24/07/23

July 24, 2023 8:13 AM

Start 8:00am

Tasks:
- Work on I2C reliability
  - Work on correcting timing in original
  - Work on DMA method
- Fix the Accel Offset function in my IMU library
- Work on UART
  - Look into reliability (not sure how much can be done)
  - Get Receive working on the PIC
- Program PIC to include I2C and PWM to have the RPI set the PWM on the PIC
- Program PIC to include I2C and ADC to have RPI read analog values

I believe more reliable I2C is probably the biggest concern right now, and should realistically be done before programming the PIC to do the PWM and ADC with I2C so I don't have to redo anything.

UART is not really a high priority right now as there is no actual use for it in mind as far as I am concerned.

The IMU library would also be good to fix just so that it can be completed and not touched again.

I2C timing options:
- Wait for Clock to be stretched by hardware
  - With

Bit 4 – CSTR Client Clock Stretching[3]

Value	Condition	Description
1		Clock is held low (clock stretching)
0		Enable clocking, SCL control is released
	SMA = 1 and RXBF = 1[6]:	Set by hardware on 7th falling SCL edge User must read I2CxRXB and clear CSTR to release SCL
	SMA = 1 and TXBE = 1 and I2CxCNT != 0:	Set by hardware on 8th falling SCL edge User must write to I2CxTXB and clear CSTR to release SCL
	when ADRIE = 1[4]:	Set by hardware on 8th falling edge of matching received address User must clear CSTR to release SCL
	SMA = 1 and WRIE = 1:	Set by hardware on 8th falling SCL edge of received data byte User must clear CSTR to release SCL
	SMA = 1 and ACKTIE = 1:	Set by hardware on 9th falling SCL edge User must clear CSTR to release SCL

```
I2C1PIEbits.CNT1IE = 1;     //Enable interrupt when byte count reaches 0
I2C1PIEbits.ACKTIE = 1;     //Enable interrupt for acknowledge clock stretching
I2C1PIEbits.WRIE = 1;       //Enable interrupt for receiving data to process and determine ACK or NACK
I2C1PIEbits.ADRIE = 0;      //Enable interrupt for processing address
I2C1PIEbits.PCIE = 0;       //Disable interrupt when stop condition detected
I2C1PIEbits.RSCIE = 0;      //Disable interrupt when restart condition detected
I2C1PIEbits.SCIE = 0;       //Disable interrupt when start condition detected
```

I had both ACKTIE and WRIE enabled which means the hardware is stretching the clock before and after the ACK bit. My software was only accounting for the clock to be stretched once per byte. I think that the ACKTIE is better cause I don't really need time to determine the ACK bit. So maybe if I disable the WRIE I can get better reliability using the hardware to control timing.

```
51      I2C1PIEbits.CNT1IE = 0;     //Enable interrupt when byte count reaches 0
52      I2C1PIEbits.ACKTIE = 0;     //Enable interrupt for acknowledge clock stretching
53      I2C1PIEbits.WRIE = 1;       //Enable interrupt for receiving data to process and determine ACK or NACK
54      I2C1PIEbits.ADRIE = 1;      //Enable interrupt for processing address
55      I2C1PIEbits.PCIE = 0;       //Disable interrupt when stop condition detected
56      I2C1PIEbits.RSCIE = 0;      //Disable interrupt when restart condition detected
57      I2C1PIEbits.SCIE = 0;       //Disable interrupt when start condition detected
```

I changed it to this. I ended up using the WRIE and not the ACKTIE because I don't want a double clock stretch after the matching address. I also enabled ADRIE. I was confused and thought that ACKTIE was doing what ADRIE tie does.

I also changed so that there are no delays and the it just waits until the TXB is empty.

With this, I am getting pretty reliable transfers on the first transmission (judging by eye) but something isnt resetting properly to allow for more transactions to occur.

Name	Type	Address	Value	Binary
<Enter new watch>				
I2C1ADB0	SFR	0x28E	0x00	00000000
I2C1ADB1	SFR	0x28F	0x00	00000000
I2C1ADR0	SFR	0x290	0xFF	11111111
I2C1ADR1	SFR	0x291	0xFE	11111110
I2C1ADR2	SFR	0x292	0xFF	11111111
I2C1ADR3	SFR	0x293	0xFE	11111110
I2C1BAUD	SFR	0x29D	0x04	00000100
I2C1BTO	SFR	0x29C	0xA3	10100011
I2C1BTOC	SFR	0x29F	0x03	00000011
I2C1CLK	SFR	0x29E	0x03	00000011
I2C1CNTH	SFR	0x28D	0x00	00000000
I2C1CNTL	SFR	0x28C	0x04	00000100
I2C1CON0	SFR	0x294	0x80	10000000
I2C1CON1	SFR	0x295	0x00	00000000
I2C1CON2	SFR	0x296	0x40	01000000
I2C1ERR	SFR	0x297	0x00	00000000
I2C1PIE	SFR	0x29B	0x1B	00011011
I2C1PIR	SFR	0x29A	0x04	00000100
I2C1RXB	SFR	0x28A	0x00	00000000
I2C1SCLPPS	SFR	0x271	0x11	00010001
I2C1SDAPPS	SFR	0x270	0x10	00010000
I2C1STAT0	SFR	0x298	0x80	10000000
I2C1STAT1	SFR	0x299	0x20	00100000
I2C1TXB	SFR	0x28B	0x00	00000000

This is before the first transactions -- which works

Name	Type	Address	Value	Binary
<Enter new watch>				
I2C1ADB0	SFR (Type of variable)	0x28E	0xFF	11111111
I2C1ADB1	SFR	0x28F	0x00	00000000
I2C1ADR0	SFR	0x290	0xFF	11111111
I2C1ADR1	SFR	0x291	0xFE	11111110
I2C1ADR2	SFR	0x292	0xFF	11111111
I2C1ADR3	SFR	0x293	0xFE	11111110
I2C1BAUD	SFR	0x29D	0x04	00000100
I2C1BTO	SFR	0x29C	0xA3	10100011
I2C1BTOC	SFR	0x29F	0x03	00000011
I2C1CLK	SFR	0x29E	0x03	00000011
I2C1CNTH	SFR	0x28D	0x00	00000000
I2C1CNTL	SFR	0x28C	0x04	00000100
I2C1CON0	SFR	0x294	0x80	10000000
I2C1CON1	SFR	0x295	0x00	00000000
I2C1CON2	SFR	0x296	0x40	01000000
I2C1ERR	SFR	0x297	0x40	01000000
I2C1PIE	SFR	0x29B	0x1B	00011011
I2C1PIR	SFR	0x29A	0x05	00000101
I2C1RXB	SFR	0x28A	0x00	00000000
I2C1SCLPPS	SFR	0x271	0x11	00010001
I2C1SDAPPS	SFR	0x270	0x10	00010000
I2C1STAT0	SFR	0x298	0x98	10011000
I2C1STAT0bits; file:C:/Program Files/Microchip/MPL	union	0x298		
I2C1STAT1	SFR	0x299	0x20	00100000
I2C1TXB	SFR	0x28B	0x00	00000000

This is after -- does not work again

stat0 and PIR are different

I tried disabling the I2C module after completing. This have me 100% reliability on bytes 0-11, and 0% reliability on byte 12. This does not really make sense, cause my for loop is supposed to wait until the TXB is empty. So the last byte should be fully transferred out before restarting the bus

```
103     while(I2C1STAT0bits.SMA == 0){};
104     if(I2C1STAT0bits.R == 1){
105         for(int i = 1; i < 13; i ++){
106             I2C1TXB = i;
107             I2C1CON0bits.CSTR = 0;
108             while(I2C1STAT1bits.TXBE == 0);
109         }
110     }
```

Except I am holding the clock before the ACK bit, which might cause issues cause I never let the clock go again. If I add a clock release command after that might fix my issue.

That did not solve my problem. But I am consistently getting 1 error per byte and it appears to be the last byte every time. Although maybe I need some time for the ACK to go through, I only enabled the clock, then the next command was to turn off the module. I believe one clock cycle is ~10us, so ill add a 15us delay.

This increased it a bit but not completely. Using the ACKTIE would also solve this issues, I might just have to deal with a double trigger at the beginning.
Or I could just not use the ADRIE. If there is going to be an hold on the ACK bit at the end of the address, then why do I need it.

I tried both ways, using ACTIE, using ACKTIE with ADRIE disabled, and nothing is working. I need to check the scope cause it is clearly getting stuck right at the beginnging cause nothing isgoing through

There is something weird going on… The SCL line is starting to cycle before any change occurs on the SDA line. A start condition is that the SDA line is pulled low while SCL is high, meaning the SCL line is cycling before a start condition occurs. This is definitely wrong.
I disabled clock stretching to see if that would fix this issue (I know it would break the script as a whole but thpught maybe I could see the start condition) but it did not fix it.

I am going to change it back to what was working semi-reliably and see if the start is occuring.
The start condition is occuring in this configuration…

It seems to be when ACKTIE is enabled. But that should only be affecting data on the 9th cycle, there should be no effect on the start condition.

I am also noticing that the error on the last byte is either 63, 31, or 15. These are all numbers that would have 1's in the LSbs of the byte. As a 1 is idle on the I2C bus, this would occur if the PIC was disabled too quickly.

Ideally I would wait to disable and reset the module until a stop condition is detected, however there is the bug where the stop condition flag is always set.
I can also use ACKT which gets set when the device is in an ACK sequence. Also, the host should be the one doing the ACK bit and the and the stop condition, so really the client should not need to be active once it empties the buffer… Technically though I believe the TXB that software interfaces with is not the last buffer before being on the bus. So maybe it needs some more time for the data to actually be sent onto the bus.

This seems to have fixed it! I did three tests of 100 transactions and had 0 byte errors. Now it is time to do more and see what it gets.

I did 10k and got a 0.08% error rate. This is not much better than when I was using the timing. But there are also possibly problems with the wiring and such cause it is just jumper cables. I am much more confident in the reliablity now that everything is event driven as opposed to delay based.

Some of the errors are that the device failed to contact the slave -- These errors can be avoided. If there is a situation where the device isnt contacted, this can be detected and the read command can be called again until it is able to make contact. This should reduce the number of errors and only show the errors that are occuring on the bus

I found that I don't really need to use clock stretching on the transaction -- I just load the data into the buffer when it is emptied the error rate is the same as if I use the clock stretching after the data.

I have it down to a 0.075% error rate. Looking at the errors, it is clear that the PIC will just randomly stop working in the middle of a transaction. Cause the data real will be some number that is one less than a power of two, which means the PIC stopped controlling the data line and ones were read. Ever error is either 255, 127, 63, 31, 15, 7, 3, or 1.

Some of the errors are 11 when it was supposed to send 10, but this is the same as the 1 error, as it is receiving 10 and then the PIC stops working on the last bit, causing the 1 to get set and then followed by 255's

This makes me think that my timing is perfect, and that the issue has something to do with the PIC getting caught in some other routine and not controlling the SDA line anymore.

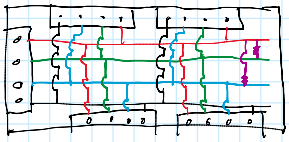
PIE0	SFR	0x4A8	0x00	00000000
PIE1	SFR	0x4A9	0x00	00000000
PIE10	SFR	0x4B2	0x00	00000000
PIE2	SFR	0x4AA	0x00	00000000
PIE3	SFR	0x4AB	0x00	00000000
PIE4	SFR	0x4AC	0x00	00000000
PIE5	SFR	0x4AD	0x00	00000000
PIE6	SFR	0x4AE	0x00	00000000
PIE7	SFR	0x4AF	0x04	00000100
PIE8	SFR	0x4B0	0x00	00000000
PIE9	SFR	0x4B1	0x00	00000000

Looking at the interrupt enable registers, I can see that only one interrupt is enabled. This is the I2C interrupt.

```
51      I2C1PIEbits.CNT1IE = 0;      //Enable interrupt when byte count reaches 0
52      I2C1PIEbits.ACKTIE = 0;      //Enable interrupt for acknowledge clock stretching
53      I2C1PIEbits.WRIE = 0;        //Enable interrupt for receiving data to process and determine ACK or NACK
54      I2C1PIEbits.ADRIE = 1;       //Enable interrupt for processing address
55      I2C1PIEbits.PCIE = 0;        //Disable interrupt when stop condition detected
56      I2C1PIEbits.RSCIE = 0;       //Disable interrupt when restart condition detected
57      I2C1PIEbits.SCIE = 1;        //Enable interrupt when start condition detected
58
```

And the only triggers I have enabled are the start condition and the address processing. Neither of these would be occuring in the middle of a transaction. I really think that this is a wiring issue.

I am pretty convinced this is an issue with wiring. The solution for this would be to solder everything. However I don't want to solder onto this RPI directly. Additionally I don't really want to have to solder onto the PIC chip. I would be able to remove the solderless breadboard however. I would just need a board that has power, Gnd, SDA and SCL terminals that connect to bus wires. Then I can solder the pullup resistors. I would still run female-male jumpers from the RPI and PIC. But it could still be an improvement.



Something simple like this is what I had in mind. I don't need this many terminals, I really only need 2. but I could make it with more space.

If I get a better connection and I can remove these errors, I should be able to get ~100% reliability. Then I could start getting the Write command to the same level of reliability once I eliminate the hardware issues.

IMU

I went back to checking for the bug with the IMU library where the accel offset was not working.

I found that I changed to the wrong bank in the function, it was supposed to be bank 1 but I had bank 2.

It is still behaving weird though. When I set the offset to 0, the value of the accelerometer output is changing. For example the z is going from 16.8k to 23.8k

Changing to 0x00FF makes it go to 27.7k
Once I get to 0x0FFF, all the readings are maxed at 32.7k

It is also not consistent in changing negatively or positively. When all are set to 0x00FF, the X and Y get more negative and the Z gets less negative. And this is not like a multiplier. Cause X transitions from negative to positive while Y and Z remain negative

	X_OUT	Y_OUT	Z_OUT
NONE	540	-660	16700
0x0000	-21440	-18750	23670
0x0008	-21380	-18700	23750
0x000F	-22900	-20200	22225
0x008F	-21850	-19150	23250
0x00FF	-20950	-18250	24200
0x08FF	-4550	-1850	32767
0x0FFF	9770	12500	32767
0x8FFF	-32768	-32768	-32768
0x1FFF	32676	32676	32676

I don't see any pattern here. Sometimes incrementing makes it largers, sometimes incrementing makes it smaller. It can flip from positive to negative. 0x8FFF and 0x1FFF both have a 0 in the MSb so it shouldn't be a signed variable type issue

The datasheet really does not give any indication of what the value in the offset register actually relates to.

