Parallel Battery Management Evaluation Board

“Power to the People”

Team 1
Antonio Alonso, David Liu, Eric Cho, Harry Katsaros, Sunwoo Park

ANALOG DEVICES
AHEAD OF WHAT’S POSSIBLE™

Project Impact: Parallel Battery Management

- Parallel battery management systems are necessary for all devices with two or more batteries powering one device or multiple components of a device
- Allows simultaneous, proportional charging for multiple devices or components

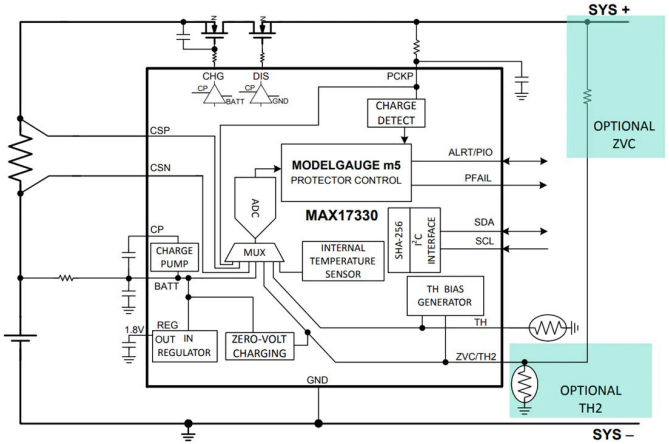
Airpod Pro

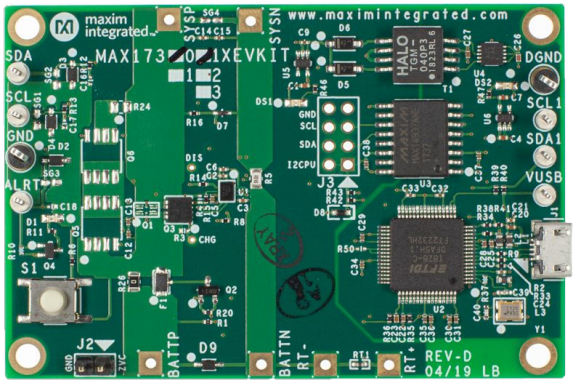
Ring Camera

Meta Quest 2

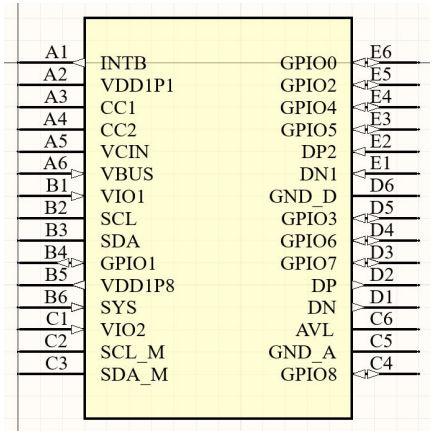
Project Description

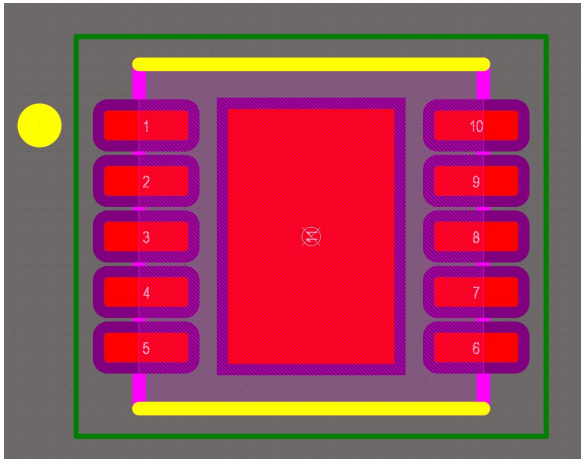
We are designing an incorporated system with a Type-C PPS input and software to showcase the parallel battery management functionality of the MAX17330 (charger, fuel gauge, and protector for lithium ion batteries) for customer technology demonstrations and trade shows.

