EDP Report, Group 10

Stefan Vučković, Benjamin Parsons-Willis, Sarah Al Dhuwaihi, Dingyuan Wang

Semester 2, 2023

Abstract

Measuring, Calculating and Displaying Heart Rate

1 Introduction

Our task this year for Electronic Design Project 2 was to produce a heart rate monitor from scratch in a group work context. Though we were provided with the parts we'd need to use in our final circuits, and were provided with the facilities for creating the PCB, the majority of the work was ours to do. This included calculating and recalculating resistor/capacitor values for an active band-pass filter, determining the correct connection arrangement for the display, and the creation of software to read the filtered signal, and determine a heart-rate value, and finally push that value to the display.

2 Methodology

As a persons' heart beats, it pushes oxygenated blood around the body. Oxygenated blood refracts light differently, resulting in a changing opacity, and this difference can be measured using an LED and a photo-diode.

The LED and photo-diode are placed into a plastic shell, each on the opposite side of the subject's finger. When the LED is driven, the photo-transistor will produce a small signal as the subject's heart beats.

The signal produced is minute and noisy, not ideal for measuring with a micro-controller. In order to clean this signal up, and enlarge the element of the signal we want to measure, we use a two-stage active band-pass filter, made up of two op-amps[1], with resistor and capacitor values assigned in such a way that signals between 1-3Hz are amplified with a gain of 4, whereas signals outside of that range are filtered out.

This filtered and amplified signal is then measured by the ADC of the micro-controller[5], which does further noise-reduction on the signal, and then calculates a heart-rate value by finding the maximum turning point of the signal, and measuring the peak-to-peak period, then calculating the frequency of the heart-rate, and thus the BPM.

This BPM value is then pushed to the display controller, which parses the serial communication from the micro-controller, and pushes the correct signals to the display matrix, thus displaying the subject's heart rate.

This principle is known as Photoelectric Plethysmography, (PPG).

[1] https://moodle.gla.ac.uk/pluginfile.php/6047727/mod_resource/content/1/MCP6291-Family-Data-Sheet-DS20001812G.pdf

3 Development

3.1 Sensor

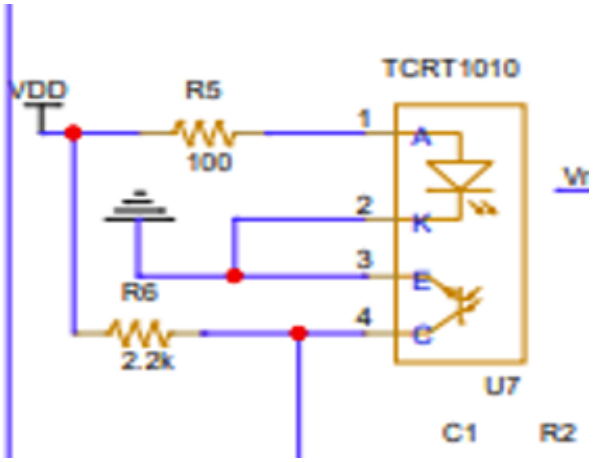

Figure 1: Sensor Circuit

We were provided a plastic shell with the LED[2] and photo-transistor[3] pre-mounted, with two connectors, each with 2 pins, for easy connection to the breadboard. Since the final circuit will be driven by 5V regulated down from a 9V battery, we tested the sensor initially with a supply voltage of 5V.

The PPG technique is most effective using wavelengths from 600-900nm, the LED[2] we were provided has a dominant wavelength of 620nm[2].

The LED required an input voltage of $2.5V$[2], and an input current of $20 - 30mA$[2]. In order to limit the current through the LED[2] we used a ballast resistor (R5) with a value of:

$$R = \frac{5V - 2.5V}{25mA} = \frac{2.5V}{0.025A} = 100\Omega$$

The photo-transistor[3] required a bias resistor (R6), through experimentation, we found that a value of $2.2k\Omega$ produced the clearest and most distinct signal.

Once we started testing the PCB, we quickly discovered that a mistake had been made on the schematic, and the LED[2] was powered from VDD, which did not have a sister connection anywhere on the board. It should have been P5V, and so we soldered a jumper wire between the two traces, solving the issue.

[2] https://moodle.gla.ac.uk/pluginfile.php/6047725/mod_resource/content/1/Kingbirght-red-led.pdf
[3] https://moodle.gla.ac.uk/pluginfile.php/6047726/mod_resource/content/1/bpv11.pdf

3.2 Power Supply

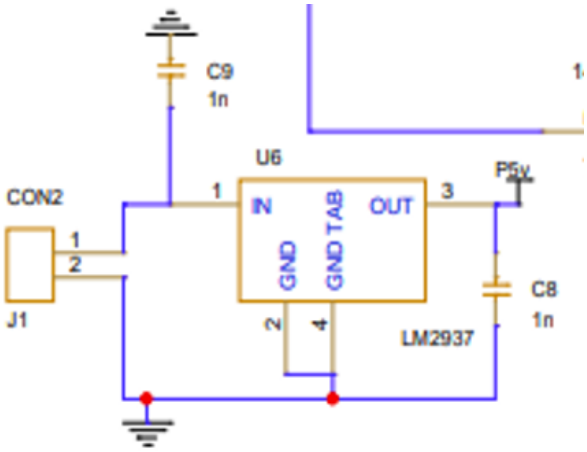

Figure 2: Power Supply Circuit

The base supply for the circuit is a 9V source, capable of driving at least 150mA; This base voltage is then regulated down to 5V using a voltage regulator[4], then 3V3 by the regulator present on the micro-controller[5], and then down to approximately 1.6V using a voltage divider, providing a reference voltage for the active band-pass filter[1].

