can just be left. You have to cycle to charge the capacitor and then discharge it. This also explains why in the other paper it said that it can be simplified into 2 states, charge and discharge. I thought that was referring to the battery, not the capacitors.

But that does not really add up with the modes mentioned. Cause you would think that like charging cell 1 would have a configuration in state 1 to charge the capacitors and then another configuration in state 2 to discharge the capacitors. But each mode targets different cells. I am wondering if State 1 just discharges all the cells into the capacitors, and then State 2 discharges all the capacitors into the cells.

Mode 1($T_0 - T_1$): In this mode, MOSFET switch S_1 is ON and all the remaining switches are in OFF state as shown in Fig. 3(a). During this mode, the highest voltage cell (B_1) is discharged through the capacitors to all other cells in the string. This mode 1 will continue until B_1 voltage becomes equal to the second-highest voltage cell (B_2) in the series-connected cells of the battery string.

But reading the descriptions, it does not seem like the discharging of one cell and the charging of another cell happens asynchronously. It seems like they happen at the same time. So if I assume that State 1 is charging the caps, and the three modes control which cell/cells is being used to charge the cell, then how does state 2 discharge. Cause the descriptions are the same, just will different cell numbers.

Mode 4 ($T_3 - T_4$): In this mode, MOSFET switch S_8 is ON and all the remaining switches are in OFF state as shown in Fig. 4(a). During this mode, the B_4 cell is discharged through the capacitors to the other cells. This mode 4 will continue until B_4 cell voltage is equal to B_3 voltage in a series-connected battery string.

In-state 1: The switches S_1, S_3, S_5 and S_7 are turned ON and S_2, S_4, S_6 and S_8 are turned OFF. The switching capacitors form a closed loop with high voltage rated cells and cell charging paths are constructed between cells and capacitors, as shown in Fig.3. During this state 1, the control operation is explained in three modes.

In-state 2: The switches S_1, S_3, S_5 and S_7 are turned OFF and S_2, S_4, S_6 and S_8 are turned ON. The switching capacitors form a closed loop with low voltage rated batteries (B_2, B_3, B_4) and discharging paths are constructed between capacitors to battery cell, as shown in Fig.4. During this state 2, the control operation is described in three modes.

In the description of the two states, it clearly states that State 1 is contracts paths to charge the capacitors from the cells. While state 2 constructs discharging paths to discharge the capacitors into the cells. So why do the modes under state 2 say that that cells are being discharged. The capacitors should be the ones being discharged.

It does make sense how just cycling state 1 and state 2 would work. It essentially just discharges all the cells into the capacitors, then discharges all the capacitors into the cells. I would assume that cells that are undercharged would have a larger voltage difference and therefore draw more current, so eventually you would average out the cell voltages. Which is all that I want. I would probably create a test circuit of these and then see if it works by oscillating state 1 and state 2. Generating the complimentary PWM signals is something that I have already figured out how to do, so that should be relatively easy to impliment on the PIC chip.

If I just need to oscillate between the two states, that dramatically simplifies the circuit. I will also not need the control pins, I just need two pins to send the complimentary PWM signals to the MOSFETs

PIC PWM from I2C

I modified the actuators script so now it should be set up to receive the PWM commands over the I2C bus and then set the PWM.

Leave 3:30

01/08/23

Tuesday, August 1, 2023 8:07 AM

Start 8:00AM

To do:
- Nick is going to bring in his wire wrap tool - then we are going to wire wrap all the I2C pins to see if the error is reduced
- UART receive on PIC ✓
- UART RPI
- PWM I2C

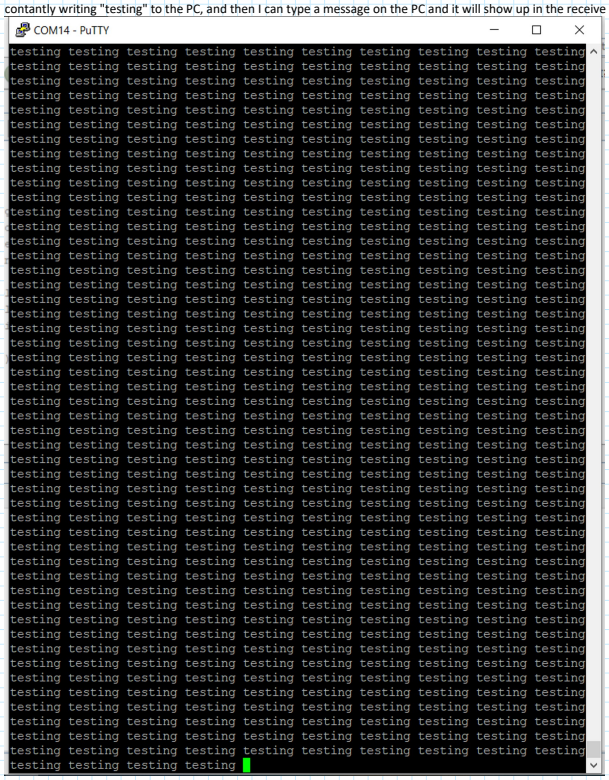
I am also going to find a new delay method for the PIC to enable interpret address - I was doing a while loop until client mode got set - but with multiple devices on the line this is not ideal cause the script will get stuck in that loop when another device is called.

