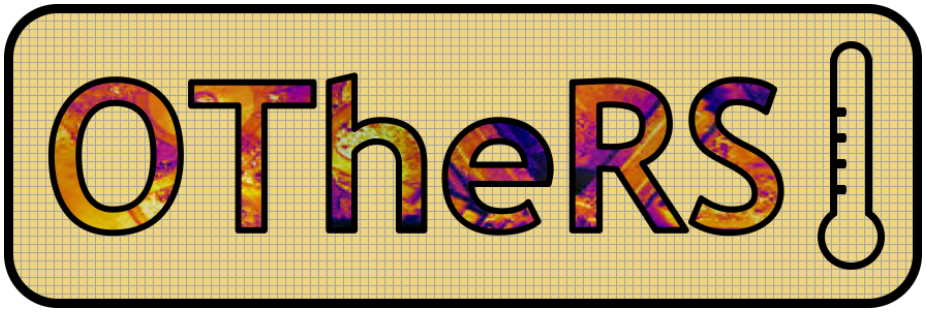
Notebook Supplement: Derivations, Design Files, and Code

Jacob Killelea *

July 4, 2019



*Team OTheRS, SID 105510162

Foreword

This document is meant as a supplement to my Senior Projects notebook. Most of my important work was done on a computer, and this document is meant to capture those efforts. To the best of my ability, this document represents exclusively my own work, with no one else's assistance except for technical consultation. However, this document necessarily refers to outside sources, datasheets, library functions, and math that I did not come up with. In addition, editing and compilation removes context. I've added some background in the beginning of each section in italics. I have *not* included work that ultimately did not contribute to the project, such as rewrites of some of the software that went nowhere, simluations that turned out not to be relevant, and PCB designs that were never used. Notable inclusions of outside information include the equations used to calculate thermistor uncertainty, the table of thermistor specifications copied from a datasheet, and register addresses for some of the chip drivers. However, in all those cases, I've transcribed them into my own original documents or code.

Best,
Jacob Killelea

Contents

1 Thermistor Error Calculations - Error propagation equations from Prof. Nabity

I took this as an opportunity to practice calculating the error in an instrumentation system that I know well. This wasn't necessarily an essential project element, but it gave me the confidence that I was constructing a well designed system.

Thermistor Error Derivation and Analysis

Since the thermistors are our reference system and provide truth data, they need to be well characterized so that we can be confident in the data we receive. The thermistors are arranged in a resistor divider system with an $18k\Omega$ pullup resistor denoted as R_1. The voltage measurement is taken at the junction between the pullup resistor and the thermistor.

Error sources and propagation to R_{th}

The resistor divider equation gives the equation for voltage measured (V_{in}) as a function of supply voltage (V_{supply}) and temperature (T):

$$V_{in} = V_{supply}\frac{R_{th}}{R_{th} + R_1} \tag{1}$$

Where $R_{th} = f(T)$.
Rearranging for R_{th}:

$$R_{th} = \frac{R_1}{V_{in}/V_{supply} - 1} \tag{2}$$

Now we can solve for the thermistor temperature as a function of the measured voltage using the thermistor equation:

$$\frac{1}{T} = \frac{1}{T_{25}} + \frac{1}{B_{25}} ln(\frac{R_{th}}{R_{25}}) \tag{3}$$

Where $T_{25} = 25°C = 298.15K$, R_{25} is the thermistor's resistance at $25°C$, equivalent to $10k\Omega$, and B_{25} is an empirical parameter supplied by the thermistor's manufacturer.

While we can take T_{25} as a precise measurement, R_{25}, B_{25}, and R_{th} have associated uncertainty. Error in the temperature measurement creeps in from these sources, including noise in the supply voltage, additional noise on the input line, deviations in the thermistor response, and imperfections in R_1, which is only constructed to some certain tolerance (we have selected 1% precision resistors here).

Because R_{th} is a derived parameter, the uncertainty in a measurement of R_{th} is the root-square sum of the sensitivity of R_{th} to a given parameter times the error in that parameter.

$$U_{R_{th}} = \sqrt{\sum_x (\frac{\partial R_{th}}{\partial x} \delta x)^2} \tag{4}$$

Partial derivatives of equation 2 with respect to its input parameters:

$$\frac{\partial R_{th}}{\partial V_{in}} = -\frac{R_1}{V_{supply}(V_{in}/V_{supply} - 1)^2} \tag{5}$$

$$\frac{\partial R_{th}}{\partial V_{supply}} = \frac{R_1 V_{in}}{V_{supply}^2(V_{in}/V_{supply} - 1)^2} \tag{6}$$

$$\frac{\partial R_{th}}{\partial R_1} = \frac{1}{V_{in}/V_{supply} - 1} \tag{7}$$

Each of these parameters has an associated error that either comes from biases (constant offsets) and random phenomena that need to be understood.

V_{in} has some bias error from the ADC that measures its value, about 0.4% of the reading. Is also vulnerable to noise. We estimate that the noise is about $1mV$.

The error in a direct measurement of V_{in} is:

$$\delta V_{in} = \frac{t}{\sqrt{N}} \sqrt{\sum_i (\delta S_i)^2} + \sqrt{\sum_j (\delta B_j)^2} \tag{8}$$

Where δS errors are precision errors and δB errors are bias errors. N is the number of measurements being taken and t comes from Student's t-distribution table for a given number of measurements.

N	t
1	12.71
3	3.182
10	2.228
100	1.984

Table 1: t distribution vs number of samples N for 95% confidence

δR_1 is much easier. Since we have 1% precision resistors with no randomness:

$$\delta R_1 = 1\% R_1 = 1\% 18k\Omega = 180\Omega \tag{9}$$

We'll also assume that δV_{supply} also has a random error of $1mV$ and no bias error, since it is easy to take a measurement of V_{supply} that is accurate to less than $1mV$.

In addition, the thermistor has its own error, δR_{th}. Nominally, this is only 1% of the thermistor's value but it increases as temperature moves away from $25°C$.

Total uncertainty in R_{th} is the root-square sum of all these terms.

$$U_{R_{th}} = \sqrt{(\delta R_{th})^2 + (\frac{\partial R_{th}}{\partial V_{in}}\delta V_{in})^2 + (\frac{\partial R_{th}}{\partial V_{supply}}\delta V_{supply})^2 + (\frac{\partial R_{th}}{\partial R_1}\delta R_1)^2} \quad (10)$$

Error propagation to T

Equation 3 is repeated here for clarity:

$$\frac{1}{T} = \frac{1}{T_{25}} + \frac{1}{B_{25}} ln(\frac{R_{th}}{R_{25}}) \quad (11)$$

Raising both sides to -1 to find a direct equation for T:

$$T = [\frac{1}{T_{25}} + \frac{1}{B_{25}} ln(\frac{R_{th}}{R_{25}})]^{-1} \quad (12)$$

Again, T is a derived parameter and the uncertainty in it depends on its sensitivity to its parameters, R_{th}, R_{25}, and B_{25}.

$$\frac{\partial T}{\partial B_{25}} = \frac{ln(R_{th}/R_{25})}{B_{25}^2(\frac{1}{T_{25}} + \frac{ln(R_{th}/R_{25})}{B_{25}})^2} \quad (13)$$

$$\frac{\partial T}{\partial R_{25}} = \frac{1}{B_{25}R_{25}(\frac{1}{T_{25}} + \frac{ln(R_{th}/R_{25})}{B_{25}})^2} \quad (14)$$

$$\frac{\partial T}{\partial R_{th}} = \frac{-1}{B_{25}R_{th}(\frac{1}{T_{25}} + \frac{ln(R_{th}/R_{25})}{B_{25}})^2} \quad (15)$$

The uncertainty in R_{th} was described in detail above, but it can be summarized as:

$$U_{R_{th}} = f(R_{th}, V_{in}, V_{supply}, R_1) \quad (16)$$

R_{25} is the thermistor's temperature at 25°. The manufacturer specifies it to have 1% uncertainty here:

$$\delta R_{25} = 1\% \times 10k\Omega = 100\Omega \quad (17)$$

B_{25} is a parameter from the datasheet and the uncertainty is manufacturer specified:

$$\delta B_{25} = 1\% \times 3435K = 34.35K \quad (18)$$

The uncertainty in T is again the root-square sum of these terms:

$$\delta T = \sqrt{(\frac{\partial T}{\partial R_{th}}\delta R_{th})^2 + (\frac{\partial T}{\partial R_{25}}\delta R_{25})^2 + (\frac{\partial T}{\partial B_{25}}\delta B_{25})^2} \quad (19)$$

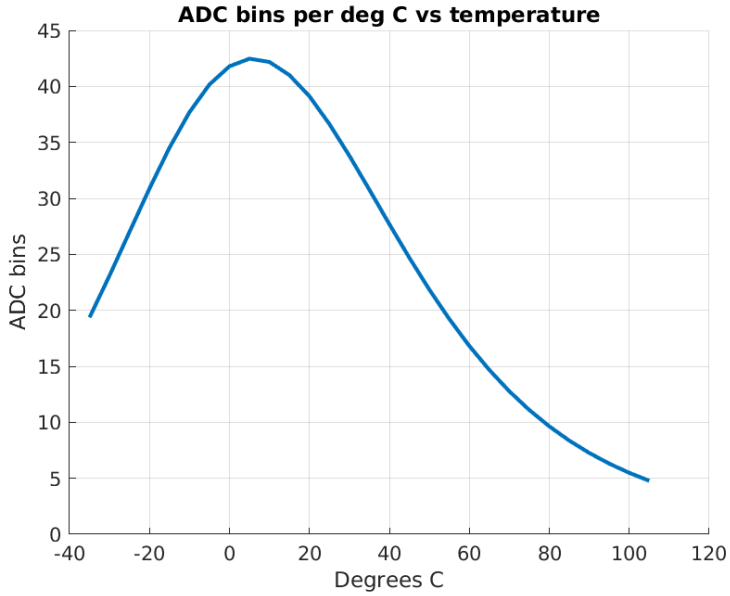


Figure 1: Sensitivity of the ADC output to changes in temperature

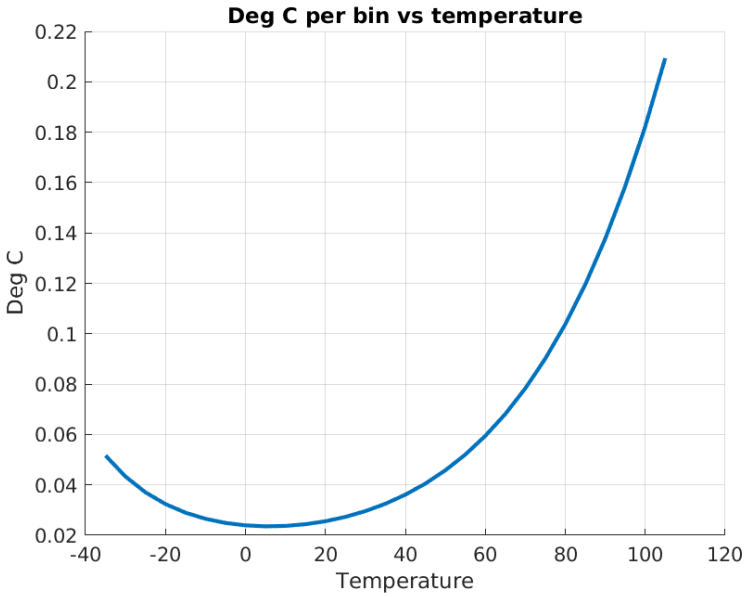


Figure 2: Resolution available vs temperature

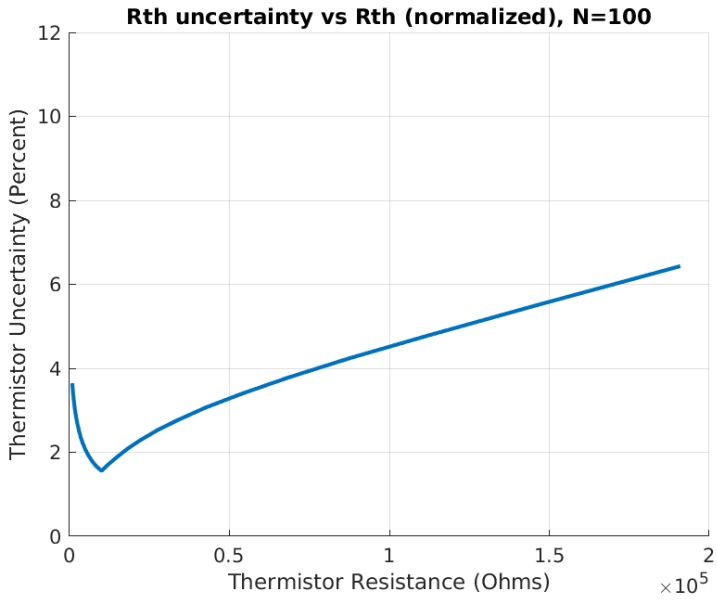


Figure 3: Thermistor value uncertainty over resistance range

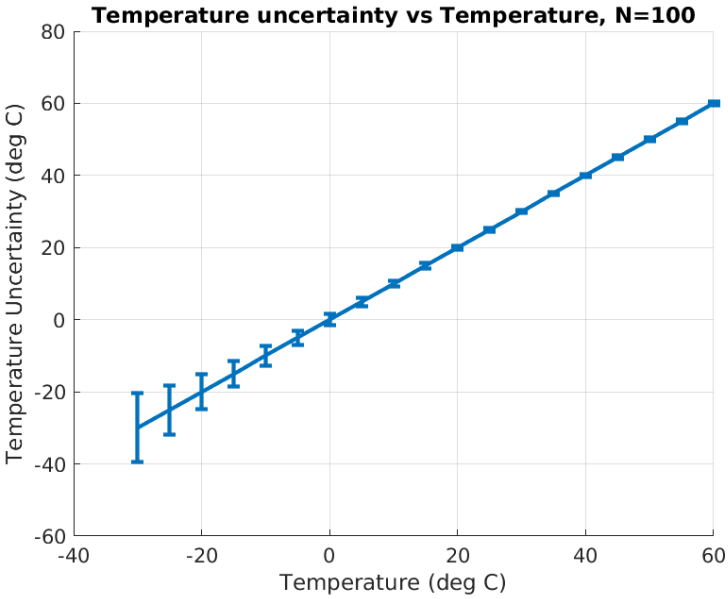


Figure 4: Overall temperature uncertainty

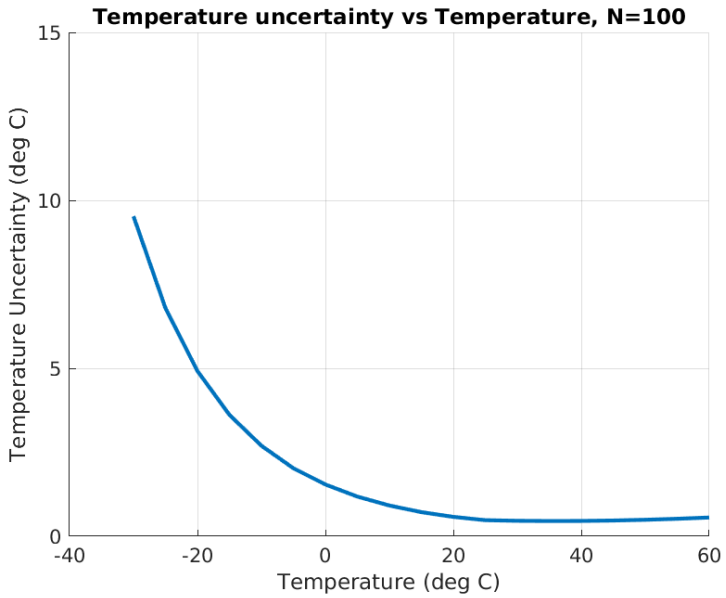


Figure 5: Overall temperature uncertainty vs temperature

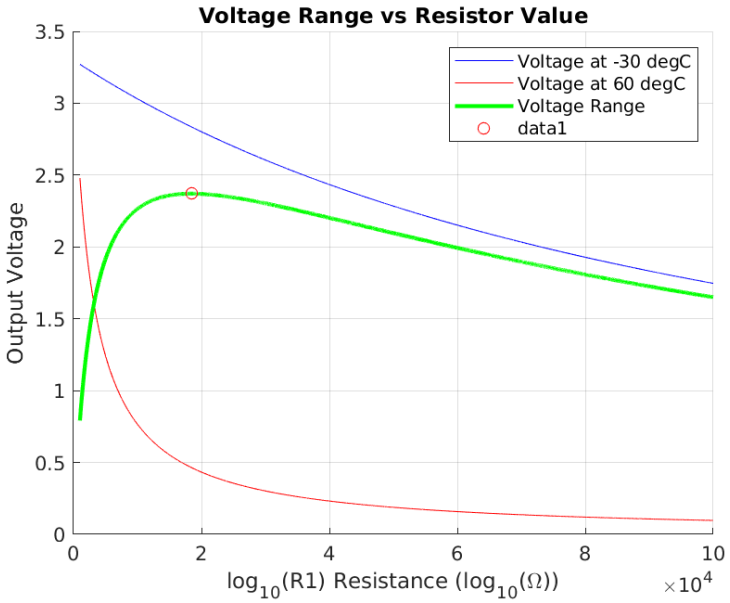


Figure 6: Choosing the pull-up resistor value to give the maximum voltage swing over the temperature range of interest