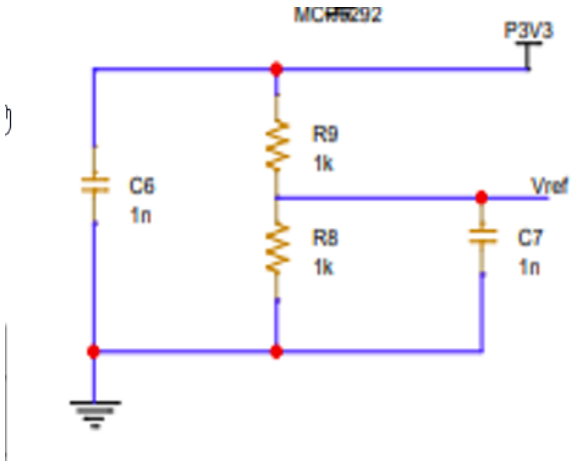

Figure 3: Voltage Divider Circuit

When testing on the final PCB, we discovered periodic noise in all of these supplies, which we realised was due to the Serial Clock of the display leaking into the 3v3 supply from the micro-controller[5], as well as 2 faulty voltage regulators[4] in a row, that provided 8.9 V output instead of the expected 5V.

This is covered in more detail in the Amplification & Filtering section.

[4] https://moodle.gla.ac.uk/pluginfile.php/6047729/mod_resource/content/1/LM2937.pdf

[5] https://moodle.gla.ac.uk/pluginfile.php/6047724/mod_resource/content/2/Freedom%20Board%20User%20Manual%201651277.pdf

3.3 Display

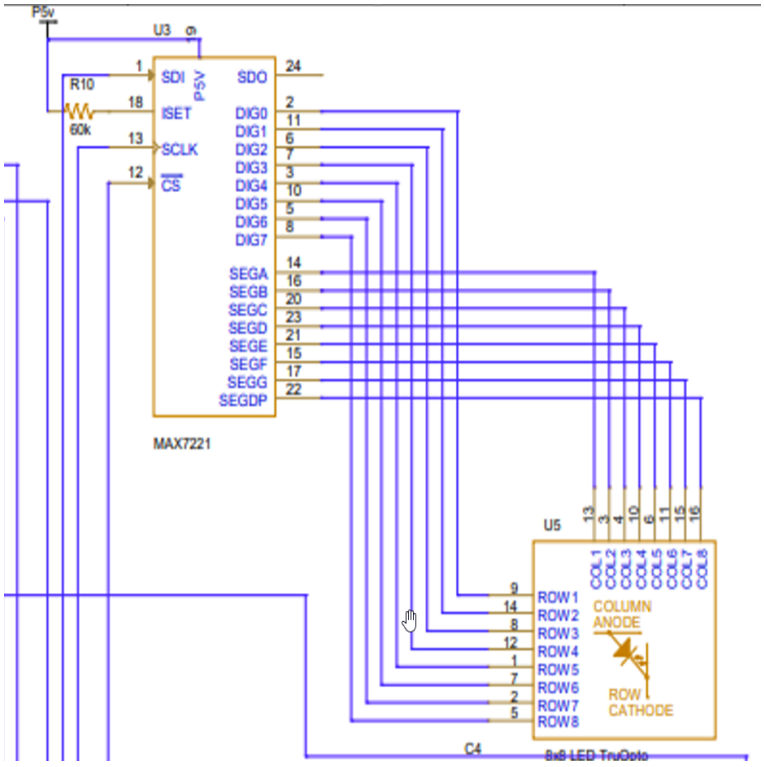

Figure 4: Display Circuit

For display, we were provided with an 8 by 8 LED matrix display[6], as well a MAX7221 display driver[7], and a 74HCT04[8] level shifter.

The level-shifter[8] is used to convert the 3v3 logic output of the micro-controller[5], to 5V logic for use with the MAX7221[7]. The MAX7221[7] display-driver is used to convert the serial output of our micro-controller[5] into the correct corresponding signals on the display matrix[6].

We were expected to figure out how to connect these components correctly from their respective data-sheets, and were given a breakout board[9] in order to test this. We were able to get a valid display output on this breakout board, and added these connections to the schematic.

Once we were able to test on the final PCB, we discovered immediately that the display[6] did not function as expected, and was unresponsive.

Initially this was due to a short affecting the entire board, however, even after this short was fixed, the display[6] still did not function. After many hours of troubleshooting, we realised that the display-driver[7] had been soldered into the PCB upside-down, and thus was absolute nonsense.

After painstakingly re-soldering the entire display-driver[7], we started to get output from the display[6], however it was offset by a single column, implying a small mistake in the connections defined on the schematic. We decided that this could be worked around in software however, and moved on.

[6] https://moodle.gla.ac.uk/pluginfile.php/6047737/mod_resource/content/1/Tru-opto-8x8-display.pdf
[7] https://moodle.gla.ac.uk/pluginfile.php/6047736/mod_resource/content/1/MAX7219-MAX7221.pdf
[8] https://moodle.gla.ac.uk/pluginfile.php/6047735/mod_resource/content/1/74HC_HCT04.pdf
[9] https://moodle.gla.ac.uk/pluginfile.php/6047734/mod_resource/content/1/8x8%20LED%20Display%20Breakout%20Board.pdf

