Electronics A2 Project

# Android-based Bluetooth Oscilloscope

Bioniscope

*Author:*
Harry Rickards

2014

# Contents


**1 Problem Analysis and Solution Design** **3**
- 1.1 Problem Definition . . . 3
- 1.2 Research into Existing Oscilloscopes . . . 5
- 1.3 Practical Investigations . . . 6
  - 1.3.1 Oscilloscope Reliability Investigation . . . 6
  - 1.3.2 Bluetooth Distance Test . . . 12
- 1.4 Numerical Parameters . . . 14
- 1.5 Communication with Android Device . . . 16

**2 System Development** **20**
- 2.1 Subsystems . . . 20
  - 2.1.1 Analogue Inputs . . . 22
  - 2.1.2 Power Supply . . . 22
  - 2.1.3 Amplifiers . . . 24
  - 2.1.4 ADCs . . . 31
  - 2.1.5 Microcontroller . . . 32
  - 2.1.6 Bluetooth Transceiver . . . 37
  - 2.1.7 User Interface . . . 39
- 2.2 Circuit . . . 43
- 2.3 Construction . . . 46
  - 2.3.1 PCB Layout . . . 46
  - 2.3.2 PCB Production . . . 51
  - 2.3.3 Obtaining Parts . . . 52
- 2.4 Enclosure . . . 52
- 2.5 Final System . . . 53

**3 Testing** **56**
- 3.1 Test Procedure . . . 56
  - 3.1.1 Overall System . . . 56
  - 3.1.2 Initial Criteria . . . 58
- 3.2 Initial Assessment of System . . . 58

**4 Evaluation** **61**
- 4.1 Report Production . . . 61

**A Fourier Series of a Square Wave** **62**

**B PIC Code** **64**

# List of Figures


1.1 Existing oscilloscopes . . . 3
1.2 Logic Analyser Example Output . . . 4
1.3 Equivalent Time Sampling Diagram . . . 6
1.4 CRO Test Setup . . . 8
1.5 CRO test waveform . . . 10
1.6 Square Wave Harmonics . . . 11
1.7 Bluetooth Experiment Setup . . . 13

2.1 Oscilloscope Subsystems . . . 21
2.2 Power supply circuit diagram . . . 23
2.3 1st Amplifier Stage . . . 25
2.4 2nd Amplifier Stage . . . 26
2.5 Amplification Circuit Simulation . . . 27
2.6 Amplification Circuit Waveform . . . 28
2.7 Amplification Circuit Frequency Spectrum . . . 29
2.8 Amplification Circuit Bode Plot . . . 30
2.9 Amplifier circuit diagram . . . 31
2.10 ADC Circuit Diagram . . . 32
2.11 ADC Power Connections . . . 33
2.12 Microcontroller Circuit Diagram . . . 34
2.13 Analogue waveform shown in Android Bioniscope app . . . 40
2.14 Digital timing diagram shown in Android Bioniscope app . . . 41
2.15 Analogue frequency spectrum shown in Android Bioniscope app . . . 42
2.16 PCB (traces) . . . 49
2.17 PCB (3D render) . . . 50
2.18 PCB (Mockup) . . . 51
2.19 The final enclosure, with the circuit inside . . . 53
2.20 The final circuit . . . 54
2.21 The final circuit in action . . . 55

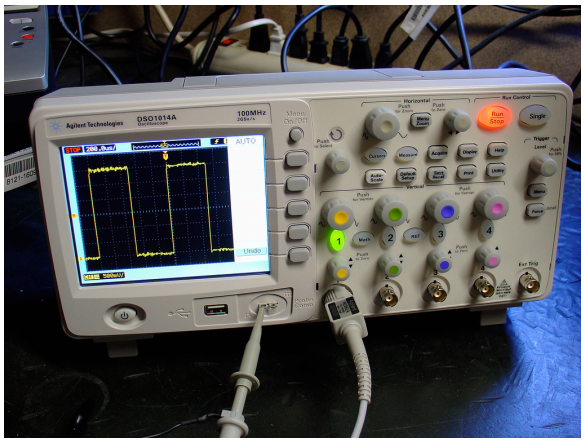
3.1 Frequency spectrum of square wave, as expected . . . 57
3.2 The bandwidth-constrained square wave after passing through the analogue circuitry . . . 59
3.3 Circuit diagram for the op amp circuit added to fix the issue with negative signals . . . 59
3.4 Breadboard photo of the op amp circuit added to fix the issue with negative signals . . . 60

A.1 Square wave . . . 62

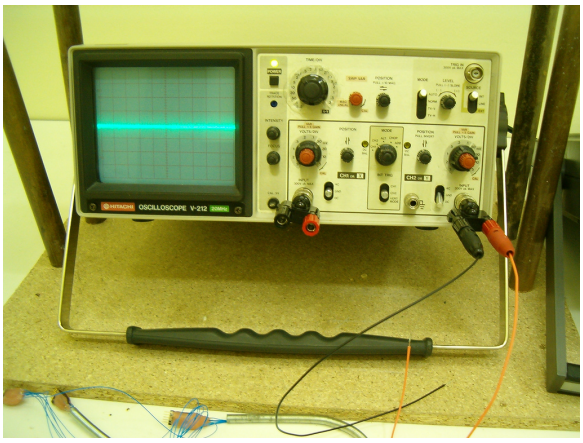
# Problem Analysis and Solution Design

## 1.1 Problem Definition

### CROs and DSOs




(a) Cathode Ray Oscilloscope




(b) Digital Storage Oscilloscope

Figure 1.1: Existing oscilloscopes[1]

Oscilloscopes available in secondary schools or sixth forms are generally basic analogue oscilloscopes known as CROs, or Cathode Ray Oscilloscopes (named so because the display is a Cathode Ray Tube).[2]

The alternative is a DSO, or Digital Storage Oscilloscope. These work in a very different way to CROs. Like in a CRO, the input signal is first passed through some analogue circuitry to amplify it, offset it and so on. However, it's then sampled into a digital signal using a DAC (Digital to Analogue Converter) and processed using a microcontroller (perhaps using the help of an FPGA[3]or a DSP[4]), before being output onto some sort of digital screen.[5]

[2]Phillips, *Using an Oscilloscope.*

[3]**Field Programmable Gate Array** a device that can be programmed to ~become a very large logic circuit. This makes it much faster than a microcontroller at doing a specific job.

[4]**Digital Signal Processor** a custom microcontroller especially designed to perform various *signal processing* functions (e.g., performing a Discrete Fourier Transform to obtain the frequency spectrum of a signal, or using Principal Component Analysis to seperate two audio signals)

[5]TiePie Engineering, *Digital Data Acquisition.*

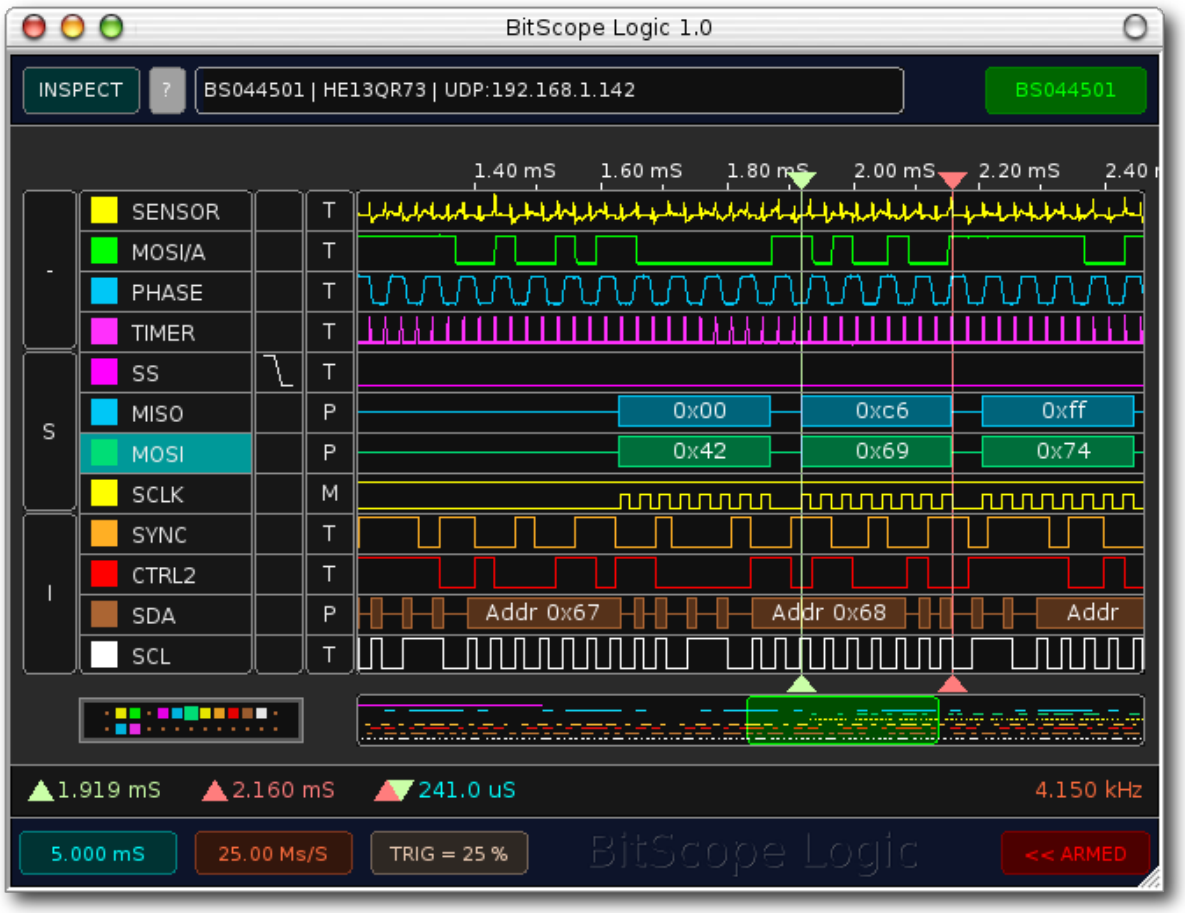



Figure 1.2: Example output from a Logic Analyser[10]

## Advantages of DSOs

DSOs offer a number of advantages over CROs. As well as relatively basic differences such as the traces being much more sharply defined, DSOs allow digital signal processing to be done on the input signal. This means that, for example, DFTs (Discrete Fourier Transforms)[6] can be run to produce a frequency spectrum of the input signal, allowing the user to see both time-domain and frequency-domain representations of the signal.[7]

Additionally, a DSO can be used to create a basic Logic Analyser. This is essentially the digital equivalent of an oscilloscope — a relatively large number of digital wires and buses are connected to the Logic Analyser and a timing diagram is output[8].[9]

[6]While DFTs are the most famous transformations, mainly due to the prevalence of the FFT (Fast Fourier Transform) algorithm, in practice DCTs (Discrete Cosine Transforms) are more commonly used as they can be calculated more quickly whilst achieving a similar result

[7]This also means that DSOs can be used as basic spectrum analysers: essentially devices that plot frequency spectrums, these have uses as far ranging as determining GSM interference when planning mobile phone networks

[8]In more advanced, standalone Logic Analysers, more complicated outputs such as decoded Ethernet packets can be output

[9]Woodward, *What's the Difference Betwen a Mixed-Signal Oscilloscope And A Logic Analyzer?*

### Project Definition

This project[11] aims to create a Digital Sampling Oscilloscope, with the following more advanced features included:

- Frequency spectrums of the input signals available
- Basic capability as a Logic Analyser

From now on, *oscilloscope* will be used to refer to a Digital Sampling Oscilloscope and CRO will be used to explicitly refer to a Cathode Ray Oscilloscope.

## 1.2 Research into Existing Oscilloscopes

Using two articles from Tong[12] and Gabotronics,[13] digital oscilloscopes can be categorised by the following criteria:

- **Bandwidth** The bandwidth[14] in Hz of the initial analogue stages of the oscilloscope. Common values range from 100 kHz to 100 MHz.
- **Type of Sampling** There are two common types of sampling, real-time and equivalent-time sampling. With real-time sampling the scope samples sequentially over the wave, whereas with equivalent-time sampling the scope captures a sample from a different part of the wave every time period. This is perhaps illustrated more clearly in fig. 1.3.

  While equivalent-time sampling offers a much faster sampling rate, it is much more complicated to implement.
- **Sample Rate** The rate (in Hz or the equivalent *samples per second*) at which the oscilloscope can take individual digital samples from the signal. Common values range from 10 kHz to 1 GHz.
- **Resolution** The number of bits each sample has. Common values are 8, 12 and 16 bits. This determines the precision of the oscilloscope, and hence the minimum noise.
- **Memory Depth** The number of samples the oscilloscope can store at one time. Measured in either words, or bits (for example, if the resolution is 8 bits then a depth of 128 words is equivalent to 1024 bits). Intuitively, this can be thought of as the horizontal resolution. Common values range from 100 samples to 10 M samples.
- **Triggers Available** The different options to trigger the oscilloscope to start capturing samples, with the most common one being edge triggering[15] (when the signal passes a threshold on a rising edge[16]). There are also more advanced triggers such

[11] As seen in various places in this document, the project was nicknamed *Bioniscope* — a portmanteau of bionic (from Android) and oscilloscope

[12] Tong, *What to Look for When Choosing an Oscilloscope*.

[13] Gabotronics, *Digital oscilloscopes for hobbyists*.

[14] Most commonly, bandwidth is taken to be the range of fraquencies where the signal has an amplitude gain of $-3\,\text{dB} \approx 71\%$

[15] Picotech, *Advanced Triggering with PicoScope*.

[16] A falling edge could also be used, but the type of edge is part of the trigger specifications and stays constant

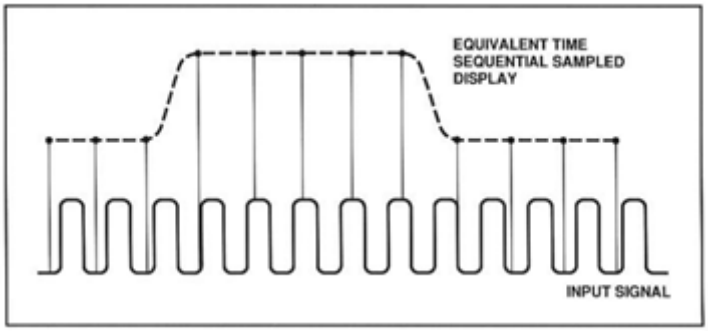



Figure 1.3: Diagram illustrating equivalent time sampling[19]

as pulse width triggering, which triggers the scope when a digital pulse of a certain width is detected. Some expensive oscilloscopes can provide extremely complicated triggers: e.g., decoding Ethernet, TCP/IP[17] and HTTP[18] packets from a digial input, and eventually triggering the analogue channels when a certain webpage is requested.

- **Input range** The minimum and maximum range of input voltages that can be detected by the oscilloscope. A typical value would be from ±50 mV to ±50 V. In many cases, this will be a relatively small range, but an active scope probe (a test probe with an amplifier inside it) will be used to broaden this range.

## 1.3 Practical Investigations

### 1.3.1 Oscilloscope Reliability Investigation

An existing oscilloscope (in this case, a CRO) will be a vital part of testing the osilloscope, so it would be wise to test it's accuracy first.

An FPGA[20]was programmed to produce a square wave at a number of different frequencies, and the wave was looked at on an oscilloscope. Due to their completely different nature compared to microcontrollers, FPGAs are usually programmed in a language called Verilog. While the details are beyond the scope of this project[21], let us just say that Verilog is a way of describing the propogation of signals based on dependencies on both time and other signals.[22] See listing 1 for a very simple Verilog example.

```
always @(posedge) clk begin
  q <= d;
end
```

Listing 1: A simple flip-flop implemented in Verilog

[17]The protocol used for internet networking

[18]The protocol used to transfer web pages over the internet

[20]**Field Programmable Gate Array** a device that can be programmed to ~become a very large logic circuit. This makes it much faster than a microcontroller at doing a specific job.

[21]Tala (*ASIC World*), found online at `http://www.asic-world.com/verilog/index.html`, is an excellent resource for learning Verilog

[22]Wikipedia, *Verilog — Wikipedia, The Free Encyclopedia.*

The FPGA was programmed to perform 50% duty-cycle clock divisions of the FPGA's internal[23] 50 MHz clock, meaning the output frequency could be easily calculated using

$$f = \frac{50\,\text{MHz}}{2^n} \tag{1.1}$$

where $n$ is the clock divisor. The accuracy of the crystal oscillator is so high (an average error value would be around 0.0005%[24]) compared to the accuracy of reading values from a small oscilloscope screen that we can ignore it in this instance.

The Verilog code used can be seen in listing 2.

```
// Output a square wave signal of various frequencies to test a CRO
module oscilloscope_test(clk, signal);
  // Accurate 50MHz clock input signal
  input clk;

  // Output square wave
  output wire signal;

  // Module that divides a frequency to produce a 50% duty-cycle output
  // using flip-flops
  clock_divider d1 (
    .clk_in(clk),
    .clk_out(signal)
  );
  // clk is divided by 2^divisor
  // So increase by 1 to halve the output frequency
  defparam d1.divisor = 21;
endmodule
// Divide a clock by 2^divisor with a 50% duty-cycle output (1:1 mark:space)
module clock_divider(clk_in, clk_out);
  // This is intended to be changed by the code calling this module
  parameter divisor = 2;

  // Input to be divided
  input clk_in;

  // Create a counter out of flip flops, and toggle clk_bit whenever the
  // counter's most significant bit toggles. This produces a 50% duty-cycle
  // output that we use to toggle clk_out.
  reg [divisor-1:0] counter = 0;
  always @(posedge clk_in) counter <= counter + 1;
  output reg clk_out = counter[divisor-1];
endmodule
```

Listing 2: Code used to produce the square wave to test the oscilloscope

[23] A high-quality crystal oscillator

[24] Semtech, *Improving the Accuracy of a Crystal Oscillator*.

**Data**

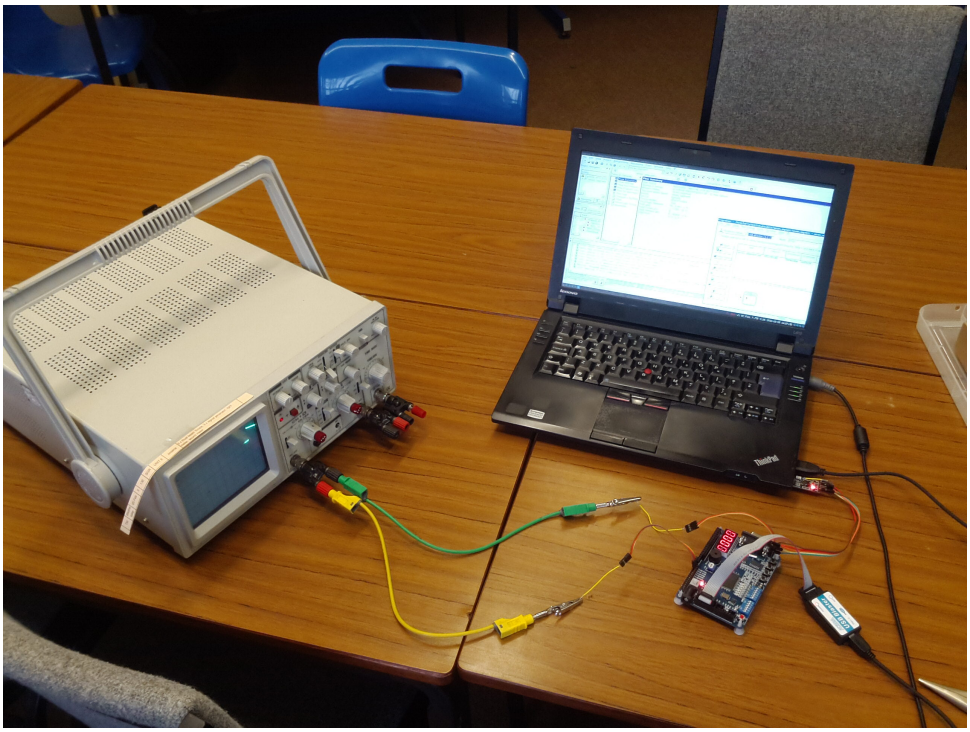


Figure 1.4: Equipment setup to test the CRO

Seen in fig. 1.4, the equipment was setup as described above. Frequency readings were taken over a number of values of $n$ ranging from 4 to 19. For each reading, the horizontal and vertical division settinsg were recorded, as well as the number of divisions counted in one period of the wave. From these, the measured frequency and amplitude were calculated.

These measured values were compared to the actual frequency (as calculated by eq. (1.1)) and amplitude (the digital output from the FPGA ranged from 0 V to it's logic-high level of 3.3 V, leading to an amplitude of 1.65 V) to find error values for the frequency and amplitude.

The numerical results of the experiment can be seen in table 1.1.

With $n = 19$, the frequency was low enough that it was very difficult to take a reading of the number of y divisions per period due to the nature of cathode ray tubes. This can be seen in the results: the frequency error jumps from less than 1% to over 90%. Because of this, $n = 19$ will be considered an anomolous result and disregarded for the rest of this analysis.