That was with the offset shift removed. The LSb of the low register is reserved. Let me try shifting over by one to see if that helps

	X_OUT	Y_OUT	Z_OUT
NONE	540	-660	16700
0x0000	-21400	-18750	23675
0x0008	-21300	-18600	23850
0x000F	-21250	-18500	23900
0x008F	-19140	-16450	25990
0x00FF	-17325	-14650	27790
0x08FF	15400	18100	32767
0x0FFF	32767	32767	32767
0x8FFF	32767	32767	32767
0x1FFF	32767	32767	32767

This set has much more of a pattern.

There is a steady direction at least. I wonder if the offset is +/- and it just centered as opposed to signed. If that is the case, 1 should be about the middle. It is 15bits so the middle value would be bit 8 being high I believe, but it gets shifted so it would actually be bit 9.

So 0x0100 would be the middle. Does this give me the same readout as not setting it at all?

	X_OUT	Y_OUT	Z_OUT
0x0100	-17340	-14650	27780

Nope. When does X and Y transition to positive? 0x04FF is close to the transition.
I am confused why Z is always increasing while the other two go from a negative to a positive value.

Maybe I can read the offset register before ever setting it and see what it sends back. Realistically if I then set that value to the register I should get the default readings again. Maybe that will give me some clarity as to what is happening.

25/07/23

Tuesday, July 25, 2023 8:11 AM

Start 8:00AM

https://www.digikey.ca/en/products/detail/on-shore-technology-inc/OSTVN04A150/1588864
https://www.digikey.ca/en/products/detail/sparkfun-electronics/PRT-12070/5230951

Something like that should work. The rails could be used for +5V, GND, SDA, SCL. And then the screw terminals could along rows rows b and j facing the center. Then the close rails would be wired on the top and the ones that have to cross could go on the bottom of the board.

https://www.digikey.ca/en/products/detail/chip-quik-inc/SBB206/7035056
This one is the same just shorter.

The screw terminals and board would really be all that I need. The only other thing is wires and two resistors, but I should be able to find those here.

I really want to get this reliability issue sorted out. Because the I2C is the foundation for a lot of other steps. Like once I have that working good, I can make the complete PIC program and then make the libraries to interface it with the PI. At which point I can cross a big portion of the Peripherals off the list. I also do not want to be trying to develop the PIC code for receiving commands over the I2C until I determine whether the wiring is causing errors. Cause I don't see a point in debugging issues not knowing whether it is a software issue or a hardware issue.

I2C

I was thinking about the I2C again. I thought that my theory from yesterday about the client stopping controlling the bus had a problem. That there must still be ACK being generated, and if the client was not controlling the SDA line, it could not generate the ACK. But since it is a read command where the client is sending the data, it is the Host that is generating the ACK bit. Meaning that if the device just stopped working, there is no way to tell from the I2C transaction. Transmitting 255 is the same as the client not sending anything cause the bus gets held high. The stop condition is also generated by the host so there is really no difference.

If I can get this method working, I may hold off on the development of the DMA software. This software would definitely be more efficient. It is much less dynamic but considering each of the PIC only need to do one specific job its better to just have it programmed to do that job and keep it simple.

UART

I installed the raspi-config using the sudo agt-get install command and used the UI to enable the serial communication.

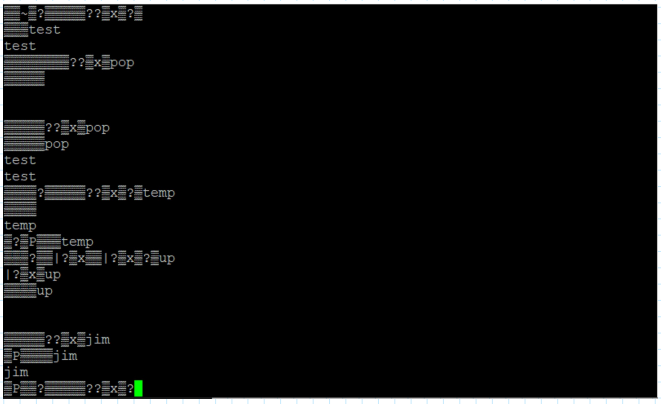
I was having trouble finding the cmdline.txt and config.txt files. In Raspbian they are under /boot but they were not this time. I found then under /boot/firmware

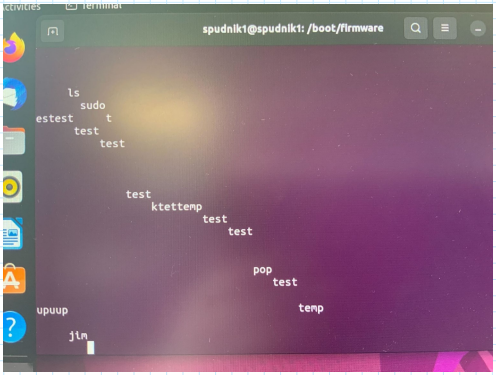
I checked and the BAUD is set to 115200. I used some tool called minicom and I was able to open a terminal similar to putty on the PI. I opened putty on my PC using the 115200 BAUD, and no parity or flow control

I am able to write things on my PC and have them show up in the terminal on the PI, however it is very inconsistent. Some messages will come through in whole, some not at all.

I think there may be a formatting issue between linux and windows, where the commands arent the same.

Also the information on Putty is jumbled characters, but the message is clear on the PI. Everything is also getting tabbed and I am not sure why.





Overall Tasks

- Camera
  - Pick out camera
  - Get camera
  - Install software to interface with camera
  - Test camera using the UI software
  - Develop C++ software to interface with camera from a script
- Magnetorquer & Reaction wheel
  - Develop PIC program to control PWM signal
  - Develop PIC program to control PWM over I2C
  - Develop C++ software to send I2C commands to control PWM to the PIC
  - Test complete system
- Sun Sensor
  - Develop PIC program to read 12 analog values
  - Develop PIC program to send readings over I2C to PI
  - Develop C++ program to read the readings from the I2C bus
  - Test complete system
- UHF
  - Determine how to use the UART on the RPI
  - Develop C++ program to read the UART Commands
  - Look into shelling into the RPI using the UART
- S-Band
  - Develop C++ software on the RPI to send data over SPI to the S-band transmitter
- IMU
  - Develop C++ software to interface with the IMU over the I2C bus

I am not sure why it is so inconsistent. Like sometimes ever message goes through and sometimes it will miss like 4 in a row I am also not sure why

I noticed the Minicom tab says "Options: 8N1"

8-N-1

3 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

See also: Serial port § Conventional notation

8-N-1 is a common shorthand notation for a serial port parameter setting or configuration in asynchronous mode, in which there is one start bit, eight (**8**) data bits, no (**N**) parity bit, and one (**1**) stop bit.[1] As such, 8-N-1 is the most common configuration for PC serial communications today.

The abbreviation is usually given together with the line speed in bits per second, as in **9600–8-N-1**. The speed includes bits for framing (stop bits, parity, etc.) and the effective data rate is lower than the bit transmission rate. For 8-N-1 encoding, only 80% of the bits are available for data (for every eight bits of data, ten bits are sent over the serial link — one start bit, the eight data bits, and the one stop bit).

Looks like that is telling me it is 8 bit, no parity, 1 stop bit. Which is the configuration that I have. It also says the BAUD is 115200 so that should be right

The terminal on the PI also says "VT102"
In Putty, under keyboard options there is an option for "VT100+", so I turned that one on. Not sure what it means.

I tried using another method to open the Serial port. I used screen to open the serial port and it gives basically yhe exact same behavior. I am thinking that there is something going on where windows and linux are using different ASCII characters for special functions like tab or other things. And it is causing there to be communicaiton errors. But I don't really have much evidence for that, I just don't know what else it would be when I know the BAUD and everything is right.
I am not sure how I would go about fixing this issue.

People on forums are talking about doing a "loopback test" where I connect the UART pins on the PI to the USB port using the adapter, and write from the USB to the UART

https://www.jeffgeerling.com/blog/2021/attaching-raspberry-pis-serial-console-uart-debugging

This person used a Mac, and it was super simple. I'm wondering if this would work better on a Unix machine instead on Windows
I think there is a PC in the lab that has Linux installed

I tried connecting the USB to the RPI and then writing a command to the USB port, while having another terminal reading the Serial pins. I did not get any output at all. I don't know why this is being so difficult.

Sun Sensor PIC

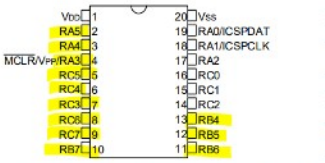
I combined my I2C script and the ADC script to make a script for the Sun sensor PIC. I now have a script on the PI that will read the 12 readings (24 bytes), and convert it to a value. I jumpered each pin from VDD to the ADC pin. My script would consistently read 4095 on the pin that was jumpered. It is a 12bit ADC which has 4096 discrete values. So 4095 is max, which makes sense.

I checked all of the pins and found that all appear to be working except for pins 1 and 3. These should be mapped to RA5 and RA3

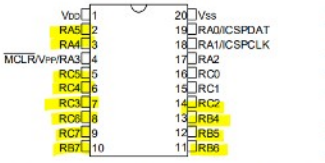
Looking at the datasheet, these pins have other peripherals they connect to.

RA3	4	—	—	—	—	—	—	—	—	—	—	—	—	IOCA3	—	MCLR VPP
RA4	3	ANA4	—	—	—	T1G(1)	—	—	CLCIN3(1)	—	—	RX3(1)	—	IOCA4	INT1(1)	CLKOUT SOSCO OSC2
RA5	2	ANA5	—	—	—	T1CKI(1) T2IN(1) SMT1WIN(1)	PWM1ERS(1)	—	—	—	—	CTS3(1)	—	IOCA5	INT2(1)	CLKIN SOSCI OSC1

RA3 does not show it can be used for analog...



These are the pins that I was using. I could try just switching to RA2 and RC2 as those are unused. I think RA3 will have to be switched as it is not showing a ANA number for the ADC. Ill try skipping RA3 and using these pins



Moving the pins worked so now only RA5 is not working. It should be noted that RA5 not only doesn't read, it also adds noise into all the other readings which is strange. Clearly that pin is doing something else

CLKIN
SOSCI
OSC1

I would guess that it is one of these ones cause none of the other peripheral modules are enabled.

12.1.2 Internal Clock Sources

The internal oscillator block contains two independent oscillators that can produce two internal system clock sources:

- High-Frequency Internal Oscillator (HFINTOSC)
- Low-Frequency Internal Oscillator (LFINTOSC)

Internal oscillator selection is performed one of two ways:

1. Program the RSTOSC Configuration bits to select one of the INTOSC sources which will be used upon a device Reset.
2. Write the New Oscillator Source Request (NOSC) bits to select an internal oscillator during run time.

In INTOSC mode, the OSC1/CLKIN and OSC2/CLKOUT pins are available for use as a general purpose I/Os, provided that no external oscillator is connected. The function of the OSC2/CLKOUT pin is determined by the CLKOUTEN Configuration bit. When CLKOUTEN is set (CLKOUTEN = 1), the pin functions as a general-purpose I/O. When CLKOUTEN is clear (CLKOUTEN = 0), the system instruction clock ($F_{OSC}/4$) is available as an output signal on the pin.

This says that I can use it for general purpose IO when set into internal clock sources. I am using the HFINTOSC which is internal.

I cant find a clear outline under "SOSCI" to tell me if I can disable it to use the pin as GPIO

Maybe I will just use a different pin for now. There is no real reason to use RA5 asside from the fact it is in line with the rest

```
159
160 uint16_t ADCread(int channel){
161     /*
162      * Channel:
163      * 0  -- RA4                6  -- RB7
164      * 1  -- RC5                7  -- RB6
165      * 2  -- RC4                8  -- RB5
166      * 3  -- RC3                9  -- RB4
167      * 4  -- RC6                10 -- RC2
168      * 5  -- RC7                11 -- RA2
169      */
```

I mapped the pins like this and now I get a reading on all 12 pins. There is some noise on pin 6, RB7 but I am not sure why. It is currently wire wrapped to the LED on the board so maybe that is causing the issue.