3.4 Amplification & Filtering

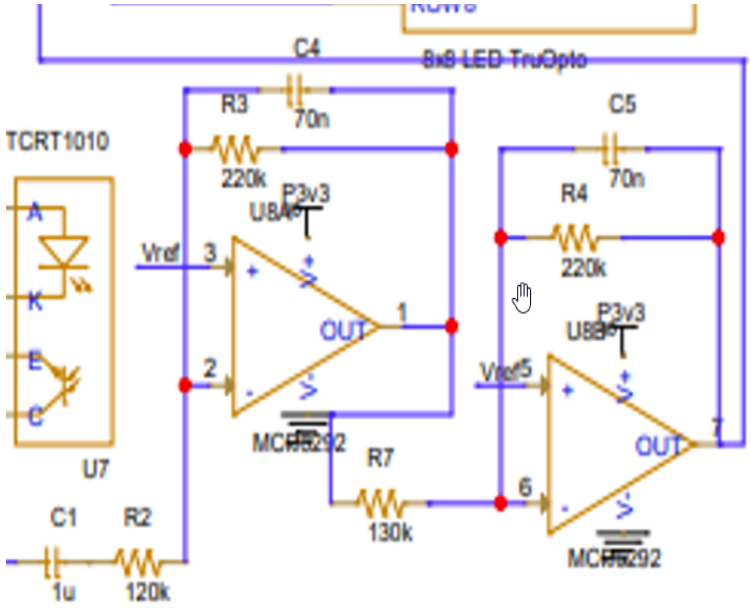

Figure 5: Filter Circuit

The signal from the sensor module has a very low amplitude, and a large amount of noise, primarily 50Hz mains noise due to a lack of shielding. In order to deal with this, we need to filter out unwanted frequencies, and amplify the desired frequencies enough for the micro-controller[5] to detect.

Our initial amplified frequency range was 1.32Hz - 10Hz, resulting in the following resistor/capacitor values:

$Stage\ 1\ Low-Pass\ Filter:\ (R2\ \&\ C1)$
$FC_1 \approx 1.3 = \frac{1}{2\pi * R2 * C1}$

$R2 \approx 120\ k\Omega$
$C1 \approx 1\ uF$

$Stage\ 1\ High-Pass\ Filter:\ (R3\ \&\ C4)$
$FC_2 \approx 10Hz = \frac{1}{2\pi * R3 * C4}$
$R3 \approx 220\ k\Omega$
$C4 \approx 70nF$

$Stage\ 1\ Gain:\ (R3\ \&\ R2)$
$Gain_1 = \frac{220\ k\Omega}{120\ k\Omega} \approx 1.83$

$Stage\ 2\ High-Pass\ Filter:\ (R4\ \&\ C5)$
$Replicates\ Stage\ 1,\ R4 = 220\ k\Omega$
$C5 = 70\ nF$

$Stage\ 2\ Gain:\ (R4\ \&\ R7)$
$Replicates\ Stage\ 1,\ R7 = 120 \sim\ k\Omega$
$Gain_2 = -\frac{R4}{R7} \approx 1.38$

$Gain\ @\ 1.3Hz - 10Hz\ \approx\ Gain_1 \times Gain_2$
$\approx 1.38^2 \approx 1.9044$

Overall, the filter was the most temperamental component of the project. We experimented with many different values, and the above ended up being non-ideal. When constructing the circuit, we initially had no output from the filter. We found that the first stage output a moderately amplified signal, but the second stage did not. Eventually we realised that someone had forgotten to write 130k on the schematic for R7, but instead wrote 130, and that resistor had been soldered to the board. We realised that this caused the second op-amp[1] to clip, as the gain was far higher than the power supply was capable of, and output no signal.

After fixing this error, we had a moderately amplified signal from the second op-amp[1], however there was still periodic behaviour, where the filtered signal would suddenly increase in voltage, and then immediately drop to ground, and then recover back to a stable value again. We found that one of the demonstrators, who had a stronger pulse signal, caused this behaviour on every pulse. This implied that the op-amp[1] was still clipping.

The next day, this noise had seemingly become more frequent, but of lower magnitude, and was occurring hundreds of times per second, this made the signal choppy and unworkable. This noise was present at every stage in our circuit, and even the LED power, and so we surmised that the issue lay with the 5V regulator circuit. In order to test this, we acquired a new voltage regulator[4] from Andrew, but the issue was still present.

The amplitude of the noise was very small, and only noticeable due to our low gain, and thus low signal amplitude. So we decided to move on and try to amplify the signal more in order to make the noise inconsequential to the final reading.

We found that increasing the gain of the op-amps[1] worked up to a point, but would then clip. The level to which we were able to amplify the signal before clipping was not enough to get a clear reading through the noise. So we decided to try to find the reason for the clipping.

We disconnected the micro-controller[5] from the PCB to test the filter in isolation, and found that the periodic noise on the 5V line was occurring only when the micro-controller[5] was plugged in, however the filter was still clipping consistently on large pulses.

When inspecting the input signal to the first op-amp, we discovered that the signal had a very high DC offset, roughly 4V! As the op-amp[1] was supplied by a 5V supply (or so we thought), this would cause the signal to be clipped.

Our initial response was to recalculate the bias resistor for the photo-transistor[3], however these values proved consistent, assuming our supply voltage was 5V. We noticed that the voltage from the regulator[4] was actually 8V, not 5V, and started testing different values for the regulator circuit, believing that the regulator[4] was new, and thus known to function correctly.

At this point, the deadline was creeping up, and so we decided to skip the op-amp[1] entirely and instead jump the raw sensor signal into the micro-controller[5], using a 22uF electrolytic capacitor to remove the DC offset. We finished this jump right before the Rankine closed, and planned to solve any issues with the software the next day, mere hours before the due date.

The next day (the due date), we discovered that our readings for the signal amplitude the previous day had been subject to a measurement error, as the probe had been scaled incorrectly. The final signal into our op-amp, directly from the sensor, was still far too small for measurement. At this point we decided to try to systematically re-solder the PCB, in order to track down the issue. Andrew helped us to track down a number of shorts and bad solder joints, and was initially surprised by the DC offset present on the filter input.