Note the high level of accuracy, particularly with regards to frequency. Due to previous bad experiences with this oscilloscope, your author was rather surprised at the results. Furthermore, almost all of the error can actually be put down to errors in the human measurement process. Take $n = 7$ as an example: the number of X divisions was measured as 5.1 to 1 decimal place, giving a percentage error of 0.98%. It is purely coincidental then, that the actual error is less than half that at 0.39%!

| | | | Measured | | | | Actual | | Error (%) | |
|---|---|---|---|---|---|---|---|---|---|---|
| $n$ | Time/div (µs) | Voltage/div (V) | X div | Y div | Frequency (kHz) | Amplitude (V) | Frequency (kHz) | Amplitude (V) | Frequency | Amplitude |
| 4 | 0.1 | 2 | 3.4 | 2.0 | 2940 | 2.0 | 3120 | 1.65 | 5.88 | 21.21 |
| 5 | 0.2 | 2 | 3.1 | 1.8 | 1610 | 1.8 | 1560 | 1.65 | 3.23 | 9.09 |
| 6 | 0.5 | 2 | 2.5 | 1.8 | 800 | 1.8 | 781 | 1.65 | 2.40 | 9.09 |
| 7 | 0.5 | 2 | 5.1 | 1.7 | 392 | 1.7 | 391 | 1.65 | 0.39 | 3.03 |
| 8 | 1 | 2 | 5.1 | 1.7 | 196 | 1.7 | 195 | 1.65 | 0.39 | 3.03 |
| 9 | 2 | 2 | 5.1 | 1.7 | 98.0 | 1.7 | 97.7 | 1.65 | 0.39 | 3.03 |
| 10 | 5 | 2 | 4.1 | 1.7 | 48.8 | 1.7 | 48.8 | 1.65 | 0.10 | 3.03 |
| 12 | 20 | 2 | 4.1 | 1.7 | 12.2 | 1.7 | 12.2 | 1.65 | 0.10 | 3.03 |
| 14 | 100 | 2 | 3.2 | 1.7 | 3.12 | 1.7 | 3.05 | 1.65 | 2.40 | 3.03 |
| 16 | 500 | 2 | 2.6 | 1.8 | 0.769 | 1.8 | 0.763 | 1.65 | 0.82 | 9.09 |
| 19 | 2000 | 2 | 2.7 | 1.7 | 0.185 | 1.7 | 0.0954 | 1.65 | 94.18 | 3.03 |

Table 1.1: Results from oscilloscope accuracy experiment

## Qualitative Results

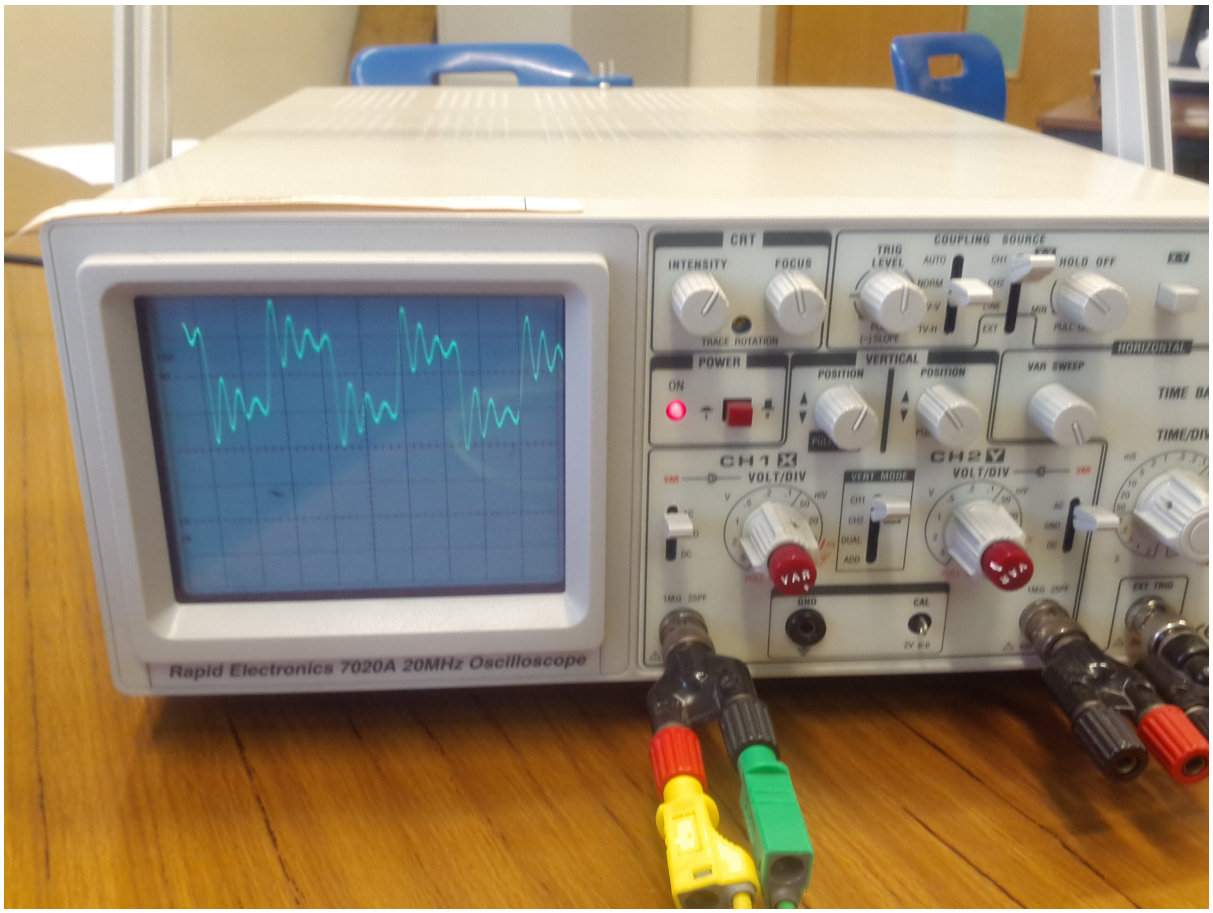


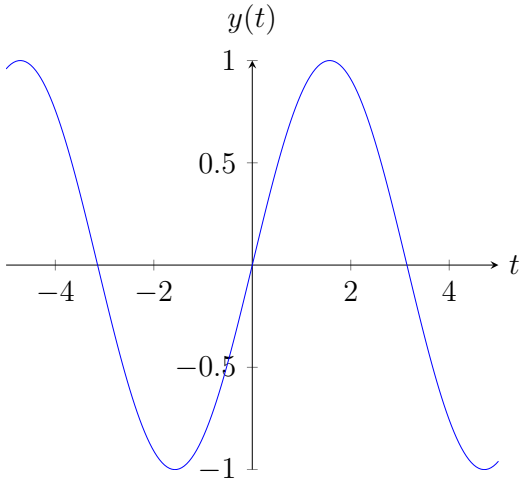

Figure 1.5: Not-so-square wave shown on the oscilloscope

In qualitative terms, however, the oscilloscope was not nearly as accurate. Looking at fig. 1.5, one can see that at high frequencies the analogue bandwidth of the oscilloscope comes into play, causing the digital input to look very much like a summation of sine waves. In fact, this was still visible to a lesser extent until $n = 12$ (corresponding to a frequency of 12.2 kHz). This means that while the oscilloscope can be used to accurately determine the frequency of high-frequency signals, it can't be used to determine the shape of a signal with any accuracy.

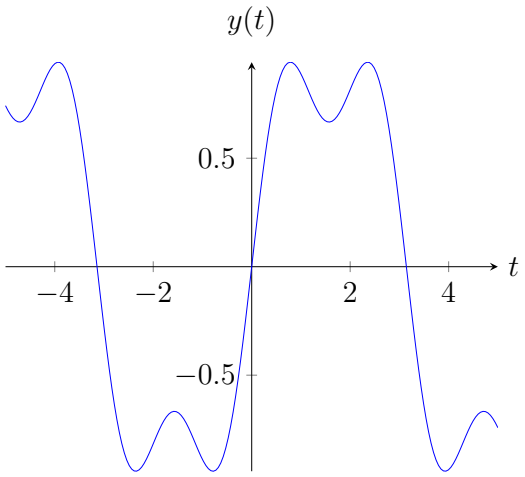
Using the maths detailed in appendix A we can plot graphs of square wave approximations up to a certain number of harmonics, as seen in fig. 1.6. Comparing these to fig. 1.5, we find that the closest match[25] is the seventh harmonic graph. This means we can approximate the analogue bandwidth of the oscilloscope as 7 times the frequency of the signal, or $7 \times 3.12\,\text{MHz} = 21.8\,\text{MHz}$[26].

[25]The match is not exact because harmonics above the seventh harmonic are present, they just have a reduced amplitude. However by looking at the number of peaks in one period of a wave we find the seventh harmonic is the highest one with a major component present.

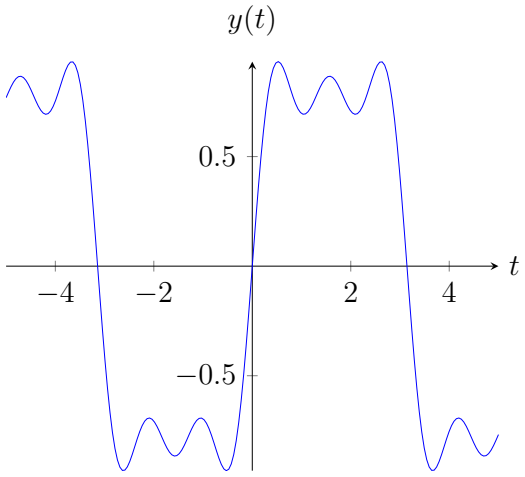
[26]Checking the datasheet of the oscilloscope, we find the quoted bandwidth is 20 MHz meaning we are accurate to within a third of a harmonic, as close as we could hope using this basic method.



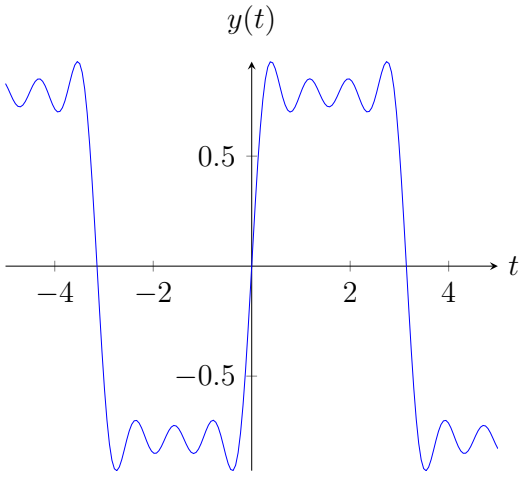

(a) First Harmonic



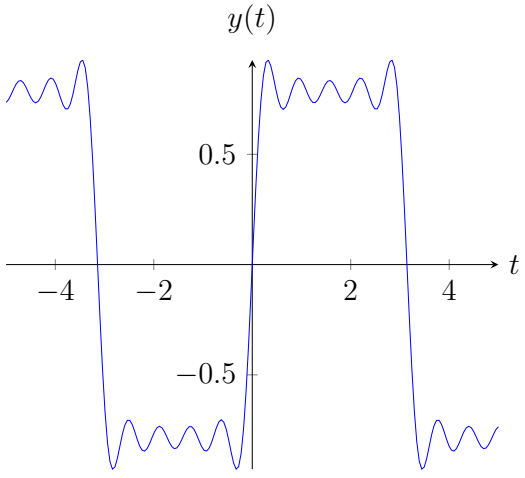

(b) Third Harmonic



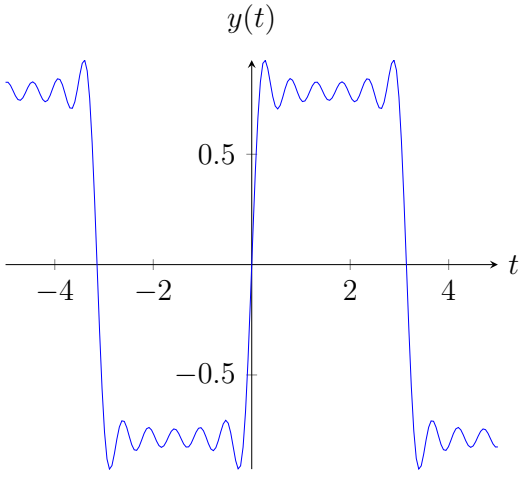

(c) Fifth Harmonic




(d) Seventh Harmonic




(e) Ninth Harmonic




(f) Eleventh Harmonic

Figure 1.6: Approximations to a square wave up to certain harmonics

### 1.3.2 Bluetooth Distance Test

As discussed later in section 1.5, Bluetooth will be used to communicate between the microcontroller and Android device. Because of this, a practical investigation was undertaken to test the reliability of the Bluetooth link at different distances.

The Bluetooth module to eventually be used in the final product, an HC06, was connected to an Arduino (an ATMega microcontroller development board allowing quick prototyping) and programmed to echo any data sent to it (so if the character $A$ was received by the HC06, it would immediately transmit back the character $A$). This code can be seen in listing 3.

A python[27] program was then written on a laptop computer (with an inbuilt Bluetooth module) to communicate with the HC06. 384[28] bytes were sent out, and the received data compared to the sent data to produce an accuracy figure. This code can be seen in listing 4[29].

This program was run at with the laptop a number of different distances from the HC06. The results are shown in table 1.2. See fig. 1.7 for a photo of the setup.

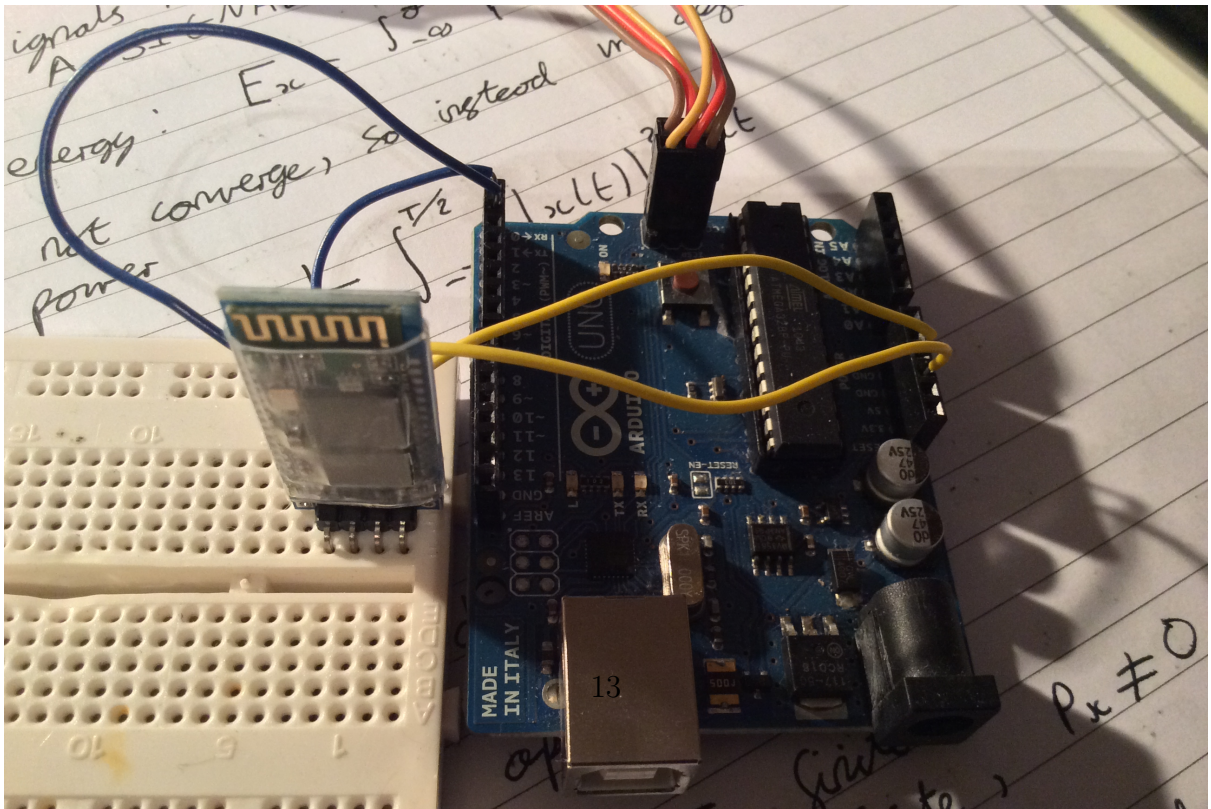
[27] A very high-level computer programming language that's suited for tasks such as this, with libraries for things such as serial communication. For more information see various online resources.

[28] 3 times 128, and a convenient compromise between speed and accuracy

[29] For details on how the Bluetooth receiver in the laptop was setup as a serial device, see the online page for this project at `http://github.com/hrickards/bioniscope`.

```
// IMPORTANT
// Delays need to be added into setup() and loop() if the Arduino is to be
// programmed over USB. If the code is left as it is, the Arduino will not
// be able to communicate with the IDE over USB, so a dedicated AVR programmer
// (or another Arduino) will be needed to upload new sketches.

// Run when the Arduino starts up initially
void setup() {
  // Begin serial communication with the HC06 at a baud rate of 9600
  Serial.begin(9600);

  // Use pin 13 (connected to an LED) to show visually when data is being
  // written
  pinMode(13, OUTPUT);
}

// Loops continuously
void loop() {
  // Turn LED off
  digitalWrite(13, LOW);

  // Wait til serial available
  while (!Serial.available());

  // Turn LED on
  digitalWrite(13, HIGH);

  // Echo back the received serial byte
  Serial.write(Serial.read());
}
```

Listing 3: The Arduino code used to echo back any characters received via Bluetooth



| Distance (m) | Reliability (%) |
|---|---|
| 0.0 | 100.00 |
| 1.0 | 100.00 |
| 2.0 | 100.00 |
| 3.0 | 100.00 |
| 4.0 | 100.00 |
| 5.0 | 100.00 |
| 5.5 | 6.51 |
| 6.0 | 0.00 |

Table 1.2: Results from bluetooth distance experiment

At distances up to and including 5 m, the link was 100% reliable. At 5.5 m that figure dropped dramatically (suggesting that the connection was lost partway through the test), and at further distances the link dropped out completely. This means that the oscilloscope should be able to be used at distances of up to 5 m away from the device.

## 1.4 Numerical Parameters

### Frequency and Sampling Rate

The sampling rate should be as high as feasibly possible to allow the oscilloscope a wide variety of uses. By Shannon ("A Mathematical Theory of Communication"), the maximum frequency[30] that we can sample is half the sampling rate. In reality, because of the basic real-time sampling method being used it must be at the very least 25 times less than the sampling rate (allowing just over 12 samples for each half-period of the wave).

ADCs in a DIP[31] package (suitable for breadboards) are not readily available beyond 2 MHz. Furthermore, the only 2 MHz DIP ADC, the AD7822, is renowned for being particularly difficult to interface with a microcontroller. Instead, an ADC with a maximum sampling rate of 1 MHz will be used (many DIP ICs can provide this sampling rate).

However, at such a high speed getting the full sampling rate out of an ADC is not usually possible with a microcontroller (because, for example, a control bit might have to go high then low again for 20 ns, but a microcontroller with a 16 MHz clock can only do this in an absolute minimum of 62.5 ns). So the oscilloscope sampling rate will be specified as at least 250 kHz.

[30] Actually, that's the maximum sinusoidal frequency that we can sample. By Fourier, all other waveforms can be represented as a sum of sinusoids, and in reality sampling at 25 times the frequency of the waveform means that we'll be able to get a high enough amount of detail for most non-sinusoidal waveforms

[31] Electronics components come in a variety of different packages of many different shapes and sizes. Generally, those that fit in a breadboard are part of the DIP (dual in-line package) family: e.g., CDIP (Ceramic DIP) and PDIP (Plastic DIP). Other packages, such as TSOP, are usually much smaller and suitable only for soldering onto PCBs. While a PCB will be used for the final oscilloscope, soldering surface-mount chips (a generic term for all packages smaller than DIP, such as TSOP) requires specialist equipment unavailable to your author.

This means the maximum frequency that should be sampled is $\frac{250\,\text{kHz}}{25} = 10\,\text{kHz}$.

## Bandwidth

The maximum frequency into the analogue amplification stage will be 10 kHz, but we also need to take into consideration harmonics (remember, by Fourier any non-sinusoidal signal can be represented as a sum of sinusoidal signals of increasing frequency). The effect of this can be seen all too clearly in fig. 1.5 To be safe, we will specify a minimum bandwidth of 1 MHz.

## Resolution

Inaccuracies in the oscilloscope output are more likely to be caused by the analogue circuitry (in particular, the digital potentiometers are only accurate to 0.25%) than a low resolution, so an 8 bit resolution will suffice (this offers an accuracy of $\approx 0.4\%$, comparable to the digital potentiometer accuracy). This also means ADCs will be more readily available, as 8 bit DIP ADCs are significantly more common than higher resolution ones.

## Memory Depth

The memory depth needs to be large enough to have a horizontally precise signal, but small enough that the samples can easily be stored in the microcontroller. For these reasons, 1024 words (equivalent to 8192 bits or 1 kB with an 8 bit resolution) will be chosen. Most microcontrollers have multiple kB of RAM, so multiple signals can easily be stored in the RAM.

## Input Range

For the minimum input range, we'll choose the standard ±50 mV, however high-bandwidth rail-to-rail op amps are only cheaply available up to ±5 V so for the maximum input range we'll choose ±5 V[32].

## Triggers Available

For the purposes of this project, no triggering will be used. The oscilloscope will capture enough samples that the intended use case will be capturing one set of samples and then analysing those results, rather than capturing a small set of samples and continuously refreshing that set like a traditional CRO. If needed, triggering could potentially be implemented in software at a later date.

[32]While the input to the op amp could be a higher voltage than the supply voltage, a digital potentiometer will be needed to automatically adjust the gain of the op amp. These only work up to to the supply voltage.

## 1.5 Communication with Android Device

As detailed in section 2.1.7, an Android device will be used as the frontend to the oscilloscope. This means that the microcontroller must somehow communicate with the device.

There are 4 feasible ways that this could be achieved: via WiFi, Bluetooth and USB.

### USB

This would perhaps be the easiest way to implement the communication. Small discrete devices, such as the CP2102,[33] convert between the relatively complicated USB protocol and the simple UART[34] protocol. In fact, nearly all microcontrollers now come with a built-in USART interface, so the microcontroller side of things would be literally 'plug and play'[35].

On the Android side, the microcontroller would appear as a serial communication device. The 'usb-serial-for-android' library[36] would make it trivial to communicate with the microcontroller from an Android application via a serial port[37].

Because the connection is wired, there would be minimal interference so the speed could potentially go as high as the maximum USB 2.0 bit rate[38] of $480\,\mathrm{Mbit\,s^{-1}}$.

However, there would be a major disadvantage to this approach: that the oscilloscope would have to be physically connected to the tablet. This means the tablet would not be able to be freely moved around while using the oscilloscope. While a traditional oscilloscope operates this way, allowing wireless connection would make the oscilloscope much more vertasile: for example, the relatively cheap oscilloscope hardware could be left permanently connected in a hard-to-access electronics project, and accessed wirelessly when needed through the Android tablet.

[33] Silicon Labs, *Single-chip USB to UART Bridge*.

[34] A UART (Universal Asynchronous Receiver/Transmitter) is actually a piece of hardware that takes in parallel data (usually a byte made up of 8 bits) and transmits it serially according to a certain serial protocol, while also receiving data serially and outputting it in parallel. While pedantically it's not correct, we'll use *UART protocol* to refer to this protocol as is commonly done. There is also a device called a USART, or Universal Synchronous/Asynchronous Receiver/Transmitter. As the name suggests, these support sending data in both a synchronous and asynchronous manner, and are very common inside microcontrollers. However, in this document we'll follow the electronics industry in taking UART and USART to mean the same thing: transmitting data asynchronously. As there is no synchronous data transmission required in the oscilloscope, this will not be an issue.

[35] Various configuration words would have to be moved to registers to initialise USART communication at the start of the program, but beyond that there would simply be registers for writing data, reading data and checking if data is available to be read

[36] mik3y, *usb-serial-for-android*.

[37] Serial ports are a throwback to the times when computers had physical RS232 serial connections, over which data could be sent and received using a very simple protocol. Nowadays, serial ports are virtual ports that send and receive data over much more complicated protocols such as USB and Bluetooth.

[38] Baud rate figures are not officially available, but it's clear from the bit rate that it would be extremely high

## WiFi

It would be far beyond the scope of this project to communicate just using a 2.4 GHz antenna, so instead a discrete WiFi module would be needed. These have traditionally been relatively expensive[39], however Texas Instruments has recently released the CC3000, a module much cheaper than the competition.[40]

If the initial project was being created on a custom PCB, then the CC3000 could simply be included as a component on the PCB. However, the initial prototype is being created on a breadboard, so a breakout board (a board that contains the CC3000 and an antenna and exposes a digital interface to the CC3000 via pins that can fit in a breadboard) is needed.

The cheapest, most widely available breakout board is Adafruit's 'Adafruit CC3000 WiFi Breakout with Onboard Ceramic Antenna'. Adafruit are a US-based company, but the board can be easily obtained in the UK via eBay. However, it costs £25.

In terms of complexity, using the CC3000 would be more complicated than using USB, but not insurmountably so. Libraries exist for both AVR and PIC microcontrollers to interface with the CC3000, and on the Android side it would simply require interfacing with a TCP socket[41] (not significantly more complex than communicating via a USB serial connection).

One advantage WiFi has over USB is that it offers a wireless connection, negating the main disadvantage of USB. In addition, it would be able to offer very fast data transfer rates like USB (also like USB, official baud rate figures are not available, but the latest standard[42] offers a bit rate of up to 600 $\mathrm{Mbit\,s^{-1}}$).

However, the main disadvantage is the price. Even with the CC3000, the cheapest available option, at £25 the WiFi module would make up a considerable amount of the total cost of the oscilloscope.

## Bluetooth

Like with WiFi, discrete Bluetooth modules are readily available. These operate in a similar way to the USB CP2102 — the microcontroller communicates with the module via USART, and the Android device via a serial port.

One example of such a module would be a JY-MCU HC-06. Via eBay, these are available relatively cheaply (about £5).

This means one key advantage of Bluetooth is simplicity. The inbuilt USART functionality of the microcontroller can be used on the microcontroller side, and a Bluetooth serial

[39] So much so that for a time in the mid-2000s it become very common for devices such as amateur robots to communicate with computers over the 2.4 GHz spectrum, but using a different protocol to WiFi. This author has not been able to find out why such devices were significantly cheaper than WiFi devices, but one suspects it is due to both the complexity of the WiFi protocol and licensing costs required to implement it.

[40] Benchoff, *Finally, TI is producing simple, cheap WiFi modules.*

[41] Again, TCP sockets are significantly outside the scope of this project, but they can essentially be thought of as virtual counterparts to physical sockets. A wire can be used to connect physical sockets together to allow communication between them, and a protocol such as TCP (the base protocol for the internet) does the same thing for virtual sockets.

[42] 802.11n, which is supported by most wifi devices nowadays

library used on the Android side. The heavy lifting of the Bluetooth communication is done by the HC-06 and software built into the Android device.

Bluetooth obviously offers a wirelesss connection, and should work at distances ofup to about 10 m. The maximum baud rate supported is 1.3824 Mbit $s^{-1}$, which is slower than both USB and WiFi but fast enough for our purposes (remember the oscilloscope takes 8-bit samples at 250 kHz so the sampling rate is only 4 Mbit $s^{-1}$).

