

Muon Telescope Test

Sawaiz Syed

May 4, 2017

Contents

| | |
|--|-----------|
| 1 Components | 1 |
| 1.1 Scintillator Panel | 1 |
| 1.1.1 Theory | 2 |
| 1.1.2 Assembly | 2 |
| 1.2 MPPC Sensor | 3 |
| 1.2.1 Hardware | 3 |
| 1.2.2 Cable | 4 |
| 1.3 MPPC High Voltage Supply | 5 |
| 1.3.1 Hardware | 6 |
| 1.3.2 Software | 8 |
| 1.4 MPPC Interface | 23 |
| 1.4.1 Hardware | 23 |
| 1.4.2 Software | 25 |
| 1.5 Frame | 26 |
| 1.5.1 Construction | 26 |
| 2 Assembly | 28 |
| 3 Software | 36 |
| 3.1 Structure | 36 |
| 3.1.1 Technologies | 36 |
| 3.1.2 Architecture | 38 |
| 3.1.3 Operation | 38 |

Abstract

The assembly, software, and operation of a three paddle plastic scintillator detector.

Chapter 1

Components

1.1 Scintillator Panel

Large scintillator panel with embedded fiber loop to channel light to attached photo-sensor. The tile has a loop pattern with a single fiber to attempt to gather light from as much of the effective area of scintillator and minimize losses from tight curves and open fiber ends. The fiber groove follows along the top surface and comes out perpendicular to end face. A rendering of a assembled panel is shown in Figure 1.1.



Figure 1.1: Fully assembled Scintilator panel.

1.1.1 Theory

The scintillator panel is a doped plastic (available from many suppliers Eljen being the one used in our assemblies) that emits light in response to being struck by an electron, alpha particle, ion, or high energy photon. The light emitted (blue in our case) is then expected to hit the embedded fiber that is able to absorb that light, and retransmit at the lower green frequency based on the characteristics of the fiber. This green light is captured and guided by the fiber to the end of finger where it can be read out. Each event is expected to produce about 30 photons, with a lower number reaching the sensor placed on the fiber side. The outer surce has coatings applied in order to shild from external light and attempt to maximise efficiency by reflecting internal light. A exploded view of all the componets is showin in Figure 1.2.

1.1.2 Assembly

Currently the parts must be hand assembled although further production may yield an injection moulding scheme. The following lists the procedure to construct a single plate.

Milling

The fingers are cut from 10mm thick scintillator and are 145 mm x 145 mm. The groove is cut with a ball mill under water flow to prevent deformation of the plastic due to heat. The milled sides are then polished (Novus plastic polish works well) and then are washed with alcohol.

Gluing

The optical fibers are cut to length (score and breaking) and the end is polished. It is placed into the groove with a two part optical cement. This binds the fiber to the milled plastic providing a optical bond. Forcing the fiber to match the contour milled.

Covering

The fingers are then covered with either degreased aluminum foil and electrical tape or a coating of reflective spray followed by a dip inside rubberized coating (successful tests have been done with Spaz Stix Ultimate Mirror Chrome and Plasti Dip) providing a more even surface finish and reduced labour. This covering shields external light, and provides higher efficiency by capturing stray photons by reflecting them back towards the fiber optic.

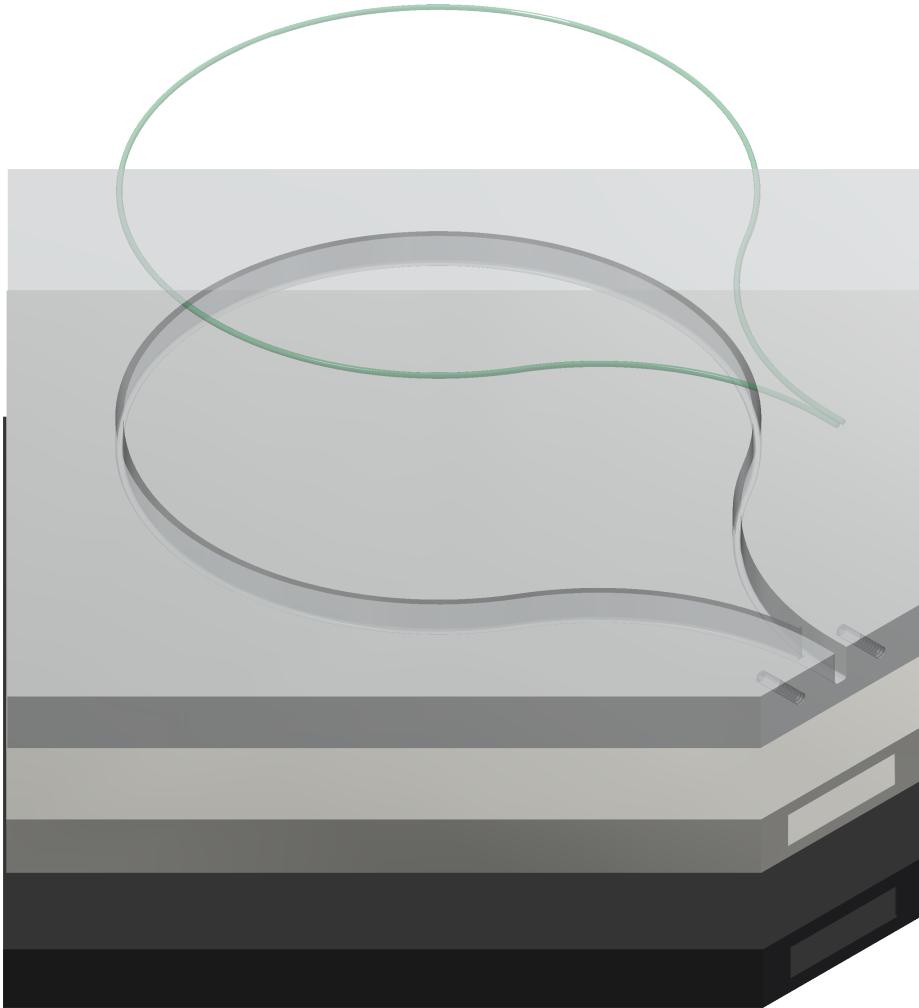


Figure 1.2: Exploded view showing coatings and fiber.

1.2 MPPC Sensor

The board stack has a MPPC, preamp, temperature sensor, and LED. Having the preamp on board, just millimeters from the sensor allows excellent signal integrity, even in high noise environments. Also having the temperature sensor on the sensor, allows for fine adjustment due to local temperature fluctuations, as the gain is dependent on temperature and bias voltage, a constant gain can be maintained by changing the bias in response to temperature.

1.2.1 Hardware

This board houses a surface mount 2.0 mm MPPC by Hamamatsu. Specifically S13360-20**VE series. The signal is amplified with a TI OPA846 op-amp. There is a LED as well as a 100K NTC thermistor. The boards are all 7 mm tall, 25 mm

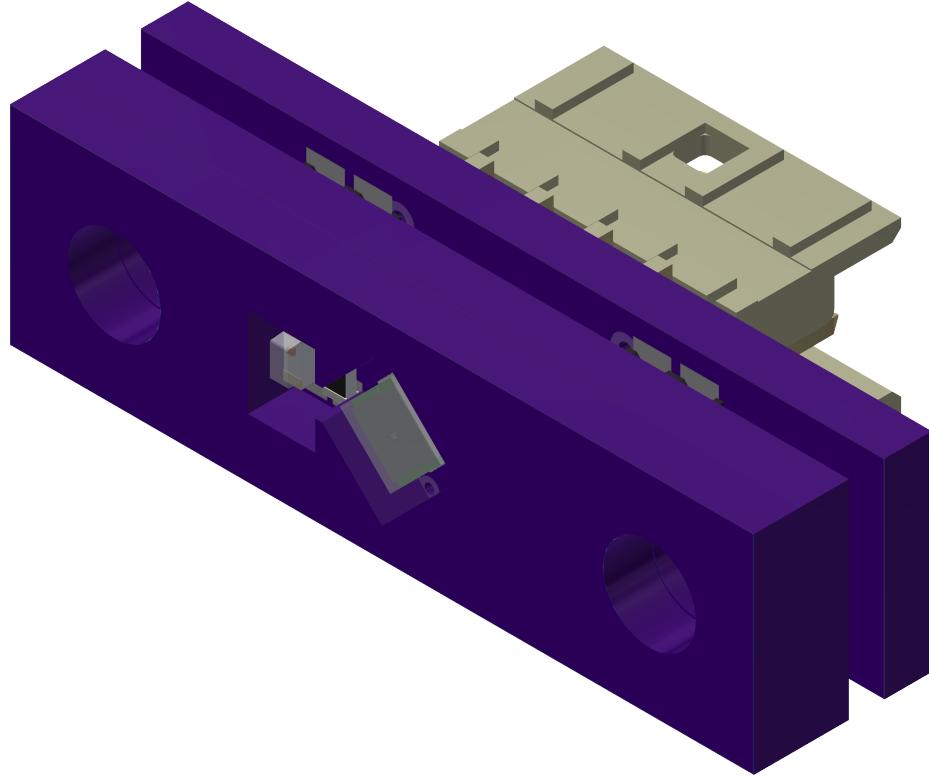


Figure 1.3: Assembled MPPC sensor board stack.

wide and approximately 6.4 mm in depth without the pins. The spacer board blocks out excess light from reaching the MPPC, and provides isolation from the LED forcing it to go through the scintillator. It is attached with two 4-40 screws that can go directly into scintillator or a mounting block.

1.2.2 Cable

There are eight pins attached to back of the final board. Those pins break out all signals needed. There is a corner mark on the PCB that lies in the top left corner. On single ended signals, *Sig-* is connected to *GND*.

The cable used is a 28AWG CAT6 cable, Monoprice's Slimrun line is an option. This provides 8 conductors that are twisted pairs, along with a 8P8C connector. Wiring standard used is the following positions.

| MPPC High Voltage BOM | | | | | | | | |
|-----------------------|------------------------|----------------------------------|--------------------|-----------------|----------|------------|-----------|--|
| Ref-Des | Function | Description | P/N | Digikey P/N | QTY | Unit Price | Ext Price | |
| C9, C10 | Op-amp Power Filter | CAP CER 1UF 25V X5R 0402 | GRM155R61E105MA12D | 490-10018-1-ND | 1 | 0.03250 | 0.03 | |
| C11 | HV Filter | CAP CER 0.047UF 50V X7R 0402 | GRM155R71H473KE14D | 490-10702-1-ND | 1 | 0.01320 | 0.01 | |
| C7 | Decoupling | CAP CER 1UF 10V X5R 0603 | CL10A105KP8NNNC | 1276-1182-1-ND | 1 | 0.01400 | 0.01 | |
| C8 | Decoupling | CAP CER 10UF 10V X5R 0603 | GRM188R61A106ME69D | 490-10475-1-ND | 1 | 0.06560 | 0.07 | |
| D1 | MPPC | 2.0mmx2.0mm MPPC | S13360-2050VE | S13360-2050VE | 1 | 22.00000 | 22.00 | |
| D2 | Led | LED GREEN CLEAR 0603 R/A SMD | LTST-S270GKT | 160-1475-1-ND | 1 | 0.11220 | 0.11 | |
| J1,J2 | Spring Connector | BATTERY CONN 1.2H 1.6MM DR 4CKT | 788640001 | WM11203CT-ND | 2 | 0.25470 | 0.51 | |
| R9, R10 | Op-amp Power Filter | RES SMD 10 OHM 5% 1/16W 0402 | RC0402JR-0710RL | 311-10JRCT-ND | 2 | 0.00470 | 0.01 | |
| R2 | NTC Therm | THERMISTOR NTC 100KOHM 0.5% 0603 | NCP18WF104D03RB | 490-11811-1-ND | 1 | 0.31270 | 0.31 | |
| R11, R12, R14 | 51 ohm Term | RES SMD 51 OHM 5% 1/16W 0402 | RC0402JR-0751RL | 311-51JRCT-ND | 3 | 0.00470 | 0.01 | |
| R13, R15 | Op-amp Gain/ HV Filter | RES SMD 1K OHM 0.1% 1/16W 0402 | RT0402BRD071KL | YAG1386CT-ND | 2 | 0.12760 | 0.26 | |
| U1 | Op-Amp | IC OPAMP VFB 1.75GHZ SOT23-5 | OPA846IDBVT | 296-14776-1-ND | 1 | 3.74460 | 3.74 | |
| J3 | Output Header | MINITEK | 98424-F52-08ALF | 609-5175-1-ND | 1 | 1.04000 | 1.04 | |
| | | | | PCB Manufacture | 3 | 1.35000 | 4.05 | |
| | | | | Assembly | 1 | 5.38000 | 5.38 | |
| | | | Testing (Labour) | 0.05 | 30.00000 | 1.50 | | |
| | | | | | Subtotal | | 39.05 | |

Table 1.1: Bill of Materials, all prices in USD.