It wouldn't matter for the PWM as much cause nothing happens outside the ISR, but for the ADC where readings are taken continuously this would mean nothing is read and the values are never updated.

The only other thing I could do is clean up the github and remove some files from the beginning where I didn't have a goal - like the random TLF file and the "threads"

UART PIC

I added the receive method for the UART on the PIC. It all appears to be working great. I am getting no errors this morning. And the receive is working properly. I have a script that is contantly writing "testing" to the PC, and then I can type a message on the PC and it will show up in the receive buffer variable I made.



I typed "Testing 1 2 3. This is a test." into the PuTTY terminal.

Name	Type	Address	Value	Binary
<Enter new watch>				
U1FIFO	SFR	0x2B0	0x1E	00011110
U1FIFObits; file:C:/Users/	union	0x2B0		
U1RXB	SFR	0x2A1	0x2E	00101110
U1RXPPS	SFR	0x272	0x11	00010001
receive; file:C:\Users	unsigned char[64]	0x501	"Testing 1 2 3. This is a test.\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0...	
timeout	int	0x545	0x000A	00000000 00001010

You can see it was written to the PIC.

I think that this is working good. I am seeing no errors today.

main.c

PIC18\UART.X\main.c

```c
/*
 * File:   main.c
 * Author: damie
 *
 * Created on July 17, 2023, 9:17 AM
 */


#include <xc.h>
#define _XTAL_FREQ 4000000
uint8_t receive[64];
int count = 0;

void TX(char data){
    int timeout = 0;
    while(U1FIFObits.TXBF && timeout < 1000){timeout++;}
    U1TXB = data;
}


//    U1CON0bits.BRGS = 1;    //Set Baud rate generator to normal speed with 4 clocks per bit
//    U1BRGH = 0;
//    U1BRGL = 0x19;  //Set BRG to 27 for baud of ~9600 (manually changed to 25 when debugging)

    /*Config I2C interrupts*/
    INTCON0bits.GIE = 1;    //Enable interrupts
    INTCON0bits.IPEN = 1;   //Enable Interrupt Priorities


    PIE4bits.U1IE = 1;
    PIE4bits.U1RXIE = 1;
    IPR4bits.U1RXIP = 1;
    PIR4bits.U1RXIF = 0;

    U1RXB = 0x00;

    U1CON1bits.ON = 1;  //Turn on UART
}

void loop(){
    #define length 7
```

```
void TX(char data){
    int timeout = 0;
    while(U1FIFObits.TXBF && timeout < 1000){timeout++;}
    U1TXB = data;
}

void setup(){
    OSCFREQ = 0b0010;   //Set HF clock to 4MHz
    OSCCON1bits.NOSC = 0b110;   //Set HF to Fosc

//    __CONFIG(FEXTOSC, 0b110);   //Configure the EXTOSC mode for a 500kHz-8MHz clock
//    OSCENbits.EXTOEN = 1;       //Enable the external oscillator
//    OSCCON1bits.NOSC = 0b111;   //Switch the clock to EXTOSC


    RC0PPS = 0x10;  //Set UART TX to C0
    TRISCbits.TRISC0 = 0;
    ANSELCbits.ANSELC1 = 0;
    U1RXPPS = 0b010001;  //Set UART RX to C1

    U1CON0bits.BRGS = 0;    //Set Baud rate generator to normal speed with 16 clocks per bit
    U1CON0bits.MODE = 0b0011;   //Set mode to 9bit even parity
    U1CON0bits.RXEN = 1;        //Enable receive
    U1CON0bits.TXEN = 1;        //Enable transmit

    U1CON2bits.RXPOL = 0;       //Regular polarity - 1 = High
    U1CON2bits.TXPOL = 0;       //Regular polarity - 1 = High
    U1CON2bits.STP = 0b00;      //Num stop bits

    /* BAUD = (Fosc * (1+BRGS*3))/(16*(BRG-1))
     * 9600 = (4e6 * (1+0*3))/(16*(BRG-1))
     * 9600 = (4e6)/(16*(BRG-1))
     * BRG-1 = 4e6/(16*9600)
     * BRG-1 = 26.0416666
     * BRG = ~27
    */

    U1BRGH = 0;
    U1BRGL = 0x19;  //Set BRG to 27 for baud of ~9600 (manually changed to 25 when debugging)

    U1CON1bits.ON = 1;  //Turn on UART
}

void loop(){
    #define length 7
    char msg[length] = "testing";
    for(int i = 0; i<length; i++){
        TX(msg[i]);
    }
    TX(0x09);

    __delay_ms(1);
}

void __interrupt(irq(U1RX)) ISR(void) {
    int timeout = 0;
    while(~U1FIFObits.RXBF && timeout < 500){timeout++;}
    receive[count] = U1RXB;
    count++;

    if(count > 63){
        count = 0;
    }
}


void main(void) {
    setup();

    while(1){
        loop();
    }
    return;
}
```

UART on RPI

I still can not get good readings between the RPI and the PC for UART.