## Chosen Solution

The required wired connection between the oscilloscope and Android device ruled out USB, leaving WiFi and Bluetooth.

Data transfer speeds are not a big issue in this use case, and Bluetooth's 1.3824 Mbit $s^{-1}$ is certainly fast enough. This means the key differences between Bluetooth and WiFi are complexity and cost. Bluetooth wins both of these, so Bluetooth was chosen as the communication method between microcontroller and Android device.

```
import serial # pySerial used for serial communication with BT receiver

# Number of bytes of data to send each time
ITERATIONS = 384

# Open a serial socket to the BT device
# If this line fails, check /dev/rfcomm0 is the right port
ser = serial.Serial('/dev/rfcomm0', 9600, timeout=1)

# Remove any initial noise by writing a newline and waiting for it to return
ser.write("\n")
ser.readline()

# Write results to data.txt
f = open("data.txt", "wb")

j = 0 # Number of times the test has been run

# Keep running the indented part until the program is stopped
while True:
    # To be filled with booleans that are true iff the data is echoed correctly
    data = []
    # The character to send and have echoed back. Will be changed by the code
    # below.
    char = 0x00

    # Do the indented part ITERATIONS number of times
    for i in range(ITERATIONS):
        # Increment char, wrapping back to 0x00 after 0xFF
        char = (char + 1) % 0xFF
        # Write the character
        ser.write(chr(char))
        # Append to data whether or not the character was echoed correctly
        data.append(ser.read() == chr(char))

    # The line of output to show to the user and write to data.txt
    # test number: reliability average
    output = "%d: %f" % (j, sum(data)*100.0/len(data))
    print(output)
    f.write(output + "\n")

    j += 1
    raw_input("...") # Wait for the user to press enter

ser.close()
f.close()
```

Listing 4: The python code used to measure the accuracy of the Bluetooth link at various distances

# System Development

## 2.1 Subsystems

The system will be split up into a number of subsystems, shown in fig. 2.1. Each of these subsystems is described in more detail in the sections below, and a brief description is given immediately below.

- Two analogue input channels come into the system, and are amplified and offset by amplification circuitry (with the gain controlled by a central microcontroller).
- These channels are both sampled using an Analogue to Digital Converter, which feeds into the microcontroller.
- A number of digital inputs are also fed straight into the microcontroller
- Through a Bluetooth transceiver, an Android device interacts with the microcontroller, receiving samples of both the analogue and digital inputs and controlling things such as the sampling rate and amplifier gain.

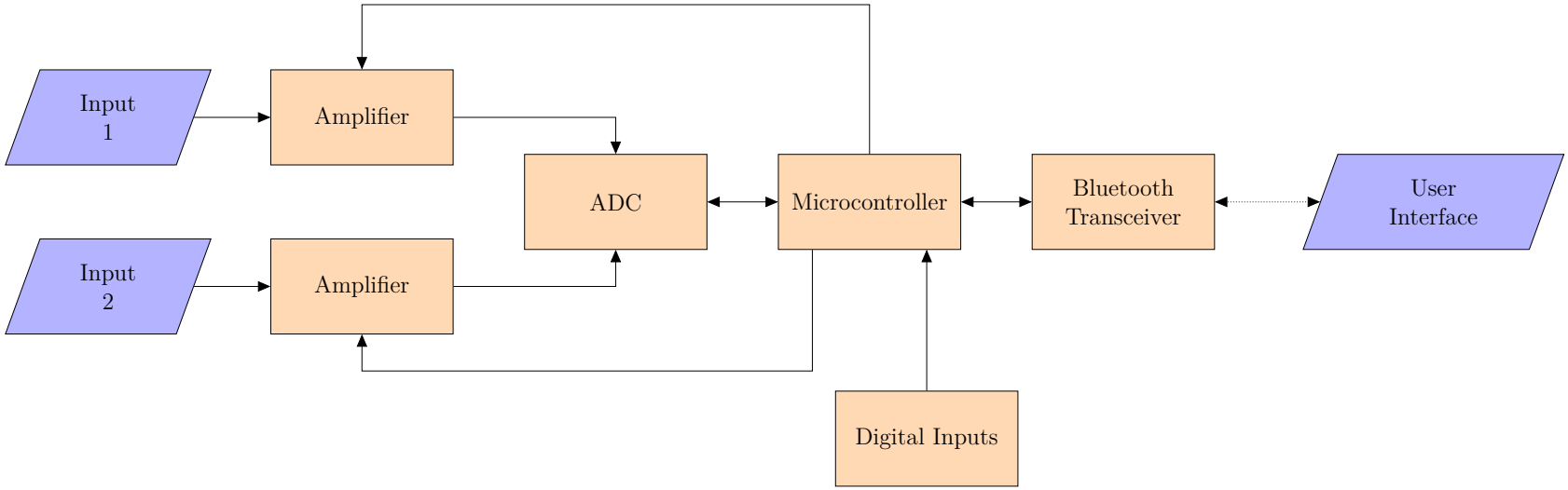



Figure 2.1: Subsystems within the oscilloscope

### 2.1.1 Analogue Inputs

For the purposes of this project wires are sufficient, however in a real-life scenario high-impedance, low-inductance test probes (usually coaxial cable) should be used.

### 2.1.2 Power Supply

#### Requirements

While it is not explicitly shown in fig. 2.1, the amplifiers, ADC, microcontroller and Bluetooth transceiver will all need to be powered.

A 5 V rail will be needed that can source the following current[1]:

- 5 mA for the two amplifiers combined
- 5 mA for the two digital potentiometers combined
- 15 mA for the ADC
- 25 mA for the microcontroller
- 30 mA for the LED
- 30 mA for the Bluetooth module

which gives a total of 110 mA.

Similarly, a −5 V rail will be needed for the first-stage amplifier that can sink at least 5 mA, and a ground rail will be needed that can sink at least 110 mA[2].

#### Circuit

While researching this part of the circuit, your author came upon a chip known as an *inverting charge pump*, a device that takes $V_s$ and 0 V as inputs and outputs $-V_s$ and 0 V as outputs[3] — essentially providing a low-current[4] dual power supply. This was implemented into the oscilloscope's power supply as it means only a positive voltage and ground need to be input, meaning it could potentially be powered by battery.

Many of the chips (primarily the ADC and microcontroller) require very precise and stable power rails, and to that effect a voltage regulator will be used. Commonly known as a 7805 (although that specific device will not be used in this case), a voltage regulator takes a relatively high voltage (e.g., 9 V) as input and outputs a very precise lower voltage (e.g., 5 V) that does not fluctuate even when the output current changes rapidly. In addition to this, decoupling capacitors will be added to the amplifier, ADC and microcontroller to further reduce voltage fluctuations (the capacitor values will be taken directly from the relevant datasheets).

[1] The figures are overestimations meaning there will definitely be enough current available.

[2] There will always be a positive load, so we do not need to worry about the ground rail being able to source the 5 mA that will flow to the −5 V rail

[3] The way it does this is complicated, but revolves around charging up a capacitor in one direction and then discharging it in the opposite direction.

[4] The highest commonly-available current rating seems to be about 1 A

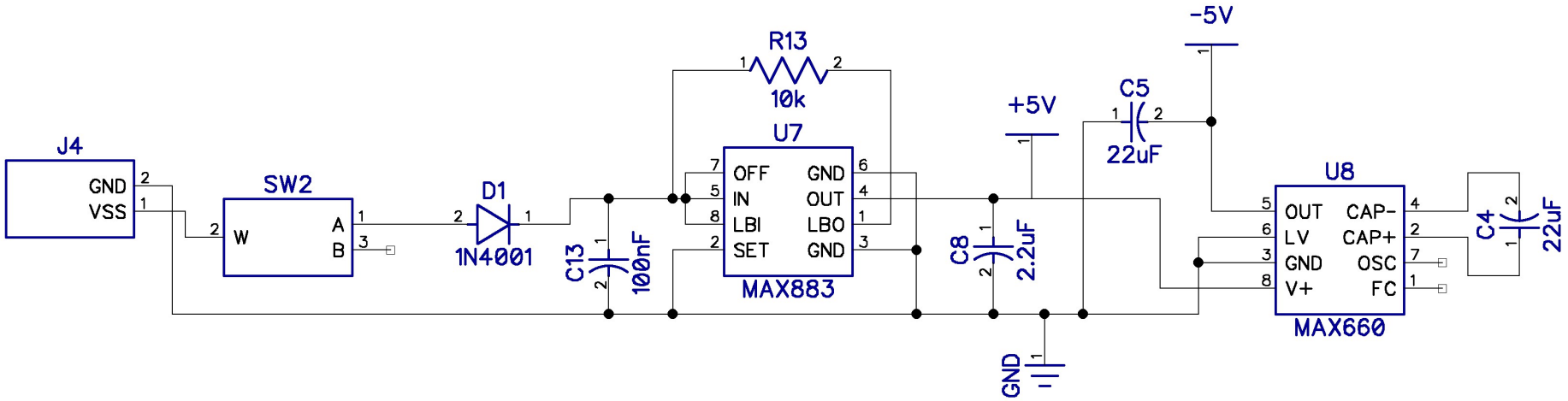



Figure 2.2: Power supply circuit diagram

These two ideas, then, give rise to the circuit diagram shown in fig. 2.2. A single-rail power supply (perhaps from a 9 V battery) is used to power the system, and a switch (to let the user switch the oscilloscope off and on) and a forward-biased diode (to limit damage if the power supply is connected backwards) are connected immediately to the power rail.

A MAX883 chip is then used as a fixed 5 V voltage regulator. Most semiconductor companies produce PDIP package 5 V voltage regulators, but Maxim Integrated's chip was chosen due the availability of free samples. As specified in the datasheet, a 100 nF capacitor is placed on the input and a 2.2 µF capacitor on the output.

A MAX660 is then used as an inverting charge pump (this specific chip was chosen for similar reasons to the MAX883). Again, as specified on the datasheet a 22 µF capacitor is placed on the output and another between $CAP-$ and $CAP+$.

### 2.1.3 Amplifiers

Each amplifier will have two purposes:

- To amplify the signal based on the Voltage Control of the oscilloscope. This is needed so the user can input a range of different voltage signals.
- To shift the signal so the minimum voltage is 0 V, rather than some negative voltage. This is required because the ADC will only accept an input greater than 0 V.

The chosen ADCs require an input voltage between 0 V and 5 V[5]. A non-inverting summing amplifier would not be suitable beause the offset added to the signal needs to remain constant while the gain changes, so instead the amplifier will be split into two stages.

The first stage will amplify the signal and offset it to be between 0 V and −5 V, below which the op-amp will saturate.

The second stage will invert this signal with a voltage gain of −1, meaning the output voltage will be between 0 V and 5 V as required.

#### First Stage

**Circuit** This stage will be implemented using a summing amplifier, with the inputs consisting of the input signal and a fixed voltage (for the sake of simplicity, the 5V supply wil be used).

See fig. 2.3 for a circuit diagram.

**Component Calculations** Suppose that the input signal varies from $-V_1$ to $+V_1$, and the amplifier has a gain of $A$ and an offset of $B$. Then the output voltage $V_o$ will be given by

$$-V_o = \frac{R_f}{R_1} V_i + \frac{R_f}{R_2} \cdot 5\,\text{V}$$

[5]The maximum voltage is actually $V_{REF+}$, which will be connected to $V_{CC}$ for simplicity, which in this case is 5 V

The first term will range from $-A \cdot V_1$ to $+A \cdot V_1$, and the second term will be $B$. So $-V_o$ will range from $B - A \cdot V_1$ to $B + A \cdot V_1$. Choosing a value of $B = 2.5\,\text{V}$ means that $-V_o$ will vary equally above and below 2.5 V, as we want[6](because the voltage must be between 0 V and 5 V, and 2.5 V is the midpoint of those two).

$R_f$ and $R_2$ will be fixed to give $\frac{R_f}{R_2} = 0.5$ to get the required offset. $R_1$ will be a digital potentiometer acting as a rheostate (variable resistor), allowing the microcontroller to control the gain of the amplifier.

The input voltage range is from $\pm 50\,\text{mV}$ to $\pm 5\,\text{V}$, so the gain $A = \frac{R_f}{R_1}$ must vary from 0.5 to at least 50. Digital potentiometers are readily available in a number of values, so the standard $10\,\text{k}\Omega$ option will be chosen. This means the potentiometer can vary from the minimum resistance (usually $75\,\Omega$) to $10\,\text{k}\Omega$.

Using the values for $A$ and $R_1$, this gives a value of $R_f = 5\,\text{k}\Omega$, meaning $A$ will range from $\frac{5\,\text{k}\Omega}{10\,\text{k}\Omega} = 0.5$ to $\frac{5\,\text{k}\Omega}{75\,\Omega} \approx 66.7$, as required. In turn, this gives a value of $R_2 = 10\,\text{k}\Omega$.

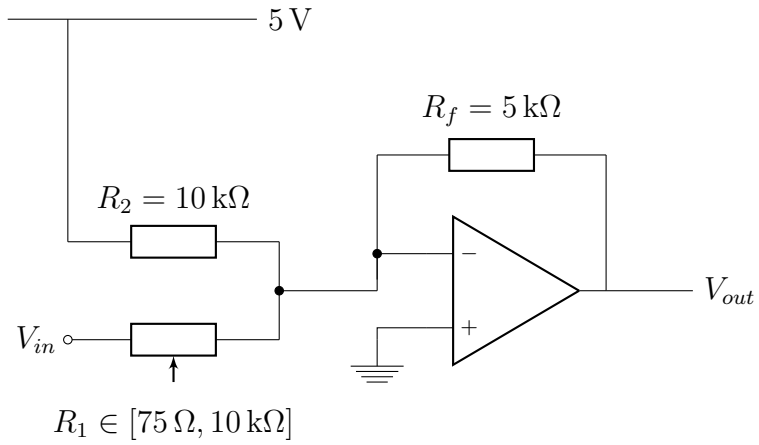



Figure 2.3: First stage amplifier circuit diagram

**Component Selection** The Microchip MCP4131-103E/P will be used for the digital potentiometer. It offers enough precision (typically 0.25%), offers a simple interface for changing the resistance (SPI) and is cheap and readily available in a PDIP package. Additionally, free samples are available from Microchip. To make the component calculations made earlier work, the $10\,\text{k}\Omega$ MCP4131-103 variety will be used.

The op-amps should be of rail-to-rail type (i.e. they can saturate at almost 0V and $V_s$) and be able to cope with $V_s = 5\,\text{V}$, $V_i = 5\,\text{V}$ and input frequencies of up to 1 MHz. The Analog Devices AD8031 meets all of these requirements (at 1 MHz it gives an open-loop gain of almost 25 dB) and is also cheap and readily available in PDIP packaging. Due to the availability of free samples, 2 AD8032ANZs (consisting of two AD8031 op amps in each chip) will be used.

[6]The op amp will saturate before outputting a voltage above 5 V, so we don't need to worry about this. If the gain or input voltage is sufficiently large, $-V_o$ could fall below 0 V. This means the next stage must saturate at 0 V and not output negative voltages, whatever the input.

### Second Stage

This stage simply has to invert the previous stage, saturating at 0 V and 5 V. For this we can use a simple inverting amplifier.

$$V_{out} = \frac{R_f}{R_1} V_{in}$$

Choosing a standard value of $R_f = 10\,\text{k}\Omega$, we obtain $R_1 = 10\,\text{k}\Omega$.

As detailed in the previous section, AD8031 op amps will be used.

See fig. 2.4 for a circuit diagram.

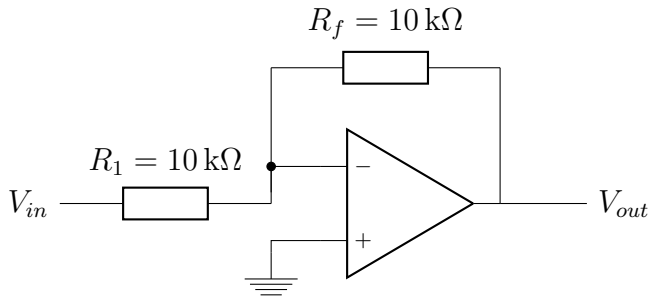



Figure 2.4: Second stage amplifier circuit diagram

### Simulation

To check the frequency response of the amplification circuit, it was simulated using LT-Spice (see fig. 2.5).

To test the simulation, a low frequency sine wave was applied, and as expected the output was a sine wave of amplitude 2.5 V between 0 V and 2.5 V. The resultant waveform can be seen in fig. 2.6. A FFT was then applied to this waveform to obtain the frequency spectrums seen in fig. 2.7. As required, the spectrum of the output was identical to the spectrum of the input, with the slightly lower amplitudes being due to a lower wave amplitude (2.5 V c.f. 5 V).

A frequency sweep was then applied to produce a Bode plot. The details of such a plot are beyond the scope of this project, but let us just note that the gain is very consistent across all frequencies we'll be using (as predicted: the op amp has a stated bandwidth of 80 MHz, much higher than 1 MHz), and while it becomes almost 9° out of phase at very high frequencies, this is not an issue for this project[7].

[7]The phase is only important to the oscilloscope when comparing the phase of two signals. To see two signals on screen at the same time base, their frequencies must be close enough together that they're both out of phase by approximately the same amount. Hence their phase difference is unchanged

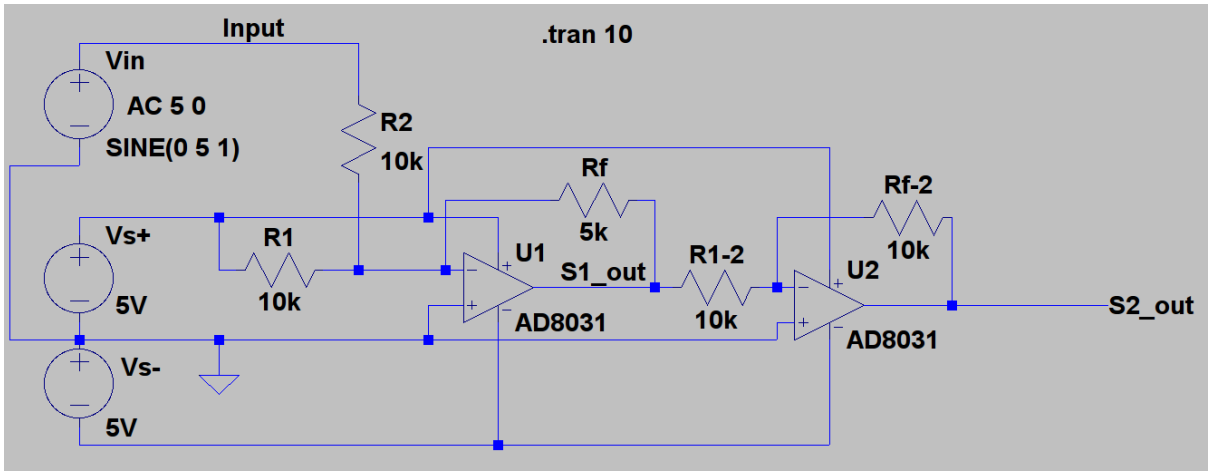



Figure 2.5: Amplification circuit in LTSpice

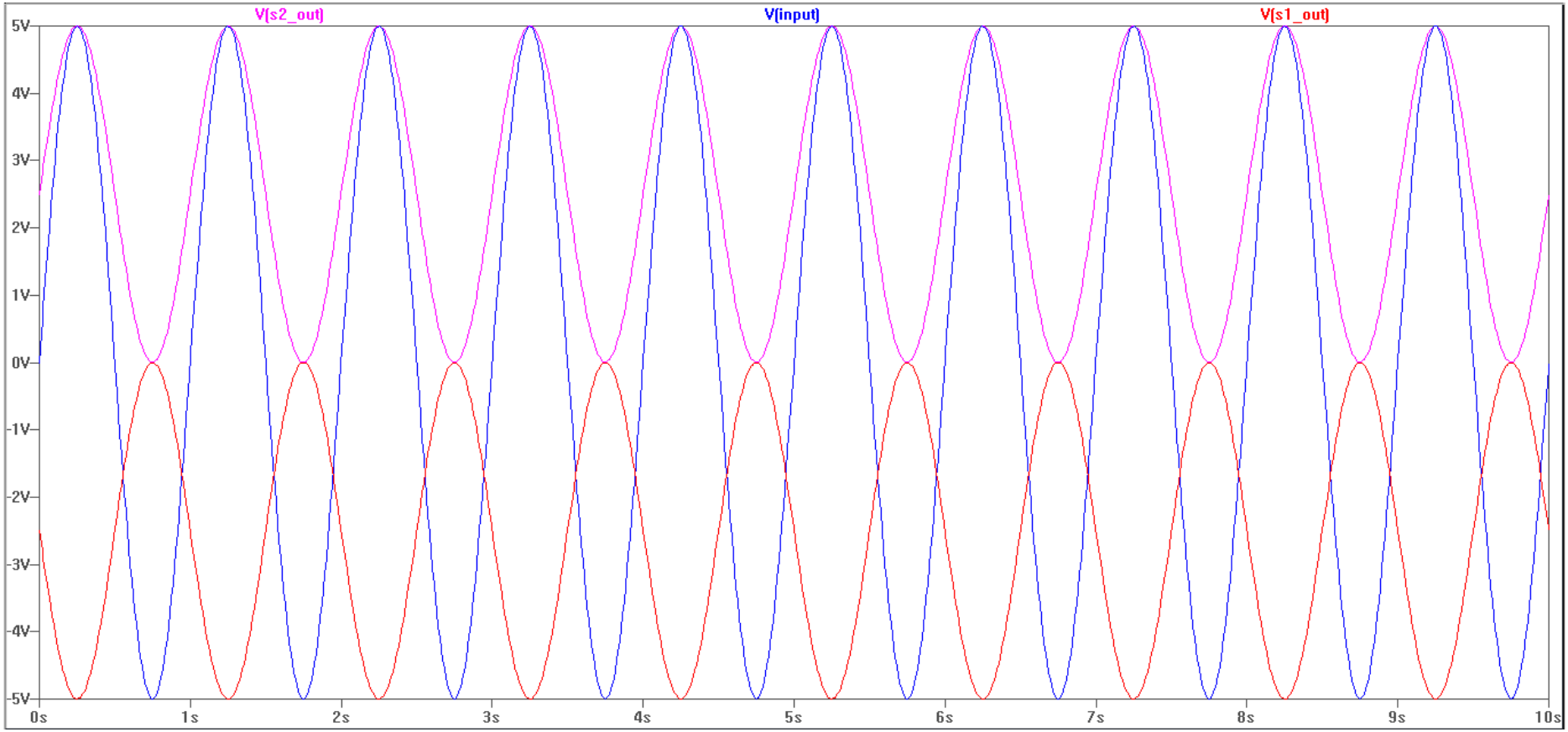



Figure 2.6: Waveform produced by amplification circuit at 1Hz (blue = original, red = first stage output, magenta = second stage output)