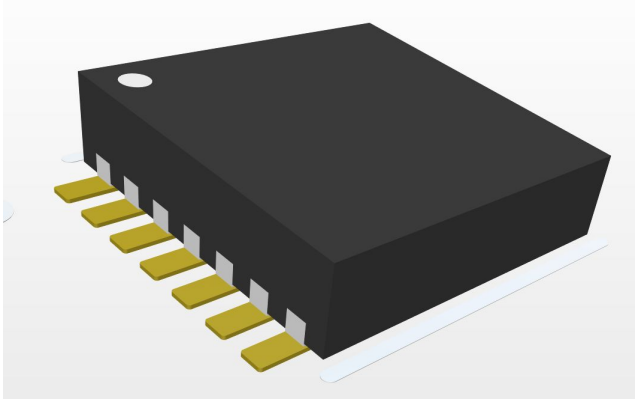




Progress: Completed Altium Schematic Symbols & Footprints

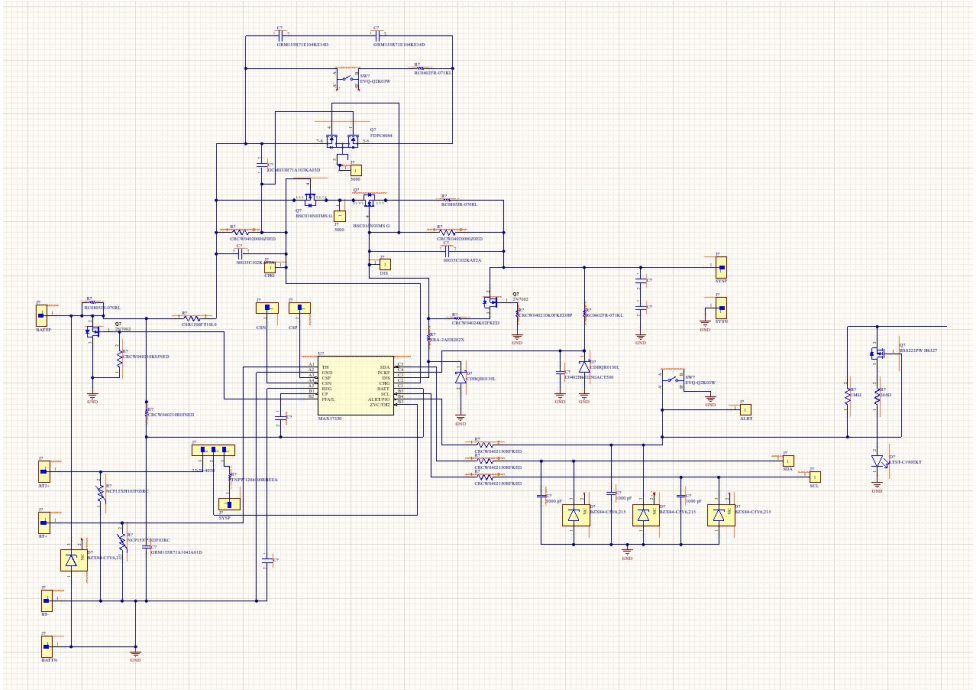






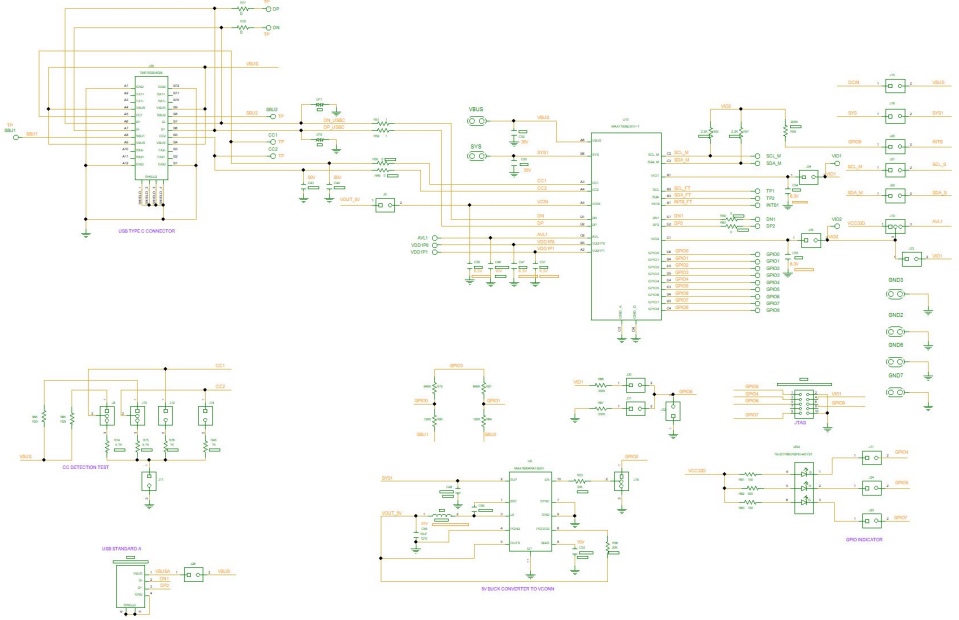
Progress: Partially Completed Schematic in Altium

- Proved to be very tedious
- Lots of components needed to be made from scratch
- Would not be feasible within our timeline

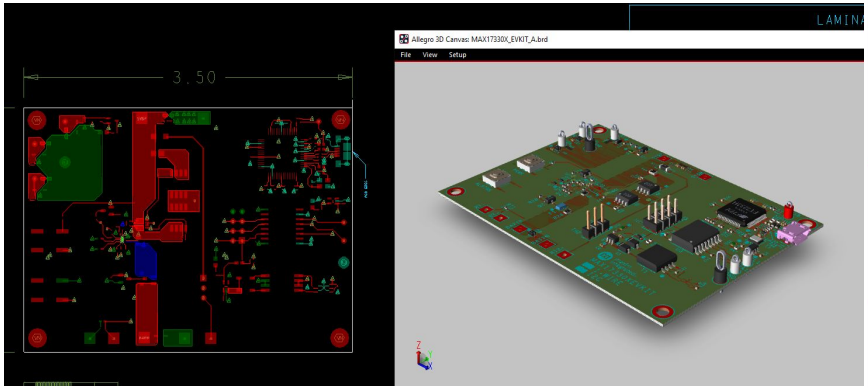


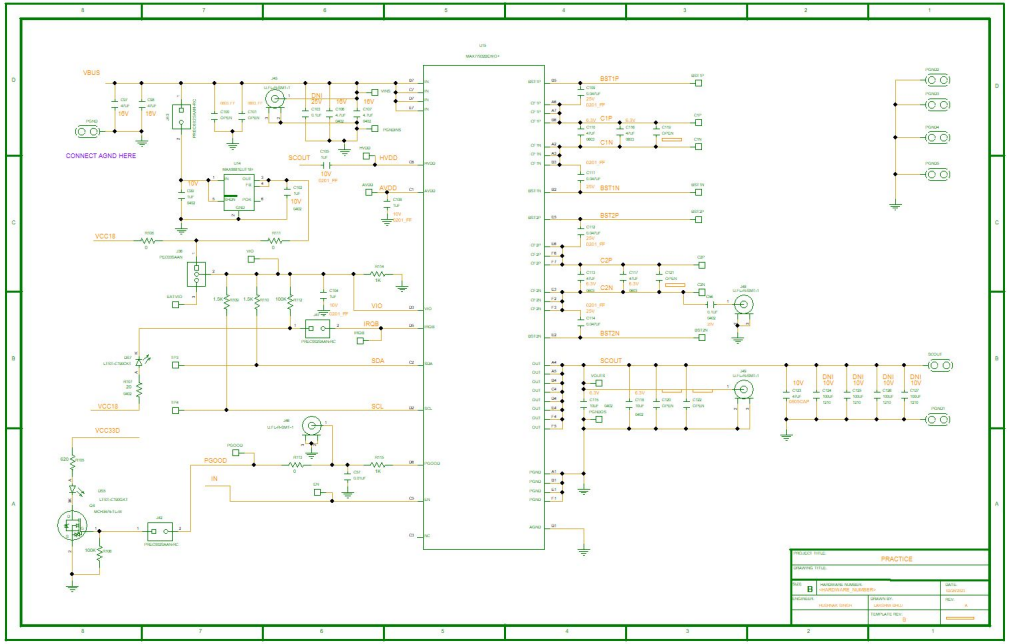
Progress: Completed Schematic in Allegro

- Switched from Altium to Allegro
- ADI shared component libraries with us
- We fear this may not work with our timeline