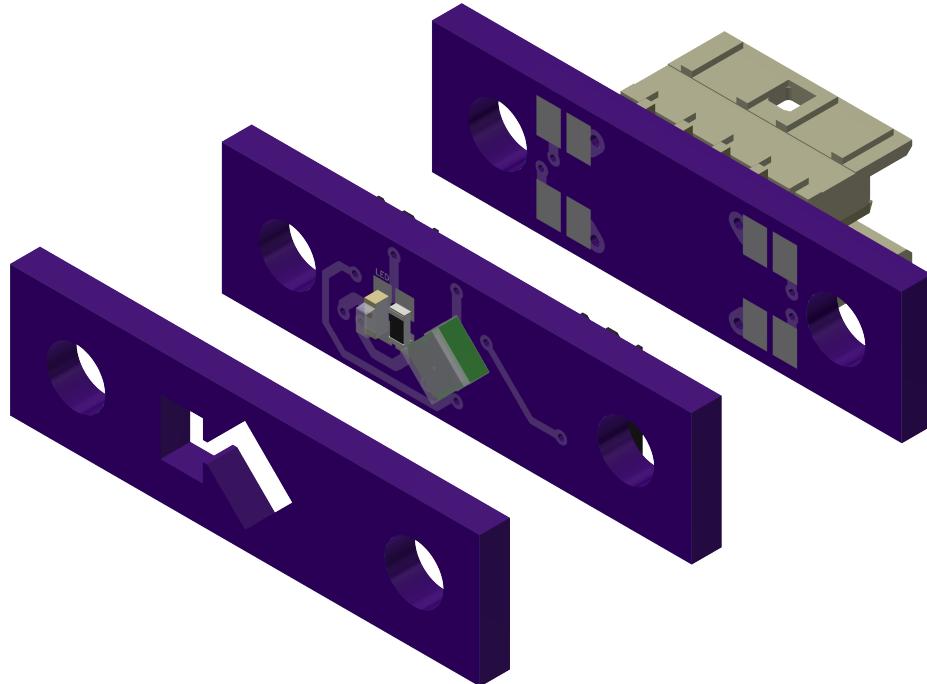


Figure 1.4: Exploded view of MPPC sensor side.

1.3 MPPC High Voltage Supply

Surface mount module that provides 52.5 V to 57.5 V in 20 mV increments. Also contains a 2 input 24bit ADC to allow for closed loop control of the voltage, and read in a thermistor mounted near the sensor. Rendering of a fully assembled module shown in Figure 1.9.

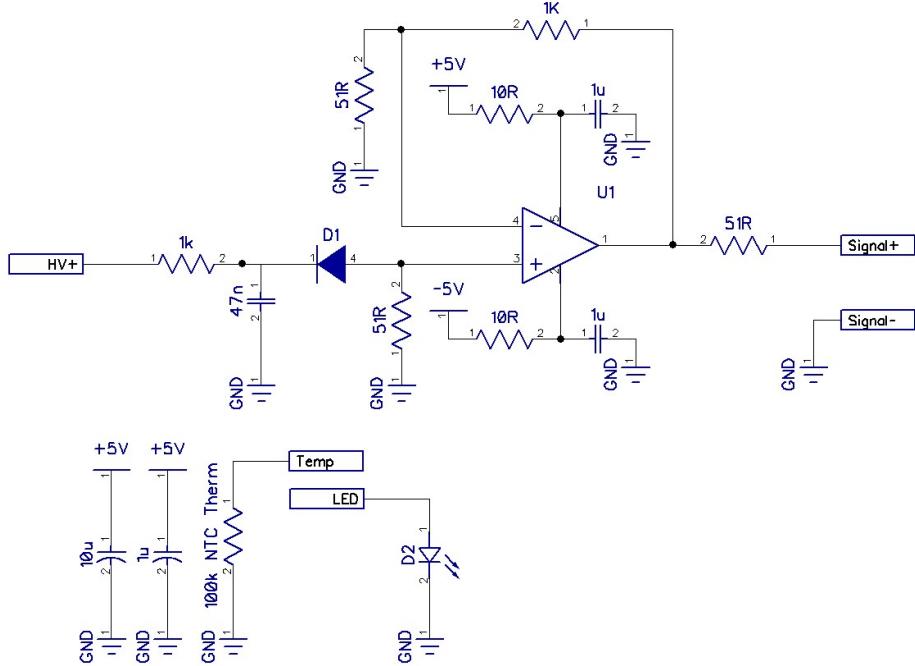


Figure 1.5: Schematic of the preamp, D1 is the MPPC.

| Sensor Interface Cable BOM | | | | | | | |
|----------------------------|-----------------------|---------------------------------------|-------------|-------------------|------|------------|-----------|
| Ref-Des | Function | Description | P/N | Supplier P/N | QTY | Unit Price | Ext Price |
| J3-F | Output Header Jack | CONN IDC SOCKET 8POS 2MM GOLD | 89361-708LF | 609-3140-ND | 1 | 1.08800 | 1.09 |
| W1 | 3 foot Ethernet Cable | Cat6 28AWG UTP Ethernet Network Cable | 14812 | 14808 | 0.5 | 2.29000 | 1.16 |
| | | | | Assembly (Labour) | 0.1 | 30.00000 | 3.00 |
| | | | | Testing (Labour) | 0.05 | 30.00000 | 1.50 |
| | | | | | | Subtotal | 6.05 |

Table 1.2: Bill of Materials, all prices in USD.

1.3.1 Hardware

The module uses a MAX1932 APD bias voltage boost converter and NAU7802SGI 24bit Analog to Digital converter (ADC) for reading temperature and output voltage. The complete bill of materials including assembly and testing is detailed in Table 1.5.

High Voltage

The module uses a MAX1932 APD bias voltage boost converter to create and maintain the output voltage. The voltage range, and step size is dictated with setting R2, R3, and R4. The minimum and maximum voltage set using the equations for on page 7 of the datasheet, and there are 8 bits (256) intermediate values that are possible the corresponding schematic is shown in Figure 1.11. The high switching portion was designed with general SMPS design requirements (short high current loops, separated ground planes, EMI shielding) providing a low ripple output ($< 5 \text{ mV}_{pp}$ ripple) and no significant back EMF.

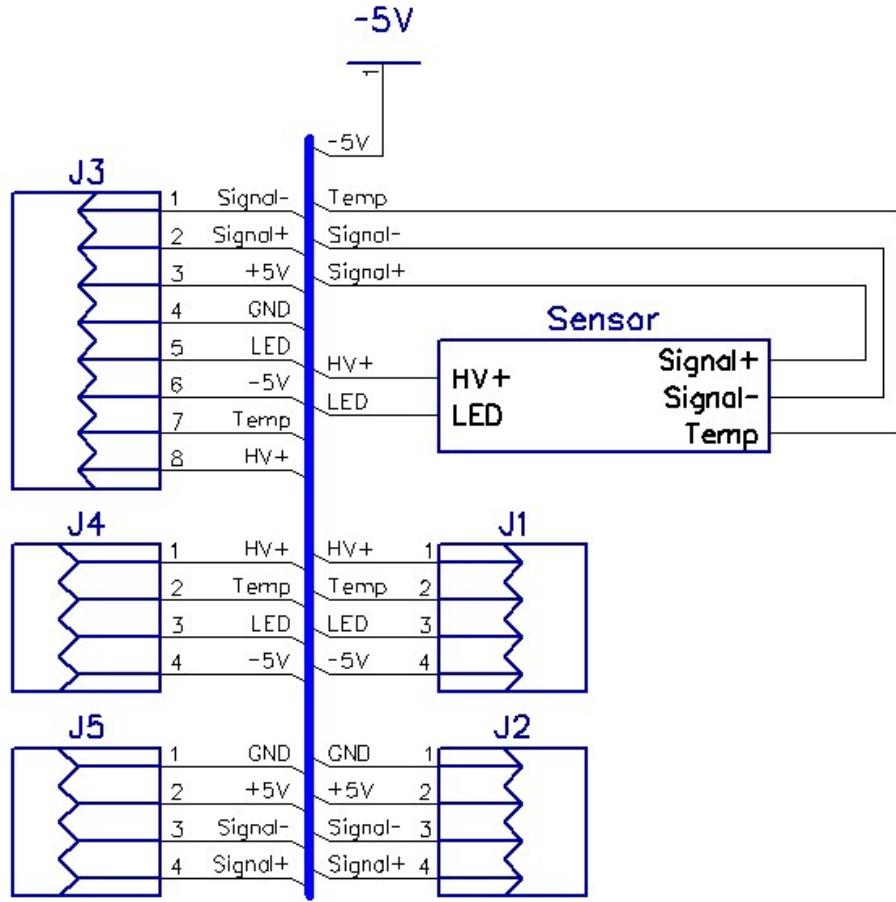


Figure 1.6: The interconnects between the boards as well as the header in the back.

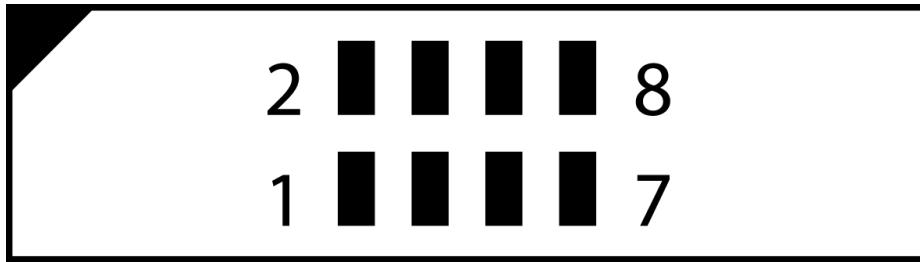


Figure 1.7: Pin numbering for Table 1.6.

ADC

The analog to digital converter is a one designed for high precision and low sampling rate. The NAU7802SGI is marketed at the load sensing market, but the high precision and low sampling rate lends itself well to the closed loop control and monitoring application. The high voltage is divided by 50 and the thermistor is put across a equivalent 100K static resistor to go along with he 100K NTC found on the MPPC

| Sensor Interface Cable Connections | | |
|------------------------------------|--------------|--------|
| Pin | Function | Type |
| 1 | Signal - | Output |
| 2 | Signal + | Output |
| 3 | $+V_S$ | Power |
| 4 | Ground | Power |
| 5 | LED | Input |
| 6 | $-V_S$ | Power |
| 7 | High Voltage | Power |
| 8 | Thermistor | Output |

Table 1.3: Pinout on the back side with pins facing you and the cut corner on top left.

| RJ-45 Pin | Cable Colour | Signal | Abv |
|-----------|--------------|--------------|------|
| MD0+ | White Orange | High Voltage | HV+ |
| MD0- | Orange | Ground | GND |
| MD1+ | White Green | Test | LED |
| MD1- | Blue | Thermistor | TEMP |
| MD2+ | White Blue | +Voltage | +Vcc |
| MD2- | Green | -Voltage | -Vcc |
| MD3+ | White Brown | Signal+ | Sig+ |
| MD3- | Brown | Signal- | Sig- |

Table 1.4: Pinout on the back side with pins facing you and the cut corner on top left.

sensor module to act as a voltage divider, see Figure 1.12 for details.

1.3.2 Software

Each module has two buses that run to it, I²C (pronounced I-two-C) and SPI. SPI requires data, clock, and a chip-select meaning each device on the bus will need an independent chip select. I²C devices have a fixed address therefore it is not possible to have multiple devices on the same bus. This can be remedied by placing multiple devices in front of a I²C multiplexer. Table 1.6 lists their functions.

High Voltage

The SPI bus is connected directly to the MAX1932. Chip select is active low and data is sent MSB (most significant bit first). Sending bytes to it sets the voltage. Setting them to 0x00 turns off the DC-DC converter portion. As values increase to 0xFF (127), the output voltage falls, linearly. Example code for Arduino that cycles through all possible values is shown in Listing 1.1.