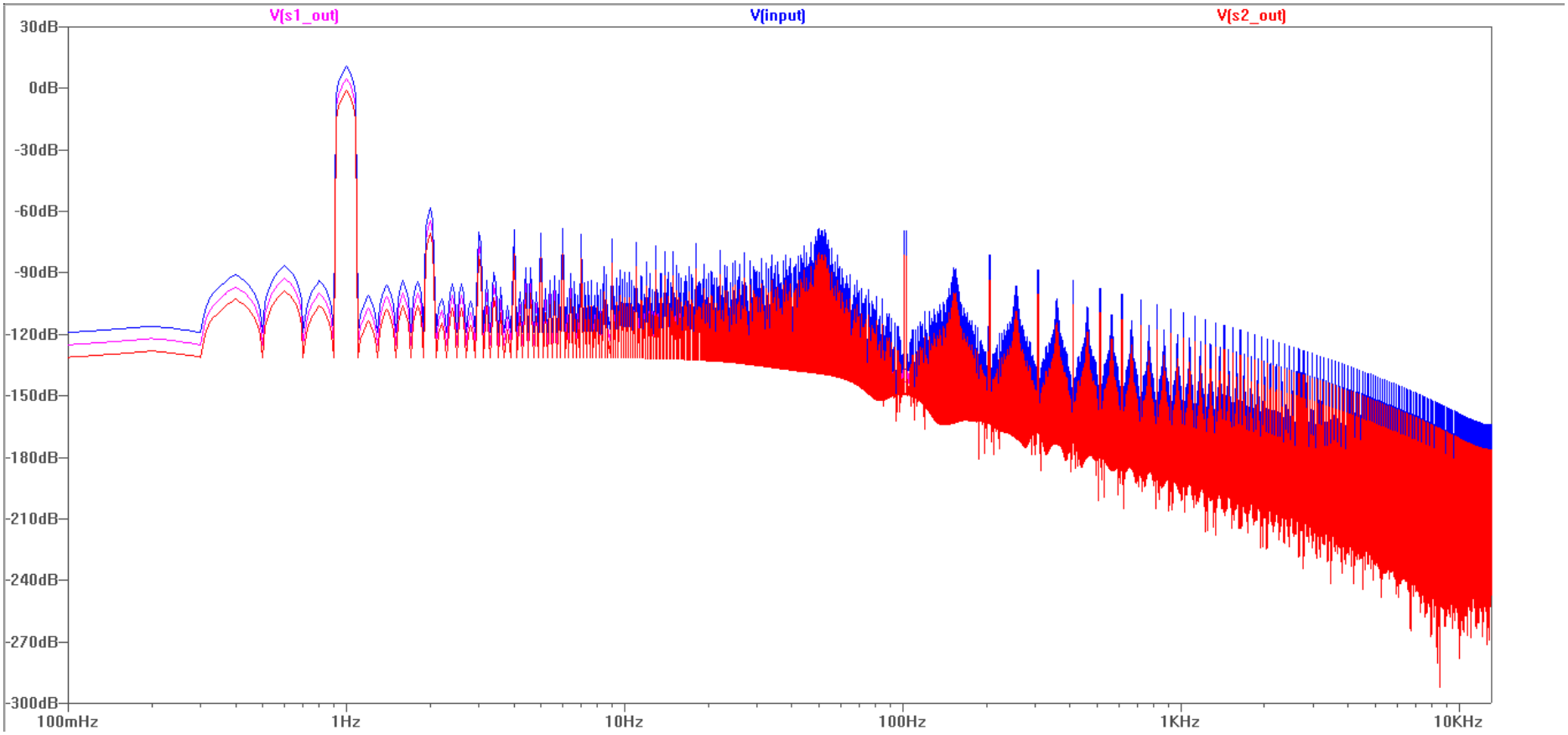



Figure 2.7: FFT of waveform produced by amplification circuit at 1Hz (red = original, blue = first stage output, magenta = second stage output)

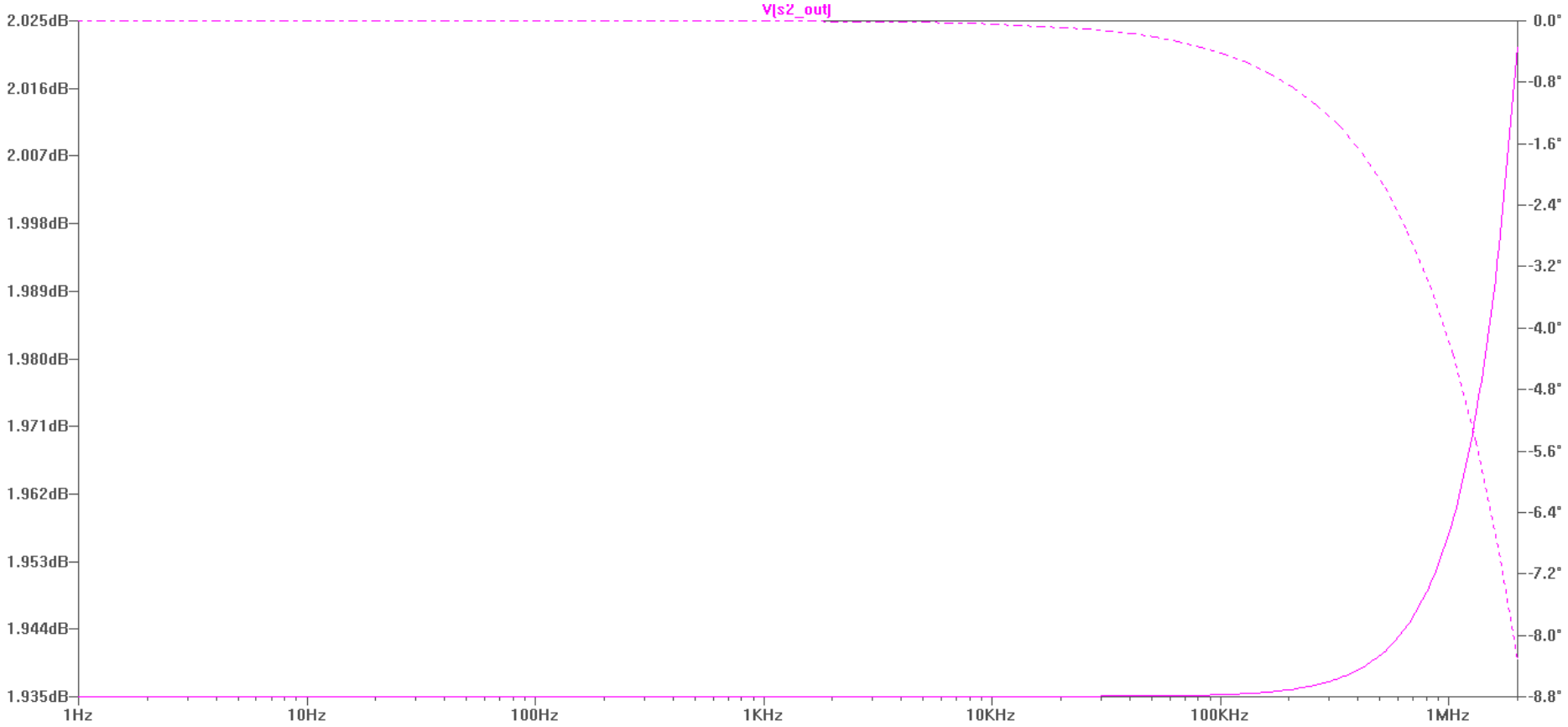



Figure 2.8: Frequency response of amplification circuit

### Overall Circuit

See fig. 2.9 for an overall circuit diagram.

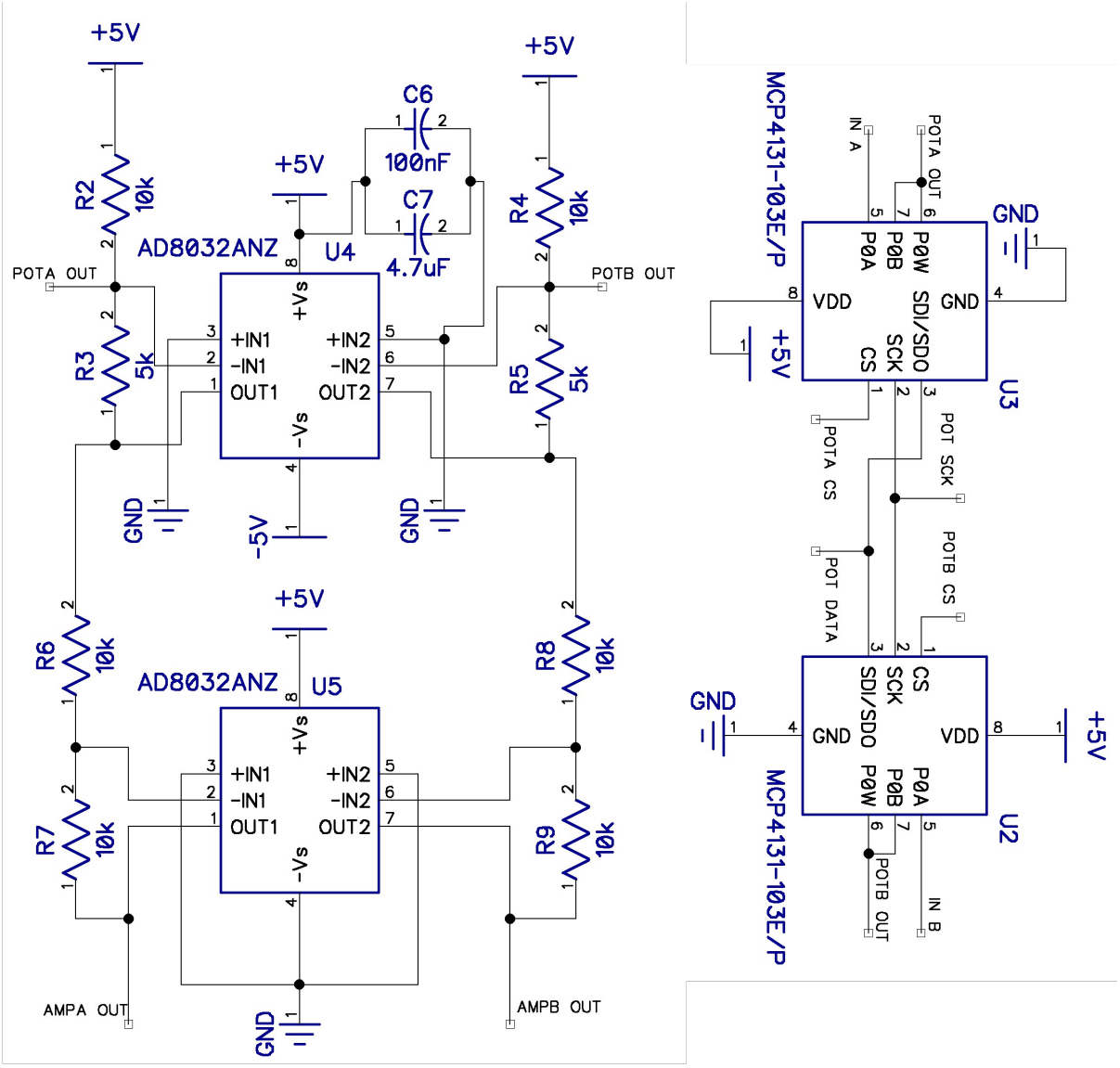



Figure 2.9: Circuit diagram of complete amplifier

## 2.1.4 ADCs

As discussed before, these should be able to sample at up to 1 MHz. The only readily available ADCs that can do this in PDIP packaging and have free samples are the 8-bit half-flash[8] Maxim Integrated MAX114 ADCs.

There are a number of modes the ADCs can be operated in, however the pipelined mode offers a very good compromise between complexity and speed. The full details are in Maxim Integrated (*MAX114/MAX118 Datasheet*), but as the ADCs are operated from a microcontroller in which there's a delay of 62.5 ns (assuming a 16 MHz clock) between each instruction, the operating procedure can be simplified to the following:

[8]Similar to flash ADCs, but requiring $\approx 2^{\frac{n}{2}}$ as opposed to $\approx 2^n$ parts

1. Use $A_0$ and $A_1$ to choose the input signal (there are only two inputs in this project, so $A_0$ and $A_1$ are connected together meaning a logic 0 selects input 0 and a logic 1 selects input 3)
2. Pull $\overline{CS}$, $\overline{RD}$ and $\overline{WR}$
3. Wait approximately 250 ns
4. Pull $\overline{CS}$, $\overline{RD}$ and $\overline{WR}$ back high
5. Read in a sample from $D_0$ through $D_7$

Taken directly from the datasheet, section 2.1.4 shows the connections that must be made to power the MAX114 and provide it with a reference voltage.

Additionally, $\overline{PWRDN}$ must be kept high to keep the ADC powered on, and $MODE$ must be kept high to keep the ADC in Read-Write mode (of which pipeline mode is a subset). Both of these should be pulled high through a resistor (3.3 kΩ being a common value) to limit current. Based on this, section 2.1.4 shows the circuit schematic for the ADC.

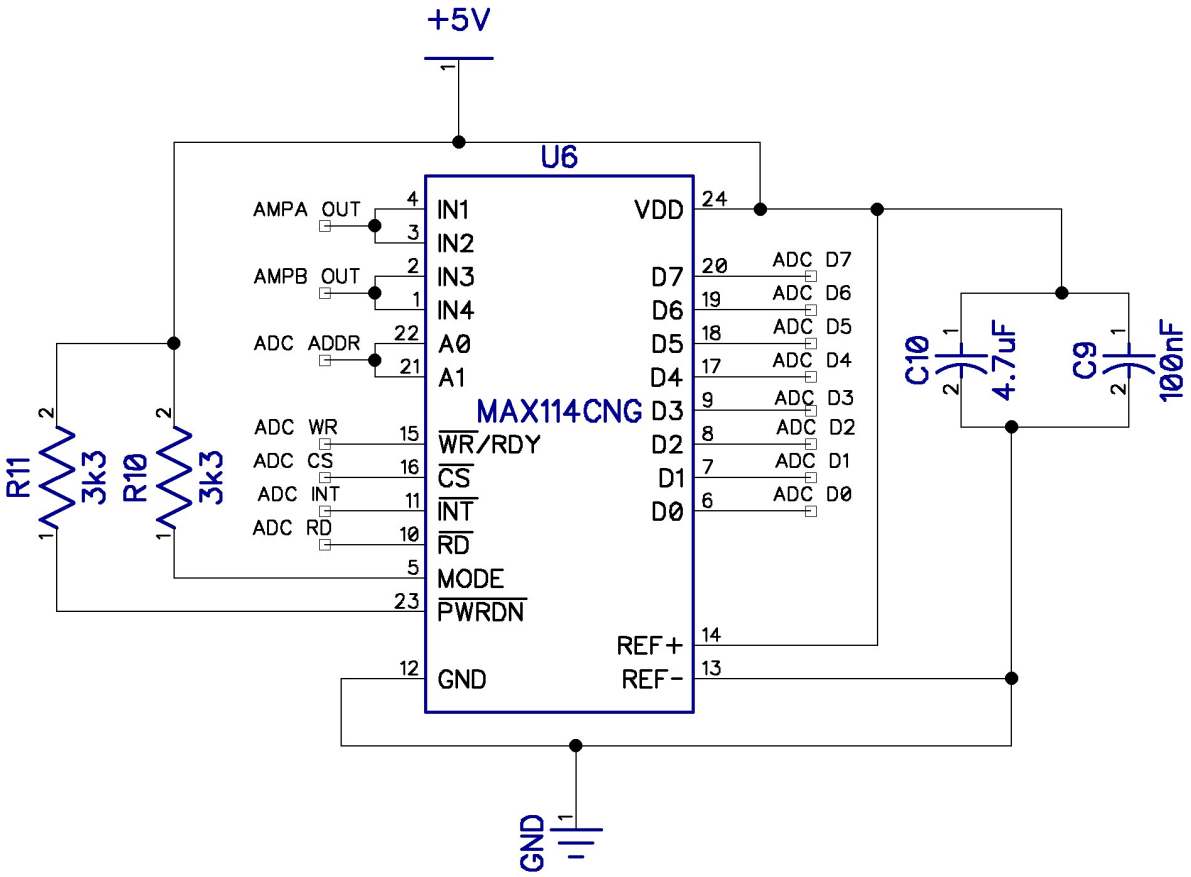



Figure 2.10: Diagram showing how ADC will be connected

### 2.1.5 Microcontroller

Due to this author's significant previous experience with AVR microcontrollers, one will be used as there are no significant disadvantages over a PIC. In addition, it is easier

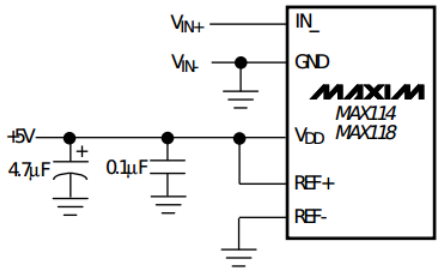



Figure 2.11: Power connections for the MAX114[9]

to obtain free samples from Atmel (the company that produce AVR microcontrollers) than it is from Microchip (the company that make PICs). The main requirements the microcontroller needs to satisfy are:

- Ability to use a 16 MHz clock
- Inbuilt UART communication (to communicate with the Bluetooth module)
- At least 32 IO pins:
  - 8 for data connections to the ADC
  - 5 for control connections to the ADC
  - 8 for digital data inputs
  - 4 ($DATA$, $SCK$ and two $CS$) for digital potentiometer communication
  - 2 ($RX$ and $TX$) for Bluetooth module communication
  - 2 ($RX$ and $TX$) for USB module communication (for debugging purposes[10]).
  - 3 for ISP header (the connections used to program the AVR)
- At least 1 kbyte of SRAM (to store the 8192 bits of samples)
- Availability in a PDIP package

The ATMega1284P was chosen because it's the only 8-bit AVR that meets the above criteria (any AVRs with more than 32 IO pins are not available in PDIP packages)!

### Circuit

For the chip to operate, a 16 MHz crystal oscillator must be connected between $XTAL1$ and $XTAL$, with a 22 pF capacitor from each pin to ground.

The following power connections must also be made:

- Both $GND$s to ground
- $V_{CC}$ and $AV_{CC}$ to 5 V, with a 100 nF and 1 µF capacitor in parallel down to ground (acting as decoupling capacitors)

[10]While in theory the Bluetooth module should just be plug and play, a spare CP2102 USB to USART module will be used to allow the microcontroller to talk to a laptop over USB to help fix any problems that occur in the prototyping process

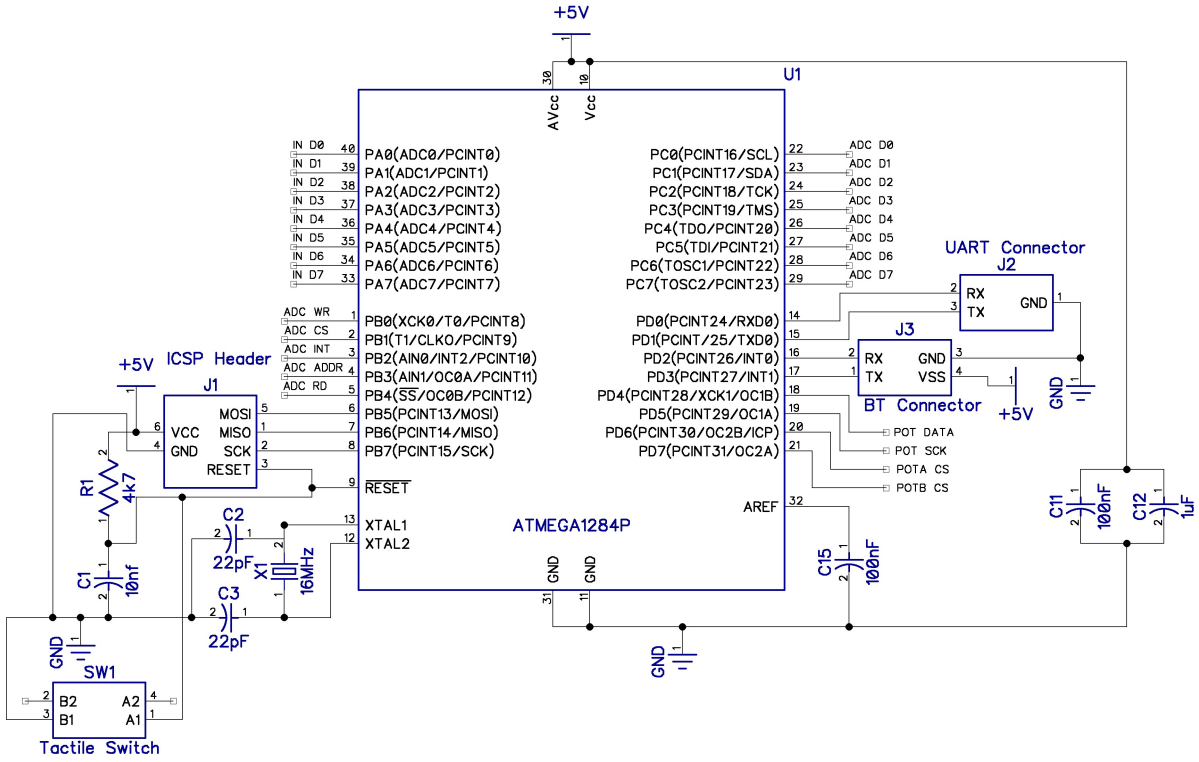



Figure 2.12: Circuit diagram showing connections to the microcontroller

$PD0$ and $PD1$ (in their role as $RXD0$ and $TXD0$ for the chip's inbuilt UART communication) must be connected to $RX$ and $TX$ respectively on the USB to UART module[11].

Similarly, $PD2$ and $PD3$ (in their role as $RXD1$ and $TXD1$ for the inbuilt UART) must be connected to $TX$ and $RX$ respectively on the Bluetooth module[12].

$PB5$, $PB6$ and $PB7$ (in their roles as $MOSI$, $MISO$ and $SCK$ in the chip's programming connections) should be connected to $MOSI$, $MISO$ and $SCK$ respectively on the ISP (programming) header. The $V_{CC}$ and $GND$ on the programming header should also be connected to $V_{CC}$ and $GND$ on the chip.

$RESET$ (an active-low input that restarts the microcontroller) should be connected to $V_{CC}$ through a 4.7 kΩ resistor, with a 10 nF decoupling capacitor down to ground. It should also be connected to $RESET$ on the ISP header (when programming the microcontroller, the computer will need to reset it) and through a switch down to ground (to allow the user to restart the microcontroller if necessary).

Bus A (i.e., $PA0$ through $PA7$) will be connected directly to the digital inputs, and bus C (i.e., $PC0$ through $PC7$) will be connected to the data outputs ($D0$ through $D7$ of the ADC). The ADC control lines will be connected to the unused pins on bus B: $WR$ to $PB0$, $CS$ to $PB1$, $INT$ to $PB2$, $ADDR$ to $PB3$ and $RD$ to $PB4$.

Finally, the digital potentiometers will be connected to the unused pins on bus D: $DATA$ to $PD4$, $SCK$ to $PD5$ and the two $CS$s to $PD6$ and $PD7$.

[11]The module labels $RX$ such that it should be connected to $RX$ on the microcontroller

[12]Opposite to the USB module, the Bluetooth module labels $RX$ such that it should be connected to $TX$ on the microcontroller

All of these connections are shown visually in fig. 2.12.

### Programming

To program the chip, an AVR programmer must be used (and connected to the ISP header as detailed above). As this author already had a USBASP programmer (approximately £5 from eBay), this was used. Alternatively, an Arduino (a very popular microcontroller prototyping board) can be connected to program AVR chips.

### Serial Interface

The Android device and the microcontroller will communicate via Bluetooth, which will emulate a serial link. In essence, a form of communication will be established where sequences of bytes can be sent by both devices simeltanously.

The following protocol[13] was designed for serial communication between the Android device and the microcontroller. After taking a full 1024 samples, the microcontroller checks the serial buffer for one of the following commands and acts on it, responding via serial if necessary.

Traditionally, serial is a textual interface, however for efficiency's sake in this case it was used as a binary interface. Each transmitted character is stored as an 8-bit character, so the protocol is based on the integer values of those characters. For example, 65 below corresponds to a capital 'A'.

`0x00` Return all samples for the first analogue channel (1024 bytes, followed by a newline.[14])

`0x01` Return all samples for the second analogue channel (1024 bytes, followed by a newline)

`0x02` Return all digital samples (1024 bytes, with each byte representing all 8 samples at one timepoint, followed by a newline)

`0x06` Wait for the next two transmitted bytes, $X$ and $Y$ and use $XY$[15] as the number of microseconds to pause in between digital samples (returns a newline)

`0x08` Wait for the next transmitted byte, $X$, and use the 7 least significant bits of $X$ as the digital pot value in an amplifier (if the most significant bit of $X$ is high, the second amplifier circuit, otherwise the first amplifier circuit) (returns a newline)

`0x09` Toggle the status of analogue channels 1 and 2, based on the first and second LSBs in the next transmitted byte $X$ (returns a newline)

`0x0A` Noop (this is the newline character) (returns a newline)

`0x0C` Return the current digital time delay (returns two bytes then a newline)

`0x11` Set the time delay between analogue samples based on the next two transmitted bytes $X$ and $Y$ (returns a newline)

[13]The commands are given sequential bytes, however some bytes are skipped because a number of command bytes were used solely in the prototyping phase

[14]The character used to indicate a new line should start in text on a computer. Here, it's just equivalent to `0x0A`

[15]For example, if $X =$ `0x50` and $Y =$ `0x1A` then use `0x501A` as the delay

`0x12` Return the current analogue time delay (returns two bytes then a newline)

So, for example, to receive samples for the second analogue channel the Android device would have to send `0x01`, then store the next 1024 received bytes.