I know both are on the same timing setting, but I want to see if they are the same.

Transmits from the RPI, the start bit is 8.80us long.

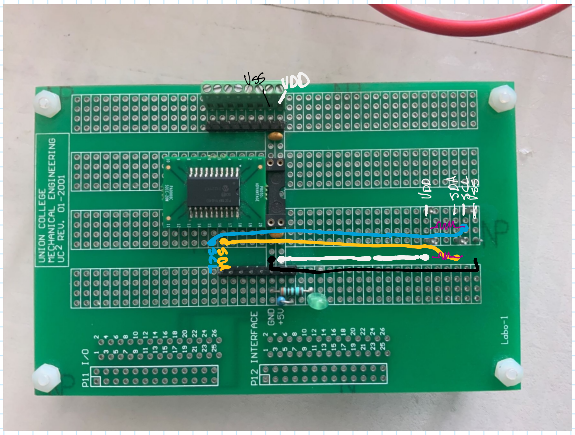
Transmits from the PC, the start bit is also 8.80us long. The timing between the two devices is perfect.

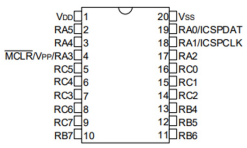
I checked the lined on each, and both devices are transmitting consistently without any issue, however neither device is reading properly.

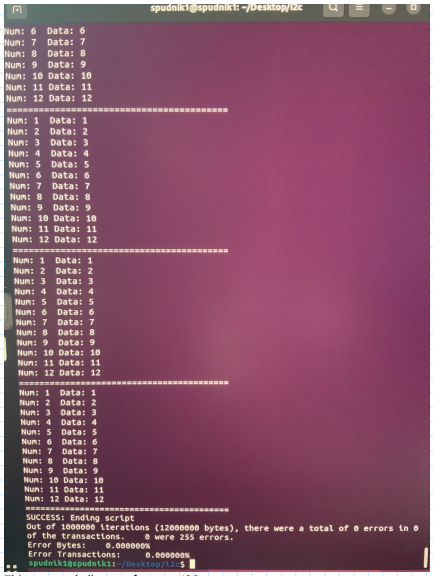
I did the loopback test on the RPI where I connect the RX and TX pins. Everything seems to work properly there.

I tried on the DTG PKIT and Putty is not reading it properly

I think the issue is the BAUD rate. The loopback test works on the DTG when I use a BAUD of 9600. But it can not handle the 115200.

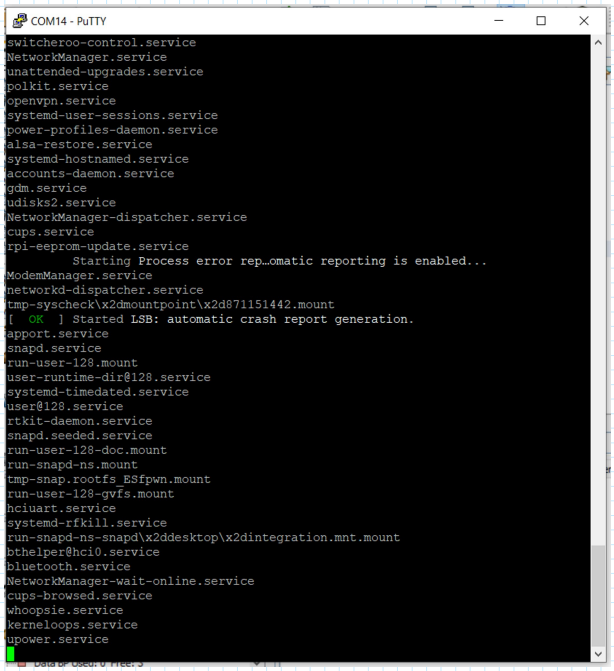




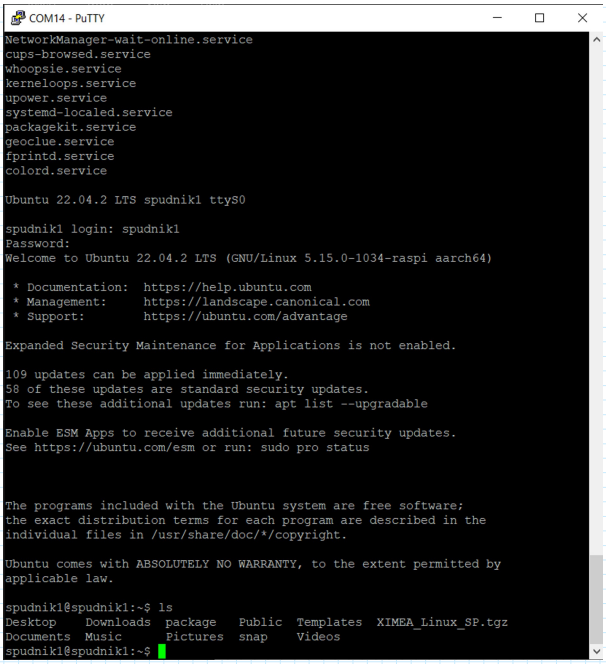


This removed all errors from my I2C
I am confident that with 12M/12M bytes transmitting correctly, I have no wiring issue

Now I should be able to develop the PWM script since I know the I2C should be capable of working properly



I have the PI sending UART to my PC



I just logged into the RPI over the UART

I should be able to run scripts

Leave - 3:30

02/08/23

August 2, 2023 8:34 AM

Start 8:30

I2C PWM

I have been working on the PIC code for the RPI to control the PWM over I2C.