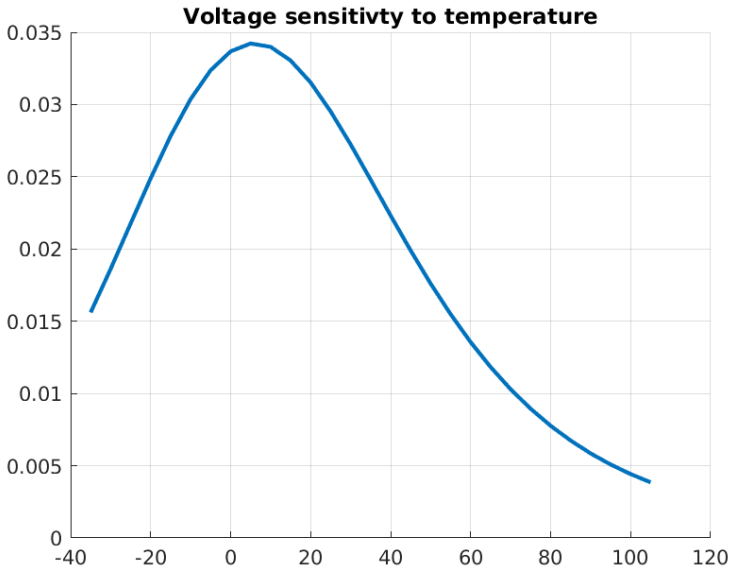


Figure 7: Change in voltage per degree of change in temperature

1.1 thermistor.m

```
% Find the optimimum second resistor for a 10K thermistor
clear all; clc; close all;

Vin           = 3.3;
Tmax          =  60;
Tmin          = -30;
adc_bits      = 12;
adc_bins      = 2^adc_bits - 1;
volts_per_bin = Vin / adc_bins;

R25 = 10e3; % Ohms
B25 = 3435; % Kelvin
T25 = 273.15 + 25; % 25 degC in Kelvin

% Load data between Tmin and Tmax
data = load('./thermistor_spec');
idx  = Tmin <= data(:, 1) & data(:, 1) <= Tmax;
data = data(Tmin <= data(:, 1) & data(:, 1) <= Tmax, :);
resistance_range = linspace(1e3, 1e5, 1e4);

min_max_temp_reading = zeros(length(resistance_range), 2);

for i = 1:length(resistance_range)
    R1 = resistance_range(i);
% -30 degC
```

```
    Rth = data(1, 2);
    output_voltage = Vin*(Rth/(Rth+R1));
    min_max_temp_reading(i, 1) = output_voltage;

    % 60 degC
    Rth = data(end, 2);
    output_voltage = Vin*(Rth/(Rth+R1));
    min_max_temp_reading(i, 2) = output_voltage;
end

% Find the resistor R1 that gives the widest voltage range across the relevant temperatures
voltage_range    = min_max_temp_reading(:, 1) - min_max_temp_reading(:, 2);
[maxval, idx]    = max(voltage_range);
ideal_resistance = resistance_range(idx);

fprintf('Ideal resistance %.2f kOhms\n', ideal_resistance/1000);

f = figure; hold on; grid on;
title('Voltage Range vs Resistor Value');
xlabel('log_{10}(R1) Resistance (log_{10}(\Omega))');
ylabel('Output Voltage');
semilogx(resistance_range, min_max_temp_reading(:, 1), 'b', ...
                        'displayname', 'Voltage at -30 degC')
semilogx(resistance_range, min_max_temp_reading(:, 2), 'r', ...
                        'displayname', 'Voltage at 60 degC')
semilogx(resistance_range, voltage_range, 'g', ...
                        'linewidth', 2, ...
                        'displayname', 'Voltage Range')
semilogx(ideal_resistance, maxval, 'ro')
legend('show')
print(f, 'images/voltage_range_vs_resistor_value', '-dpng')

% Calculate resolutions
data          = load('./thermistor_spec');
R1            = 18e3;                       % Chosen resistor spec
Rth           = data(:, 2);                 % Thermistor R
temps         = data(:, 1);                 % Temperatures at which R is measured
Vout          = Vin .* Rth ./ (Rth + R1); % Vout at each temperature
bins_per_volt = 1/volts_per_bin;            % Voltage resolution of the ADC

% calculate senitivity of voltage to temperature
dVdC = zeros(length(Vout), 1);
for i = 2:length(Vout)
    dV = abs(Vout(i) - Vout(i-1));
    dC = abs(temps(i) - temps(i-1));
    dVdC(i) = dV/dC;
end

% Temperature resolution
bins_per_degC = bins_per_volt .* dVdC;
degC_per_bin = 1 ./ bins_per_degC;

f = figure; hold on; grid on;
title('Voltage sensitivty to temperature');
plot(temps(2:end), dVdC(2:end), 'linewidth', 2)
print(f, 'images/voltage_sensitivity_to_temperature', '-dpng')
```

```
f = figure; hold on; grid on;
title('ADC bins per deg C vs temperature');
xlabel('Degrees C');
ylabel('ADC bins');
plot(temps(2:end), bins_per_degC(2:end), 'linewidth', 2)
print(f, 'images/adc_bins_per_deg_C_vs_temperature', '-dpng')

f = figure; hold on; grid on;
title('Deg C per bin vs temperature');
xlabel('Temperature');
ylabel('Deg C');
plot(temps(2:end), degC_per_bin(2:end), 'linewidth', 2)
print(f, 'images/deg_c_per_bin_vs_temperature', '-dpng')

%% Error analysis
% N = number of samples, t = coeff for 95% condfidence interval
N = 1; t = 12.71;
% N = 3; t = 3.128;
% N = 10; t = 2.228;
% N = 100; t = 1.984;

Rth_err    = Rth .* (data(:, 4)./100); % Deviation from correct (percent err times ohms)
R1_unc     = R1 * 0.01;                % 1% error
Vin_unc    = 0.001;                    % 1mV? TODO: Measure supply voltage noise
Vout_bias  = 0.004 .* Vout;            % (-0.6% to +0.1%) + 1 mV noise
Vout_noise = 0.001;                    % 1 mV TODO: Measure output voltage noise

dRthdVin  = -R1 ./ (Vout .* (Vin./Vout - 1).^2);
dRthdVout = R1*Vin ./ ((Vin./Vout - 1).^2 .* Vout.^2);
dRthdR1   = (Vin./Vout - 1).^-1;

% Vout uncertainty
Vout_unc = (t/sqrt(N)) * sqrt(Vout_noise.^2) + sqrt(Vout_bias.^2);

% Rth uncertainty
Rth_unc = sqrt( (Rth_err).^2               ...
              + (dRthdVin .* Vin_unc).^2   ...
              + (dRthdVout .* Vout_unc).^2 ...
              + (dRthdR1 .* R1_unc).^2     ...
              );

f = figure; hold on; grid on;
title(sprintf('Rth uncertainty vs Rth (normalized), N=%d', N))
ylim([0 12]);
xlabel('Thermistor Resistance (Ohms)');
ylabel('Thermistor Uncertainty (Percent)');
plot(Rth, 100*Rth_unc./Rth, 'linewidth', 2)
print(f, sprintf('images/rth_uncertainty_vs_rth_normalized_N%d', N), '-dpng')

% Temperature error
% Emperical fit
T = @(Rth) (1/T25 + log(Rth/R25)/B25).^-1 - 273.15;

% Sensitivities
dTdB25 = log(Rth/R25) ./ (B25^2 * (1/T25 + log(Rth/R25)/B25).^2);
dTdR25 =  1           ./ (B25*R25*(1/T25 + log(Rth/R25)/B25).^2);
dTdRth = -1           ./ (B25*R25*(1/T25 + log(Rth/R25)/B25).^2);
```

```
% Uncertainty in terms
B25_unc  = 0.01 * B25; % 1%, Kelvin
R25_unc  = 0.01 * R25; % 1%, Ohms

% Temperature Uncertainty
T_unc = sqrt((dTdB25 .* B25_unc).^2 ...
               + (dTdR25 .* R25_unc).^2 ...
               + (dTdRth .* Rth_unc  ).^2);

f = figure; hold on; grid on;
title(sprintf('Temperature uncertainty vs Temperature, N=%d', N))
ylim([0, 15]);
xlabel('Temperature (deg C)');
ylabel('Temperature Uncertainty (deg C)');
idx = -30 <= temps & temps <= 60;
plot(temps(idx), T_unc(idx), 'linewidth', 2)
print(f, sprintf('images/temp_uncertainty_vs_temp_N%d', N), '-dpng')

f = figure; hold on; grid on;
ylim([-60, 80]);
title(sprintf('Temperature uncertainty vs Temperature, N=%d', N))
xlabel('Temperature (deg C)');
ylabel('Temperature Uncertainty (deg C)');
idx = -30 <= temps & temps <= 60;
errorbar(temps(idx), temps(idx), T_unc(idx), 'linewidth', 2)
print(f, sprintf('images/temp_errorbar_vs_temp_N%d', N), '-dpng')

% Sanity checks
assert(N > 0);
assert(all(Rth_unc > 0), 'Rth_unc < 0');
assert(all(T_unc > 0), 'T_Unc < 0');
```

1.2 Thermistor Specification - Values copied from from datasheet - Vishay document 29078

```
% NTCLE413-428 10k 1% NTC thermistor, R25 = 10kOhms +/- 1%, B25 = 3435K +/- 1%
% T (degC) Rt (Ohms) Rt/R25   R tol (%) alpha (%/K) T-tol (+/- degC) Rmin (Ohm) Rmax (Ohm)
-40.0      190953    19.095   4.24     -5.46     0.78   182848     199057
-35.0      145953    14.595   3.93     -5.30     0.74   140213     151693
-30.0      112440    11.244   3.63     -5.14     0.71   108354     116526
-25.0      87285     8.7285   3.35     -4.99     0.67   84364      90206
-20.0      68260     6.8260   3.07     -4.85     0.63   66164      70355
-15.0      53762     5.3762   2.80     -4.71     0.60   52254      55270
-10.0      42636     4.2636   2.55     -4.57     0.56   41549      43723
-5.0       34038     3.4038   2.30     -4.44     0.52   33254      34822
0.0        27348     2.7348   2.07     -4.31     0.48   26783      27913
5.0        22108     2.2108   1.84     -4.19     0.44   21702      22515
10.0       17979     1.7979   1.62     -4.08     0.40   17689      18270
15.0       14706     1.4706   1.40     -3.96     0.35   14499      14912
20.0       12094     1.2094   1.20     -3.86     0.31   11949      12239
25.0       10000     1.0000   1.00     -3.75     0.27   9900.0     10100
30.0       8310.8    0.83108  1.19     -3.65     0.33   8211.7     8409.8
35.0       6941.1    0.69411  1.38     -3.55     0.39   6845.5     7036.7
40.0       5824.9    0.58249  1.56     -3.46     0.45   5734.1     5915.6
```

45.0	4910.6	0.49106	1.73	-3.37	0.51	4825.6	4995.7
50.0	4158.3	0.41583	1.90	-3.28	0.58	4079.2	4237.3
55.0	3536.2	0.35362	2.06	-3.20	0.65	3463.2	3609.2
60.0	3019.7	0.30197	2.22	-3.12	0.71	2952.5	3086.8
65.0	2588.8	0.25888	2.38	-3.04	0.78	2527.3	2650.4
70.0	2228.0	0.22280	2.53	-2.96	0.85	2171.7	2284.3
75.0	1924.6	0.19246	2.67	-2.89	0.92	1873.1	1976.0
80.0	1668.4	0.16684	2.81	-2.82	1.00	1621.5	1715.3
85.0	1451.3	0.14513	2.95	-2.75	1.07	1408.5	1494.2
90.0	1266.7	0.12667	3.08	-2.69	1.15	1227.7	1305.8
95.0	1109.2	0.11092	3.21	-2.62	1.22	1073.6	1144.8
100.0	974.26	0.097426	3.34	-2.56	1.30	941.74	1006.8
105.0	858.33	0.085833	3.46	-2.50	1.38	828.62	888.04

2 Maximum SPI Wire Length Calculations

The very high frequency of the Lepton's SPI communications make the choice of wiring and the length of wire important considerations. This section outlines the maximum theoretical wire length that we could hope to achieve. While my actual test results didn't line up perfectly, knowing the upper limits set by physics was an important step in proving this system.

2.1 Explanation

SPI (Serial Peripheral Interface) is a synchronous protocol for transferring data from one device to another. It is fast and very simple, but it's designed for use on PCBs, not the long lengths of wire that we are using for the Lepton cameras.

The signaling protocol for one-way communication is as follows:

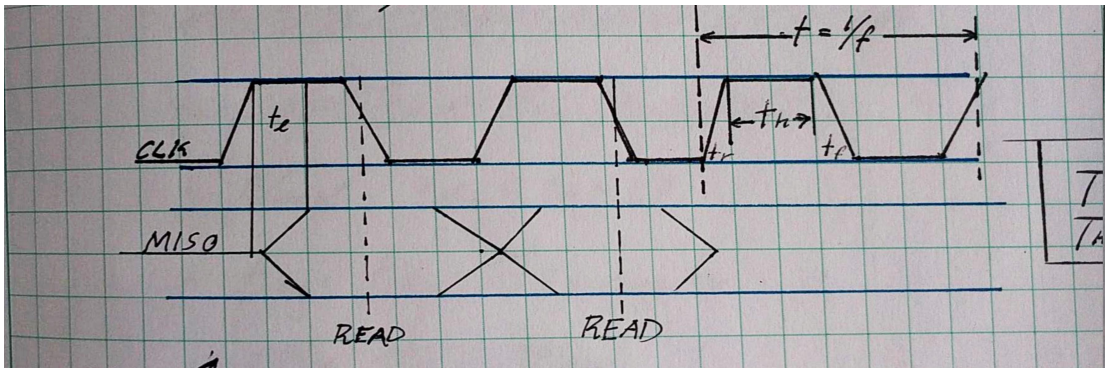


Figure 8: A picture from my notebook, page 101

Here, the master device asserts the Chip Select (CS) line low (not shown), then asserts the Clock (CLK) line high. The slave device places the next bit of data on the Master-In-Slave-Out (MISO) line on the rising edge of CLK, and the master then pulls CLK low as it reads the bit. The time taken for the slave to place the bit on the line and for the voltage to travel to the master

device is therefore half the clock period. Because the FLIR Leptons operate off a nominally 20MHz SPI clock, only 25 nanoseconds are available for the slave to see the rising CLK line, respond, and for the master device to see the change on the MISO line.

The propagation delay of electricity in copper wire is approximately $t_{prop} = 5ns/meter$, and the signal must travel this distance twice, plus whatever time the Lepton actually needs to respond to the signal. Therefore:

$$t_{delay} = 2lt_{prop} + t_{lepton} \leq \frac{1}{2f} \tag{20}$$

This informs the maximum theoretical length of wire that can be used with the SPI protocol. However, what we have found is that electrical cross talk between the clock and signal lines becomes overwhelming first and this is not the practical limit unless more advanced techniques are used to limit electrical noise.

2.2 clockskew.m

```
clear all; clc; close all;

propigation_delay = 5e-9; % 5 ns/meter
freq = 4e6:1e3:20e6;
chip_time = 9e-9;         % time for the slave to respond to the rising clock edge

longwire_results = [
% Wire length, SPI Speed (Hz), success
4.5,           20e6,           0
4.5,           10e6,           0
4.5,            8e6,           0
4.5,            4e6,           0

4,             20e6,           0
4,             10e6,           0
4,              8e6,           0
4,              4e6,           0

3.5,           20e6,           0
3.5,           10e6,           0
3.5,            8e6,           0
3.5,            4e6,           0

3,             20e6,           0
3,             10e6,           0
3,              9e6,           0
3,              8e6,           0
3,              4e6,           0

2,             20e6,           0
2,             15e6,           0
2,             10e6,           0
2,              9e6,           0
```

```
1.5,            20e6,           0
1.5,            15e6,           1
1.5,            14e6,           1
1.5,            13e6,           1
1.5,            12e6,           0
1.5,            11e6,           0
1.5,            10e6,           0

0.5,            20e6,           0
0.5,            15e6,           1
];


figure; hold on; grid on;
for chip_time = (0:5:20)*1e-9
    len = ( (1./(2*freq)) - chip_time ) ./ (2*propigation_delay);
    loglog(freq, len, 'linewidth', 2, 'displayname', sprintf('%f ns', chip_time*1e9));
end

% plot failures
failures = longwire_results((longwire_results(:,  3) == 0), :);
scatter(failures(:, 2), failures(:, 1), 'r*', 'displayname', 'SPI failures');

% plot successes
successes = longwire_results((longwire_results(:,  3) == 1), :);
scatter(successes(:, 2), successes(:, 1), 'g*', 'displayname', 'SPI success');


set(gca, 'XScale', 'log', 'YScale', 'log');
title('Max line length for SPI clock skew w/ variable slave delay');
xlabel('SPI Frequency [Hz]');
ylabel('Line Length [m]');
legend('show', 'location', 'southwest');
```

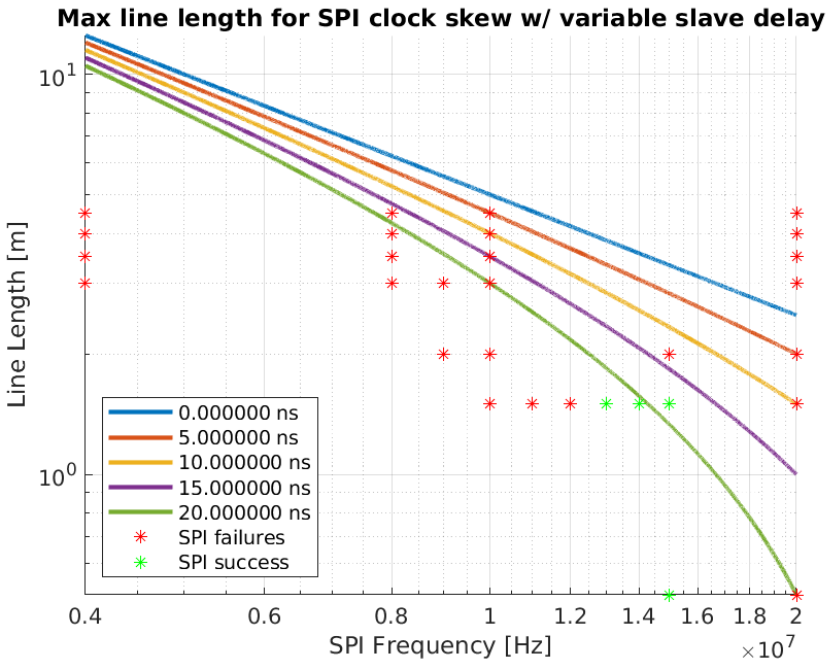


Figure 9: Predicted upper limits on SPI wire length (Y axis) and bus frequency (X axis) for various levels of internal delay in the FLIR Lepton. Red dots are failed test points and the green dots are successes, but I'm going to print this in black and white so you might have to take my word for it.

3 Code

3.1 Raw Capture

This may be the single most important thing I did for this project.
The single most important thing.
The thing that was the most important.
That thing.
It's the utility that grabs a single still image off the Lepton cameras. It should have been finished months before it was - it probably should have already existed and we should have checked for it when trading on cameras. Before this software, we had no way to automatically grab images from the cameras. We were clicking 'capture' by hand on a video application whenever we wanted a new image. The deliverable here is small - a few hundred lines of code. However, those lines took me a very long time to get right. The Lepton is meant for video streams viewed by humans, so little glitches and out occasional out of order sections aren't a big deal normally, but they're a huge problem when trying to capture a single still image. I tried the "keep it simple, stupid" approach and failed. Just reading in

the pixels isn't enough, this program has to be smart enough to recognize when things are out of order, how to rearrange block of pixels that have already been received and stored once a sync message is received, and how to detect when the transmission is incomplete or broken in ways that can't easily be fixed.

3.1.1 raw_capture.c

```
// A utility for capturing raw images from a FLIR Lepton 3.5
// call like: ./raw_capture 1 0 > image &&  ../../scripts/parse_txt_image.py image
// 'image' is a grid of numbers in text form
// returns 0 on success
// returns -1 if not all image segments are in order
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#include <LEPTON_SDK.h>
#include <LEPTON_OEM.h>
#include <LEPTON_SYS.h>
#include <LEPTON_RAD.h>
#include <LEPTON_Types.h>
#include <LEPTON_ErrorCodes.h>

#include "util.h"
#include "global_vars.h"
#include "packet.h"
#include "segment.h"

#define LOG(...)
// #define LOG(...) fprintf(stderr, __VA_ARGS__)

// I2C vars
uint16_t i2c_number = -1;
LEP_CAMERA_PORT_DESC_T i2c_port;

// SPI protocol vars
int      spi_fd    = -1;
int      spi_speed = 15000000;
uint8_t  spi_mode  = SPI_MODE_3;
char     spi_path[255];
uint8_t  spi_bits_per_word = 8;

// SPI communication vars
uint8_t spi_data[PACKET_SIZE] = {0x0};
```

```
// Image is made of 4 segments that are each made of 60 packets
segment_t segments[4];

void read_image() {
    uint32_t mismatches = 0; // number of times gotten out of sync
    segment_t segment;

    for (int32_t seg = 1; seg <= NUM_SEGMENTS; seg++) {
        for (int32_t pak = 0; pak < PACKETS_PER_SEGMENT; pak++) {
            packet_t *packet_ptr = &segment.packets[pak];

            if (read(spi_fd, spi_data, PACKET_SIZE) != PACKET_SIZE) // Read SPI
                LOG("SPI failed to read enough bytes!\n");

            // parse the bytes into a packet_t struct
            packet_parse(spi_data, packet_ptr); // NOTE: memory copy

            if (!packet_ptr->valid) { // handle drop packets
                LOG("drop (%x)\n", spi_data[0]);
                pak--;
                continue;
            }

            LOG("expected %d.%2d got %d.%2d\n", seg, pak,
                                                packet_ptr->segment_no,
                                                packet_ptr->packet_no);

            if (0 <= packet_ptr->packet_no && packet_ptr->packet_no < PACKETS_PER_SEGMENT)
                pak  = packet_ptr->packet_no;

            if (pak != packet_ptr->packet_no) { // out of sync
                LOG("mismatch %d\n", mismatches);
                pak = -1;
                mismatches++;
                usleep(1000);
                if (mismatches == 100) {
                    LOG("100 mismatches - resetting\n");
                    mismatches = 0;
                    close(spi_fd);
                    usleep(200*1000);
                    open_spi_port(spi_path);
                }
                continue;
            }

            // segment number is only valid on packet 20
            if (packet_ptr->packet_no == 20) {
                LOG("Expected segment %d, got %d\n", seg, packet_ptr->segment_no);
                if (packet_ptr->segment_no == 0) {
                    LOG("Invalid segment number. Going back to 1\n");
                    seg = 1;
                    continue;
                } else if (1 <= packet_ptr->segment_no
                             && packet_ptr->segment_no <= NUM_SEGMENTS) {
                    LOG("Resetting segment number.\n");
                    seg                = packet_ptr->segment_no;
                    segment.segment_no = packet_ptr->segment_no;
```

```
                }
            }

            segments[seg-1] = segment; // NOTE: memory copy
        }
        usleep(1000/106);
    }
}

int main(int argc, char *argv[]) {
    // setup
    // parse opts
    if (argc < 3) {
        fprintf(stderr, "Usage: %s i2c-number spi-number\n", argv[0]);
        return -1;
    }

    // I2C bus number
    switch (atoi(argv[1])) {
        case 0:
            i2c_number = 0;
            break;
        case 1:
            i2c_number = 1;
            break;
        default:
            pabort("Need to define I2C number as 0 or 1");
    }

    // Lepton config
    if (LEP_OpenPort(i2c_number, LEP_CCI_TWI, 400, &i2c_port) != LEP_OK)
        pabort("Couldn't open i2c port!");

    // Disable telemetry (changes packet lengths)
    if (LEP_SetSysTelemetryEnableState(&i2c_port, LEP_TELEMETRY_DISABLED) != LEP_OK)
        pabort("Couldn't disable telemetry!");

    // Enable radiometry
    if (LEP_SetRadEnableState(&i2c_port, LEP_RAD_ENABLE) != LEP_OK)
        pabort("Couldn't enable radiometry!");

    // SPI bus number
    if(set_spi_number(argv[2]) < 0)
        return -1;

    spi_fd = open_spi_port(spi_path);

    // Read the image
    for (uint32_t i = 0; i < 5; i++) {
        read_image();
    }

    LOG("Got segment numbers ");
    for (uint32_t seg = 0; seg < 4; seg++)
        LOG("%d ", segments[seg].segment_no);
    LOG("\n");
```

```
    // Do we have every segment in the right order
    bool transmission_ok = true;
    for (uint32_t seg = 0; seg < 4; seg++)
        transmission_ok = transmission_ok && (segments[seg].segment_no == seg+1);

    if (transmission_ok)
        fprintf(stderr, "OK\n");
    else
        fprintf(stderr, "ERROR\n");

    for (uint32_t seg = 0; seg < 4; seg++) {
        segment_t *segment = &segments[seg];
        for (uint32_t pak = 0; pak < 60; pak++) {
            packet_t *packet = &segment->packets[pak];
            for (uint32_t i = 0; i < 80; i++) {
                printf("%d ", packet->data[i]);
            }
            if (pak % 2 == 1)
                printf("\n");
        }
    }

    if (transmission_ok)
        return 0;
    return -1;
}
```

3.1.2 packet.h

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#ifndef PACKET_H_
#define PACKET_H_
#include <stdint.h>
#include <stdbool.h>