Andrew eventually traced the issue to the voltage regulator[4], which was faulty. When we replaced the voltage regulator[4] with a working one, and put a 1uF capacitor on the filter input, the filter ceased clipping, however it was filtering out our desired signal.

We realised that, due to our previous attempts to increase the gain/remove the DC offset, we had inconsistent capacitor and resistor values for our filter. So, 4 hours before the deadline, we recalculated those values.

The average frequency of a PPG signal is between 0.35 & 2Hz[10]. Based on this, we decided on a band-pass filter range of 0.3 Hz & 3.1 Hz, in order to have a range of tolerance.

$$Stage\ 1\ Low-Pass\ Filter:\ (R2\ \&\ C1)$$
$$FC_1 \approx 0.3Hz = \frac{1}{2\pi * R2 * C1}$$

$$R2 \approx 500\ k\Omega$$
$$C1 \approx 1\ uF$$

$$Stage\ 1\ High-Pass\ Filter:\ (R3\ \&\ C4)$$
$$FC_2 \approx 3.1Hz = \frac{1}{2\pi * R3 * C4}$$
$$R3 \approx 1\ M\Omega$$
$$C4 \approx 50nF$$

$$Stage\ 1\ Gain:\ (R3\ \&\ R2)$$
$$Gain_1 = \frac{1\ M\Omega}{500\ k\Omega} \approx 2$$

$$Stage\ 2\ High-Pass\ Filter:\ (R4\ \&\ C5)$$
$$Replicates\ Stage\ 1,\ R4 = 1\ M\Omega$$
$$C5 = 50\ nF$$

$$Stage\ 2\ Gain:\ (R4\ \&\ R7)$$
$$Replicates\ Stage\ 1,\ R7 = 500\ k\Omega$$
$$Gain_2 = -\frac{R4}{R7} \approx 2$$

$$Gain\ @\ 0.3Hz - 3.1Hz\ \approx\ Gain_1 \times Gain_2$$
$$\approx 2^2 \approx 4$$

After re-soldering the op-amp[1] with these expected values, we had an amplified and distinct pulse signal output from the filter. We then repaired the previously scratched input trace, and plugged in the micro-controller[5]. Suddenly, our periodic power-supply noise had returned, and was present in the filtered signal, and we only had 2-3 hours until the deadline.

Figure 6: Noisy Signal

[10] https://www.researchgate.net/figure/Frequency-spectrum-of-PPG-signal_fig3_269333538

We probed the op-amp[1] pins, and found that the noise was present in the reference voltages for both amplifiers. We probed further, and the noise was still present in our 3V3 line, however at this point the regulator was known to be working correctly, and the noise was not present when the micro-controller[5] was not connected.

Upon loading some display test code to our controller, we noticed that the noise was only present when the display was being written to. This implied that there was a short between our display traces, and our 3V3 line. We tracked it down to the Serial Clock line from the controller bleeding into the LED power, spreading this noise throughout the entire board.

Upon fixing this short, we had a clean, distinct heart-rate signal into the micro-controller[5], and we moved on to software with only 2 hours to the deadline.

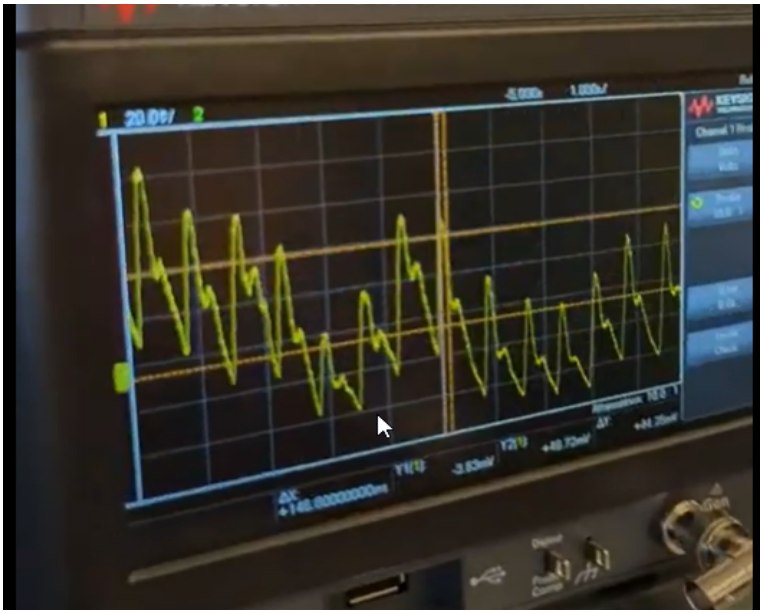

Figure 7: Good Signal

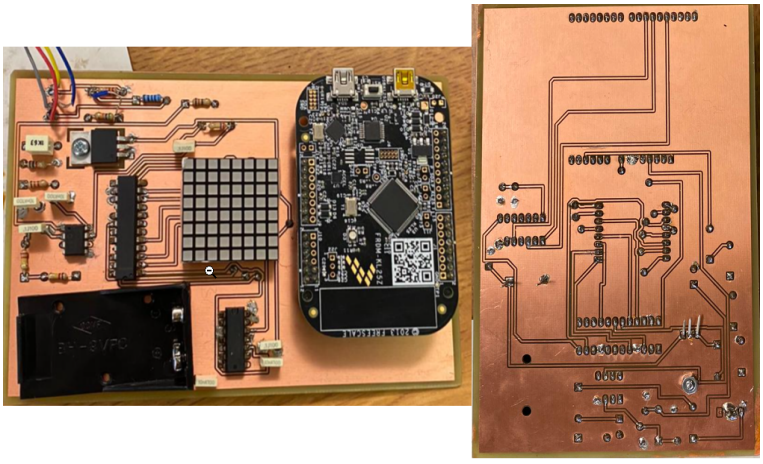

Figure 8: Final Soldered Board

4 Software Design

4.1 Overview

In order to calculate and display heart-rate from our filtered signal, we need to do processing on the signal using a micro-controller[5].