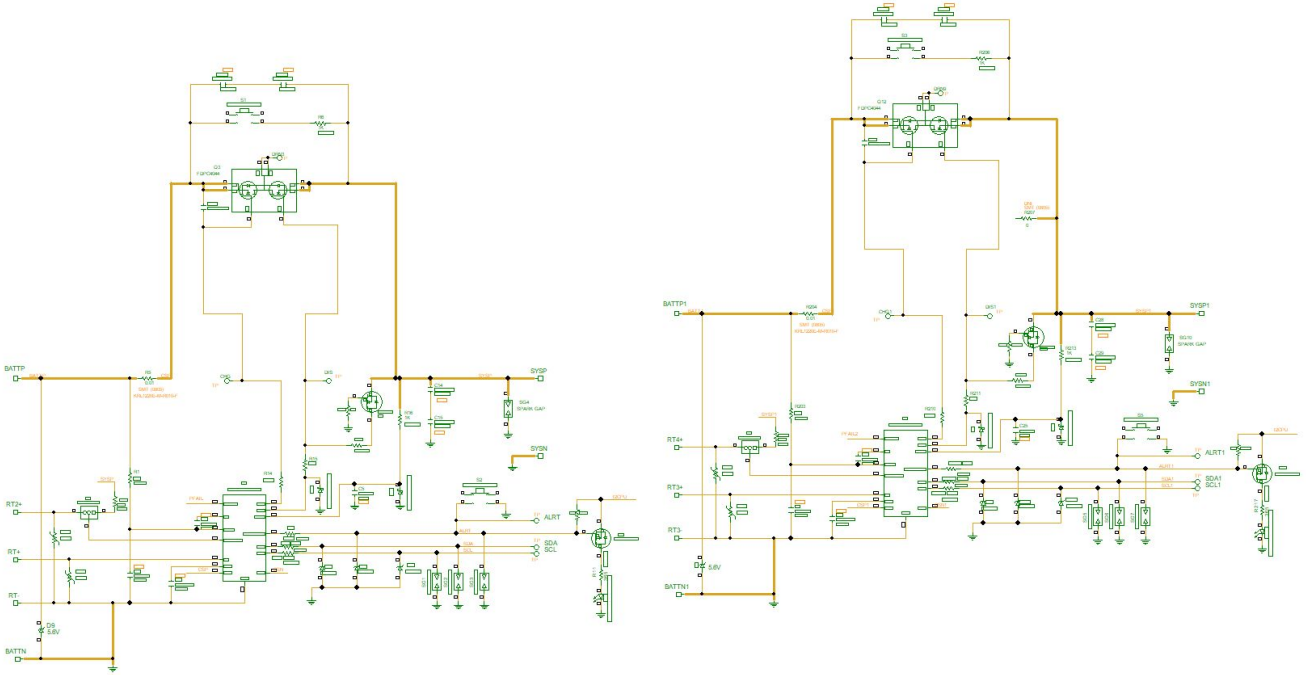


Progress: Completed Schematic in Allegro



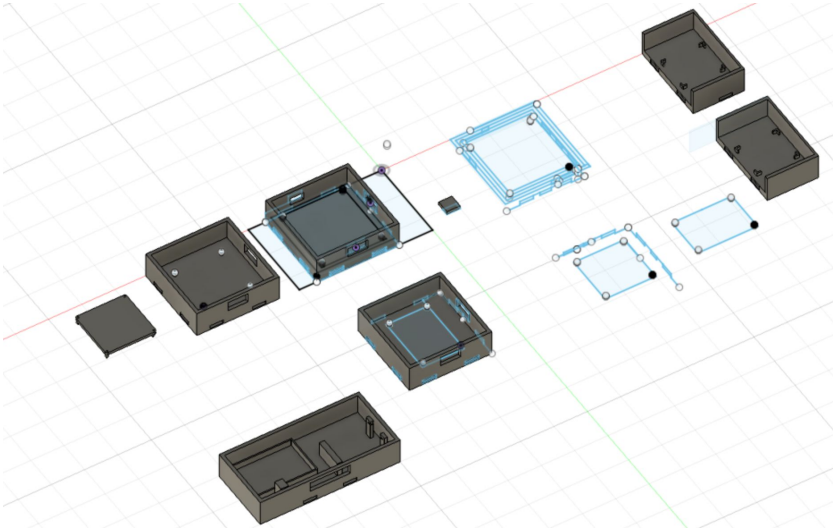


Progress: Completed Schematic in Allegro



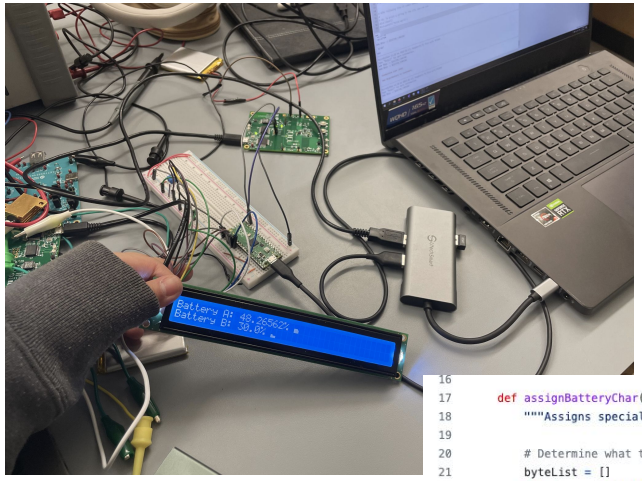
Progress: 3D Printed Enclosure for Demo Boards

- We are struggling to meet the timeline with our PCB Design
  - Time difference and physical distance makes it difficult to meet with ADI for help
  - Software and package differences make cooperation with ADI harder
  - Up to 3 week manufacture time
- Plan B: 3D Printed enclosure for working demo boards
  - Same functionality