// TODO: memory alignment?
typedef struct packet {
    bool     valid;
    int32_t  segment_no; // -1 on all packets but number 20
    int32_t  packet_no;
    uint16_t data[80];
} packet_t;

void packet_parse(uint8_t *raw_data, packet_t *packet);

#endif // PACKET_H_
```

3.1.3 packet.c

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#include "packet.h"
#include <string.h>

void packet_parse(uint8_t *raw_data, packet_t *packet) {
    packet->valid      = !((raw_data[0] & 0x0F) == 0x0F);
```

```
    packet->packet_no  = raw_data[1];

    if (packet->packet_no == 20) // only valid on packet 20
        packet->segment_no = raw_data[0] >> 4;
    else
        packet->segment_no = -1;

    for (uint32_t i = 0; i < 80; i++) {
        size_t idx = 2*i + 4;
        packet->data[i] = (raw_data[idx] << 8) | raw_data[idx+1];
    }
}
```

3.1.4 segment.h

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#ifndef SEGMENT_H_
#define SEGMENT_H_

#include "packet.h"

typedef struct segment {
    uint32_t segment_no;
    packet_t packets[60];
} segment_t;

void segment_append_packet(segment_t *seg, packet_t *pak);

#endif // SEGMENT_H_
```

3.1.5 segment.c

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#include "segment.h"
#include <stdint.h>
#include <string.h>

void segment_append_packet(segment_t *seg, packet_t *pak) {
    seg->packets[pak->packet_no] = *pak;
}
```

3.1.6 util.h

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#ifndef _RAW_CAPTURE_UTIL_H_
#define _RAW_CAPTURE_UTIL_H_

#include <stdio.h>
#include <fcntl.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#include <LEPTON_SDK.h>
#include <LEPTON_OEM.h>
#include <LEPTON_SYS.h>
#include <LEPTON_RAD.h>
#include <LEPTON_Types.h>
#include <LEPTON_ErrorCodes.h>


// Functions
void pabort(const char *s);
int open_spi_port(const char *path);
int set_spi_number(const char *arg);

#endif // _RAW_CAPTURE_UTIL_H_
```

3.1.7 util.c

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#include "util.h"
#include "global_vars.h"

// read argv for spi number (should be 0 or 1)
int set_spi_number(const char *arg) {
    int spi_number = atoi(arg);
    switch (spi_number) { // SPI bus number
        case 0:
            strcpy(spi_path, "/dev/spidev0.0");
            break;
        case 1:
            strcpy(spi_path, "/dev/spidev0.1");
            break;
        default:
            fprintf(stderr, "Invalid SPI number %d. Options are 0 or 1.\n",
                            spi_number);
            return -1;
    }
    return 0;
}

// Open SPI port and do all the ioctl setup
int open_spi_port(const char *path) {
    int ret;
    int spi_fd = open(path, O_RDWR);
    if (spi_fd < 0)
        pabort("can't open device");

    ret = ioctl(spi_fd, SPI_IOC_WR_MODE, &spi_mode);
    if (ret == -1)
        pabort("can't set spi mode");

    ret = ioctl(spi_fd, SPI_IOC_RD_MODE, &spi_mode);
```

```
    if (ret == -1)
        pabort("can't get spi mode");

    ret = ioctl(spi_fd, SPI_IOC_WR_BITS_PER_WORD, &spi_bits_per_word);
    if (ret == -1)
        pabort("can't set bits per word");

    ret = ioctl(spi_fd, SPI_IOC_RD_BITS_PER_WORD, &spi_bits_per_word);
    if (ret == -1)
        pabort("can't get bits per word");

    ret = ioctl(spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &spi_speed);
    if (ret == -1)
        pabort("can't set max speed hz");

    ret = ioctl(spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &spi_speed);
    if (ret == -1)
        pabort("can't get max speed hz");
    return spi_fd;
}

// Exit program with error message
void pabort(const char *s) {
    perror(s);
    abort();
}
```

3.1.8 global_vars.h

```
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#ifndef _RAW_CAPTURE_GLOBAL_VARS_H_
#define _RAW_CAPTURE_GLOBAL_VARS_H_

// Defines
#define IMAGE_WIDTH (160)
#define IMAGE_HEIGHT (120)
#define NUM_SEGMENTS (4)
#define PACKETS_PER_SEGMENT (60)
#define PACKETS_PER_FRAME (PACKETS_PER_SEGMENT*NUM_SEGMENTS)
#define PACKET_SIZE (164)
#define PACKET_SIZE_UINT16 (164/2)

extern uint16_t *image_ptr;

// I2C vars
extern uint16_t i2c_number;
extern LEP_CAMERA_PORT_DESC_T i2c_port;

// SPI protocol vars
extern int      spi_fd;
extern int      spi_speed;
extern uint8_t spi_mode;
extern char     spi_path[255];
extern uint8_t spi_bits_per_word;

// SPI communication vars
```

```
extern uint8_t packet[PACKET_SIZE];

#endif // _RAW_CAPTURE_GLOBAL_VARS_H_
```

3.1.9 restart_camera.c

```
// A utility for capturing raw images from a FLIR Lepton 3.5
// call like: ./raw_capture 1 0 > image &&  ../../scripts/parse_txt_image.py image
// 'image' is a grid of numbers in text form
// returns 0 on success
// returns -1 if not all image segments are in order
// part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

#include <LEPTON_SDK.h>
#include <LEPTON_OEM.h>
#include <LEPTON_SYS.h>
#include <LEPTON_RAD.h>
#include <LEPTON_Types.h>
#include <LEPTON_ErrorCodes.h>

#include "util.h"
#include "global_vars.h"
#include "packet.h"
#include "segment.h"

#define LOG(...)
// #define LOG(...) fprintf(stderr, __VA_ARGS__)

// I2C vars
uint16_t i2c_number = -1;
LEP_CAMERA_PORT_DESC_T i2c_port;

void pabort(const char *msg) {
    perror(msg);
    abort();
}

int main(int argc, char *argv[]) {
    // parse opts
    if (argc < 2) {
        fprintf(stderr, "Usage: %s i2c-number\n", argv[0]);
        return -1;
    }

    // I2C bus number
```

```
    switch (atoi(argv[1])) {
        case 0:
            i2c_number = 0;
            break;
        case 1:
            i2c_number = 1;
            break;
        default:
            pabort("Need to define I2C number as 0 or 1");
    }

    // Lepton config
    if (LEP_OpenPort(i2c_number, LEP_CCI_TWI, 400, &i2c_port) != LEP_OK)
        pabort("Couldn't open i2c port!");

    if (LEP_RunOemReboot(&i2c_port) != LEP_OK)
        pabort("Couldn't reboot!");

    // LEP_STATUS_T status;
    // LEP_GetSysStatus(&i2c_port, &status);


    // Disable telemetry (changes packet lengths)
    if (LEP_SetSysTelemetryEnableState(&i2c_port, LEP_TELEMETRY_DISABLED) != LEP_OK)
        pabort("Couldn't disable telemetry!");

    // Enable radiometry
    if (LEP_SetRadEnableState(&i2c_port, LEP_RAD_ENABLE) != LEP_OK)
        pabort("Couldn't enable radiometry!");

}
```

3.1.10 Makefile

```
# part of raw_capture. Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>

CC = gcc
CFLAGS = -Wall -march=native
CFLAGS += -O3
# CFLAGS += -g

LIB_PATH = ../raspberrypi_libs/lepton-sdk-fork
CFLAGS += -I$(LIB_PATH)

SDK_LIB = $(LIB_PATH)/Debug/libLEPTON_SDK.a

EXEC = raw_capture

OUTDIR = Build

OBJS = $(OUTDIR)/raw_capture.o $(OUTDIR)/util.o  $(OUTDIR)/packet.o $(OUTDIR)/segment.o

.PHONY: all
all: $(EXEC)

$(EXEC): $(OUTDIR) $(OBJS) $(SDK_LIB)
```

```
    $(CC) $(CFLAGS) -o $@ $(OBJS) $(SDK_LIB)

restart_camera: $(OUTDIR) restart_camera.c  $(SDK_LIB)
    $(CC) $(CFLAGS) -o $@ restart_camera.c $(SDK_LIB)


$(OUTDIR)/%.o: %.c
    $(CC) -c $(CFLAGS) -o $@ $^

$(OUTDIR):
    mkdir -p $@

$(SDK_LIB):
    $(MAKE) -C $(LIB_PATH)

.PHONY: clean
clean:
    rm -rf $(OUTDIR)
    rm -rf $(EXEC)

.PHONY: sdkclean
sdkclean:
    $(MAKE) clean -C $(LIB_PATH)
```

3.2 ADC128D818 Driver Library

This chip is on the Raspberry Pi's PCB and reads from the thermistors mentioned in the error derivation section. These chips were also used in the reference data system, but this code was not because I handed off responsibility for that software to Pierre.

3.2.1 adc128.h

```
#pragma once

/*
 * ADC128.h - A libray for the ADC128D818 I2C ADC chip.
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
 */


#include <stdint.h>
#include "adc128_registers.h"

#ifdef ARDUINO
#include <Arduino.h>
#include <Wire.h>
#endif // ARDUINO

#ifdef __linux__
#include <i2cdevice.h>
#include <memory>
#endif // __linux__


class ADC128 {
public:
    ADC128(uint8_t address);

    void begin();

    int enableStart(bool immediate = true);
    int disableStart(bool immediate = true);

    int enableInterrupts(bool immediate = true);
    int disableInterrupts(bool immediate = true);
    int clearInterruptPin(bool immediate = true);
    int enableInterruptPin(bool immediate = true);

    int reset();
    bool is_busy();

    int enableContiniousConversion();
    int disableContiniousConversion();

    int oneshot();
    int setMode1(bool immediate = true);

    int enableExternalVref(bool immediate = true);
    int disableExternalVref(bool immediate = true);

    uint16_t analogRead(uint8_t chan);

private:
    uint8_t addr = 0x00;

    void i2c_bus_init();
    int reg_write(uint8_t reg, uint8_t data);
    uint8_t reg_read(uint8_t reg);
    int writeConfig();
    int writeAdvConfig();

    // configuration register defaults
    // NOTE: these are currently read from the chip and set in begin()
```

```
    uint8_t config_data =  config::start
                         | config::int_clear;

    uint8_t adv_config_data = adv_config::external_vref
                            | adv_config::mode_bit0;

#ifdef __linux__
    std::unique_ptr<I2CDevice> i2c; // Linux i2c backend
#endif // __linux__
};
```

3.2.2 adc128.cpp

```
/*
 * ADC128.h - A libray for the ADC128D818 I2C ADC chip.
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */


#include "adc128.h"
#include "adc128_registers.h"

ADC128::ADC128(uint8_t address) {
    addr = address;
    // I2C Setup
    i2c_bus_init();
}

void ADC128::begin() {
    config_data     = reg_read(reg::config);
    adv_config_data = reg_read(reg::adv_config);

    // reset defaults
    reset();
    // shutdown
    disableStart(true);
    // external vref
    enableExternalVref(false);
    // mode 1
    setMode1(false);
```

```
    // conversion rate -> continious
    enableContiniousConversion();
    // enable all channels
    reg_write(reg::chan_disable, 0x00);
    // mask all interrupts
    reg_write(reg::int_mask, 0xFF);
    // startup
    enableStart(true);
    // turn off all interrupts
    disableInterrupts(false);
    enableInterruptPin(false);
    writeConfig();
    writeAdvConfig();
}


int ADC128::enableStart(bool immediate) {
    config_data |= config::start;
    if (immediate)
        return writeConfig();
    else
        return 0;
}


int ADC128::disableStart(bool immediate) {
    config_data &= ~config::start;
    if (immediate)
        return writeConfig();
    else
        return 0;
}


int ADC128::enableInterrupts(bool immediate) {
    config_data |= config::int_enable;
    if (immediate)
        return writeConfig();
    else
        return 0;
}


int ADC128::disableInterrupts(bool immediate) {
    config_data &= ~config::int_enable;
    if (immediate)
        return writeConfig();
    else
        return 0;
}


int ADC128::clearInterruptPin(bool immediate) {
    config_data |= config::int_clear;
    if (immediate)
        return writeConfig();
    else
```

```
        return 0;
}

int ADC128::enableInterruptPin(bool immediate) {
    config_data &= ~config::int_clear;
    if (immediate)
        return writeConfig();
    else
        return 0;
}


int ADC128::reset() {
    reg_write(reg::config, config::restore_defaults);
}


int ADC128::enableContiniousConversion() {
    return reg_write(reg::conversion_rate, conversion_rate::continious);
}


int ADC128::disableContiniousConversion() {
    return reg_write(reg::conversion_rate, 0);
}


int ADC128::oneshot() {
    return reg_write(reg::oneshot, oneshot::enable);
}

int ADC128::setMode1(bool immediate) {
    adv_config_data |= adv_config::mode1;
    if (immediate)
        return writeAdvConfig();
    else
        return 0;
}

int ADC128::enableExternalVref(bool immediate) {
    adv_config_data |= adv_config::external_vref;
    if (immediate)
        return writeAdvConfig();
    else
        return 0;
}

int ADC128::disableExternalVref(bool immediate) {
    adv_config_data &= ~adv_config::external_vref;
    if (immediate)
        return writeAdvConfig();
    else
        return 0;
}

bool ADC128::is_busy() {
    return (reg_read(reg::busy) & busy::not_ready) > 0;
```

```
}

uint16_t ADC128::analogRead(uint8_t chan) {
    // uint8_t channel_reg = reg::chan0 + chan;
    uint8_t channel_reg = 0x20 + chan;

#ifdef ARDUINO
    Wire.beginTransmission(addr);
    Wire.write(channel_reg);
    Wire.endTransmission();

    Wire.requestFrom(addr, 2);
    while (!Wire.available());

    uint8_t highbyte = Wire.read();
    uint8_t lowbyte = Wire.read() & 0xF0;

    return (highbyte << 4) | (lowbyte >> 4);
#else // Raspberry Pi
    uint8_t buf[2] = {0};
    i2c->write(&channel_reg, 1);
    i2c->read(buf, 2);
    return (buf[0] << 4) | (buf[1] >> 4);
}
#endif

void ADC128::i2c_bus_init() {
#ifdef ARDUINO
    Wire.begin();
#else // Raspberry Pi
    i2c = std::unique_ptr<I2CDevice>(new I2CDevice(addr, "/dev/i2c-0"));
#endif
}

inline int ADC128::reg_write(uint8_t reg, uint8_t data) {
#ifdef ARDUINO
    Wire.beginTransmission(addr);
    Wire.write(reg);
    Wire.write(data);
    return Wire.endTransmission();
#else
    uint8_t buf[] = {reg, data};
    i2c->write(buf, 2);
#endif
}

inline uint8_t ADC128::reg_read(uint8_t reg) {
#ifdef ARDUINO
    Wire.beginTransmission(addr);
    Wire.write(reg);
    Wire.endTransmission();

    Wire.requestFrom(addr, 1);
    while (!Wire.available());
    return Wire.read();
#else
    uint8_t buf[1] = {0};
```

```
    i2c->write(&reg, 1);
    i2c->read(buf, 1);
    return buf[0];
#endif
}

inline int ADC128::writeConfig() {
    return reg_write(reg::config, config_data);
}

inline int ADC128::writeAdvConfig() {
    return reg_write(reg::adv_config, adv_config_data);
}
```

3.2.3 adc128_registers.h - Register addresses copied from datasheet - Texas Instruments document SNAS483F

```
#pragma once

/*
 * ADC128 Registers - all register addresses for the ADC128D818 chip, from TI datasheet
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */


// register addresses
namespace reg {
    // chip configuration registers
    const uint8_t config          = 0x00;
    const uint8_t int_status      = 0x01;
    const uint8_t int_mask        = 0x03;
    const uint8_t conversion_rate = 0x07;
    const uint8_t chan_disable    = 0x08;
    const uint8_t oneshot         = 0x09;
    const uint8_t deep_shutdown   = 0x0A;
    const uint8_t adv_config      = 0x0B;
    const uint8_t busy            = 0x0C;

    // channel reading registers
    const uint8_t chan0 = 0x20;
    const uint8_t chan1 = 0x21;
```

```
    const uint8_t chan2 = 0x22;
    const uint8_t chan3 = 0x23;
    const uint8_t chan4 = 0x24;
    const uint8_t chan5 = 0x25;
    const uint8_t chan6 = 0x26;
    const uint8_t chan7 = 0x27;

    // channel limit registers
    const uint8_t limit0_hi = 0x2A;
    const uint8_t limit0_lo = 0x2B;
    const uint8_t limit1_hi = 0x2C;
    const uint8_t limit1_lo = 0x2D;
    const uint8_t limit2_hi = 0x2E;
    const uint8_t limit2_lo = 0x2F;
    const uint8_t limit3_hi = 0x30;
    const uint8_t limit3_lo = 0x31;
    const uint8_t limit4_hi = 0x32;
    const uint8_t limit4_lo = 0x33;
    const uint8_t limit5_hi = 0x34;
    const uint8_t limit5_lo = 0x35;
    const uint8_t limit6_hi = 0x36;
    const uint8_t limit6_lo = 0x37;
    const uint8_t limit7_hi = 0x38;
    const uint8_t limit7_lo = 0x39;
} // end namespace reg


namespace config {
    const uint8_t start = 1 << 0;
    const uint8_t int_enable = 1 << 1;
    /* bit 2 reserved */
    const uint8_t int_clear = 1 << 3;
    /* bits 4 thru 6 reserved */
    const uint8_t restore_defaults = 1 << 7;
} // end namespace config


namespace conversion_rate {
    const uint8_t continious = 1 << 0;
} // end namespace conversion_rate


namespace chan_disable {
    const uint8_t ch0_disable = 1 << 0;
    const uint8_t ch1_disable = 1 << 1;
    const uint8_t ch2_disable = 1 << 2;
    const uint8_t ch3_disable = 1 << 3;
    const uint8_t ch4_disable = 1 << 4;
    const uint8_t ch5_disable = 1 << 5;
    const uint8_t ch6_disable = 1 << 6;
    const uint8_t ch7_disable = 1 << 7;
} // end namespace chan_disable

namespace oneshot {
    const uint8_t enable = 1 << 0;
} // end namespace oneshot
```

```
namespace deep_shutdown {
    const uint8_t enable = 1 << 0;
} // end namespace deep_shutdown


namespace adv_config {
    const uint8_t external_vref = 1 << 0;
    const uint8_t mode_bit0     = 1 << 1;
    const uint8_t mode_bit1     = 1 << 2;
    const uint8_t mode1 = adv_config::mode_bit0;
} // end namespace adv_config


namespace busy {
    const uint8_t busy      = 1 << 0;
    const uint8_t not_ready = 1 << 1;
} // end namespace busy
```

3.2.4 Library Makefile

```
CXX := g++
CFLAGS := -I.
LFLAGS :=

.PHONY: all
all: lib

lib: adc128.o i2cdevice.o

%.o: %.cpp
    $(CXX) $(CFLAGS) -c $^ -o $@ $(LFLAGS)

.PHONY: clean
clean:
    rm *.o
```

3.2.5 thermistor_temperature.cpp

```
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <wiringPi.h>
#include "adc128.h"

double calculate_temperature(uint16_t adc_reading);

int main(int argc, char *argv[]) {
    usleep(100000); // wait more than 33ms on poweron
    ADC128 adc(0x1D);

    while(adc.is_busy())
        usleep(100000);

    adc.begin();
```

```
    while (true) {
        for (uint8_t chan = 0; chan < 8; chan++) {
            uint16_t val = adc.analogRead(chan);
            printf("%6f ", calculate_temperature(val) - 273.15);
        }
        printf("\n");
        delay(500);
    }
}

double calculate_temperature(uint16_t adc_reading) {
    const double t25 = 298.15; // kelvin
    const double b25 = 3435;   // units kelvin but just so the units all match
    const double r25 = 10e3;   // nominal thermistor resistance
    const double r1  = 18e3;   // resistor divider resistance

    // fraction of VCC
    double v_frac = ((double) adc_reading) / ((double) 0xFFF);
    // thermistor resistance
    double rth = r1 * v_frac/(1-v_frac);
    // temperature from emperical formula
    double tmp = 1/t25 + log(rth/r25)/b25;
    double temperature = 1/tmp;
    return temperature;
}
```

3.3 Arduino Heater Driver Firmware

This is the firmware that actually runs on the heater driver (simulation electronics) system.

```
// PCA9865 - a library for the PCA9865 PWM expander and LED driver chip
// Arduino heater driver firmware for team OTheRS

/*
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software,
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#include "pca9865.h"
```

```
const uint8_t OE   = A3; // output enable, active low
const uint8_t ADDR = 0x5D;
const uint8_t CHAN = 11;

PCA9865 pca(ADDR);

void setup() {
    Serial.begin(9600);
    pinMode(OE, OUTPUT);
    digitalWrite(OE, LOW);
    delay(500);

    pca.begin();
    while (!Serial);
}

void loop() {
    /* All action handled by interrupt */
}

void serialEvent() {
    uint8_t chan  = (uint8_t) Serial.parseInt();
    uint8_t value = (uint8_t) Serial.parseInt();

    pca.analogWrite(chan, value);

    while (Serial.available()) // a new line can get interpreted as a zero
        Serial.read();         // so we need to read in the rest of the buffer
                               // to prevent another call to serialEvent
}
```

3.4 PCA9865PW Driver Library

The PCA9865 is a PWM driver IC that modulates all the MOSFETs on the heater drivers. This code is an important part of the simulation electronics system.