### ADC Communication

The way in which the microcontroller will communicate with the ADC is described in section 2.1.4. Note that while $INT$ is not required for communication, it is connected as an input to the microcontroller in case the communication protocol is changed (i.e., a different mode is used over pipelined) during the prototyping process.

### Digital Potentiometer Commmunication

The resistance of the digital potentiometers is set using an SPI interface. While the details can get more complicated, SPI (Serial Peripheral Interface) is essentially a very simple interface, consisting of four pins: $SCLK$, $MOSI$, $MISO$ and $\overline{SS}$.

There are two devices: a master and a slave. $SCLK$ is simply a clock output from master. On either the rising or falling edge of $SCLK$, a bit of data is transmitted from master to slave on $MOSI$ and slave to master on $MISO$. $\overline{SS}$ is simply an active low slave select (i.e. the slave only communicates when $\overline{SS}$ is low).

There are four different SPI modes, of which the digital potentiometers support two ($CPOL = 0 = CPHA$ and $CPOL = 1 = CPHA$). In both modes, data is captured on the rising edge and propogated on the falling edge. The difference is simply the first value of the clock: in the first mode, it's low, and in the second it's high.

The maximum clock supported by the digital potentiomers is 10 MHz, which is faster than we can reliably communicate at with a 16 MHz microcontroller anyway.

To send a 16-bit byte to the digital potentiometer the following must be done:

1. Pull $\overline{SS}$ low
2. Wait one $SCLK$ period
3. Starting from a high (an artbitrary choice), pulse $SCLK$
4. On each falling edge, send a bit of data until 16 bits have been sent
5. Pull $SCLK$ back high
6. Wait one clock period
7. Pull $\overline{SS}$ back high

The potentiometer has 128 resistance levels that can be chosen from (with 0 being 75 Ω and 127 being 10 kΩ). Suppose we want to set the resistance to a level $x \in [0, 127]$. Then let $P$ be the 7-bit binary number representing $x$. To set the resistance to $x$, we write 9 zeroes followed by $P$.

As detailed in the serial protocol above, the Android device sends a byte $B$, where the MSB determines which potentiometer to change the the 7 LSB determine the value to send. To determine the address from this we can run

```
Byte address = (control & 0x80) ? 0x01: 0x00;
```

to set *address* to the MSB of control. Similarly, we can set the value by getting the 7 LSB using

```
Byte level = control & 0x7F;
```

#### Program

To perform all of the above tasks, then, the microcontroller will need to run the following program:

- Setup serial communication (Bluetooth)
- Setup potentiometer SPI communicator
- Setup ADC communication and initialise data variables
- Loop forever:
  - Loop 1024 times:
    - Take sample from first analogue channel, if enabled
    - Take sample from second analogue channel, if enabled
    - Take 1-bit sample from each digital channel
  - Look at the most recent serial command, receive up to 2 more bytes depending on the command, and perform any actions and send any data it specifies
  - Clear most recent serial command

ISRs will not be used for serial communication as an oscilloscope is heavily reliant on accurate timing — something that ISRs will interrupt.

There are two main programming languages that can be used for AVR microcontrollers: Assembly and C. While Assembly was taught in the course, your author has more extensive previous personal experience using C to program microcontrollers. Additionally, the higher level of C means less code will be needed to obtain the same result, and that code wil be more logically structured. Furthermore, modern C compilers can optimise C code to a smaller amount of Assembly than any human programmer, meaning the only reason to use Assembly is for educational purposes.

The completed PIC program can be seen in appendix B.

### 2.1.6 Bluetooth Transceiver

As discussed in section 1.5, Bluetooth was chosen to communicate with the Android device acting as the user interface. In particular, the JY-MCU HC-06 was chosen.

This communicates with the microcontroller via the inbuilt UART interface, as illustrated in the code in listing 5.

```
#define FOSC    16000000          // Clock speed of the microcontroller
#define BAUD    19200             // Baud rate to communicate with
#define UBRR    (FOSC/16/BAUD-1)  // Calculated constant used to set speed

// Import libraries for interrupts and USART
#include <avr/io.h>
#include <avr/interrupt.h>

// Setup USART. Should be called at the start of the program.
static void USARTInit(void) {
  // Set baud rate
  UBRR0H = (Byte) (UBRR >> 8);
  UBRR0L = (Byte) UBRR;

  UCSR0B = (1<<RXEN0)|(1<<TXEN0); // Enable receiver and transmitter
  UCSR0C = (1<<USBS0)|(3<<UCSZ00); // Set frame format: 8 data, 2 stop bit

  // Enable an interrupt when a byte is received
  UCSR0B |= (1 << RXCIE0);
  sei();  // Enable global interrupts
}

// Write a byte
void USARTWriteByte(Byte data) {
  // Wait until the transmit buffer is empty (i.e. all previous bytes have
  // been sent)
  while (!(UCSR0A & (1<<UDRE0)));

  // Send the byte by putting it into the transmit buffer
  UDR0 = data;
}

// Read a byte
Byte USARTReadByte(void) {
  loop_until_bit_is_set(UCSR0A, RXC0); // Wait until data exists in the buffer
  return UDR0; // Return the received byte
}

// The interrupt called when a byte is received
ISR(USART_RX_vect) {
  // The byte that has been received
  Byte data = UDR0; // Do something with this: the byte that has been received
}
```

Listing 5: Example code used to communicate over Bluetooth via the USART interface

### 2.1.7 User Interface

A high-quality enough screen that interfaced directly with the microcontroller would be quite expensive, so instead an Android device was chosen as the user interface.

Android was chosen over the alternatives (iOS, etc) primarily because it's the most popular mobile operating system, allowing the greatest number of people to use the oscilliscope. Additionally, it allows easy and open development, whereas competitors such as iOS require costly developer licenses, proprietary software and specific hardware.

As it's by far the most common and most supported language used for Android, Java was used to create an Android application that interfaced with the oscilloscope. Discussing the development of this application is far beyond the scope (pun not intended) of this project, but the code is fully documented and available on GitHub at `http://github.com/hrickards/bioniscope`. A number of screenshots of the working app are shown in figs. 2.13 to 2.15.

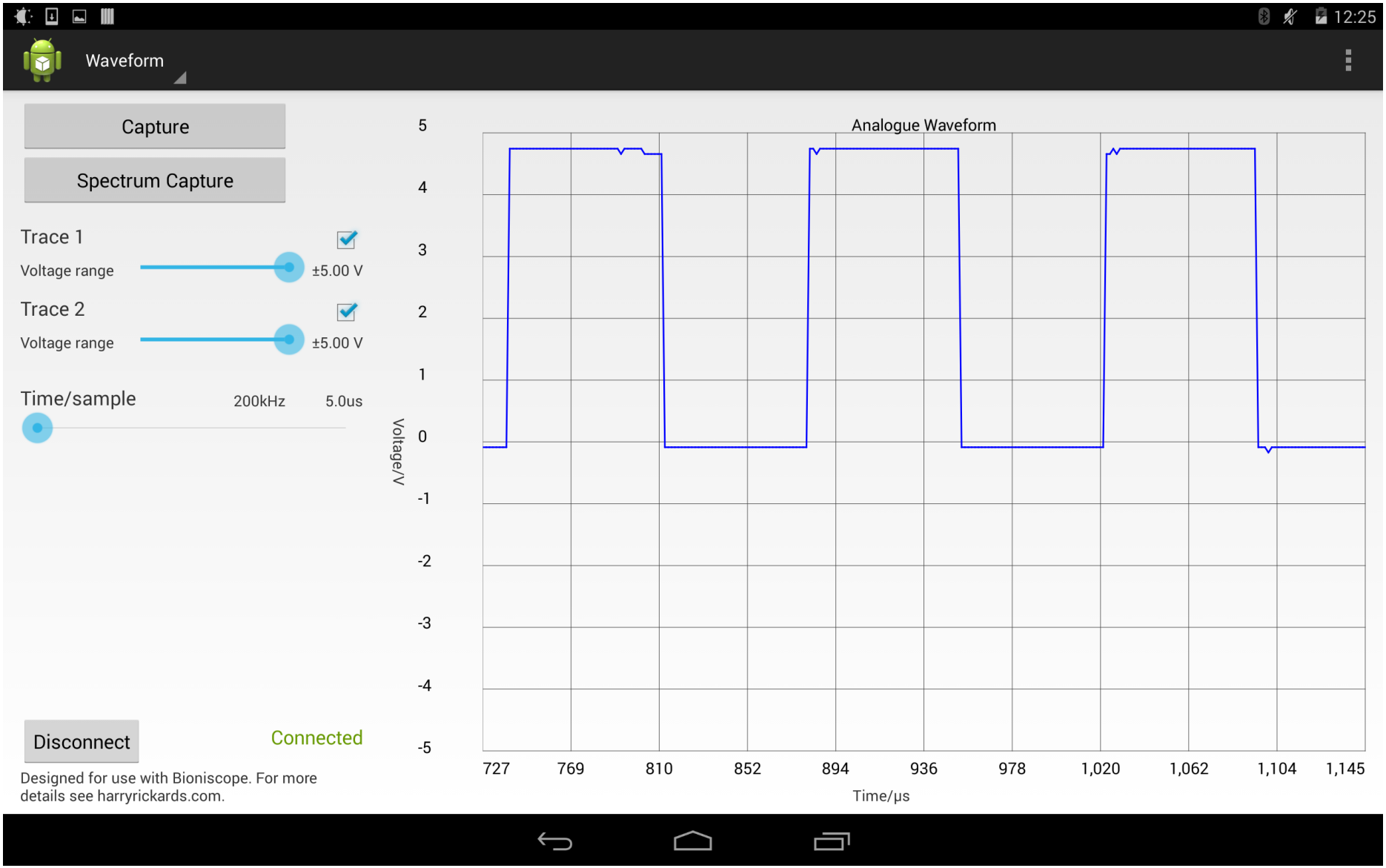



Figure 2.13: Analogue waveform shown in Android Bioniscope app

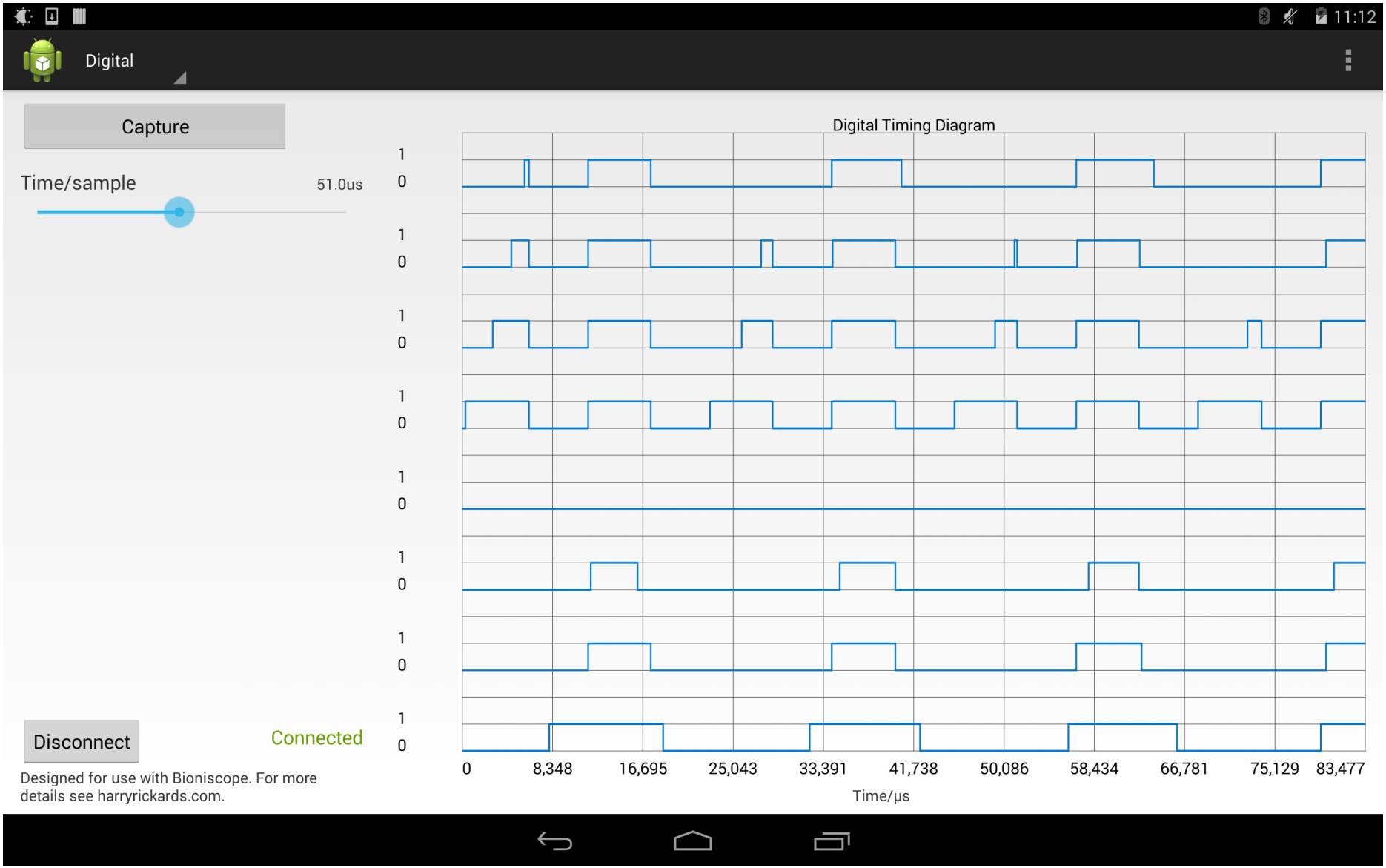



Figure 2.14: Digital timing diagram shown in Android Bioniscope app

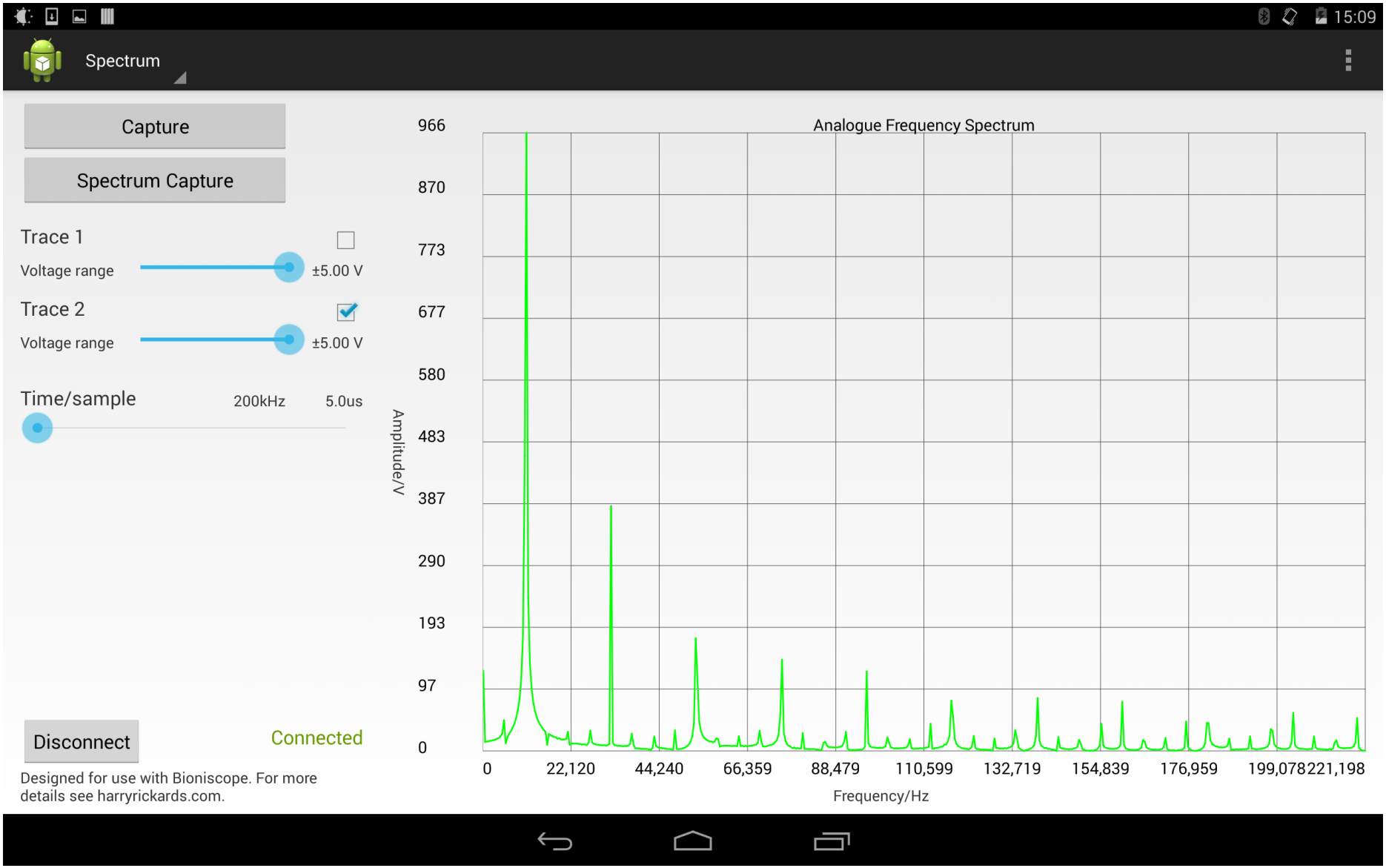
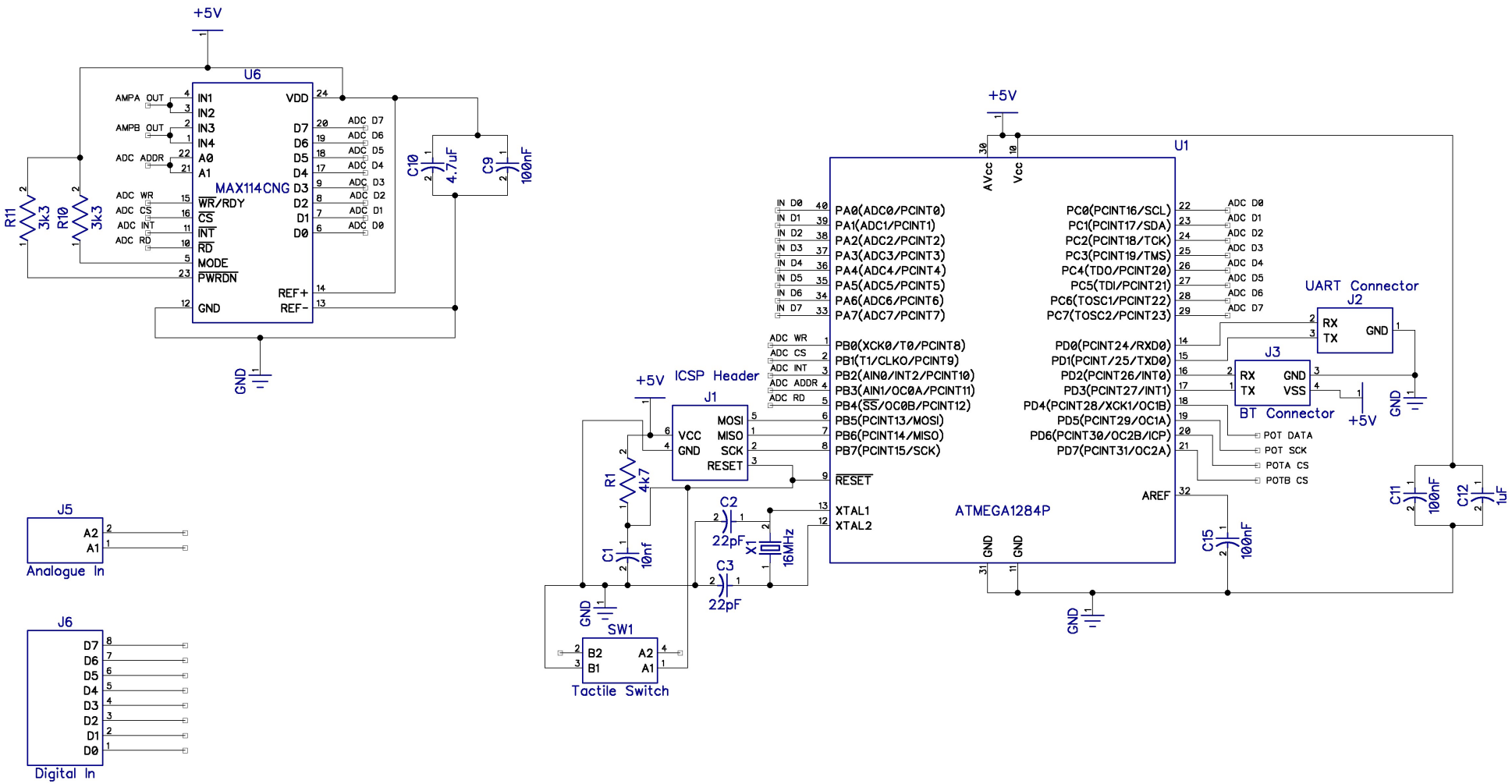


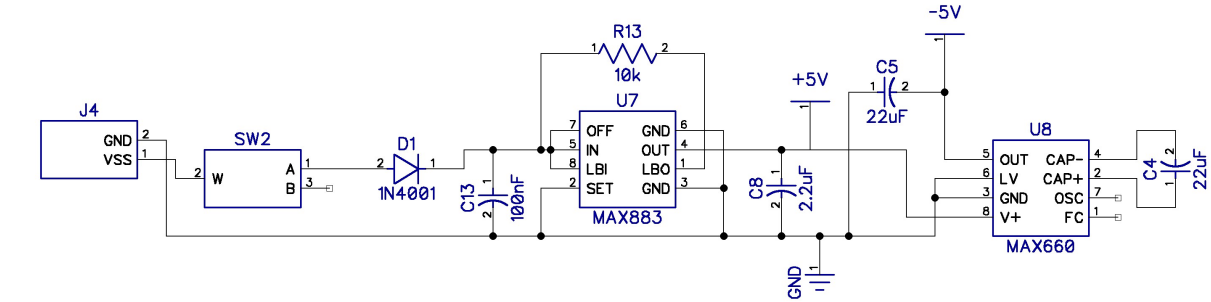

Figure 2.15: Analogue frequency spectrum shown in Android Bioniscope app

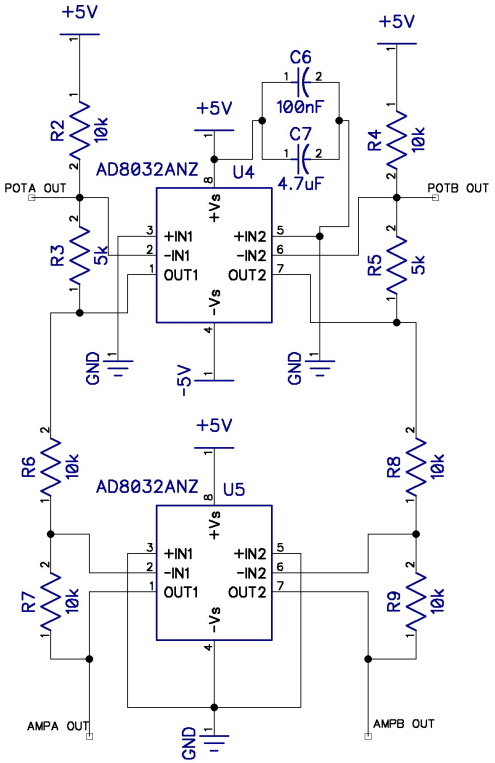
## 2.2 Circuit

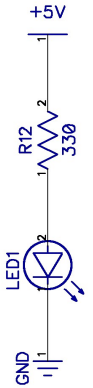
Combining all of the subsystems in section 2.1, the overall circuit diagrams shown below ar reached.