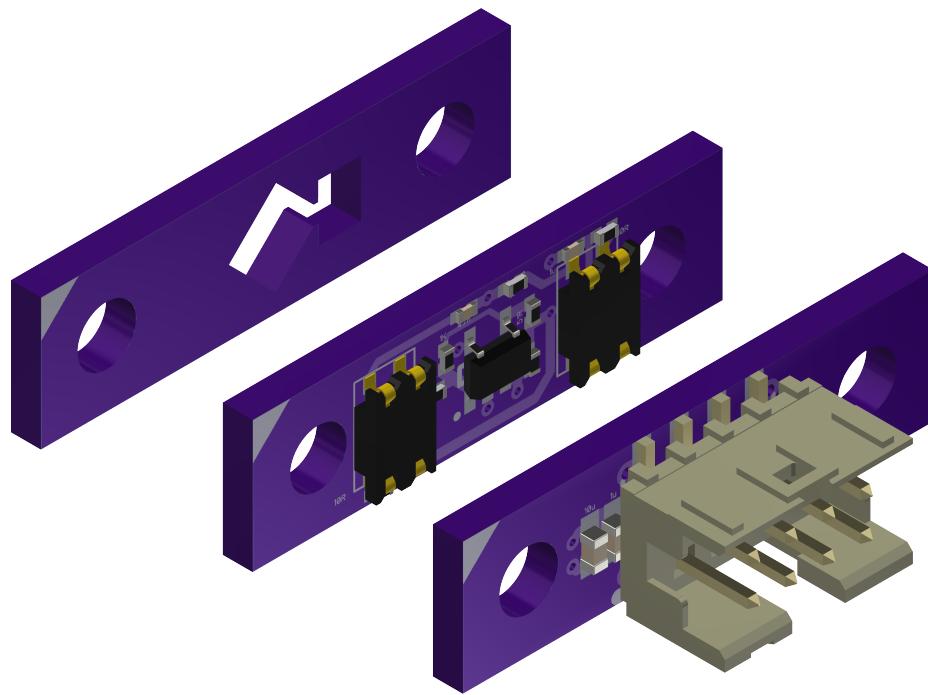


Figure 1.8: Exploded view of connector side.

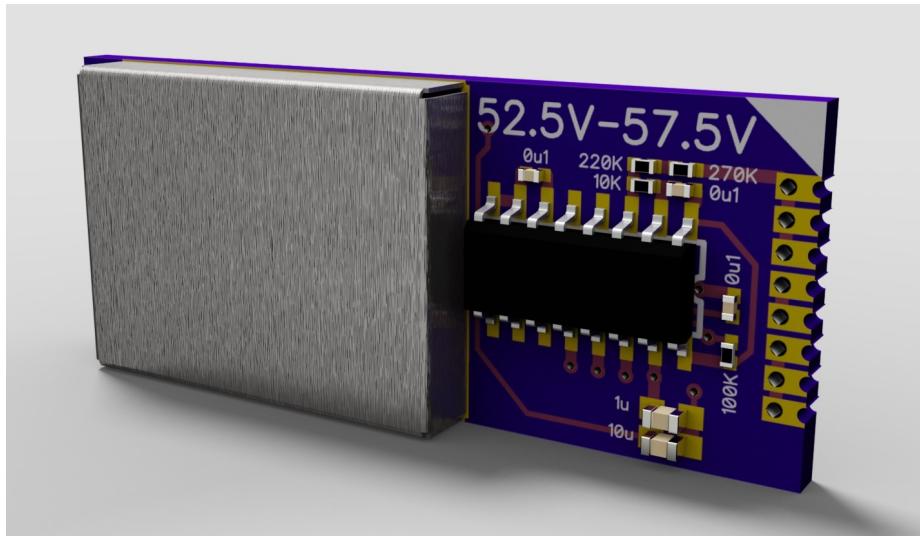


Figure 1.9: Board rendering with EMI shield.

```
1 //Example Arduino code that loops through all voltages.  
2 //SCLK --> Pin 13 for Uno or Duemilanove  
3 //DIN --> Pin 11 for Uno or Duemilanove  
4  
5 #include <SPI.h>  
#define CS 2 //Any pin
```

| MPPC High Voltage BOM | | | | | | | | |
|-----------------------|------------------------|--|--------------------|----------------------|------|------|---------|-----------|
| Ref-Des | Function | Description | P/N | Digikey P/N | QTY | Unit | Price | Ext Price |
| C1 | Input Decoupling | 1F 16V Ceramic Capacitor X7R 1206 | 160R18W105KV4E | 709-1068-1-ND | 1 | | 0.0890 | 0.09 |
| C2 | Boost Capacitor | 0.047F 500V Ceramic Capacitor X7R 1206 | C1206C473KCRACTU | 399-6743-1-ND | 1 | | 0.1300 | 0.13 |
| C3 | Pre Filter Capacitor | 0.10F 250V Ceramic Capacitor X7R 1206 | C1206C104KARACTU | 399-4674-1-ND | 1 | | 0.1485 | 0.15 |
| C4 | Post Filter Capacitor | CAP CER 1UF 100V XTR 1206 | 12061C105KA72A | 478-6226-1-ND | 1 | | 0.1570 | 0.16 |
| C5 | Compensation Capacitor | CAP CER 0.22UF 16V Y5V 0402 | CL05F224Z05NNNC | 1276-1060-1-ND | 1 | | 0.0112 | 0.01 |
| D1 | Boost Diode | DIODE GEN PURP 100V 200MA SOD80 | LL4148 | LL4148FSCT-ND | 1 | | 0.0420 | 0.04 |
| L1 | Boost Inductor | FIXED IND 150UH 320MA 2.4 OHM | SRN4018-151M | SRN4018-151MCT-ND | 1 | | 0.2400 | 0.24 |
| L2 | Smoothing Inductor | FIXED IND 330UH 90MA 15.99 OHM | CBC2518T331K | 587-3059-1-ND | 1 | | 0.1400 | 0.14 |
| Q1 | Boost Switch | MOSFET N-CH 100V 0.17A SOT-23 | BSS123L | BSS123LCT-ND | 1 | | 0.1032 | 0.10 |
| R1 | Current Sens Resistor | RES SMD 806 OHM 0.1% 1/16W 0402 | RT0402BRD07806RL | YAG1481CT-ND | 1 | | 0.1276 | 0.13 |
| R2, R6 | Adjustment Span/Temp | RES SMD 100K OHM 0.1% 1/16W 0402 | RT0402BRD07100KL | YAG1343CT-ND | 2 | | 0.1276 | 0.26 |
| R3 | Minimum Voltage Res | RES SMD 2.37KOHM 0.1% 1/16W 0402 | CPF0402B2K37E1 | A102770CT-ND | 1 | | 0.2082 | 0.21 |
| R4 | DAC Resistor | RES SMD 24.9KOHM 0.1% 1/16W 0402 | RT0402BRD0724K9L | YAG1393CT-ND | 1 | | 0.1276 | 0.13 |
| R5 | Compensation Res | RES SMD 20 OHM 0.1% 1/16W 0402 | CPF0402B20RE1 | A102706CT-ND | 1 | | 0.1973 | 0.20 |
| U1 | Boost Controller | IC SUPPLY BIAS APD 12-TQFN | MAX1932ETC+T | MAX1932ETC+TCT-ND | 1 | | 7.2828 | 7.28 |
| U2 | ADC | IC ADC 24BIT I2C/SRL 16-SOP | NAU7802SGI | NAU7802SGI-ND | 1 | | 1.3442 | 1.34 |
| C6 | 0.1uF | 0.10F 10V Ceramic Capacitor X5R 0402 | CL05A104KPF5NNNC | 1276-1022-1-ND | 2 | | 0.0048 | 0.01 |
| C7 | 1uF | 1F 6.3V Ceramic Capacitor X6S 0402 | GRM155C80J105KE15D | 490-6281-1-ND | 1 | | 0.0055 | 0.01 |
| C8 | 10uF | 10F 6.3V Ceramic Capacitor X5R 0603 | GRM155R60G106ME44D | 490-10693-1-ND | 1 | | 0.0380 | 0.04 |
| S1 | RF Shield | RFI SHIELD CAN 20X15X3MM | S02-20150300 | 952-2632-ND | 1 | | 3.5300 | 3.53 |
| R7 | 220k 1% | RES SMD 220K OHM 1% 1/16W 0402 | RC0402FR-07220KL | 311-220KLRCT-ND | 1 | | 0.0054 | 0.01 |
| R9 | 270k 1% | RES SMD 270K OHM 1% 1/16W 0402 | RC0402FR-07270KL | 311-270KLRCT-ND | 1 | | 0.0054 | 0.01 |
| R8 | 10k 0.1% | RES SMD 10K OHM 1% 1/16W 0402 | RC0402FR-0710KL | 311-10.0KLRCT-ND | 1 | | 0.0054 | 0.01 |
| | | | | PCB Manufacture | 1 | | 1.5333 | 1.53 |
| | | | | Assembly | 1 | | 9.3000 | 9.30 |
| | | | | Testing (Labour) | 0.05 | | 30.0000 | 1.50 |
| | | | | Calibration (Labour) | 0.05 | | 30.0000 | 1.50 |
| | | | | Subtotal | | | | 28.03 |

Table 1.5: Bill of materials All prices in USD.

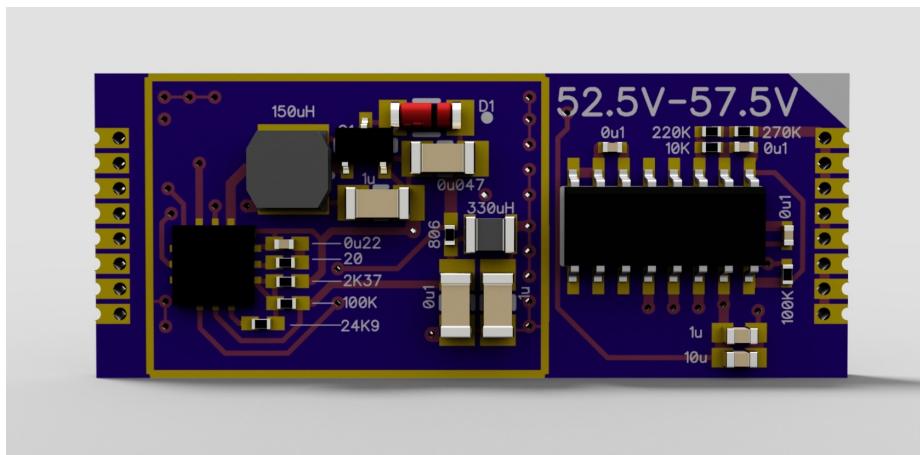


Figure 1.10: EMI shield removed showing switching region.

```
7 void setup (){
9     Serial.begin(9600);
11    pinMode(CS, OUTPUT);
13    digitalWrite(CS, HIGH);
15    SPI.begin();
17    SPI.setBitOrder(MSBFIRST);
19 }

void loop (){
21    for (int i = 0 ; i < 256 ; i++){
23        digitalWrite(CS, LOW);
25        SPI.transfer(i);
27    }
29 }
```

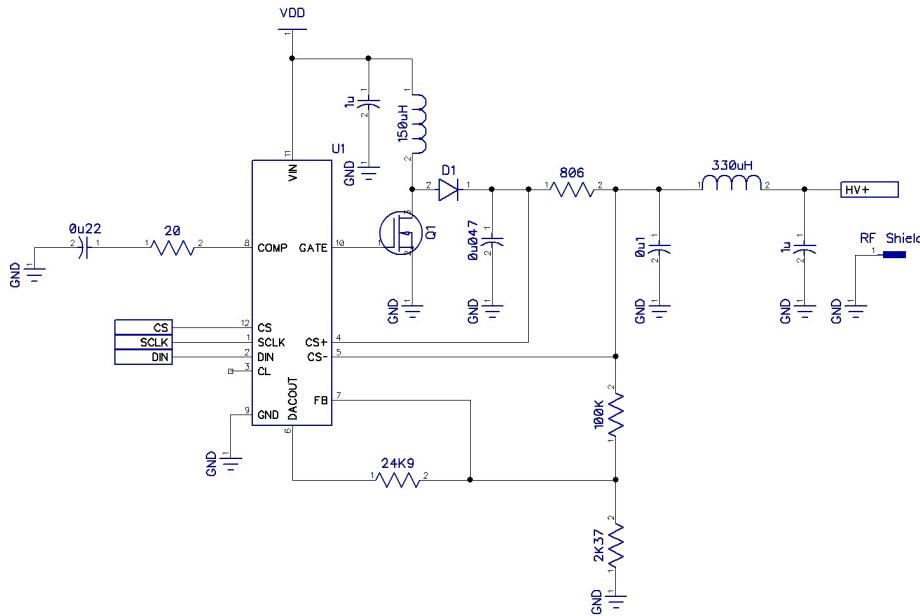


Figure 1.11: Schematic for high voltage boost converter.