3.4.1 pca9865.h

```
// PCA9865 - a library for the PCA9865 PWM expander and LED driver chip
// Header file

/*
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
```

```
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#pragma once

#include <stdint.h>

// Arduno Uno
#ifdef ARDUINO
#include <Arduino.h>
#include <Wire.h>
#endif

// Raspberry Pi
#ifdef __linux__
#include <i2cdevice.h>
#include <memory> // std::unique_ptr
#endif


class PCA9865 {
    public:
        PCA9865(uint8_t address);

        void begin();
        void writeConfig();
        void analogWrite(uint8_t channel, uint8_t val);

        // NOTE: default is to call syncConfig() right after changing
        // any parameter, but this can be deferred for the sake of efficiency
        // in case many things are changed at once
        void enableRestart( bool immediate = true);
        void disableRestart(bool immediate = true);
        // we want the external clock disabled
        void enableExtclk( bool immediate = true);
        void disableExtclk(bool immediate = true);
        // auto increment is a convience feature
        void enableAi( bool immediate = true);
        void disableAi(bool immediate = true);
        // enter sleep mode
        void enableSleep( bool immediate = true);
        void disableSleep(bool immediate = true);
        // whether to respond to sub addresses
        void enableSub1( bool immediate = true);
        void disableSub1(bool immediate = true);
        void enableSub2( bool immediate = true);
        void disableSub2(bool immediate = true);
        void enableSub3( bool immediate = true);
        void disableSub3(bool immediate = true);
        // whether to respond to all call address
```

```
        void enableAllCall( bool immediate = true);
        void disableAllCall(bool immediate = true);

        // Invert the PWM signals (NOTE 0% becomes FULL-BLAST with this)
        void enableInvert( bool immediate = true);
        void disableInvert(bool immediate = true);

        // Looks like totem pole driving is needed, but
        // figuring out exactly what the correct configuration is
        // is still TODO
        void enableTotemPole( bool immediate = true);
        void disableTotemPole(bool immediate = true);

        // check if each of the config bits is set
        // NOTE: default behavior is to fetch the actual
        // register off the chip each time. This does
        // present the risk of a bad I2C packet overwriting
        // the variable with garbage data.
        // If that happens we can just request the data again.
        bool isRestart(  bool fetch = true);
        bool isExtclk(   bool fetch = true);
        bool isAi(       bool fetch = true);
        bool isSleep(    bool fetch = true);
        bool isSub1(     bool fetch = true);
        bool isSub2(     bool fetch = true);
        bool isSub3(     bool fetch = true);
        bool isAllCall(  bool fetch = true);
        bool isInvert(   bool fetch = true);
        bool isTotemPole(bool fetch = true);

    private:
        void i2c_bus_init();
        void writeRegister(uint8_t reg, uint8_t data);
        uint8_t readRegister(uint8_t reg);

        uint8_t addr;      // I2C address
        uint8_t reg_mode1 = 0b00000000; // state of the MODE1 register
        uint8_t reg_mode2 = 0b00001011; // state of the MODE2 register

// Raspberry Pi
#ifdef __linux__
        std::unique_ptr<I2CDevice> i2c; // I2C backend
#endif
};
```

3.4.2 pca9865.cpp

```
// PCA9865 - a library for the PCA9865 PWM expander and LED driver chip
// Main source file

/*
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
```

```
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#include <pca9865.h>
#include <pca9865_constants.h>

namespace {
    // print a binary number as fixed width
    void sprint_bin(uint8_t num, char *dest) {
        for (uint8_t i = 0; i < 8; i++) { // each bit
            if ( ((num >> i) & 1) == 1 ) // bit is a 1
                dest[7-i] = '1';
            else
                dest[7-i] = '0';
        }
    }

}


PCA9865::PCA9865(uint8_t address) {
    addr = address;
}

// Initialize config registers
void PCA9865::begin() {
    i2c_bus_init();

    reg_mode1 = readRegister(reg_addr::MODE1);
    reg_mode2 = readRegister(reg_addr::MODE2);

    // disable basically everything
    disableRestart(false);
    disableExtclk(false);
    disableSleep(false);
    disableAi(false);
    disableSub1(false);
    disableSub2(false);
    disableSub3(false);
    disableAllCall(false);

    disableInvert(false);
    enableTotemPole(false);

    // reg_mode2 = 0b00001100;
    writeConfig();
```

```
#ifdef ARDUINO
    // zero out all the channels on start
    // this behavior works on arduino because begin() will be called once during setup()
    // but on Linux it doesn't make sense because I'm compiling this down to command line
    // utilities that I call repeatedly
    for (uint8_t chan = 0; chan < 16; chan++)
        analogWrite(chan, 0);
#endif // ARDUINO
}

// write configuration to the actual chip
inline void PCA9865::writeConfig() {
    writeRegister(reg_addr::MODE1, reg_mode1);
    writeRegister(reg_addr::MODE2, reg_mode2);
}

void PCA9865::enableRestart(bool immediate) {
    reg_mode1 |= mode1::RESTART;
    if (immediate)
        writeConfig();
}

void PCA9865::disableRestart(bool immediate) {
    reg_mode1 &= ~mode1::RESTART;
    if (immediate)
        writeConfig();
}

void PCA9865::enableExtclk(bool immediate) {
    reg_mode1 |= mode1::EXTCLK;
    if (immediate)
        writeConfig();
}

void PCA9865::disableExtclk(bool immediate) {
    reg_mode1 &= ~mode1::EXTCLK;
    if (immediate)
        writeConfig();
}

void PCA9865::enableAi(bool immediate) {
    reg_mode1 |= mode1::AI;
    if (immediate)
        writeConfig();
}

void PCA9865::disableAi(bool immediate) {
    reg_mode1 &= ~mode1::AI;
    if (immediate)
        writeConfig();
}

void PCA9865::enableSleep(bool immediate) {
    reg_mode1 |= mode1::SLEEP;
    if (immediate)
        writeConfig();
```

```
}

void PCA9865::disableSleep(bool immediate) {
    reg_mode1 &= ~mode1::SLEEP;
    if (immediate)
        writeConfig();
}

void PCA9865::enableSub1(bool immediate) {
    reg_mode1 |= mode1::SUB1;
    if (immediate)
        writeConfig();
}

void PCA9865::disableSub1(bool immediate) {
    reg_mode1 &= ~mode1::SUB1;
    if (immediate)
        writeConfig();
}

void PCA9865::enableSub2(bool immediate) {
    reg_mode1 |= mode1::SUB2;
    if (immediate)
        writeConfig();
}

void PCA9865::disableSub2(bool immediate) {
    reg_mode1 &= ~mode1::SUB2;
    if (immediate)
        writeConfig();
}

void PCA9865::enableSub3(bool immediate) {
    reg_mode1 |= mode1::SUB3;
    if (immediate)
        writeConfig();
}

void PCA9865::disableSub3(bool immediate) {
    reg_mode1 &= ~mode1::SUB3;
    if (immediate)
        writeConfig();
}

void PCA9865::enableAllCall(bool immediate) {
    reg_mode1 |= mode1::ALLCALL;
    if (immediate)
        writeConfig();
}

void PCA9865::disableAllCall(bool immediate) {
    reg_mode1 &= ~mode1::ALLCALL;
    if (immediate)
        writeConfig();
}

void PCA9865::enableInvert(bool immediate) {
```

```
    reg_mode2 |= mode2::INVRT;
    if (immediate)
        writeConfig();
}

void PCA9865::disableInvert(bool immediate) {
    reg_mode2 &= ~mode2::INVRT;
    if (immediate)
        writeConfig();
}

void PCA9865::enableTotemPole(bool immediate) {
    reg_mode2 |= mode2::OUTDRV;
    if (immediate)
        writeConfig();
}

void PCA9865::disableTotemPole(bool immediate) {
    reg_mode2 &= ~mode2::OUTDRV;
    if (immediate)
        writeConfig();
}

bool PCA9865::isRestart(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::RESTART) > 0;
}

bool PCA9865::isExtclk(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::EXTCLK) > 0;
}

bool PCA9865::isAi(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::AI) > 0;
}

bool PCA9865::isSleep(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::SLEEP) > 0;
}

bool PCA9865::isSub1(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::SUB1) > 0;
}

bool PCA9865::isSub2(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::SUB2) > 0;
```

```
}

bool PCA9865::isSub3(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::SUB3) > 0;
}

bool PCA9865::isAllCall(bool fetch) {
    if (fetch)
        reg_mode1 = readRegister(reg_addr::MODE1);
    return (reg_mode1 & mode1::ALLCALL) > 0;
}

bool PCA9865::isInvert(bool fetch) {
    if (fetch)
        reg_mode2 = readRegister(reg_addr::MODE2);
    return (reg_mode1 & mode2::INVRT) > 0;
}

bool PCA9865::isTotemPole(bool fetch) {
    if (fetch)
        reg_mode2 = readRegister(reg_addr::MODE2);
    return (reg_mode1 & mode2::OUTDRV) > 0;
}

void PCA9865::analogWrite(uint8_t chan, uint8_t percent) {
    uint16_t on_time  = 1; // counter time before input is turned on
    uint16_t off_time = (((float) percent*4094)/100.0) + 1;

    uint8_t channel_on_l;
    uint8_t channel_on_h;
    uint8_t channel_off_l;
    uint8_t channel_off_h;

    switch (chan) { // match channel to apropriate registers
        case 0:
            channel_on_l  = reg_addr::LED0_ON_L;
            channel_on_h  = reg_addr::LED0_ON_H;
            channel_off_l = reg_addr::LED0_OFF_L;
            channel_off_h = reg_addr::LED0_OFF_H;
            break;
        case 1:
            channel_on_l  = reg_addr::LED1_ON_L;
            channel_on_h  = reg_addr::LED1_ON_H;
            channel_off_l = reg_addr::LED1_OFF_L;
            channel_off_h = reg_addr::LED1_OFF_H;
            break;
        case 2:
            channel_on_l  = reg_addr::LED2_ON_L;
            channel_on_h  = reg_addr::LED2_ON_H;
            channel_off_l = reg_addr::LED2_OFF_L;
            channel_off_h = reg_addr::LED2_OFF_H;
            break;
        case 3:
            channel_on_l  = reg_addr::LED3_ON_L;
            channel_on_h  = reg_addr::LED3_ON_H;
```

```
        channel_off_l = reg_addr::LED3_OFF_L;
        channel_off_h = reg_addr::LED3_OFF_H;
        break;
    case 4:
        channel_on_l  = reg_addr::LED4_ON_L;
        channel_on_h  = reg_addr::LED4_ON_H;
        channel_off_l = reg_addr::LED4_OFF_L;
        channel_off_h = reg_addr::LED4_OFF_H;
        break;
    case 5:
        channel_on_l  = reg_addr::LED5_ON_L;
        channel_on_h  = reg_addr::LED5_ON_H;
        channel_off_l = reg_addr::LED5_OFF_L;
        channel_off_h = reg_addr::LED5_OFF_H;
        break;
    case 6:
        channel_on_l  = reg_addr::LED6_ON_L;
        channel_on_h  = reg_addr::LED6_ON_H;
        channel_off_l = reg_addr::LED6_OFF_L;
        channel_off_h = reg_addr::LED6_OFF_H;
        break;
    case 7:
        channel_on_l  = reg_addr::LED7_ON_L;
        channel_on_h  = reg_addr::LED7_ON_H;
        channel_off_l = reg_addr::LED7_OFF_L;
        channel_off_h = reg_addr::LED7_OFF_H;
        break;
    case 8:
        channel_on_l  = reg_addr::LED8_ON_L;
        channel_on_h  = reg_addr::LED8_ON_H;
        channel_off_l = reg_addr::LED8_OFF_L;
        channel_off_h = reg_addr::LED8_OFF_H;
        break;
    case 9:
        channel_on_l  = reg_addr::LED9_ON_L;
        channel_on_h  = reg_addr::LED9_ON_H;
        channel_off_l = reg_addr::LED9_OFF_L;
        channel_off_h = reg_addr::LED9_OFF_H;
        break;
    case 10:
        channel_on_l  = reg_addr::LED10_ON_L;
        channel_on_h  = reg_addr::LED10_ON_H;
        channel_off_l = reg_addr::LED10_OFF_L;
        channel_off_h = reg_addr::LED10_OFF_H;
        break;
    case 11:
        channel_on_l  = reg_addr::LED11_ON_L;
        channel_on_h  = reg_addr::LED11_ON_H;
        channel_off_l = reg_addr::LED11_OFF_L;
        channel_off_h = reg_addr::LED11_OFF_H;
        break;
    case 12:
        channel_on_l  = reg_addr::LED12_ON_L;
        channel_on_h  = reg_addr::LED12_ON_H;
        channel_off_l = reg_addr::LED12_OFF_L;
        channel_off_h = reg_addr::LED12_OFF_H;
        break;
```

```
        case 13:
            channel_on_l  = reg_addr::LED13_ON_L;
            channel_on_h  = reg_addr::LED13_ON_H;
            channel_off_l = reg_addr::LED13_OFF_L;
            channel_off_h = reg_addr::LED13_OFF_H;
            break;
        case 14:
            channel_on_l  = reg_addr::LED14_ON_L;
            channel_on_h  = reg_addr::LED14_ON_H;
            channel_off_l = reg_addr::LED14_OFF_L;
            channel_off_h = reg_addr::LED14_OFF_H;
            break;
        case 15:
            channel_on_l  = reg_addr::LED15_ON_L;
            channel_on_h  = reg_addr::LED15_ON_H;
            channel_off_l = reg_addr::LED15_OFF_L;
            channel_off_h = reg_addr::LED15_OFF_H;
            break;
        default:
            channel_on_l  = reg_addr::LED0_ON_L;
            channel_on_h  = reg_addr::LED0_ON_H;
            channel_off_l = reg_addr::LED0_OFF_L;
            channel_off_h = reg_addr::LED0_OFF_H;
            break;
    }

    // TODO: this works but might make more sense to have auto increment
    // TODO: only bottom 4 bits of the high byte register control duty cylce
    //       so ANDing with 0x0F instead of 0xFF, but need to verify this is
    //       correct. The top 4 bits are apparently some kind of control

    // writeRegister(channel_on_h,  (on_time >> 8)  & 0xFF);
    writeRegister(channel_on_h,  (on_time >> 8)  & 0x0F);
    writeRegister(channel_on_l,   on_time        & 0xFF);

    // writeRegister(channel_off_h, (off_time >> 8) & 0xFF);
    writeRegister(channel_off_h, (off_time >> 8) & 0x0F);
    writeRegister(channel_off_l,  off_time       & 0xFF);
}

void PCA9865::i2c_bus_init() {
#ifdef ARDUINO
    Wire.begin();
#else // Raspberry Pi
    i2c = std::unique_ptr<I2CDevice>(new I2CDevice(addr, "/dev/i2c-1"));
#endif
    }
// write a byte to a register
void PCA9865::writeRegister(uint8_t reg, uint8_t data) {
#ifdef ARDUINO
    Wire.beginTransmission(addr);
    Wire.write(reg);
    Wire.write(data);
    Wire.endTransmission();
#else // Raspberry Pi
    uint8_t buf[] = {reg, data};
    i2c->write(buf, 2);
```

```
#endif
}

uint8_t PCA9865::readRegister(uint8_t reg) {
#ifdef ARDUINO
    Wire.requestFrom(addr, (uint8_t) 1); // ISO C++ warns if an integer literal
    while (!Wire.available());           // is converted implicitly to a uint8_t
    uint8_t result = Wire.read();
#else // Raspberry Pi
    uint8_t result = reg; // TODO: check this
    i2c->write(&result, 1);
    i2c->read(&result, 1);
#endif

    return result;
}
```

3.4.3 pca9865_constants.h - Register addresses copied from datasheet - NXP PCA9865PW datasheet rev 4

```
// PCA9865 - a library for the PCA9865 PWM expander and LED driver chip.
// Constants file

/*
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#pragma once

namespace mode1 {
    const uint8_t RESTART = 1 << 7; // Restart the chip
    const uint8_t EXTCLK  = 1 << 6; // use an external clock. NOTE: should always be zero for
    const uint8_t AI      = 1 << 5; // Auto increment reg addr after writing the last register
    const uint8_t SLEEP   = 1 << 4; // Low power mode
    const uint8_t SUB1    = 1 << 3; // respond to sub addr 1
    const uint8_t SUB2    = 1 << 2; // respond to sub addr 2
    const uint8_t SUB3    = 1 << 1; // respond to sub addr 3
    const uint8_t ALLCALL = 1 << 0; // respond to all call addr
} // end namespace mode1

namespace mode2 {
    const uint8_t INVRT  = 1 << 4; // invert output logic
```

```
    const uint8_t OCH    = 1 << 3; // output change on I2C STOP vs slave ACK
    const uint8_t OUTDRV = 1 << 2; // totem pole vs open drain

    /* related to the ~OE pin, but exact functionality is unclear */
    const uint8_t OUTNEh = 1 << 1;
    const uint8_t OUTNEl = 1 << 0;
} // end namespace mode2

// Register addresses
namespace reg_addr {
    const uint8_t MODE1         = 0x00; // r/w Mode register 1
    const uint8_t MODE2         = 0x01; // r/w Mode register 2
    const uint8_t SUBADR1       = 0x02; // r/w I2C bus subaddress 1
    const uint8_t SUBADR2       = 0x03; // r/w I2C bus subaddress 2
    const uint8_t SUBADR3       = 0x04; // r/w I2C bus subaddress 3
    const uint8_t ALLCALLADR    = 0x05; // r/w LED All Call I2C bus address
    const uint8_t LED0_ON_L     = 0x06; // r/w LED0 output and brightness control byte 0
    const uint8_t LED0_ON_H     = 0x07; // r/w LED0 output and brightness control byte 1
    const uint8_t LED0_OFF_L    = 0x08; // r/w LED0 output and brightness control byte 2
    const uint8_t LED0_OFF_H    = 0x09; // r/w LED0 output and brightness control byte 3
    const uint8_t LED1_ON_L     = 0x0A; // r/w LED1 output and brightness control byte 0
    const uint8_t LED1_ON_H     = 0x0B; // r/w LED1 output and brightness control byte 1
    const uint8_t LED1_OFF_L    = 0x0C; // r/w LED1 output and brightness control byte 2
    const uint8_t LED1_OFF_H    = 0x0D; // r/w LED1 output and brightness control byte 3
    const uint8_t LED2_ON_L     = 0x0E; // r/w LED2 output and brightness control byte 0
    const uint8_t LED2_ON_H     = 0x0F; // r/w LED2 output and brightness control byte 1
    const uint8_t LED2_OFF_L    = 0x10; // r/w LED2 output and brightness control byte 2
    const uint8_t LED2_OFF_H    = 0x11; // r/w LED2 output and brightness control byte 3
    const uint8_t LED3_ON_L     = 0x12; // r/w LED3 output and brightness control byte 0
    const uint8_t LED3_ON_H     = 0x13; // r/w LED3 output and brightness control byte 1
    const uint8_t LED3_OFF_L    = 0x14; // r/w LED3 output and brightness control byte 2
    const uint8_t LED3_OFF_H    = 0x15; // r/w LED3 output and brightness control byte 3
    const uint8_t LED4_ON_L     = 0x16; // r/w LED4 output and brightness control byte 0
    const uint8_t LED4_ON_H     = 0x17; // r/w LED4 output and brightness control byte 1
    const uint8_t LED4_OFF_L    = 0x18; // r/w LED4 output and brightness control byte 2
    const uint8_t LED4_OFF_H    = 0x19; // r/w LED4 output and brightness control byte 3
    const uint8_t LED5_ON_L     = 0x1A; // r/w LED5 output and brightness control byte 0
    const uint8_t LED5_ON_H     = 0x1B; // r/w LED5 output and brightness control byte 1
    const uint8_t LED5_OFF_L    = 0x1C; // r/w LED5 output and brightness control byte 2
    const uint8_t LED5_OFF_H    = 0x1D; // r/w LED5 output and brightness control byte 3
    const uint8_t LED6_ON_L     = 0x1E; // r/w LED6 output and brightness control byte 0
    const uint8_t LED6_ON_H     = 0x1F; // r/w LED6 output and brightness control byte 1
    const uint8_t LED6_OFF_L    = 0x20; // r/w LED6 output and brightness control byte 2
    const uint8_t LED6_OFF_H    = 0x21; // r/w LED6 output and brightness control byte 3
    const uint8_t LED7_ON_L     = 0x22; // r/w LED7 output and brightness control byte 0
    const uint8_t LED7_ON_H     = 0x23; // r/w LED7 output and brightness control byte 1
    const uint8_t LED7_OFF_L    = 0x24; // r/w LED7 output and brightness control byte 2
    const uint8_t LED7_OFF_H    = 0x25; // r/w LED7 output and brightness control byte 3
    const uint8_t LED8_ON_L     = 0x26; // r/w LED8 output and brightness control byte 0
    const uint8_t LED8_ON_H     = 0x27; // r/w LED8 output and brightness control byte 1
    const uint8_t LED8_OFF_L    = 0x28; // r/w LED8 output and brightness control byte 2
    const uint8_t LED8_OFF_H    = 0x29; // r/w LED8 output and brightness control byte 3
    const uint8_t LED9_ON_L     = 0x2A; // r/w LED9 output and brightness control byte 0
    const uint8_t LED9_ON_H     = 0x2B; // r/w LED9 output and brightness control byte 1
    const uint8_t LED9_OFF_L    = 0x2C; // r/w LED9 output and brightness control byte 2
    const uint8_t LED9_OFF_H    = 0x2D; // r/w LED9 output and brightness control byte 3
    const uint8_t LED10_ON_L    = 0x2E; // r/w LED10 output and brightness control byte 0
```

```
    const uint8_t LED10_ON_H    = 0x2F; // r/w LED10 output and brightness control byte 1
    const uint8_t LED10_OFF_L   = 0x30; // r/w LED10 output and brightness control byte 2
    const uint8_t LED10_OFF_H   = 0x31; // r/w LED10 output and brightness control byte 3
    const uint8_t LED11_ON_L    = 0x32; // r/w LED11 output and brightness control byte 0
    const uint8_t LED11_ON_H    = 0x33; // r/w LED11 output and brightness control byte 1
    const uint8_t LED11_OFF_L   = 0x34; // r/w LED11 output and brightness control byte 2
    const uint8_t LED11_OFF_H   = 0x35; // r/w LED11 output and brightness control byte 3
    const uint8_t LED12_ON_L    = 0x36; // r/w LED12 output and brightness control byte 0
    const uint8_t LED12_ON_H    = 0x37; // r/w LED12 output and brightness control byte 1
    const uint8_t LED12_OFF_L   = 0x38; // r/w LED12 output and brightness control byte 2
    const uint8_t LED12_OFF_H   = 0x39; // r/w LED12 output and brightness control byte 3
    const uint8_t LED13_ON_L    = 0x3A; // r/w LED13 output and brightness control byte 0
    const uint8_t LED13_ON_H    = 0x3B; // r/w LED13 output and brightness control byte 1
    const uint8_t LED13_OFF_L   = 0x3C; // r/w LED13 output and brightness control byte 2
    const uint8_t LED13_OFF_H   = 0x3D; // r/w LED13 output and brightness control byte 3
    const uint8_t LED14_ON_L    = 0x3E; // r/w LED14 output and brightness control byte 0
    const uint8_t LED14_ON_H    = 0x3F; // r/w LED14 output and brightness control byte 1
    const uint8_t LED14_OFF_L   = 0x40; // r/w LED14 output and brightness control byte 2
    const uint8_t LED14_OFF_H   = 0x41; // r/w LED14 output and brightness control byte 3
    const uint8_t LED15_ON_L    = 0x42; // r/w LED15 output and brightness control byte 0
    const uint8_t LED15_ON_H    = 0x43; // r/w LED15 output and brightness control byte 1
    const uint8_t LED15_OFF_L   = 0x44; // r/w LED15 output and brightness control byte 2
    const uint8_t LED15_OFF_H   = 0x45; // r/w LED15 output and brightness control byte 3
    const uint8_t ALL_LED_ON_L  = 0xFA; // w/r zero load all the LEDn_ON registers, byte 0
    const uint8_t ALL_LED_ON_H  = 0xFB; // w/r zero load all the LEDn_ON registers, byte 1
    const uint8_t ALL_LED_OFF_L = 0xFC; // w/r zero load all the LEDn_OFF registers, byte 0
    const uint8_t ALL_LED_OFF_H = 0xFD; // w/r zero load all the LEDn_OFF registers, byte 1
    const uint8_t PRE_SCALE     = 0xFE; // r/w prescaler for PWM output frequency
    const uint8_t TestMode      = 0xFF; // r/w defines the test mode to be entered
} // end namespace reg_addr
```