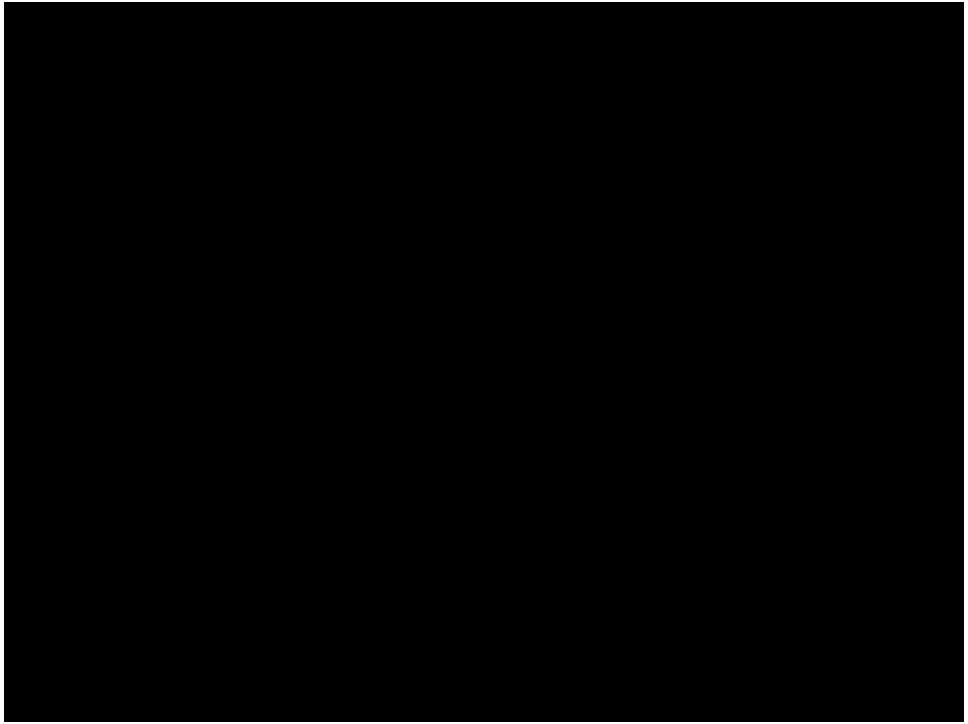
Progress: Pico and LCD Communication

- Programmed Pico to read effective battery levels from MAX17330 via I2C
- Pico communicates with LCD to display live readings
- Incorporation of basic battery graphics



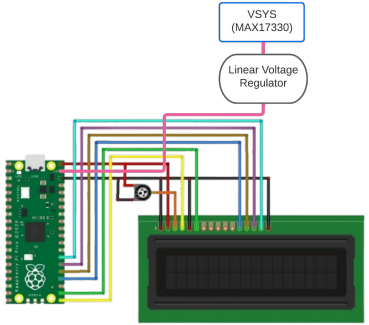

```
16
17      def assignBatteryChar(self, battPercentage, charAddr):
18          """Assigns special character to a given character address"""
19
20          # Determine what the character should look like
21          byteList = []
22          for pixLevel in range(8):
23              if battPercentage/(0.125*(pixLevel+1)) >= 1:
24                  byteList = [0x1f] + byteList
25              else:
26                  # Determined shift by knowing how many pixels need light
27                  # One pixel is 2.5% (0.025) and one line is 12.5% (0.125)
28                  shiftBy = int((battPercentage % 0.125)//0.025)
29
30                  # shiftBy is the number of pixels to turn on
31                  # then bitwise & to only get last 5 bits
32                  byteList = [(0b1111100000 >> shiftBy) & 0x1f] + byteList
33                  for _ in range(7-pixLevel):
34                      byteList = [0x00] + byteList
35                  break
36
37          # set as a custom character
38          self.lcd.custom_char(charAddr%8,bytearray(byteList))
39
```