main.cSUN

PIC18\sunSensor.X\main.c

```
/** I N C L U D E S **********************************************************/
#include <xc.h>
#include <stdio.h>                      // Standard IO functions
#include <stdlib.h>                     // Standard library functions
#include <string.h>

#define _XTAL_FREQ 4000000     // One instruction cycle is 1 microsaec.

//This array hold the ADPCH bits for each pin
uint8_t channel_map[12] = {
    0b000100,   //RA4
    0b010101,   //RC5
    0b010100,   //RC4
    0b010011,   //RC3
    0b010110,   //RC6
    0b010111,   //RC7
    0b001111,   //RB7
    0b001110,   //RB6
    0b001101,   //RB5
    0b001100,    //RB4
    0b010010,    //RC2
    0b000010    //RA2
};

uint8_t data[24];

void configI2C(){
    I2C1CON0bits.EN = 0;    //Disable I2c
    //SCL1 -- RC1
    //SDA1 -- RC0

    //Set all pins to digital not analog
    ANSELCbits.ANSELC0 = 0;
    ANSELCbits.ANSELC1 = 0;

    /*Configure pins*/
    //Set pins
    TRISCbits.TRISC0 = 0;
    TRISCbits.TRISC1 = 0;
    //Set pins to open drain
    ODCONCbits.ODCC0 = 1;
    ODCONCbits.ODCC1 = 1;

    //Set pins to SDA and SCL output
    RC0PPS = 0x22;  //Set pin C0 to SDA
    RC1PPS = 0x21;  //Set pin C1 to SCL

    //Set pins to SDA and SCL input
    I2C1SDAPPS = 0b010000;  //Set C0 to SDA
    I2C1SCLPPS = 0b010001; //Set C1 to SCL

    /*Config I2C*/
```

```
    I2C1CON0bits.MODE = 0b000; //Set to client 7bit w/o masking
    I2C1CON0bits.CSTR = 0;      //Enable clocking (not stretching clock)

    I2C1CON1bits.CSD = 0;       //Disable clock stretching

    I2C1CON2bits.ACNT = 0;      //First transmission after address will be loaded to the byte count register
    I2C1CON2bits.GCEN = 1;      //Enable general address call -- will respond to address 0x00
    I2C1CON2bits.ABD = 0;
    I2C1CON2bits.SDAHT = 0b00;

    I2C1PIEbits.CNT1IE = 0;     //Enable interrupt when byte count reaches 0
    I2C1PIEbits.ACKTIE = 0;     //Enable interrupt for acknowledge clock stretching
    I2C1PIEbits.WRIE = 0;       //Enable interrupt for receiving data to process and determine ACK or NACK
    I2C1PIEbits.ADRIE = 1;      //Enable interrupt for processing address
    I2C1PIEbits.PCIE = 0;       //Disable interrupt when stop condition detected
    I2C1PIEbits.RSCIE = 0;      //Disable interrupt when restart condition detected
    I2C1PIEbits.SCIE = 1;       //Enable interrupt when start condition detected

    //Clear stop, start and restart flags
    I2C1PIRbits.PCIF = 0;
    I2C1PIRbits.SCIF = 0;
    I2C1PIRbits.RSCIF = 0;

    I2C1BTOC = 0x03;            //Set clock source to LFINTOSC (32kHz)
    I2C1BTObits.TOREC = 1;      //If timeout, reset module
    I2C1BTObits.TOBY32 = 0;     //Enable 32 prescaling
    I2C1BTObits.TOTIME = 0x23;  //Set timeout time to 35ms (32kHz / 32 prescale * 35 = 35ms)

    I2C1ADR0 = 0xFF;   //Set 7bit address
    I2C1ADR1 = 0xFF;   //Set 7bit address

    I2C1CLK = 0b0011;   //Set the clock to MFINTOSC (500kHz)
    I2C1BAUD = 0x04;    //Freq = I2C1CLK/(BAUD+1) = 500kHz/(4+1) = 100kHz

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


    IPR7bits.I2C1IP = 1;    //Set I2C interrupt to high prio
    PIR7bits.I2C1IF = 0;    //Clear interrupt bit
    PIE7bits.I2C1IE = 1;    //Enable I2C interrupt

    I2C1PIRbits.PCIF = 0;   //Clear stop flag

    I2C1CON0bits.EN = 1;    //Enable I2c

    __delay_ms(10);

    I2C1PIEbits.PCIE = 0;       //Enable interrupt when stop condition detected
    I2C1PIEbits.RSCIE = 1;      //Enable interrupt when restart condition detected
    I2C1PIEbits.SCIE = 1;       //Enable interrupt when start condition detected
}

void i2cStart(){
```

```
    /*This function gets called by the ISR when the start condition is detected*/
    while(I2C1STAT0bits.SMA == 0){};    //Wait until client mode is active
    if(I2C1STAT0bits.R == 1){   //If a read command was given
        I2C1CON0bits.CSTR = 0;  //Enable the clock (Stop stretching)
        for(int i = 0; i < 24; i ++){
            I2C1TXB = data[i];  //Load data into the transmit buffer
            //I2C1TXB = i;
            while(I2C1STAT1bits.TXBE == 0); //Wait until the buffer is emptied
        }
    }
    while(I2C1CON1bits.ACKT == 0);  //Wait for an ACK to make sure the byte is fully sent

    /*RESET FOR NEXT COMMUNICATION*/
    I2C1CON0bits.EN = 0;
    I2C1PIR = 0x00; //Clear the register that holds the flags for conditions
    I2C1CON0bits.EN = 1;
}

void ADCsetup(){
    /*Configure Pin*/
    TRISA = TRISA | 0b00010100; //Set RA2, RA4 to inputs
    TRISB = TRISB | 0b11110000; //Set RB4, RB5, RB6, RB7 to inputs
    TRISC = TRISC | 0b11111100; //Set RC2, RC3, RC4, RC5, RC6, RC7 to inputs

    ANSELA = ANSELA | 0b00010100; //Set RA2, RA4 to analog inputs
    ANSELB = ANSELB | 0b11110000; //Set RB4, RB5, RB6, RB7 to analog inputs
    ANSELC = ANSELC | 0b11111100; //Set RC2, RC3, RC4, RC5, RC6, RC7 to analog inputs

    /*Configure ADC*/
    ADREFbits.NREF = 0;     /*Set negative reference
                             * 0: Vref- is connected to AVss,
                             * 1: Vref- is connected to external Vref-*/

    ADREFbits.PREF = 0b00;    /*Set positive reference
                               * 11: Vref+ is connected to internal FVR module
                               * 10: Vref+ is connected to external Vref+
                               * 01: Reserved
                               * 00: Vref+ is connected to VDD*/

    /*IF 0b11 was selected above, configure the FVR module*/
    FVRCONbits.ADFVR = 0b11;    /*Configure the output voltage of the FVR
                                 * 11: 4x = 4.096V
                                 * 10: 2x = 2.048V
                                 * 01: 1x = 1.024V
                                 * 00: OFF*/
    FVRCONbits.EN = 0;  //Enable FVR module

    ADCON0bits.FM = 1;      //Right justify
    ADCON0bits.CS = 1;      //Set clock to ADCRC Clock (~600kHz)
    ADACQ = 32;             //Set acquisition time
}

uint16_t ADCread(int channel){
    /*
     * Channel:
     * 0  -- RA4                6  -- RB7
```

```
     * 1  -- RC5                7  -- RB6
     * 2  -- RC4                8  -- RB5
     * 3  -- RC3                9  -- RB4
     * 4  -- RC6                10 -- RC2
     * 5  -- RC7                11 -- RA2
     */

    /*Set the ADC channel to ground and read to clear charge*/
    ADPCH = 0b111011;
    ADCON0bits.GO = 1;  //Start acquisition
    while(ADCON0bits.GO); //Wait until done

    /*Set channel to selected pin and measure ADC*/
    ADPCH = channel_map[channel];   //Set ADC to pin
    ADCON0bits.ON = 1;      //Turn on ADC
    ADCON0bits.GO = 1;  //Start acquisition
    while(ADCON0bits.GO); //Wait until done
    uint8_t resultHigh = ADRESH;    //Read high register
    uint8_t resultLow = ADRESL;     //Read low register

    /*Convert 2 8bit to 1 16bit*/
    uint16_t result = resultHigh;
    result = result << 8;
    result = result | resultLow;

    return result;
}

void __interrupt(irq(I2C1)) ISR(void){
    if(I2C1PIRbits.SCIF == 1){
        I2C1PIRbits.SCIF = 0;
        i2cStart();
    }
}

void loop(){
    I2C1CNTL = 0x19;    //Set count to 24
    uint16_t temp = 0;
    for(int i = 0; i < 12; i++){
        temp = ADCread(i);
        data[2*i] = temp >> 8;
        data[2*i+1] = temp & 0xFF;
    }
}

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

    configI2C();
    ADCsetup();
    while(1){
        loop();
    }
    return;
}
```

This is the script that I have running on the PIC to be able to read the analog values.

Leave 3:30

26/07/23

July 26, 2023 8:07 AM

Time: 7.75h

Start 8:00

Nick asked me to look into using the external clock for the PIC as it is supposed to be more accurate than the internals.

I know yesterday I also saw something in the datasheet regarding tuning the internal clocks -- not sure if this was all of them or just a specific clock source.

12. OSC - Oscillator Module (With Fail-Safe Clock Monitor)

The oscillator module contains multiple clock sources and selection features that allow it to be used in a wide range of applications while maximizing performance and minimizing power consumption.

Clock sources can be supplied either internally or externally. External sources include:

- External clock oscillators
- Quartz crystal resonators
- Ceramic resonators
- Secondary Oscillator (SOSC)

Internal sources include:

- High-Frequency Internal Oscillator (HFINTOSC)
- Low-Frequency Internal Oscillator (LFINTOSC)
- Analog-to-Digital Converter RC Oscillator (ADCRC)

Special features of the oscillator module include:

- Oscillator Start-up Timer (OST): Ensures stability of quartz crystal or ceramic resonators
- 4x Phase-Locked Loop (PLL): Frequency multiplier for external clock sources
- HFINTOSC Frequency Adjustment: Provides the ability to adjust the HFINTOSC frequency
- Clock switching: Allows the system clock to switch between internal or external sources via software during run time
- Fail-Safe Clock Monitor (FSCM): Designed to detect a failure of the system clock (FOSC), primary external clock (EXTOSC) or secondary external clock (SOSC) sources. The FSCM automatically switches to an internal clock source upon detection of an FOSC failure.

The Reset Oscillator (RSTOSC) Configuration bits determine the type of oscillator that will be used when the device runs after a Reset, including when the device is first powered up (see the table below).

Table 12-1. RSTOSC Selection Table

RSTOSC	SFR Reset Values			Clock Source
	NOSC / COSC	NDIV / CDIV	OSCFRQ	
111	111	0000 (1:1)	0010 (4 MHz)	EXTOSC per FEXTOSC
110	110	0010 (4:1)		HFINTOSC @ 1 MHz
101	101	0000 (1:1)		LFINTOSC
100	100	0000 (1:1)		SOSC
011	Reserved			
010	010	0000 (1:1)	0010 (4 MHz)	EXTOSC + 4x PLL(1)
001	Reserved			
000	000	0000 (1:1)	1000 (64 MHz)	HFINTOSC @ 64 MHz

Note:

1. EXTOSC must meet the PLL specifications (see the data sheet Electrical Specifications).

If an external clock source is selected by the RSTOSC bits, the External Oscillator Mode Select (FEXTOSC) Configuration bits must be used to select the External Clock mode. These modes include:

- ECL: External Clock Low Power mode
- ECM: External Clock Medium Power mode
- ECH: External Clock High Power mode
- LP: 32 kHz Low-Gain Crystal mode

12.1.1 External Clock Sources

An external clock source can be used as the device system clock by performing one of the following actions:

- Program the RSTOSC and FEXTOSC Configuration bits to select an external clock source that will be used as the default system clock upon a device Reset.
- Write the NOSC and NDIV bits to switch the system clock source during run time.

12.1.1.1 EC Mode

The External Clock (EC) mode allows an externally generated logic level signal to be the system clock source. When operating in EC mode, an external clock source is connected to the OSC1/CLKIN input pin. The OSC2/CLKOUT pin is available as a general purpose I/O pin or as the CLKOUT signal pin.

EC mode provides three Power mode selections:

- ECH: High Power mode
- ECM: Medium Power mode
- ECL: Low Power mode

The Oscillator Start-up Timer (OST) is disabled when EC mode is selected; therefore, there is no delay in operation after a Power-on Reset (POR) or wake-up from Sleep. Because the PIC® MCU design is fully static, stopping the external clock input will have the effect of halting the device while leaving all data intact. Upon restarting the external clock, the device will resume operation as if no time had elapsed.

The figure below shows the pin connections for EC mode.

Figure 12-2. External Clock (EC) Mode Operation

12.1.1.2 LP, XT, HS Modes

The LP, XT and HS modes support the use of quartz crystals or ceramic resonators connected to the OSC1 and OSC2 pins, as shown in the figures below. These three modes select a low, medium, or high-gain setting of the internal inverter-amplifier to support various resonator types and speeds.

The LP Oscillator mode selects the lowest gain setting of the internal inverter-amplifier, and consumes the least amount of current. LP mode is designed to drive 32.768 kHz tuning-fork type crystals (watch crystals), but can operate up to 100 kHz.

The XT Oscillator mode selects the intermediate gain setting of the internal inverter-amplifier. Current consumption is at a medium level when compared to the other two modes. XT mode is best suited to drive crystal and ceramic resonators with a frequency range up to 4 MHz.

The HS Oscillator mode selects the highest gain setting of the internal inverter-amplifier, and consumes the most current. This mode is best suited for crystal and ceramic resonators that require operating frequencies up to 20 MHz.

The figures below show typical circuits for quartz crystal and ceramic resonators.

Table 12-2. NOSC/COSC Clock Source Selection Table

NOSC / COSC	Clock Source
111	EXTOSC(1)
110	HFINTOSC(2)
101	LFINTOSC
100	SOSC
011	Reserved
010	EXTOSC + 4xPLL(3)
001	Reserved
000	Reserved

Notes:

1. EXTOSC is configured via the FEXTOSC Configuration bits.
2. HFINTOSC frequency is determined by the FRQ bits.
3. EXTOSC must meet the PLL specifications (see the data sheet Electrical Specifications).

Table 12-3. NDIV/CDIV Clock Divider Selection Table

NDIV / CDIV	Clock Divider
1111-1010	Reserved
1001	512
1000	256
0111	128
0110	64
0101	32
0100	16
0011	8
0010	4
0001	2
0000	1

12.4 Active Clock Tuning (ACT)

Many applications, such as those using UART communication, require an oscillator with an accuracy of ± 1% over the full temperature and voltage range. To meet this level of accuracy, the Active Clock Tuning (ACT) feature utilizes the SOSC frequency of 32.768 kHz to adjust the frequency of the HFINTOSC over voltage and temperature.

Important: Active Clock Tuning requires the use of a 32.768 kHz external oscillator connected to the SOSCI/SOSCO pins.

Active Clock Tuning is enabled via the Active Clock Tuning Enable (ACTEN) bit. When ACTEN is set (ACTEN = 1), the ACT module uses the SOSC time base to measure the HFINTOSC frequency, and uses the HFINTOSC Frequency Tuning (TUN) bits to adjust the HFINTOSC frequency. When ACTEN is clear (ACTEN = 0), the ACT feature is disabled, and user software can utilize the TUN bits to adjust the HFINTOSC frequency.

Important: When the ACT feature is enabled, the TUN bits are controlled directly through module hardware and become read-only bits to user software. Writes to the TUN bits when the ACT feature is enabled are ignored.

The figure below shows the Active Clock Tuning block diagram.

