

Touch gesture recognition using body capacitance

## Project Report

Raivis Stroganovs 625826

BEng Computer Engineering

Mr. Hassan Parchizadeh

Dr. Rinat Khusainov

# 1 Table of Contents

2	Introduction.....	4
2.1	Problem description .....	4
2.2	Outcomes and goals .....	4
2.3	GD system architecture .....	5
2.4	Organization of the report .....	6
3	Background research.....	6
3.1	Capacitive touch sensing .....	6
3.1.1	Measuring charge time.....	7
3.1.2	Distributing the voltage between the object and another capacitor.....	7
3.2	Capacitive touchscreens.....	8
3.3	Current capacitive technology evaluation.....	10
3.4	Similar research .....	10
3.5	Disney touché .....	10
3.6	SmartSkin.....	11
4	Designing GD system .....	13
4.1	Hardware .....	13
4.1.1	Initial design .....	13
4.1.1.1	Initial circuit .....	14
4.1.1.2	PWM frequency sweep .....	16
4.1.1.3	Hardware prototypes .....	17
4.1.2	Function generator AD9850 .....	18
4.1.2.1	Programming AD9850 .....	18
4.1.2.2	AD9850 Library methods and usage .....	23
4.1.3	Amplification and electrode excitation stage.....	24
4.1.3.1	Thevenin Equivalent Circuit.....	24
4.1.4	Peak detector and buffer stage .....	29
4.1.5	Wireless communication – Bluetooth .....	31
4.1.5.1	HC-06 Bluetooth module .....	31
4.1.6	Main processing unit .....	33
4.1.6.1	TI ARM TMS570 .....	33
4.1.6.1.1	Problems with Texas Instruments TMS570.....	36
4.1.6.2	Atmel Xmega16e5 .....	37
4.1.6.2.1	Breakout board.....	38

4.1.6.2.2	Configuring xmega system clock .....	39
4.1.6.2.3	Configuring USART for serial communication with Bluetooth .....	41
4.1.6.2.4	Configuring ADC .....	44
4.1.6.2.5	GD systems firmware .....	45
4.1.7	PCB manufacturing and circuit design.....	47
4.1.7.1	PCB etching.....	47
4.1.7.2	Final circuit and PCB design.....	49
4.1.8	Conclusion .....	50
4.2	Software .....	50
4.2.1	Qt library.....	50
4.2.2	Support Vector Machines (SVM) .....	51
4.2.2.1	Theory.....	51
4.2.2.2	LibSVM.....	53
4.2.3	Main application.....	56
4.2.3.1	MainWindow class.....	56
4.2.3.2	Learner class .....	59
4.2.3.3	Settings class.....	59
4.2.3.4	SerialTerminal class .....	60
4.2.3.5	SimulateKeyboard class .....	61
5	Testing .....	62
6	Conclusion .....	64
	Appendices .....	66
7	Logbooks.....	110
8	Bibliography.....	122

## 2 Introduction

### 2.1 Problem description

These days most touch sensitive devices are designed either to recognize where it has been touched or whether is touched or not. Many objects around us in everyday life can be potentially used as touch interactive surface providing they are made out of conductive material. With the conventional way it would only be possible to detect whether the object has been touched or not. However, by exciting the object with different frequencies it is possible to detect how much skin is touching it. Essentially the system could recognize whether the object is grabbed, pinched, touched by one or more fingers or any other gesture which have different amount of skin touching it. There are many surfaces, objects and liquids which can be transformed into touch sensitive devices without additional buttons or touchscreens like door handles, mobile phones, lamps, desks, walls etc. With this technology we could extend the human-computer interaction with everyday objects via simple and inexpensive solution. As mentioned early only requirement of the object is it being conductive or in worst case scenario a conductive material has to be implemented in the object, since it's not mandatory to touch directly the sensor, due to its ability to sense human interaction with its magnetic field.

### 2.2 Outcomes and goals

Main goal of the project was to develop and design a device which would be able to do GD (gesture detection) on the objects connected to it. This would include designing and making the main PCB and writing software for the PC and Android smart phones. However, due to lack of time and unforeseen challenges it was infeasible.

As mentioned above one of the goals was designing and making the main PCB for learning purposes. This process includes:

1. Designing the PCB in Eagle CAD software
2. Etching the PCB in home environment
3. Applying solder mask
4. Soldering

For the software side of the project, the main goals were as following:

1. Design PC side application in C++/Qt
2. Learn LibSVM and write a wrapper to simplify user interactions with it
3. Port the application mentioned in 1<sup>st</sup> point for smart phones

Most of the goals were successfully achieved, except for increasing the device sensitivity. One of the requirements was to develop a sensor which can be interfaced with human skin directly and used as touch interface. Unfortunately, the current design wasn't able to excite the human body in different frequencies enough to detect change from the input signal. On the other hand, the GD system is able to sense the presence of human body and differentiate from multiple gestures when touching a conductive object, including liquids. As of now GD system could be used for detecting gestures on objects only, with some

modification it can be potentially used as smart light switch, where different gestures enable different lighting in the room or by manufacturing a copper frame in a smart phone, the GD system could sense how is the phone being held in hand.

### 2.3 GD system architecture

Standard way of doing capacitive touch sensing is by using one fixed frequency. However GD system is using a technique called frequency sweep, which enables to do capacitive touch sensing in multiple frequencies. For example, in frequencies ranging from 100 kHz up to 50 Mhz.

For frequency sweep the micro-processor will generate AC signal in specific frequency range to excite the electrode. The electrode can be any surface connected to the ADC of the micro-processor. The micro-processor measures the amplitude of the specific frequency the electrode was excited. Essentially, constructing spectrum for different levels of excitement. By varying the capacitance the spectrum should in theory change. A human touch basically changes, in almost all cases increases, capacitance. The way human is touching the electrode, or the amount of skin contacted with the electrode will produce a different output. In other words, electrical properties of human touch will change the excitement spectrum. Then by using simple machine learning algorithms like SVM or neural network, for pattern recognition, the way of human touching the electrode can be detected. In short, with frequency sweep technique it is not only possible to detect a touch, but as well how it was touched. Interestingly this is all achieved through single electrode. Please see Figure A-1 for the basic architecture of the touch sensing technique

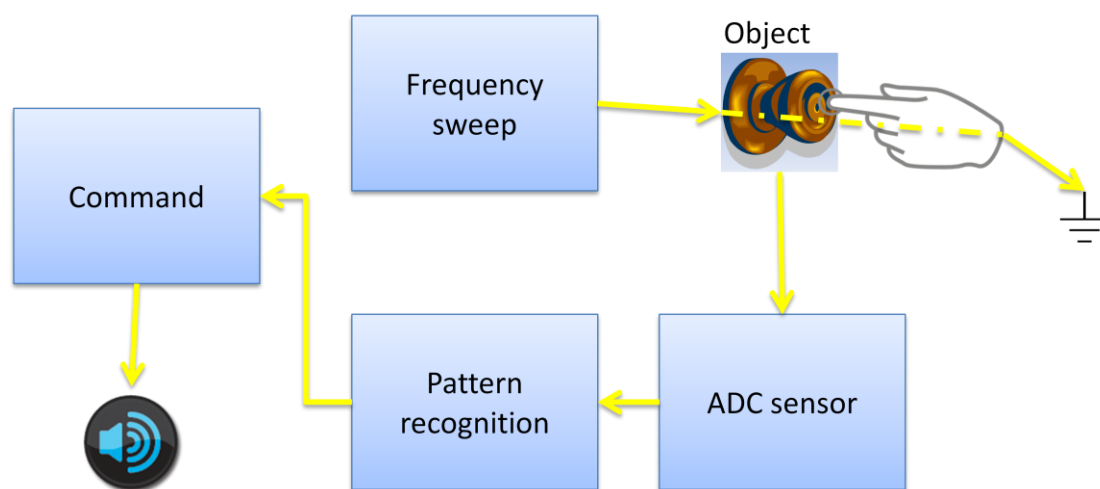


Figure A-1 Touch sensing architecture

The technique of frequency sweep has already been used for various medical applications. For example, already in year 1979 it was proposed to be used for analysis of neuromuscular junction continuity<sup>1</sup>. As well it has been used for wireless communication and different kinds of proximity sensors.

<sup>1</sup> <http://informahealthcare.com/doi/abs/10.3109/03091907909160663>

The project is highly inspired by Disney's research called touché<sup>2</sup>. More or less they are using exactly the same approach GD system is using.

There are two more articles (projects) which are using frequency sweep technique to do touch recognition. Both of them have been highly inspired by touché and they constructed much cheaper version of the touché. Sprites<sup>3</sup> method involves using a standard AVR micro-controller to do frequency sweep from 0-3.5MHz. For recognition, just like touché Sprite is using SVM algorithm. And the other one by DZL<sup>4</sup> reconstructed touché using arduino, however for gesture recognition he is measuring the highest point of the frequency sweep. Which is very unreliable compared to pattern recognition using machine learning.

Even though both of the "cheap" implementations are using PWM for exciting the electrode, the GD system is using a proper sine wave generator, just like touché. More precisely it uses AD9850 function generator. This unit can generate up to 50MHz pure sine wave. The decision was made to use a proper sine wave generator because of the electrical properties of the human body. More than 99% of the human body's resistance is at the skin on average which is about  $1M\Omega$ <sup>5</sup>. On the other hand AC signal can pass through the human skin and go through the least impedance in the corresponding phase and amplitude of AC signal. Internal human resistance is around  $100\Omega$ .

## 2.4 Organization of the report

The report is organized is organized by first introducing with relevant theory and afterwards with small incremental steps the actual process is explained. Some details are left out because at the time of writing they seemed obvious or too basic. Almost all chapters and sub chapters are supported by diagrams to better grasp the underlying theory and practical accomplishments.

## 3 Background research

There has been tremendous research done on interacting with computers in different ways, like developing tactile displays, infrared multi-touch surfaces, hand gesture recognition via camera, infrared white-board touch detection, acoustic touch sensing etc.

However, the most related touch detection technology to this project is capacitive touch sensing. Mainly due to being relatively cheap technology for basic touch detection, excluding capacitive touch screens.

### 3.1 Capacitive touch sensing

Mainly there are two approaches used when building a capacitive touch sensing device:

1. Measuring charge time
2. Distributing the voltage between the object and another capacitor

---

<sup>2</sup> <http://www.disneyresearch.com/wp-content/uploads/touchechi2012.pdf>

<sup>3</sup> <http://spritesmods.com/?art=engarde&page=1>

<sup>4</sup> <http://dzlsevilgeniuslair.blogspot.se/2012/05/arduino-do-touche-dance.html>

<sup>5</sup> <http://www.meo.etc.upt.ro/materii/cursuri/IBM/1.pdf>

### 3.1.1 Measuring charge time

This is the simplest and most intuitive approach to use as capacitive touch sensing technology. Basically, it consists of the object being connected between two pins to a micro-controller see Figure A-1. By enabling pin 1 we set the timer to run and wait until pin 2 read logic 1, at that point the timer should be stopped. And we have ourselves the charge time. Depending whether the object is being touched or not the charge time will change significantly.

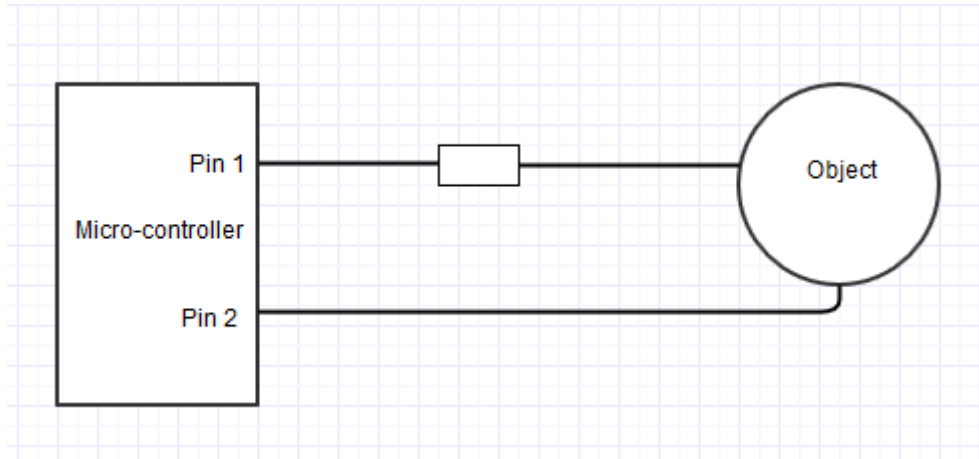


Figure A-1 Capacitive touch sensing by measuring charge time

This circuit is just a simple RC charging circuit, which can be described as following:

$$\tau = RC = \frac{1}{2\pi f_c}$$

Where R is the resistor attached to object and C is the objects capacitance.  $f_c$  is the cutoff frequency. However for simple capacitive detection, the system engineer only needs to be aware of the R variable and measure the  $\tau$ . Obviously from these two parameters it is possible to calculate the objects capacitance. However, just by measuring the charge time it is possible to determine if the object is touched or not, no need for calculating the capacitance, since it is possible to calculate what would be the charge time of the object not being touched, and if the charge time changes by significant amount a touch has been detected.

### 3.1.2 Distributing the voltage between the object and another capacitor

This is slightly more complicated way to detect a touch using capacitance. However, compared to the other method, the touch can be detected using only one pin from the micro-controller. It consists of charging the object or the capacitive button (C1), while discharging the capacitor which holds the sample for ADC to convert (C2), and then connecting those two capacitors in parallel for the charge to be distributed between them. If the capacitance of C1 is equal to C2 the resulting voltage should be half of the reference voltage, in most micro-controller design it would 2.5 V, since they are using 5V as reference. If C1 is less than C2 it's the resulting voltage would be more than 2.5 V and if C1 has higher capacitance than C2 its resulting voltage would be more than 2.5V accordingly. Ideally, when

a finger is placed on the touch sensitive button, the resulting C1 capacitance should become higher than when idle and increasing the resulting voltage correspondingly. See Figure A-1 and Figure A-2 for visualizing the process described above.

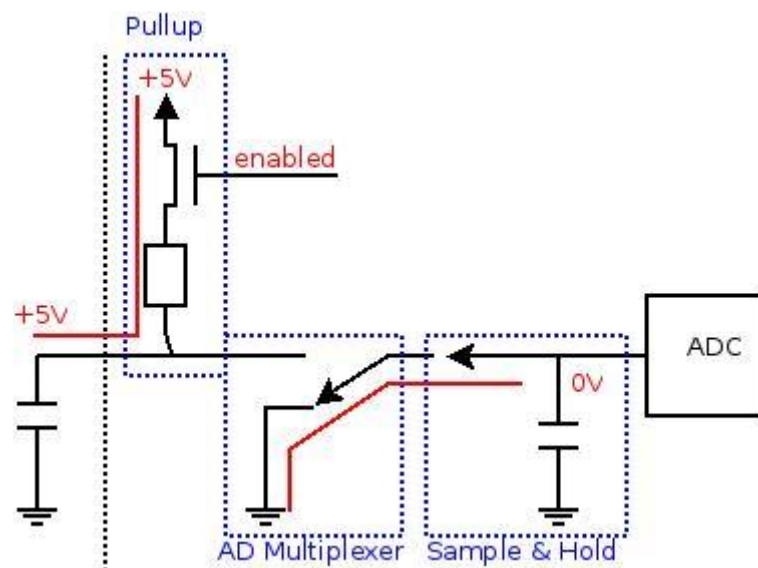


Figure A-1 Step 1: Discharge the ADC sample holding capacitor (C2) and charge the objects or buttons capacitor (C1)

Source: <http://tuomasnylund.fi/drupal6/content/capacitive-touch-sensing-avr-and-single-adc-pin>

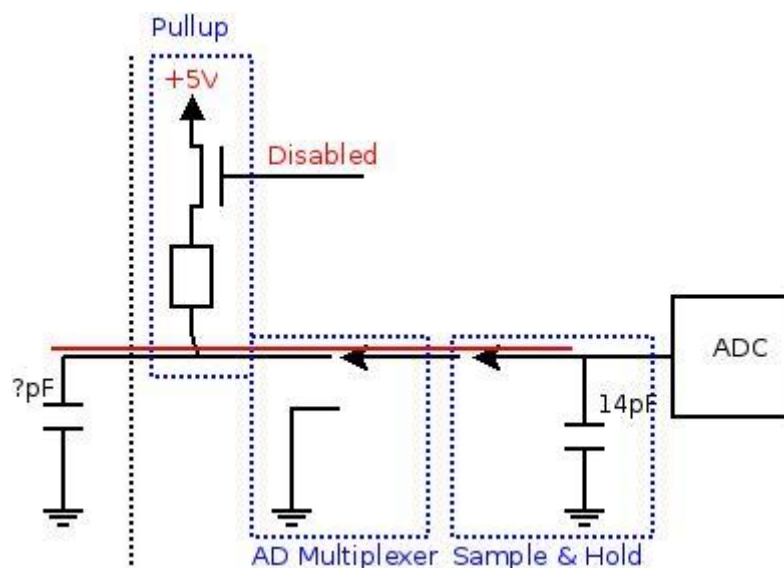


Figure A-2 Step 2: Connect C1 and C2 to distribute charge between them and initiate ADC conversion

Source: <http://tuomasnylund.fi/drupal6/content/capacitive-touch-sensing-avr-and-single-adc-pin>

## 3.2 Capacitive touchscreens

Nowadays most commonly used capacitive touch sensing technologies are touchscreens. Basically there are two types of capacitive touchscreens – surface capacitive sensing and projected capacitive sensing. In our everyday smart phones you'll probably find a projected capacitive sensing technology, where the screen consists of a grid for X and Y. By moving a conductive material over the screen, in most cases a finger, it interacts with the magnetic field of the screen and it is possible to calculate where in the grid the most interaction was



detected (See Figure A-1). Essentially this technology can construct 2D image of the touch event, and it is able to detect multiple touches simultaneously. We can imagine that projected capacitive touchscreen is a matrix of thousands of small buttons.