First, the signal needs to be sampled using the ADC of the micro-controller. Then the signal needs to be digitally filtered, normalised, and then the peak-to-peak period needs to be measured, and from that, a BPM can be extrapolated.

4.2 Sampling

The program consists of a continuous sampling loop, these samples are pushed into a sample buffer for later use. Before the loop starts, an initial sample is taken for use with the lag filter as a starter value.

The samples are taken using the ADC.read_u16() method, in order to avoid wasting clock cycles on expensive floating point calculations. The buffer is set to a length of 128, in order to save on expensive floating point division when calculating an average over the buffer, you can instead use a right shift to perform division.

The sampling loop is executed at a set rate, the minimum of which is defined by the Nyquist sampling criterion. The average frequency of a PPG signal is between 0.35 & 2Hz[11], thus the minimum sampling rate is 4Hz.

Therefore, on each loop, the micro-controller waits a maximum of 250ms, in our final implementation, the controller waits 30ms, resulting in a sampling frequency of 66Hz. Through testing, we found this was the best value for our chosen buffer size, balancing accuracy and memory consumption.

The sample buffer is implemented as a circular buffer, using head and tail pointers to track the position of the front and end of the buffer, allowing for easy traversal.

Circular Buffer Implementation:

```
unsigned short pre_filter_buf[8];
unsigned short post_filter_buf[128];
const unsigned short TREND_REMOVAL_INIT_LENGTH = 128;
unsigned char post_filter_buf_head = 0;
unsigned char post_filter_buf_tail = 0;
while (true) {
    }
    unsigned short input = in.read_u16(); // Input from ADC.

    pre_filter_buf[pre_filter_buf_cursor] = input;
    pre_filter_buf_cursor++;
    if (pre_filter_buf_cursor == 8) {
        if (post_filter_buf_tail >= TREND_REMOVAL_INIT_LENGTH) {
            post_filter_buf_tail = 0;
        }
        if (post_filter_buf_head >= TREND_REMOVAL_INIT_LENGTH) {
            post_filter_buf_head = 0;
        }
        if (post_filter_buf_tail == post_filter_buf_head) {
            post_filter_buf_head++;
        }
    }
```

[11] https://www.researchgate.net/figure/Frequency-spectrum-of-PPG-signal_fig3_269333538

4.3 Filtering & Normalization

When measuring noisy/weak signals, our samples can vary wildly. To combat this, we implemented a lag filter, in order to smooth these varied inputs into a smooth continuous function.

```
if (!_do_lag_filter) {
    do_lag_filter = true; // Record initial sample.
    BPM = (60/( timercount * 0.03 ));
} else {
    // Cheap bit arithmetic, saves floating point division.
    BPM = ((60 / (timercount * 0.03)) >> 2) + ((BPM >> 2)+(BPM >> 1));
}
```

Once the sample buffer has filled for the first time, an initial average is calculated by finding the sum of all the samples, and right shifting by 7, in order to cheaply calculate the mean over 128 samples. This average is then continually updated in place without array traversal by new values:

```
// Initial avg calculation
unsigned int avg = 0;
unsigned char cursor = 0;
while (cursor < 128) {
    avg += sample_buf[cursor];
    cursor++;
}
avg = avg >> 7; // Cheap division by 128.
_do_average = true;

// Continuous update
if (_do_average) {
    avg -= avg >> 7;
    avg += sample >> 7;
    sample = sample - avg;
}
```

4.4 Heart Rate Calculation

The heart rate calculation is performed by monitoring the input for rising voltages, and once a threshold number of rising samples is reached, we look for a threshold number of falling samples. This allows us to identify our maximum turning point for each pulse, at which point we read the value of the timer from the last pulse, calculate heart rate, push to display, and restart timer.

The specific threshold values were chosen through testing with a real subject, we decided to use a higher threshold value for the rising element of the signal than the falling element, as the fall is sharp, and the rise is gradual.

Depending on the threshold values and the rate of the measured pulse, this algorithm is prone to missing/misreading pulses, and this caused some spurious misreadings in the final implementation, however, overall this algorithm was accurate and produced a BPM reading that generally agreed with the smart watch we used to test it.

This algorithm was based on the concept of a Schmitt Trigger[12]

[12] https://en.wikipedia.org/wiki/Schmitt_trigger

4.5 Display

To display heart-rate, we need to divide the display into subsections, each holding a single digit, with the leftmost only able to display 1.