12.5.8 OSCEN

Name: OSCEN
Address: 0x0B3

Oscillator Enable Register

Bit	7	6	5	4	3	2	1	0
	EXTOEN	HFOEN	MFOEN	LFOEN	SOSCEN	ADOEN		PLLEN
Access	R/W	R/W	R/W	R/W	R/W	R/W		R/W
Reset	0	0	0	0	0	0		0

Bit 7 – EXTOEN External Oscillator Enable

Value	Description
1	EXTOSC is explicitly enabled, operating as specified by FEXTOSC
0	EXTOSC can be enabled by a peripheral request

Bit 6 – HFOEN HFINTOSC Enable

Value	Description
1	HFINTOSC is explicitly enabled, operating as specified by OSCFRQ
0	HFINTOSC can be enabled by a peripheral request

Bit 5 – MFOEN MFINTOSC Enable

Value	Description
1	MFINTOSC is explicitly enabled
0	MFINTOSC can be enabled by a peripheral request

Bit 4 – LFOEN LFINTOSC Enable

Value	Description
1	LFINTOSC is explicitly enabled
0	LFINTOSC can be enabled by a peripheral request

Bit 3 – SOSCEN Secondary Oscillator Enable

Value	Description
1	SOSC is explicitly enabled, operating as specified by SOSCPWR
0	SOSC can be enabled by a peripheral request

Bit 2 – ADOEN ADCRC Oscillator Enable

Value	Description
1	ADCRC is explicitly enabled
0	ADCRC may be enabled by a peripheral request

Bit 0 – PLLEN PLL Enable(1)

Value	Description
1	EXTOSC multiplied by the 4x system PLL is used by a peripheral request
0	EXTOSC is used by a peripheral request

Note:
1. This bit only controls external clock source supplied to the peripherals and has no effect on the system clock.

8.5.1 CONFIG1

Name: CONFIG1
Address: 0x300000

Configuration Byte 1

Bit	7	6	5	4	3	2	1	0
		RSTOSC[2:0]				FEXTOSC[2:0]		
Access		R/W	R/W	R/W		R/W	R/W	R/W
Reset		1	1	1		1	1	1

Bits 6:4 – RSTOSC[2:0] Power-Up Default Value for COSC
This value is the Reset default value for COSC and selects the oscillator first used by user software. Refer to COSC operation.

Value	Description
111	EXTOSC operating per FEXTOSC bits
110	HFINTOSC with HFFRQ = 4 MHz and CDIV = 4:1. Resets COSC/NOSC to b'110'.
101	LFINTOSC
100	SOSC
011	Reserved
010	EXTOSC with 4x PLL, with EXTOSC operating per FEXTOSC bits
001	Reserved
000	HFINTOSC with HFFRQ = 64 MHz and CDIV = 1:1. Resets COSC/NOSC to b'110'.

Bits 2:0 – FEXTOSC[2:0] External Oscillator Mode Selection

Value	Description
111	ECH (external clock) above 8 MHz
110	ECM (external clock) for 500 kHz to 8 MHz
101	ECL (external clock) below 500 kHz
100	Oscillator not enabled
011	Reserved (do not use)
010	HS (crystal oscillator) above 4 MHz
001	XT (crystal oscillator) above 500 kHz, below 4 MHz
000	LP (crystal oscillator) optimized for 32.768 kHz

The software does not have "CONFIG1" defined for me to set the values. I also tried just using the address but that does not work.

None.

2.5.14 Specifying Configuration Bits

The `#pragma config` directive can be used to program the Configuration bits for a device. The pragma has the form:

```
#pragma config setting = state|value
```

where *`setting`* is a configuration setting descriptor (e.g., `WDT`), *`state`* is a descriptive value (e.g., `ON`) and *`value`* is a numerical value.

Use the native keywords discussed in the Differences section to look up information on the semantics of this directive.

2.5.14.1 EXAMPLE

The following shows Configuration bits being specified using this pragma.

```
#pragma config WDT=ON, WDTPS = 0x1A
```

2.5.14.2 DIFFERENCES

The 8-bit compilers have used the `__CONFIG()` macro for some targets that did not already have support for the `#pragma config`.

The 16-bit compilers have used a number of macros to specify the configuration settings.

The 32-bit compilers supported the use of `#pragma config`.

https://ww1.microchip.com/downloads/en/devicedoc/S0002053g.pdf

I found this that says there is a __CONFIG() macro. But not how to use it.

```
#define _XTAL_FREQ 1000000
#include <xc.h>

void setup(){
    __CONFIG(FEXTOSC, 0b110);
    OSCENbits.EXTOEN = 1;
    OSCCON1bits.NOSC = 0b111;

    /*Config pin for testing*/
    TRISBbits.TRISB7 = 0;
}

void loop(){
    LATBbits.LATB7 = 1;
    __delay_ms(100);
    LATBbits.LATB7 = 0;
    __delay_ms(100);
}

void main(void) {
    setup();
    while(1){
        loop();
    }
    return;
}
```

I made this test script. I can see on the scope that the B7 pin has a period of 200ms (100ms on 100ms off)
When I pull the EXT clock wire out of the CPLD, the period suddenly changes to 50ms, which would make sense if its defaulting back to the 4MHz HFINTOSC

I was just using RB5 as the external oscillator pin which was set by default. I could set it to another pin and then I should have everything I need.

Actually. CLKIN is not reprogrammable. It has to be on RB5. So I guess that is settled.

I modified my UART script to use this was the Oscillator. I redid the calculation to set the BAUD register, but got a value of 6.5. last time by decimal was 0.04 so I felt the whole number would be close enough. I found I could set BRGS value to 1 which would multiply the value by 4 -- which means I use the same number on this clock as the last one.

I found I still needed to reduce the register value by 1 or 2 to get the right timing, same as before. This didn't seem to make a difference.

Battery Management System

What is a battery management system (BMS)?

- Monitors battery
- Protects against over and under charging
- Estimates remaining charge
- Optimizes battery performance

5.4 Battery Pack

The CubeSat Battery pack shall be made with four Panasonic NCR18650B cells, connected in series, and held together structurally with a 3d printed metal bracket. The battery pack shall be charged up by the solar panels while in operation. The battery pack also serves as the only form of energy storage for the CubeSat once it has been launched.

The batteries will be mounted to the structure if the CubeSat with the use of an aluminum beam that goes across the middle ring in the structure as seen in Figure 5-9 below. The current mounting design still requires additional testing and iterations to ensure that the system would be able to undergo the stresses experienced during launch and deployment without failing. The battery bracket is going to be manufactured using additive manufacturing methods while the mounting beam will be machined out of aluminum.

A resistive heater shall be placed in between the four cells to provide the necessary heating that may be required during operation in eclipse.

The battery back uses 4 cells connected in Series.

Panasonic

Lithium Ion

NCR18650B

Features & Benefits

- High energy density
- Long stable power and long run time
- Ideal for notebook PCs, boosters, portable devices, etc.

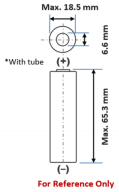
Specifications

Rated capacity(1)	Min. 3200mAh
Capacity(2)	Min. 3250mAh Typ. 3350mAh
Nominal voltage	3.6V
Charging	CC-CV, Std. 1625mA, 4.20V, 4.0 hrs
Weight (max.)	48.5 g
Temperature	Charge*: 0 to +45°C Discharge: -20 to +60°C Storage: -20 to +50°C
Energy density(3)	Volumetric: 676 Wh/l Gravimetric: 243 Wh/kg

* At temperatures below 10°C, charge at a 0.25C rate.

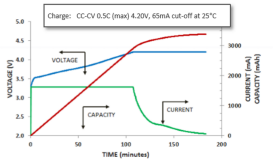
(1) At 20°C (2) At 25°C (3) Energy density based on bare cell dimensions

Dimensions

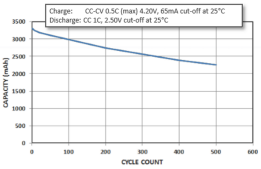


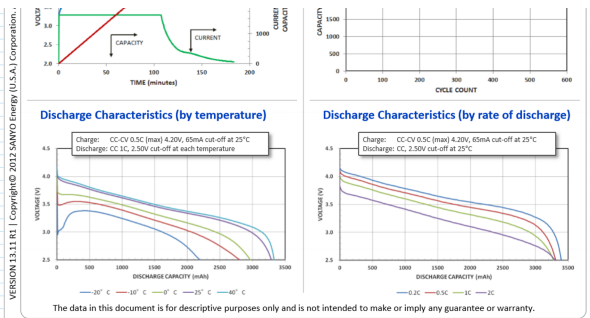
For Reference Only

Charge Characteristics



Cycle Life Characteristics



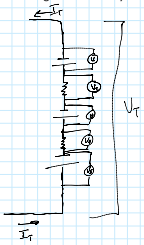


The data in this document is for descriptive purposes only and is not intended to make or imply any guarantee or warranty.

For more information on how Panasonic can assist you with your battery power solution needs, visit us at www.panasonic.com/industrial/batteries-oem, e-mail secsales@us.panasonic.com, or call (469) 362-5600.

Measuring the voltage on each cell is pretty simple. That is just an analog reading between + and -.
Reading the current would involve a shunt resistor between cells where the current is measured.

Although I don't think you can get the current from each individual cell if they are in series.



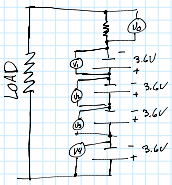
Normally, current in = current out — Is that true with batteries?
I would assume current is equal but voltage changes which is how energy is stored

I am not finding a clear answer online. My interpretation is that the current is the same in and out and then the voltage of the battery changes. But I haven't found it explicitly stated.

https://www.quora.com/Does-more-current-flow-into-a-battery-than-out-of-it#:~:text=The%20current%20flowing%20out%20of,negative%20end%20of%20the%20battery.

While this is Quora, it makes the most sense to me so Ill believe it

This means I really only need 1 shunt for the whole circuit, and a voltage on each cell, so a minimum of 5 analog pins.



$3.6 \times 4 = 12 + 2.4$
$14.4V$

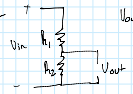
The Max possible voltage at V_0 is the 14.4V source
The PIC can not handle that
but realistically, the batteries will never be shorted.
What is the minimum W of the satellite?
1.323W is the average modal power consumption for Power saving mode
(Might be different now with RPI)

$P = IV \quad I = \frac{V}{R}$

$P = \frac{V^2}{R} \quad R = \frac{U^2}{P} = \frac{14.4^2}{1.323} = 156.73\Omega$

Meaning in the lowest power mode the circuit should have ~156.73Ω of resistance
If I want a max of 4.096V (FVR module max) over the shunt

Voltage divider



$V_{out} = \frac{R_2}{R_1+R_2} \cdot V_{in}$

$\frac{V_{out}}{V_{in}}(R_1+R_2) = R_2$

$\frac{V_0}{V_i}R_1 + \frac{V_0}{V_i}R_2 = R_2$

$\frac{V_0}{V_i}R_1 = R_2 - \frac{V_0}{V_i}R_2$

$\frac{V_0}{V_i}R_1 = R_2\left(1 - \frac{V_0}{V_i}\right)$

$\frac{\frac{V_0}{V_i}R_1}{1-\frac{V_0}{V_i}} = R_2$

$\frac{\frac{4.096}{14.4}(156.73)}{1-\frac{4.096}{14.4}} = \frac{0.284(156.73)}{1-0.284} = \frac{44.581}{0.715} = 623.2\Omega$

This has to be wrong. I want to minimize this resistance, to have minimal effect on the rest of the circuit generating 4V is way too much.

If I used a 1Ω resistor, I would have $\frac{1}{157.73} \cdot 14.4 = 0.09V$ → there is no internal reference for voltages this low

I dont know if this makes sense → as Power increases, R_1 decreases, then V decreases.

$V = IR$ means increase in current
Means increase in voltage

One of my assumptions must be false

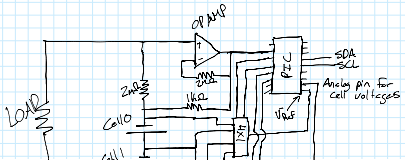
$P = VI \quad I = \frac{1.323W}{14.4V} = 0.0918A$

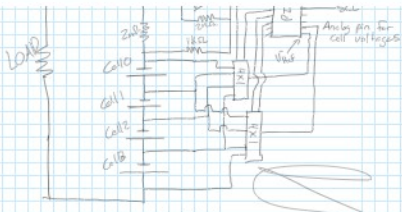
$V = IR$ (if 1Ω) $V = 0.0918V$ which is what I before

$I = \frac{10W}{14.4V} = 0.694A \rightarrow 0.694V$

technically that power consumption does not include the resistor, so there is some error

IF idel pwr is 10W, the power lost to the resistor is $P = \frac{V^2}{R} = \frac{0.694^2}{1} = 0.48W$ → this is a lot considering the power limitations





This circuit should have everything to monitor the batteries
It measures the total current and the voltage of each cell

3 Analog pins
4 bytes for multiplexing — I noticed there need to be ones, I was thinking digital —

https://www.digikey.ca/en/products/detail/analog-devices-inc/ADG704BRMZ-REEL/994602

This is an example of an analog multiplexer — what is the name
However I am going to have to address the resistance — it is supposed to be ~25Ω of resistance, so I imagine I need a big pulldown resistor to minimize the effect on the reading. Although I guess concerning adding error to the reference and signal lines

Although surely the ADC has some sort of pull down. But the ADC would be pulling down to 0-nF likely...