3.4.4 Library Makefile

```
CXX := g++
CFLAGS := -I.
LFLAGS :=

.PHONY: all
all: lib

lib: pca9865.o i2cdevice.o

%.o: %.cpp
    $(CXX) $(CFLAGS) -c $^ -o $@ $(LFLAGS)

.PHONY: clean
clean:
    rm *.o
```

3.5 I2C Device

Both the ADC128 and PCA9865 chips depend on this library when running on the Raspberry Pi.

3.5.1 i2cdevice.h

```
/*
 * i2cdevice.h - A linux I2C compatibility layer to easy I2C access
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#pragma once

#include <stdint.h>
#include <linux/i2c-dev.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>

class I2CDevice {
public:
    I2CDevice(uint8_t addr, const char *bus = "/dev/i2c-1");
    ~I2CDevice();
    ssize_t write(const uint8_t *buf, size_t count);
    ssize_t read(uint8_t *buf, size_t count);

private:
    int _fd;
};
```

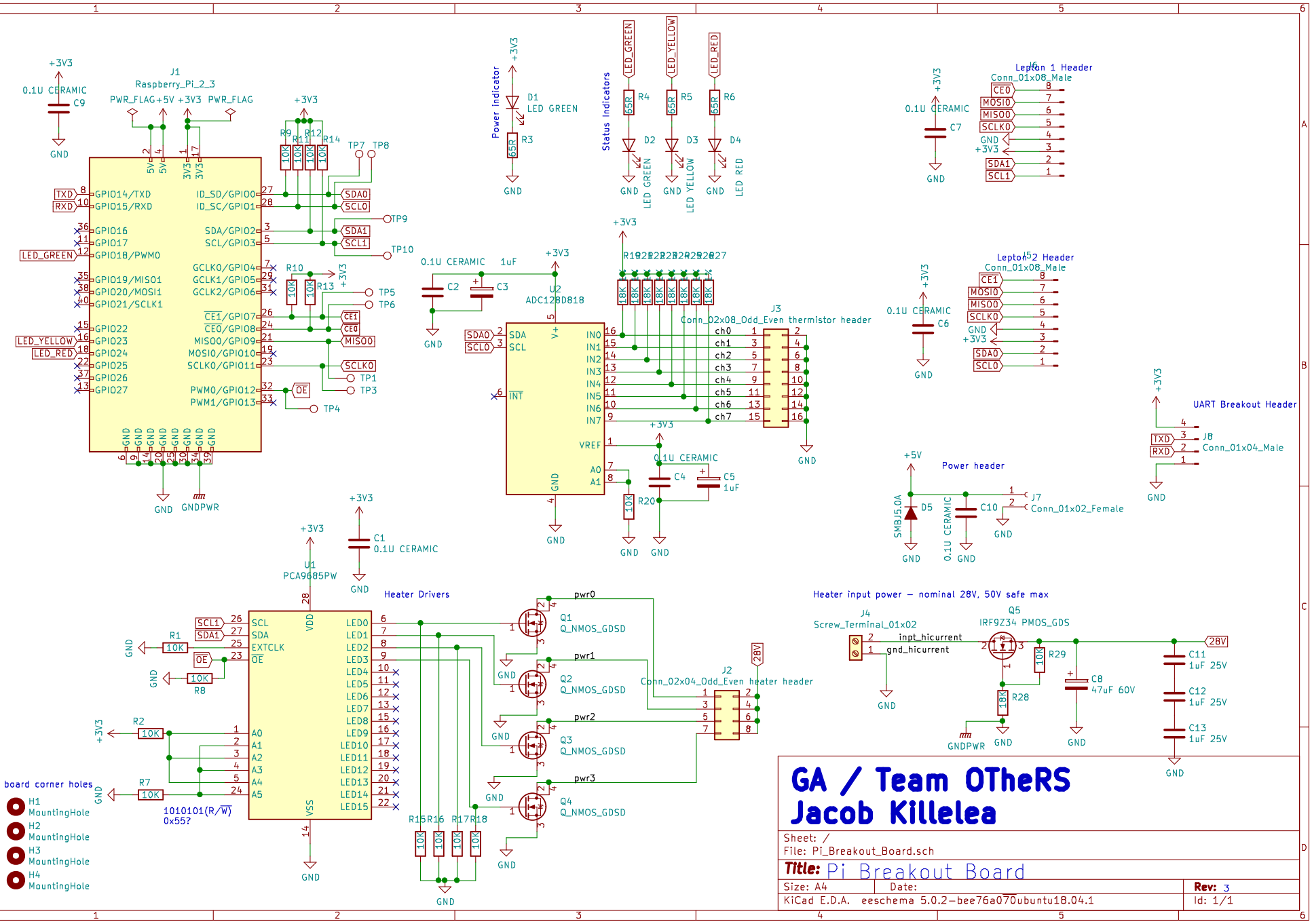
3.5.2 i2cdevice.cpp

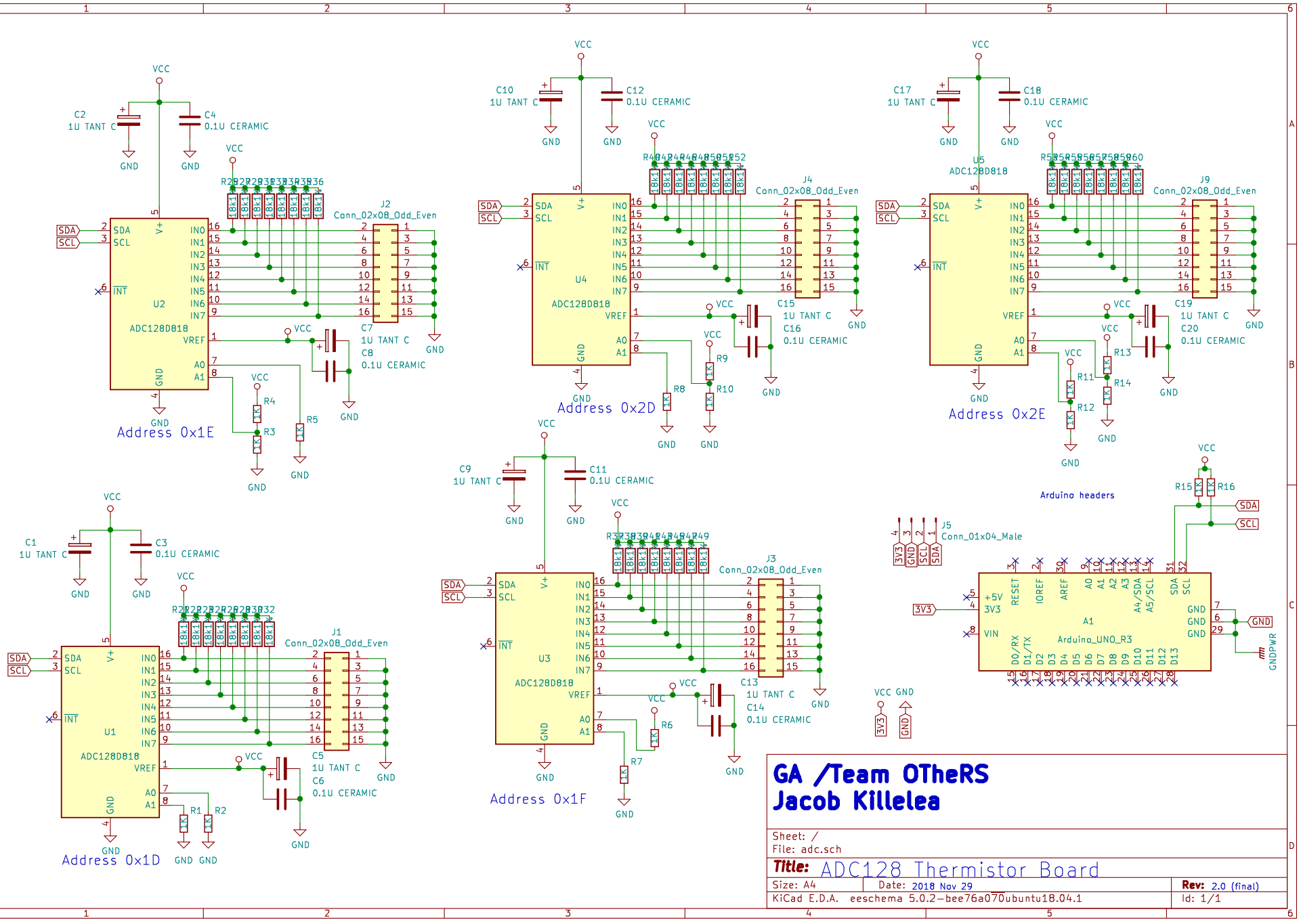
```
/*
 * i2cdevice.cpp - A linux I2C compatibility layer to easy I2C access
 * Copyright 2019 Jacob Killelea <jkillelea@protonmail.ch>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy of
 * this software and associated documentation files (the "Software"), to deal in the
 * Software without restriction, including without limitation the rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies of the Software,
 * and to permit persons to whom the Software is furnished to do so, subject to the
 * following conditions:
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
```

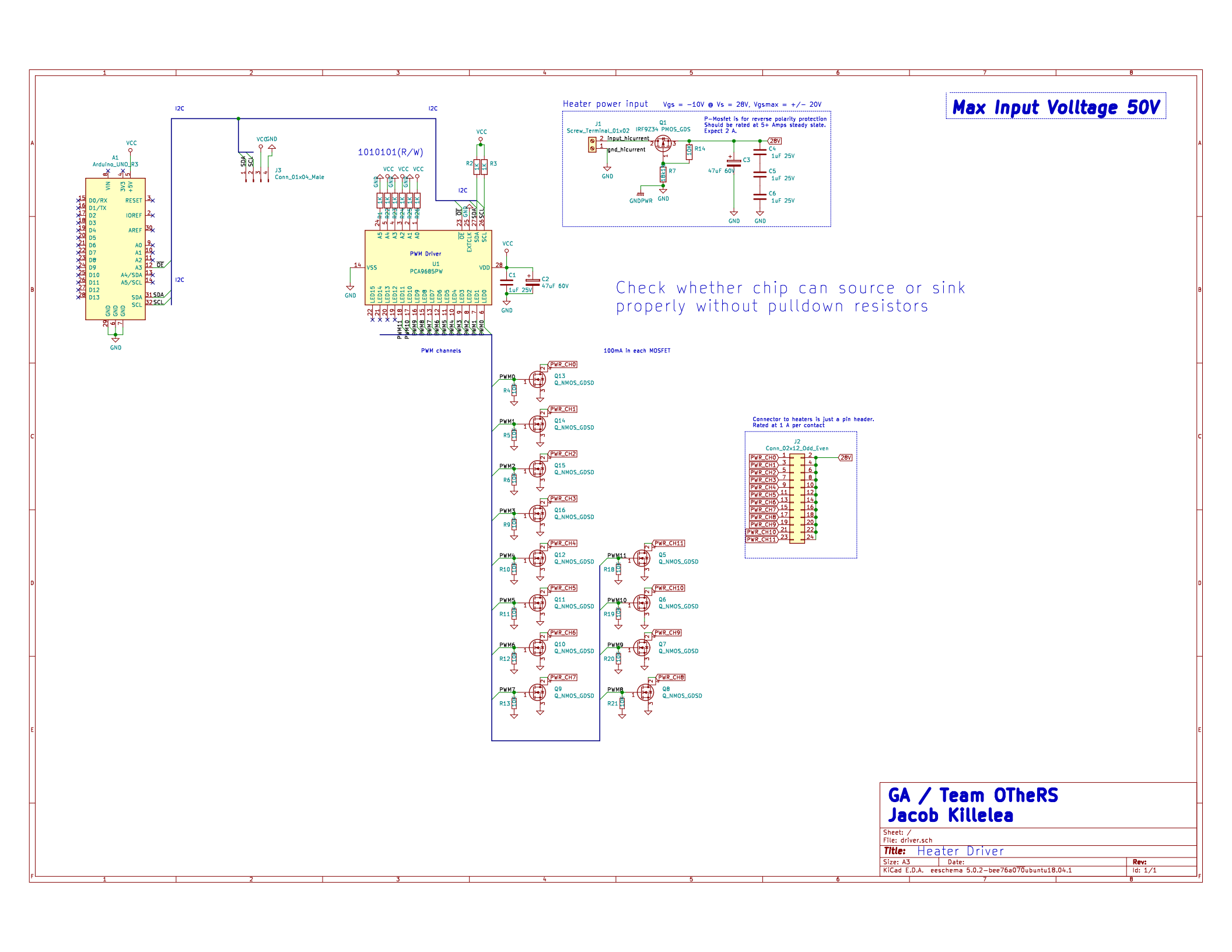
```
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
 * INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
 * HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE
 * OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 */

#include <i2cdevice.h>

#include <stdio.h>
#include <stdlib.h>

I2CDevice::I2CDevice(uint8_t addr, const char *bus) {
    _fd = open(bus, O_RDWR);
    if (_fd < 0) {
        perror("open");
        abort();
    }
    if (ioctl(_fd, I2C_SLAVE, addr)) {
        perror("ioctl");
        abort();
    }
}

I2CDevice::~I2CDevice() {
    close(_fd);
}

ssize_t I2CDevice::write(const uint8_t *buf, size_t count) {
    return ::write(_fd, buf, count);
}

ssize_t I2CDevice::read(uint8_t *buf, size_t count) {
    return ::read(_fd, &buf, count);
}
```

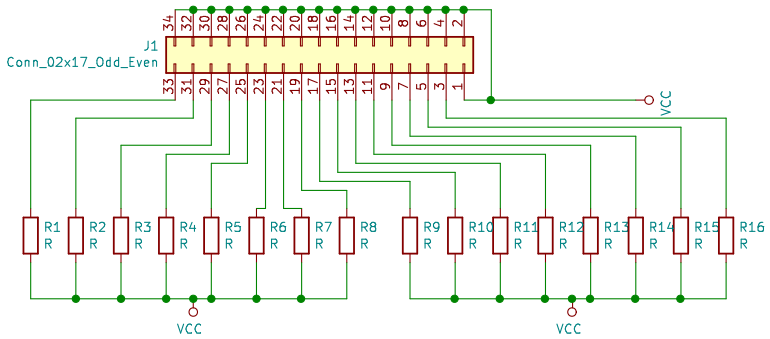
4 Electrical Design Schematics

These are the schematics for all of the custom electronic systems in the project. Nobody else in the project had the experience to do this, and while I explained these elements to people, none of them had the time to learn the development software and so I did every PCB in the project. They enabled key elements of the project - in particular, we tried several times to run a test with both Lepton cameras and the electronics stack, but until the PCB for the Raspberry Pi simplified wiring, it was a huge mess on the breadboard and just took too long to set everything up.


J1
Raspberry_Pi_2_3
PWR_FLAG +5V +3V3 PWR_FLAG
GPIO14/TXD
GPIO15/RXD
GPIO16
GPIO17
GPIO18/PWM0
GPIO19/MISO1
GPIO20/MOSI1
GPIO21/SCLK1
GPIO22
GPIO23
GPIO24
GPIO25
GPIO26
GPIO27
ID_SD/GPIO0
ID_SC/GPIO1
SDA/GPIO2
SCL/GPIO3
GCLK0/GPIO4
GCLK1/GPIO5
GCLK2/GPIO6
CE1/GPIO7
CE0/GPIO8
MISO0/GPIO9
MOSI0/GPIO10
SCLK0/GPIO11
PWM0/GPIO12
PWM1/GPIO13
TXD
RXD
LED_GREEN
LED_YELLOW
LED_RED
SDA0
SCL0
SDA1
SCL1
CE1
CE0
MISO0
SCLK0
OE
GNDPWR
+3V3
0.1U CERAMIC
C9
R9 R11 R12 R14
10K
TP7 TP8 TP9 TP10 TP5 TP6 TP1 TP3 TP4
R10 R13
Power indicator
D1
LED GREEN
R3
65R
Status Indicators
R4 R5 R6
D2 D3 D4
LED GREEN
LED YELLOW
LED RED
J6
Lepton 1 Header
Conn_01x08_Male
C7
J5
Lepton 2 Header
C6
U2
ADC128D818
SDA
SCL
INT
V+
VREF
A0
A1
IN0 IN1 IN2 IN3 IN4 IN5 IN6 IN7
C2 C3 1uF
C4 C5 1uF
R20
J3
Conn_02x08_Odd_Even thermistor header
ch0 ch1 ch2 ch3 ch4 ch5 ch6 ch7
18K
J8
UART Breakout Header
Conn_01x04_Male
J7
Power header
Conn_01x02_Female
D5
SMBJ5.0A
C10
U1
PCA9685PW
C1
Heater Drivers
SCL
SDA
EXTCLK
OE
VDD
VSS
LED0 LED1 LED2 LED3 LED4 LED5 LED6 LED7 LED8 LED9 LED10 LED11 LED12 LED13 LED14 LED15
A0 A1 A2 A3 A4 A5
R1 R2 R7 R8
1010101(R/W)
0x55?
J2
Conn_02x04_Odd_Even heater header
pwr0 pwr1 pwr2 pwr3
Q1 Q2 Q3 Q4
Q_NMOS_GDSD
R15 R16 R17 R18
Heater input power – nominal 28V, 50V safe max
J4
Screw_Terminal_01x02
inpt_hicurrent
gnd_hicurrent
Q5
IRF9Z34 PMOS_GDS
R29
R28
C8
47uF 60V
C11 C12 C13
1uF 25V
28V
board corner holes
H1 MountingHole
H2 MountingHole
H3 MountingHole
H4 MountingHole
GA / Team OTheRS
Jacob Killelea
Sheet: /
File: Pi_Breakout_Board.sch
Title: Pi Breakout Board
Size: A4
Date:
Rev: 3
KiCad E.D.A. eeschema 5.0.2-bee76a070ubuntu18.04.1
Id: 1/1


GA /Team OTheRS
Jacob Killelea
Sheet: /
File: adc.sch
Title: ADC128 Thermistor Board
Size: A4
Date: 2018 Nov 29
Rev: 2.0 (final)
KiCad E.D.A. eeschema 5.0.2-bee76a070ubuntu18.04.1
Id: 1/1
Address 0x1E
Address 0x1D
Address 0x2D
Address 0x1F
Address 0x2E
Arduino headers
ADC128D818
Arduino_UNO_R3
Conn_02x08_Odd_Even
Conn_01x04_Male
1U TANT C
0.1U CERAMIC
VCC
GND