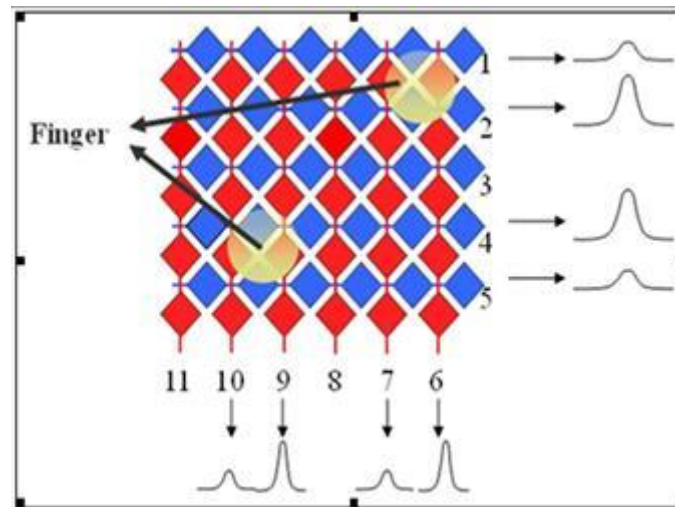


Figure A-1 Projected capacitive touchscreen, with fingers placed on the grid

Source:

[http://www.digitimes.com/supply\\_chain\\_window/story.asp?datepublish=2010/11/29&pages=PR&seq=202](http://www.digitimes.com/supply_chain_window/story.asp?datepublish=2010/11/29&pages=PR&seq=202)

Before multi-touch capacitive touchscreens, there was a simpler version of the technology where, a small amount of voltage is applied to every corner of the screen (see Figure A-2) and when the user touches the screen a small amount of current is consumed by the human body. By measuring the ratio between the current consumption from each corner it is possible to calculate where the finger was placed. The closer the finger to corresponding corner the more current is consumed accordingly. Note: This technology supports single touch.

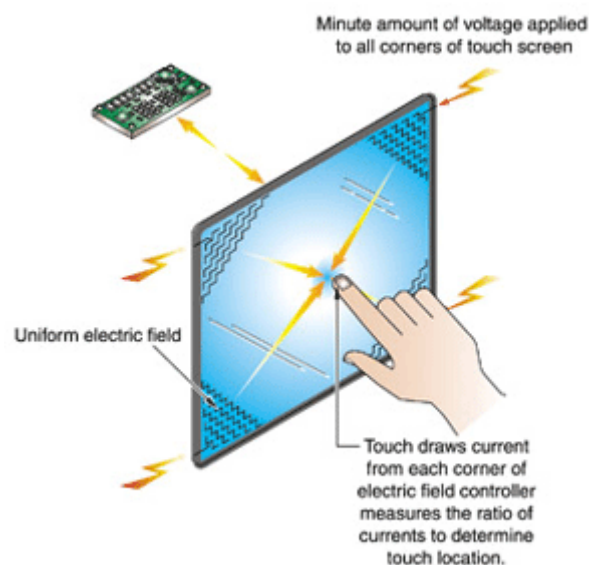


Figure A-2 Surface capacitive touch sensing

Source: [http://www.electrotest.com.sg/cap\\_touch.htm](http://www.electrotest.com.sg/cap_touch.htm)

### 3.3 Current capacitive technology evaluation

For their corresponding purposes both technologies mentioned above are quite reliable. However, each technology has its own limitations as well. For example, a capacitive touchscreen is quite expensive due to construction of a capacitive grid. On the other hand, standard capacitive touch sensing can only detect whether a touch event has occurred or not. It is not able to distinguish between multiple touches or gestures. See Table 3-1 for comparison between GD system and two approaches mentioned above.

Feature	Capacitive touch sensing button	Capacitive touchscreen	GD system
Multiple touches	No	Yes, with ability to calculate position	Yes, cannot calculate position
Responsiveness	Medium	High	Low, limited to ADC speed
Cost	Very cheap	Expensive	Cheap
Easiness of implementation	Very easy, needs one or two components, low processing power required	Hard, needs high processing power, due to analyzing data as 2D image of the touch event, also difficult to manufacture due to multiple microscopic layers used in construction	Medium. Requires only conductive object and couple small components.
Range of application	High, can be used and applied everywhere. The conductive material can be bent and transformed as necessary. Any conductive material can be used as sensor	Low, mainly limited to flat surfaces, constructing a flexible grid makes the technology even more expensive	High, just like with capacitive touch sensing, relatively anything conductive can be used as a sensor.

Table 3-1 Comparison between commonly used capacitive touch technologies and GD system

The main limitation of advanced touch sensing technology is that it can't be easily shaped in any form necessary. Most of such technologies are limited to rectangular and flat shapes. However, GD system has the ability to be interfaced with an object in basically any shape, like a door handle or a glass cup etc.

### 3.4 Similar research

### 3.5 Disney touché

Over the years there have been a lot of similar research and projects regarding the problems mentioned above. This project itself is highly inspired by one in particular – touché. They have essentially done exactly what this project intended to do, detecting different touch

gestures using frequency sweep. In other words, they were extending the capabilities of standard capacitive touch button. Initially this project was intended to extend the original capabilities of the touché project; however, due to unforeseen difficulties this project was only able to reproduce some of the features of the touché. There are some notable differences, like choices of programming languages. The GD system application can be potentially ported to any architecture and OS with little or no changes at all to the code – Linux, Windows, Mac, Android, iOS. Also this project has been designed to use less expensive components. They were using ARM microprocessor, which are generally more expensive than *xmega* micro-controllers. Only reason ARM micro-processor was considered while designing GD system was the potential of recognizing the gesture on the microprocessor itself. Neither of us accomplished that task.

However, touché accomplished higher sensitivity of the device, than GD system did. Probably due to more accurate ADC, higher current applied to object and better designed filtering system. While design GD system, those were the main limiting factors and challenges when working on this project.

### 3.6 SmartSkin

SmartSkin is a touch sensing technology very similar to the capacitive touchscreens. It also uses a grid for detecting touch in 2D. The main difference between projected capacitive touchscreens and SmartSkin is that they are exciting the grid's Y axis and reads the result in its X axis. The X axis itself acts as antenna. When a human interacts with the magnetic field, it draws some power away from the original signal, essentially reducing the amplitude of the sine wave. By multiplexing through each column and reading the resulting sine wave amplitude in each of the X axis sensors, they are able to construct a 2D image of the magnetic fields interaction with the human (See Figure A-1 and Figure A-2). Afterwards it's just using simple computer vision algorithms for detecting finger tip touches and determining their positions.

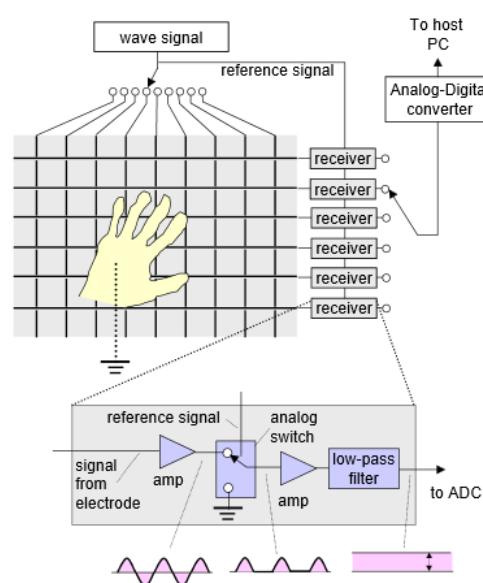


Figure A-1 SmartSkin sensor mesh for detecting touch

Source: <https://vs.inf.ethz.ch/edu/SS2005/DS/papers/surfaces/rekimoto-smartskin.pdf>

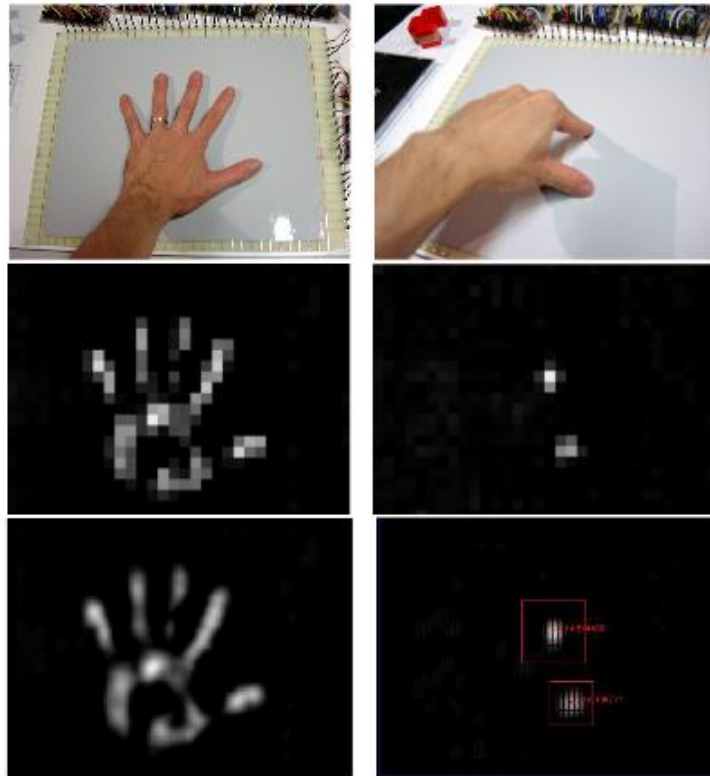


Figure A-2 Output from the SmartSkin mesh

Source: <https://vs.inf.ethz.ch/edu/SS2005/DS/papers/surfaces/rekimoto-smartskin.pdf>

This technology might seem very different from GD system. It is quite interesting, because SmartSkin is using similar technique to excite the grid. Even though, instead of frequency sweep they are exciting in one fixed frequency, they are basically doing the same measurements as the GD system. Also GD system with very little tweaking could be adapted to this technology. Only addition necessary is a multiplexer to select column to excite with sine wave and multiplexer for receiving at the X axis.

Due to modular design of the GD device an interface module for such technology can be designed and constructed without altering the original system. However, it would involve writing completely separate PC side application for analyzing the data, since they are very different from what the GD system is detecting and sending to the software.

## 4 Designing GD system

Mainly designing the GD system consists of two distinctive parts – hardware design and software design. The firmware for the micro-controllers will be considered part of hardware design in this report. The GD system was being designed while taking into account system scalability, portability and ease of use. It should be rather easy to add more modules to the system than it has as of now.

### 4.1 Hardware

#### 4.1.1 Initial design

At the very beginning, GD system was prototyped with Arduino, with very basic frequency sweep circuitry. Where instead of using proper sine wave function generator, a PWM was used. Main draw backs of using the PWM was that, the DC signals can only propagate through the skin. Due to this reason, the GD system sensitivity is reduced significantly.

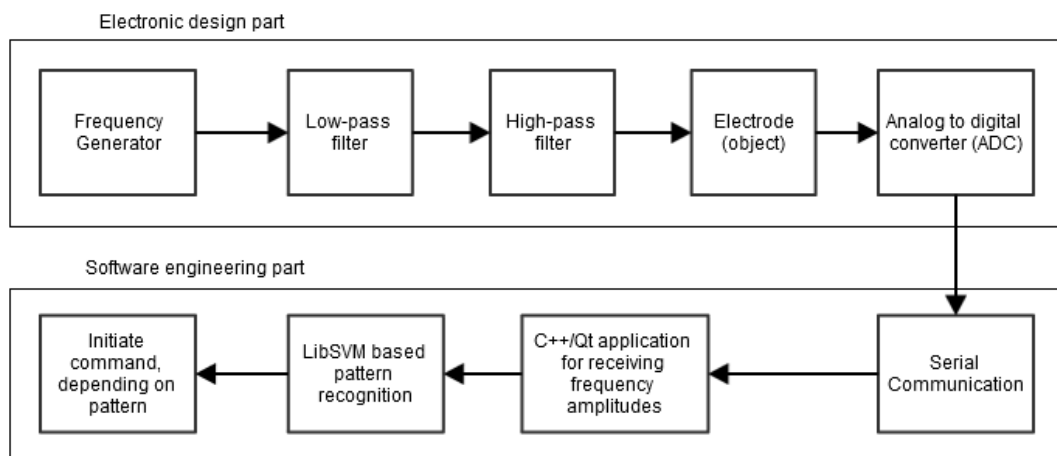


Figure A-1 Overall design of the initial GD system

In Figure A-1 we can see that as previously mentioned, the task project was divided in to two parts. However, the initial design will be only about the hardware, because the PC interface changed very slightly throughout the project. And the basic communications principles between the initial design and current software are essentially the same.

#### 4.1.1.1 Initial circuit

See Figure A-1 for initial circuit diagram using Arduino.

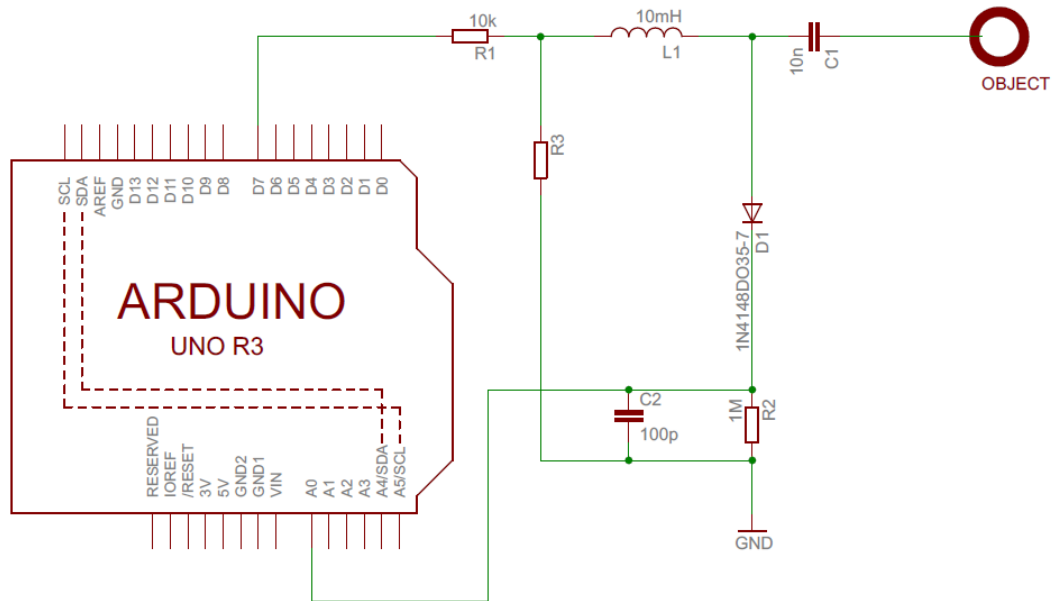


Figure A-1 Initial design circuit diagram

The circuit basically consists of two parts, LC circuit for filtering out unwanted frequency components from the square wave, essentially outputting sine wave shaped signal and bare bone envelope detector with discharging resistor.

The envelope detector's (R2, D1, and C2) cut off frequency can be calculated the same way as for low-pass filter via:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi * 100 * 10^{-12} * 1 * 10^6} = 1.59 \text{ kHz}$$

Note that envelope detector here is used as peak detector. It doesn't need to follow the carrier signals envelope; the output should be the highest amplitude of the signal.