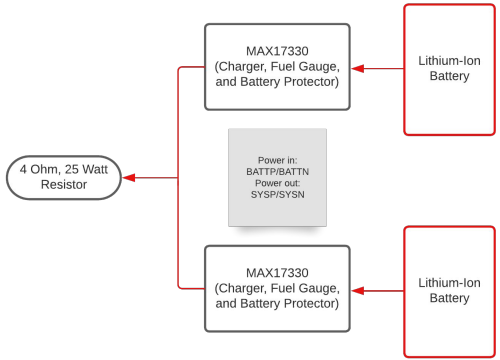
Progress: LCD Updating Battery Levels

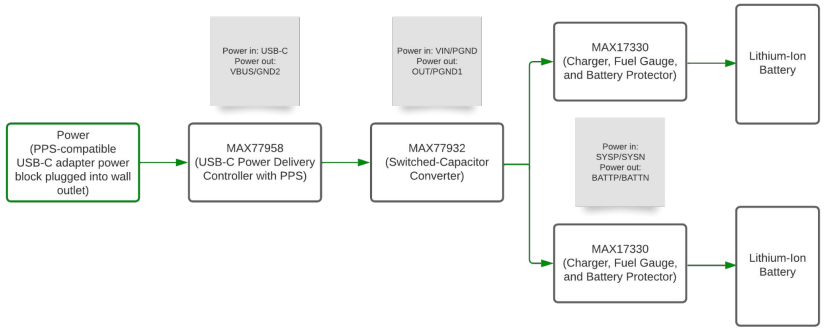


Progress: Full Physical Prototype Completed

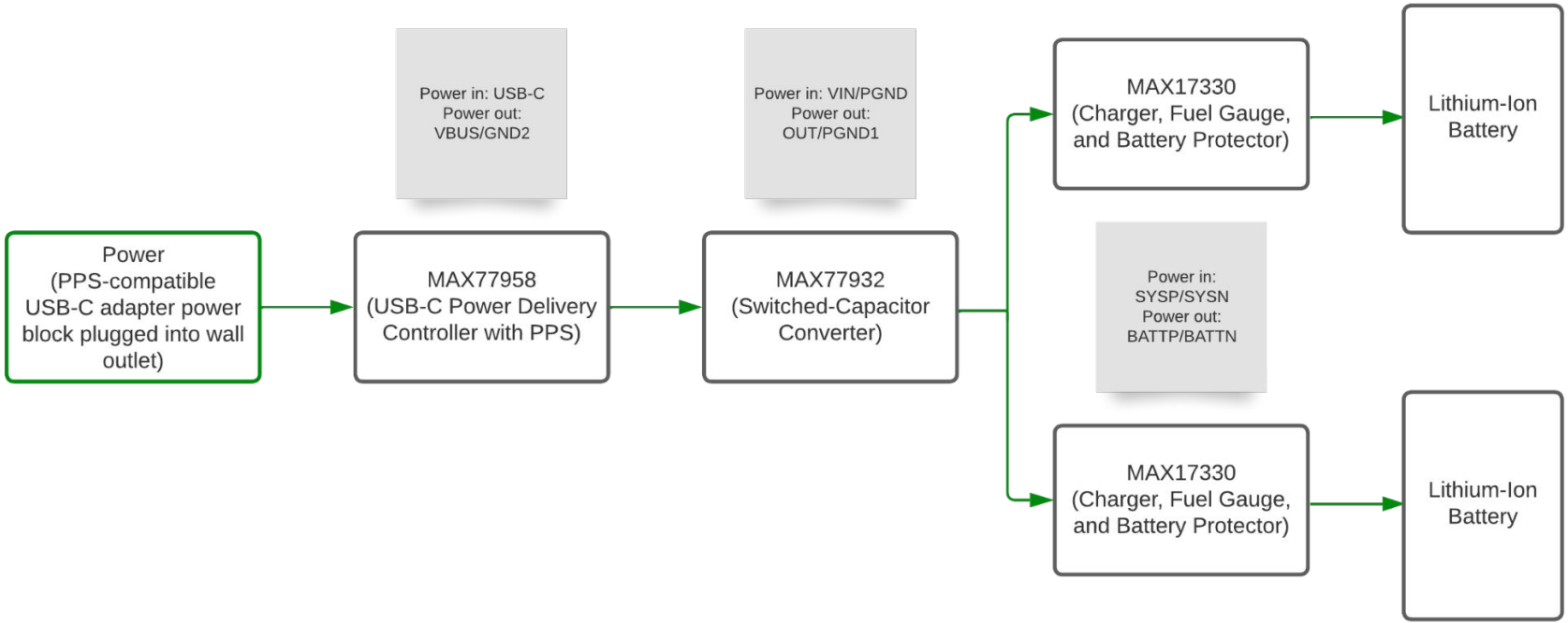
- Orientation for charging and discharging
- Connections between the evaluation (EV) boards
  - MAX77958
  - MAX17330
  - MAX77932
- Connections between EV boards, Pico, and LCD



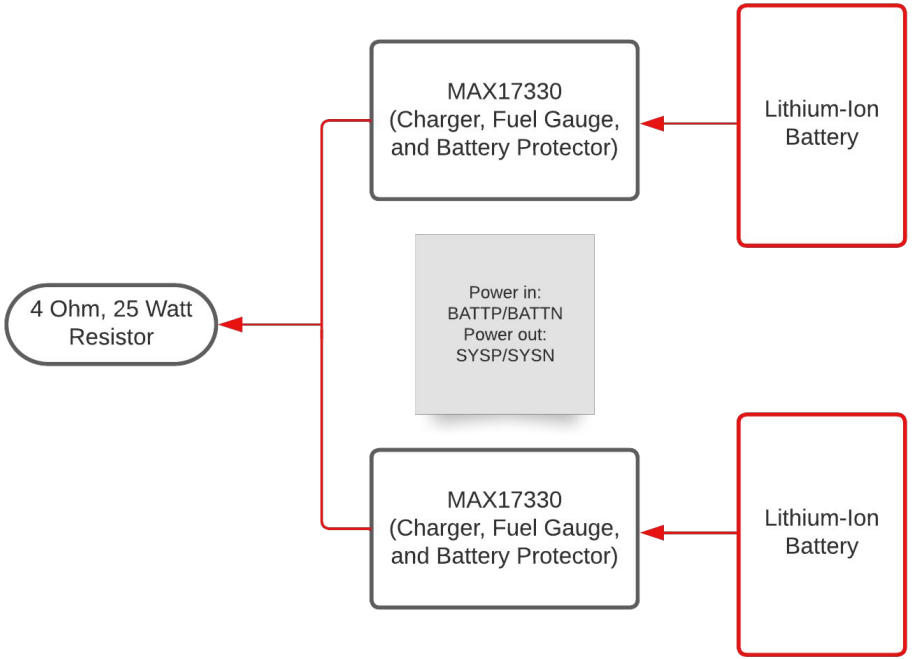




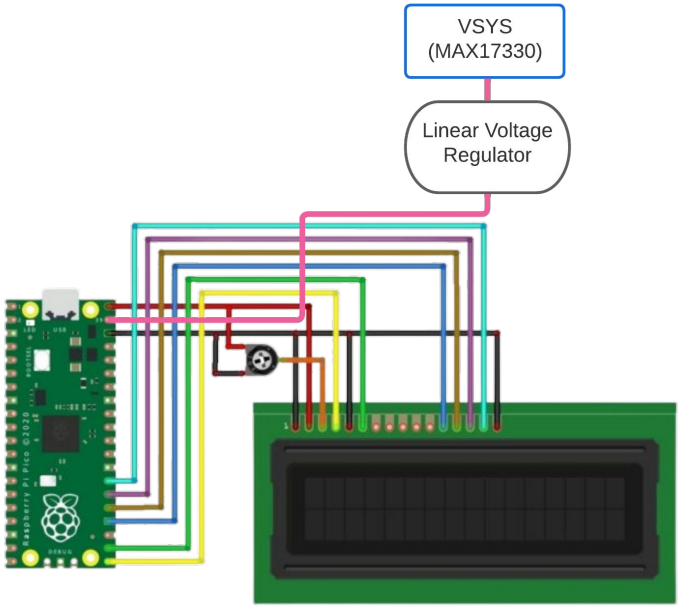
Progress: Charging Power Setup & Power Flow