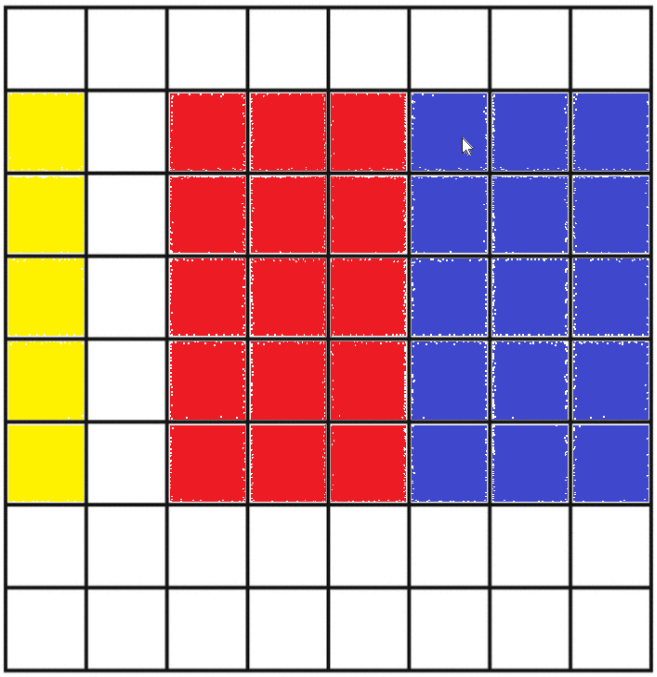

Figure 9: Display Subsections

By dividing our display in this way, we are able to represent values between 0 and 199. This is acceptable for our expected heart-rate range. We then define 5x3 bit arrays for each possible digit, from 0-9, and place these in an array, in their respective index, for easy access.

```
// Small exerpt from char array, real array is too large for report.
bool chars[10][15] = {
        {
                1,1,1,
                1,0,1,
                1,0,1,
                1,0,1,
                1,1,1
        },
        {
                0, 1, 0,
                0, 1, 0,
                0, 1, 0,
                0, 1, 0,
                0, 1, 0
        } // and so on...
}


// Get place values.
unsigned char hundreds = to_display / 100 % 10;
unsigned char tens = to_display / 10 % 10;
unsigned char ones = to_display % 10;
```

We re-use the serial display code provided on Moodle, providing it with a newly generated char array on every display update. In order to generate this array, we iterate over the middle 5 rows of the display, creating a new char for each.

```
unsigned char build[5] = {
            0x00,
            0x00,
            0x00,
            0x00,
            0x00
};
for (char i = 0; i < 5; i++) {
    unsigned char line = 0;

    if (hundreds == 1) {
        line += (1 << 6);
    }

    line += (chars[tens][(i*3)] << 4);
    line += (chars[tens][(i*3)+1] << 3);
    line += (chars[tens][(i*3)+2] << 2);
    line += (chars[ones][(i*3)] << 1);
    line += (chars[ones][(i*3)+1]);
    line += (chars[ones][(i*3)+2] << 7);
    build[i] = line;
}
unsigned char final[8] = {
    head_and_foot[0],
    head_and_foot[1],
    build[0],
    build[1],
    build[2],
    build[3],
    build[4],
    head_and_foot[2]
};
pattern_to_display(final);
```

In an ideal case, for each row, we set the first bit of the char if the number to display is greater than or equal to 100, and we leave the second bit unset. The next 3 bits are set to their respective pre-defined row values, for the tens place, and the next 3 bits are set to their respective values for the ones place.

When testing our display on the final board, we discovered a small error in the PCB design had caused the entire display to be offset by a single bit. Thus we offset the above array generation by a single bit in the opposite direction, correcting for the error.