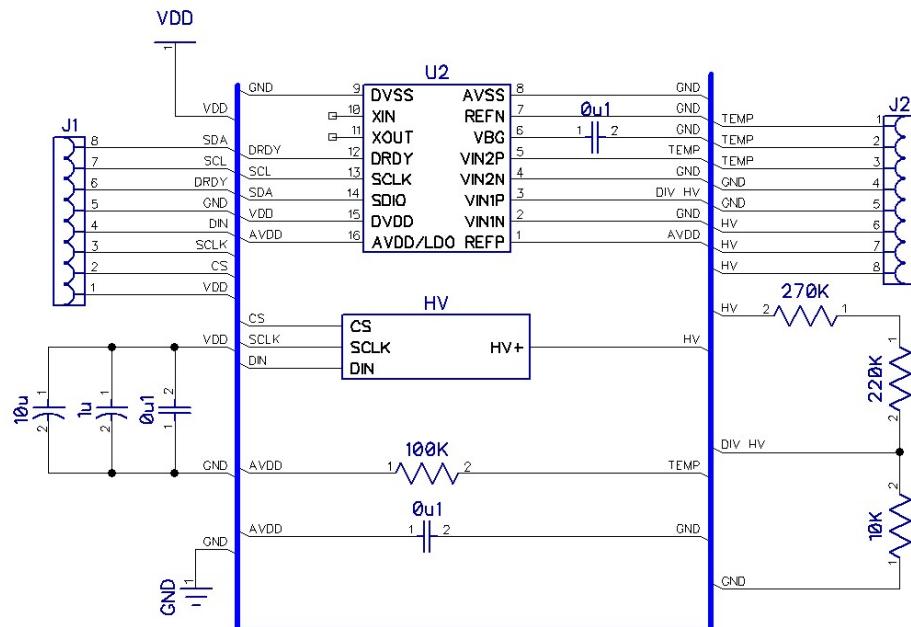


Figure 1.12: Schematic showing contacts, adc, and high voltage hierarchy block.

```

21 }   digitalWrite(CS, HIGH);
22 Serial.println(i);
23 }
  
```

| MPPC High Voltage Pinout | | |
|--------------------------|-------|---------------------------------|
| Pin | Label | Description |
| 1 | 3V3 | Regulated 3.3 V in |
| 2 | CS | <i>SPI</i> bus Chip Select |
| 3 | SCLK | <i>SPI</i> bus clock |
| 4 | DIN | <i>SPI</i> bus data in |
| 5 | GND | Common Ground |
| 6 | DRDY | ADC conversion done |
| 7 | SCL | <i>I²C</i> bus clock |
| 8 | SDA | <i>I²C</i> bus data |

Table 1.6: Pins and thier fuctions.

Listing 1.1: "max1932.ino"

ADC

The ADC communicates through the I^2C bus, the device address is permanently programed to 0101010 (0x2A). It is fully compliant with the I^2C protocol and the datasheet has the register map (page 28) and protocol description (page 14).

```

1 #include "nau7802/NAU7802.h"
2 #include "nau7802/NAU7802.cpp"
3 #include <Wire.h>

5 // Create the NAU7802 ADC object
NAU7802 adc = NAU7802();

7 void setup(void) {
9     Serial.begin(9600);
10    Wire.begin();           // Join i2c bus
11    adc.begin();           // Initialize ADC
12    adc.avcc2V4();         // Set voltage range to 0-2.4V
13}

15 void loop(void) {
16     // Read Channel 1
17     adc.selectCh1();    // Channel 1 has 50x Divided High voltage
18     Serial.print("Chanel 1: ");
19     Serial.println(adc.readmV());

21     // Read Channel 2
22     adc.selectCh2();    // Channel 2 has thermistor
23     Serial.print("Chanel 2: ");
24     Serial.println(adc.readmV());
25}

```

Listing 1.2: "nau7802.ino"

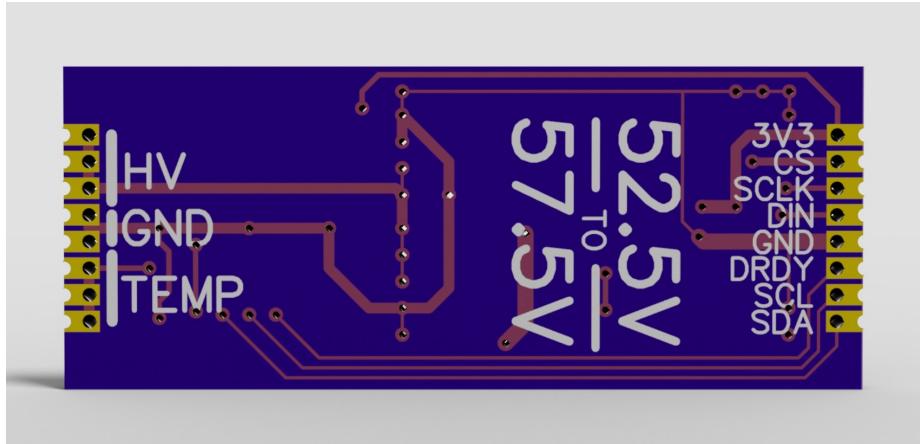


Figure 1.13: Back is flat to allow flush mounting to mother-board.

```

1 //Arduino library for interfacing with the nau7802 24bit ADC
2 #ifndef __NAU7802_H__
3 #define __NAU7802_H__

5 //Uncomment one of the following based on needs.
6 #define NAU7802_DEFAULT_WIRE
7 // #define NAU7802_SOFTWARE_WIRE
8 // #define NAU7802_ESP8266_WIRE
9
10 #include <Arduino.h>
11 #include <Wire.h>

13 #ifdef NAU7802_SOFTWARE_WIRE
14 #include <SoftwareWire.h>
15 #endif //NAU7802_SOFTWARE_WIRE

17 #define NAU7802_I2CADDR 0x2A //Static Adress

19 #define NAU7802_PU_CTRL 0x00 //REG0x00:PU_CTRL Powerup Control
20 #define NAU7802_CTRL1 0x01 //REG0x01:CTRL1 Control 1
21 #define NAU7802_CTRL2 0x02 //REG0x02:CTRL2 Control 2

23 #define NAU7802_OCAL1_B2 0x03 //REG0x03:OCAL1_B2 CH1 OFFSET Calibration[23:16]
24 #define NAU7802_OCAL1_B1 0x04 //REG0x04:OCAL1_B1 CH1 OFFSET Calibration[15:08]
25 #define NAU7802_OCAL1_B0 0x05 //REG0x05:OCAL1_B0 CH1 OFFSET Calibration[07:00]
26 #define NAU7802_GCAL1_B3 0x06 //REG0x06:GCAL1_B9 CH1 GAIN Calibration[31:24]
27 #define NAU7802_GCAL1_B2 0x07 //REG0x07:GCAL1_B2 CH1 GAIN Calibration[23:16]
28 #define NAU7802_GCAL1_B1 0x08 //REG0x08:GCAL1_B1 CH1 GAIN Calibration[15:08]
29 #define NAU7802_GCAL1_B0 0x09 //REG0x09:GCAL1_B0 CH1 GAIN Calibration[07:00]

31 #define NAU7802_OCAL2_B2 0x0A //REG0x0A:OCAL2_B2 CH2 OFFSET Calibration[23:16]
32 #define NAU7802_OCAL2_B1 0x0B //REG0x0B:OCAL2_B1 CH2 OFFSET Calibration[15:08]
33 #define NAU7802_OCAL2_B0 0x0C //REG0x0C:OCAL2_B0 CH2 OFFSET Calibration[07:00]
34 #define NAU7802_GCAL2_B3 0x0D //REG0x0D:GCAL2_B3 CH2 GAIN Calibration[31:24]
35 #define NAU7802_GCAL2_B2 0x0E //REG0x0E:GCAL2_B2 CH2 GAIN Calibration[23:16]
36 #define NAU7802_GCAL2_B0 0x0F //REG0x0F:GCAL2_B0 CH2 GAIN Calibration[15:08]
37 #define NAU7802_GCAL2_B1 0x10 //REG0x10:GCAL2_B1 CH2 GAIN Calibration[07:00]

39 #define NAU7802_I2C_CTRL 0x11 //REG0x11:I2C_CTRL I2C Control

41 #define NAU7802_ADC_B2 0x12 //REG0x12:ADC_OUT_B2 ADC Conversion Result [23:16]
42 #define NAU7802_ADC_B1 0x13 //REG0x13:ADC_OUT_B1 ADC Conversion Result [15:08]
43 #define NAU7802_ADC_B0 0x14 //REG0x14:ADC_OUT_B0 ADC Conversion Result [07:00]

45 #define NAU7802_ADC_REG 0x15 //REG0x15:ADC_REG ADC Registers

47 #define NAU7802 OTP_B1 0x16 //REG0x16:OTP_B1 OTP[15:08]
48 #define NAU7802 OTP_B0 0x17 //REG0x17:OTP_B0 OTP[07:00]

49 #define NAU7802_RES_00 0x18 //REG0x18:RES_00 RESERVED
50 #define NAU7802_RES_01 0x19 //REG0x19:RES_01 RESERVED
51 #define NAU7802_RES_02 0x1A //REG0x1A:RES_02 RESERVED

53 #define NAU7802_PGA_REG 0x1B //REG0x1B:PGA_REG Pre-Gain Amplifier
54 #define NAU7802_POW_CTRL 0x1C //REG0x1C:POW_CTRL Power Control

57 #define NAU7802_RES_03 0x1D //REG0x16:RES_03 RESERVED
58 #define NAU7802_RES_04 0x1E //REG0x1E:RES_04 RESERVED

59 #define NAU7802_REV_ID 0x1F //REG0x1F:READ_ONLY MFG test and RevisionID

```

```

61 //REG0x00:PU_CTRL      Powerup Control
63 #define NAU7802_AVDDS    7 //AVDD source select          1:Internal LDO      0:AVDD Pin Input      (0-Default)
64 #define NAU7802_OSCS     6 //System clock source select  1:External Crystal   0:Internal RC OScillator (0-Default)
65 #define NAU7802_CR       5 //Cycle ready (Read Only)  1:ADC Data Ready    0:No ADC Data      (0-Default)
66 #define NAU7802_CS       4 //Cycle start ADC        1:Start ADC         0:Stop ADC        (0-Default)
67 #define NAU7802_PUR      3 //Power up ready (Read Only) 1:Power up, Ready   0:Power down, not ready (0-Default)
68 #define NAU7802_PUA      2 //Power up analog circuit 1:Power up analaog  0:Power down analog (0-Default)
69 #define NAU7802_PUD      1 //Power up digital circuit 1:Power up digital  0:Power down digital (0-Default)
70 #define NAU7802_RR       0 //Register Reset        1:Reset all save RR 0:Normal Operation (0-Default)
71 //REG0x01:CTRL1        Control 1
73 #define NAU7802_CRP      7 //Conversion Ready Polarity 1:Active Low        0:Active high      (0-Default)
74 #define NAU7802_DRDY_SEL 6 //DRDY pin fuction       1:Output clock      0:Output conv ready (0-Default)
75 //define NAU7802_VLDO2   5 //Select LDO Voltage      1 2V4      1 2V7      1 3V0      1 0 3V3      1 3V6      1 3V9      0 4V2      0 4V5      (0-Default)
76 //define NAU7802_VLDO1   4 //Select LDO Voltage      1 2V4      1 2V7      0 3V0      0 3V3      1 3V6      1 3V9      0 4V2      0 4V5      (0-Default)
77 //define NAU7802_VLDO0   3 //Select LDO Voltage      1 0          1 0          0 1          0 1          0 1          0 1          0 1          0 1          (0-Default)
78 //define NAU7802_GAINS2  2 //Select gain           1 1          1 1          1 0          0 0          0 0          0 0          0 0          (0-Default)
79 //define NAU7802_GAINS1  1 //Select gain           1 128      1 x64      0 x32      0 x16      1 x8       1 x4       0 x2       0 x1       (0-Default)
80 //define NAU7802_GAINS0  0 //Select gain           1 0          1 0          0 1          0 1          0 1          0 1          0 1          (0-Default)
81 //REG0x02:CTRL2        Control 2
82 #define NAU7802_CHS     7 //Analog input channel 1:Channel 2          0:Channel 1      (0-Default)
83 //define NAU7802_CRS2   6 //Converstion Rate samp/sec 1 320      1 N/A      0 N/A      0 N/A      1 80       1 40       0 20       0 10       (0-Default)
84 //define NAU7802_CRS1   5 //Converstion Rate samp/sec 1 320      1 N/A      0 N/A      0 N/A      1 80       1 40       0 20       0 10       (0-Default)
85 //define NAU7802_CRS0   4 //Converstion Rate samp/sec 1 0          1 0          0 1          0 1          0 1          0 1          0 1          (0-Default)
86 //define NAU7802_CAL_ERR 3 //Calibration Error      1:Calibration failed 0:No          (0-Default)
87 //define NAU7802_CALS   2 //Start Calibration      1:Start calibration 0:Calibration finished (0-Default)
88 //define NAU7802_CALMOD1 1 //Calibration Selection 1:Gain Cal      1:Offset Cal    0:RESERVED   0: Offset Cal (0-Default)
89 //define NAU7802_CALMOD0 0 //Calibration Selection 1:System       0:System      1:RESERVED   0: Internal  (0-Default)
90 //REG0x11:I2C_CTRL     I2C Control
91 #define NAU7802_CRSRD   7 //Pull SDA low on conversion 1:Enalbe      0:Disable      (0-Default)
92 #define NAU7802_FRD     6 //Fast ADC DATA        1:Enalbe REQ REG0x15[7]:1 0:Disable      (0-Default)
93 #define NAU7802_SPE     5 //Enable strong pullup  1:Enalbe 1K6 pullup 0:Disable 1K6 pullup (0-Default)
94 #define NAU7802_WPD     4 //Disable weak pullup  1:Disable 50K pullup 0:Enable 50K pullup (0-Default)
95 #define NAU7802_SI      3 //Short Inputs        1:Short Inputs      0:Inputs floating (0-Default)
96 #define NAU7802_BOPGA   2 //PGA bounout current source 1:2.5uA Current to PGA+ 0:Current disabled (0-Default)
97 #define NAU7802_TS      1 //Temprature Sensor Select 1:Temprature to PGA 0:Temp disabled (0-Default)
98 #define NAU7802_BGCP    0 //Bandgap chopper      1:Disable      0:Enable      (0-Default)
99 //class NAU7802 {
100 public:
101 NAU7802();
102
103 #ifdef NAU7802_DEFAULT_WIRE
104     boolean begin(uint8_t addr = NAU7802_I2CADDR);
105 #endif //NAU7802_DEFAULT_WIRE
106
107 #ifdef NAU7802_SOFTWARE_WIRE
108     boolean begin(uint8_t sda, uint8_t scl, uint8_t addr = NAU7802_I2CADDR);
109 #endif //NAU7802_SOFTWARE_WIRE
110
111 #ifdef NAU7802_ESP8266_WIRE
112     boolean begin(uint8_t sda, uint8_t scl, uint8_t addr = NAU7802_I2CADDR);
113 #endif //NAU7802_ESP8266_WIRE