I'll check the datasheet

IF that power is 10W, the power lost to the resistor is
$P = \frac{V^2}{R} = \frac{0.694^2}{1} = 0.48W$ — this is a lot considering the power limitations

apparently 2-8 mΩ shunts are common
now $0.694A \cdot 2\times10^{-3} = 0.0014V$

$P = \frac{0.0014^2}{0.002} = 0.0196W$ that is way better, but I would need a very small reference voltage. Maybe the 2mV

At the low end $0.0918A \cdot 0.002\Omega = 0.000184V$

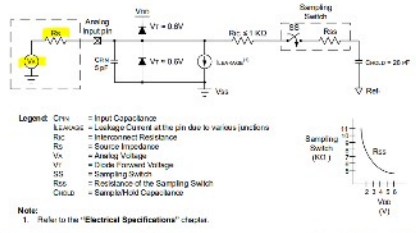
IF my 12bit ADC measures 0–2mV, I have a resolution of 0.488μV which is plenty
2mV would relate to $\frac{2mV}{2m\Omega} = 1A = 14.4W$

I'm being dumb — you would never use a 2mV reference, Use the 2.048V FVR on the PIC and use an Amp with a gain of 1k on the voltage
Actually, I can use a 2k gain and then use the 4.096V FVR — this way I don't have to change references to measure the cell voltage

40.3 ADC Acquisition Requirements

For the ADC to meet its specified accuracy, the charge holding capacitor (C_{HOLD}) must be allowed to fully charge to the input channel voltage level. The analog input model is shown in Figure 40-4. The source impedance (R_S) and the internal sampling switch (R_{SS}) impedance directly affect the time required to charge the capacitor C_{HOLD}. The sampling switch (R_{SS}) impedance varies over the device voltage (V_{DD}). The maximum recommended impedance for analog sources is 10 kΩ. As the source impedance is decreased, the acquisition time may be decreased. After the analog input channel is selected (or changed), an ADC acquisition time must be completed before the conversion can be started. To calculate the minimum acquisition time, Equation 40-1 may be used. This equation assumes an error of 1/2 LSb. The 1/2 LSb error is the maximum error allowed for the ADC to meet its specified resolution.

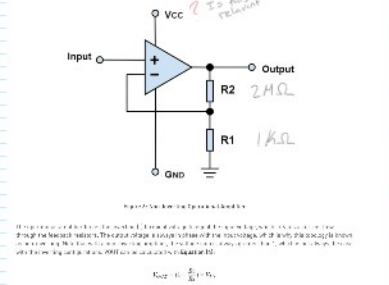
Figure 40-4. Analog Input Model



Legend: CPIN = Input Capacitance
ILEAKAGE = Leakage Current at the pin due to various junctions
RIC = Interconnect Resistance
RS = Source Impedance
VA = Analog Voltage
VT = Diode Forward Voltage
SS = Sampling Switch
RSS = Resistance of the Sampling Switch
CHOLD = Sample/Hold Capacitance

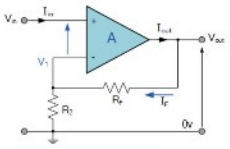
Note:
1. Refer to the "Electrical Specifications" chapter.

My understanding from this is that the 4Ω is would be fine
impedance is resistance + reactance — $f=50$ Hertz reactance ≈ 0
$4\Omega \ll 10k\Omega$
Maybe it is fine without pulldown

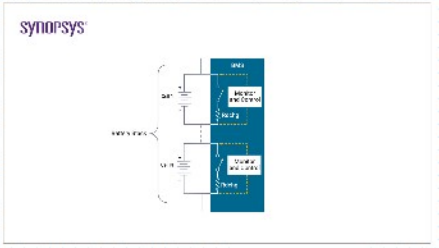


Non-inverting Operational Amplifier Configuration

This doesn't have VCC



https://www.synopsys.com/glossary/what-is-a-battery-management-system.html



This website talks about having a system to bypass cells during charging when the cell is close to its limit to allow all cells to charge evenly

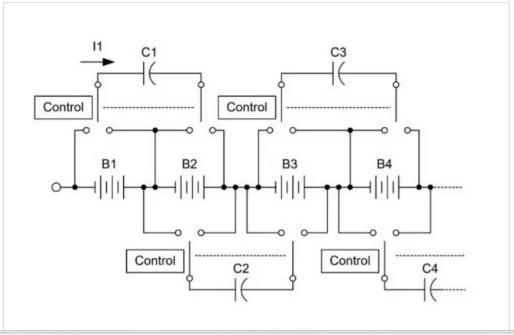
I already outlined a system for monitoring the voltage — I just need a transistor to switch when the charge reaches a certain threshold.

The website says to use an "appropriately sized" discharge resistor. No idea what that would be

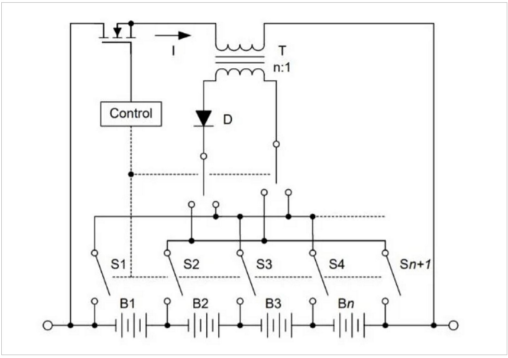
https://www.planetanalog.com/a-primer-on-battery-management-system-bms-for-evs/

I started reading this to see if I could get more information on this discharge/bleed resistor. However the way that they are explaining it is that the energy is not diverted to other cells, but rather wasted as heat in the resistors. The CubeSat does not have enough power to be wasting any. I need to look further into the active system where power is transferred from one cell to another.

There are other solutions that enable energy transfer from high cells to low cells using specific circuits rather than using the bypass resistor for the same. One such approach is to redistribute the energy between the cells by connecting the capacitor to a high cell and a low cell and is typically referred to as the charge shuttles method. This method also allows faster equilibration in cells placed far apart in the pack by providing the capability of remote cell connection as shown in Figure 2 [3]. The drawback of this technique is high losses incurred during the charging phase of the capacitors. The known efficiency of this process is only about 50% and efficiency is higher only during the end of discharge as the transfer rate is proportional to voltage differences.



A cell-balancing method called inductive converters overcomes the disadvantage of small voltage differences between cells. In this method, the battery pack energy is transferred to a single cell by channeling the battery pack current through a transformer as shown in Figure 3 [4]. The transformer is connected to the cell that requires an additional charge. The downside of this approach is the use of an additional transformer which leads to an increase in cost and size along with reduced overall efficiency.

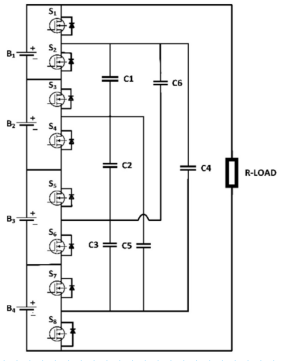


https://eepower.com/technical-articles/active-and-passive-battery-pack-balancing-methods/

The Capacitive method seems fairly simple, even if it is slightly less efficient than inductive. I still don't completely understand how it works. What Capacitors are connected to charge a certain cell.

https://pdf.sciencedirectassets.com/308665/1-s2.0-S2352152X21X00059/1-s2.0-S2352152X21002693/main.pdf?X-Amz-Security-Token=IQoJb3JpZ2luX2VjELH%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJIMEYCIQC5gPDieAY6oensL19wYVUZJydAHUuYURUyInLPl923EwlhAJI1LvaANA8Tic48kFqqvTErsutBYXoPRozKZ6heq3utKrMFCEoQBRoMMDU5MDAzNTQ2ODY1IgzBWXf8nKfCie1k7AwqkAXt3VE8OapYIaIJrTtfBxJEqfXK059L%2BRSTe0uPHWNWddoBbo3bPvBuOxetIwJDeV1pgZQXmc6cvY%2FwCxg%2FbAWdPo2dau9IbHhrZ63TEycDTpHh9OF0DaDxBOnyxsN2xgkhk2p%2B18N1NykMtK6eGWH7foS%2FnFjXrFTmUMPfsYkcm6Tf6tOtzOZtRlspbqjS8AXxw59N5kyXfz3g7xUn4c8wPkr3ZPss1o2bRfuUZgMqunOTwOPz7P%2F6DNgWGSjYeJBkADGBdm%2FQN1bCGYKpTQ0N4RCvDUpwy5Cl70MeDY4tKRDaYYepaD2TjZW3l4fVd5mfA1qLxAIHhAnQ1Sf%2Fr%2FasradA1cB%2FTNbYo1MY02V%2FcndkV6pgICGbu61D4EiQ%2BvZH0sLhhxVdjJHCWNRRHvF5i7WiuR8bGlmVehe98fUEo485iXCeyyNIwAI7R79CoMSzHV0qRRVALUYXKQVxYEWa7vdnW0uAPVwhGzwmnu9klQ5LUKKlbNqozVsWfzYiaaWnNjALIlkSi6IVPOtX8KHidVmiuqjGObzq5f6l9gjAV114br1rwTy4Afx8wBfZQ4SdlG2iXmDZ9WnMyzoDdP5gGxfzeZUqPRnzKRea6cSWdz2BleReoGf2ysEqybbm0ORUSiStNcAqZvWPvm5BJaaqheBf7vKNN5mgRq%2Fm5gvwddMzh9hJBb2y43NAY4zpKYhAfQ7Ubby7x8sq72gOWCgGGjO6fBrcI28piDq5wQVXNzykeTRuEP%2FFLjxGaxu0h765JwY%2BrgZI7lU8tL19nAMkGhkPuix%2FlhxG03gRFt2EoVEqSrzgjHHXR8dOV8BtiVE4GHCAqucaB99d1O1SRIG8TYyCYcSFRjTwWmsjAEaBDCnnlWmBjqwAVr6Blao5fSqBidAdCNmDLkVedHzypFGuvCvF25Sh5kUuyV6lxwPCxT%2F7J7QQ6TZUdy31dpqI9eBTJjdcx8IwXk7qhWfV%2BSADbJJ%2F9NyfRXPB371hQ6CTIt4VmUNK6cb8nbrsOAQWxhu9IVWbbII17BL6pIsqV1w19SxKNXds4%2FVNNeZs%2FOX4R6mKQiNTGBGYP1ggN339XhbeHyHKjs%2BZS09MD8YHzxoV2ENlauWXJx9&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20230726T175147Z&X-Amz-SignedHeaders=host&X-Amz-Expires=300&X-Amz-Credential=ASIAQ3PHCVTY7U2P6RAK%2F20230726%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=10c9bae262399b38c4962484781981140880892dd34c2aed1fa7b8e5b9e013b&hash=9e011d29d5c9278c9de1ab3a5813ae399bb5ef877ecf2eddd86541d03c753729&host=68042c943591013ac2b2430a89b270f6af2c76d8dfd086a07176afe7c76c2c61&pii=S2352152X21002693&tid=spdf-e80e06d9-1e91-471b-8953-df4b84dc46ec&sid=8cfc7d174526e14db6-a20a-462171a9493dgxrqa&type=client&tsoh=d3d3LnNjaWVuY2VkaXJlY3QuY29t&ua=0607520a070756005352&rr=7ece7e201ab61a03&cc=ca

I found this research paper which is on a method for balancing 4 cells. The paper found that their method had an efficiency of around 90% when a slow enough frequency was used. This is really good. The circuit is also super simple, it is just a few capacitors and 8 transistors.



The control system appears to be state based -- it checks the status of the charge in each of the cells, and then depending on the relative charge it reconfigures the capacitors to discharge some cells into other cells. I'm sure it is a bit more complicated to implement but seems reasonable. The states are clearly defined in the paper.

With this I would need 8 IO pins.

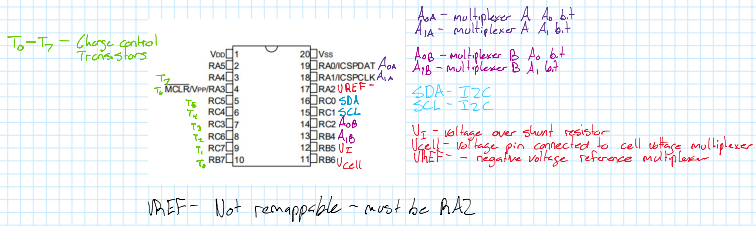
I think I need to check and make sure I can get all of this on one IC

Pins:
1 analog for current
1 analog for voltages
1 analog for VREF-
4 digital for multiplexers
8 digital for control circuit
2 I2C pins
As long as I don't run into any issues with reserved pins, I think that I have used all of the pins exactly....

That makes 17, then there is VSS and VDD.

So I have one spare pin.

I don't think that anything on the PIC will be critical enough to require an external oscillator.



VREF- Not remappable - must be RA2

Leave 4:00

27/07/23

Thursday, July 27, 2023 8:02 AM

Start 8:00AM

I have been thinking more about the Battery management system.
I have designed the monitering circuit, and found a design to handle charge balancing
The only other things I can think of is preventing overcharging
and preventing overdischarging

I think the only method to prevent over discharging is to reduce the load by disabling peripherals. Although I think the main computer should be in charge of that

I wonder if the current design has some method to prevent overcharging by like disconnecting the output of the panels?

Nothing is labelled or clear in the image in the TRA, I will need to find the eagle file to look into this further

I was thinking the last pin I have could be used to control a FET on the output of the converter that generates the charging voltage
I had also considered having 1 digital pin that would run to the RPI and serve as an emergency trigger - If something happened with the I2C and the RPI was not getting the battery status and the cells were getting too low, the PIC could turn on this digital pin to signal that power consumption needs to be reduced ASAP

But I think I need to use the last Pin for controlling the overcharge, unless I gave that responsibility to the PI. I think it would make sense to have the Pi in control of enabling/disabling hardware
But again I feel like I almost need another emergency wire for overcharging if I go this route

I may review the paper and see if I can use less pins for the switches. If I used some digital logic I should be able to connect to only 3 pins to set the 6 Modes

Journal of Energy Storage 38 (2021) 102521

operating states such as charging and discharging states. For analysis purposes, it is assumed that the four-cell module voltages are $V_1>V_2>V_3>V_4$.

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.

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.

Mode 2 (T_1-T_2): In this mode, MOSFET switches S_1 and S_3 are in ON state and all the remaining switches are in OFF state as shown in Fig. 3 (a) and Fig. 3(b). During this mode, the B_1 and B_2 cells are discharged through the capacitors to the other cells. This mode 2 will continue until B_1 & B_2 voltage becomes equal to the third-highest voltage cell (B_3) in the series-connected cells of the battery string.

Mode 3 (T_2-T_3): In this mode, MOSFET switches S_1, S_3 and S_5 are in ON state and all the remaining switches are in OFF state as shown in Fig. 3(a), Fig. 3(b) and Fig. 3(c). During this mode B_1, B_2 and B_3 cells discharge through the capacitors to the fourth cell B_4. This mode 3 will continue until four cells voltage is equal in the series-connected cells of the battery string.

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.

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.