Max Input Volltage 50V
Heater power input
Vgs = −10V @ Vs = 28V, Vgsmax = +/− 20V
P−Mosfet is for reverse polarity protection
Should be rated at 5+ Amps steady state.
Expect 2 A.
J1
Screw_Terminal_01x02
input_hicurrent
gnd_hicurrent
Q1
IRF9Z34 PMOS_GDS
R14
10Ω
R7
18kΩ
C3
47uF 60V
C4
1uF 25V
C5
1uF 25V
C6
1uF 25V
28V
GND
GNDPWR
A1
Arduino_UNO_R3
J3
Conn_01x04_Male
SDA
SCL
I2C
1010101(R/W)
U1
PCA9685PW
PWM Driver
VSS
VDD
OE
EXTCLK
C1
1uF 25V
C2
47uF 60V
PWM channels
100mA in each MOSFET
Check whether chip can source or sink
properly without pulldown resistors
Connector to heaters is just a pin header.
Rated at 1 A per contact
J2
Conn_02x12_Odd_Even
PWR_CH0
PWR_CH1
PWR_CH2
PWR_CH3
PWR_CH4
PWR_CH5
PWR_CH6
PWR_CH7
PWR_CH8
PWR_CH9
PWR_CH10
PWR_CH11
Q13
Q14
Q15
Q16
Q12
Q11
Q10
Q9
Q5
Q6
Q7
Q8
Q_NMOS_GDSD
GA / Team OTheRS
Jacob Killelea
Sheet: /
File: driver.sch
Title: Heater Driver
Size: A3
Date:
Rev:
KiCad E.D.A. eeschema 5.0.2−bee76a0?ubuntu18.04.1
Id: 1/1



H1 MountingHole 2mm
H2 MountingHole 2mm
H3 MountingHole 2mm
H4 MountingHole 2mm
H5 MountingHole 2mm
H6 MountingHole 2mm
H7 MountingHole 2mm
H8 MountingHole 2mm

GA / Team OTheRS
Jacob Killelea

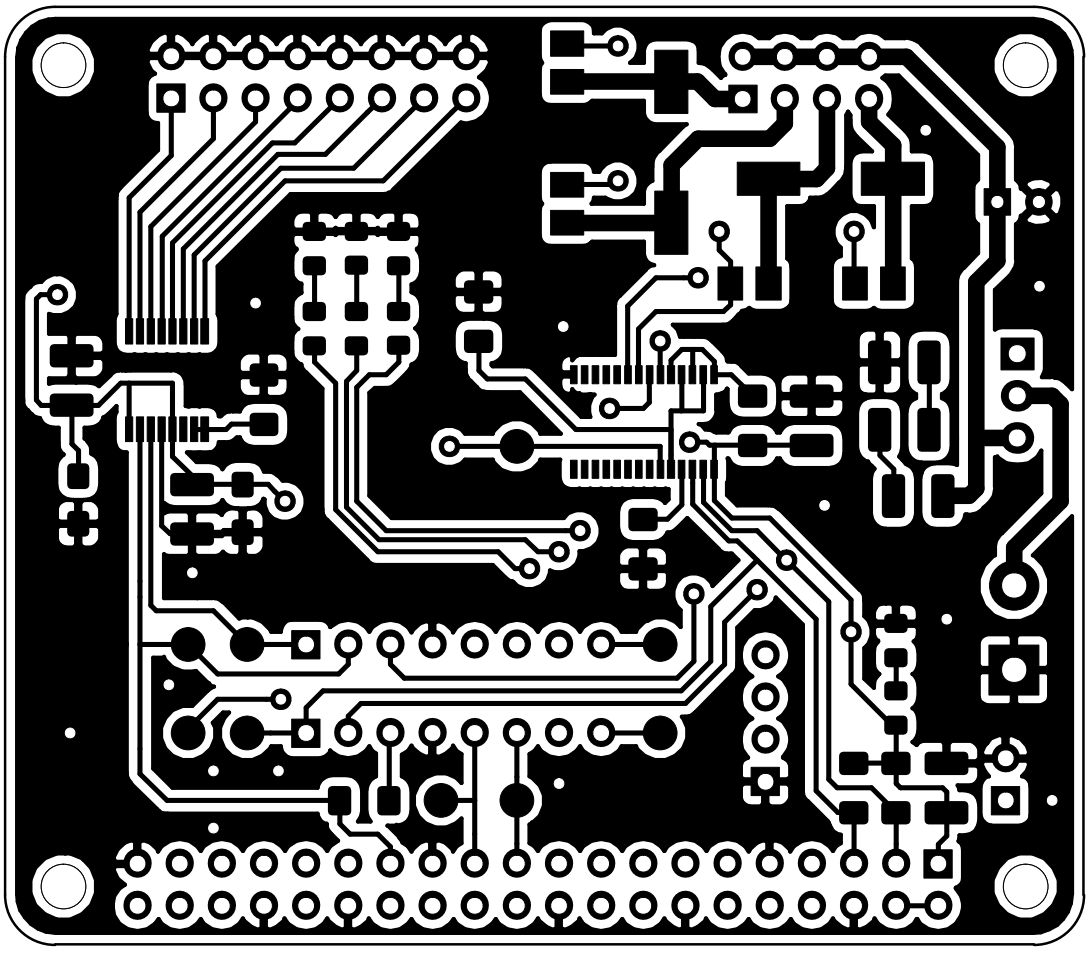
Sheet: /
File: heater_plate.sch

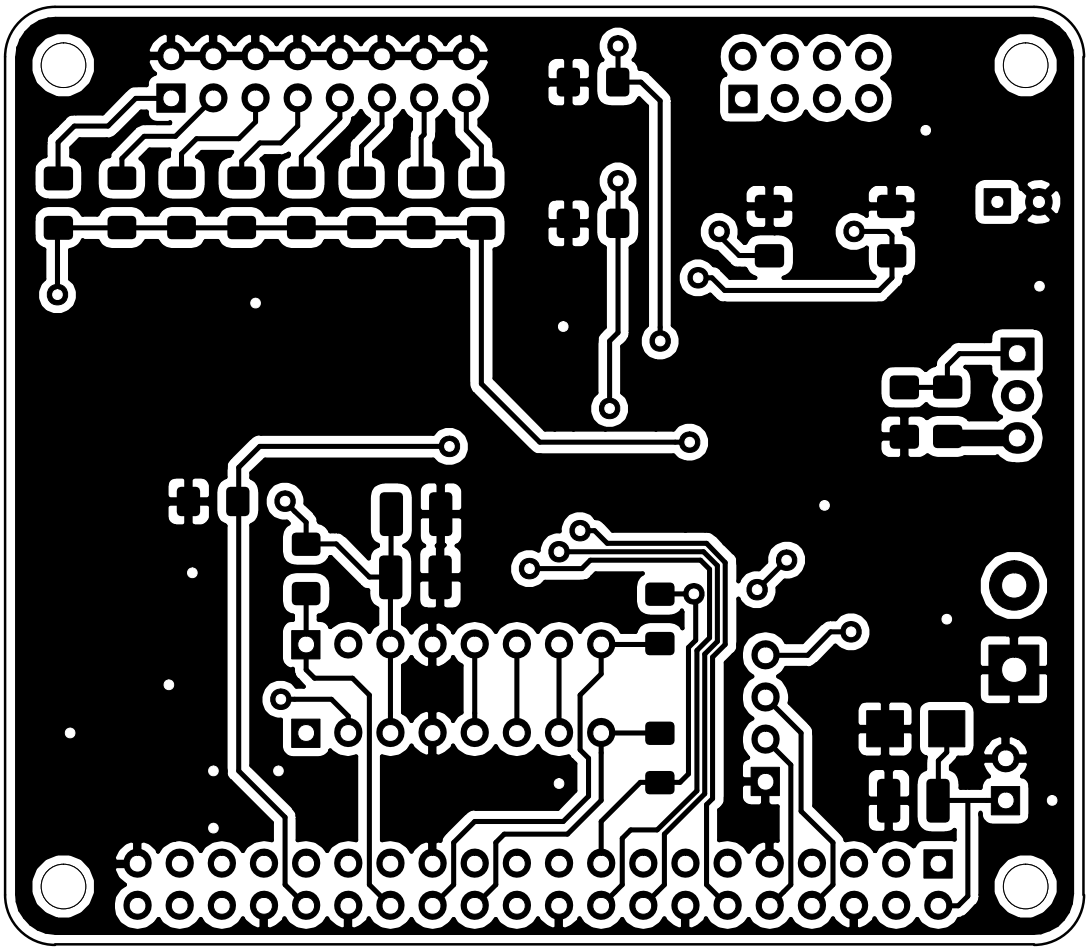
Title: Heater Plate

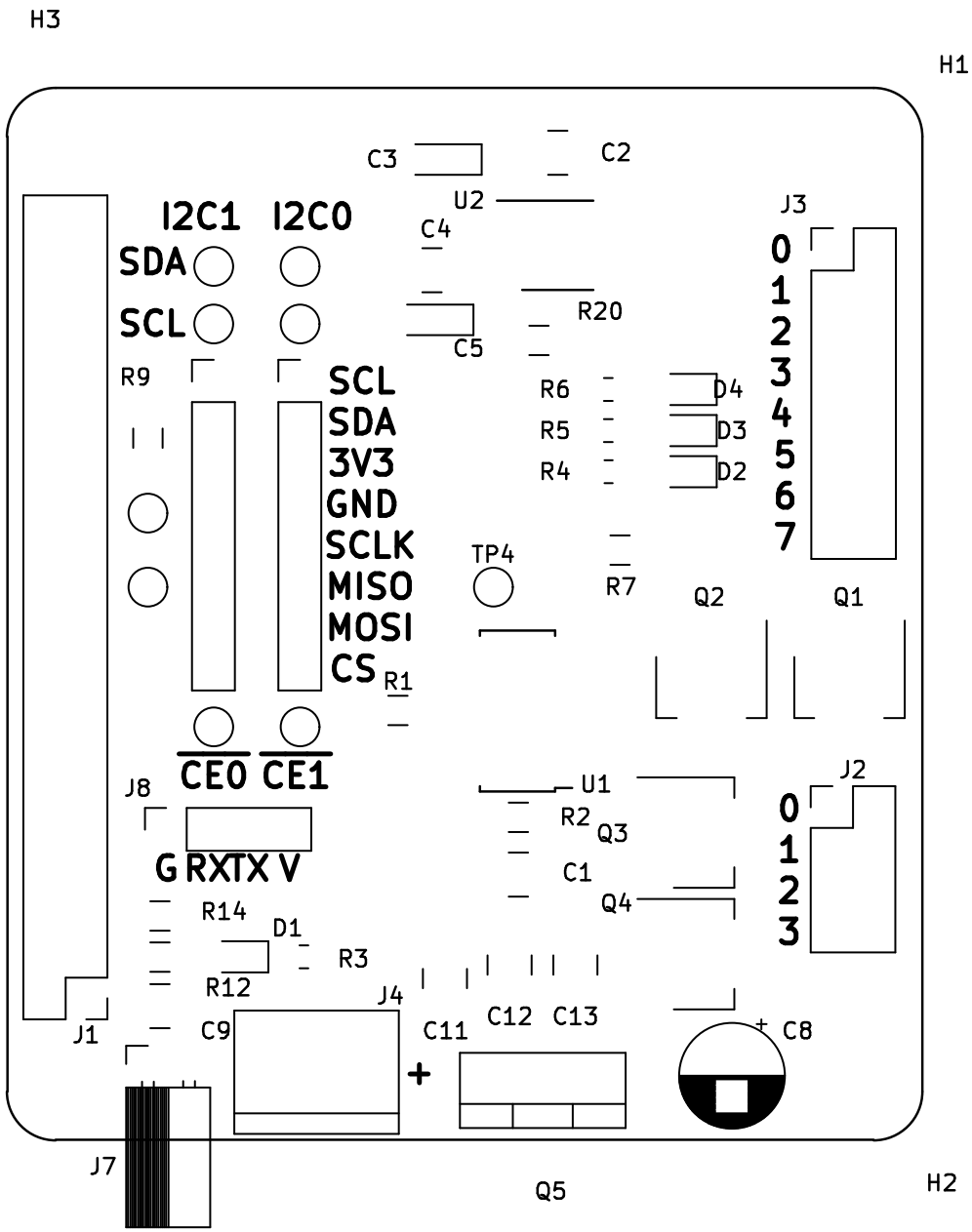
Size: A4	Date:	**Rev:**
KiCad E.D.A. eeschema 5.0.2-bee76a0~ubuntu18.04.1		Id: 1/1

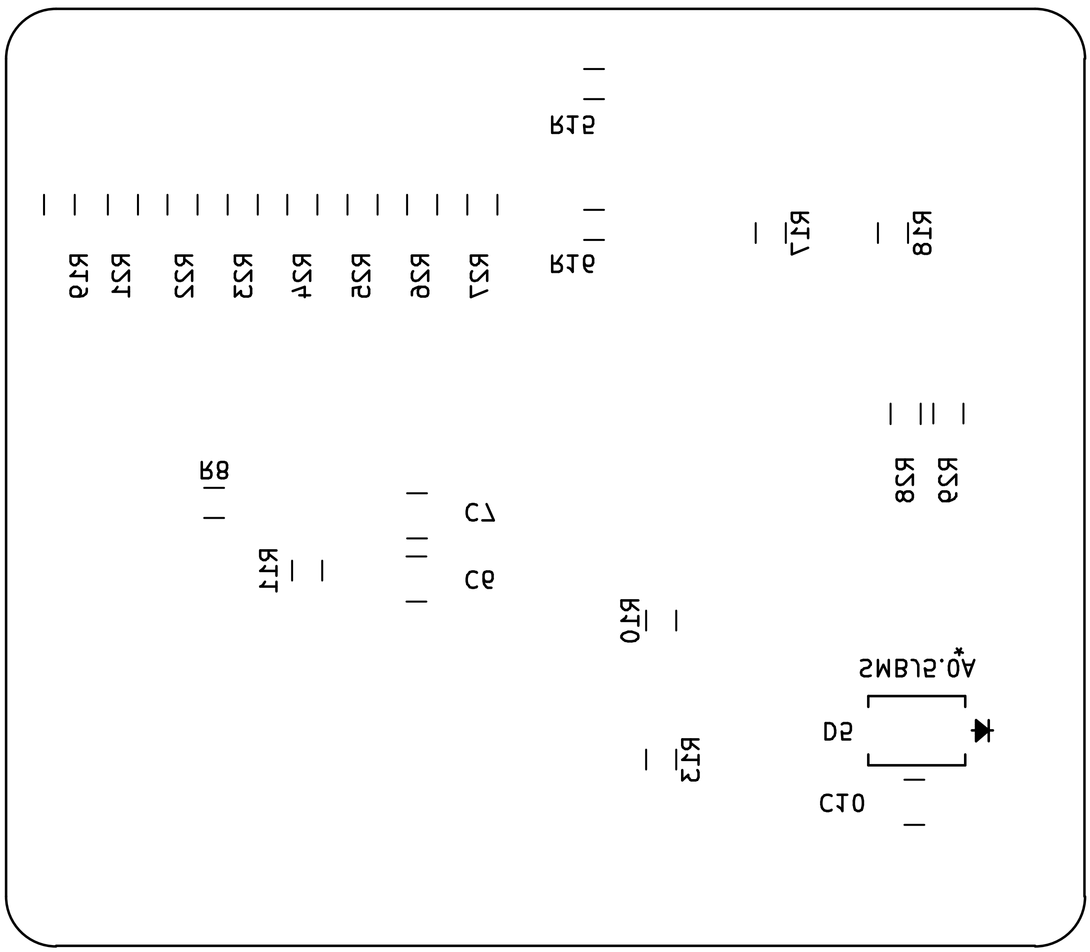
5 Electrical Design PCB Files

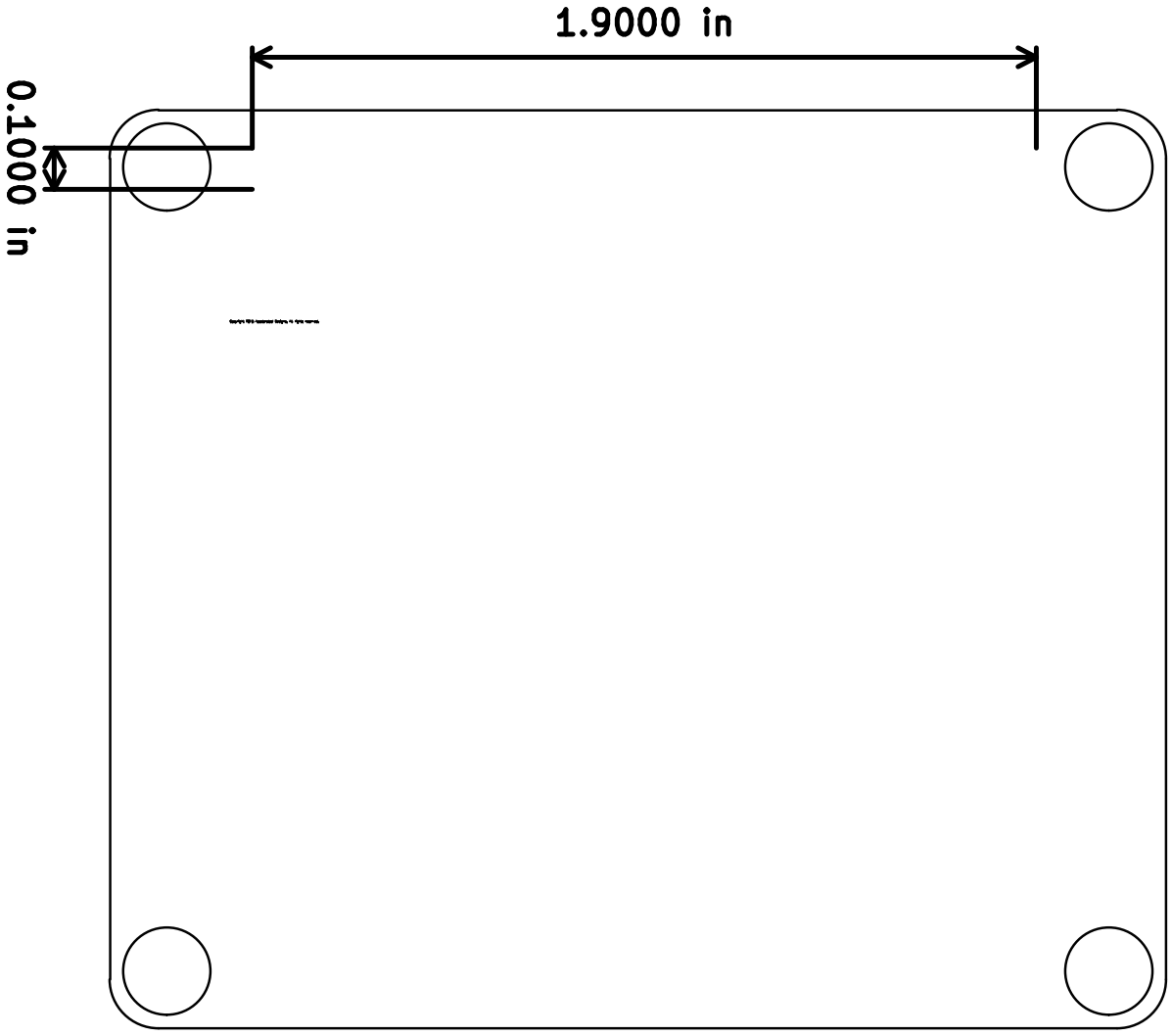
The PCB layouts are secondary to the actual schematics, but I've included them since they took so much time to lay out well.