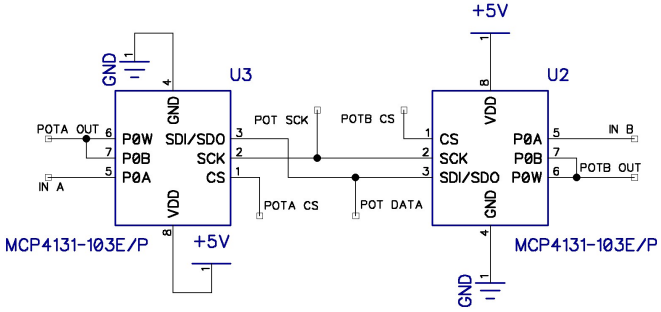


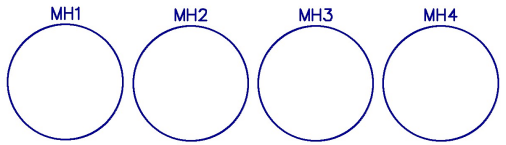

| | | |
|---|---|---|
| Oscilloscope Project | | |
| Sheet<br>Digital | Summary<br>Switch RX, TX on UART, cap on AREF<br>Fix C11, C12 orientation | Rev<br>V1 |
| Date: 12/02/14 | | Harry Rickards |
| Licensed CC BY-SA 4.0 | | #001 |




| | | |
|---|---|---|
| Oscilloscope Project | | |
| Sheet<br>Analogue | Summary<br>LBO to Vin through pullup | Rev<br>V1 |
| Date: 12/02/14 | | Harry Rickards |
| Licensed CC BY-SA 4.0 | | #002 |

## 2.3 Construction

Initially, a number of the subsystems were built individually on breadboards. This allowed the basic circuit details to be checked without requiring an infeasibly large circuit to be built on a breadboard.

### 2.3.1 PCB Layout

Once this prototype verification step was performed, the circuit was laid out on a PCB. This process consists of three main steps: inputting the schematic into the computer, laying out the schematic components onto the board and routing traces (the PCB equivalent of wires) between the components.

Initially this was attempted using a piece of software called Eagle. A significant number of electronics projects use Eagle, from small hobbyist projects right up to industrial products. However, as this author started to lay components onto the PCB using Eagle, an error was thrown: the free version of Eagle only allows a maximum PCB size of 100 mmx80 mm. For reasons discussed later, the PCB size needed to be 100 mmx100 mm. The cheapest paid-version of Eagle is $169, meaning it was much too expensive to purchase for the sake of this project. Instead, an alternative was sought.

This author researched and briefly evaluated two alternatives: KiCAD and DipTrace. KiCAD is a piece of open source software, meaning it's free to use [16], and DipTrace, a commercial piece of software with a free version available limited to 2 signal layers[17] and 300 pins[18].

Both are very capable pieces of software, and KiCAD would certainly be favourable for philosophical reasons. Howerver, the majority of the ICs used in the circuit are relatively uncommon. This means that CAD footprints (files containing the physical dimensions of the chip) are not readily available, so they must be manually created for each IC instead. This is a much quicker process in DipTrace than in KiCAD (and indeed than in Eagle), and for this reason alone DipTrace was chosen.

There were no major issues in creating the circuit schematic in DipTrace, although a number of revisions were required for small improvements. Laying out the PCB and routing the traces was much harder. DipTrace contains tools to automatically place and

[16]There are two disctinct definitions of free that should be considered here: gratis free (without cost to the user; this is somewhat confusingly also called free as in beer) and libre free (without compromise to the user's liberties; this is also known as free as in speech and essentially means that the user has the right to run the software however they want, as well as the right to modify it, meaning the source code is freely available). Software that's Free (with a capital F) is also known as open-source software (there are major differences between the two terms, however for the purposes of this explanation the reader can consider them identical) meaning it's both free as in gratis and free as in libre. Common examples include LibreOffice (hence the name) and GNU/Linux (the operating system running everything from the majority of super computers to televisions to traffic lights). For more on this subject, see Richard Stallman's book '*Free as in Freedom*', which is also Free and hence freely available on the internet.

[17]It would be impractical to have all of the traces in a PCB in the same horizontal plane, as they would not be able to cross at any point. Instead, a number of planes, or *layers*, are used. PCBs are made by pressing together a number of 2-sided PCB 'chapters', meaning that common numbers of layers are 2, 4 or 6 (with higher numbers possible but more expensive). As it's the cheapest available and offers enough trace routing, a 2 layer board will be used for this project

[18]This project uses somewhere in the region of 250 pins, so this limit is not a problem

route, however the electrical engineering community strongly recommends against using such auto routers. Placing and routing is a task that's extremely hard for a computer to find an optimal solution to[19]. While software can make use of heuristics[20], it's a commonly held belief that even the best autorouting software pales when compared to a human with a small amount of experience (it may help to think of PCB routing as art: while a computer can be programmed to come up with pieces of art, even a small child can create much better ones).

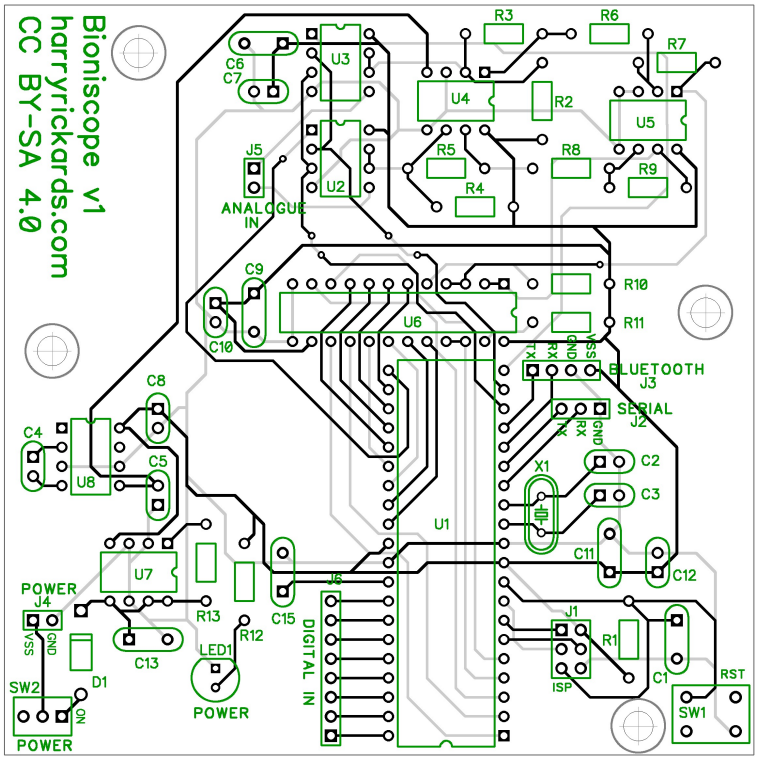
Because of this, your author set about learning how to place and route PCBs. The American electronics retailer Sparkfun provide a number of online tutorials and videos regarding PCB routing which were extremely helpful. Eventually, the PCB was routed, and while an expert would have done much better it sufficed for the purposes of this project (particularly as there were no high voltages or high-speed signals ($\geq 50\,\mathrm{MHz}$) involved). To finish the board, mounting holes and silkscreen[21] (the explanatory text and markings on a PCB) were added.

The routed PCB diagram can be seen in a scale version below, and a larger version in fig. 2.16. A 3D render the board can be seen in fig. 2.17. To check component footprints (sizes), an early version of the PCB was printed onto paper and attached to foam, as seen in fig. 2.18

[19]There is a classification of such problems, known as NP. Very briefly: a *running time* can be assigned to a given algorithm. For example, any algorithm to sort a list of numbers is at most $\mathcal{O}(n \log n)$, meaning that as the list of numbers grows longer, the running time of the algorithm grows slower than $n \log n$, where $n$ is the length of the list. We say an algorithm is polynomial-time if the running time is polynomial (so e.g., it's $\mathcal{O}\left(n^3\right)$ but not $\mathcal{O}(2^n)$). We then say an NP problem is one where the solution to the problem can be checked with a polynomial-time algorithm, but there is no polynomial-time algorithm to find that solution. An NP-hard problem is then a problem that is at least as hard as the hardest NP problems. Routing just one trace, with no obstacles to avoid, is an NP-hard problem

[20]A technique designed to find a good-enough solution to a problem, when finding the optimal one would take too long

[21]The process is just screen printing, which originally used silk as the screen, hence the name *silkscreen*




| Oscilloscope Project | | |
|---|---|---|
| Sheet<br>PCB | Summary | Rev<br>V1 |
| Date: 12/02/14 | | Harry Rickards |
| Licensed CC BY-SA 4.0 | | #003 |

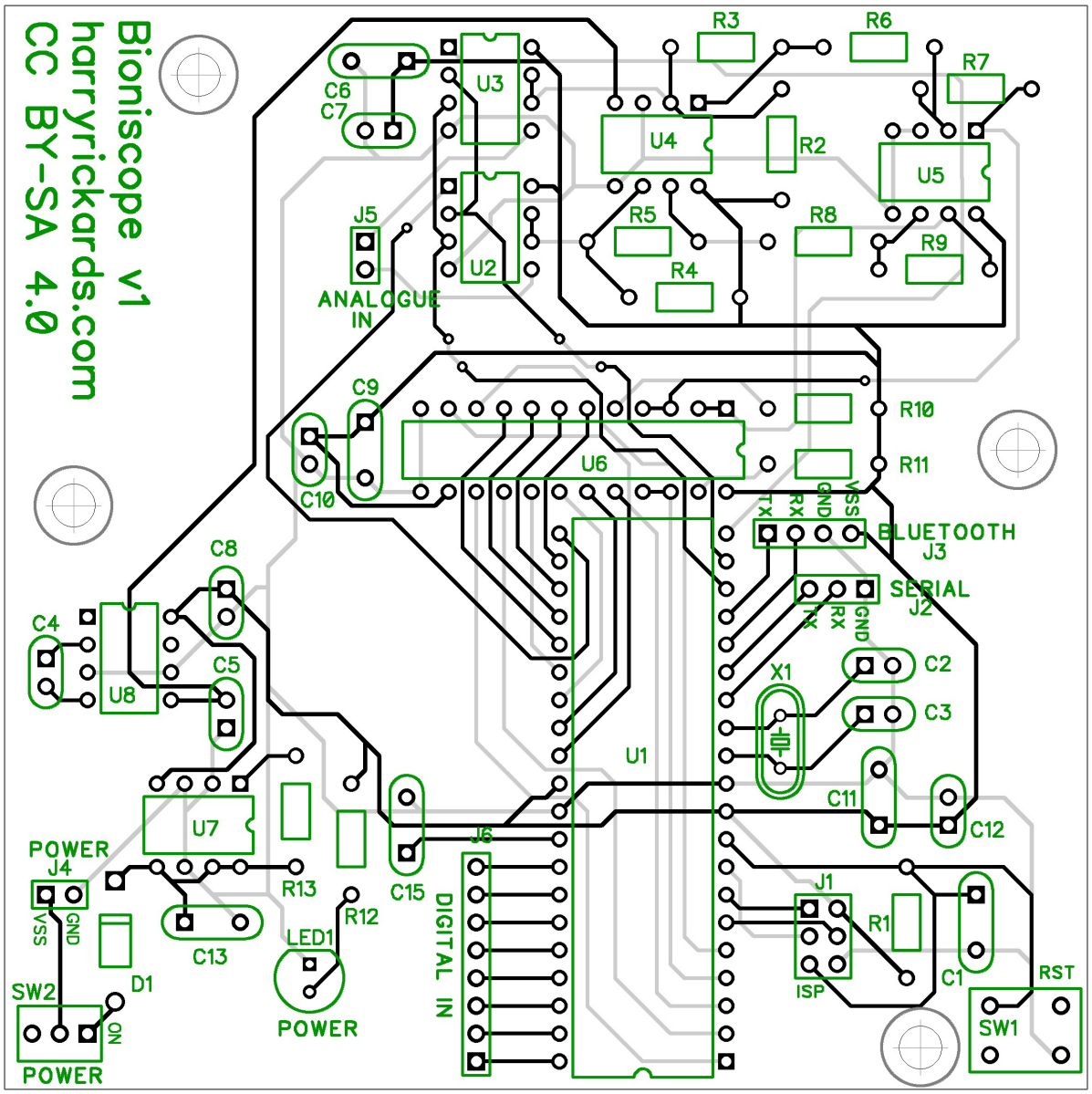



Figure 2.16: Routed PCB

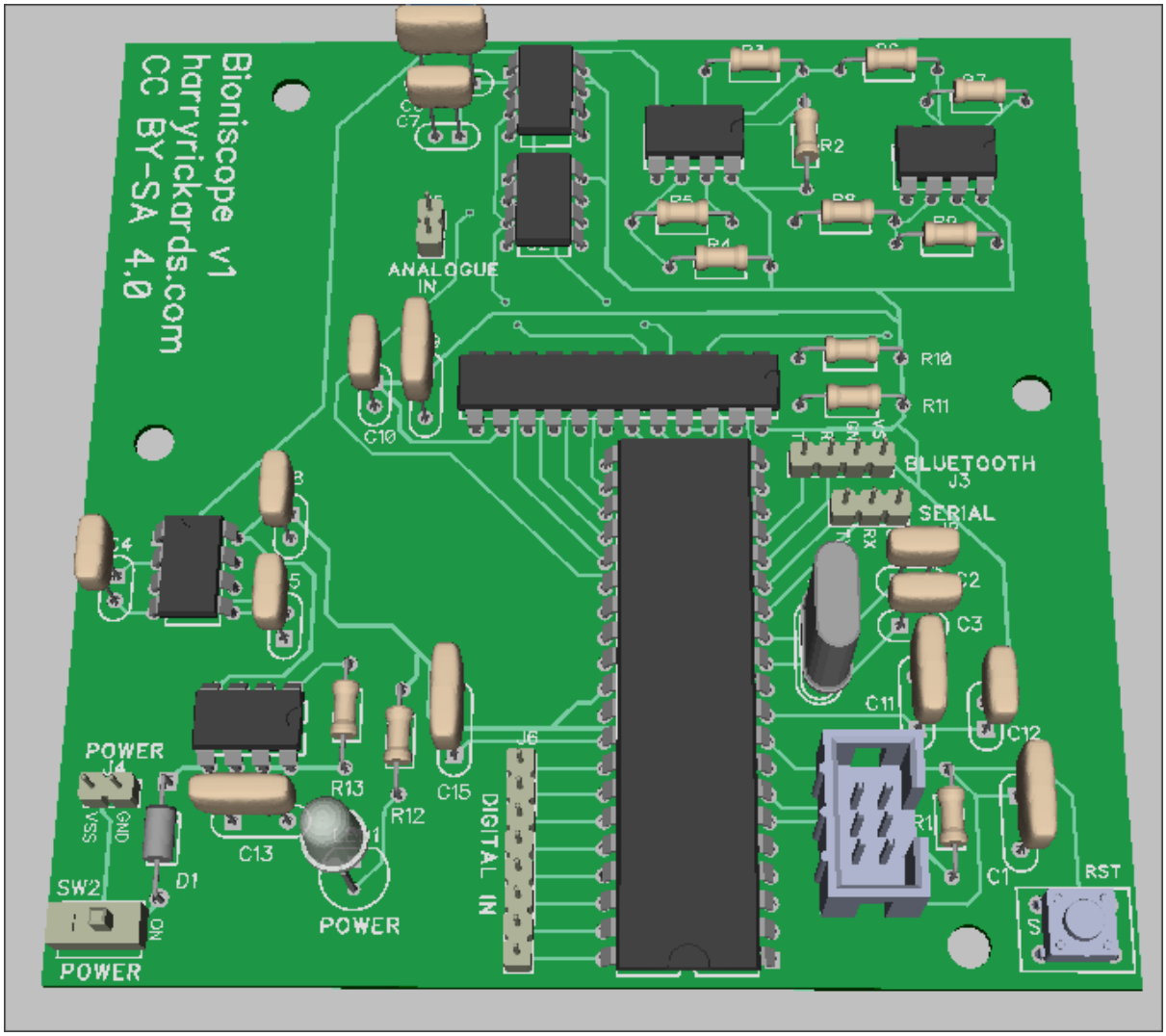



Figure 2.17: 3D render of the PCB populated with components (produced by DipTrace)

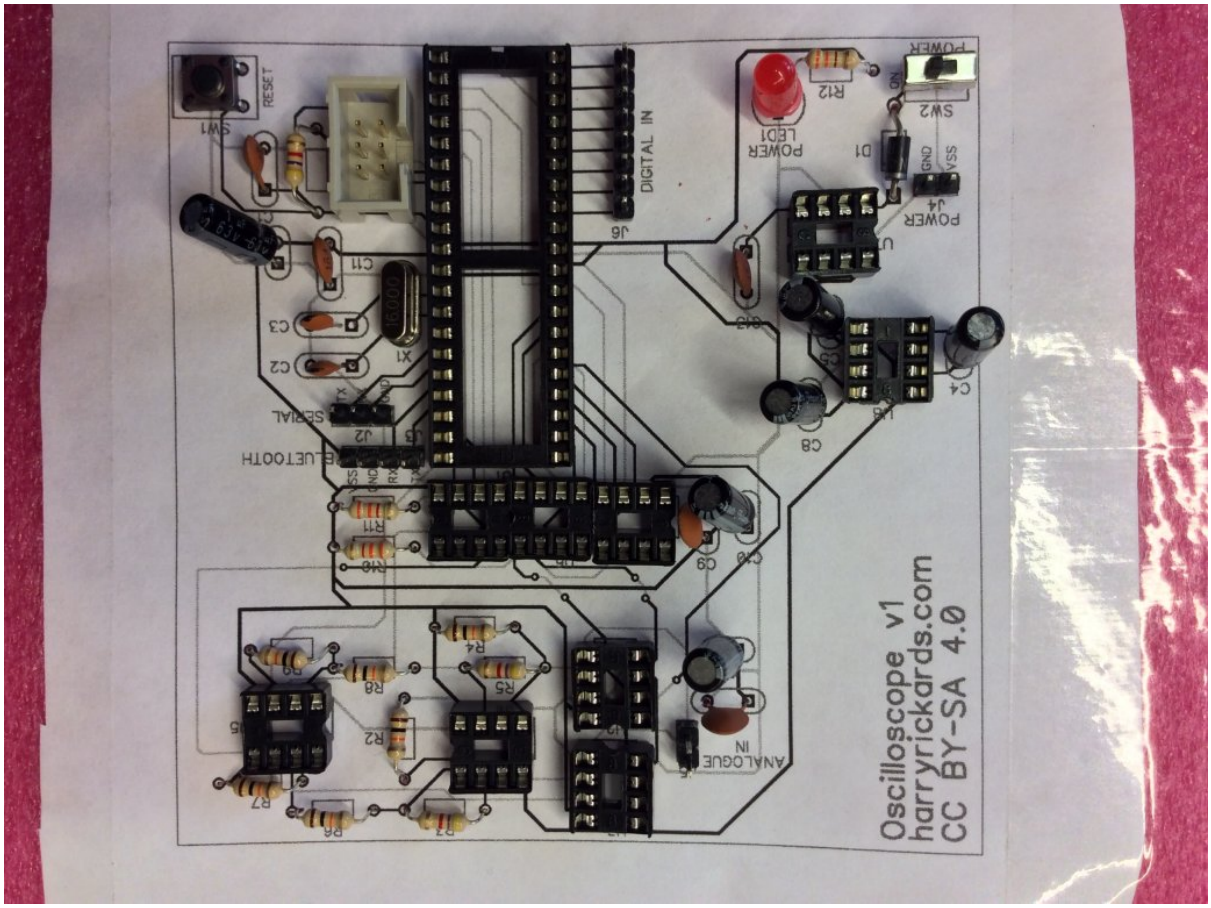



Figure 2.18: Photograph of a mockup of the PCB produced using paper and foam, allowing the sizes and placements of components to be checked

### 2.3.2 PCB Production

The final step in the process was to get the board manufactured. In an American or European fab house (the name for the company that prints the boards) this would be extremely expensive. For example, popular EuroCircuits quoted a price of over over £90 to produce just **one** board.

Instead, a Chinese proxy to a Chinese fab house was chosen (nearly all low cost Chinese retailers use the Golden Phoenix fab house, but ordering directly from them requires a minimum order of 100 square inches, far beyond the size of this project[22]). DFRobot was chosen because it offered more affordable delivery options than the other Chinese options[23]. The boards cost $23.99 for 10 boards, with a $16 delivery charge on top. At a cost of $2.40 per board, ordering from a Chinese retailer made the boards over 60 times

[22]The proxies do something called panelisation: group a number of small boards onto a larger board, have the large board made at a low cost and then sell on the small boards for a small amount of profit

[23]EMS post is universally available for only a couple of dollars, but can take upwards of a month to arrive as it's simply air mail. Many Chinese retailers will also send goods worth hundreds of pounds through EMS post, simply writing '*birthday card*' on the side meaning that while the retailer doesn't have to pay any export fees there is a high chance of the package spending weeks stuck in customs in the UK. Instead, a courier service was chosen. Most Chinese PCB retailers only offer UPS and FedEx, which would have cost around $30 for this order. DFRobot offered DHL as well, which was just over half the cost at $16

cheaper than ordering from a European fab house!

As required, the boards were 100 mmx100 mm and 2 layers. There were less choices available (the boards hade to be green, and coated in HASL[24]) than from a European fab house, but that didn't matter for this project. Due to the recent Chinese New Year, the boards took slightly longer than usual to be produced, although still took only 2 working days, something that EuroCircuits charge over £150 extra for.

### 2.3.3 Obtaining Parts

All the ICs had to be specially obtained for the circuit, as they weren't available in school. Rather than have to purchase them, your author made good use of the samples programmes run by most semiconductor companies.

Take, for example, the MAX114 ADC. Such a specialist chip would have to be ordered from a large electronics distributor such as Mouser, who sell it at £4.71 per chip. Instead, a number (in this case, 2 were obtained along with 2 other Maxim ADCs of a similar cost for personal projects) can be freely obtained from Maxim Integrated, the semiconductor company who make the chip, by filling out a samples request. Maxim then sent the chips within 3 days (one of those days being Boxing Day), and they were delivered from the Philliphines in a further 2 days. All in all, the chips were delivered in the same time it would have taken had they been ordered from Mouser, but £25 had been saved[25].

One might question why companies such as Maxim provide this free service, and the answer is twofold. If an engineer used a free sample from Maxim when prototyping the '*next big thing*', a lot more chips would be required when producing the real product and Maxim would make a lot more money than it lost by giving out free samples to lots of engineers. Similar reasoning applies to students: if the engineers at multi-billion dollar companies all received free samples from Maxim when they were in college and hence developed brand loyalty, Maxim will make lots of money.

It should be pointed out that Maxim are not the only company running this scheme: as of writing, Maxim, Analog Devices, Texas Instruments (or National Semiconductor), Atmel, Fairchild Semiconductor, Microchip and Linear Technology are just some of the more well-known companies that offer this service (in fact, the first 4 provided free samples for this project). Furthermore, if one becomes a little more adventorous and searches a little further afield, free samples can be found for everything from enclosures to superbright LEDs. If the reader is interested in more information, you're advised to consult Fried (*Finding Parts: Getting Samples from Manufacturers*).

## 2.4 Enclosure

To protect the circuit, an enclosure was needed. There were only a limited number of places were mounting holes could go on the PCB, so a custom-made enclosure had to be made rather than using a prebuilt one with standard mounting hole locations.

[24]If the boards were to be sold commercially, then HASL couldn't be used as it contains lead so breaks the EU Restriction of Hazardous Substances Directive, but for this hobby project using HASL is fine. This does suggest another reason why European fab houses are more expensive: hazardous substances such as lead can't be used in the board, whereas in China that's not an issue

[25]Assuming the standard £10 for delivery from Mouser.