From the calculations we can assume that all frequencies above 1.59 kHz should produce the peak value, because higher frequency shouldn't let the capacitor C2 discharge. Also, it means that the frequency sweep between two frequencies have to change slower than 1.59 kHz, to give enough time for the envelope detector to settle. Because in the circuit a diode is used, more specifically 1N4148, which is a standard signal diode, 0.6 V will be lost from the original signal, due to characteristics of the diode. Due to this cut-off frequency, the frequency sweep has to start above 1.59 kHz, initially GD system started frequency sweep at 2 kHz.

We can calculate the resonant frequency for the LC (C1 and L1 in circuit diagram) circuit using the following equation:

$$f = \frac{1}{2\pi\sqrt{LC}} = \frac{1}{2\pi\sqrt{10 * 10^{-3} * 10 * 10^{-9}}} = 15.915 \text{ kHz}$$

By assuming that electrodes or objects capacitance is very small, the highest amplitude from the input signal will be around 16 kHz. When a human touches the object, or interacts with its magnetic field the resonant frequency will change, and produce the highest amplitude in different frequency.

For example if human touch added 20nF of capacitance to the LC circuit, then the resonant frequency should be around:

$$f = \frac{1}{2\pi * \sqrt{10 * 10^{-3} * 12 * 10^{-9}}} = 14.529kHz$$

See Figure A-2 and Figure A-3 how resonant frequency is changed when interacting with the sensor. The change might be small, but it can be clearly visible that the resonant frequency changed from around 55<sup>th</sup> frequency reading to around 59<sup>th</sup> frequency.

Then either by detecting where the peak is located or, just applying machine learning, we can distinguish the gesture.

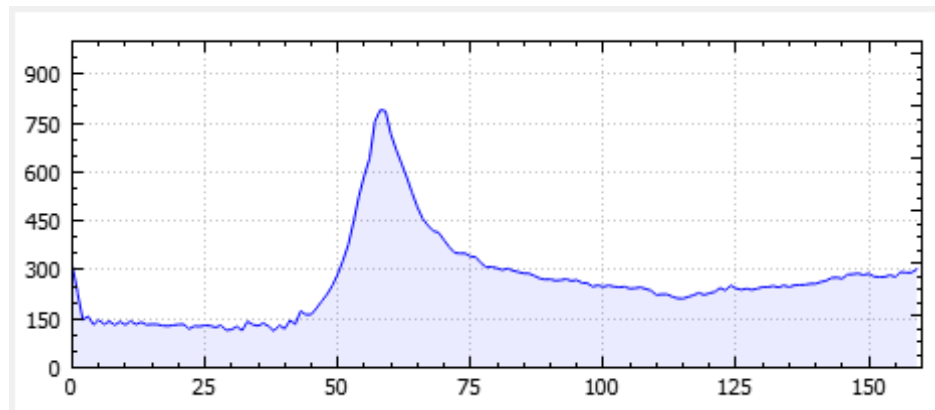


Figure A-2 Amplitude (in raw ADC format) vs frequency (stored in array from 2kHz to 3.5 MHz). Hand near by the sensor

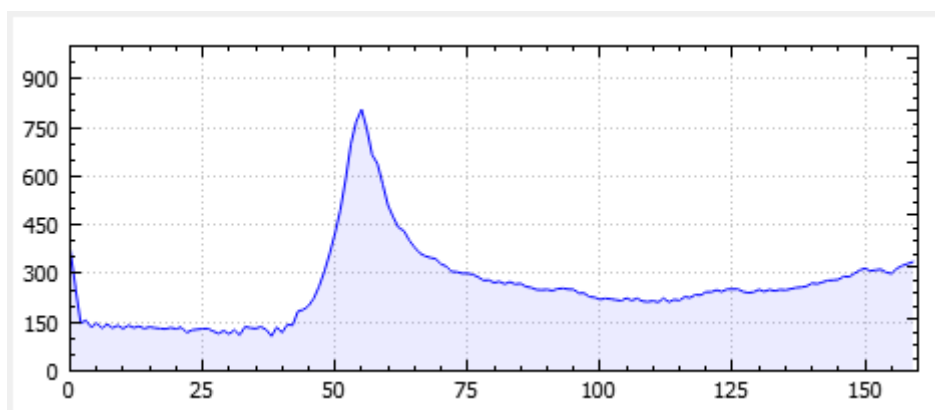


Figure A-3 Amplitude (in raw ADC format) vs frequency (stored in array from 2kHz to 3.5 MHz). Idle sensor, no human interaction

#### 4.1.1.2 *PWM frequency sweep*

The AVR Atmel micro-controllers conveniently are able to produce a PWM signal from 0 to 3.5 Mhz. The touché system used a frequency sweep from 0-3.5 Mhz.

In the initial implementation of the frequency sweep consists of 160 frequencies. Ranging from 0 Hz up to 16Mhz with 100kHz step.

This is how we can set up the wave generator in AVR

```
TCCR1A=0b10000010;    //-Set up frequency generator
```

```
TCCR1B=0b00011001;    //-+
```

Basically it is set in Fast PWM mode, with TOP being ICR1A register

And compare mode is set to:

Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at

BOTTOM, (non-inverting mode)

And clock selected without pre-scaler.

For AVR it means that the timer works with FCPU frequency, which in Arduino case is 16 MHz. Also it is taking 160 different frequencies. That means  $16\text{MHz} / 160 = 100\text{ kHz}$  as one step frequency. Hence, the device is sweeping from 0Hz up to 16MHz

While doing experiments with the touch sensitive surface interesting results were produced. Considerable change could be detected when an electrode was attached to the water, which was in a glass container. Immediately the resulting output signal was much more stable coming from the glass itself, instead of the liquid. The insulating material (glass) helped to make the finger and the water to act as a proper capacitor. Due to this “discovery” the normal electrode was tested by insulating with mylar also known as BoPET polyester film, which has high dielectric properties and is commonly used to make foil capacitors. Just like expected the reliability of the system increased significantly.

See AppendixE for initial designs source code.



#### 4.1.1.3 Hardware prototypes

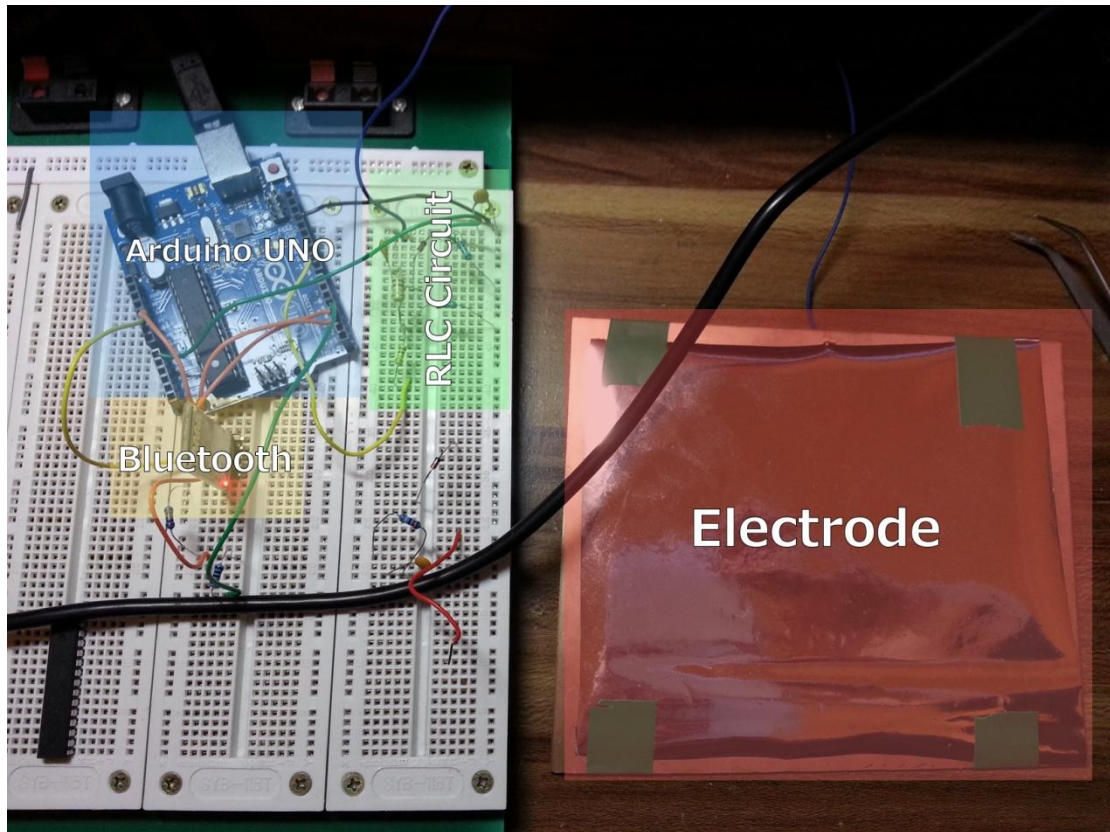


Figure A-1 Early prototype assembled in breadboard

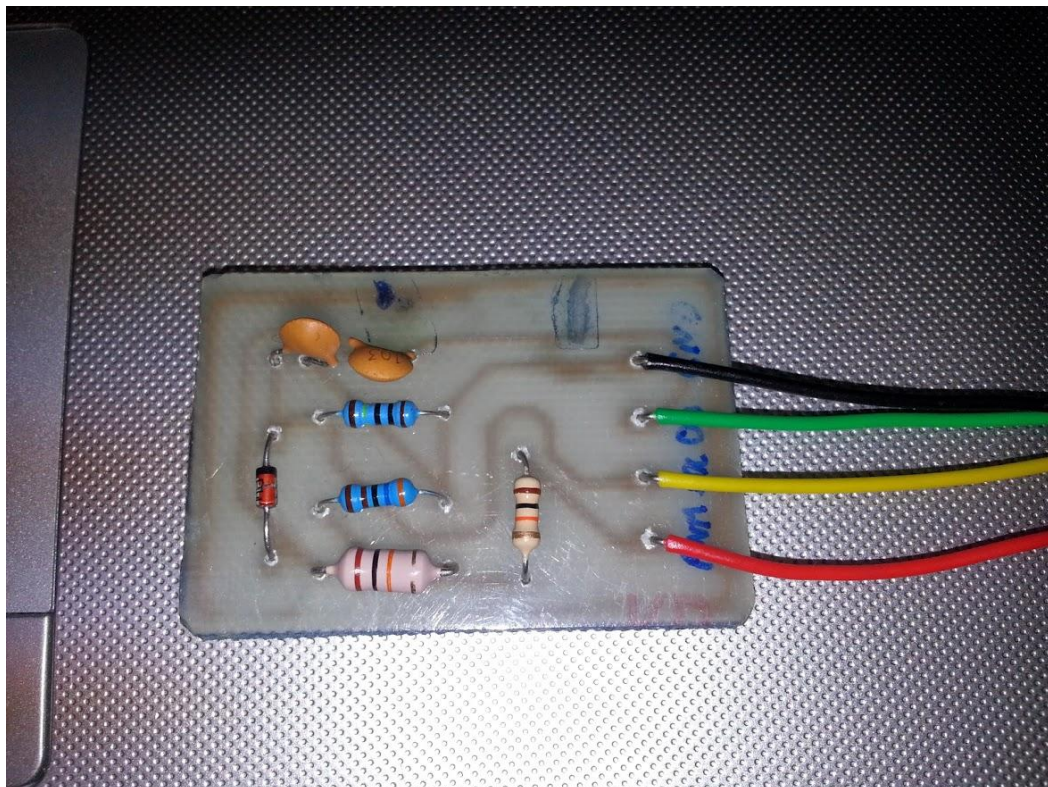


Figure A-2 the breadboard prototype transferred to a PCB

#### 4.1.2 Function generator AD9850

After successful initial prototype, the following task was to interface a sine wave generator to increase the sensitivity of the device. As mentioned in the introduction, by using AC signal instead of DC the sensor's signal can propagate through internal fluids in human body, increasing the sensitivity by extending the range of frequencies responding to human interactions.

For the AC generator any pure function generator IC should do the task reasonably well. However, it was decided to use AD9850 function generator, due to its low cost and development breakout board availability see Figure A-1. AD9850 is able to produce a pure sine wave up to 40 MHz.



Figure A-1 AD9850 breakout board

Source: <http://tpatphoto.com/images/tpat/BI/TPAT-BI00076.jpg>

Obviously there are alternatives for function generators. The IC chosen in this project is considered to be “old” and successor has already been developed - AD5932. Essentially the new IC does the same job, as the old one. However the “old” AD9850 was chosen because there is more documentation available online. Moreover, it was easier to buy a development board for the AD9850, than for AD5932.

The operation of the AD9850 is rather simple; with a similar protocol to SPI the micro-controller sends the desired sine wave. More precisely, the function generator waits for micro-controller to send in 40 bits the frequency of the wave and its phase. Afterwards, it will produce 4 outputs – low jitter square wave, sine wave and both of the waves inverted. Square wave will be with amplitude of 5V and the sine wave with amplitude of 1V. The amplitude cannot be changed. That has to be done by an amplifier. Note that the sine wave is not AC signal, it will fluctuate between 0 and 1V instead of -0.5 to 0.5. So when designing the amplification stage an offset was introduced to the opamp.

##### 4.1.2.1 Programming AD9850

The AD9850 contains 40 bit register where 32 bits are for controlling the frequency, 5 bits controlling the phase and 2 bits for power down function. The function generator can be booted in two modes in parallel mode or serial mode. In this project to reduce the number of pins used and decrease the complexity of the PCB, serial mode was chosen.

To enable serial mode pins D0 and D1 has to be connected to logic 1 and D2 to logic 0 (see Figure A-1 ). Then through a SPI like protocol the data is sent through pins W\_CLK, FQ\_UD and DATA. The functions of the bits in the data are shown in the Table 4-1.

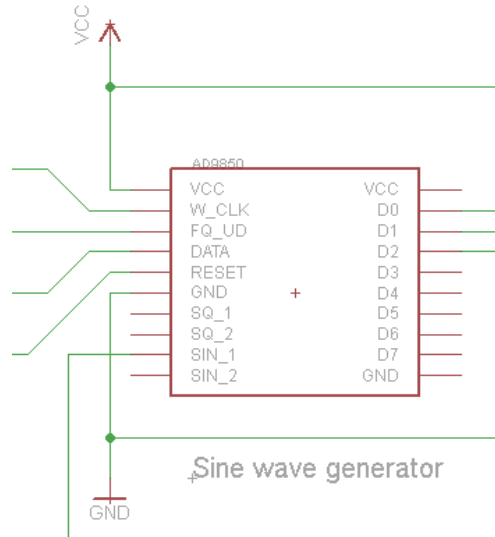


Figure A-1 AD9850 configured in serial mode

W0	Freq-b0 (LSB)	W14	Freq-b14	W28	Freq-b28
W1	Freq-b1	W15	Freq-b15	W29	Freq-b29
W2	Freq-b2	W16	Freq-b16	W30	Freq-b30
W3	Freq-b3	W17	Freq-b17	W31	Freq-b31 (MSB)
W4	Freq-b4	W18	Freq-b18	W32	Control
W5	Freq-b5	W19	Freq-b19	W33	Control
W6	Freq-b6	W20	Freq-b20	W34	Power-Down
W7	Freq-b7	W21	Freq-b21	W35	Phase-b0 (LSB)
W8	Freq-b8	W22	Freq-b22	W36	Phase-b1
W9	Freq-b9	W23	Freq-b23	W37	Phase-b2
W10	Freq-b10	W24	Freq-b24	W38	Phase-b3
W11	Freq-b11	W25	Freq-b25	W39	Phase-b4 (MSB)
W12	Freq-b12	W26	Freq-b26		
W13	Freq-b13	W27	Freq-b27		

Table 4-1 Serial load bit function assignments<sup>6</sup>

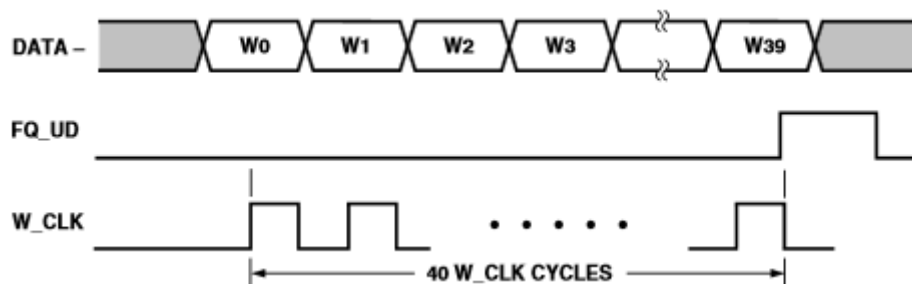


Figure A-2 Serial load frequency/phase updating sequence<sup>6</sup>

According to the datasheet, the Table 4-1 and Figure A-2, the data is sent starting from least significant bit (LSB) of the frequency and finishing with the most significant (MSB) bit of the

<sup>6</sup> Table/figure is an extract from AD9850 datasheet: [http://www.analog.com/static/imported-files/data\\_sheets/AD9850.pdf](http://www.analog.com/static/imported-files/data_sheets/AD9850.pdf)

phase. Also when all 40 bits have been sent, FQ\_UD is triggered so the AD9850 stores the received data in its register responsible for generating the output.