I was able to get it working and it seems fairly reliable. I will need to do a reliability test on the PIC to actually measure the reliability.

I tried disabling clock stretching and it appears that it will still functino without, which is great.

I also found that it does not need the while loop to wait for client mode.

I think this is just cause it is a write command, so the Read bit is by default 0. I think that I should still include these things cause if someone ever wanted to add a read command the whole thing would break.

I modified my code to make sure that I can write to multiple PWM signals. I was able to get two working for sure. I think I will develop the library completely and give it a test run on the RPI. I will then be able to test all 5 PWM signals.

I noticed that there needs to be some delay between the commands to make sure the PIC doesnt get interrupted before setting the PWM. I found 50us was sufficient.

I will add that into the library code so that the end user does not need to worry about it.

```
107 void i2cStart(){
108     /*This function gets called by the ISR when the start condition is detected*/
109     int timeout = 0;
110     while(I2C1STAT0bits.SMA == 0 && timeout < 10){timeout++;};     //Wait until client mode is active
111     timeout = 0;
112     if(I2C1STAT0bits.R == 1){    //If a read command was given
113         I2C1CON0bits.CSTR = 0;  //Enable the clock (Stop stretching)
114         for(int i = 0; i < 24; i ++){
115             I2C1TXB = data[i];  //Load data into the transmit buffer
116             while(I2C1STAT1bits.TXBE == 0 && timeout < 100){timeout++;}; //Wait until the buffer is emptied
117             timeout = 0;
118         }
119     }
120     while(I2C1CON1bits.ACKT == 0 && timeout < 5){timeout++;};  //Wait for an ACK to make sure the byte is fully sent
121 
122     /*RESET FOR NEXT COMMUNICATION*/
123     I2C1CON0bits.EN = 0;
124     I2C1PIR = 0x00; //Clear the register that holds the flags for conditions
125     I2C1CON0bits.EN = 1;
126 }
```

I added the "timeout" condition on all the while loop delays. I am concerned that if something goes wrong the software will get stuck in these loops and then not function properly. Using the timeout will mean that it cant get perminently stuck on the loop. I set the values for the timeout condition experiemntally by reducing it until I saw errors, the increasing it a bit to give a window. It does not need to be super accurate, it is just something to prevent the PIC from getting stuck.

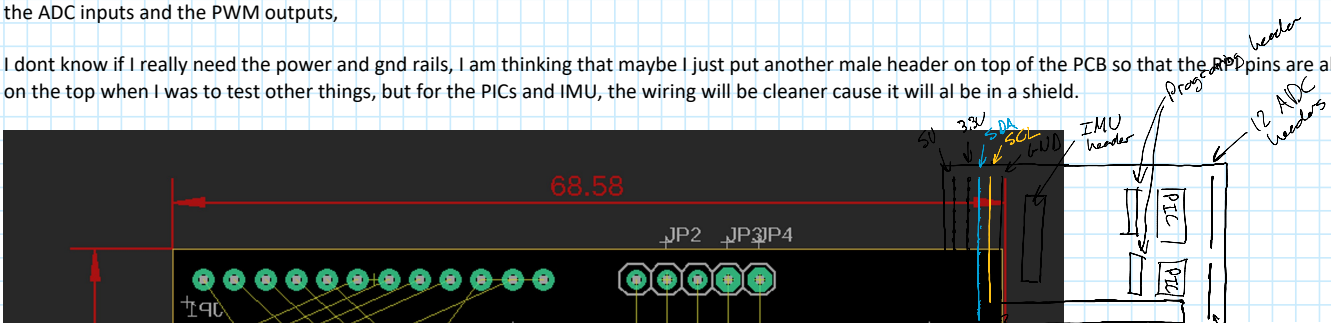
Everything seems to be working right now. I am having a weird reliability issue where there are occasional errors whenever I just program it as opposed to programming it in debug mode. It works 100% in debug mode. I am not sure what is going on there.