Mode 5 (T_4-T_5): In this mode, MOSFET switches S_8 and S_6 are in ON state and all the remaining switches are in OFF state as shown in Fig. 4 (a) and Fig. 4(b). During this mode, B_4 and B_3 cells are discharged through the capacitors to the other cells. This mode 5 will continue until B_4 and B_3 cell voltage is equal to B_2 cell voltage in a series-connected battery string.

Mode 6 (T_5-T_6): In this mode, MOSFET switches S_8, S_6 and S_4 are in ON state and all the remaining switches are in OFF state as shown in Fig. 4(a), Fig. 4(b) and Fig. 4(c). During this mode B_4, B_3 and B_2 cells are discharged through the capacitors to the fourth cell (B_1). This mode 6 will continue until four cells voltage is equal in the series-connected cells of the battery string.

Mode	S1	S2	S3	S4	S5	S6	S7	S8
State 1	1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0
3	1	0	1	0	1	0	0	0
State 2	0	1	0	1	0	1	0	1
4	0	0	0	0	0	0	0	1
5	0	0	0	0	0	1	0	1
6	0	0	0	1	0	1	0	1

Mode (ABC)	S1	S2	S3	S4	S5	S6	S7	S8
1 (111)	1	0	0	0	0	0	0	0
2 (110)	1	0	1	0	0	0	0	0
3 (101)	1	0	1	0	1	0	0	0
State 1 (100)	1	0	1	0	1	0	1	0
4 (011)	0	0	0	0	0	0	0	1
5 (010)	0	0	0	0	0	1	0	1
6 (001)	0	0	0	1	0	1	0	1
State 2 (000)	0	1	0	1	0	1	0	1

S1

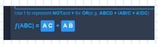
A/BC	00	01	11	10
0	0	0	0	0
1	1	1	1	1

S1 = A

S2

A/BC	00	01	11	10
0	1	0	0	0
1	0	0	0	0

S2 = !A&!B&!C



S3

A/BC	00	01	11	10
0	0	0	0	0
1	1	1	0	1

S3 = A & (!B v B&!C)

S4

A/BC	00	01	11	10
0	1	1	0	0
1	0	0	0	0

S4 = !A & !B

S5

A/BC	00	01	11	10
0	0	0	0	0
1	1	1	0	0

S5 = A & !B

S6

A/BC	00	01	11	10
0	1	1	0	1
1	0	0	0	0

S6 = !A & (!B v B&!C)

S7

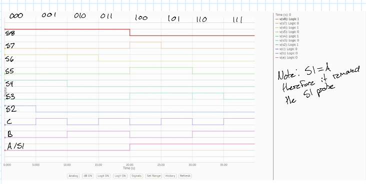
A/BC	00	01	11	10
0	0	0	0	0
1	1	0	0	0

S7 = A& !B & !C

S8

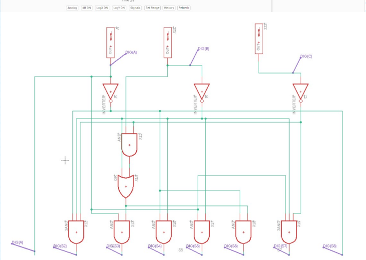
A/BC	00	01	11	10
0	1	1	1	1
1	0	0	0	0

S8 = !A



Note: S1=A therefore it removed the S1 probe

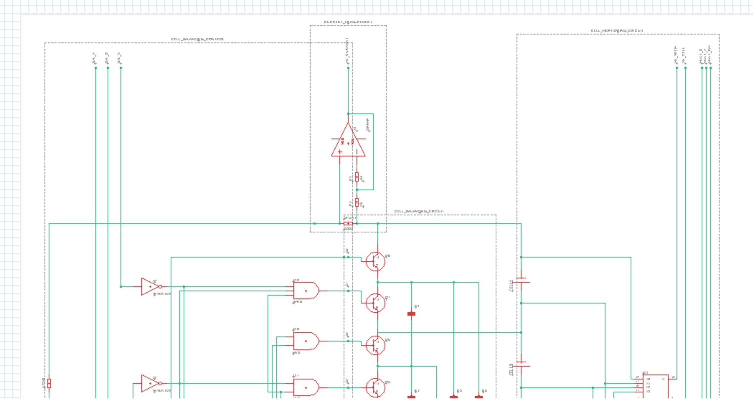
Mode (ABC)	S1	S2	S3	S4	S5	S6	S7	S8
1 (111)	1	0	0	0	0	0	0	0
2 (110)	1	0	1	0	0	0	0	0
3 (101)	1	0	1	0	1	0	0	0
State 1 (100)	1	0	1	0	1	0	1	0
4 (011)	0	0	0	0	0	0	0	1
5 (010)	0	0	0	0	0	1	0	1
6 (001)	0	0	0	1	0	1	0	1
State 2 (000)	0	1	0	1	0	1	0	1

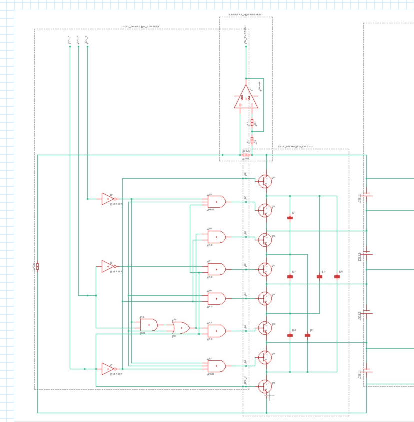


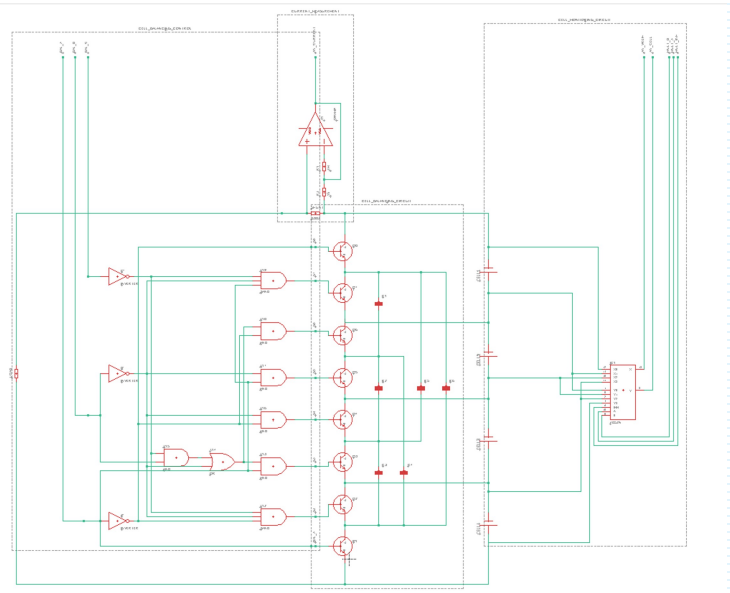
8.3.5 Cell Balancing

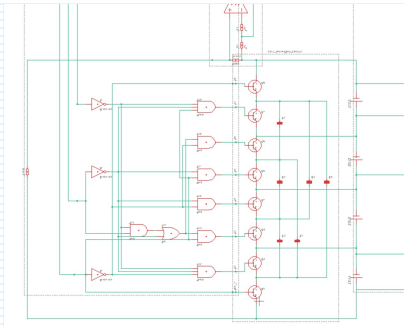
The BQ79606A-Q1 integrates a MOSFET for each cell to enable passive balancing with a minimum of external components. Passive cell balancing slowly discharges individual higher voltage cells to balance the voltage across all of the cells in the stack. Cell balancing reduces the aging rate differences between cells to extend the battery pack overall lifetime. The drawback to passive balancing is heat generation. The energy during discharge is dissipated across an external resistor generating heat. The cell balancing current must be chosen as a tradeoff between the time it takes to balance and the heat generated in the process. The cell balancing algorithm is fully configurable and runs autonomously once enabled. Cell balancing is terminated either when the individual timer expires, or the cell voltage reaches a programmed threshold.

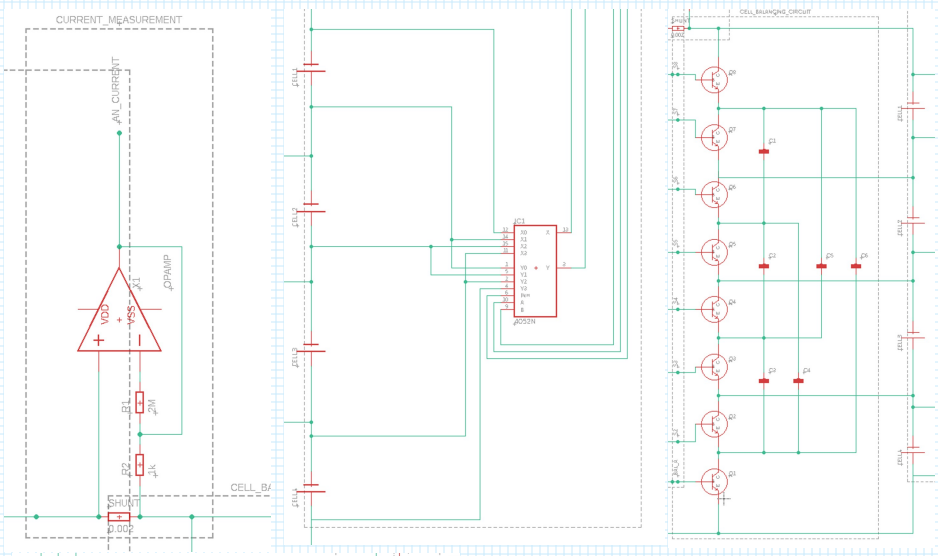
My digital logic circuit appears to work!

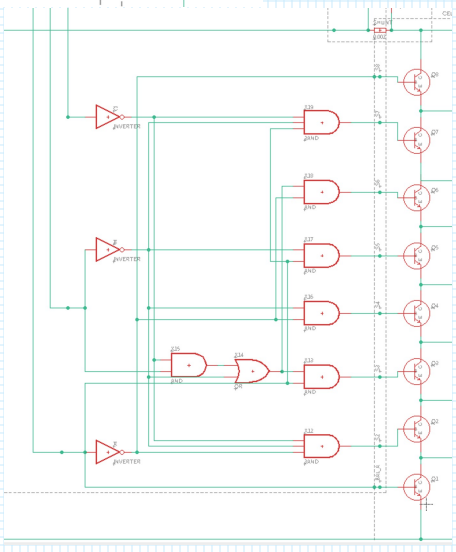




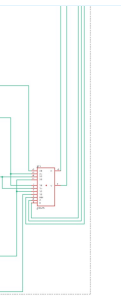








I put a the cell balancing circuit, cell balancing control circuit, current measurement circuit, and cell measurement circuit into one EAGLE file. This should cover all of the systems, unless I choose to allow this chip to cut power to various peripherals or the charging circuit.



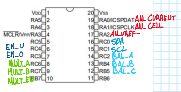
8.3.5 Cell Balancing

The BQ79606A-Q1 integrates a MOSFET for each cell to enable passive balancing with a minimum of external components. Passive cell balancing slowly discharges individual higher voltage cells to balance the voltage across all of the cells in the stack. Cell balancing reduces the aging rate differences between cells to extend the battery pack overall lifetime. The drawback to passive balancing is heat generation. The energy during discharge is dissipated across an external resistor generating heat. The cell balancing current must be chosen as a tradeoff between the time it takes to balance and the heat generated in the process. The cell balancing algorithm is fully configurable and runs autonomously once enabled. Cell balancing is terminated either when the individual timer expires, or the cell voltage reaches a programmed threshold.

External resistors set the cell balancing current. Figure 18 illustrates the circuit and current flow during balancing. Cell balancing is available with a CBDONE comparator function for cell voltages greater than 2.8V. ADC reads are available during cell balancing. Cell balancing sequencing is programmable to balance cells in two banks, the odd cells and the even cells. Additionally, a cell balancing comparator is integrated that monitors the cell voltages and terminates cell balancing once the voltage V_{CBDONE} threshold is reached. The cell balancing time is programmable for each individual cell. Additionally, a duty cycle timing function is built into the BQ79606A-Q1 to switch between banks during balancing to achieve a simultaneous stack balance. Using these timing features, the host microcontroller controls the specific algorithm used for cell balancing.

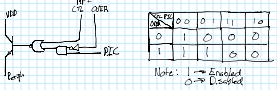
While active, the status of the individual cell balancing switch is indicated in the CB_SW_STAT register. As the cell balancing for each cell completes, the CB_DONE register is updated. When the timer or voltage is satisfied for a particular cell, the switch is disabled and the corresponding CB_DONE[CELL*] bit is set.

The chip that they were going to use only uses a passive cell balancing method, so it is wasting energy when balancing the cells. My circuit uses the capacitive balancing which should have a much better efficiency.



Now I have extra pins for EM_O and EM_U for my emergency pins.
If the main computer does not acknowledge Overcharging or Undercharging and a critical threshold is reached, the PIC will trigger the emergency pins.

I want these to disable charger for Overcharging & on all power to all peripherals excluding BMS PIC, RF1, UHF during undercharge.
The pins will also run to GPIO pins on the RF1, informing the main computer of the issue.
I also want the RF1 to be able to override the PIC to disable in case there is an issue.



SPI				
0	0	0	1	1
0	1	0	0	0
1	1	1	0	0

Note: 1 → Enabled
0 → Disabled

If I added one of these circuits to a both the battery charger and the power loop going to peripherals then my BMS would be complex of:
- Monitoring Individual Cell voltage
- Monitoring total current
- Balancing Cell charge actively
- Disabling charging to protect against overcharge (Overridable)
- Disabling peripherals to protect against undercharge (Overridable)

I should added a primary power loop & secondary power loop

Primary includes:
- BMS
- RF1
- UART

Secondary includes
- everything else

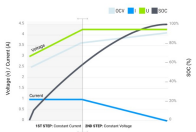
Thus the BMS will have outputs of:
- I2C
- Primary EN, 3.3V
- Secondary EN, 3.3V
- EM_O
- EM_U

Inputs of:
- Charger output
- RF1 Override U
- RF1 Override O

And some GPIOs

I still need to find a battery charger - unsure which. Thinks this will be done on the PCC
Also I'm aware that a lot of Managment systems have temperature measurements and protection.
Might need to add this too

https://www.swft.com/energizing-ist/charging-your-lithium-ion-batteries-5-expert-tips-longer-lifespan#:~:text=Charging%20properly%20of%20lithium%20ion,voltage%20to%20preserve%20the%20electrode.