Corresponding to the datasheet (page 8) the output frequency is determined by the following equation:

$$f_{out} = \frac{\Delta phase * CLKIN}{2^{32}}$$

Where  $\Delta phase$  is the value of the 32 bit frequency word, confusing because one would expect “phase” to be actually the phase of the sine wave, which it isn’t;  
CLKIN is the input reference frequency, in case of the development board used in the project it is 125 MHz;

$f_{out}$  is the output frequency in MHz;

When designing a library to interface the AD9850, obviously the equation was transformed that the input is the desired frequency ( $f_{out}$ ) and the output is the 32 bit value ( $\Delta phase$ )

$$\Delta phase = \frac{f_{out} * 125 * 10^6}{2^{32}}$$

So if a desired frequency was 2 kHz, then  $\Delta phase$  which would be sent to the AD9850 would be calculated like this:

$$\Delta phase = \frac{2000 * 125 * 10^6}{2^{32}} = 68700$$

However, the actual phase for the generated wave is stored in 5 bits, with an increment of 11.25°. In the library, the phase 5 bit value is calculated as following:

$$phase\ 5\ bit\ integer = \left\lfloor \frac{phase\ in\ degrees}{11.25} \right\rfloor$$

By following the information and calculations above two libraries were constructed - one for Arduino and one for Xmega16e5. Due to their similar architectures, the libraries are essentially the same. However, Arduino provides a very convenient way of addressing pins via their own numbering system, while pure AVR C doesn’t support such arguments to be passed to the constructor. So in xmega library in the *AD9850.h* the engineer must provide the PORT group. In the case of the project it was PORTC. See Figure A-3 for flowchart implementing above instructions.

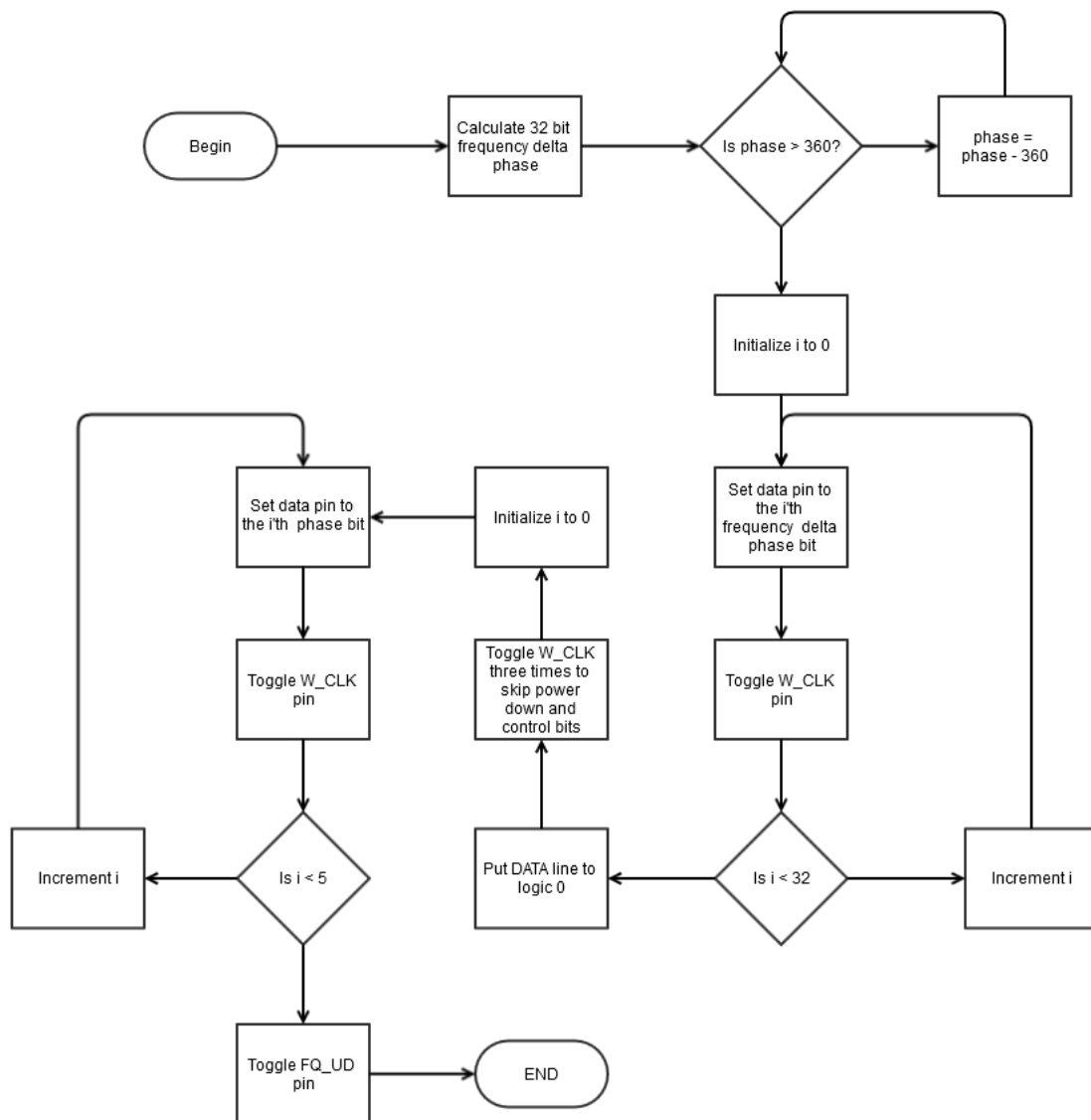


Figure A-3 Flowchart for function which sends the desired frequency and phase to AD9850

To further optimize the speed of communication later it was decided that phase won't be necessary in the case of this project and also to reduce the computation time the frequency can be sent as calculated 32 bit integer. More or less the flowchart for the additional function is the same; it skips the phase calculation and frequency calculation and sends the frequency integer which was provided as an argument immediately.

The initial frequency integer can be calculated only once at the very beginning when the micro-controller is initializing. In addition, the necessary incremental step can be calculated as an integer as well only once at the initialization stage, further reducing computation time. It won't be as intuitive to set the frequency though. See Figure A-4 for function generator library testing.



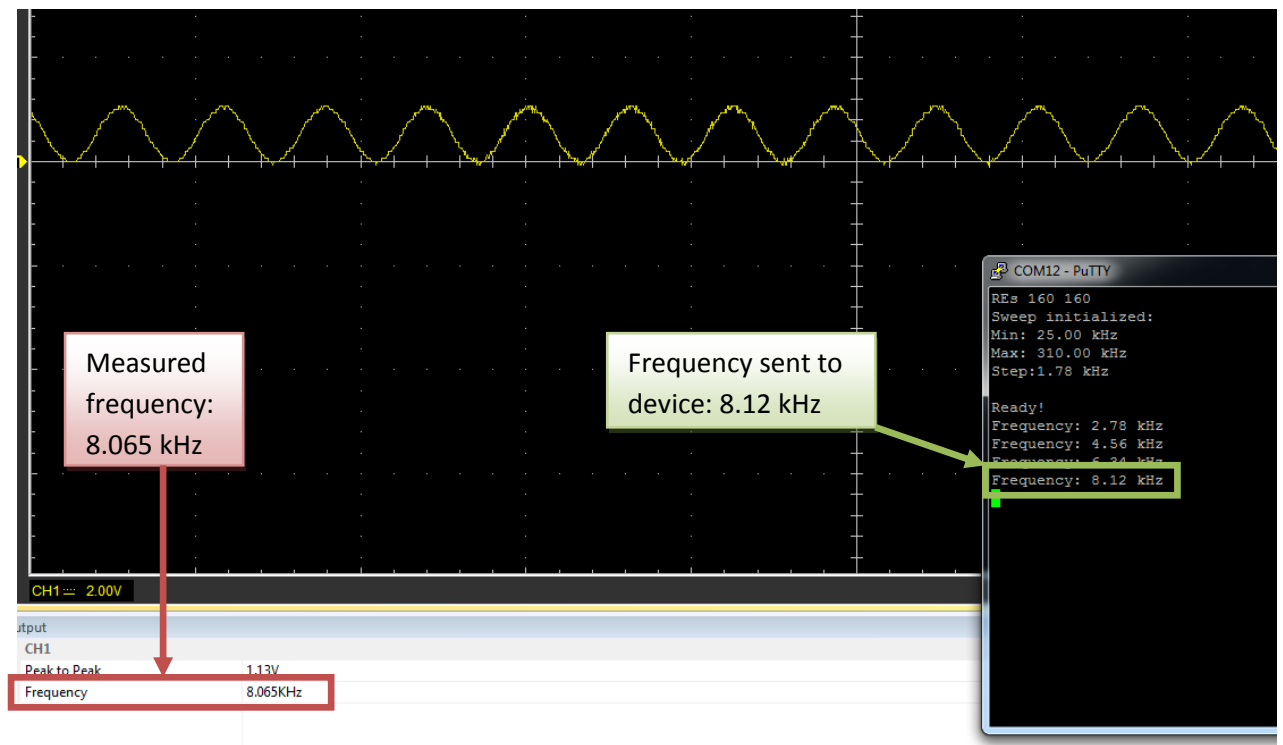


Figure A-4 Function generator output measure by oscilloscope and putty sending which frequency to set. As you can see the measured frequency differs by 0.06 kHz. It could be number of things, but most likely it's the cheap USB oscilloscope - Hantek 6022BE

One of the problems encountered while interfacing the AD9850 was noisy input pins (see Figure A-5). The AD9850 is able to read the pins in very high speeds (according to datasheet the maximum clock speed can be around 285 MHz) and the slightest noise on the channel can be read as an input. Apparently the noise was coming from unwanted and accidental resonant circuit. To fix it on every input pin small resistor was introduced, in case of this project 75Ω (See Figure A-6).

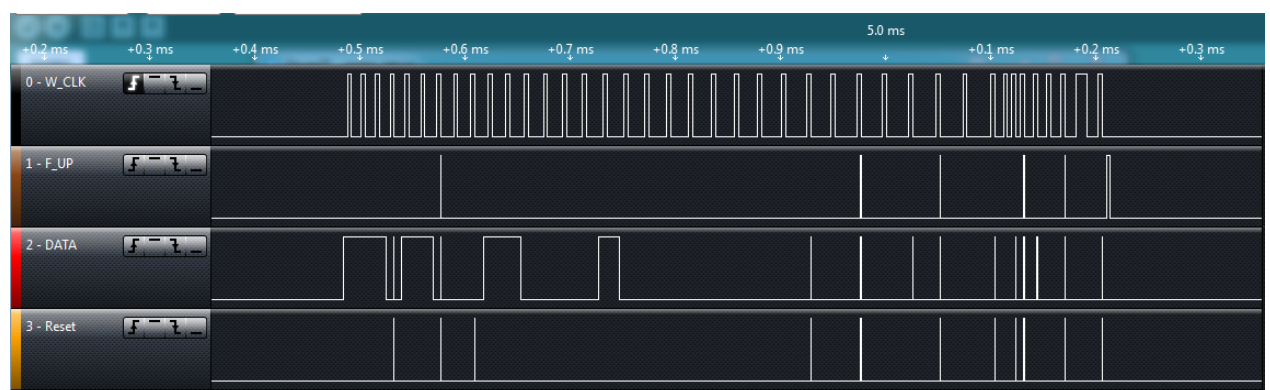


Figure A-5 Measured input from the micro-controller to the AD9850 without the 75Ω resistors. Note the very narrow lines are the noise on the channel.

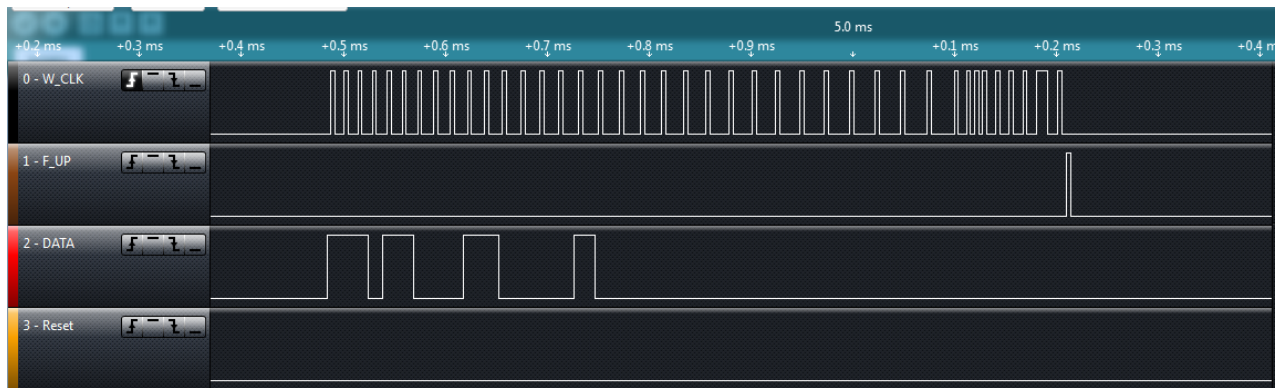


Figure A-6 Transmission between micro-controller and AD9850 with the 75Ω resistors on the line. As you can see there is no noise at all

#### 4.1.2.2 AD9850 Library methods and usage

As briefly mentioned before, in Arduino library the engineer in the constructor (the library is written as a Class in C++ instead of C) just needs to specify which corresponding pin is assigned to the function generator. However, the xmega's library has additional configuration properties in the *AD9850.h* file to set the PORT group. Other than that, the constructor also accepts pin numbers, but associated to the PORT group.

In both devices you would construct the object as following:

```
AD9850 funcGenerator(W_CLK, FQ_UD, DATA, RESET);
```

The next two steps are as following to initialize the function generator:

```
funcGenerator.init(); //Sets all pins to logic 0
funcGenerator.doReset(); // will do reset sequence and set output to 0Hz
```

To change the frequency of the function generator, the engineer can choose to execute one of these methods:

1. `funcGenerator.osc(2000,0);` // Sets AD9850 to generate 2 kHz wave with 0 phase. Arguments are as following (frequency in Hz, phase in degrees)
2. `funcGenerator.oscInt(34359);` // Sets AD9850 to generate 1 kHz wave with 0 phase. Only argument is the integer corresponding which is acceptable to AD9850

The library which was ported for xmega, with little or even no modification can be used for all Atmel AVT mega series (note, there is no X) micro-controllers. Due their similiarity in syntax.

#### 4.1.3 Amplification and electrode excitation stage

Since sine wave signal generated by AD9850 doesn't come as AC and has amplitude of 1V, the next step was to design simple amplification circuit. TI OPA830 was chosen due to its high-speed and high ratings by other engineers.

OPA830 is classified as low-power, single power supply, voltage feedback amplifier. It can operate on single power supply between +3V or +5V and if necessary it can be configured to operate on  $\pm 5V$ . Bandwidth of the opamp is 250 MHz with a slew rate of 550V/ $\mu s$ . Even though the final frequencies used are only in range from 0-300kHz, the OPA830 allows the GD system to be expanded further and adapted for different environments.

Initially, the minimum slew rate was calculated necessary for frequency sweep from 0 to 3.5 MHz and with peak to peak of 5V:

$$slew\ rate = \frac{2\pi fV}{10^6} = \frac{2\pi * 3.5 * 10^6 * 5}{10^6} = 109.95\ V/\mu s$$

So it means that for the GD system to be able to work in the desired frequency range an opamp with minimum slew rate of 109.95 V/ $\mu s$  is needed. As previously mentioned, much faster opamp was chosen, whose maximum frequency can be calculated as following:

$$f = \frac{slew\ rate * 10^6}{2\pi V} = 17.5\ MHz$$

Basically the GD system with the current design can potentially do a frequency sweep up to 17.5 MHz with signal peak to peak of 5V.

##### 4.1.3.1 Thevenin Equivalent Circuit

For the amplification a non-inverting configuration for the op-amp was used. However, a slightly more complicated version was used due to the necessity for offsetting the incoming signal to produce true AC.