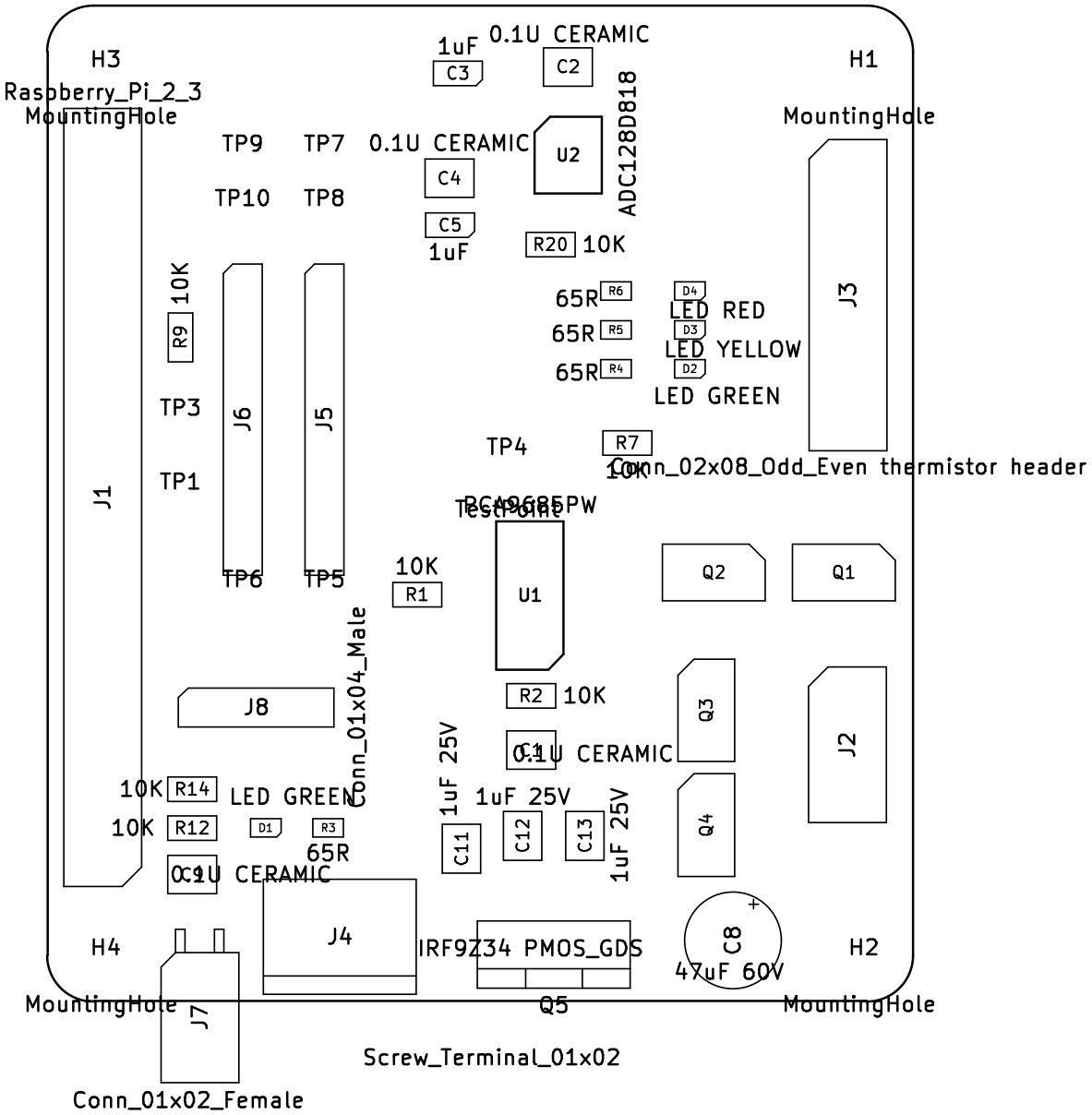




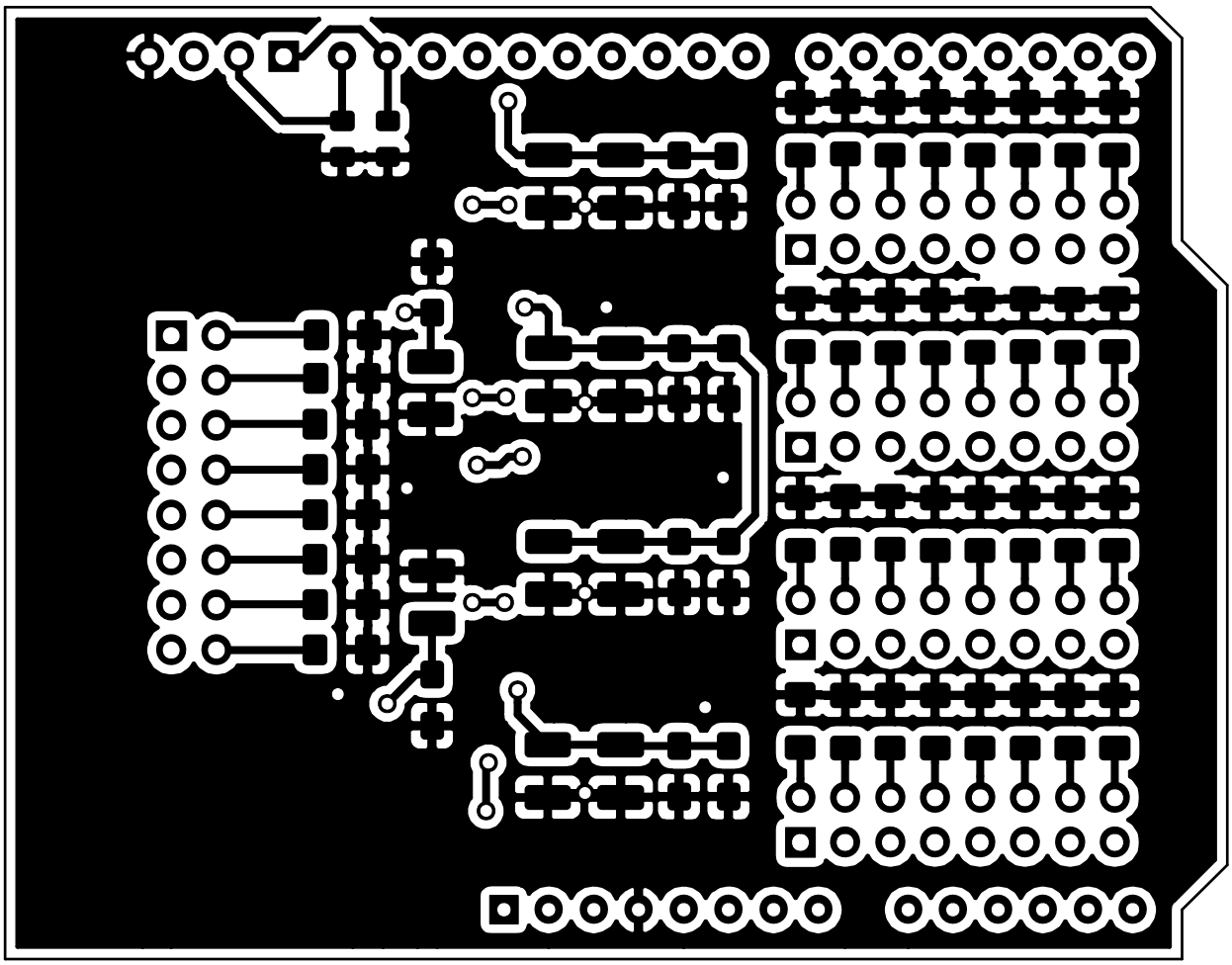

H3
H1
C3
C2
U2
I2C1
I2C0
C4
J3
SDA
SCL
R20
C5
R9
SCL
SDA
3V3
GND
SCLK
MISO
MOSI
CS
R6
R5
R4
D4
D3
D2
0
1
2
3
4
5
6
7
TP4
R7
Q2
Q1
R1
CE0
CE1
J8
U1
J2
R2
Q3
G
RXTX
V
C1
Q4
0
1
2
3
R14
D1
R3
R12
J4
J1
C9
C11
C12
C13
C8
J7
Q5
H2

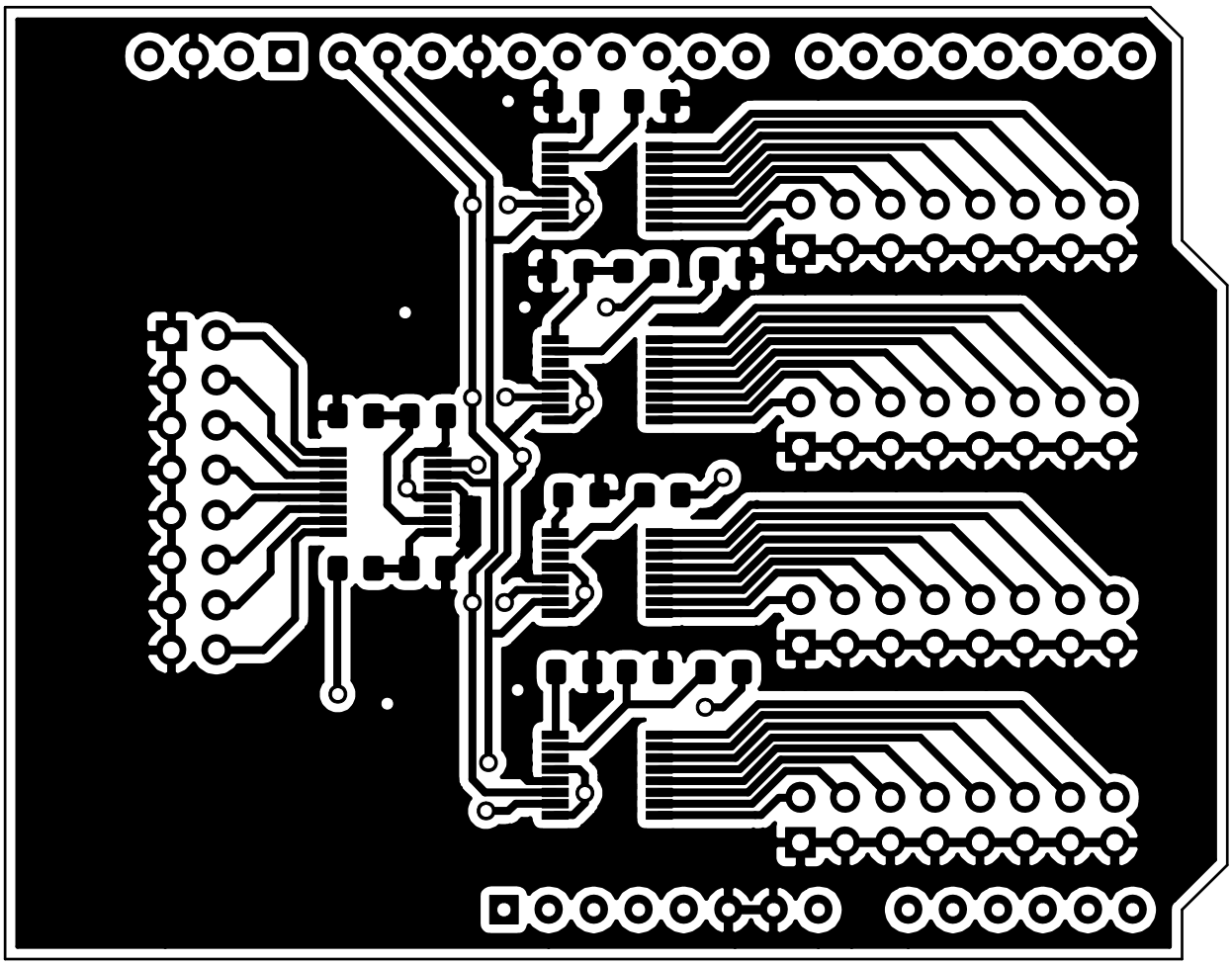

R15
R16
R17
R18
R19
R21
R22
R23
R24
R25
R26
R27
R28
R29
R8
R11
C7
C6
R10
R13
SMBJ5.0A
D5
C10


1.9000 in
0.1000 in

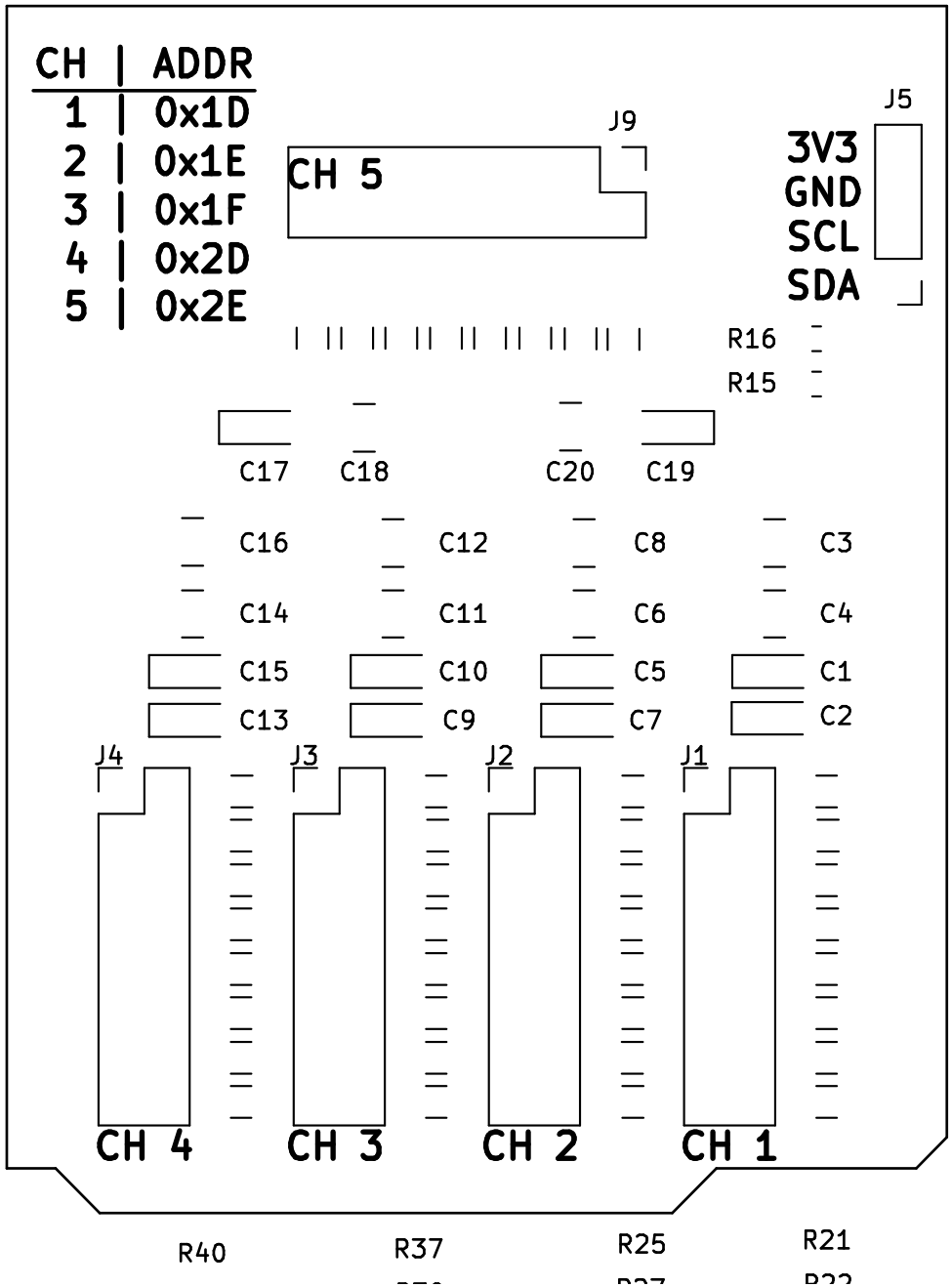

H3
Raspberry_Pi_2_3
MountingHole
H1
MountingHole
1uF
C3
0.1U CERAMIC
C2
ADC128D818
U2
TP9
TP7
0.1U CERAMIC
C4
TP10
TP8
C5
1uF
R20
10K
65R
R6
D4
LED RED
65R
R5
D3
LED YELLOW
65R
R4
D2
LED GREEN
J3
10K
R9
TP3
J6
J5
TP4
R7
10K
Conn_02x08_Odd_Even thermistor header
TP1
J1
TestPoint
BCA9685PW
U1
TP6
TP5
10K
R1
Q2
Q1
Conn_01x04_Male
J8
R2
10K
Q3
J2
1uF 25V
C1
0.1U CERAMIC
10K
R14
LED GREEN
10K
R12
D1
R3
65R
C11
1uF 25V
C12
C13
1uF 25V
Q4
0.1U CERAMIC
C9
H4
J4
IRF9Z34 PMOS_GDS
C8
47uF 60V
H2
MountingHole
J7
Q5
MountingHole
Screw_Terminal_01x02
Conn_01x02_Female

R15
10K
R16
10K
R17
10K
R18
10K
R58
18K 1%
R57
18K 1%
R56
18K 1%
R55
18K 1%
R54
18K 1%
R53
18K 1%
R52
18K 1%
R51
18K 1%
R8
10K
R11
10K
C8
0.1U CERAMIC
C11
0.1U CERAMIC
R10
10K
R12
10K
C10
0.1U CERAMIC
*