Progress: Discharging Power Setup & Power Flow

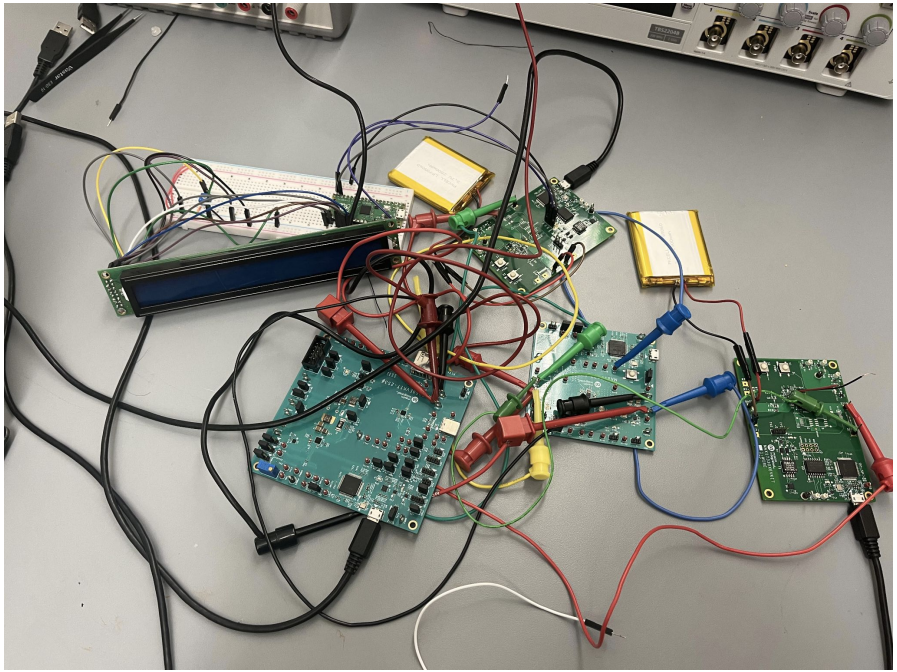


Progress: Raspberry Pi Pico Pin Connections with LCD



Raspberry Pi Pico Pin	LCD Pin	Potentiometer Pin
GPIO 16	RS	-
GPIO 17	E	-
GPIO 18-21	D4-D7	-
VBUS (5V)	VCC, LED+	VIN
GND	R/W, VSS, LED-	GND
-	VEE	VOUT

Physical Prototype



Progress: Pico Reading of Chip Registers

- Used I2C to read RepSOC register to find battery charge level (as %)
- Used datasheets to find important and relevant register addresses