In many circuits potentiometer circuits can be simplified with Thevenin's theorem. The theorem allows converting a more complicated circuit consisting only from resistors, voltage sources and current sources in equivalent voltage source connected in series with a resistor (see Figure A-1 ).

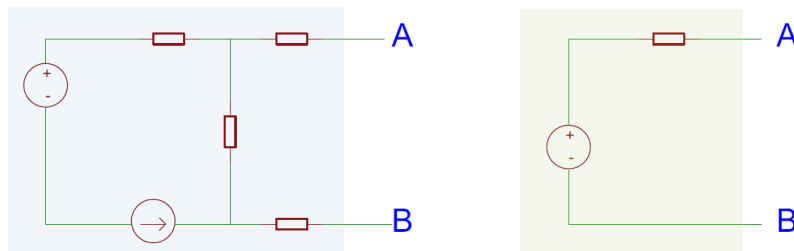


Figure A-1 Example of Thevenin's theorem, where circuit on the left is simplified to equivalent circuit on the right



Basically, we have to convert Figure A-2 circuit to a Thevenin equivalent (Figure A-3) Voltage ( $V_{the}$ ) and Thevenin resistance ( $R_{the}$ ). In our case  $V_{the}$  is the same as  $V_{os}$  (no load present) and  $R_{the}$ .

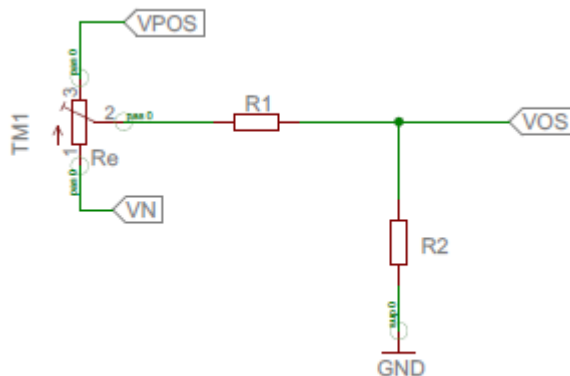


Figure A-2 the real life model

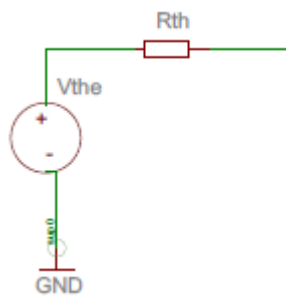


Figure A-3 Necessary thevenin equivalent

In the diagram above,  $V_{POS}$  is positive voltage and  $V_N$  is negative voltage. Also  $N$  in the equation is state of the potentiometer in normalized form. And  $V_{pot}$  is the output voltage from potentiometer.

1. First we can replace the potentiometer with its thevenin equivalent (see Figure A-4) and then substitute it back into original circuit.

$$V_{pot} = V_{POS} - \left( N * Re * \frac{V_{POS} - V_N}{Re} \right) = V_{POS} - (N * (V_{POS} - V_N))$$

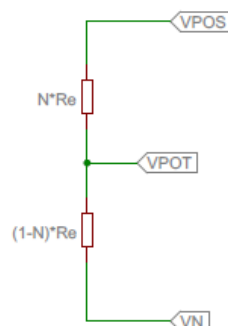


Figure A-4 Potentiometer equivalent circuit with two resistors

2. To find the potentiometers resistance ( $R_{pot}$ ) we can just short out the source  $V_{POS}$  and  $V_N$ , essentially making the two resistors parallel

$$\frac{1}{R_{pot}} = (N * R_e) \parallel [(1 - N) * R_e] = \frac{(1 - N) * R_e + N * R_e}{N * (1 - N) * R_e * R_e}$$

$$R_{pot} = N * (1 - N) * R_e$$

3. So the final steps are to substitute the Thevenin circuit for the potentiometer into the original circuit and find the thevenin replacement. (See Figure A-5)

$$V_{os} = \frac{V_{pot} * R_2}{R_{pot} + R_1 + R_2} = \frac{(V_{POS} - [N * (V_{POS} - V_N)]) * R_2}{N * (1 - N) + R_1 + R_2}$$

$$R_{the} = (R_{pot} + R_1) \parallel R_2 = (N * (1 - N) + R_1) \parallel R_2$$

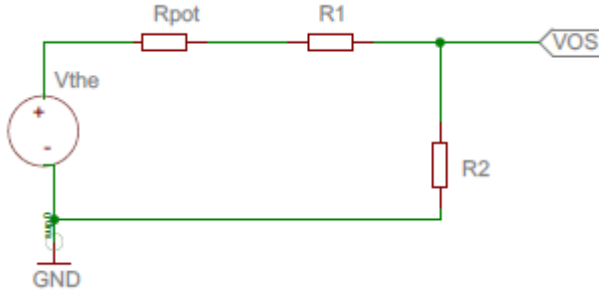


Figure A-5 Thevenin equivalent circuit of the potentiometer

4. The equation can be further simplified, if we assume that  $R_2 < R_1$  and  $R_e < R_1$ , then it becomes:

$$R_{the} \approx R_2$$

$$V_{os} \approx (V_N + (V_P - V_N)N) * \frac{R_2}{R_1}$$

At this stage we have  $V_{os}$  and  $R_{the}$  expressions which are necessary for Thevenin substitute. Now consider this non-inverting amplification circuit (see Figure A-6):

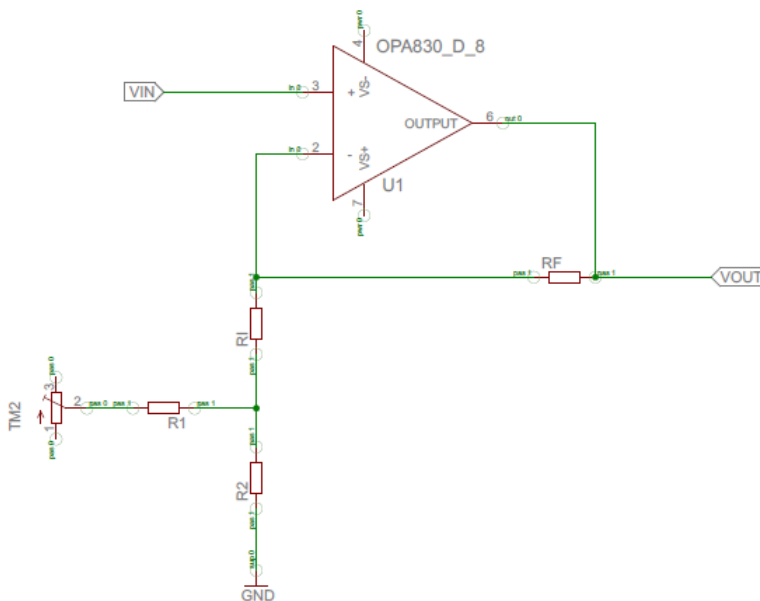


Figure A-6 Non-inverting amplification circuit with potentiometer for offsetting the output

The output voltage ( $V_{out}$ ) in that kind of circuit would equal:

$$V_{out} = V_{in} * \left(1 + \frac{R_f}{R_i + R_{the}}\right) - V_{os} * \left(\frac{R_f}{R_i + R_{th}}\right)$$

With the assumptions made previously we know that  $R_{the} = R_2$  and if further we assume that  $R_2 < R_i$ , then  $R_2$  can be ignored. Basically using Thevenin theorem the circuit simplifies to one shown in Figure A-7.

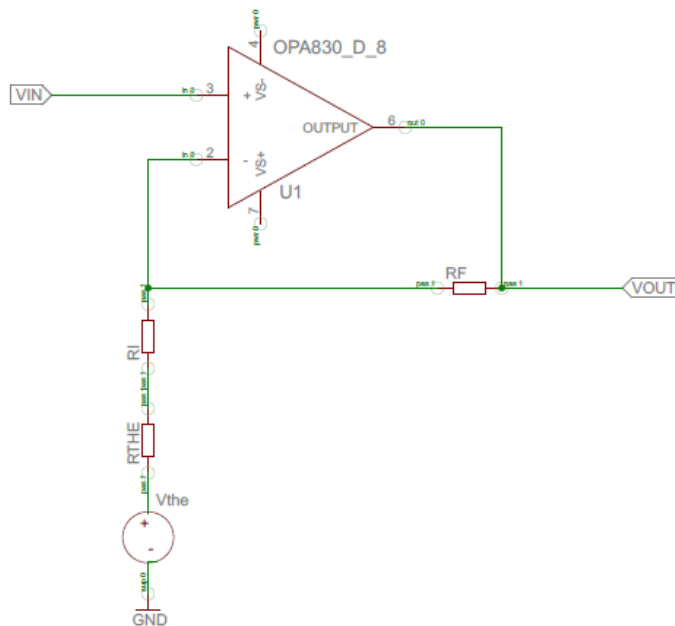


Figure A-7 Potentiometer substituted in Thevenin network

Now we can calculate the resistors necessary for the amplification of the original signal. For now we shall assume that  $V_{os}$  is 0V to simplify the choosing of the resistors –  $R_f$  and  $R_i$ .

The input signal's maximum voltage is 1 V ( $V_{in}$ ), the ADC of xmega16e5 ideally works up to 2.54V. So since we taken out offset of  $V_{os}$ , we need to calculate the amplification  $2 * 2.54 \approx 5$  V, because afterwards the signal will be shifted by half to make a true AC signal. And we end up sending back to xmega16e5 the maximum value of 2.5V. Also let's choose the  $R_{the}$  as 100  $\Omega$ . Note that the equation for calculating  $V_{out}$  starts to be almost exactly like for a normal non-inverting amplifier.

$$5 = 1 + \frac{R_f}{R_i + 100} = 1 + \frac{62000}{16000 + 100} \approx 4.85 \text{ V}$$

The values which were chosen for this stage in the circuit are as following:

1.  $R_f = 62 \text{ k}\Omega$
2.  $R_i = 16 \text{ k}\Omega$
3.  $R_{the} = 100 \text{ }\Omega$

4.  $R1 = 10\text{ k}\Omega$

The values should produce an AC signal from -2.42 V to 2.42 V, providing the potentiometer is correctly calibrated. See figure for the circuit responsible for the offset stage and object excitation.

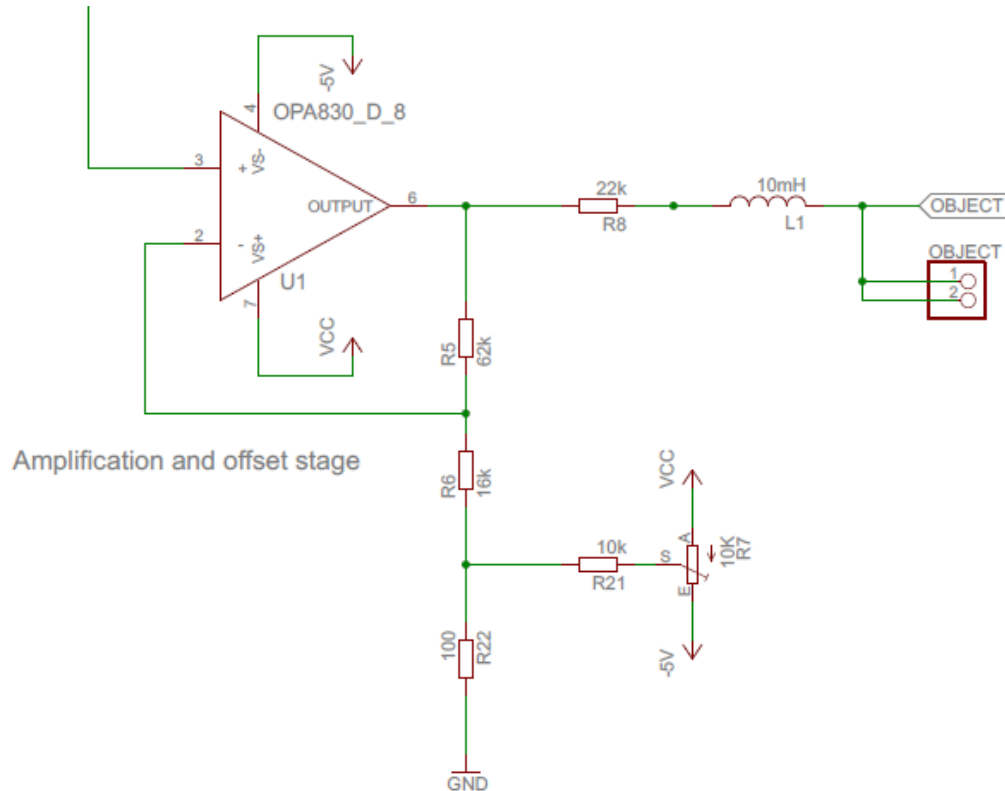


Figure A-8 Circuit for amplification and offset + object excitation

Note the 10mH inductor L1, which is used as bias inductor in this circuit. This is a common practice, to narrow the excitation range of the signal. In other words it compresses the frequency response range from high frequencies like 20 Mhz to the range of 3.5Mhz.

#### 4.1.4 Peak detector and buffer stage

Obviously micro-controller isn't able to easily read the amplitude of a sine wave signal. It is possible by sampling the signal in high frequency at least 2 times the frequency of the input signal. Which in the case of GD system is very inconvenient and with most micro-controllers impossible, due to the frequencies used in the system. A more convenient way of detecting the amplitude of the signal is to use a peak detector also known as precision rectifier. By using such device the micro-controller just needs to read the analog value as it would be done normally, no need for sampling. Also it extends the range of signals whose amplitude can be read with micro-controller.

The peak detector (precision rectifier) basically consists of a super diode, which is just an opamp in voltage follower mode with a feedback loop after signal diode. In such configuration the opamp compensates the 0.6 voltage drop introduced by the diode. Hence it's called super diode, because it doesn't have voltage drop.

Another essential part of the peak detector is to store the highest voltage in some sort of capacitor. If only a capacitor is used without automatic reset, the capacitor can be almost any value. However, if automatic reset is introduced in the system, then the value of capacitor and reset resistor has to be calculated to fit your frequencies by using standard low pass filter equation.

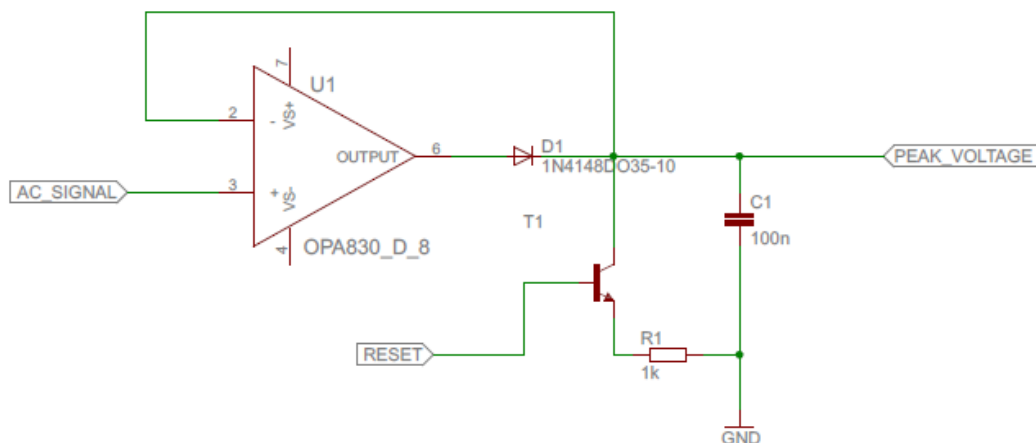


Figure A-1 Peak detector with manual reset function, which can be controlled via micro-controller

Only at the very beginning GD system was using peak detector as seen in Figure A-1. After understanding what frequencies would be involved and the convenience of not resetting the detector after every frequency, it was decided to use peak detector with the resistor R1 permanently connected to the ground, essentially making an automatic reset.

First of all before choosing the resistor for automatic peak detector it was necessary to measure how quick the frequencies are changing, so we could give the maximum time available to charge the capacitor.

The speed measured was 8.4 sweeps a second. One sweep consisted of 160 data points. In total  $160 * 8.4 = 1344$  peaks were detected a second. Note that the peaks detected a second is proportional of the frequency between measurements, which is 1.3 kHz.

Basically we need to choose resistor and a capacitor which would let frequencies lower than 1.3 kHz through the system. Which is very convenient since, the starting frequency of the system is around 2 kHz, and in some cases it was changed to 20 kHz. Fun fact: Some engineers consider calibrating peak detector as black art.