5 PCB Design

5.1 Masks, Top & Bottom

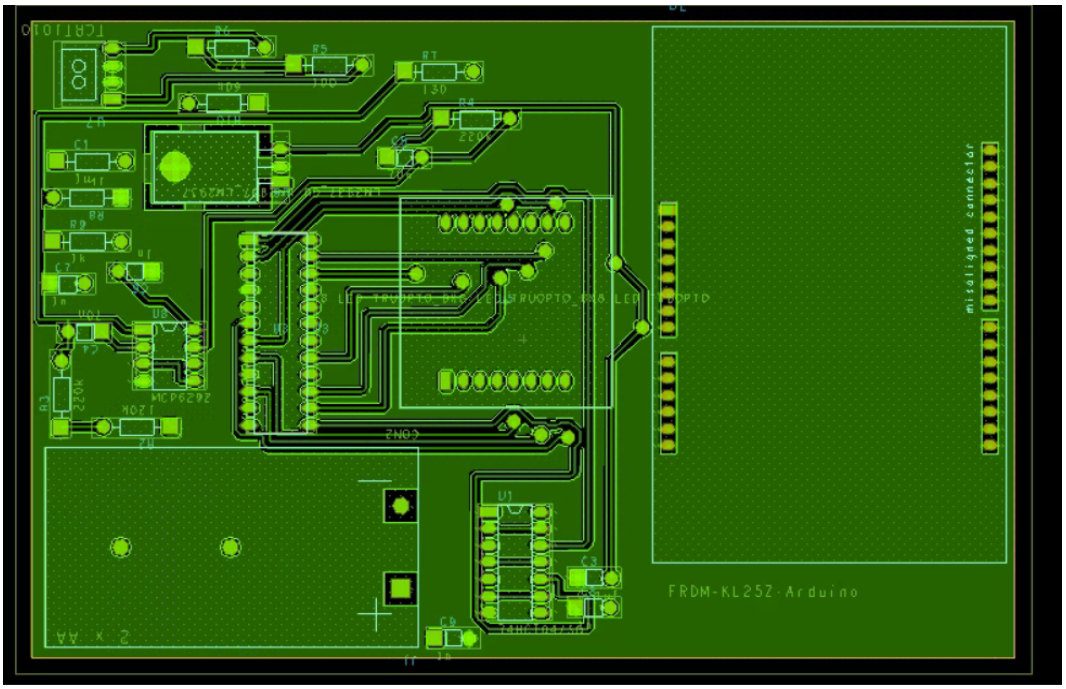

Figure 10: The top mask of our PCB

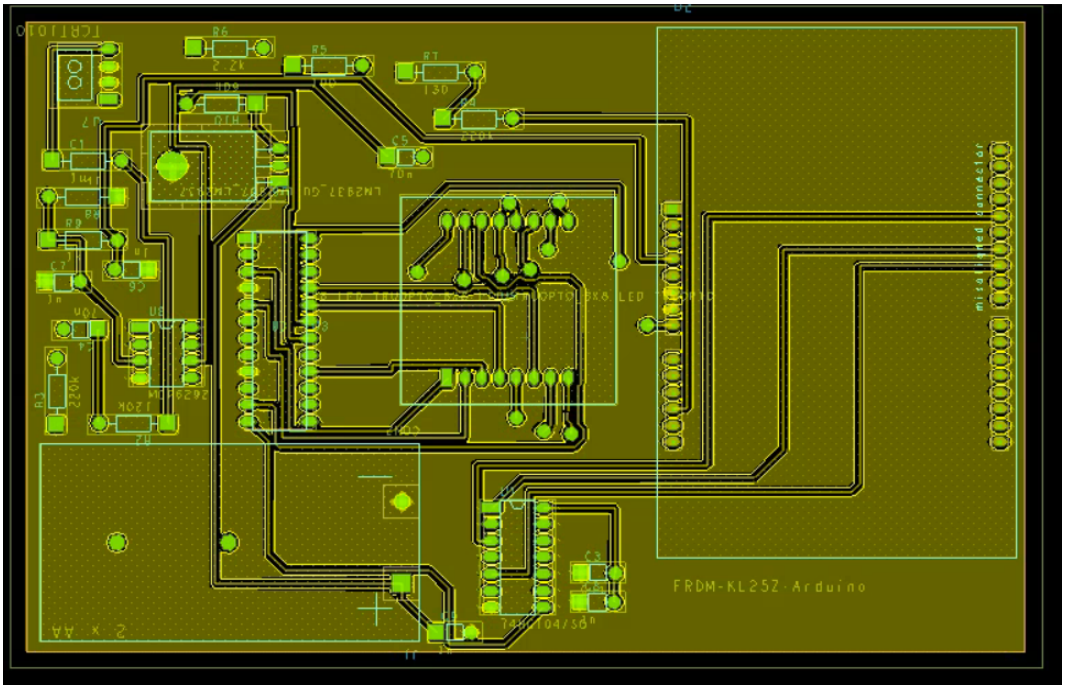

Figure 11: The bottom side of our PCB

5.2 Routing

When routing the PCB, we aimed to minimise the number of vias in an attempt to lower the complexity of soldering/debugging the PCB later. We aimed to place components in chains that best fit the logical process they were performing, keeping components that are close together in the schematic, close together on the final PCB.

A challenge when routing the PCB was that the micro-controller & display could only be connected through the bottom layer of the PCB. In the end, we were able to keep our via count to just 10.

Later, when testing the PCB, we discovered that the decoupling capacitor for the 5V power line had been placed too far away to be effective. We solved this by re-soldering the capacitor directly to the 5V line and ground plane, closer to the voltage regulator[4]. This was likely causing an element of the noise we observed when testing our filter[1] circuit.

5.3 Soldering & Testing

A few days after submitting our final PCB design, we were provided with a freshly printed board. The initial soldering process took roughly 2 days, following this however, the board was non-functional, and would require roughly 50 hours in the lab to get to a working state.

During the testing process described in the Filtration & Amplification section, our board was damaged/modified numerous times. This resulted in a few of the pads being lifted completely from the board, traces being scratched, and unexpected shorts to the ground plane. We solved the scratched traces by replacing the connections with jumper wires, and spent many hours under the microscope tracking down shorts. Andrew helped a great deal to repair the 5V line using a bent resistor leg.

6 Results

In the end, we were able to submit a working heart-rate monitor, which was able to display a somewhat accurate value out to the display. Our monitor was able to detect whether a finger was inserted or not, and display a zero value accordingly. We tested our heart rate monitor by comparing its results with those of a Google Pixel Watch on the same wrist. While doing this, we had our subject (Stefan) perform various physical activities to increase his heart rate, measure it, and compare it to the known value from the smart watch. As seen here:

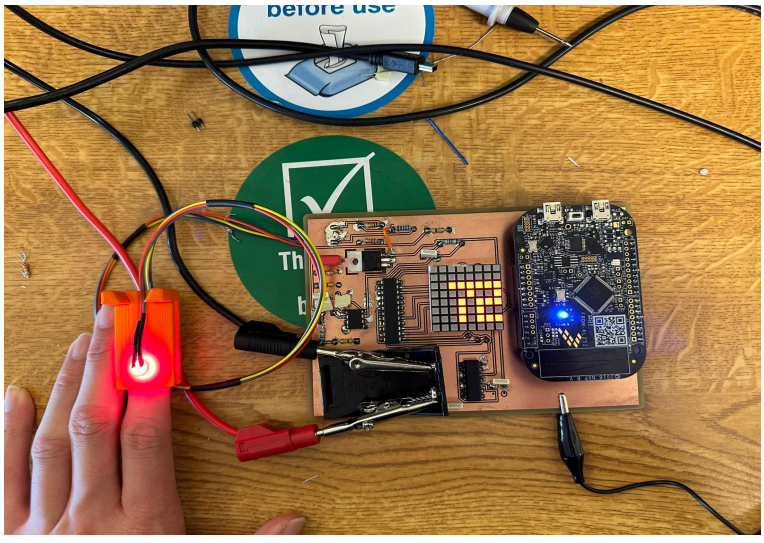

Figure 12: The monitor displaying a valid (corroborated by a pixel watch) heart rate for the subject.

Before debugging the PCB, we implemented a first-draft heart-rate calculation algorithm, however because of how long it took to produce a valid, reliable input signal, we were unable to test it until the last moment.