```

```

123   long  readADC();
124   float  readmV();
125
126   void  selectCh1();
127   void  selectCh2();
128   void  selectTemp();
129
130   void  rate010sps();
131   void  rate020sps();
132   void  rate040sps();
133   void  rate080sps();
134   void  rate320sps();
135
136   void  pga128x();
137   void  pga64x();
138   void  pga32x();
139   void  pga16x();
140   void  pga8x();
141   void  pga4x();
142   void  pga2x();
143   void  pga1x();
144   void  pgaDisable();
145
146   void  extAvcc(float extAvcc);
147   void  avcc2V4();
148   void  avcc2V7();
149   void  avcc3V0();
150   void  avcc3V3();
151   void  avcc3V6();
152   void  avcc3V9();
153   void  avcc4V2();
154   void  avcc4V5();
155
156 private:
157 #ifdef NAU7802_DEFAULT_WIRE
158   TwoWire wire = TwoWire();
159 #endif //NAU7802_DEFAULT_WIRE
160
161 #ifdef NAU7802_SOFTWARE_WIRE
162   SoftwareWire wire = SoftwareWire(2,3);
163 #endif //NAU7802_SOFTWARE_WIRE
164
165 #ifdef NAU7802_ESP8266_WIRE
166   TwoWire wire = TwoWire();
167 #endif //NAU7802_ESP8266_WIRE
168
169   uint8_t _i2caddr;
170   float _avcc = 3.3;
171
172   void  resetSettings();
173
174   void  write(uint8_t reg, uint8_t val);
175   void  writeBit(uint8_t reg, uint8_t bit);
176   void  clearBit(uint8_t reg, uint8_t bit);
177
178   uint8_t  read(uint8_t reg);
179   uint32_t read24(uint8_t reg);
180   uint32_t read32(uint8_t reg);
181   boolean  readBit(uint8_t reg, uint8_t bit);

```

```

183     void  readUntilTrue(uint8_t reg, uint8_t bit);
184     void  readUntilFalse(uint8_t reg, uint8_t bit);
185 };
186
187 #endif //__NAU7802_H__

```

Listing 1.3: "NAU7802.h"

```

1 //Readout for nau7802
2 #include "NAU7802.h"
3
4 //Instantiates a new NAU7802 class
5 //=====
6 NAU7802::NAU7802() {
7 }
8
9 // Setups the HW
10 //=====
11 #ifdef NAU7802_DEFAULT_WIRE
12 boolean NAU7802::begin(uint8_t addr) {
13     _i2caddr = addr;
14     wire = TwoWire();
15     wire.begin();
16     resetSettings();
17     return true;
18 }
19 #endif //NAU7802_DEFAULT_WIRE
20
21 #ifdef NAU7802_SOFTWARE_WIRE
22 boolean NAU7802::begin(uint8_t sda, uint8_t scl, uint8_t addr) {
23     _i2caddr = addr;
24     wire = SoftwareWire(sda, scl);
25     wire.begin();
26     resetSettings();
27     return true;
28 }
29 #endif //NAU7802_SOFTWARE_WIRE
30
31 #ifdef NAU7802_ESP8266_WIRE
32 boolean NAU7802::begin(uint8_t sda, uint8_t scl, uint8_t addr) {
33     _i2caddr = addr;
34     wire = TwoWire();
35     wire.begin(sda, scl);
36     resetSettings();
37     return true;
38 }
39 #endif //NAU7802_ESP8266_WIRE
40
41 void NAU7802::resetSettings(){
42     writeBit(NAU7802_PU_CTRL, NAU7802_RR);           //Reset Registers
43     clearBit(NAU7802_PU_CTRL, NAU7802_RR);          //Clear Reset Registers
44     writeBit(NAU7802_PU_CTRL, NAU7802_PUD);         //Power up digital
45     readUntilTrue(NAU7802_PU_CTRL, NAU7802_PUR);    //Wait until power up
46     writeBit(NAU7802_PU_CTRL, NAU7802_PUA);         //Power up analog
47
48     writeBit(NAU7802_ADC_REG, 4);                   //Disable chopper function
49     writeBit(NAU7802_ADC_REG, 5);                   //Disable chopper function
50

```

```

52     writeBit(NAU7802.PGA_REG, 0);           //Disable chopper function
53     writeBit(NAU7802.PGA_REG, 4);           //Bypass PGA
54
55     writeBit(NAU7802 CTRL2, NAU7802.CALS); //Begin calibration
56     readUntilFalse(NAU7802 CTRL2, NAU7802.CALS); //Wait for calibration to finish
57
58     writeBit(NAU7802_I2C_CTRL, NAU7802.SPE); //Enable Strong Pullup
59
60     writeBit(NAU7802_PU_CTRL, NAU7802_CS); //Start Conversion
61 }
62
63 //Reads set channel
64 //=====
65 long NAU7802::readADC(){
66     readUntilTrue(NAU7802_PU_CTRL, NAU7802_CR);
67     uint32_t adcVal = read24(NAU7802_ADC_B2);
68     writeBit(NAU7802_PU_CTRL, NAU7802_CS);
69     if(adcVal & 0x00800000){
70         adcVal = ~adcVal+1;
71         adcVal = -1*(adcVal & 0x00FFFFFF);
72     }
73     return adcVal;
74 }
75
76 float NAU7802::readmV(){
77     return (-avcc/(float)16777216)*(float)readADC();
78 }
79
80 //Select channel
81 //=====
82 void NAU7802::selectCh1(){
83     clearBit(NAU7802_CTRL2, 7);
84 }
85 void NAU7802::selectCh2(){
86     writeBit(NAU7802_CTRL2, 7);
87 }
88 void NAU7802::selectTemp(){
89 }
90
91 //Set Sampling rate
92 //=====
93 void NAU7802::rate010sps(){
94     clearBit(NAU7802_CTRL2, 4);
95     clearBit(NAU7802_CTRL2, 5);
96     clearBit(NAU7802_CTRL2, 6);
97 }
98 void NAU7802::rate020sps(){
99     clearBit(NAU7802_CTRL2, 4);
100    clearBit(NAU7802_CTRL2, 5);
101    writeBit(NAU7802_CTRL2, 6);
102 }
103 void NAU7802::rate040sps(){
104     clearBit(NAU7802_CTRL2, 4);
105     writeBit(NAU7802_CTRL2, 5);
106     clearBit(NAU7802_CTRL2, 6);
107 }
108 void NAU7802::rate080sps(){
109     clearBit(NAU7802_CTRL2, 4);
110     writeBit(NAU7802_CTRL2, 5);
111     writeBit(NAU7802_CTRL2, 6);
112 }

```

```

112 void NAU7802::rate320sps(){
113     writeBit(NAU7802_CTRL2, 4);
114     writeBit(NAU7802_CTRL2, 5);
115     writeBit(NAU7802_CTRL2, 6);
116 }
117 //=====
118 //=====
119 void NAU7802::pga128x(){
120 }
121 void NAU7802::pga64x(){
122 }
123 void NAU7802::pga32x(){
124 }
125 void NAU7802::pga16x(){
126 }
127 void NAU7802::pga8x(){
128 }
129 void NAU7802::pga4x(){
130 }
131 void NAU7802::pga2x(){
132 }
133 void NAU7802::pga1x(){
134 }
135 void NAU7802::pgaDisable(){
136 }
137 //=====
138 //=====
139 void NAU7802::extAvcc(float extAvcc){
140     clearBit(NAU7802_PU_CTRL, NAU7802_AVDDS); // Disable LDO
141     _avcc=extAvcc;
142 }
143 void NAU7802::avcc2V4(){
144     writeBit(NAU7802_PU_CTRL, NAU7802_AVDDS); // Enable LDO
145     writeBit(NAU7802_CTRL1, NAU7802_VLDO2); // Set AVCC to 2.4V
146     writeBit(NAU7802_CTRL1, NAU7802_VLDO1); // Set AVCC to 2.4V
147     writeBit(NAU7802_CTRL1, NAU7802_VLDO0); // Set AVCC to 2.4V
148     _avcc=2.4; // Local AVCC variable
149 }
150 void NAU7802::avcc2V7(){
151     writeBit(NAU7802_PU_CTRL, NAU7802_AVDDS); // Enable LDO
152     writeBit(NAU7802_CTRL1, NAU7802_VLDO2); // Set AVCC to 2.7V
153     writeBit(NAU7802_CTRL1, NAU7802_VLDO1); // Set AVCC to 2.7V
154     clearBit(NAU7802_CTRL1, NAU7802_VLDO0); // Set AVCC to 2.7V
155     _avcc=2.7; // Local AVCC variable
156 }
157 void NAU7802::avcc3V0(){
158     writeBit(NAU7802_PU_CTRL, NAU7802_AVDDS); // Enable LDO
159     writeBit(NAU7802_CTRL1, NAU7802_VLDO2); // Set AVCC to 3.0V
160     clearBit(NAU7802_CTRL1, NAU7802_VLDO1); // Set AVCC to 3.0V
161     writeBit(NAU7802_CTRL1, NAU7802_VLDO0); // Set AVCC to 3.0V
162     _avcc=3.0; // Local AVCC variable
163 }
164 void NAU7802::avcc3V3(){
165     writeBit(NAU7802_PU_CTRL, NAU7802_AVDDS); // Enable LDO
166     writeBit(NAU7802_CTRL1, NAU7802_VLDO2); // Set AVCC to 3.3V
167     clearBit(NAU7802_CTRL1, NAU7802_VLDO1); // Set AVCC to 3.3V
168     clearBit(NAU7802_CTRL1, NAU7802_VLDO0); // Set AVCC to 3.3V
169     _avcc=3.3; // Local AVCC variable
170 }
171 }
```

```

174 void NAU7802::avcc3V6 () {
175     writeBit (NAU7802_PU_CTRL, NAU7802_AVDD5); //Enable LDO
176     clearBit (NAU7802_CTRL1, NAU7802_VLDO2); //Set AVCC to 3.6V
177     writeBit (NAU7802_CTRL1, NAU7802_VLDO1); //Set AVCC to 3.6V
178     writeBit (NAU7802_CTRL1, NAU7802_VLDO0); //Set AVCC to 3.6V
179     _avcc=3.6; //Local AVCC variable
180 }
181 void NAU7802::avcc3V9 () {
182     writeBit (NAU7802_PU_CTRL, NAU7802_AVDD5); //Enable LDO
183     clearBit (NAU7802_CTRL1, NAU7802_VLDO2); //Set AVCC to 3.9V
184     writeBit (NAU7802_CTRL1, NAU7802_VLDO1); //Set AVCC to 3.9V
185     clearBit (NAU7802_CTRL1, NAU7802_VLDO0); //Set AVCC to 3.9V
186     _avcc=3.9; //Local AVCC variable
187 }
188 void NAU7802::avcc4V2 () {
189     writeBit (NAU7802_PU_CTRL, NAU7802_AVDD5); //Enable LDO
190     clearBit (NAU7802_CTRL1, NAU7802_VLDO2); //Set AVCC to 4.2V
191     clearBit (NAU7802_CTRL1, NAU7802_VLDO1); //Set AVCC to 4.2V
192     writeBit (NAU7802_CTRL1, NAU7802_VLDO0); //Set AVCC to 4.2V
193     _avcc=4.2; //Local AVCC variable
194 }
195 void NAU7802::avcc4V5 () {
196     writeBit (NAU7802_PU_CTRL, NAU7802_AVDD5); //Enable LDO
197     clearBit (NAU7802_CTRL1, NAU7802_VLDO2); //Set AVCC to 4.5V
198     clearBit (NAU7802_CTRL1, NAU7802_VLDO1); //Set AVCC to 4.5V
199     clearBit (NAU7802_CTRL1, NAU7802_VLDO0); //Set AVCC to 4.5V
200     _avcc=4.5; //Local AVCC variable
201 }

202 //Low level read and write procedures
203 //=====
204
205 void NAU7802::write (uint8_t reg, uint8_t val) {
206     wire.beginTransmission (.i2caddr);
207     wire.write ((uint8_t)reg);
208     wire.write (val);
209     wire.endTransmission ();
210 }

211 void NAU7802::writeBit (uint8_t reg, uint8_t bit) {
212     uint8_t val = read (reg) | (1<<bit);
213     write (reg, val);
214 }

215 void NAU7802::clearBit (uint8_t reg, uint8_t bit) {
216     uint8_t val = read (reg) & ~(1<<bit);
217     write (reg, val);
218 }

219 uint8_t NAU7802::read (uint8_t reg) {
220     wire.beginTransmission (.i2caddr);
221     wire.write ((uint8_t)reg);
222     wire.endTransmission ();
223
224     wire.requestFrom ((uint8_t).i2caddr, (uint8_t)1);
225     return wire.read ();
226 }

227 boolean NAU7802::readBit (uint8_t reg, uint8_t bit) {
228     //create bitmask
229     uint8_t bitmask = 1<<bit;