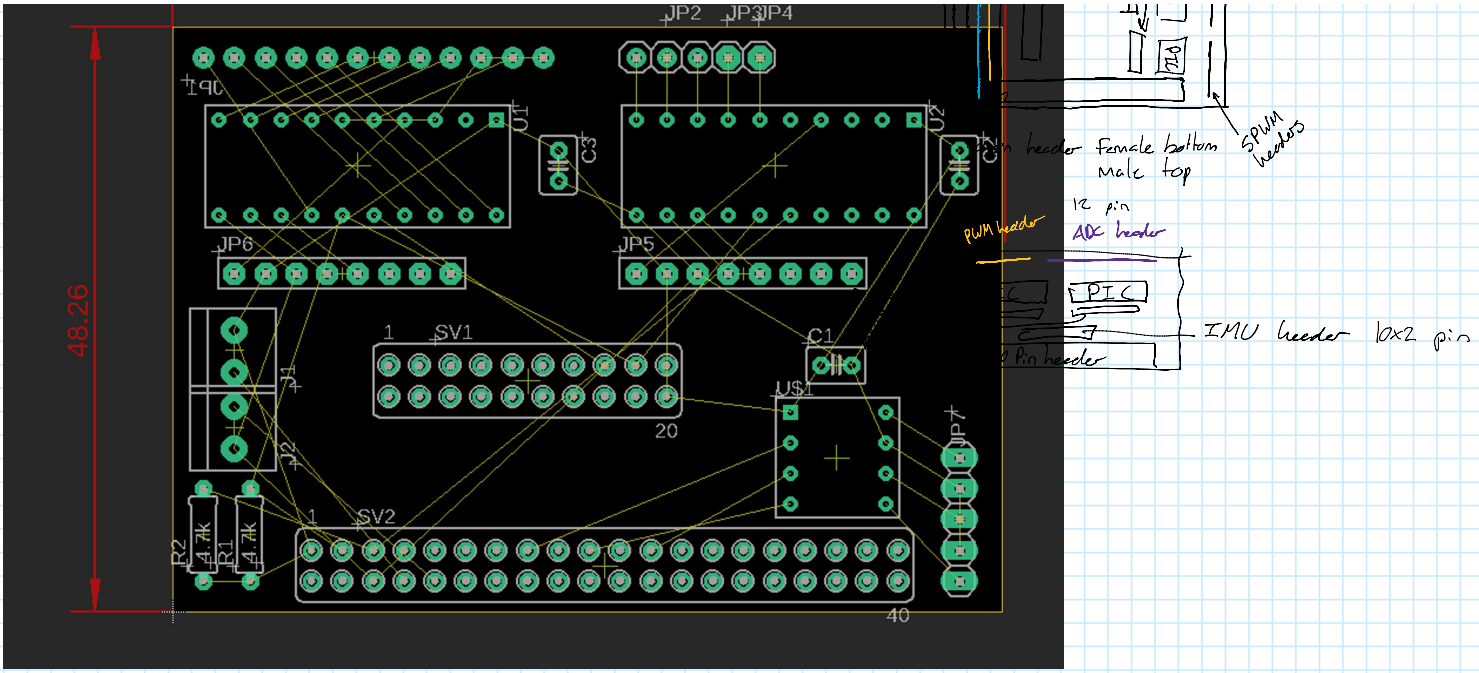
Testing

I think that it would be really cool to make a PCB that will hold the two PIC chips and the IMU, and then have a header that connects back onto the RPI.

So I could have the 40 pin female header on bottom of the board, and then the top could have headers for programming each PIC, a 5V, 3.3V and GND rail, headers for the ADC inputs and the PWM outputs,

I dont know if I really need the power and gnd rails, I am thinking that maybe I just put another male header on top of the PCB so that the RPI pins are all still accessible on the top when I was to test other things, but for the PICs and IMU, the wiring will be cleaner cause it will al be in a shield.




JP2
JP3
JP4
JP1
U1
U2
C3
C2
JP6
JP5
SV1
1
20
C1
U$1
JP7
J1
J2
R2
4.7k
R1
4.7k
SV2
1
40
48.26
Female bottom
Male top
SPWM headers
12 pin
ADC header
PWM header
PIC
IMU header 10x2 pin