LI-ION CHARGING PROCESS CC/CV

A battery charger is supposed to apply a constant current until the desired voltage is reached, then the charger switches to a constant voltage until the current drops below a certain threshold.

Datasheet for the cells says that they need 1650mA CC and 4.2V CV

The previous design used a Buck-Boost converter before the Battery Management IC.

I can use the same thing

> The LTC4020 battery charger can provide a constant-current/constant-voltage charge algorithm (CC/CV), constant-current charging (CC), or charging with an optimized 4-step, 3-stage lead-acid battery charge profile. Maximum converter and battery charge currents are resistor programmable.

It does the CC and CV, do I just need to tell it when to do each? That is pretty simple.

https://www.analog.com/media/en/technical-documentation/data-sheets/ltc4020.pdf

28/07/23

July 28, 2023 9:18 AM

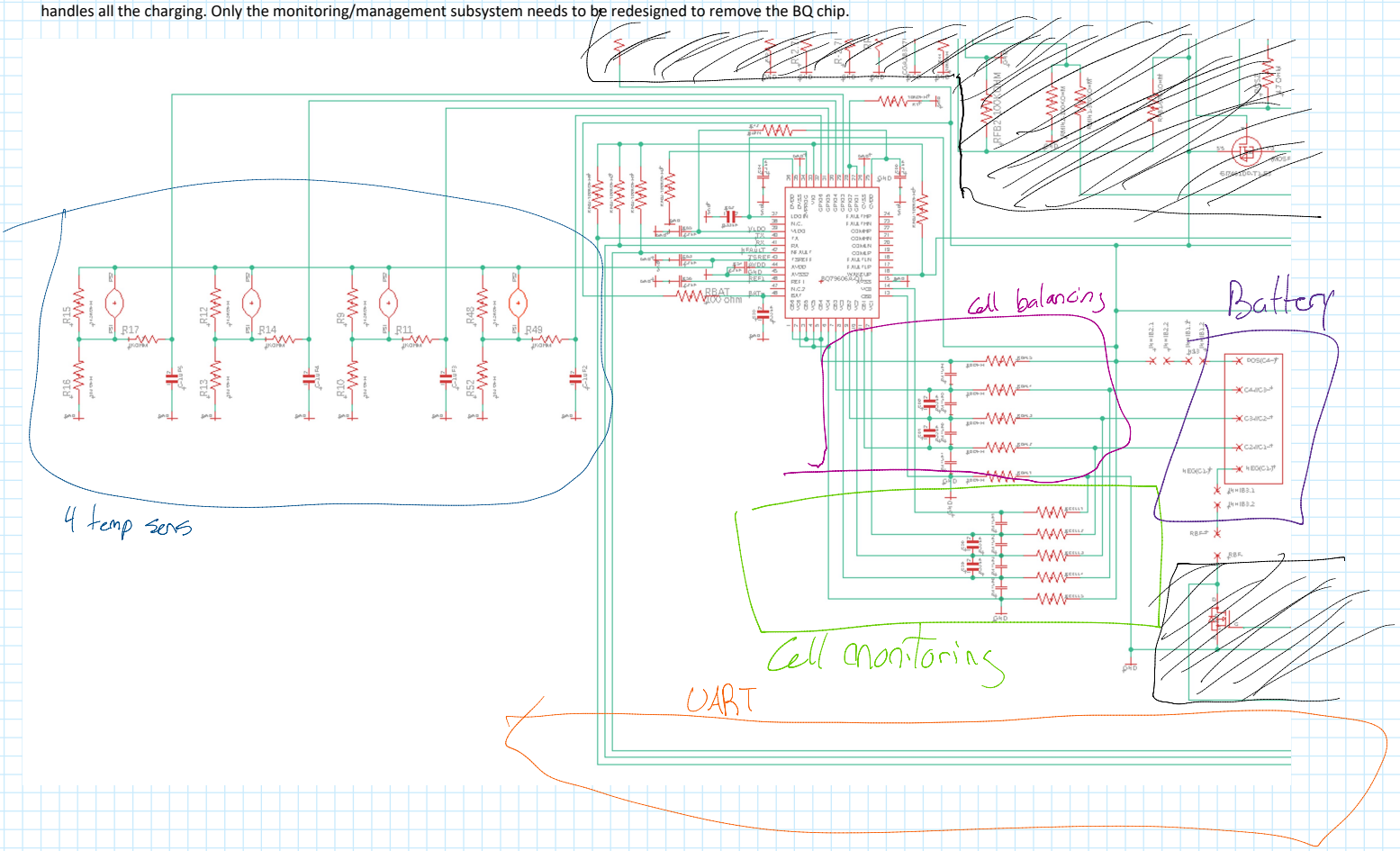
Start 8:00

I have been reading through the datasheet for the Buck-Boost converter battery charger. It seems that this is used to charge batteries, It also appears to have some protection built in. I am not sure what the other BMS chip they were using was for. Maybe just the cell balancing but I am not sure. Like I mentioned before, that chip uses passive cell balancing which is not good.

I have also found the Eagle file for the complete circuit shown in the TRR report. Looking at the circuit, it actually makes sense. I think whoever wrote the report did not know what they were talking about cause the report makes no sense. The report makes it sound like the LTC chip is just a voltage converter. And that it converts the solar voltage down to a charging voltage and sends that to the BQ chip. Then the BQ chip charges the battery and monitors the cells and keeps all the cells in a healthy operating range.

This is not the case. The BQ chip is NOT a battery charger like stated in the report, it is only monitoring and management. The LTC chip does all of the charging on the overall battery, then the BQ chip is used to manage the individual cells in the battery to make sure they remain in a safe operating mode.

This means that I do not need to redesign as much as I thought. I have looked into the LTC chip datasheet and it seems like a good choice, and it handles all the charging. Only the monitoring/management subsystem needs to be redesigned to remove the BQ chip.

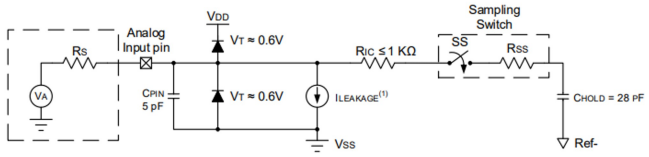


That is all the physical connections between the BQ chip and the rest of the circuit. I already designed a circuit for the cell monitoring and management. The UART will be replaced with I2C. The temperature sensor circuitry can be reused, I will just need to run the wires to the PIC.

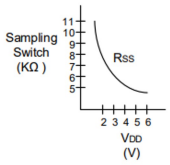
The only concerns that I have are whether I can accurately multiplex the analog pins. More specifically, the VREF- pin.

Additionally, I am concerned that the voltage might be too high for the upper cells. Cause each cell is only supposed to go to 4.2 max (probably wouldn't push it to the max). But I have to move VREF- up the battery back to only measure the cell voltage. I am concerned that even though the analog to VREF- will be <4.2V, the analog to GND will be much higher. I am concerned that this may cause issues.

Figure 40-4. Analog Input Model



Legend: CPIN = Input Capacitance
ILEAKAGE = Leakage Current at the pin due to various junctions
RIC = Interconnect Resistance
RS = Source Impedance
VA = Analog Voltage
VT = Diode Forward Voltage
SS = Sampling Switch
RSS = Resistance of the Sampling Switch
CHOLD = Sample/Hold Capacitance



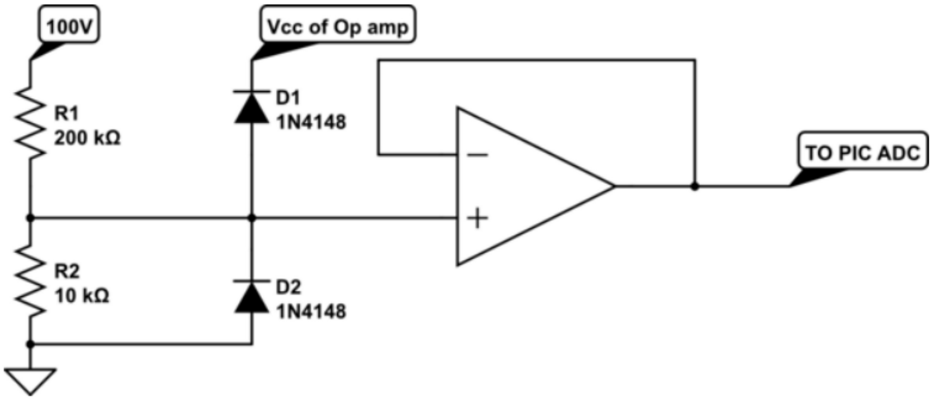
Note:
1. Refer to the **"Electrical Specifications"** chapter.

This is the ADC input circuit model.
I do not understand the purpose of the Diodes. If the analog voltage is in between Vss and Vdd, there should be no current through either diode. If the voltage is higher than higher than Vdd, I will be running current to Vdd which seems strange. Vdd is 3.3V, and the ADC has a FVR of 4.096V. With the 0.6V forward voltage, the diode will not allow current to flow until 3.3+0.6=3.9 volts is seen on the analog input. Which is lower than the 4.098 FVR, meaning that with the designed inputs, it will run current to Vdd…

I was researching and found this:

4

You may want to put a protection diode on the input of the op amp, the high voltage could easily burn up the op amp if it somehow crossed the barrier of the resistor. Take care while routing, sometimes through hole parts can be advantageous to avoid arcing and to provide separation as creepage and clearance for 100V is 0.71mm.



simulate this circuit – Schematic created using CircuitLab

This is the exact same diode configuration, and is apparently a protection circuit…

https://learn.digilentinc.com/Documents/376

This circuit is supposed to short the time to Vdd or Vss. This website says that the idea is that the Vdd and Vss should be able to handle show overvoltages. If there is a lot of current or the signal will be on for a long time, this will not protect.

I think that my idea of moving the ground reference will not work. I think I am going to end up damaging the PIC, or at the very least not being able to measure the full range.

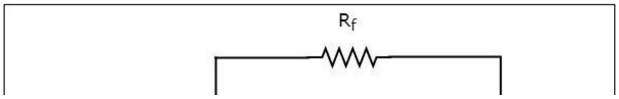
Instead, what if I found a way to subtract the voltage at the negative terminal to the voltage at the positive terminal, then sent that to the ADC. I would keep VREF- on GND, and just subtract the voltages

Subtractor

A subtractor is an electronic circuit that produces an output, which is equal to the difference of the applied inputs. This section discusses about the op-amp based subtractor circuit.

An op-amp based subtractor produces an output equal to the difference of the input voltages applied at its inverting and non-inverting terminals. It is also called as a **difference amplifier**, since the output is an amplified one.

The **circuit diagram** of an op-amp based subtractor is shown in the following figure –

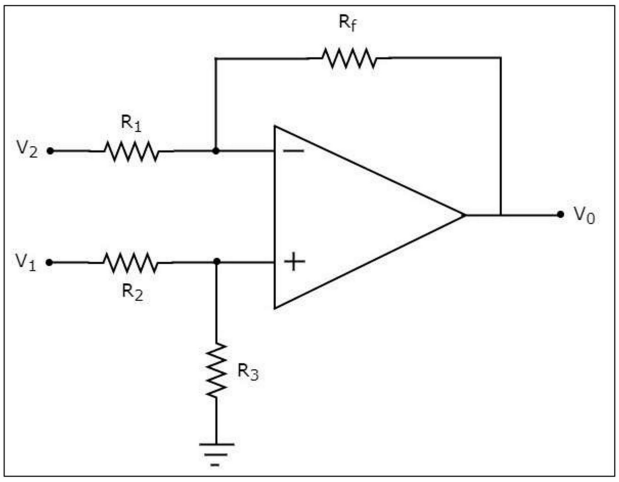


Subtractor

A subtractor is an electronic circuit that produces an output, which is equal to the difference of the applied inputs. This section discusses about the op-amp based subtractor circuit.

An op-amp based subtractor produces an output equal to the difference of the input voltages applied at its inverting and non-inverting terminals. It is also called as a **difference amplifier**, since the output is an amplified one.

The **circuit diagram** of an op-amp based subtractor is shown in the following figure –



https://www.tutorialspoint.com/linear_integrated_circuits_applications/linear_integrated_circuits_applications_arithmetic_circuits.htm#:~:text=An%20op%2Damp%20based%20subtractor,output%20is%20an%20amplified%20one.

Thus, the op-amp based subtractor circuit discussed above will produce an output, which is the difference of two input voltages V_1 and V_2, when all the resistors present in the circuit are of same value.

https://www.digikey.ca/en/products/detail/analog-devices-inc/AD8137YRZ-REEL7/671064
https://www.digikey.ca/en/products/detail/texas-instruments/THS4524MDBTREP/3588982?s=N4IgjCBcpgbB0QGMoDMCGAbAzgUwDQgD2UA2iAMwUAcArNdSALgEAOAlCAMrsBOASwB2AcxABfQgCYADAE4ALFFApIGHAWJkQSliA5cAkkPa4RuPhMIBaKcuRR%2BAV00lI5Ws3GSQd9yAATAVRUC1wTASwAAnQAW1ZMYIELLyA

FUNCTIONAL BLOCK DIAGRAM

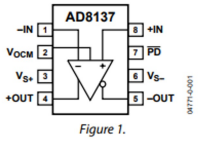
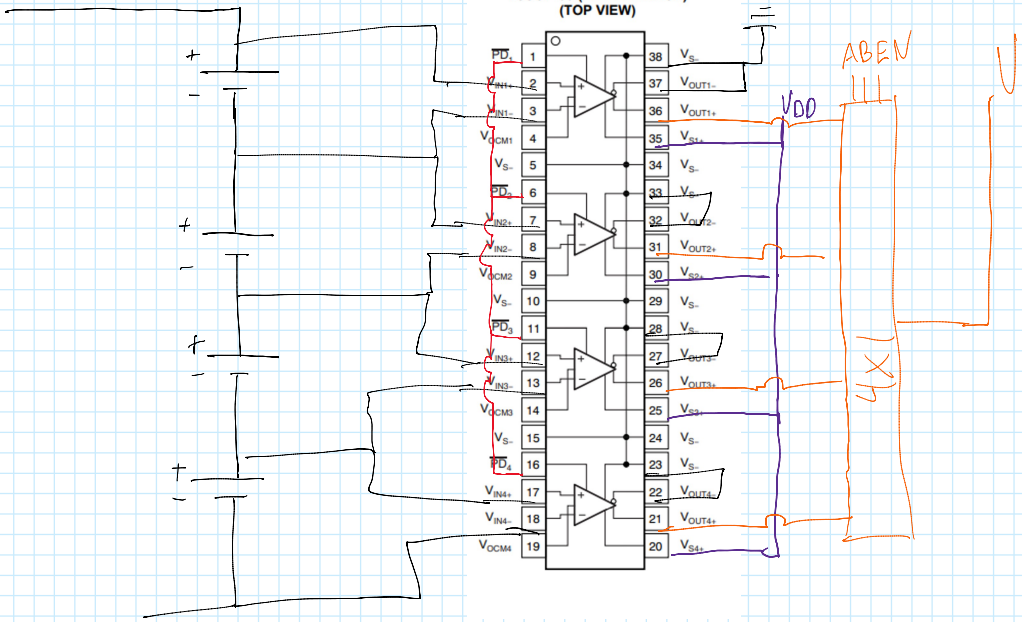


Figure 1.