```

```

234     if (read(reg) & bitmask){
235         return true;
236     }
237     return false;
238 }
239
240 uint32_t NAU7802::read24(uint8_t reg) {
241     uint32_t val;
242
243     wire.beginTransmission(_i2caddr);
244     wire.write((uint8_t)reg);
245     wire.endTransmission();
246
247     wire.requestFrom((uint8_t)_i2caddr, (uint8_t)3);
248     val = wire.read();           // receive high byte
249     val <= 8;                  // shift byte to make room for new byte
250     val |= wire.read();        // receive mid byte
251     val <= 8;                  // shift both bytes
252     val |= wire.read();        // receive low byte
253     return val;
254 }
255
256 uint32_t NAU7802::read32(uint8_t reg) {
257     uint32_t val;
258
259     wire.beginTransmission(_i2caddr);
260     wire.write((uint8_t)reg);
261     wire.endTransmission();
262
263     wire.requestFrom((uint8_t)_i2caddr, (uint8_t)4);
264     val = wire.read();           // receive [31:24] byte
265     val <= 8;                  // shift byte
266     val = wire.read();           // receive [23:16] byte
267     val <= 8;                  // shift bytes
268     val |= wire.read();        // receive [15:08] byte
269     val <= 8;                  // shift bytes
270     val |= wire.read();        // receive [07:00] byte
271     return val;
272 }
273
274 void NAU7802::readUntilTrue(uint8_t reg, uint8_t bit) {
275     //create bitmask
276     uint8_t bitmask = 1<<bit;
277     bool readUntil = false;
278     //Just keep reading until bit requested is true
279     while(readUntil == false){
280         if(read(reg) & bitmask){
281             readUntil=true;
282         }
283     }
284 }
285
286 void NAU7802::readUntilFalse(uint8_t reg, uint8_t bit) {
287     //create bitmask
288     uint8_t bitmask = 1<<bit;
289     bool readUntil = false;
290     //Just keep reading until bit requested is false
291     while(readUntil == false){
292         if(~read(reg) & bitmask){
293             readUntil=true;
294         }

```

296 }

Listing 1.4: "Nau7802.cpp"

1.4 MPPC Interface

A single assembled board breaks out singals and provides all the interfaces that 4 MPPC Sesnor boards need. The power to each module is supplied via the MPPC high voltage modules attached board. It is able to be run independently for testing and debugging, or on a backplane with many others for more channels.

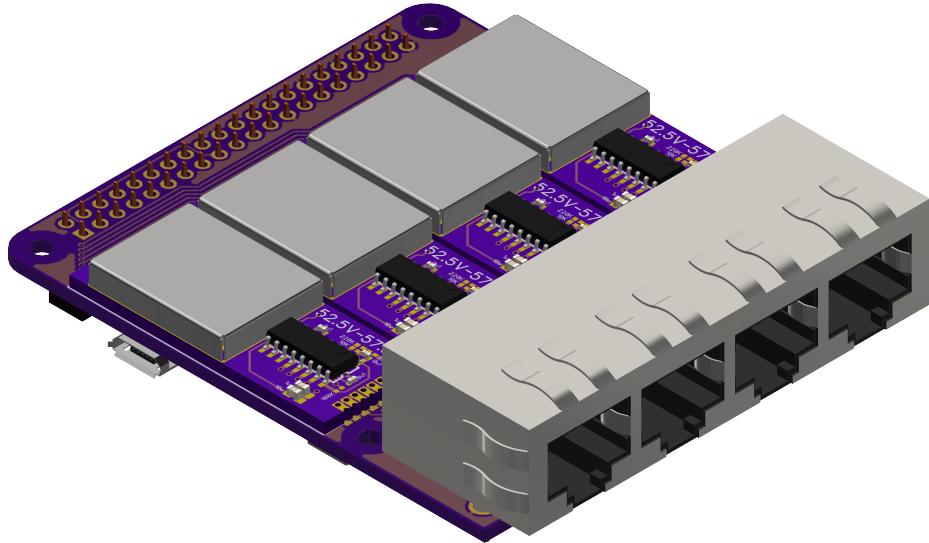


Figure 1.14: Top view of assembled MPPC interface board.

1.4.1 Hardware

The board attaches directly to a Raspberry Pi using the centered GPIO headers. The Raspberry Pi is mounted to the board via four standoffs. It is a stand alone board requiring no backplane. This board can be mounted via the M5 screw holes around the edge.

Power

Power is supplied through two pins on the edge connector, or via the micro USB port on board. There is a 600mA buck regulator that provides power when the USB port is used. The board should be powered by 3.3V and a negative power supply module with convert it to -3.3V for the biasing for the op-amps.

High Voltage

High voltage is supplied through a daughter board. Each channel has a independent voltage source that can be temperature modulated and monitored via the on board ADC. Further information about the modules can be found in Section.

Microcontroller

The side of the board has a Arduino compatible (ATmega328) microcontroller, along with a USB part, USB to UART converter and other supporting circuitry. The button is connected to the reset pin. The programing port on board is a Tag-Connect 6 pin, and is only used for programing the Arduino bootloader.

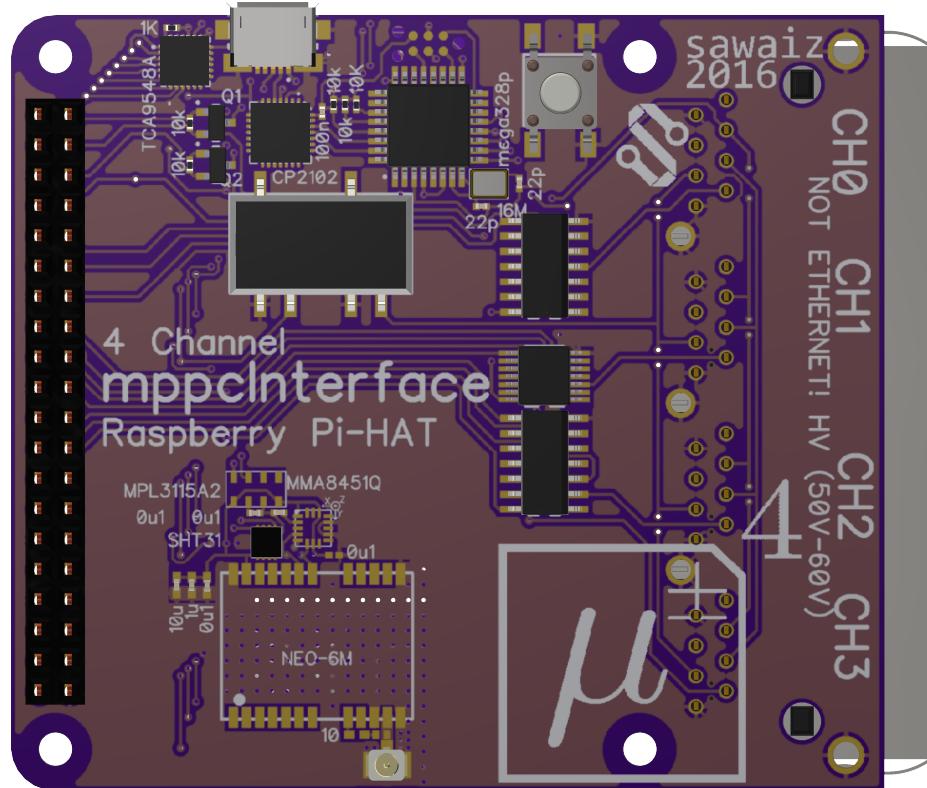


Figure 1.15: Bottom of populated board showing sensors and AVR.

I2C Switch

As all the ADC modules on the daughter boards have the same address, a 8 channel I2C switch ([TCA9548A](http://www.ti.com/lit/ds/symlink/tca9548a.pdf)) is used to communicate to them independently. The pins attached to it are referenced as master, as the microcontroller is acting as the master controlling it.

Pinout

There are eight RJ-45 jacks on the edge of the board, each jack has the following pinout. The signal lines are kept on a single twisted pair so if a differential signal is used, higher signal integrity is maintained. The pinout matches that of the MPPC sensor board.

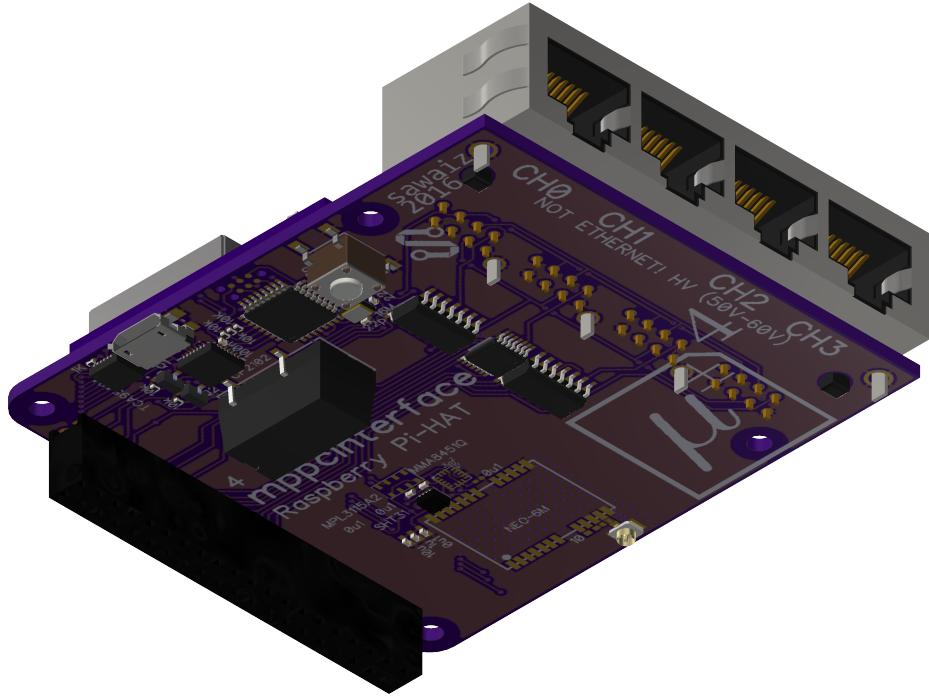


Figure 1.16: Isometric view of bottom.