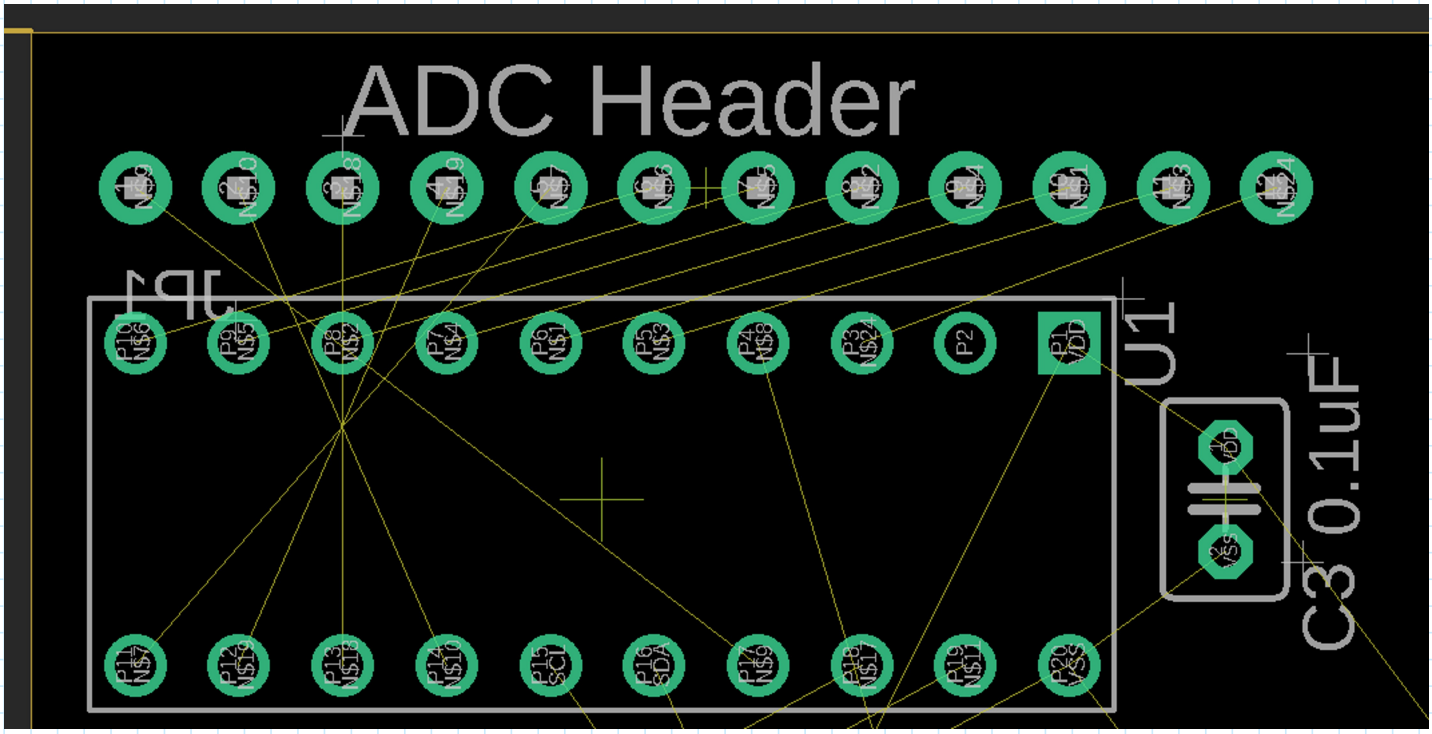
Leave 4:30

03/08/23

August 3, 2023 8:38 AM

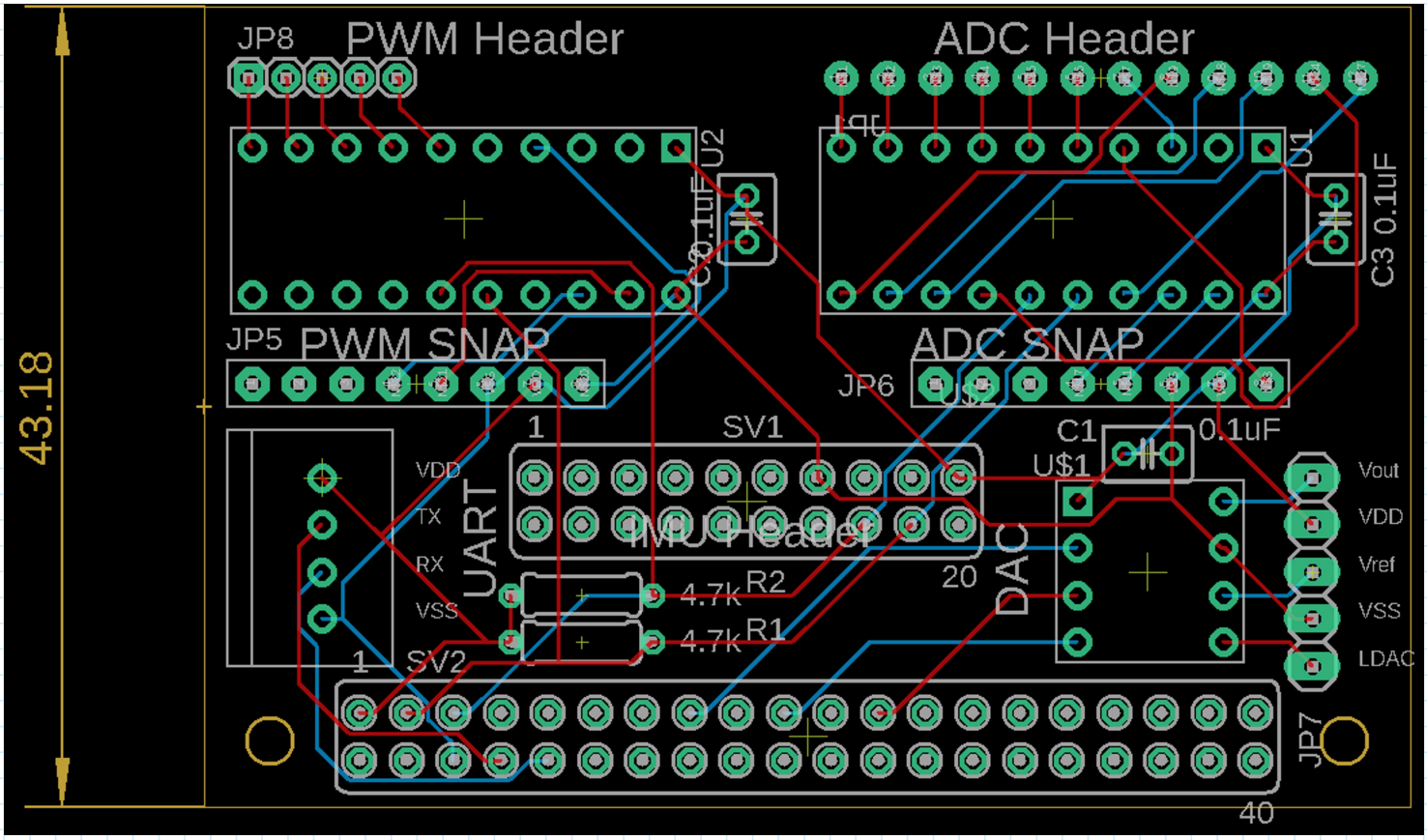
Start 8:30

Work on PCB



I cant have this arrangement of pins without needing vias. I might swap the pins to make wiring easier. I think it would be better if P11-P14 was ADC 9-12.

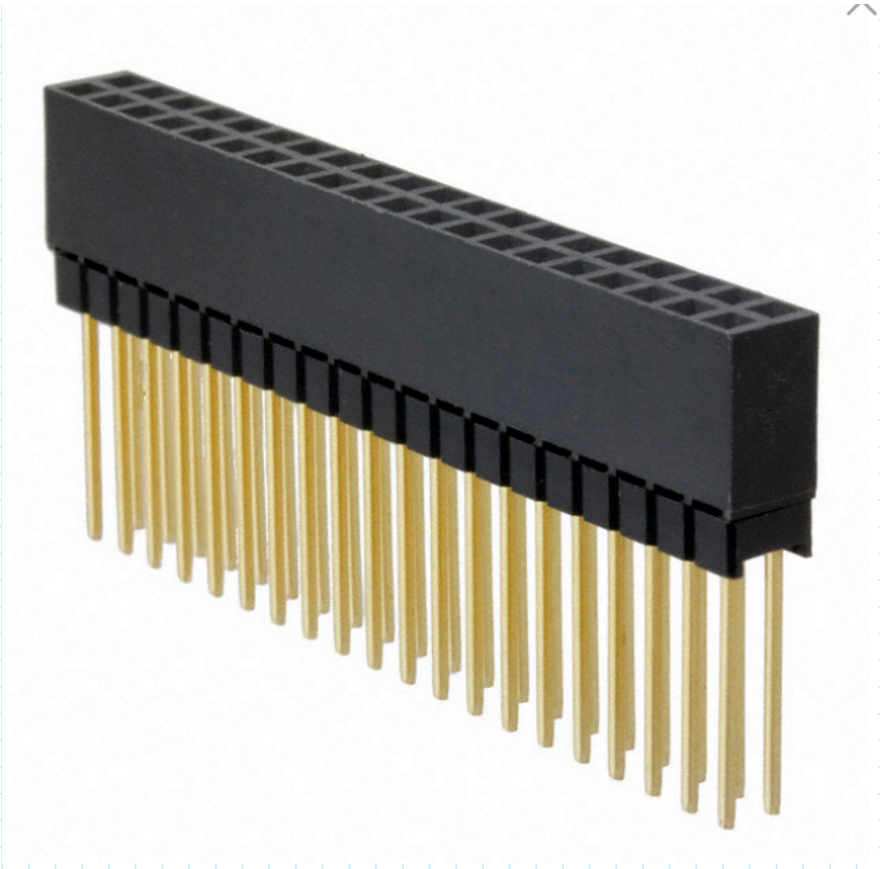
Then ADC-1 would be P-10, and I could count up on that side of the board.



This is the most recent version. I believe everything should be correct. I will have to go through and check. My only concern is the hole sizes for my custom parts.

Parts List

Part	Quantity
PIC	2
0.1uF Cap	3
8 pin male header	2
12 pin male header	1
5 pin male header	2
2x10 female header	1
2x20 female header	1
4x1 Terminal block	1
4.7k resistor	2
DAC	1



This header for the PI should make it so that all the pins can still be accessed when the Protoboard is connected. Which would be convenient as it doesn't affect the functionality of the rest of the PI

I modified the footprint for the 2x20 and 2x10 headers to increase the drill size to 0.040". They were previously 36 which would likely be too small.

I am also remembering that in the past we were having with the PIC being powered from the RPI power supply. Maybe I should use a regulator to drop the 5V from the PI to 3.3V. That is what Nick did on his arduino sheild and it worked.

Leave 3:00

Start 4:30
Added voltage regulator as external supply terminal

Stop 5:30

04/08/23

August 4, 2023 8:18 AM

Start 8:15

I think it may be convenient to add a male header for the RX and TX of the UART. I have had trouble when debugging looking at the signal on the scope cause there is no where to attach with the screw terminal. If I had room to just stick a 2 pin male header beside when it would be way easier.