Assuming we have 100nF capacitor used then the resistor would be:

$$f_c = \frac{1}{2\pi RC}$$

$$R = \frac{1}{2\pi C f_c} = \frac{1}{2\pi * 100 * 10^{-9} * 1344} \approx 1.18 \text{ k}\Omega$$

Since a resistor of 1.18 k $\Omega$  doesn't exist, a 1.2 k $\Omega$  will be used, hence the cut-off frequency will become:

$$f_c = \frac{1}{2\pi * 100 * 10^{-9} * 1200} = 1.32 \text{ kHz}$$

By using a resistor of 1.2 k $\Omega$  the cut-off frequency will be 1.32 kHz, which is very close to the desired frequency response. There will be negligible loss. However, even bigger resistor can be used to further filter the peak detectors data; obviously more data is lost in the process.

There is one more issue to be addressed with peak detector. When micro-controller reads the value directly from the peak detector, it discharges the capacitor and in the process some data is lost. To avoid unwanted discharge, in precision peak detector GD system is using another opamp as buffer. Due to high input impedance of the opamp, it is possible to configure it in voltage follower mode and read the value of the capacitor without discharging it. The final circuit for precision peak detector ended up as seen in Figure A-2.

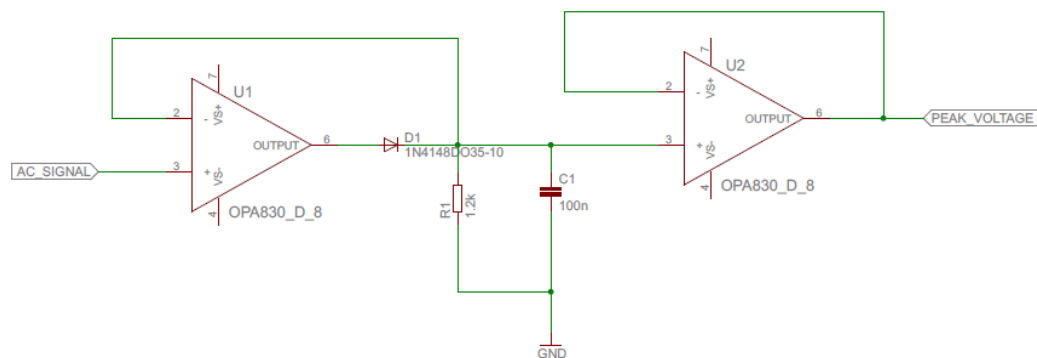


Figure A-2 Precision peak detector, with buffer stage to avoid unwanted discharge while reading the value

#### 4.1.5 Wireless communication – Bluetooth

For communication between PC and GD hardware it was decided to use wireless technology, more precisely Bluetooth technology. Bluetooth enables the hardware to link with any device which supports it, including computers, smart phones etc. In other words, it gives the GD system more freedom of usage.

Bluetooth technology is a standard for exchanging data in short distances, using 2.4 GHz to 2.485 GHz. For different functions there are different protocols available for Bluetooth. Some of them are Link management protocol (LMP), Audio/video remote control profile (AVRCP), Radio frequency communication (RFCOMM) etc. With those protocols Bluetooth can be used to do wide range of tasks with generic drivers available in the devices, network access, serial communication, message exchange, human interface emulation (mouse, keyboard, joysticks etc.), printing service and many more.

Initially it was decided that gesture recognition could be done on the GD system hardware and HID (human interface device) profile could be used to interface with the phone and computer. Unfortunately due to unforeseen difficulties with the ARM processor that had to be dropped for time being.

Instead of HID profile, GD system uses Serial Port Profile (SPP), which basically acts as wireless serial port. More precisely, it emulates RS-232 protocol, which is commonly used by any micro-controller. Due to this reason it was quite easy to interface it. Also SPP is the most familiar form of communication when it comes to micro-controllers, where there is no need for special drivers; it's all supported by OS. Even more attractive is that this profile is supported by Android and iOS devices. However, to use the SPP with Apple products the engineer has to acquire RFCOMM license from them. Another notable feature of Bluetooth technology is that other than Apple's RFCOMM license it doesn't need any license to be used with rest of the device. The Bluetooth special interest group encourages the usage of their technology, only thing asking in return is acknowledgment that their technology is being used.

##### 4.1.5.1 HC-06 Bluetooth module

HC-06 (See Figure A-1) Bluetooth module probably is the cheapest such device on the market. Obviously it only supports one profile which is the Serial Port Profile. Nonetheless it is great for prototyping and supports quite fast baud rates up to 1382400 bps.



Figure A-1 HC-06 Bluetooth module with SPP support

Note that there is HC-05 as well, which is slightly more advanced version of Bluetooth, it also supports master mode, where basically other SPP enabled Bluetooth devices are able to connect to it. On the contrary HC-06 is only able to act in slave mode. The pinouts and

firmware for HC-06 and HC-05 are essentially the same. The main difference is that HC-05 expects “\n\r” after every command and HC-06 doesn’t.

Stock HC-06 runs in baud rate of 9600. Its identifier name is “HC-06” and default pin 1234. Obviously the module can be programmed to whatever name, pin code and baud rate necessary for the application. See Table 4-2 for programming commands accepted by HC-06

Command	Response	Comment
<b>AT</b>	OK	Verifies that it has entered programming mode
<b>AT+VERSION</b>	OKlinvorV1.8	Return firmware’s version
<b>AT+NAMEblu</b>	OKsetname	Sets the module’s name to “blu”
<b>AT+PIN1234</b>	OKsetPIN	Sets the PIN to “1234”
<b>AT+BAUD1</b>	OK1200	Sets baud rate to 1200
<b>AT+BAUD2</b>	OK2400	Sets baud rate to 2400
<b>AT+BAUD3</b>	OK4800	Sets baud rate to 4800
<b>AT+BAUD4</b>	OK9600	Sets baud rate to 9600 (default)
<b>AT+BAUD5</b>	OK19200	Sets baud rate to 19200
<b>AT+BAUD6</b>	OK38400	Sets baud rate to 38400
<b>AT+BAUD7</b>	OK57600	Sets baud rate to 57600
<b>AT+BAUD8</b>	OK115200	Sets baud rate to 115200
<b>AT+BAUD9</b>	OK230400	Sets baud rate to 230400
<b>AT+BAUDA</b>	OK460800	Sets baud rate to 460800
<b>AT+BAUDB</b>	OK921600	Sets baud rate to 921600
<b>AT+BAUDC</b>	OK1382400	Sets baud rate to 1382400

Table 4-2 HC-06 list of programming commands

The programming mode is entered automatically when the Bluetooth module is first powered up and stays in it until it gets connected to a master. Note that programming has to be done by directly interfacing with the Bluetooth module. It cannot be done wirelessly.

HC-06 needs to be programmed only once, it stores its configuration in his own EEPROM memory and next it will boot up with the properties specified earlier. In the case of GD system the module is configured to communicate with baud rate of 115200 bps, it has name of “kulaks” (which in my native Language means “fist”) and PIN code has been left as it is by default “1234”.

The GD system board is designed to allow interfacing with the HC-06 directly on the board for more convenient programming. Otherwise it would only be possible through xmega16e5 micro-controller. But now any serial device can be connected for communication/programming like FTDI232. However it is very important to remember that the module works in 3.3 V logic level, if 5V are applied it may damage the device!

When integrating the Bluetooth module on the GD system main board, it had the option to connect indicator LED to the module (See Figure A-2). Basically, the LED is able to show two states of the HC-06. If the LED is blinking it means that the Bluetooth module is in AT mode (programming mode) or in active search mode (not connected/paired).



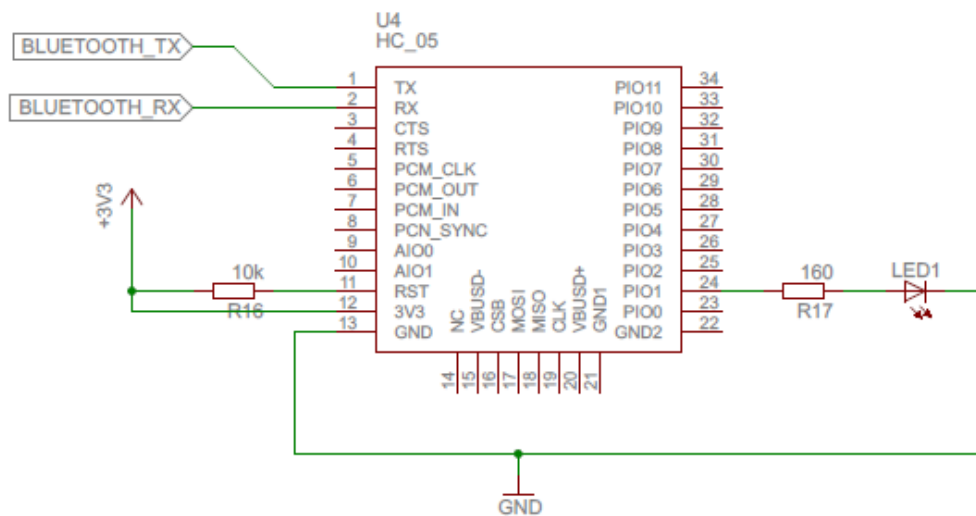


Figure A-2 HC-06 Bluetooth interface circuit diagram with current state LED connected

#### 4.1.6 Main processing unit

One of the integral parts of the project was to choose the main processing unit. At the very early stage in the project development it was decided that the pattern recognition could be done on the microprocessor itself. By doing so, the final gesture could be sent to the PC or smart phone, significantly reducing the amount of data transmitted between them. Multiple ARM platforms were considered to accomplish this task, ranging from rather slow 32 bit ARM microprocessors to Raspberry Pi. Unfortunately, due to lack of support and documentation for the chosen TI ARM CPU this solution was eventually dropped in favor of Atmel's new Xmega series micro-controllers.

##### 4.1.6.1 TI ARM TMS570

The first micro-controller chosen for the project was Texas Instruments ARM TMS570 microprocessor based on Cortex R4 family. Which is a quite cheap microprocessor, but is able provide high performance. It is able to clock up to 180 MHz, has 12 bit DAC and hardware support floating point operations.

As mentioned earlier multiple ARM processors were considered see Table 4-3.

Feature	STM32 Discovery	Raspberry Pi	NXP LPC1769	TMS570
ADC	1x16 channels	NA	1x8 Channels	2x24 Channels
ADC resolution	12 bit	NA	12 bit	12 bit
Max clock speed	72 MHz	1 GHz	100 MHz	180 MHz
RAM	128Kb	512 Mb	64 Kb	192 Kb, can be extended, support EMIF <sup>7</sup>
GPIO pins	80	26	70	58
SPI	2	1	1	3
I <sup>2</sup> C	2	1	3	1
USART	2	1	4	2
Program Memory	512 Kb	>1 Gb	512 Kb	1280 Kb
Price	£ 26.70	£ 29.95	£ 19.80	£ 11.20

Table 4-3 Comparison between MCUs considered while building GD system

Most of the functions provided by the micro-controllers wouldn't be used anyway, because GD system basically only needs 4 main features – Fast ADC, High performance, enough RAM for pattern recognition and at least one USART.

Due to fixed RAM size STM32 and NXP LPC1769 were quickly discarded and it had to be chosen between Raspberry Pi and TMS570. It was decided to use TMS570, because it felt that Raspberry Pi provides way more power than it is necessary for the system. Also it is missing ADC; an external ADC controller would be needed to be implemented. To minimize the cost and power requirements TMS570 was chosen. A development kit was acquired from Texas instruments for this micro-controller see Figure A-1.



Figure A-1 Texas Instruments TMS570 Hercules family development kit

<sup>7</sup> External memory interface, TMS570 support hardware interface with SDRAM

The TMS570 by default is using real time operating system (RTOS) FreeRTOS. It is free and open source RTOS system which can be embedded in commercial products without revealing source code. See FreeRTOS highlights in Table 4-4.

FreeRTOS highlights	
Pre-emptive scheduling option	Easy to use message passing
Co-operative scheduling option	Round robin with time slicing
ROMable	Mutexes with priority inheritance
6k to 10k ROM footprint	Recursive mutexes
Configurable / scalable	Binary and counting semaphores
Compiler agnostic	Very efficient software timers
Some ports never completely disable interrupts	Easy to use API

Table 4-4 FreeRTOS technology highlights

Source: <http://www.freertos.org/RTOS.html>

Even though RTOS is a useful addition to the MCU, also it allows for more flexibility when a micro-controller runs RTOS, mainly due to the capabilities of multi-tasking. The way GD system was designed there was no need for RTOS, hence no RTOS features were used while coding the ARM development board, except for running one task which does all the necessary computation, like reading ADC, applying filter and sending data through USART.

Also TI has made configuration for the different processes “easy”, with a code generator called “Hercules HALCoGEN”. It allows configuring the speed of USART or setup ADC through a GUI. See Figure A-2 for the main configuration GUI window.

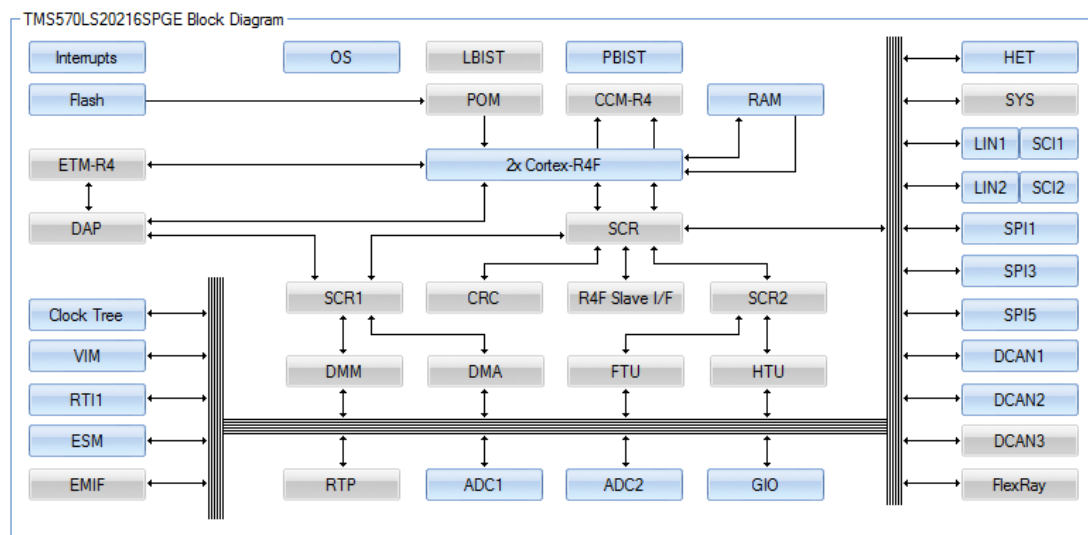


Figure A-2 TMS570 configuration block diagram from HALCoGEN

#### 4.1.6.1.1 Problems with Texas Instruments TMS570

At the beginning the MCU looked very powerful, and rather easy to use. However, there were some important things not considered when choosing TMS570. Apparently it is important to consider the community behind the micro-controller or company, whether the product is actually finished and the support, company is willing to give. Texas instruments was lacking in all three. The MCU chosen for GD system basically doesn't have any community at all; it is a very rarely used micro-controller. However, to be successful using the MCU community isn't the main issue.

The most important of all is how much support the manufacturer is willing to give for their products. It was rather surprising how Texas Instruments was lacking in this field. It seems that TI is only capable of giving support for very basic problems, like input or output issues, but when it comes to something more complicated like issues with RTOS suddenly they are ignoring the question. The intention is not to bash Texas Instruments but rather warn about potential problems when using a device or micro-controller without community. Probably, if someone is a business partner with TI, they would have high quality support.

Furthermore, it is important to investigate whether the chosen MCU is not half finished. Note the gray fields in Figure A-2, those are features yet to be implemented. And for GD system the most important feature, which is why TMS570 was chosen in first place was EMIF, and as it can be seen in the Figure A-2, it is not yet supported. Also it was investigated whether EMIF could be initialized by hand, but the free datasheet provided by TI is just description of the features available in the system, nothing technical at all about registers corresponding to a particular module.

The final issue experienced when working on this project with this MCU, was the outdated documentation and help files. Apparently they were written for completely different system, which is very similar to TMS570, but vector addresses for different hardware functions are different between them, so it was quite confusing in the beginning why it wasn't working. Nonetheless better some kind of documentation than nothing.

Once again, the intention wasn't to bash the Texas Instruments, but only to warn about the potential issues and some more things to be considered when choosing a MCU. To recap, it is mandatory to consider the company's reputation regarding support and the community behind them. Also it is important to remember, designing ARM development board is a project on its own. Because of the problems experienced described above, it was decided to switch to a familiar environment of 8bit micro-controllers, so no more time would be lost on frustration with TI TMS570. More specifically, Atmel AVR series micro-controllers were chosen.