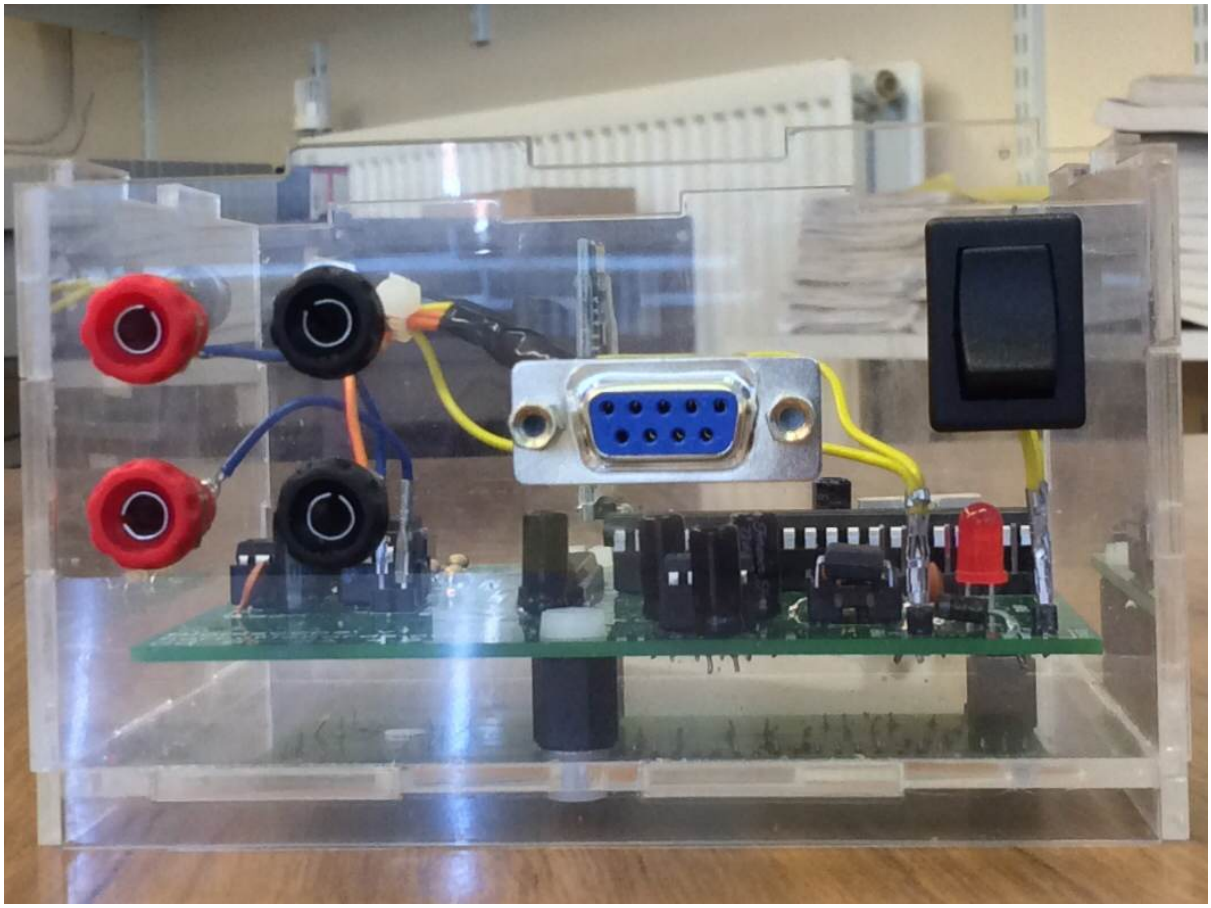


Figure 2.19: The final enclosure, with the circuit inside

The options available to this author were to produce the enclosure using either a 3D printer or a laser cutter. For easy examiner inspection, the enclosure needed to be transparent and only coloured plastic filament was available for the 3D printer, meaning the laser cutter needed to be used.

A case was designed using the industry standard 2D cad package *2D Design*. It was laser cut out of transparent acrylic and solvent welded together using *dichloromethane*. The final enclosure can be seen in fig. 2.19.

## 2.5 Final System

See figs. 2.20 and 2.21 for the final system.

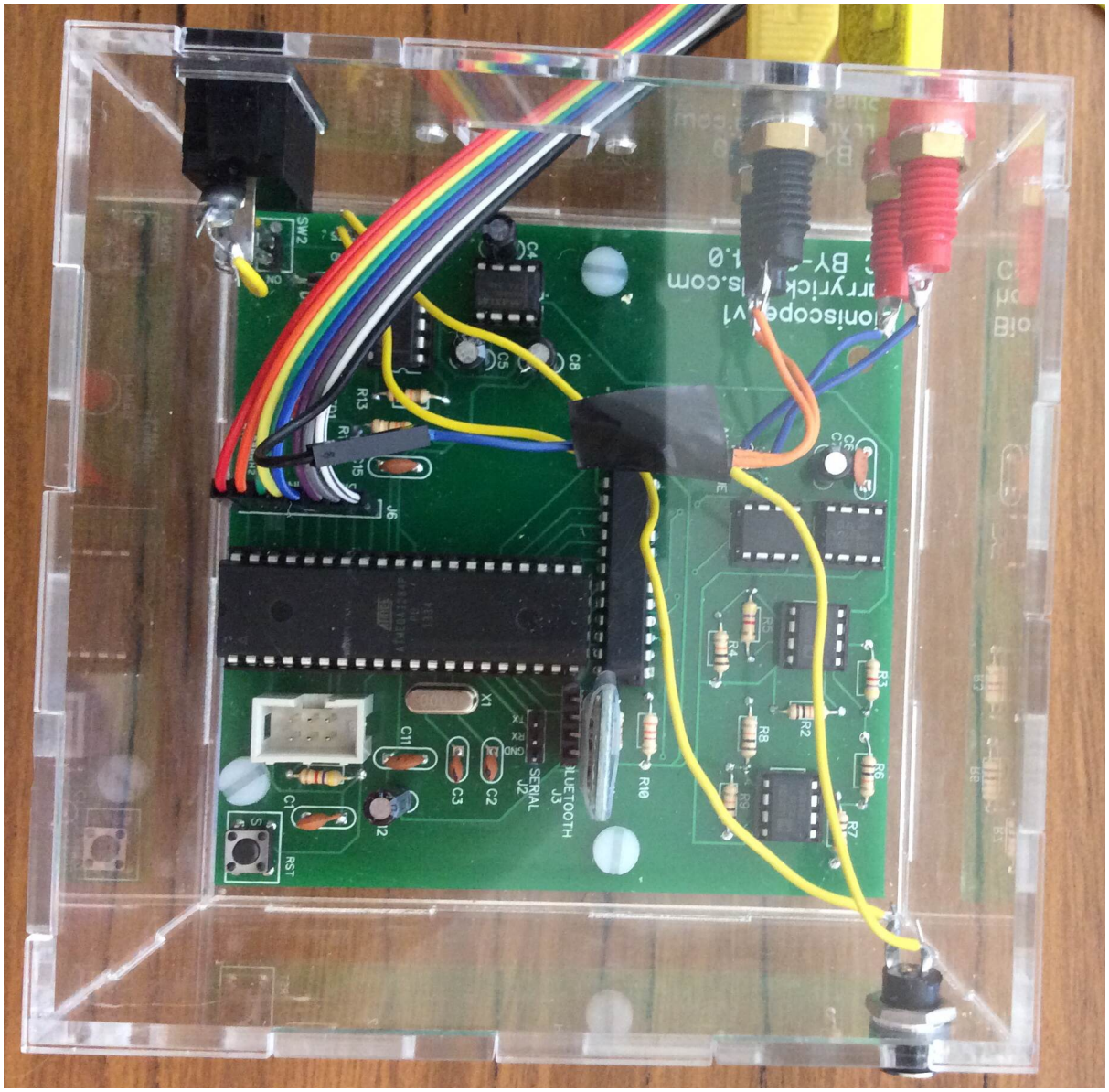



Figure 2.20: The final circuit

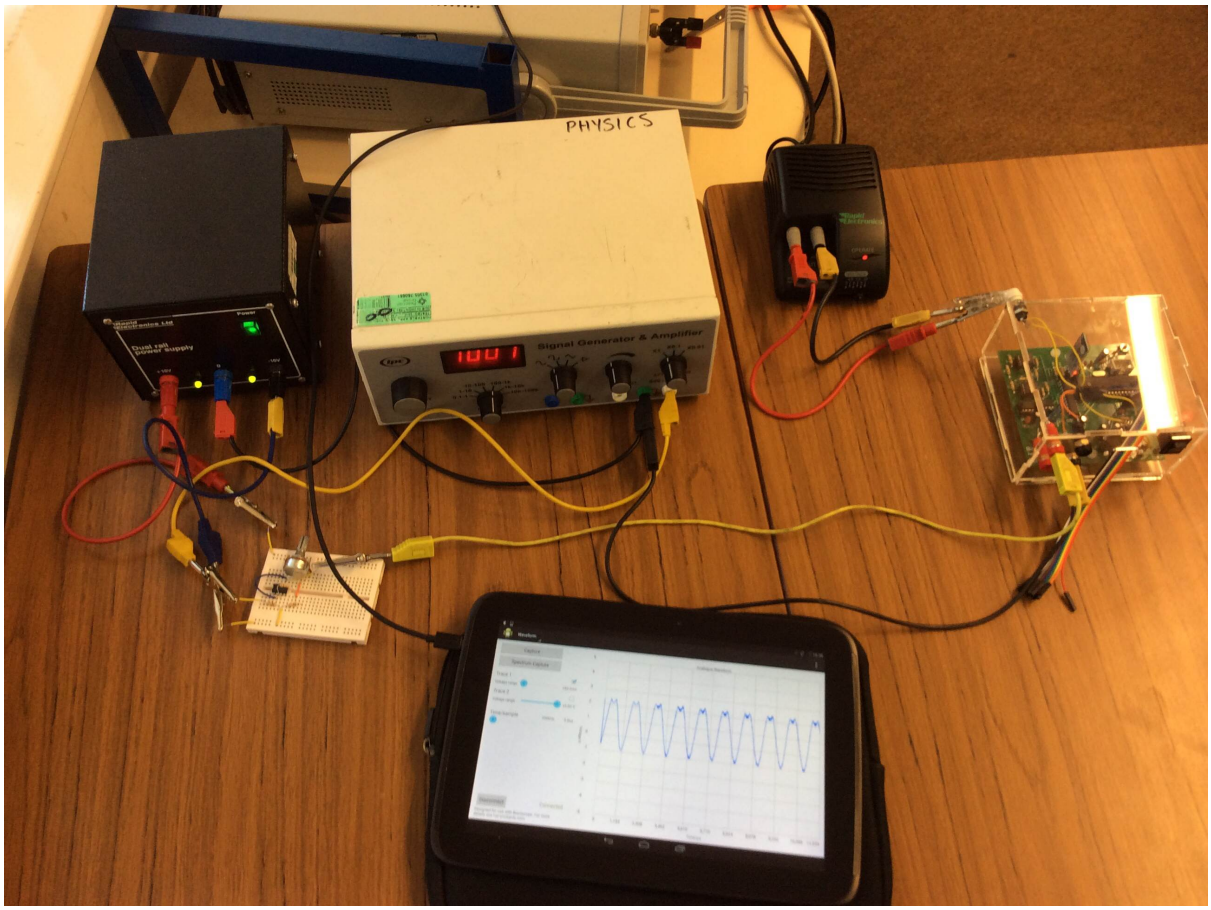



Figure 2.21: The final circuit in action

# Testing

## 3.1 Test Procedure

### 3.1.1 Overall System

Three different tests will be done to test the overall system. To run these tests a square wave and a sinusoid wave will be produced using a signal generator at 100 Hz, 1 kHz, 2 kHz, 5 kHz and 10 kHz. The signal generator has an accuracy of 2 decimal places, or between 1 Hz and 100 Hz in this test. While the error is not explicitly stated, the signal generator is a commercial product so the error will be less than 1 Hz (the accuracy).

To reduce error, all measurements will be repeated 3 times and an average value found and recorded. While the oscilloscope has a high enough impedance, capacitance and inductance that they may have an effect on the test signal, the signal generator will take this into account when calculating the signal frequency.

The sinusoid wave will be fed into the oscilloscope at each different frequency, and the time period and amplitude recorded from the waveform view. These will be compared to the known values to compute an accuracy value for the oscilloscope. Data will be recorded in table 3.1.

Similarly, the square wave will be fed into the oscilloscope and the time period recorded from the digital view. This will be compared to the known value to compute an accuracy value, and data will be recorded in table 3.2.

The square wave was also fed into the analogue input and the scope switched to spectrum mode. As seen in fig. 3.1 the spectrum looked exactly as one would expect for a square wave signal (based on the prediction made from appendix A).

| Frequency (kHz) | | | Amplitude (V) | | |
|---|---|---|---|---|---|
| Actual | Measured | % error | Actual | Measured | % error |
| 0.101 | 0.0963 | 4.65% | 4.9 | 4.8 | 2.04% |
| 1.01 | 1.01 | 0.00% | 4.9 | 4.8 | 2.04% |
| 1.99 | 2.04 | 2.51% | 4.9 | 4.7 | 4.08% |
| 5.01 | 5.11 | 2.00% | 4.9 | 4.8 | 2.04% |
| 10.1 | 10.1 | 0.00% | 4.9 | 4.8 | 2.04% |

Table 3.1: Results from analogue wave testing

| Frequency (kHz) | | |
|---|---|---|
| Actual | Measured | % error |
| 0.100 | 0.110 | 10.0 % |
| 1.06 | 1.00 | 5.66% |
| 2.00 | 2.02 | 1.00% |
| 4.99 | 4.43 | 11.2 % |
| 10.0 | 8.40 | 16.0 % |

Table 3.2: Results from digital wave testing

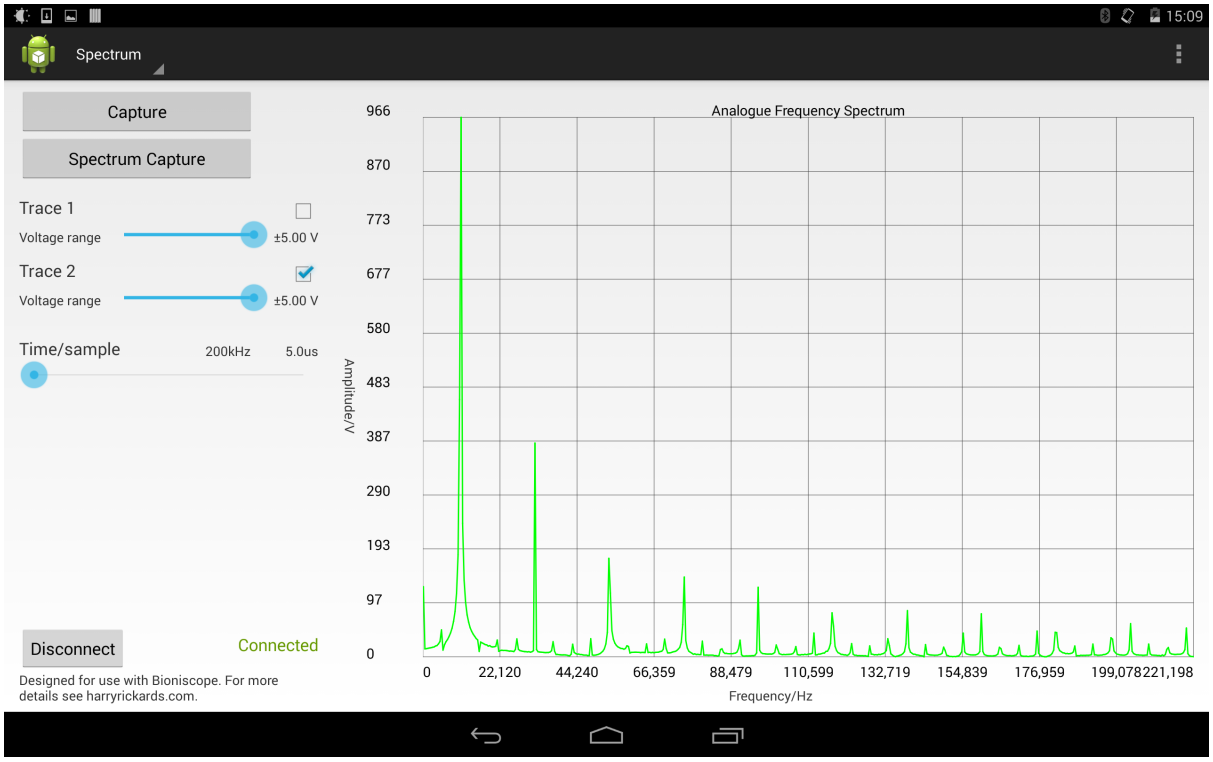



Figure 3.1: Frequency spectrum of square wave, as expected

### Results

Looking at the data above, error values are reasonably consistent so there are no outliers that need to be taken care of.

Taking means, we find the average error to be 8.8% for the digital wave and 1.8% for the analogue wave. While this would probably not be accurate enough for a commercial product, it is certainly accurate enough for this project. Besides, while some of the error undoubtedly comes from inaccuracies in the time delay code (it has to delay to the nearest whole multiple of 62.5 ns as the microcontroller uses a 16 MHz clock), the error could almost certainly be significantly reduced by using compnents with stricter tolerance values (some are 10% at the moment).

### 3.1.2 Initial Criteria

Next, the system needs to be checked to ensure it meets the numerical parameters specified in section 1.4.

#### Frequency

As seen above in section 3.1.1, the oscilloscope is accurate and copes with a 10 kHz wave as required by the system.

#### Bandwidth

To check the analogue bandwidth of the system, a 1 MHz square wave will be produced using a signal generator. This will be input into the oscilloscope, and a traditional CRO used to measure the output from the analogue circuitry. Using the figures and working detailed in section 1.3.1 we can compare the visual output to calculate the total bandwidth of the system.

The bandwidth of the CRO is already known to be approximately 20 MHz (from section 3.1.1) so provided the calculated bandwidth is less than 20 MHz we know this is the analogue bandwidth of the oscilloscope. If it's 20 MHz we know the analogue bandwidth is at least 20 MHz and hence greater than our requirement of 1 MHz (from section 1.4).

As we're counting discrete harmonics here, the frequency of the generated signal simply needs to be at least 50% accurate to ensure 100% accuracy for the number of harmonics. As a commercial signal generator will be used, it will certainly reach this level of accuracy. The final answer for the bandwidth will be accurate to within 1 harmonic of the fundamental frequency. While this is a significant error, there's no way to reduce this error value without using specialist equipment (i.e., a traditional RF spectrum analyser) or significantly increasing the complexity of the test. As we only need to verify the bandwidth is greater than 1 MHz, this is accurate enough for our purposes.

When the experiment was performed, the CRO showed the output seen in fig. 3.2. As can be seen by comparing to fig. 1.6, the square wave is present to significantly more than the eleventh harmonic. So the bandwidth is greater than 11 MHz, more than meeting the initial numerical parameters (which specified 1 MHz).

## 3.2 Initial Assessment of System

As discussed above, the system met all initial quantitative criteria. However, while performing the above tests a critical issue was discovered.

Whenever the input signal fell below 0 V, the output from the analogue circuitry was clipping. Upon further investigation, it was discovered this was because the digital potentiometer can only cope with voltages between 0 V and $V_{SS}$ — not those below 0 V.

Despite this, the oscilloscope still performed accurately and met the rest of it's goals. While at this stage it did have limited usefuleness for analogue signals (as a large number of analogue signals fall below 0 V), this did not impede the digital performance of the oscilloscope.

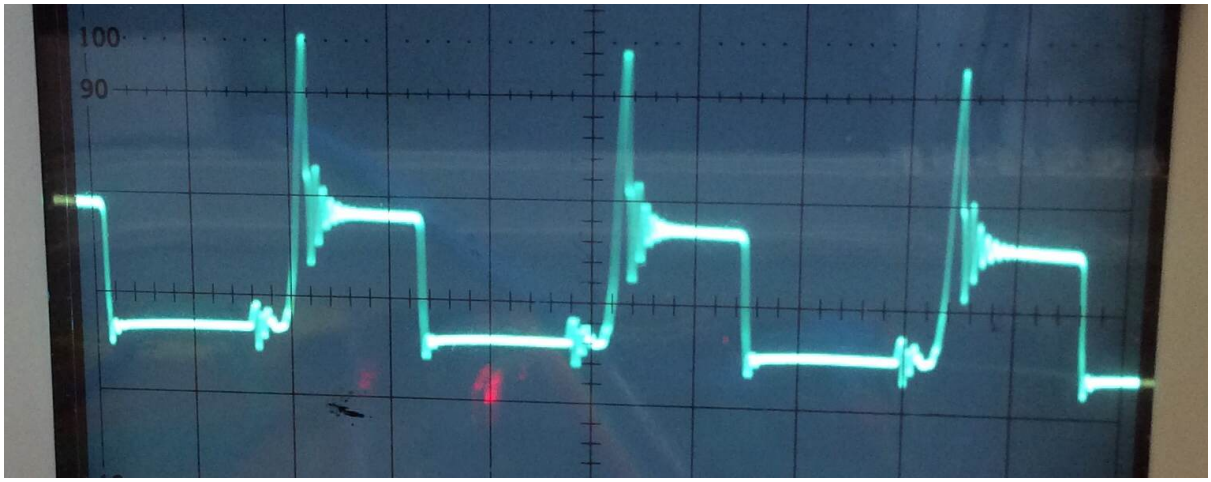



Figure 3.2: The bandwidth-constrained square wave after passing through the analogue circuitry

To overcome this issue, an op amp circuit was set up to scale a signal by $\frac{1}{2}$ and add on an offset of 2.5 V. This was implemented using a standard noninverting summing amplifier, as shown in figs. 3.3 and 3.4.

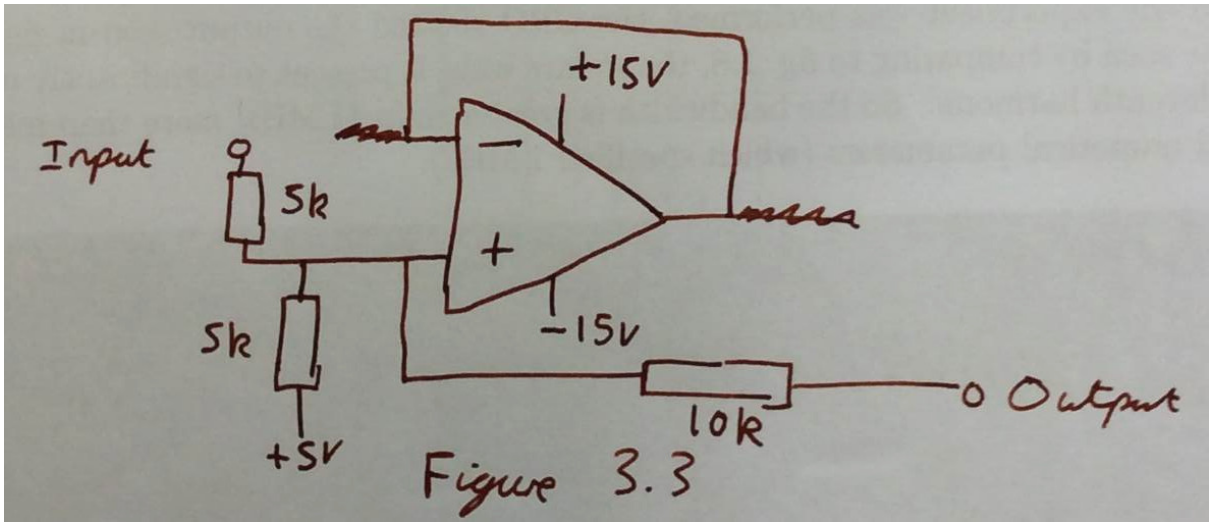



Figure 3.3: Circuit diagram for the op amp circuit added to fix the issue with negative signals

Now when testing the oscilloscope with signals that fell below 0 V it worked fine. At this point, the system met all quantitative criteria and also met the aim of the project: to produce a digital sampling oscilloscope that also provided basic spectrum analysis and logic analyser functionality.