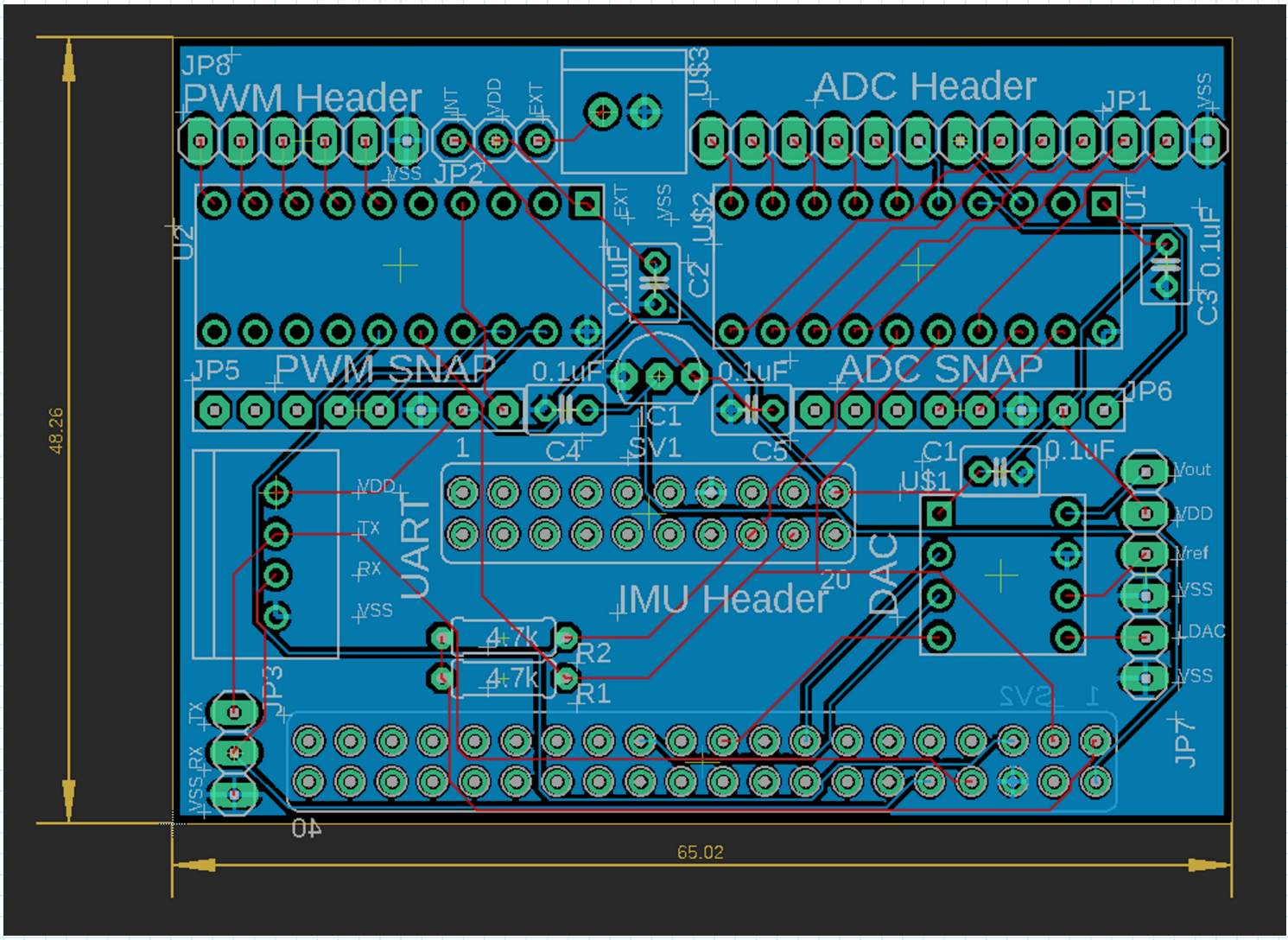
Part	Quantity
PIC	2
0.1uF Cap	5
8 pin male header	2
12 pin male header	1
5 pin male header	2
2x10 female header	1
2x20 female header	1
4x1 Terminal block	1
4.7k resistor	2
DAC	1
2 pin male header	1
3 pin male header	1
Jumper cap	3
2x1 Terminal block	1
Voltage regulator	1

Nick recommended adding a GND pin at each header, as well as making a GND copper pour layer on the bottom.

Part	Quantity	
PIC	2	X
0.1uF Cap	5	✓
8 pin male header	2	✓
13 pin male header	1	✓
6 pin male header	2	✓
2x10 female header	1	X
2x20 female header	1	X
4x1 Terminal block	1	✓
4.7k resistor	2	✓
DAC	1	X
3 pin male header	2	✓
Jumper cap	3	X
2x1 Terminal block	1	✓
Voltage regulator	1	X

✓ - should be on campus
X - Have to order

CAMOutputs



I finished the PCB layout. I think that it is ready to order at this point. I will get Nicks approval and then order it.
I will also need to make my Digikey order for all the components I need to solder onto the PCB.

Once I get everything and assemble, I should be able to run tests on the PWM and ADC scripts, ensure that there is no issues with multiple PIC's on the I2C bus, develop the SPI code for the RPI,

https://www.digikey.ca/en/mylists/list/N5XICY3AZQ
Here is my digikey list with all the components I should need.

Leave 3:00