RepSOC Register (006h) ←

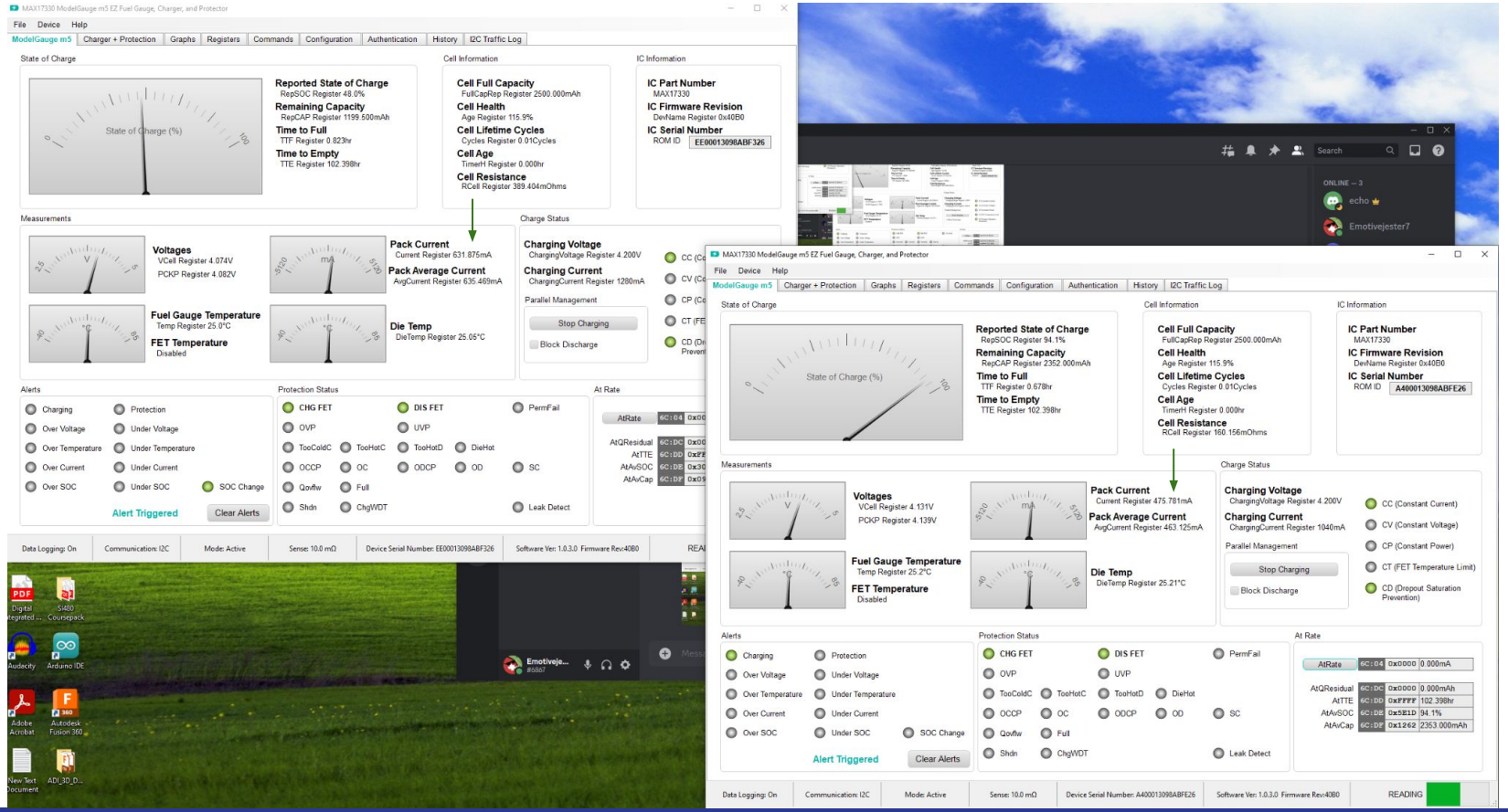
Register Type: Percentage

Nonvolatile Backup: None

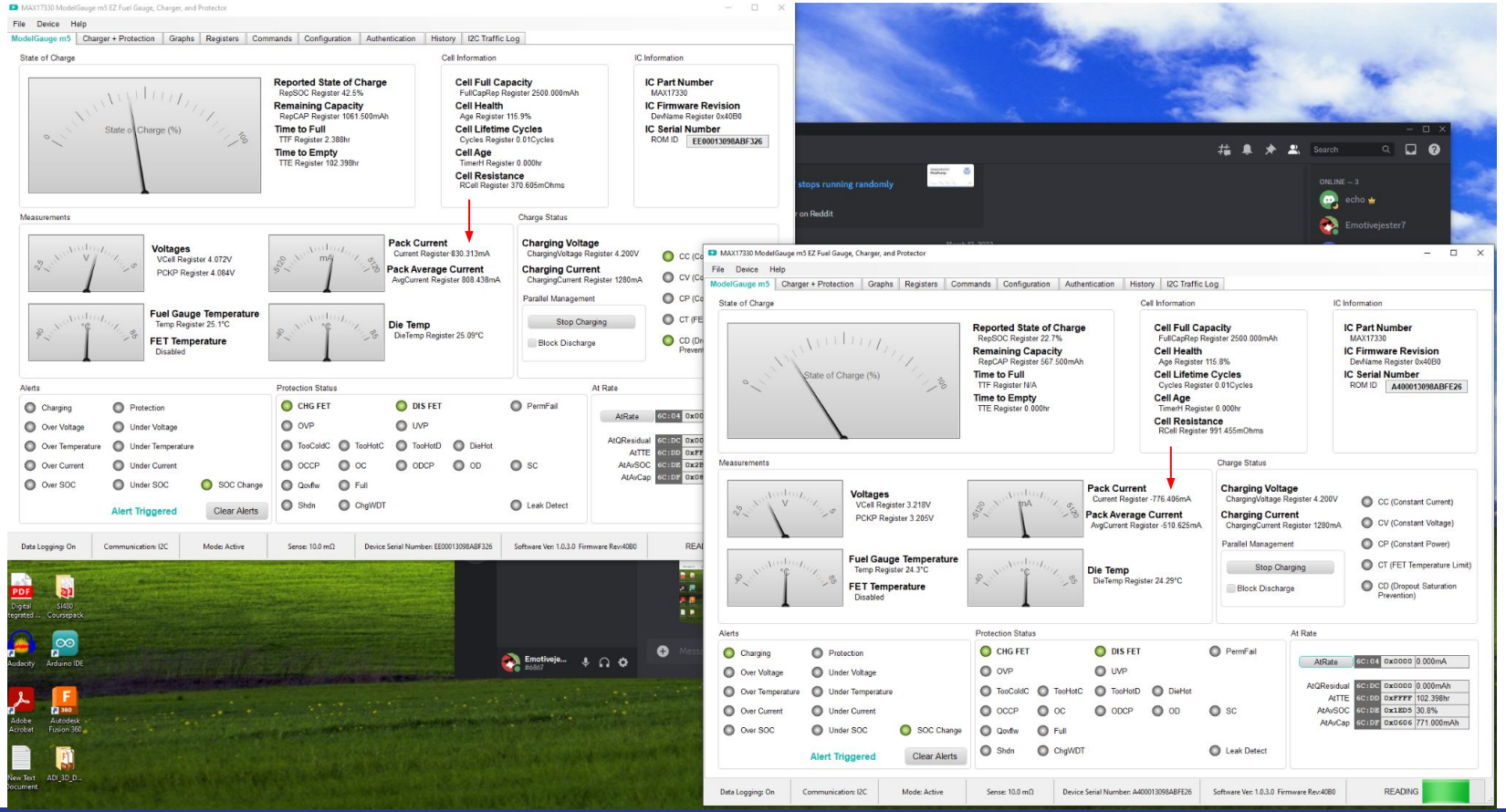
RepSOC is a filtered version of the AvSOC register that prevents large jumps in the reported value caused by changes in the application such as abrupt changes in load current. RepSOC corresponds to RepCap and *FullCapRep*. RepSOC

```
class readRegs:
    def __init__(self, i2c):
        """Initialize object"""
        self.i2c = i2c

    def readfrom_mem(self,addr,memaddr,nbytes,addrsize=8):
        """Reads defined register and returns list with 8bit-ints of size nbytes"""
        bstring = self.i2c.readfrom_mem(addr,memaddr,nbytes,addrsize=addrsize)
        return [bstring[i] for i in range(nbytes)]

    def readRepSOC(self,addr=0x36,memaddr=0x06,addrsize=8):
        data = self.readfrom_mem(addr=addr,memaddr=memaddr,nbytes=2,addrsize=addrsize)
        return ((data[1] << 8) + data[0])/256
```

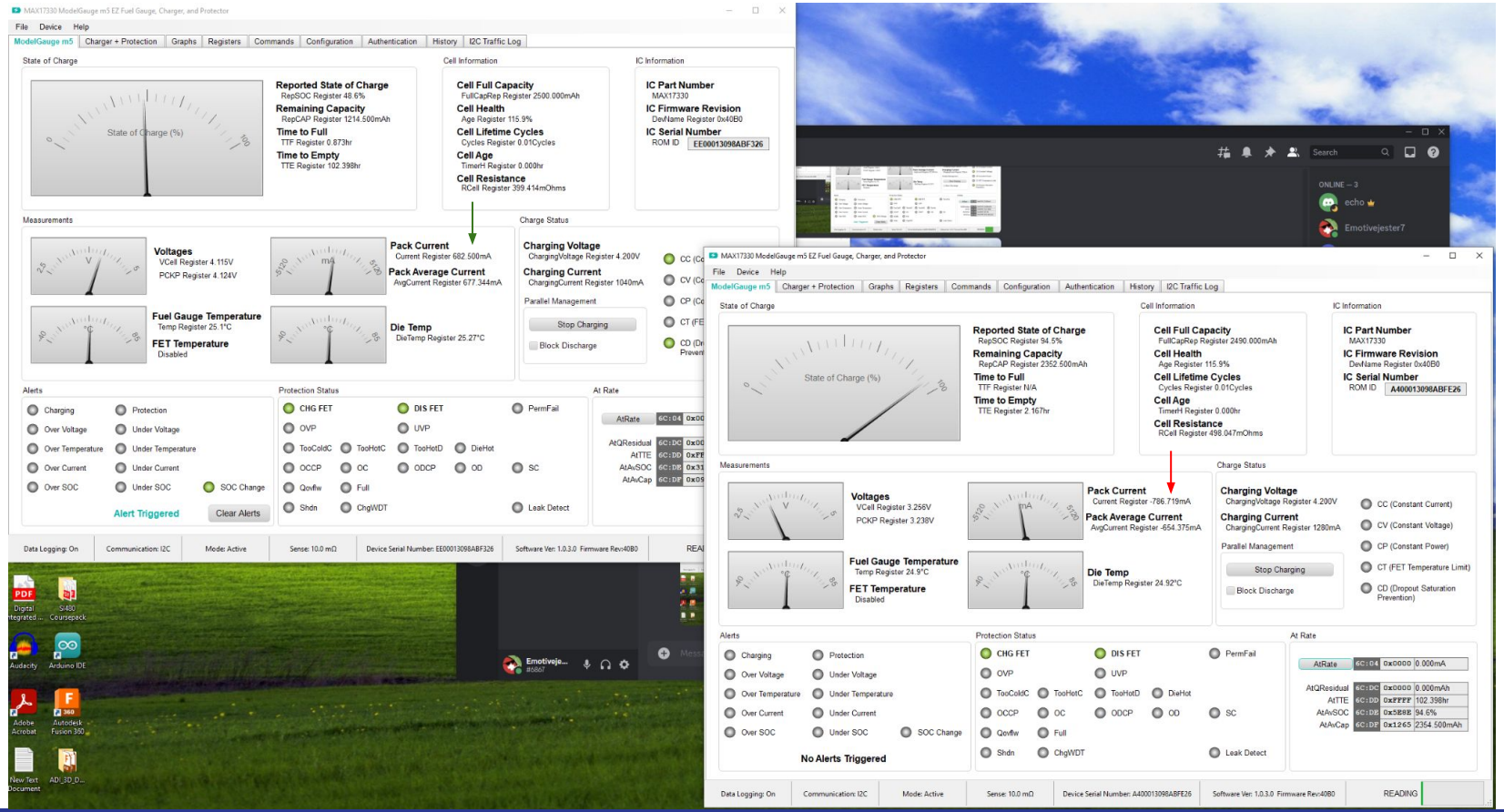
Progress: Verified Parallel Charging (via EV Kit GUI)