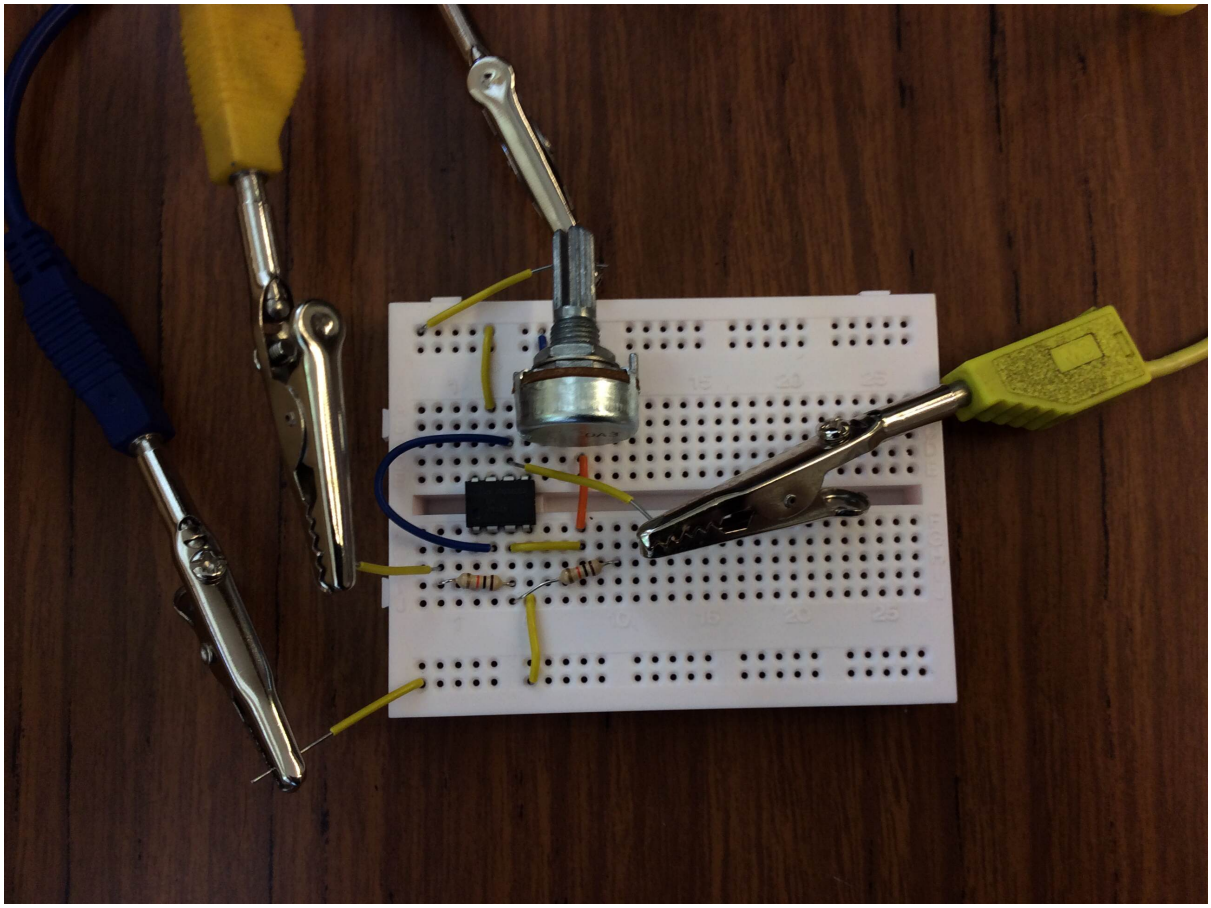


Figure 3.4: Breadboard photo of the op amp circuit added to fix the issue with negative signals

# Evaluation

As detailed in section 3.1.2, all initial criteria were met and system testing verified this.

The aim of this project was to create a Digital Sampling Oscilloscope, and that aim certainly succeeded. While it may not be the fastest or most accurate DSO available it's relatively unique in that it communicates with an Android tablet. The total cost (estimated to be about £35 if free samples were not available) also comes to much less than other DSOs, even the cheapest of which cost hundreds of pounds.

There were two other secondary aims of the project: to generate frequency spectrums and to have basic logic analyser capability. As seen in the testing and general screenshots, frequency spectrums are certainly generated. While their accuracy was not numerically tested as in-depth as the waveforms, the spectrums are generated from the waveforms using an industry-standard algorithm so will have the same standard of accuracy.

Basic logic analyser functionality was also implemented. At the moment, only timing diagrams can be displayed and no triggering functionality exists, which makes it less useful than a standalone logic analyser but still much better than a standard CRO. In the future, more advanced triggering could be implemented, as well as things like packet decoding (so data packets can be decoded and displayed all inside the logic analyser). This could all be implemented in software, so the hardware would not need to be modified.

## 4.1 Report Production

This report was produced in LaTeX. Whilst offering many advantages over alternatives that could have been chosen (such as Microsoft Word), LaTeXoffers superior typesetting, referencing and integration of mathematical equations and figures.

For more information on LaTeX, please consult a resource such as `http://www.sharelatex.com`.

# Fourier Series of a Square Wave

Consider a square wave $f(t)$ of amplitude $a$ and period $L$. An example is shown in fig. A.1.

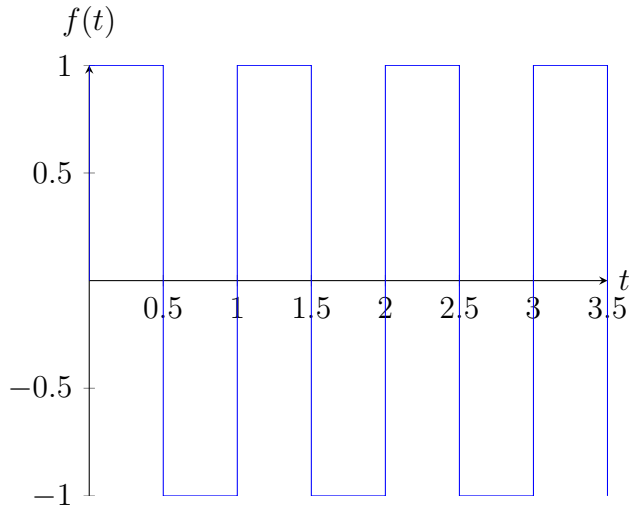



Figure A.1: Square wave with amplitude 1 and frequency 1 Hz

Now by any good mathematical methods textbook (in this case, we cite the excellent Riley, Hobson, and Bence (*Mathematical Methods for Physics and Engineering*)), a periodic function[1] $f(x)$ can be represented by the following series (known as a Fourier series)

$$f(x) = \frac{a_0}{2} + \sum_{r=1}^{\infty} \left[ a_r \cos\left(\frac{2\pi r x}{L}\right) + b_r \sin\left(\frac{2\pi r x}{L}\right) \right]$$

where

$$\begin{aligned} a_r &= \frac{2}{L} \int_{x_0}^{x_0+L} f(x) \cos\left(\frac{2\pi r x}{L}\right) \, \mathrm{d}x \\ b_r &= \frac{2}{L} \int_{x_0}^{x_0+L} f(x) \sin\left(\frac{2\pi r x}{L}\right) \, \mathrm{d}x \end{aligned}$$

[1] Strictly, the function must meet a number of conditions, called the Dirichlet criteria, but we shall just take it as a given that the square wave does meet all of these (the proof of each of them is essentially trivial depending upon your definition of a square wave)

where $x_0$ is arbitrary and $L$ is the period of the function.

A handy time-saving trick is to realise that the square wave is an odd function (i.e., $\forall x\ f(-t) = -f(t)$) so there will be no cosine terms in the series (again, this needs to be made more rigorous, but see a maths textbook) and so $a_r = 0$.

To evaluate $b_r$, we take $t_0 = -\frac{L}{2}$ and split up the integral into two parts. From $-\frac{L}{2}$ to 0 $f(t) = -a$ and from 0 to $\frac{L}{2}$ $f(t) = +a$. We then have

$$\int_{-\frac{L}{2}}^{\frac{L}{2}} f(t) \sin\left(\frac{2\pi r t}{L}\right) \mathrm{d}t = a \int_{-\frac{L}{2}}^{0} -\sin\left(\frac{2\pi r t}{L}\right) \mathrm{d}t + a \int_{0}^{\frac{L}{2}} \sin\left(\frac{2\pi r t}{L}\right) \mathrm{d}t$$

which, using the substitution $u = -t$ on the left integral, equals

$$2a \int_{0}^{\frac{L}{2}} \sin\left(\frac{2\pi r t}{L}\right) \mathrm{d}t = \frac{aL}{\pi r} \left[\cos\left(\frac{2\pi r t}{L}\right)\right]_{\frac{L}{2}}^{0} = \frac{aL}{\pi r} \left[1 - \cos\left(\pi r\right)\right]$$

Noting that $\cos 0\pi = 1$, $\cos 1\pi = -1$, $\cos 2\pi = 1$ and so on, we obtain

$$b_r = \frac{2a}{\pi r} \left[1 - (-1)^r\right] = \begin{cases} r \text{ even} & 0 \\ r \text{ odd} & \frac{4a}{\pi r} \end{cases}$$

and hence

$$\boxed{f(t) = \frac{4a}{\pi} \left(\sin \omega t + \frac{\sin 3\omega t}{3} + \frac{\sin 5\omega t}{5} + \ldots\right)}$$

where the angular frequency $\omega = \frac{2\pi}{L}$. Plotting the first terms of this equation gives the various graphs shown in fig. 1.6.

# PIC Code

```
/* ************************************************************************
   main.c
************************************************************************ */
#include <util/delay.h>
#include <avr/interrupt.h>

#include "USBUSART.h"
#include "Command.h"
#include "Global.h"
#include "Debugger.h"
#include "Pots.h"
#include "Sampler.h"

int main(void) {
  // Setup the command interface (Bluetooth)
  CommandSetup();

  #ifdef _DEBUG_
    // Setup the debugging interface (USB)
    DebuggerSetup();
  #endif

  // Setup potentiometer control interface
  PotsSetup();

  // Setup sampler (analogue and digital
  SamplerSetup();

  // Enable interrupts
  // sei();

  // Loop forever
  while(1) {
    // Take samples
    SamplerSample();
```

```
    // Respond to commands
    CommandRun();

    #ifdef _DEBUG_
      // Send any debugging info over USB
      DebuggerRun();
    #endif
  }
}
```

```c
/* ****************************************************************************
   Global.c
**************************************************************************** */
#include "Global.h"

int DigitalTimeDelay = 0;
int AnalogueTimeDelay = 0;
Byte AnalogueSampleChannels = 0xFF;
```

```
/* ************************************************************************
   Command.c
************************************************************************ */
#include <avr/io.h>
#include <avr/interrupt.h>
#include "Global.h"
#include "BTUSART.h"
#include "Pots.h"
#include "Sampler.h"

Byte Command  = 0x0A; // Most recently received command
Byte Data1          = 0x00; // First optional data byte sent with Command
Byte Data2          = 0x00; // Second optional data byte sent with Command

// Setup command interface
void CommandSetup(void) {
  // Initialise bluetooth
  BTUSARTInit();
}

// Receive command
void ReceiveCommand(void) {
  // Received command
  Command = BTUSARTRead();

  // Commands we need to receive more data on
  switch (Command) {
    // Set potentiometer value in amplifier
    case 0x08:
      Data1 = BTUSARTRead();
      break;

    // Set time delay between digital samples
    case 0x06:
      Data1 = BTUSARTRead();
      Data2 = BTUSARTRead();
      break;

    // Set time delay between analogue samples
    case 0x11:
      Data1 = BTUSARTRead();
      Data2 = BTUSARTRead();
      break;

    // Set which analogue channels to sample
    case 0x09:
      Data1 = BTUSARTRead();
      break;
```

```
    default:
        break;
  }
}

// Respond to commands
void CommandRun(void) {
  int i;

  // Check serial data
  ReceiveCommand();

  // Respond differently based on command
  switch(Command) {
    // Noop
    case 0x0A:
      break;

    // Set potentiomer resistance
    // 7LSB for value
    // MSB ? (channel B) : (channel A)
    case 0x08:
      // PotsSet takes care of everything for us
      PotsSet(Data1);
      break;

    // Return digital samples
    case 0x02:
      for (i=0; i<NUM_SAMPLES; i++) { BTUSARTTransmit(DigitalSamples[i]); }
      break;

    // Return analogue samples for the first channel
    case 0x00:
      for (i=0; i<NUM_SAMPLES; i++) { BTUSARTTransmit(AnalogueSamplesA[i]); }
      break;

    // Return analogue samples for the second channel
    case 0x01:
      for (i=0; i<NUM_SAMPLES; i++) { BTUSARTTransmit(AnalogueSamplesB[i]); }
      break;

    // Set time delay between digital samples
    case 0x06:
      DigitalTimeDelay = (Data1 << 8) | Data2;
      break;

    // Return the digital time delay
```

```
    case 0x0C:
      // MSB first
      BTUSARTTransmit(DigitalTimeDelay >> 8);
      BTUSARTTransmit(DigitalTimeDelay & 0xFF);
      break;

    // Set analogue time delay
    case 0x11:
      AnalogueTimeDelay = (Data1 << 8) | Data2;
      break;

    // Return the analogue time delay
    case 0x12:
      // MSB first
      BTUSARTTransmit(AnalogueTimeDelay >> 8);
      BTUSARTTransmit(AnalogueTimeDelay & 0xFF);
      break;

    // Look at bits 0 and 1 in Data1 to enable/disable analogue channels
    case 0x09:
      AnalogueSampleChannels = Data1;
      break;

    // Return error
    default:
      BTUSARTTransmit(0xFF);
      break;
  }

  // Reset command back to noop
  Command = 0x0A;

  // Return a newline
  BTUSARTTransmit(0x0A);
}
```

```c
/* ************************************************************************
   Debugger.c
************************************************************************ */
#include "Global.h"
#include "USBUSART.h"
#include "Sampler.h"

// Setup debugging
void DebuggerSetup(void) {
  // Initialise USB USART
  USBUSARTInit();
}

// Called repeatedly to do some form of debugging
void DebuggerRun(void) {
  int i;
  for (i=0; i<NUM_SAMPLES; i++) { USBUSARTTransmit(AnalogueSamplesA[i]); }
  USBUSARTTransmit(0x0A);
}
```

```
/* ************************************************************************
   BTUSART.c
************************************************************************ */
#define BAUD    9600      // Baud rate
#define UBRR    (F_CPU/16/BAUD-1)

#include <avr/io.h>
#include "Global.h"

void BTUSARTInit(void) {
  // Set baud rate
  UBRR1H = (Byte) (UBRR>>8);
  UBRR1L = (Byte) UBRR;
  // Enable receiver and transmitter
  UCSR1B = (1<<RXEN1)|(1<<TXEN1);
  // Set frame format: 8data, 2stop bit
  UCSR1C = (1<<USBS1)|(3<<UCSZ10);
}

void BTUSARTTransmit(Byte data) {
  // Wait for empty transmit buffer
  while (!( UCSR1A & (1<<UDRE1)));

  // Put data into buffer, sends the data
  UDR1 = data;
}

Byte BTUSARTRead(void) {
  // Wait until data exists
  loop_until_bit_is_set(UCSR1A, RXC1);
  return UDR1;
}
```

```c
/* ************************************************************************
   USBUSART.c
************************************************************************ */
#define BAUD    9600      // Baud rate
#define UBRR    (F_CPU/16/BAUD-1)

#include <avr/io.h>
#include "Global.h"

void USBUSARTInit(void)
{
  /* Set baud rate */
  UBRR0H = (Byte) (UBRR>>8);
  UBRR0L = (Byte) UBRR;
  /* Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
  /* Set frame format: 8data, 2stop bit */
  UCSR0C = (1<<USBS0)|(3<<UCSZ00);
}

void USBUSARTTransmit(Byte data)
{
  /* Wait for empty transmit buffer */
  while (!( UCSR0A & (1<<UDRE0)));

  /* Put data into buffer, sends the data */
  UDR0 = data;
}
```

```c
/* ************************************************************************
   Sampler.c
************************************************************************ */
#include <avr/io.h>
#include <util/delay.h>
#include "Global.h"
#include "ADC.h"

// Sample registers
Byte DigitalSamples[NUM_SAMPLES];
Byte AnalogueSamplesA[NUM_SAMPLES];
Byte AnalogueSamplesB[NUM_SAMPLES];

void SamplerSetup(void) {
  // Ensure digital input pins are inputs
  DDRA = 0x00;

  // Setup ADC
  ADCSetup();
}

void delay_us(int num) {
  int i = 0;
  for (i = 0; i < num; i++) {
    _delay_us(1);
  }
}

void SamplerSample(void) {
  int i = 0;
  if (DigitalTimeDelay<2) {
    // Take digital samples just as inputs to PORTA
    for (i=0; i<NUM_SAMPLES; i++) {
      DigitalSamples[i] = PINA;
    }
  } else {
    // Take digital samples just as inputs to PORTA
    for (i=0; i<NUM_SAMPLES; i++) {
      DigitalSamples[i] = PINA;
      delay_us(DigitalTimeDelay);
    }
  }

  // Take analogue samples using the functions in ADC.c
  // The four different cases based upon the value of AnalogueSampleChannels

  // Channel A enabled based on bit 0
  Byte aEnabled = AnalogueSampleChannels & 0x01;
```

```
  // Channel B enabled based on bit 1
  Byte bEnabled = AnalogueSampleChannels & 0x02;

  // A enabled and B enabled
  if (aEnabled && bEnabled) {
    for (i=0; i<NUM_SAMPLES; i++) {
      AnalogueSamplesA[i] = ADCSample(0x00);
      AnalogueSamplesB[i] = ADCSample(0xFF);
      delay_us(AnalogueTimeDelay);
    }
  // A enabled and B not enabled
  } else if (aEnabled) {
    for (i=0; i<NUM_SAMPLES; i++) {
      AnalogueSamplesA[i] = ADCSample(0x00);
      delay_us(AnalogueTimeDelay);
    }
  // B enabled and A not enabled
  } else if (bEnabled) {
    for (i=0; i<NUM_SAMPLES; i++) {
      AnalogueSamplesB[i] = ADCSample(0xFF);
      delay_us(AnalogueTimeDelay);
    }
  // Error: neither channel enabled
  } else {
    for (i=0; i<NUM_SAMPLES; i++) {
      AnalogueSamplesA[i] = 0x00;
      AnalogueSamplesB[i] = 0xFF;
    }
  }
}
```

```
/* ************************************************************************
   ADC.c
************************************************************************ */
#include <avr/io.h>
#include <util/delay.h>
#include "Global.h"
#include "USBUSART.h"

#define WR    0x01<<0 // PB0
#define CS    0x01<<1 // PB1
#define INT   0x01<<2 // PB2
#define ADDR  0x01<<3 // PB3
#define RD    0x01<<4 // PB4

void ADCSetup(void) {
  // Ensure ADC data outputs are input pins
  DDRC = 0x00;

  // Output high on CS, RD, WR and ADDR
  DDRB |= CS | RD | WR | ADDR;
  PORTB |= CS | RD | WR | ADDR;

  PORTC = 0x00;

  // INT input
  // DDRB &= ~INT;
}

// Operating in pipelined mode
Byte ADCSample(char address) {
  // Set address of the ADC
  if (address == 0xFF) {
    PORTB |= ADDR;
  } else {
    PORTB &= ~ADDR;
  }

  // Pull CS, RD and WR low
  PORTB &= ~CS & ~RD & ~WR;

  // Small delay to let the ADC gather the data
  // 0.25 to 10uS
  // Minus 2 clock cycles for the IO lines
  // Gives a minimum of approx 0.1uS
  _delay_us(0.1);

  // Pull CS, RD and WR back high
  PORTB |= CS | RD | WR;
```

```
  // Tintl + Tid = max 545 ns
  // _delay_us(1);

  // Take the sample
  return PINC;
}
```

```
/* ****************************************************************************
   Pots.c
**************************************************************************** */
#include "Global.h"
#include "SPI.h"
#include "Pots.h"

void PotsSetup(void) {
  // Setup SPI
  SPISetup();

  // Set initial values to their highest
  PotsSet(0x7F);
  PotsSet(0xFF);
}

// Set pot resistance to value of byte (0 to 127 inc.)
// The MSB gives the address, and the 7 LSB give the level
void PotsSet(Byte control) {
  Byte address = (control & 0x80) ? 0x01 : 0x00; // Get MSB
  Byte level = control & 0x7f; // 7 LSB

  // Send 9 zeroes followed by 7 data bits
  SPIWriteCommand(0x00, level, address);
}
```

```
/* ****************************************************************************
   SPI.c
**************************************************************************** */
#include "Global.h"
#include <avr/io.h>
#include <util/delay.h>

// Bit-bangs SPI
// Master -> slave communication only at this point as that's all that's
// needed for the digital pots
// Controls two slaves

#define SCLK _BV(PD5)
#define DATA _BV(PD4)
#define SS1 _BV(PD6)
#define SS2 _BV(PD7)

// Setup inputs/outputs for SPI
void SPISetup(void) {
  // Set DATA, SCLK and SS output
  DDRD |= DATA | SCLK | SS1 | SS2;

  // Output SS high (active low) and SCLK low (SPI mode 11)
  PORTD |= SS1 | SS2;
  PORTD &= ~SCLK;
}

// Write a single byte of data. Doesn't control SS.
static void SPIWriteByte(Byte data) {
  unsigned char i;
  for (i = 0; i < 8; i++) {
    // Output DATA based on MSB of data
    if (data & 0x80) {
          PORTD |= DATA;
    } else {
          PORTD &= ~DATA;
    }

    // Cycle clock
    PORTD |= SCLK;
    _delay_us(20);
    PORTD &= ~SCLK;
    _delay_us(20);

    // Bit-shift data so we look at the next MSB next
    data <<= 1;
  }
}
```

```
// Write an SPI command (consisting of control byte then data byte)
void SPIWriteCommand(Byte control, Byte data, Byte address) {
  // Pull SS low
  if (address) { PORTD &= ~SS2; }
  else { PORTD &= ~SS1; }

  SPIWriteByte(control);
  SPIWriteByte(data);

  // Pull SS back high
  if (address) { PORTD |= SS2; }
  else { PORTD |= SS1; }
}
```

# List of Corrections

# Bibliography


[1] Brian Benchoff. *Finally, TI is producing simple, cheap WiFi modules*. accessed 26/01/14. 2013. URL: http://hackaday.com/2013/01/12/finally-ti-is-producing-simple-cheap-wifi-modules/.

[2] Bitscope. *Bitscope Logic Analyzer*. accessed 26/01/14. URL: http://www.bitscope.com/software/logic/06.png.

[3] Limor Fried. *Finding Parts: Getting Samples from Manufacturers*. accessed 19/02/14. URL: http://www.ladyada.net/library/procure/samples.html.

[4] Gabotronics. *Digital oscilloscopes for hobbyists*. accessed 02/01/14. 2013. URL: http://www.gabotronics.com/resources/hobbyists-oscilloscopes.htm.

[5] jepoirrier. *Oscilloscope with noise*. accessed 18/02/14. URL: http://flic.kr/p/6uyTN1.

[6] Maxim Integrated. *MAX114/MAX118 Datasheet*. accessed 02/01/14. URL: http://datasheets.maximintegrated.com/en/ds/MAX114-MAX118.pdf.

[7] mightyohm. *New Oscilloscope*. accessed 18/02/14. URL: http://flic.kr/p/6rdh6K.

[8] mik3y. *usb-serial-for-android*. accessed 26/01/14. URL: https://github.com/mik3y/usb-serial-for-android.

[9] W. Phillips. *Using an Oscilloscope*. accessed 26/01/14. URL: http://www.doctronics.co.uk/scope.htm.

[10] Picotech. *Advanced Triggering with PicoScope*. accessed 02/01/14. 2014. URL: http://www.picotech.com/education/oscilloscopes/advanced-triggering.html.

[11] K.F. Riley, M.P. Hobson, and S.J. Bence. *Mathematical Methods for Physics and Engineering*. 2nd ed. Cambridge University Publishing, 2002.

[12] Julian Roy. *Real-time processing system for dual-comb spectroscopy*. accessed 26/01/14. 2013. URL: http://nutaq.com/en/blog/real-time-processing-system-dual-comb-spectroscopy-part-1.

[13] Semtech. *Improving the Accuracy of a Crystal Oscillator*. accessed 18/02/14. URL: www.semtech.com/images/datasheet/xo_precision_std.pdf.

[14] Claude Shannon. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27 (July 1948), pp. 379–423, 623–656. URL: http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf.

[15] Silicon Labs. *Single-chip USB to UART Bridge*. accessed 26/01/14. URL: http://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-9.pdf.

[16] Deepak Kumar Tala. *ASIC World*. accessed 01/02/14. URL: http://www.asic-world.com/verilog/index.html.

[17] TiePie Engineering. *Digital Data Acquisition*. accessed 26/01/14. URL: http://www.tiepie.com/en/classroom/Measurement_basics/Digital_Data_Acquisition.

[18] Alan Tong. *What to Look for When Choosing an Oscilloscope*. accessed 02/01/14. 2014. URL: http://www.picotech.com/applications/oscilloscope_tutorial.html.

[19] Wikipedia. *Verilog — Wikipedia, The Free Encyclopedia*. accessed 01/02/14. 2013. URL: http://en.wikipedia.org/wiki/Verilog.

[20] Joel Woodward. *What's the Difference Betwen a Mixed-Signal Oscilloscope And A Logic Analyzer?* accessed 26/01/14. 2012. URL: http://electronicdesign.com/test-amp-measurement/what-s-difference-between-mixed-signal-oscilloscope-and-logic-analyzer.