1.4.2 Software

The device acts as both a master and slave device. It takes instructions from an external I2C bus, and executes them on the AVR microcontroller to set voltages, provide telemetry, and perform closed loop control.

AVR Firmware

The microcontroller on board takes commands sent to the whole module, and forwards them to each daughterboard. Its primary control loop involves reading the temperature, finding the associated required bias voltage based on the temperature and setting the voltage on the high voltage supply, then checking if the output voltage is within specification of the required output, and fixing it if not. This loops for all eight channels. Otherwise there is an instruction queue of commands that come into the module.

Module

The master can communicate to up to 128 slave devices the address is set within software. The master can request the temperature, or current voltage of any of the eight channels on board, select between different voltage to temperature curves, and set the bias voltage, or pulse the LED on the modules.

1.5 Frame

The frame is constructed out of 4 m of Misumi 2020 aluminium extrusion, cast angle brackets, t-nuts, M5 screws, and rubber feet. The assembly process requires only a 4 mm hex key.

1.5.1 Construction

The frame is designed to be easy to assemble, take apart, and modify for different experiments and highly rugged. It is constructed out of metric aluminium extrusion and cast aluminium angle brackets, with rubber feet at the bottom. A full bill of materials is shown in Table 1.7. The panels it is meant to hold fit within the 20 mm extrusion height, and the remaining area is filled with a soft tubing spring. The panels are supported on both sides by $\frac{3}{32}$ inch acrylic sheet and a addition panel provides a mounting location for the electronics. The assembly process requires only a 4 mm hex key, and can be made vibration resistant through the application of threadlocker (blue color code).

| Bill of Materials | | | | | | |
|---------------------|-----------------|---------------|---------------------------------|--------------------|--------|--|
| Description | Item | Supplier | QTY | Unit | Ext | |
| Aluminium Extrusion | KHFS5-2020-1000 | Misumi | 4 | 6.41 | 25.64 | |
| Angle Brackets | HBLFSNF5 | Misumi | 64 | 0.63 | 40.32 | |
| M5 T-Nuts | HNKK5-5 | Misumi | ¹⁵⁶ / ₁₀₀ | 15.26 | 23.81 | |
| M5x6 | 91292a189 | McMaster-Carr | ²⁸ / ₅₀ | 4.63 | 2.59 | |
| M5x10 | 91292A124 | McMaster-Carr | ¹²⁸ / ₁₀₀ | 8.15 | 11.20 | |
| M5x25 | 91292a129 | McMaster-Carr | ⁴ / ₂₅ | 6.85 | 1.10 | |
| Acrylic Sheet | 8560k182 | McMaster-Carr | ⁷ / ₆ | 8.96 | 10.45 | |
| Rubber Feet | 9540K717 | McMaster-Carr | ⁴ / ₂₅ | 7.42 | 1.19 | |
| Labor | | | 2 | 30.00 | 60.00 | |
| | | | | Subtotal | XXX.XX | |
| | | | | Tax | %8.000 | |
| | | | | Transport | %XX.XX | |
| | | | | Total (USD) | 189.24 | |

Table 1.7: Materials required for one frame.

| Aluminum Extrusion Cut List | | | |
|-----------------------------|-----|-------------|-------------------------------------|
| Description | QTY | Length (mm) | Note |
| Vertical Beam | 4 | 396 | M5 thread tapped one end 20 mm deep |
| Horizontal Beam | 16 | 150 | |

Table 1.8: One vertical and four horizontal to be cut from each 1 m section. 1 mm kerf.

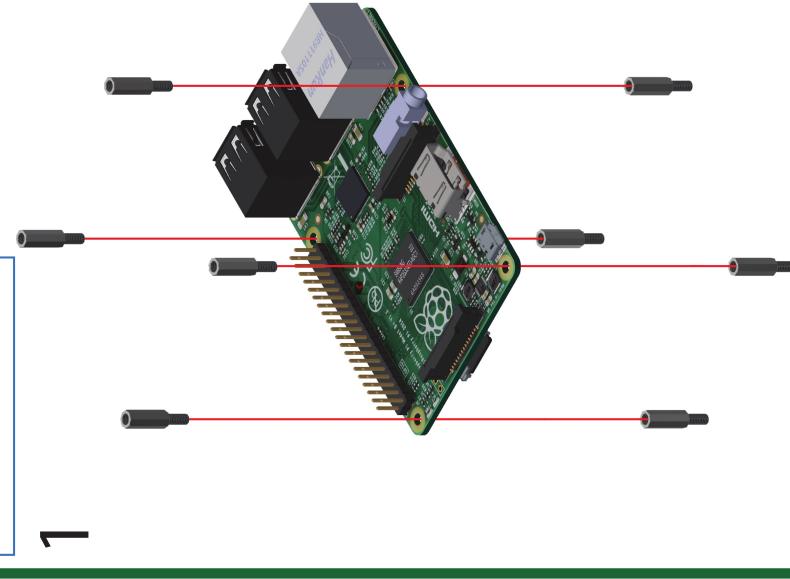
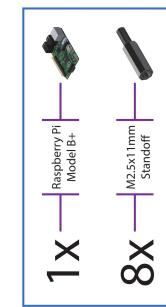
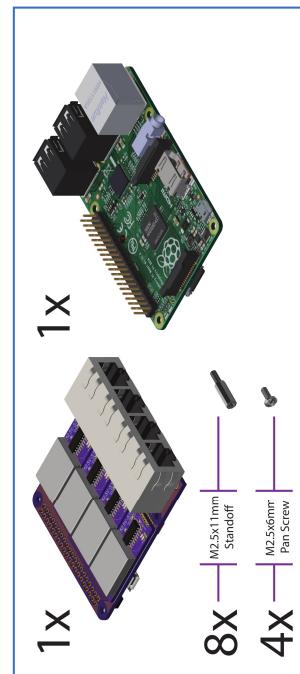
Assembly

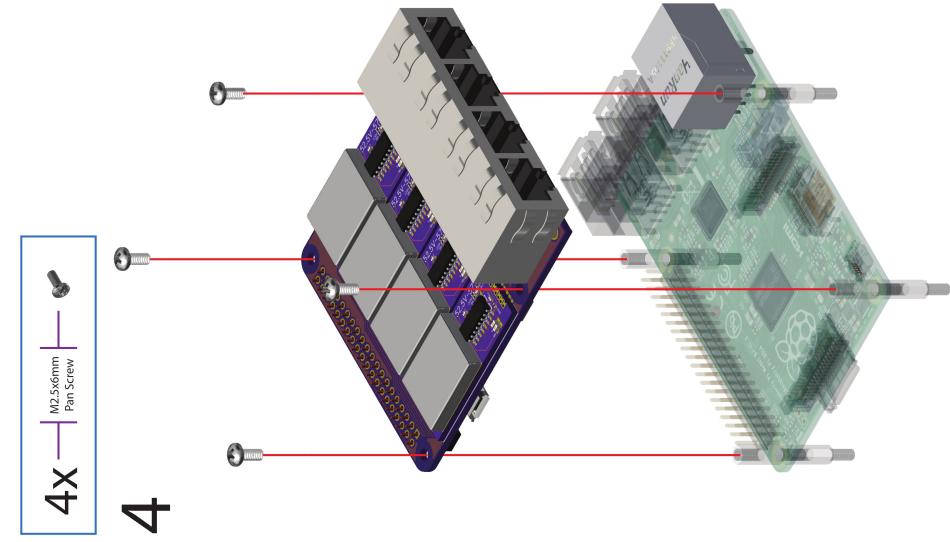
Insert assembly PDF output form Illistrator

Chapter 2

Assembly

Four Channel MPPC Interface





4

Flash OS on SD card

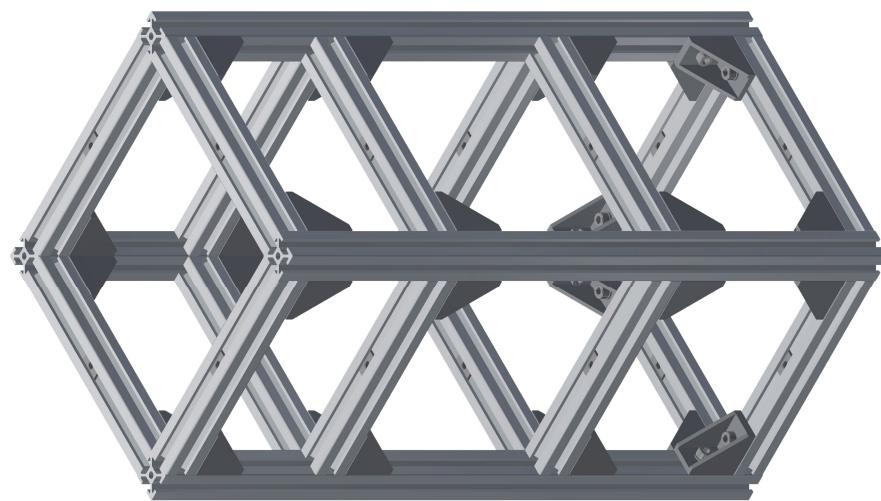
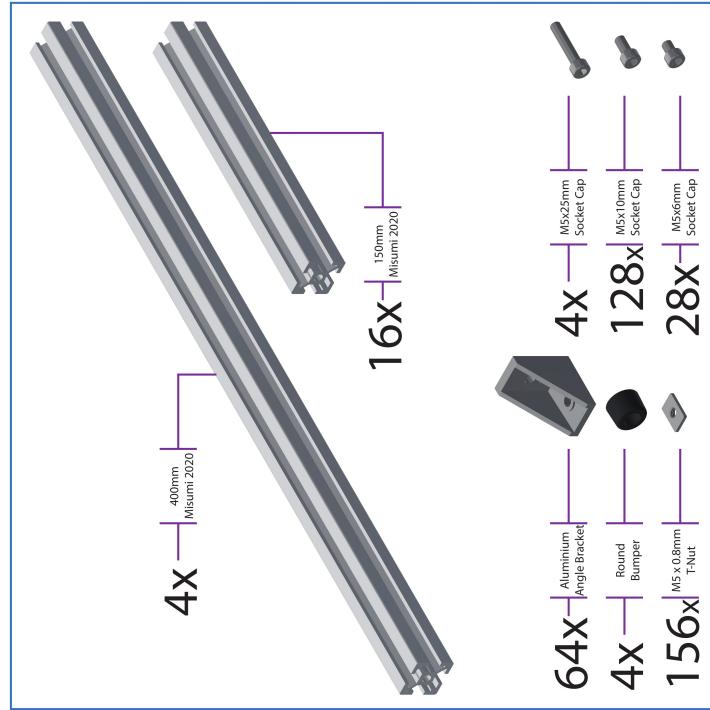
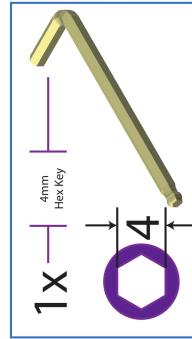
2

https://downloads.raspberrypi.org/raspbian_lite_latest

```
> df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       112G  107G   4.5G  97% /
/dev/sdb1       120G   2.4G  118G   2% /dev/sdX1
> sudo umount /dev/sdX
> sudo dd if=<imagePath.img> of=</dev/sdX>
```