PD ⇒ Power down – reduces current when not being used

DEVICE INFORMATION

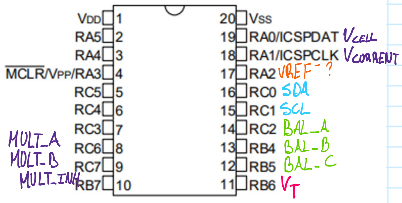
TSSOP-38 (DBT PACKAGE)
(TOP VIEW)



https://app.ultralibrarian.com/details/16728419-103F-11E9-AB3A-0A3560A4CCCC/Texas-Instruments/THS4524MDBTREP?ref=digikey&exports=EagleV6&open=exports

I downloaded an Eagle model for this 4 channel subtractor

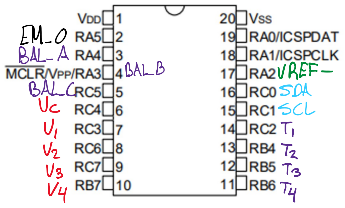
It did not work.



Without multiplex

- 4 Vcell
- 1 Vcurrent
- 2 I2C
- 3 Bal
- 4 V_T

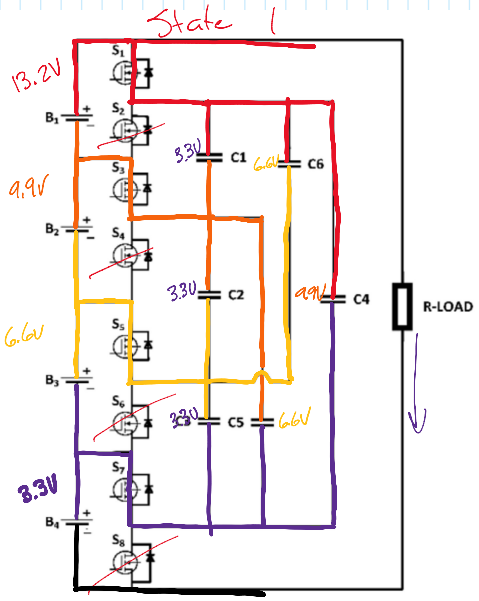
→ 14 pin I might not need multiplexing

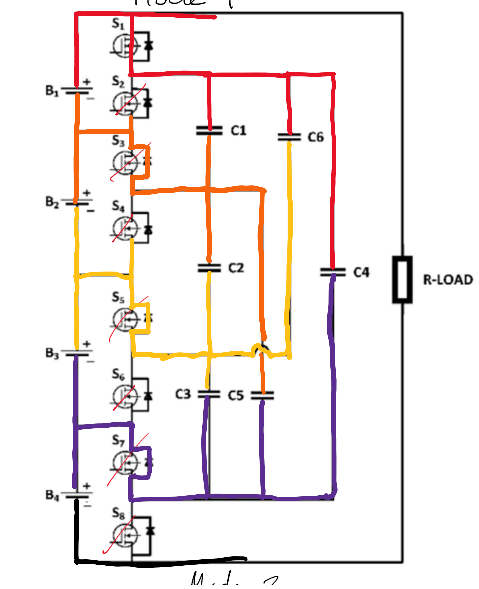


Do I need to measure the total battery voltage? or can I sum the cell voltages — Theoretically it has to be the sum — but maybe it would be helpful to measure too for reference — I would need to step it down

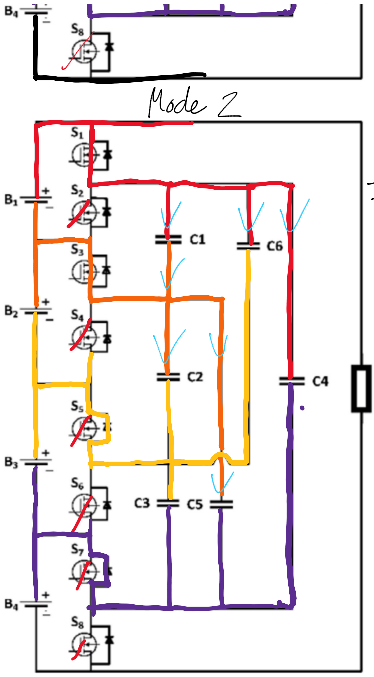
And I have 3 left over pins
I was planning to give the PIC the capability to cut power to the charge circuit to prevent over charging and cut power to peripherals to prevent undercharging.
But the LTC already does that, so I don't think that is needed.

I don't think their is any protection/management for undercharge in the circuit, so maybe I can still include that





Mode (ABC)	S1	S2	S3	S4	S5	S6	S7	S8
1 (111)	1	0	0	0	0	0	0	0
2 (110)	1	0	1	0	0	0	0	0
3 (101)	1	0	1	0	1	0	0	0
State 1 (100)	1	0	1	0	1	0	1	0
4 (011)	0	0	0	0	0	0	0	1
5 (010)	0	0	0	0	0	1	0	1
6 (001)	0	0	0	1	0	1	0	1
State 2 (000)	0	1	0	1	0	1	0	1



I am starting to see how this works
The switch allows current to flow out of the cell
This will enable discharging of the cell

When the switch is not closed, the diode gives a path for the power from the closed switches to go
I forget the diodes in my initial Eagle schematic

I still don't really understand why there needs to be state 1 and state 2, and when each is used. From what I can tell, they function the same, but in reverse. I guess that B4 is the highest voltage, state 1 cant balance cause all modes charge B4. And if B1 is the highest voltage, state 2 cant balance. But if B2 or B3 are the highest voltage, Either state should work?

	M1	M2	M3	M4	M5	M6
B1	D	D	D	C	C	C
B2	C	D	D	C	C	D
B3	C	C	D	C	D	D
B4	C	C	C	D	D	D

So Lets say the voltages are

B1	4.2
B2	3.6
B3	3.3
B4	3.0

Clearly State 1 is needed. Which mode is best? The obvious answer is Mode 1 -- this will discharge B1 the highest voltage.

But Mode 2 would discharge the two highest cells into the two lowest cells.
Which may be better if B1 and B2 are both at a high voltage.

B1	4.2
B2	4.1
B3	2.5
B4	2.3

Like with this configuration, B1 and B2 would discharge into B3 and B4 which could be best.
How do I combine the voltages to see what is the best?

	M1	M2	M3	M4	M5	M6
Discharge	B1	B1, B2	B1, B2, B3	B4	B4, B3	B4, B3, B2
Charge	B2, B3, B4	B3, B4	B4	B1, B2, B3	B2, B1	B1

What if I average the cells voltages for charging and discharging cells, then maximize the difference.

B1	4.2
B2	4.1
B3	2.5
B4	2.3

Using this example again, I can fill in the average voltage into each cell in the table above.

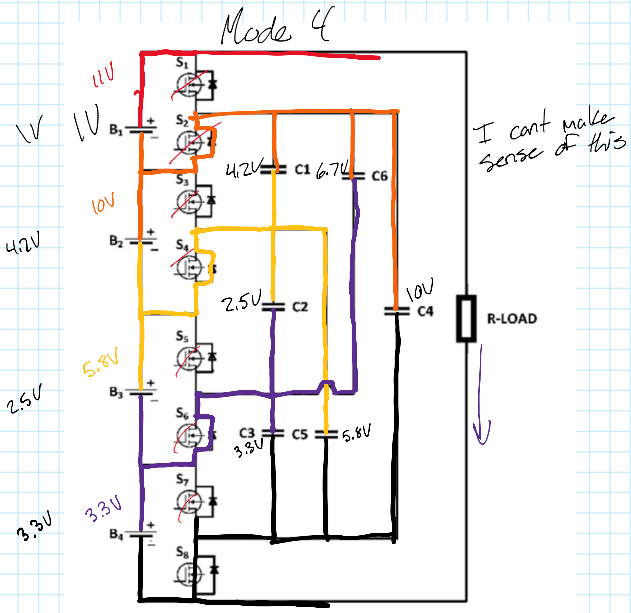
	M1	M2	M3	M4	M5	M6
Discharge	4.2	4.15	3.6	2.3	2.4	2.966
Charge	2.966	2.4	2.3	4.25	4.15	4.2
Difference	1.233	1.75	1.3	-1.95	-1.75	-1.204

Mode 2 clearly has the largest difference. This is what I was expecting.

Cell	Voltage
B1	1
B2	4.2
B3	2.5
B4	3.3

	M1	M2	M3	M4	M5	M6
DISCHARGE	1	2.6	2.566667	3.3	2.9	3.333333
CHARGE	3.333333	2.9	3.3	2.566667	2.6	3.3
DIFFERENCE	-2.33333	-0.3	-0.73333	0.733333	0.3	0.033333

Something like this says to use cell B4 to charge the rest. But B2 has a higher voltage, so what would happen? Will B4 just charge B1 and B3, while B2 stays constant? Or will B2 start discharging into the others?



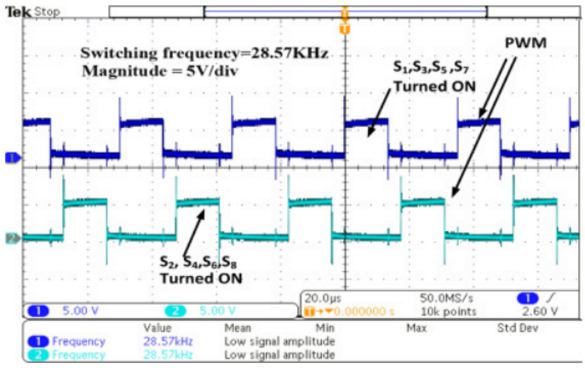
In the proposed prototype system, the dSPACE RTI2102 micro box generates a pair of complementary PWM signals to drive the MOSFET switches of the circuit. The cell voltages are monitored using the AD modules. Tektronix MD3012 Oscilloscope is used to capture the output results. The list of hardware components used in the implementation of the proposed structure is given in Table3.

Table 3. Specifications of the experimental setup parameters.

S.No	Components	Quantity	Rating/Value
1	Samsung ICR 18650-26J lithium-ion cells	4	Cell nominal voltage: 3.63V, Ah Rating: 2550 mAh Standard charge: CCCV, 1.50A, 4.20 ±0.05V, 150mA cut-off Discharge cut-off voltage:2.75V

I am reading through the research paper again to try and see if I can figure out what they were doing. They has said they are using a switching frequency of 28kHz and are controlling the FETs with a PWM signal. I don't really understand why you would need a PWM signal if you were using a conditional measurement to determine the state.

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.



This is looking like they just switch between State 1 and State 2, none of the modes defined are actually used…

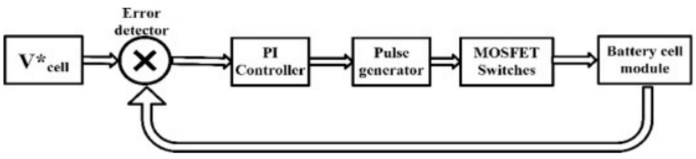
With the proposed cell balancing technique, the unbalanced cell voltages are equalized. It takes nearly 1100 seconds for cell voltages equalization at 28.57KHz switching frequency. Fig. 14 shows the voltages across the cells after equalization, i.e., B_1=3.95V, B_2=3.86V, B_3=3.95V, B_4=3.86V.

But does anything even charge in state 1 and state 2? The position of the FETS was outlined but It wasn't verbally described like the others.

But the system is "closed loop" meaning there needs to be feedback, it cant just be always cycling state 1 and state 2

2. Description of the proposed cell voltage balancing strategy

The block diagram of the proposed Closed-Loop Switched-Capacitor Structure (CLSCS) is shown in Fig. 1. A closed-loop control strategy based on the switched capacitor concept has been developed to maintain the constant voltage across each cell. In this closed-loop control strategy, the actual voltage of each cell is sensed and compared with the reference voltage V^*_{cell}. If the cell's actual voltage is greater than or equal to the reference value, then the switch corresponding to that cell turns OFF; otherwise, the switch is turned ON until the cell gets charged to its nominal value. This process is repeated in a closed-loop manner. So, that cell voltage will be equalized/balanced.



Download : Download high-res image (105KB)

Download : Download full-size image

Fig. 1. Block diagram of the proposed control strategy.

I don't understand what they mean by the switch corresponding to a cell. Cause there are 2 switches per cell. I guess that each switch is used in the different mode.

But the way that they word it suggests that like Cell 2 can be targeted, but there is no configuration of modes that will cause only Cell 2 to be charged. So in mode 1, switch 1 turns on which drains cell 1, that adds up. But in mode 3, S1, S3, S5 are on, which drains Cell 1, 2, 3 into cell 4.

You cant target cell 2 specifically. So do you just alternate between State 1 and State 2, and do like a tiered conditional

Like in mode 1:

```
if(B3 > ref){
        S5 on
} else {
        S5 off
        If(B2 > ref){
```

```
            S3 on
        } else {
            S3 off
            If(B1 > ref)
        }
}
```