#### 4.1.6.2 Atmel Xmega16e5

As mentioned before, after many weeks of failures and frustration with TI TMS570 ARM MCU, it was finally decided that a more familiar company and micro-controller should be used for the GD system. Basically there were two main options to choose from – mega series and Xmega series. See Table 4-5 for comparison between two equally priced AVR mega and AVR Xmega micro-controllers.

Feature	Atmega32AU	XMega16E5
Max clock speed	20MHz	32MHz
ADC	1x8 Channels	2x16 Channels
ADC resolution	10 bit	12 bit
ADC speed	15 samples per second	300 ksps <sup>8</sup>
DAC	NA	1x8 channels
USART	1	2
SPI	1	1
I <sup>2</sup> C	1	1
Programming interface	ISP	PDI
GPIO	32	26
Operating voltage	5V	3.3V

Table 4-5 Comparison between mega and xmega series micro-controller in the same price tag

It was decided to try and use the new XMega16E5 (for pinout see Figure A-1) micro-controller, even though it was a little bit risky, since the micro-controller was release around November 2013. By learning from previous mistakes, Atmel's support and community behind it was thoroughly examined. Apparently, the new XMega series micro-controllers are very similar to the predecessor *mega*. The syntax is almost the same; furthermore it uses the same environment for programming.

---

<sup>8</sup> Kilo samples per second

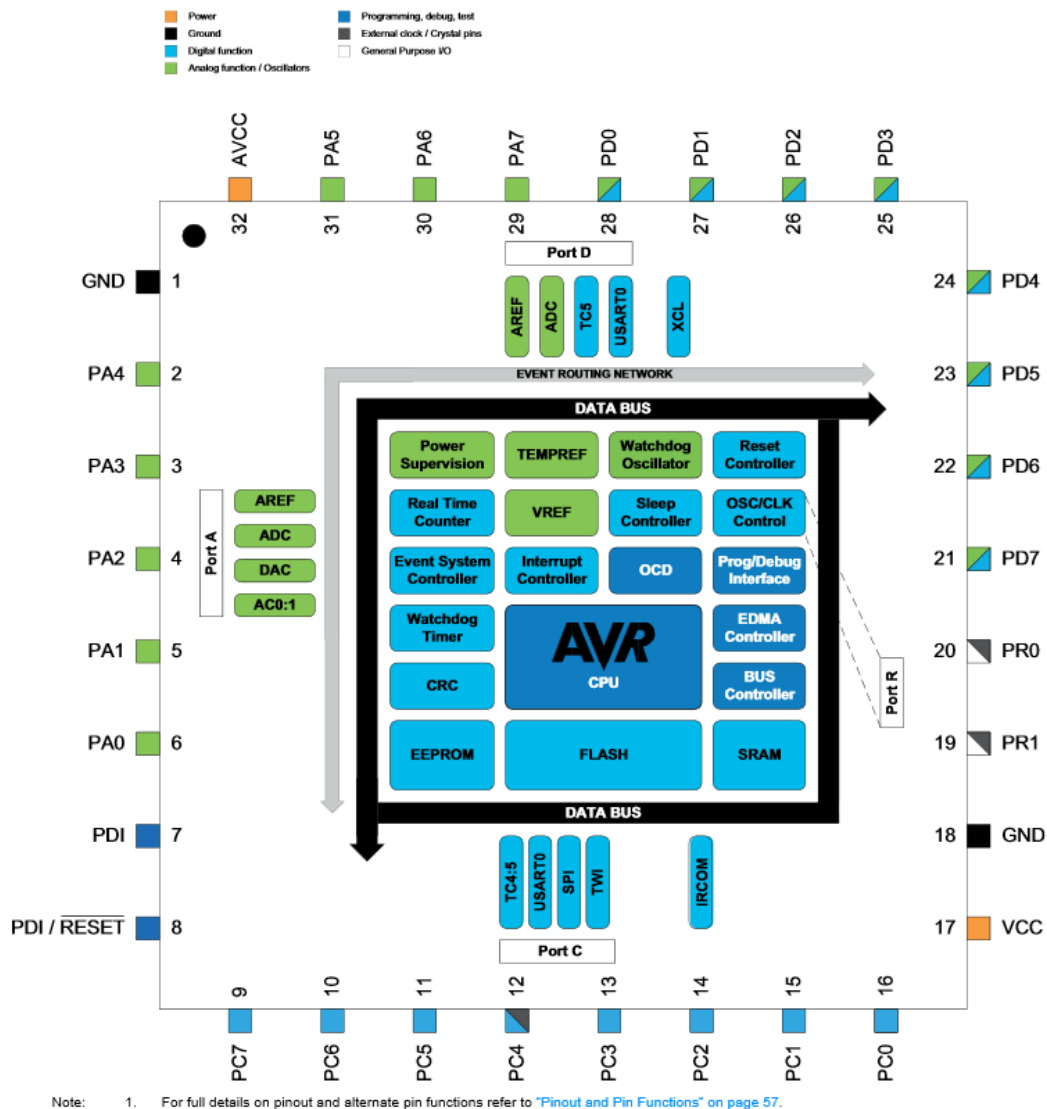


Figure A-1 Xmega16E5 pinout and block diagram  
Source: xmega16e5 datasheet

#### 4.1.6.2.1 Breakout board

The first step was to design a simple breakout board for the new XMega16E5, since there were no development kits available on the market. Even then, Atmel provides with a lot of material on how the PCB should be designed for xmega series.

The resulting circuit diagram and custom made PCB can be seen in Appendix A.

Overall there were no issues with the breakout board itself but rather getting the XMega to program, that is to be recognized by a programmer. As you can see in the Table 4-5 under *programming interface* section the xmega is using programming debugging interface (PDI). PDI is a new programming standard introduced by Atmel, where compared to standard ISP it uses only two wires for programming and debugging. ISP uses 3 wires. PDI works somewhat similar to I<sup>2</sup>C where data line is bi-directional.

Any device with USART can be converted to support PDI see Figure A-1. Due to this reason a standard USBasp can be potentially converted to program xmega series micro-controllers

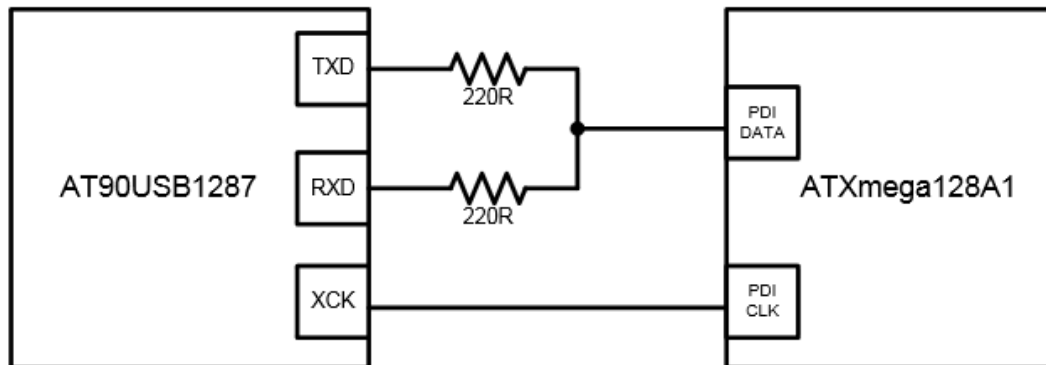


Figure A-1 PDI connection between AT90USB\* and xmega series micro-controller

The possibility of converting the USBASP to support PDI was investigated. However, in the process two xmega15e5 micro-controllers were damaged. Apparently the jumper on the USBASP which states that it can switch between 5V and 3.3V is only for the power line, the transmission line still works in 5V (logic 1 is 5V). See Figure A-2 for circuit diagram on how to convert USBASP to support PDI with maximum voltage on all lines 3.3V.

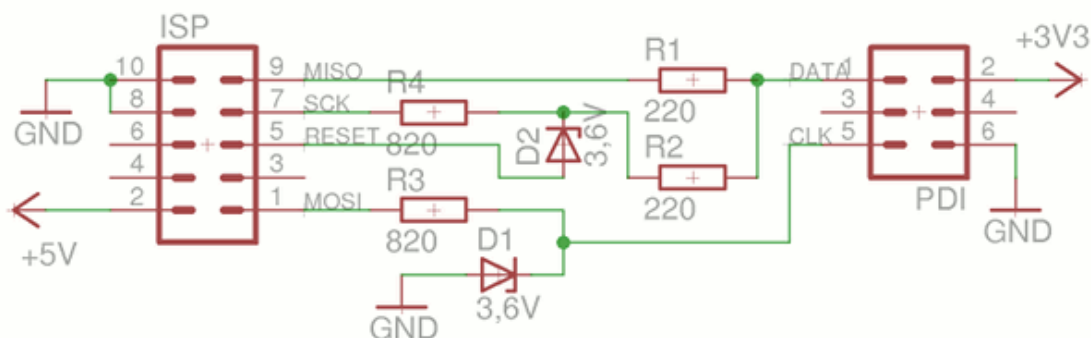


Figure A-2 Standard 5V ISP to 3.3V PDI convertor

Source: <http://szulat.blogspot.co.uk/2012/08/atxmega-programmer-for-050.html>

Even though standard USBASP was successfully converted to support PDI, it was decided to acquire an official AVR programmer to avoid such errors in future. More precisely, AVR ISP MK2 was used for programming the device. One of the main advantages of an official AVR programmer is that it can be used in Atmel Studio directly.

#### 4.1.6.2.2 Configuring xmega system clock

There is one notable difference between setting system clock in xmega compared to mega series. The system clock in xmega is set in run-time, basically in code itself, instead of rewriting fuses. Many times I've accidentally written incorrect fuses in the micro.

By default and on every reset the xmega's clock will be reset back to 2MHz internal clock. And the system clock can be changed during normal operation at any time.

We have the following options, when choosing clock source (datasheet page 96):

1. Internal oscillators
  1. 32kHz ultra low power oscillator
  2. 32.768 kHz calibrated oscillator
  3. 32MHz run-time calibrated oscillator
  4. 8 MHz calibrated oscillator
2. External clock source
  1. 0.4 MHz – 16 MHz crystal oscillator
  2. External clock input
  3. 32.768 kHz crystal oscillator

This section of the report will be about how to set up internal 32 MHz oscillator without external clock calibration. If for any reason a precise 32 MHz clock is needed, please refer to page 99 in xmega datasheet about digital frequency locked loop (DFLL). The DFLL requires 32.768 kHz external clock connected to the device. However, from tests done while designing GD system, 32 MHz clock is precise enough for 115200 baud rate for USART. No data corruption has been encountered while sending data.

It was surprisingly simple to set up internal clock source for Xmega16e5. First, in OSC\_CTRL register we must enable the clock source which we intend to use with the micro-controller. In the case of GD system bit 1 RC32MEN has to be set high to enable internal 32 MHz clock. See Table 4-6.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
+0x00	CTRL	–	RC8MLPM	RC8MEN	PLLEN	XOSCEN	RC32KEN	RC32MEN	RC2MEN	104
+0x01	STATUS	–	–	RC8MRDY	PLLRDY	XOSCRDY	RC32KRDY	RC32MRDY	RC2MRDY	104
+0x02	XOSCCTRL	FRQRANGE[1:0]		X32KLPM	XOSCPWR	XOSCSEL[3:0]				105
					XOSCSEL[4]					
+0x03	XOSCFAIL	–	–	–	–	PLLFDF	PLLFDEN	XOSCFDF	XOSCFDEN	106
+0x04	RC32KCAL	RC32KCAL[7:0]								107
+0x05	PLLCTRL	PLLSRC[1:0]		PLLDIV	PLLFAC[4:0]				107	
+0x06	DFLLCTRL	–	–	–	–	–	RC32MMCREF[1:0]		–	107
+0x07	RC8MCAL	RC8MCAL[7:0]								108

Table 4-6 Register summary for oscillator

More details about the individual bits in OSC\_CTRL register see datasheet page 104.

After internal 32 MHz clock has been enabled, a good practice is to wait until the clock stabilizes. As it can be seen in table there is a register called STATUS. The register holds information whether a particular clock has stabilized or not. If it has stabilized, the corresponding bit will be set to HIGH. In the case of GD system, again bit 1 RC32MRDY has to be checked.

After initializing the internal crystal, for us to be able to switch to it, first it's necessary to trigger *configuration change protection* mechanism and the clock source has to be changed in 4 clock cycles. For more details please refer to page 13 in datasheet. The internal 32 MHz



clock can be selected in *CLK\_CTRL* register, with bits *SCLKSEL*. See Table 4-7 for possible *SCLKSEL* configurations.

SCLKSEL[2:0]	Group configuration	Description
000	RC2MHZ	2MHz from 8MHz internal oscillator
001	RC32MHZ	32MHz internal oscillator
010	RC32KHZ	32.768kHz internal oscillator
011	XOSC	External oscillator or clock
100	PLL	Phase locked loop
101	RC8MHZ	8MHz internal oscillator
110	–	Reserved
111	–	Reserved

**Table 4-7 System clock selection**  
Source [xmega16e5 datasheet](#)

After carefully following the datasheet, the internal 32 MHz clock can be selected with the following code:

```
void setUp32MhzInternalOsc()
{
    OSC_CTRL |= OSC_RC32MEN_bm; //Setup 32Mhz crystal

    while(!(OSC_STATUS & OSC_RC32MRDY_bm));

    CCP = CCP_IOREG_gc; //Trigger protection mechanism
    CLK_CTRL = CLK_SCLKSEL_RC32M_gc; //Enable internal 32Mhz crystal
}
```

#### 4.1.6.2.3 Configuring USART for serial communication with Bluetooth

The first thing to do when setting up serial communication is to specify the baud rate. 32 MHz clock will be used for calculations. Xmega16E5 datasheet provides equations for calculating baud rate for different USART modes. See Table 4-8.

Operating mode	Conditions	Baud rate calculation <sup>(1)</sup>	BSEL value calculation
Asynchronous Normal Speed mode (CLK2X = 0)	$BSCALE \geq 0$ $f_{BAUD} \leq \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} \cdot 16(BSEL + 1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE} \cdot 16 f_{BAUD}} - 1$
	$BSCALE < 0$ $f_{BAUD} \leq \frac{f_{PER}}{16}$	$f_{BAUD} = \frac{f_{PER}}{16((2^{BSCALE} \cdot BSEL) + 1)}$	$BSEL = \frac{1}{2^{BSCALE}} \left( \frac{f_{PER}}{16 f_{BAUD}} - 1 \right)$
Asynchronous Double Speed mode (CLK2X = 1)	$BSCALE \geq 0$ $f_{BAUD} \leq \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{2^{BSCALE} \cdot 8(BSEL + 1)}$	$BSEL = \frac{f_{PER}}{2^{BSCALE} \cdot 8 f_{BAUD}} - 1$
	$BSCALE < 0$ $f_{BAUD} \leq \frac{f_{PER}}{8}$	$f_{BAUD} = \frac{f_{PER}}{8((2^{BSCALE} \cdot BSEL) + 1)}$	$BSEL = \frac{1}{2^{BSCALE}} \left( \frac{f_{PER}}{8 f_{BAUD}} - 1 \right)$
Synchronous and master SPI mode	$f_{BAUD} < \frac{f_{PER}}{2}$	$f_{BAUD} = \frac{f_{PER}}{2(BSEL + 1)}$	$BSEL = \frac{f_{PER}}{2 f_{BAUD}} - 1$

Notes: 1. The baud rate is defined to be the transfer rate bit per second (bps).

**Table 4-8 Equations for calculating Baud Rate register setting**

Source: xmega16e5 datasheet

The USART for micro-controller has to be configured in 115200 bps and in Asynchronous mode, just like HC-06 Bluetooth module. With assumption that  $BSCALE$  with 0 and USART in double speed mode will produce usable  $BSEL$  value, the calculations are as following:

$$BSEL = \frac{32 * 10^6}{2^0 * 8 * 115200} - 1 = 33.72 \approx 34$$

The  $f_{baud}$  with  $BSEL$  of 34 would be:

$$f_{baud} = \frac{32 * 10^6}{2^0 * 8(34 + 1)} = 114285.71$$

The error with  $BSEL$  being 34 would be 0.79%, which is acceptable.

There is nothing particularly special setting up the serial, it is more or less described quite thoroughly in the datasheet. However, notable feature implemented in XMega architecture is the ability to remap functional pins. In case of GD system, serial RX and TX pins were remapped in a more convenient position for designing PCB.

By default RX and TX pins for USART0 are pins PC3 and PC2, as mentioned before they were remapped. More precisely, GD system has remapped USART0 to PC6 and PC7 accordingly. You can see the functions of the PORTC in Table 4-9.