Progress: Verified Parallel Discharging

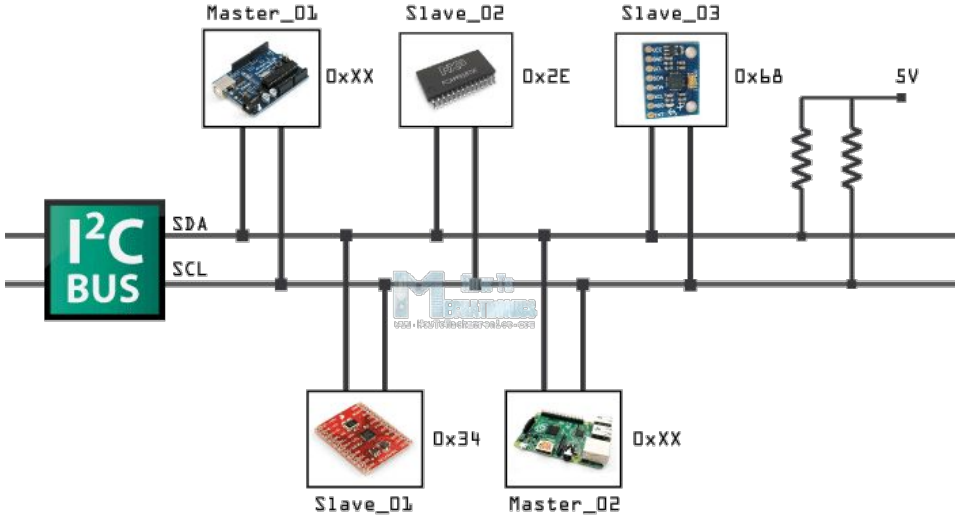


Progress: Verified One Charging, One Discharging

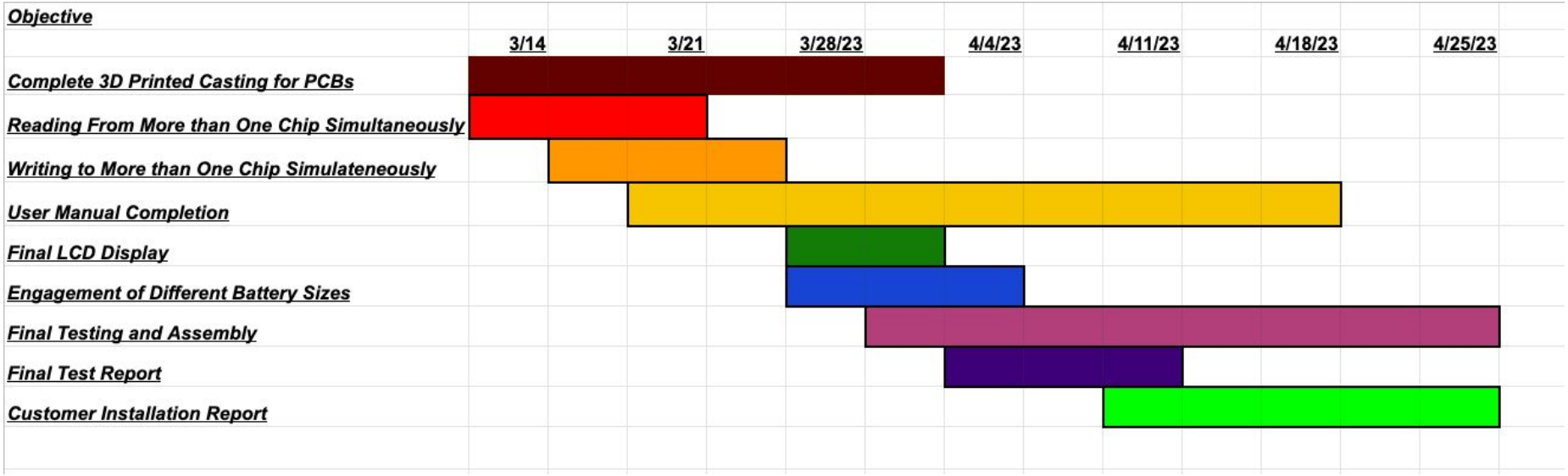


To be Completed

1. Reading from multiple chips
   a. I2C communication
2. Writing to multiple chips
   a. I2C communication
3. Developing 3D printed enclosure
   a. Demo-boards
   b. Pico
   c. Batteries
   d. LCD
4. Final LCD Display
   a. Display more values such as charging current and voltages



Updated Gantt Chart



Thank you!