3



Muon Telescope Frame

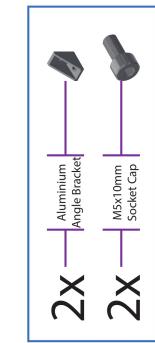
1

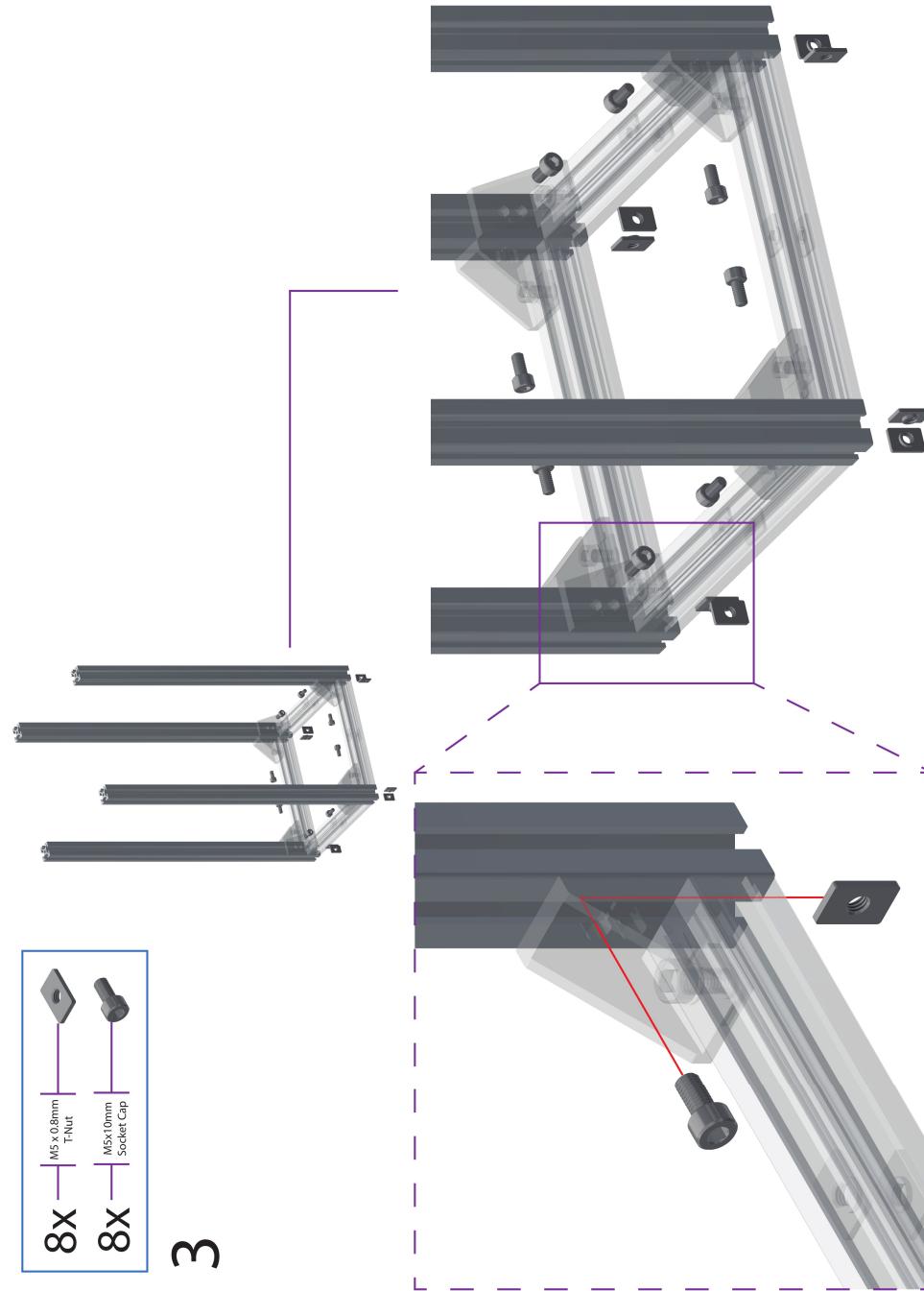


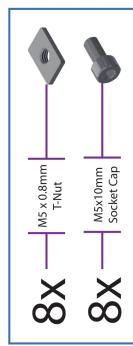
16x



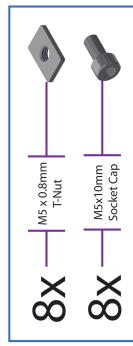
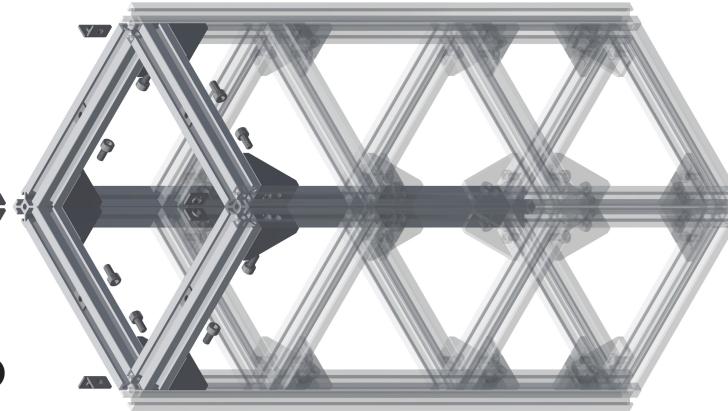
2



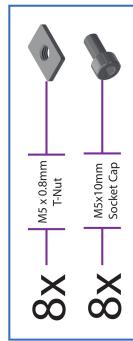
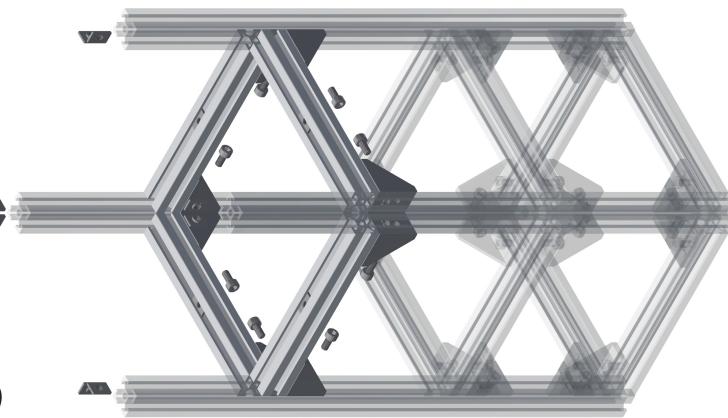




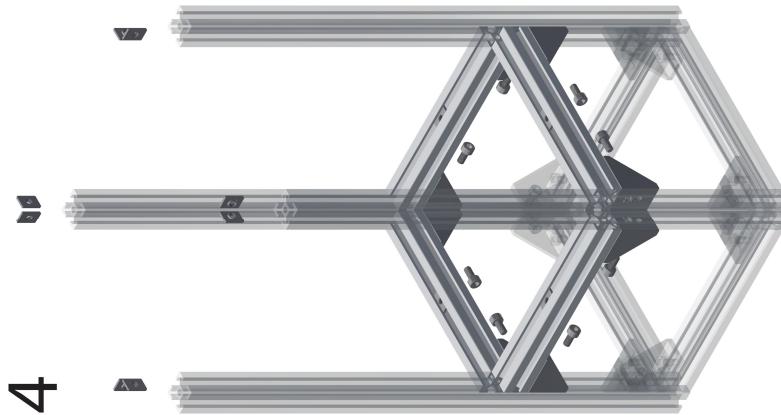
6

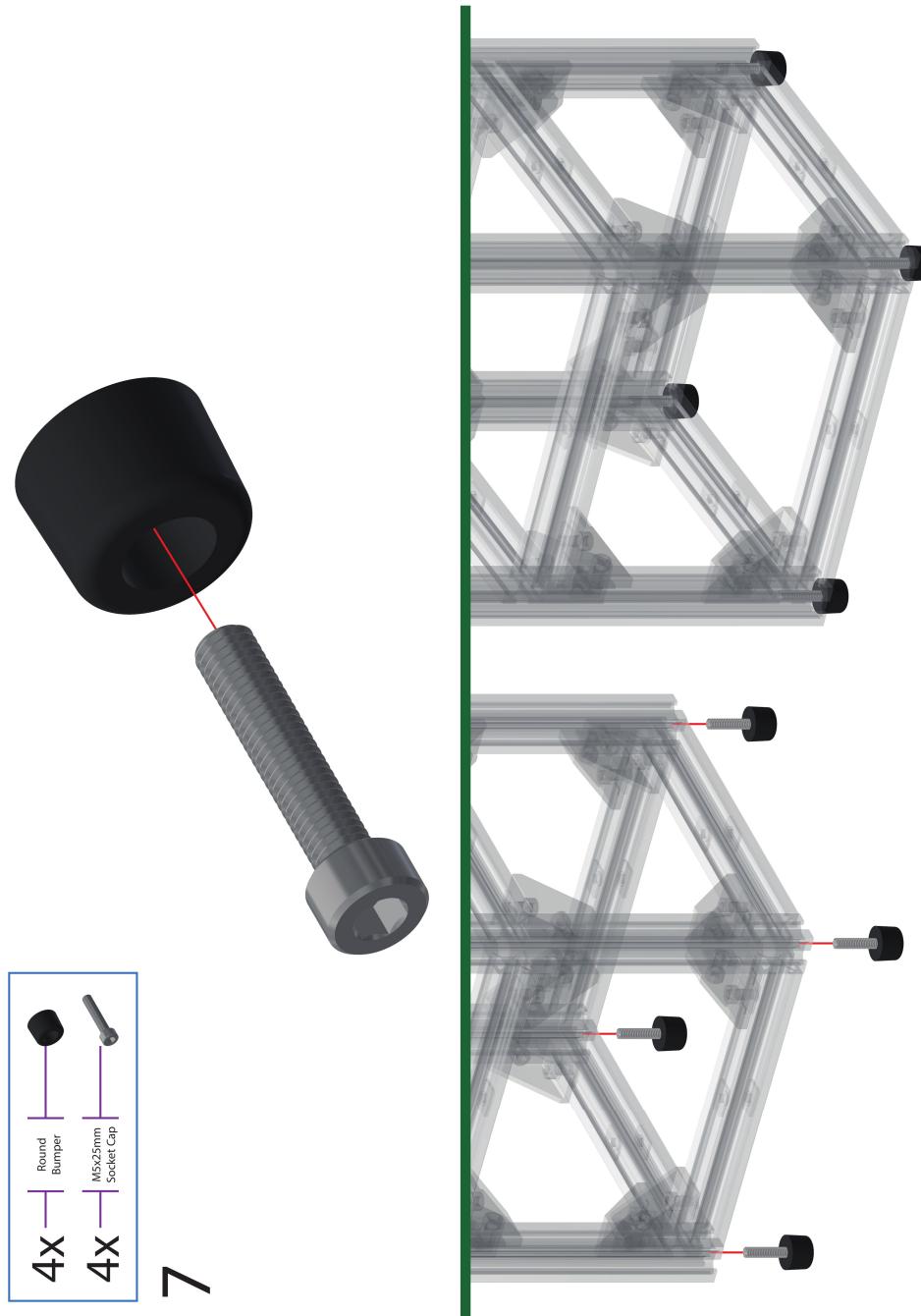


5



4





Chapter 3

Software

3.1 Structure

3.1.1 Technologies

The application is built on many modern web technologies. This allows the application to be easily scaleable, modular, and deployable. The following sections list the technologies and how they are used.

Docker

Docker is a linux containerization platform. By creating virtual containers that share the same underlying kernel, docker is able to provide an internal routing network as well as separation and modularity of modules designed. Docker is also being used for deployment, instead of requiring the user to install multiple packages and configure them, or doing so through a makefile designed for the distro installation, we are able to target just the processor architecture, meaning installing docker on either a ARM or x86 processor is the only requirement. The containers are configured to install debian and then only the software required for the execution of that module, this allows debugging and testing of single modules, and removal of modules not needed for an application.

Node.js

Node.js is a Javascript runtime built on Chrome's V8 engine. It is used for back-end processing and is the server-side language we use. Its event driven asynchronous model allows serving multiple requests without blocking. It is also highly extensible via NPM (Node Package Manager) which provides many modules to speed up workflow. Some of the modules we use as well as their functions are shown below.

- *Express* - Web server for hosting dynamic content. Backbone of our REST API.

- *Sequelize* - MySql interface that allows direct queries.
- *RaspIO* - Input-output capabilities for the Raspberry Pi.

We also have many custom built modules such as those for communicating with the GPS module and other hardware that is specific to our design.

MySQL

MySQL is a open source relational database that is used to store and access data produced by the detectors. MySQL provides a convient command line interface and allows database replicaiton for .The database scheema an structure is described in its own section.

Polymer

Polymer is a library designed to make using web-components introduced and accepted in the latest web standards easy to use and implement. Web components allow html imports allowing web front-ends to be built modularly. It also includes a library of web components that follow Google's material design guidleines presenting a consistant style. With HTML imports, the HTML of the page is able to only fetch what it needs, and speeds up page loading. It also allows easy reuse of sections designed. Polymer also includes a data-binding system allowing communication between provided and custom built modules.

NGINX

NGINX is a high preformance low resource web server. It is being used to not only serve the static content, but also to proxy-pass the requests for dynamic content and load balancne them between the intacnes running in other containers.

Firebase

Firebase provides a full backend server management interface as well as hosting, database, and other tools. We wil be primarily using it for its user authentication. It intergrates easily into polymer, and can provide security tokens to users.

Slack

Slack is a business focused highly intergrateable instant messaging platform designed to replace fragmented email communication. We will be using it for sending the state of the machine and getting its ip address, location and other onformation on boot.

3.1.2 Architecture

The architecture is designed in a manner to maximise the utility of all the modules and components created. Because of this the architecture varies depending on if it is a server, a client, or takes both roles.

3.1.3 Operation

Building and running the system is fairly simple and requires a single command to bring all the docker containers up.

```
cd cosmicNetwork  
sudo docker-compose up --file arm.dockercompose.yml --build
```