Our algorithm had no issues with detecting heart rates accurately between 60-100BPM, however the value would begin to jump around once the 100BPM threshold was crossed. We found it would supply an accurate reading, and then jump to a wildly low reading. In order to combat this, we raised the aggressiveness of our lag filter, trying to counteract the spurious low readings that were throwing the calculations.

In the end, we ran out of time to completely solve this issue, however the lag filter worked quite well to solve this issue in the moment. On hindsight, this issue was likely being caused by slight variations in our input signal, and a slightly wobbly method for finding the maximum turning point of each pulse. We hypothesise that small amounts of noise at a sample rate of 66Hz could cause a turning point to be missed, thus doubling, and sometimes tripling the peak-to-peak period of the pulse for that reading. This noise would have a larger effect as the pulse-rate increased, as there is a shorter period of rising/falling samples per pulse.

We found that our circuit drew 100mA - 150mA at peak load, easily within the bounds of a 9V battery.

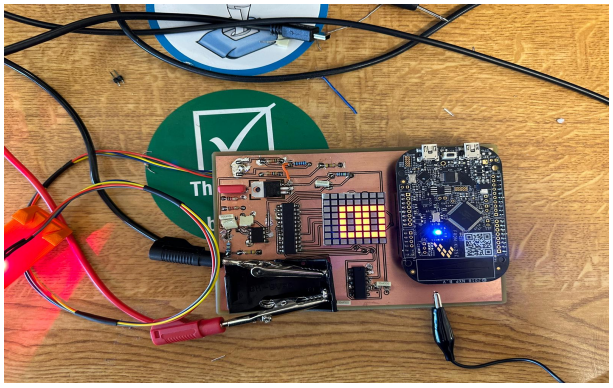

Figure 13: A photo of the monitor correctly recognizing that there is no finger.

7 Conclusions

Throughout the course of this project, many lessons were learnt, often through arduous trial and error.

We learnt the importance of getting your PCB design correct, and double checking values before using them, many of our issues with filtration could have been easily avoided had we performed a brief sanity check on the filter values before soldering them.

We learnt how to use the appropriate tooling to design our circuit and translate that into a PCB design, as well as how to use Platform-IO to write software for the micro-controller through CLion.

We learnt through firsthand experience how debugging without careful rigor can mean you do more damage than good, and leave the board further from functioning than when you started.

We learnt the importance of trusting your instincts over the reliability of components (voltage regulator[4]), as we invested a large amount of time trying to find a root cause for the abnormally high voltages throughout the board, assuming that the regulator was infallible.

We learnt the importance of documenting your work on the board, and of reviewing your peers. All of us made mistakes at some point in this project that could have been avoided by having another set of eyes on the problem.

We learnt the importance of organization, and keeping progress steady throughout the project, as we found that debugging took far longer than expected, and delayed us greatly.

Finally, we learnt the importance of testing your test equipment, as we made many errors when probing our board, that led us to make invalid conclusions about its' current state, that only served to muddy the waters.

Overall this project was inspiring, and a great opportunity to get real hands-on experience working with electronics towards a defined goal. It was enjoyable working together, and we would wholeheartedly do it again if given the choice.

If given an opportunity to retry, we would hope to implement some extra features, such as a trace-display mode, with a physical button to swap between, as well as increase the rigor and reliability of our heart rate calculation algorithm.

8 Supporting Documents & Information

8.1 Circuit Diagram

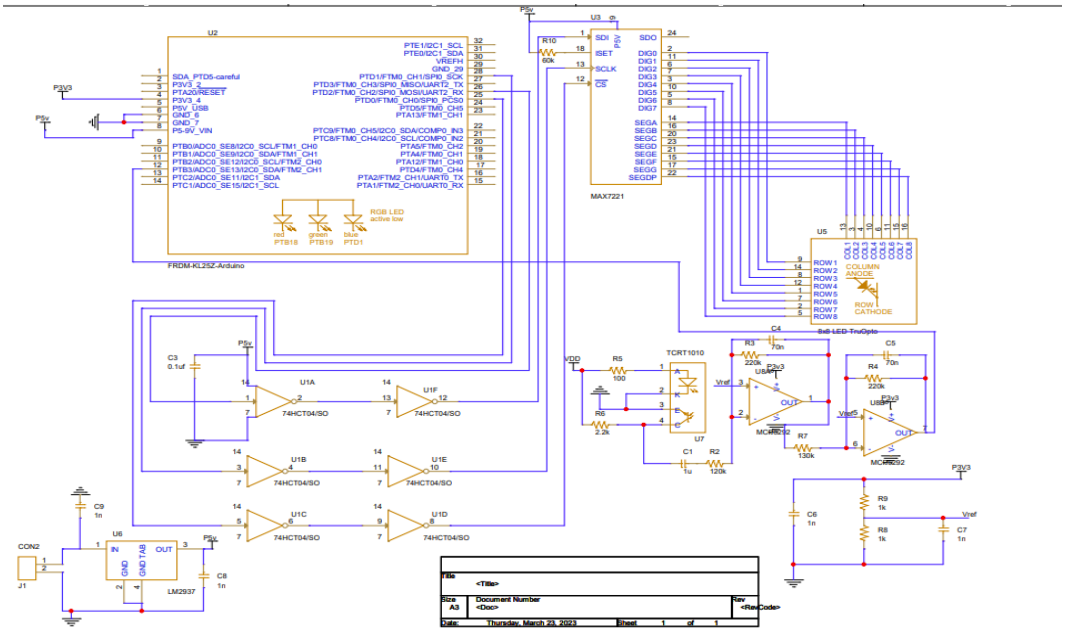

Figure 14: Our Circuit Schematic (see file for higher resolution))