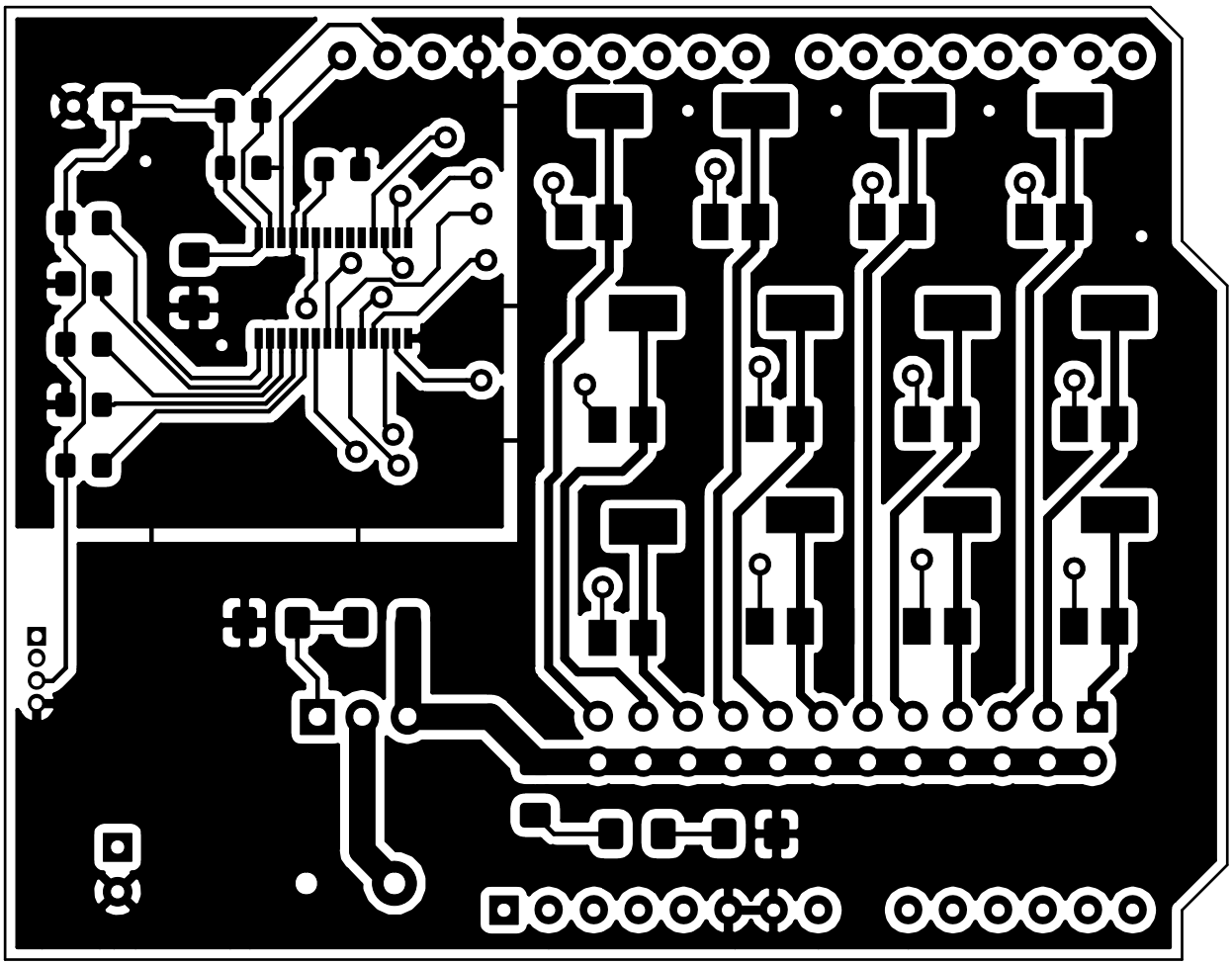


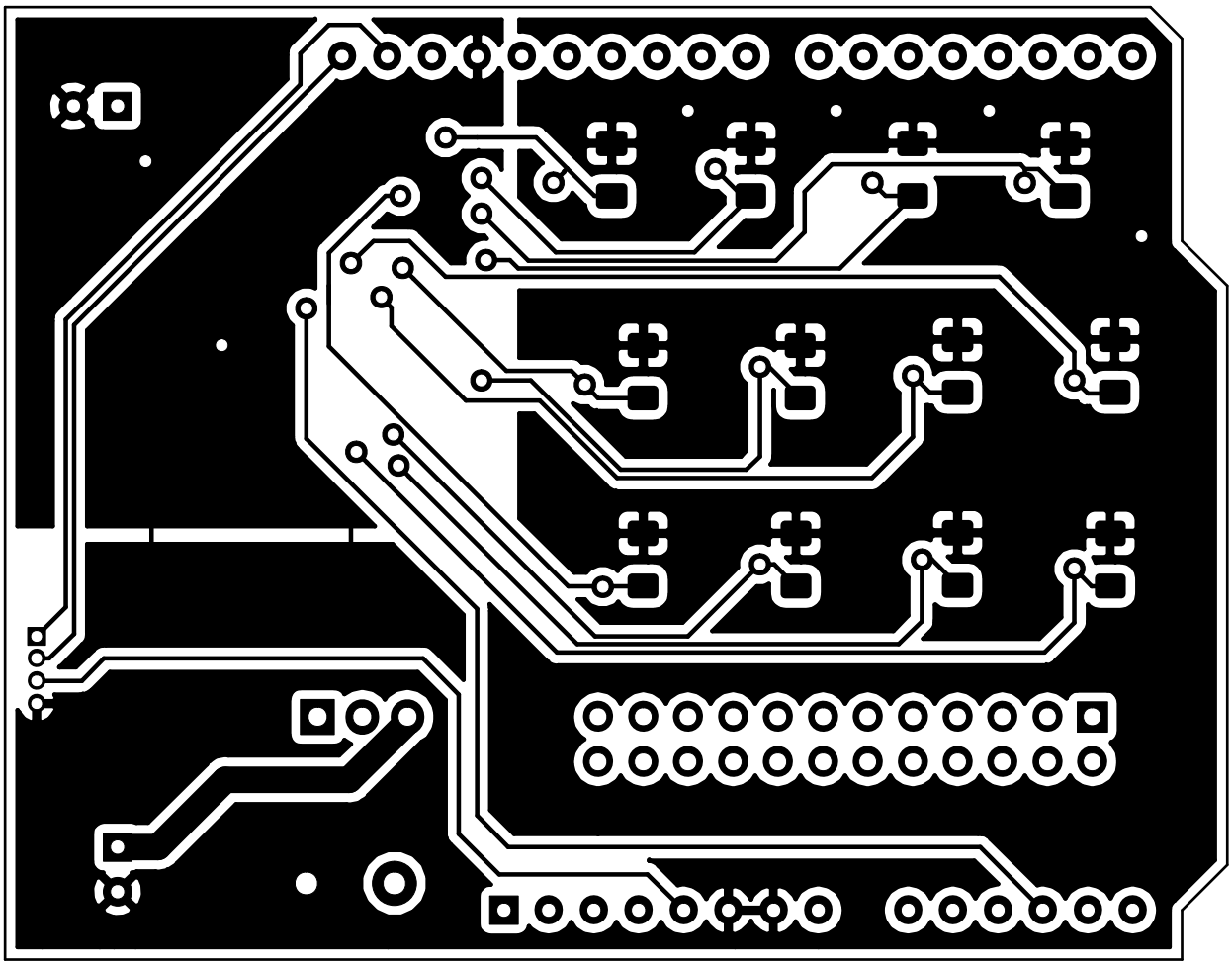


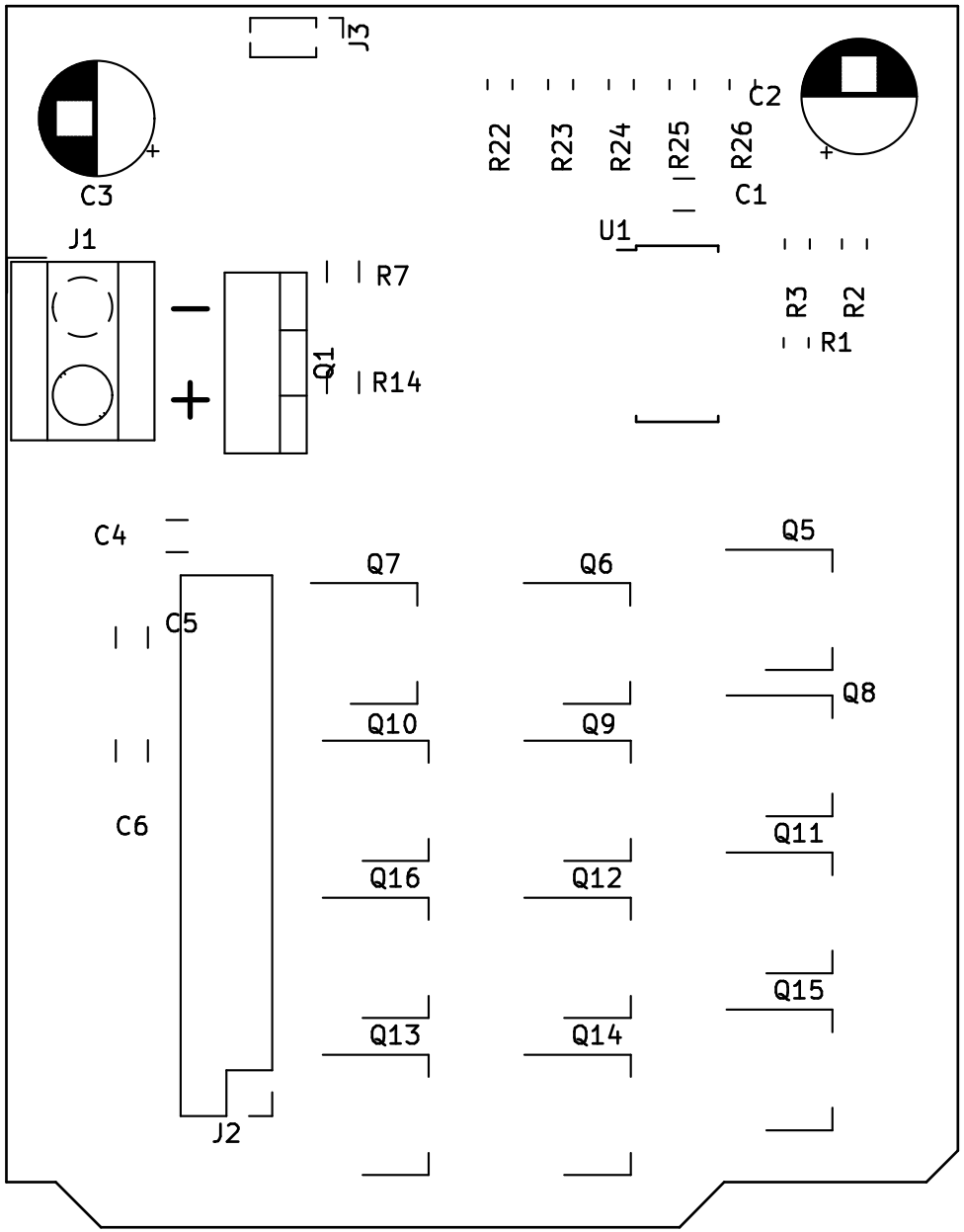
R59 R58 R57 R56 R55 R54 R53

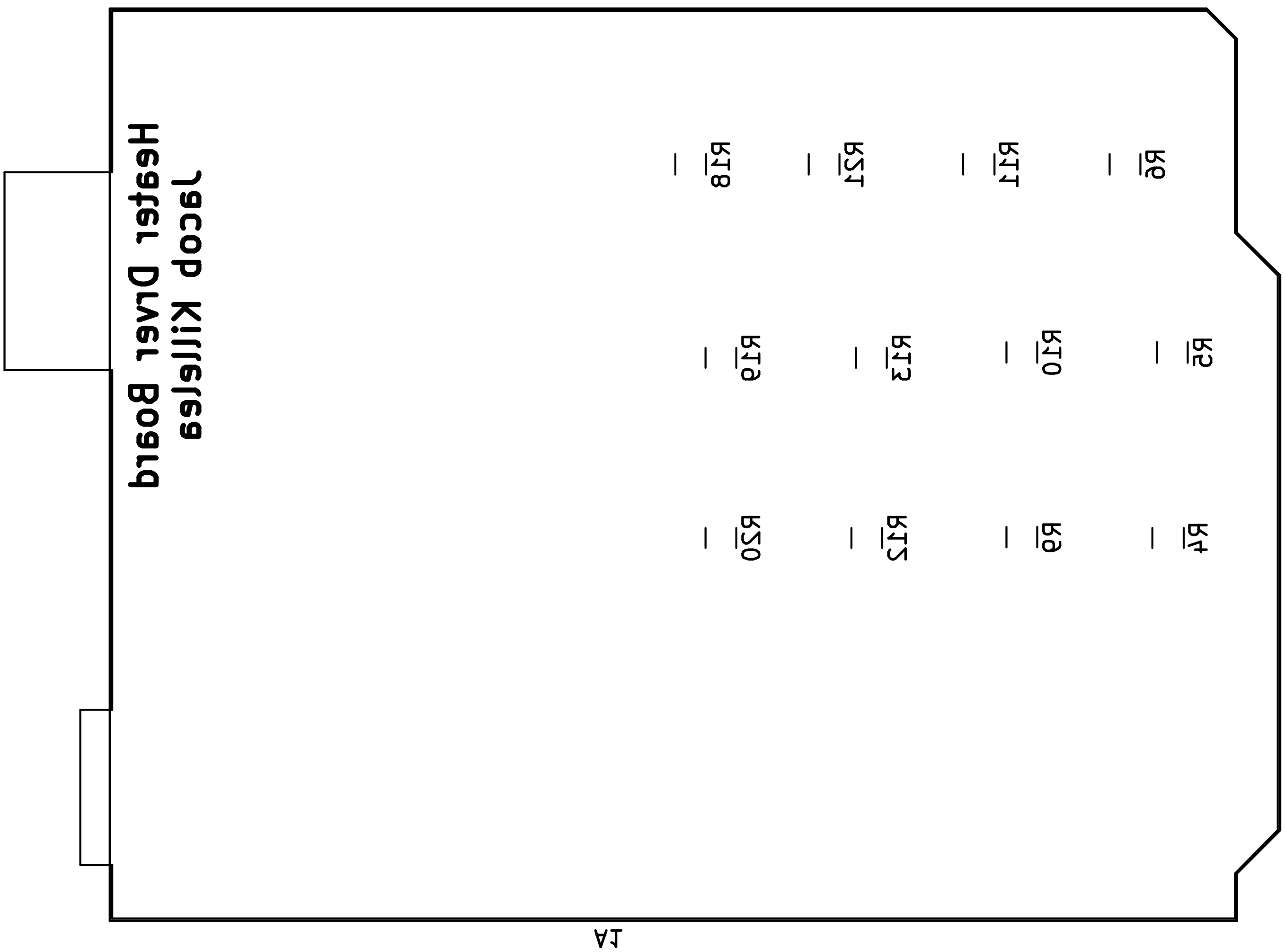


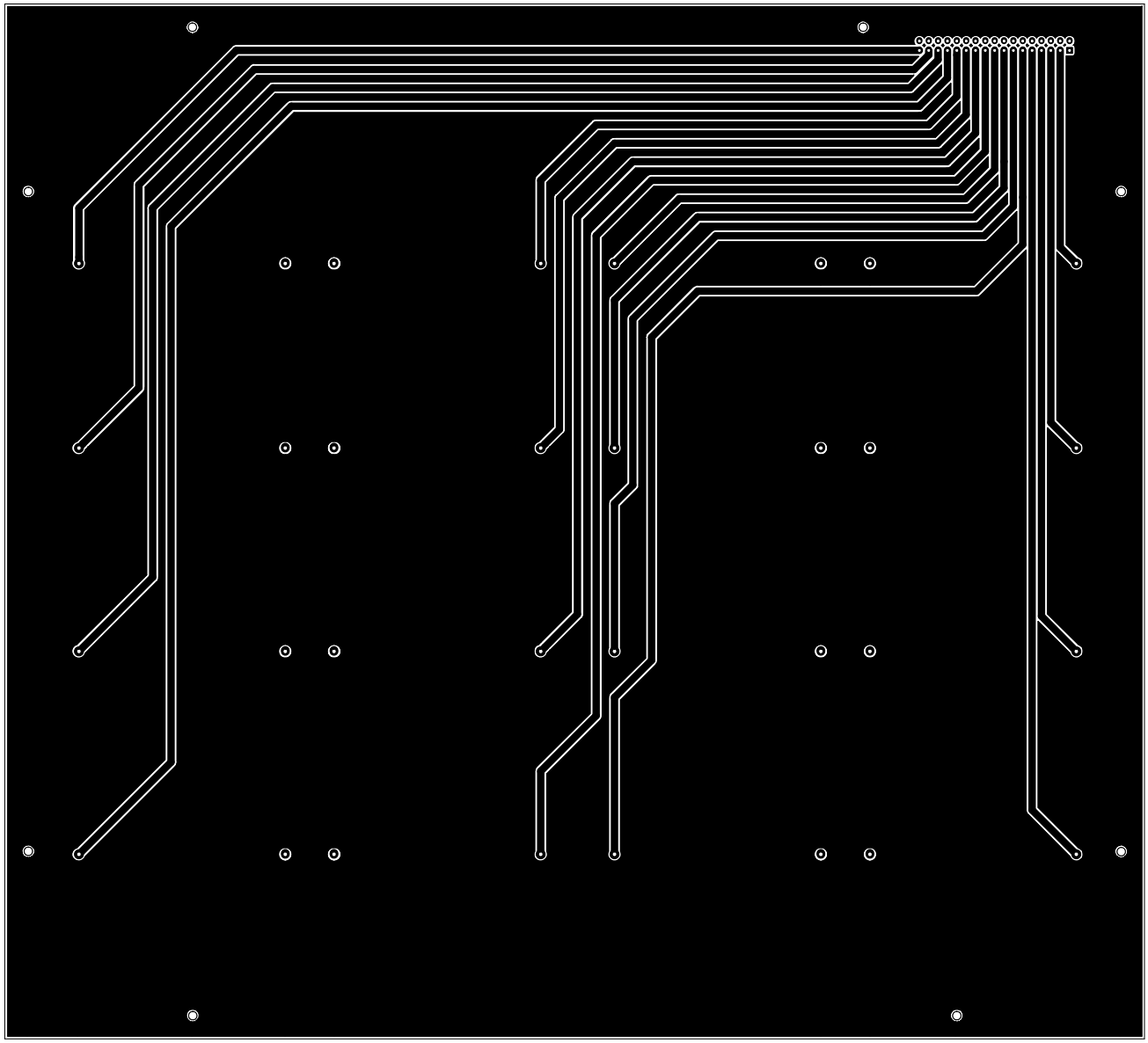
ADC128D818 Thermistor Board
Jacob Killelea
R1
R2
U1
R3
R4
R5
U2
R14
R13
U5
R7
R6
U3
R11
R12
R8
R10
U4
R9
A1

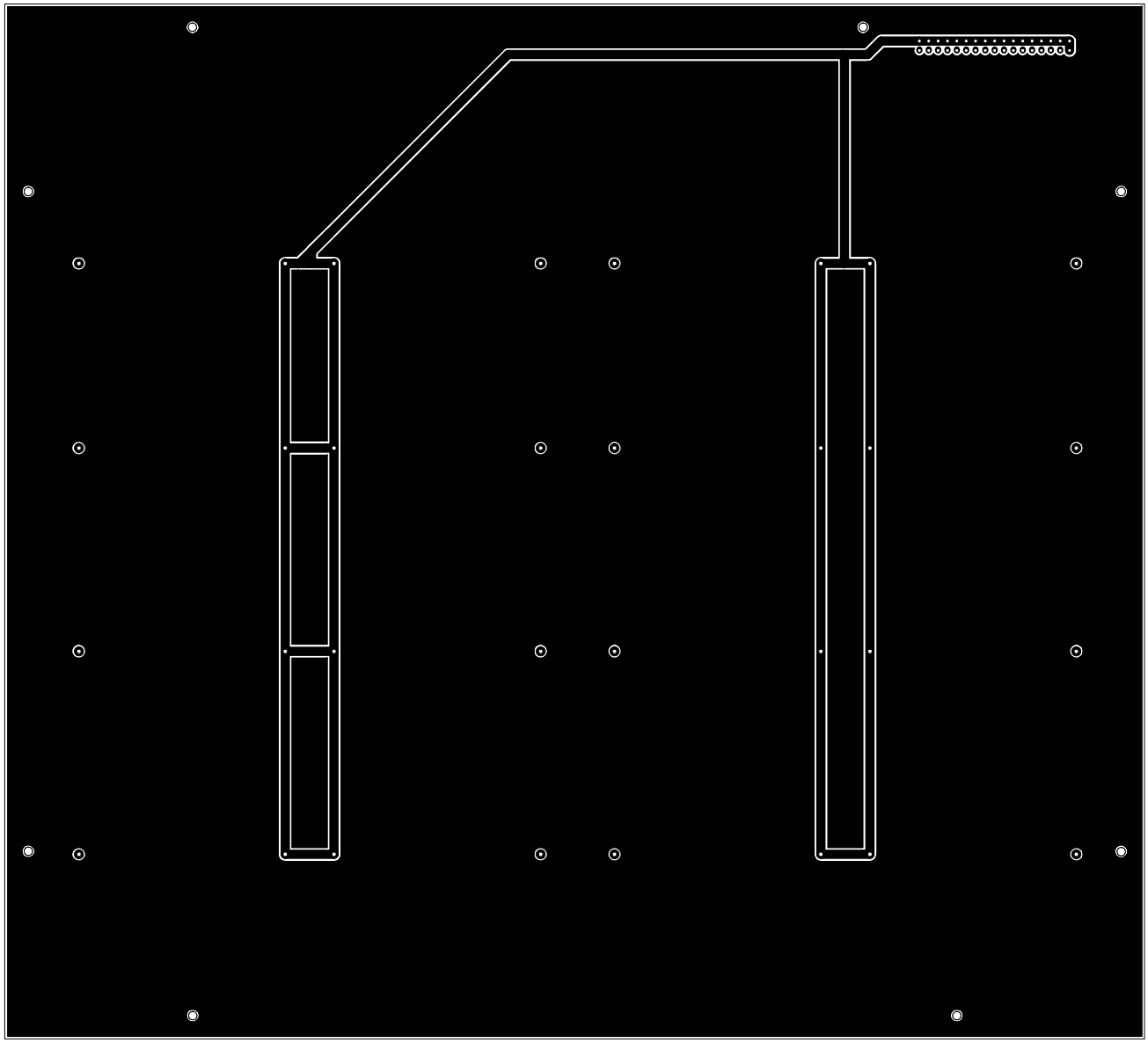


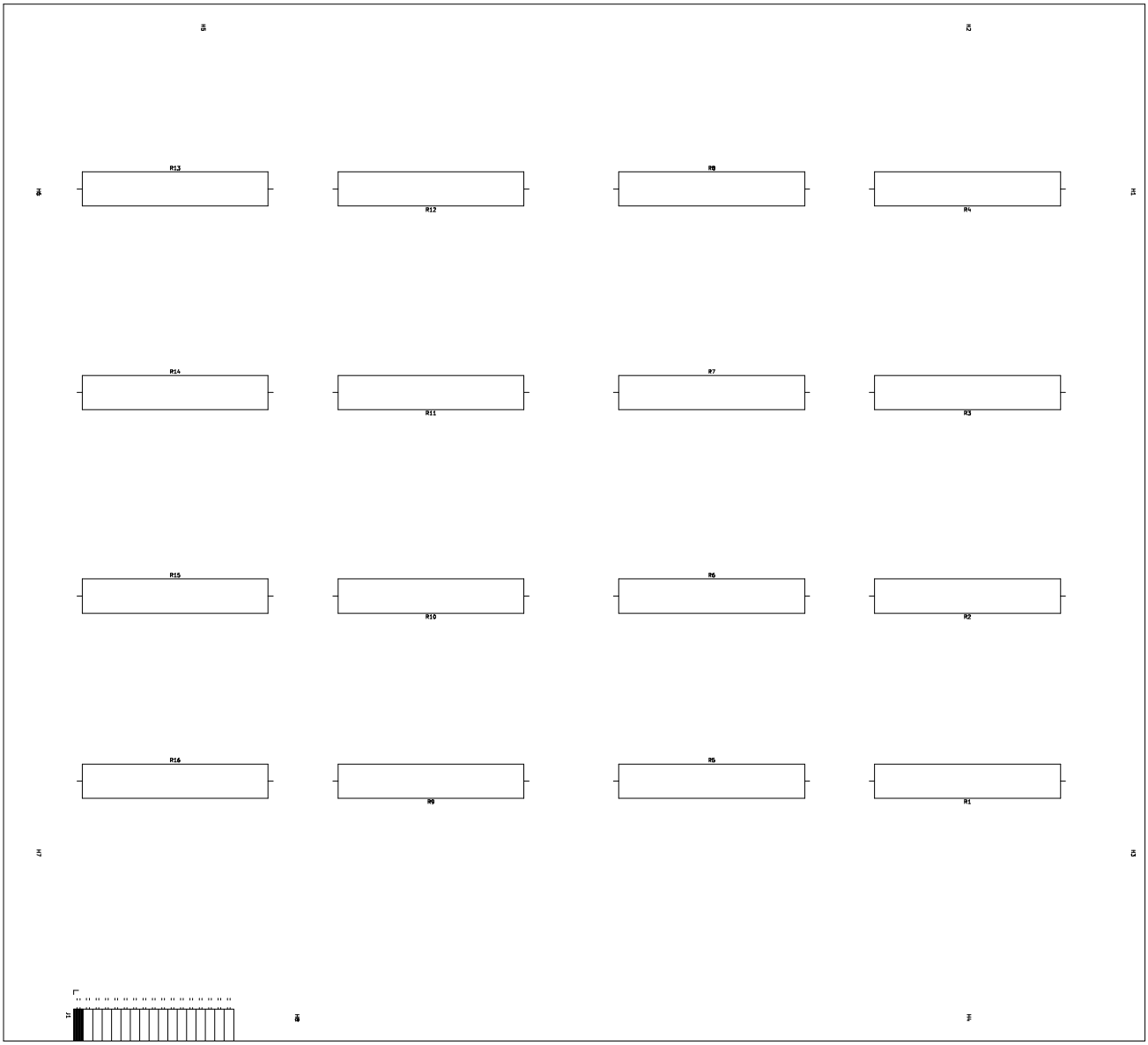



J3
C3
C2
C1
R22
R23
R24
R25
R26
U1
J1
R7
Q1
R14
R3
R2
R1
C4
C5
C6
Q7
Q6
Q5
Q8
Q10
Q9
Q11
Q16
Q12
Q15
Q13
Q14
J2


Heater Drver Board
Jacob Killelea
R6
R11
R21
R18
R5
R10
R13
R19
R4
R9
R12
R20
V1






R13
R12
R8
R4
R14
R11
R7
R3
R15
R10
R6
R2
R16
R9
R5
R1
J1