PORT C	Pin #	TCC4	WEXC	TCC5	USARTC0	SPIC	TWI	XCL (LUT)	EXTCLK	AC OUT
PC0	16	OC4A	OC4ALS				SDA	IN1/OUT0		
PC1	15	OC4B	OC4AHS		XCK0		SCL	IN2		
PC2	14	OC4C	OC4BLS		RXD0			IN0		
PC3	13	OC4D	OC4BHS		TXD0			IN3		
PC4	12	OC4A	OC4CLS	OC5A		SS		IN1/OUT0	EXTCLK	
PC5	11	OC4B	OC4CHS	OC5B	XCK0	SCK		IN2		
PC6	10	OC4C	OC4DLS		RXD0	MISO		IN0		AC1OUT
PC7	9	OC4D	OC4DHS		TXD0	MOSI		IN3		AC0OUT

**Table 4-9 PORTC alternative functions**

Source: xmega16e5 datasheet

Every group of ports have a REMAP register, and just by following the instructions in the register you can switch between the alternative functions of the pins in the corresponding port. See Table 4-10 with REMAP register for PORTC in xmega16e5.

The pin remap functionality is available for PORTC and PORTD only.

Bit	7	6	5	4	3	2	1	0
+0x0E	–	–	–	USART0	TC4D	TC4C	TC4B	TC4A
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial value	0	0	0	0	0	0	0	0

- **Bit 7:5 – Reserved**  
These bits are unused and reserved for future use. For compatibility with future devices, always write these bits to zero when this register is written.
- **Bit 4 – USART0: USART0 Remap**  
Setting this bit to one will move the pin location of USART0 from Px[3:0] to Px[7:4].

**Table 4-10 Pin Remap register**

Source: xmega16e5 datasheet

Basically we end up configuring the serial communication as following:

```
void SerialC::setUpSerial()
{
    //For the sake of example, I'll just REMAP the USART pins from PC3 and PC2
    //to PC7 and PC6
    PORTC_REMAP |= 0x16; //See page 152 in datasheet, remaps the USART0

    PORTC_OUTSET = PIN7_bm; //Let's make PC7 as TX
    PORTC_DIRSET = PIN7_bm; //TX pin as output

    PORTC_OUTCLR = PIN6_bm;
    PORTC_DIRCLR = PIN6_bm; //PC6 as RX

    // Baud rate selection
    // BSEL = (32000000 / (2^0 * 8*115200)) -1 = 34.7222 -> BSCALE = 0
    // FBAUD = ( (32000000)/(2^0*8(34+1)) ) = 114285.71 -> it's alright

    USARTC0_BAUDCTRLB = 0; //Just to be sure that BSCALE is 0
    USARTC0_BAUDCTRLA = 0x22; // 34

    //Disable interrupts
    USARTC0_CTRLA = 0;
    //8 data bits, no parity and 1 stop bit
    //USARTC0_CTRLA = USART_CMODE0_bm | USART_PMODE0_bm | USART_CHSIZE_8BIT_gc;
    USARTC0_CTRLA = USART_CHSIZE_8BIT_gc;

    //Enable receive and transmit
    USARTC0_CTRLB = USART_TXEN_bm | USART_CLK2X_bm | USART_RXEN_bm; // And
    enable high speed mode
}
```

#### 4.1.6.2.4 Configuring ADC

The final module used in GD system hardware is the ADC. As mentioned in the table xmega16e5 has 12 bit ADC with ability to do 300 thousand samples per second. The ADC can be configured for single-ended or differential measurements. Also it has built in programmable amplification circuit. The ADC can provide unsigned and signed results.

Like for any other micro-controller the ADC conversion can be initiated either via software or external interrupt. However, it also has the ability to automatically initiate conversion in specific time intervals with a system called EDMA the conversion results can automatically be transferred to specified memory location.

The GD system's ADC has been configured to operate in single-ended conversion, which is the basic and most common type of ADC. Also it has been set to use external reference, which is pin PA0, and PA1 is set as ADC input.

```
ADCA_CH0_CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc; //Configure ADC conversion as single ended
ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN1_gc; // Set ADC to pin1, or PA1, because PA0 = AREFA
```

Obviously by default ADC isn't enabled to do 300 thousand conversions per second. To do that, it is necessary to disable current protection mechanism, or in case of GD system it was lowered to LOW protection mechanism (see Table 4-11), essentially doing slightly less conversions than 300k.

The sample rate can be calculated using the following equation:

$$\text{Sample rate} = \frac{f_{adc}}{0.5 * (\text{resolution} + \text{sampval}) + \text{gainfactor}}$$

- *Fadc* is ADC clock speed (GD system has it configured Fclk/32)
- *Resolution* is either 8 or 12 bits
- *sampval* is the value programmed in the sampling control register (by default 0)
- *gainfactor* is the internal amplification, by default it is 0 (which means 1x gain)

By taking all variables into account the sample rate of the GD system is as follows:

$$\text{Sample rate} = \frac{1000000}{0.5 * (12 + 0) + 0} = 166666 \frac{\text{samples}}{\text{second}}$$

CURRLIMIT[1:0]	Group configuration	Description
00	NO	No limit
01	LOW	Low current limit, max. sampling rate 225kSPS
10	MED	Medium current limit, max. sampling rate 150kSPS
11	HIGH	High current limit, max. sampling rate 75kSPS

Table 4-11 Table of maximum conversion rate in respect to current limit  
Source: xmega16e5 datasheet

The configuration in AVR C++ code translates as following:

```
void setUpADC()
{
    ADCA_CH0_CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc; //Configure ADC conversion
    as single ended
    ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN1_gc; // Set ADC to pin1, or PA1,
    because PA0 = AREFA

    //Sample rate = 166666 samples.
    // 1000000/(0.5*(12+0)+0) = 166666
    // See page 351
    ADCA_CTRLB = ADC_CURRLIMIT_LOW_gc | ADC_RESOLUTION_12BIT_gc; //Low current
    limit, up to 225ksps, and 12 bit resolution
    //ADCA_CTRLB = ADC_FREERUN_bm;
    //ADCA_REFCTRL = ADC_REFSEL_INTVCC_gc; // Internal reference VCC/1.6
    ADCA_REFCTRL = ADC_REFSEL_AREFA_gc; // External reference AVCC - 0.6
    ADCA_PRESCALER = ADC_PRESCALER_DIV32_gc; // Prescaler div 4

    ADCA_CTRLA = ADC_ENABLE_bm; //Enable ADC
}
```

Note that xmega series micro-controllers aren't able to read higher voltage than AREF-0.6V. Maximum AREF can be 3.6V. In case of GD system it is 3.3V, with a jumper it can be switched to very stable 2.7V. Furthermore, a constant 0.135 must be subtracted, due to properties of the ADC

According to datasheet to extract the voltage following calculations must be done:

$$ADC \text{ results in } V = \frac{ADC \text{ RAW} * AREF - 0.6}{4096} - 0.135$$

#### 4.1.6.2.5 GD systems firmware

The firmware of GD system is designed in a way that engineer can interact with the device at any time. Basically, on every cycle the firmware checks whether any new commands have been received via USART. See Table 4-12 for available commands.

Command	Result
<b>T</b>	Will test communication between Bluetooth and PC using printf and scanf functions
<b>N</b>	Increments current output frequency by single step calculated for frequency sweep
<b>B</b>	Decreases current output frequency by single step calculated for frequency sweep
<b>R</b>	Resets the output frequency
<b>C</b>	It will prompt user to enter new configuration for frequency sweep, in the following structure: [min,max,sweeps].
<b>S</b>	Stop frequency sweep
<b>D</b>	Start frequency sweep
<b>Q</b>	Hard Resets the GD hardware

Table 4-12 List of supported commands programmed in GD system

When frequency sweep has been initiated and all the necessary samples have been collected, the data is low pass filtered and then sent through USART in the following structure:

`{156.77 324.33 134.32 24.56 112.56 ... 123.98 90.12}`

Note that the first and last data points are not separated by space between curly brackets, rest of the data points are separated. Data which are received in the curly brackets will be interpreted as the results from frequency sweep at the PC end.

See Appendix B for firmware code and Figure A-1 for firmware's flowchart

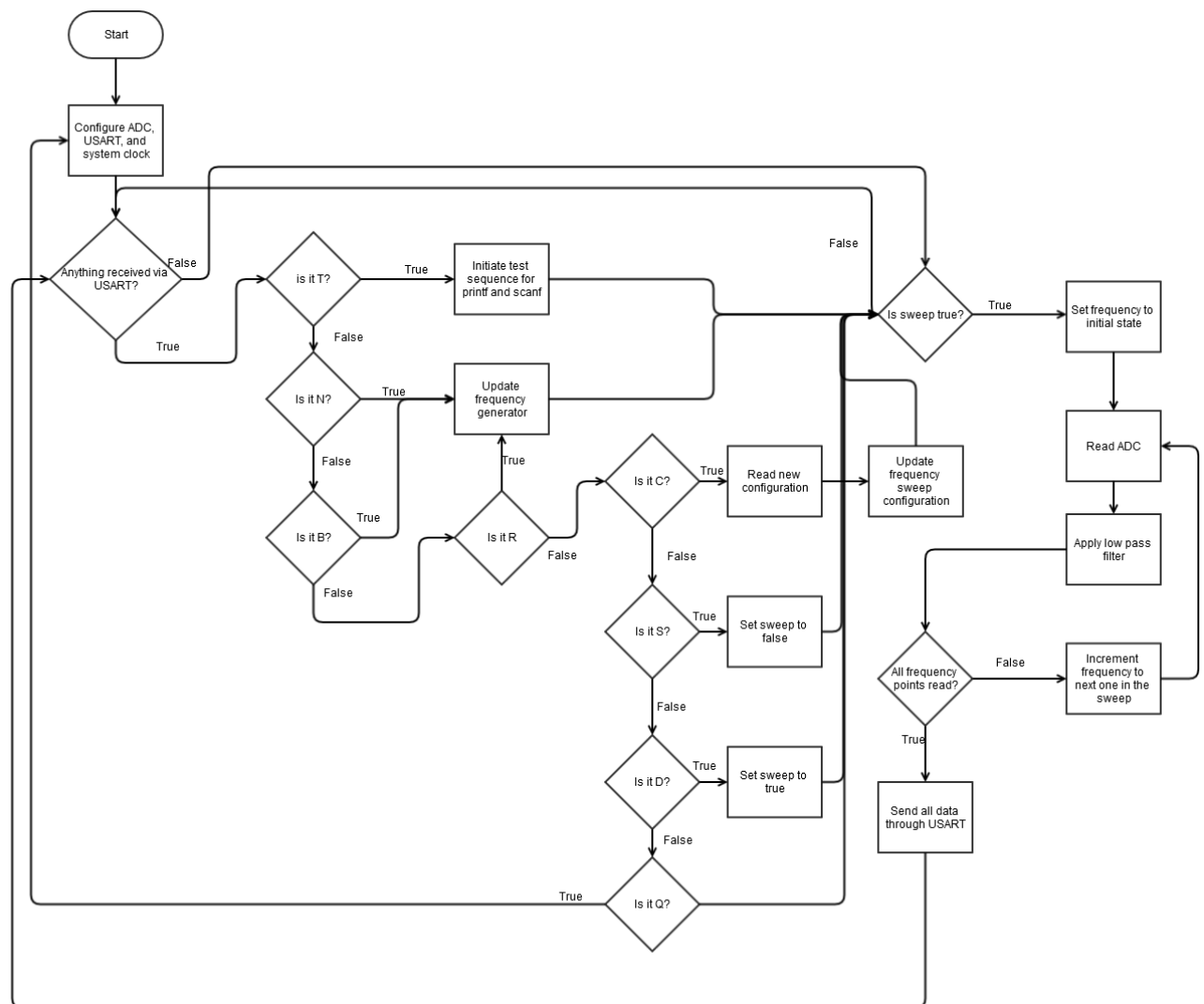


Figure A-1 Flowchart of the GD system's firmware

#### 4.1.7 PCB manufacturing and circuit design

##### 4.1.7.1 PCB etching

One of the essential tasks of the project was to design and manufacture a PCB for the GD system. Manufacturing PCB at home environments is not an easy task. Many limitations apply when designing PCB which can be etched:

1. Vias should be as thick as possible
2. Vias can't be filled, they must be soldered from both sides with a small wire
3. Don't use wire with width less than 0.012 inch, smaller will almost definitely result in broken pathways when etching
4. The smaller the circuit the easier it is to etch it
5. If possible avoid double sided boards, otherwise another level of complexity is introduced in manufacturing process

At home environment there are two common ways of manufacturing the PCB. They only differ from the method how the circuit is transferred to the copper plate. The actual etching between those two methods is the same.

1. Toner transfer
2. Photoresist coating transfer

Toner transfer can be only done if the circuit is printed on a glossy paper with a laser printer. Then with laminator by heating up the glossy paper placed downwards on the copper plate, most of the powder will be transferred to the plate.

Photoresist method allows usage of ink printers. This is the method used when manufacturing GD PCB, also this method will be described in detail further in this section.

1. Preparing the picture of the PCB

Obviously the manufacturing starts with PCB design, which is then printed on A4 sized paper. Note each side is printed separately and without any silkscreen components. Then to make the paper transparent, it just needs to be soaked in cooking oil. If a double side board will be etched, the two pieces of PCB must be glued together to form a pocket for copper plate.

2. Preparing the copper board.

The copper board needs to be cut slightly bigger than the PCB being made. Afterwards it must be sanded with very fine sand paper to get rid of unnecessary bumps and make the surface more even. Finally the plate must be cleaned from any dust particles and grease with acetone. If any particles will be left on it or grease, the board will etch at those points much slower than places where it is clean. By waiting until dirty places are etched valuable copper wires could be destroyed in the process.



### 3. Coating the copper board

Place the copper board in air tight and opaque container. And by using positive 20, the plate should be coated with photoresist lacquer. If double sided board is being manufactured, first coat one side and heat the container with 65°C for 15 min. Then turn the board around coat the other side and continue heating the container for further 25 min.

### 4. Transferring PCB image to the copper plate

First place the copper plate in the transparent pocket and expose each side in UV oven for 1:50 min. Then place the board in photoresist developer, like poly methyl methacrylate for around 1 min. Place the developed board under running water to wash away developer

### 5. Etching the PCB

Heat up sodium persulphate to 40-50°C and place the photo developed board into the solution. Then just wait for 20-30 min until the visible copper is all dissolved. See Figure A-1 for etched PCB.

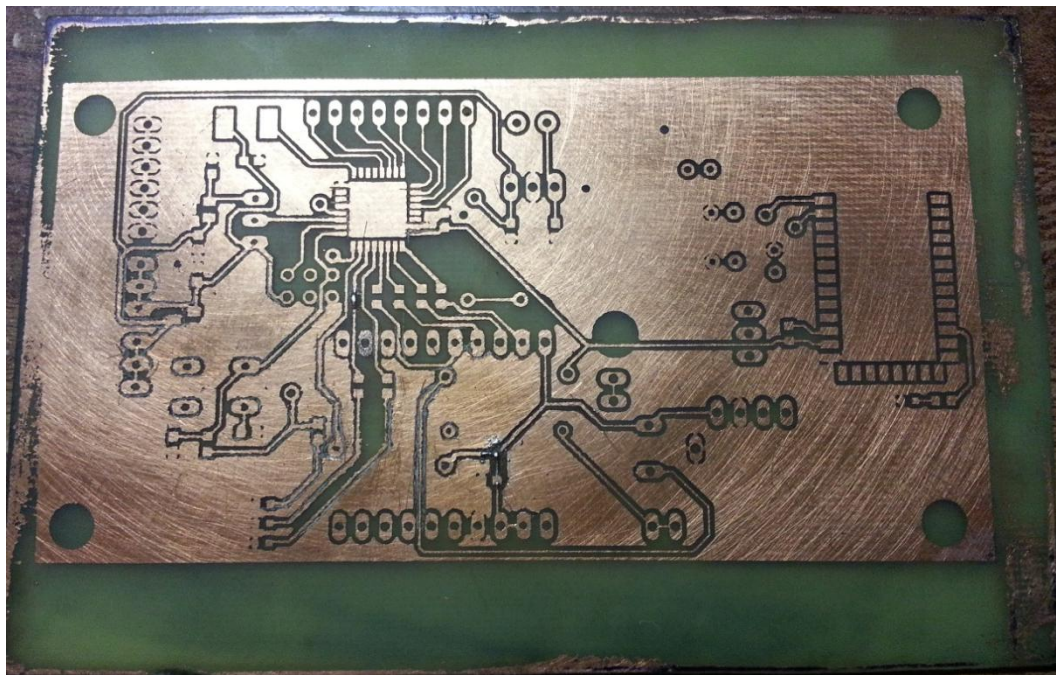


Figure A-1 Just etched PCB

6. Final step in making the PCB is to apply solder mask. For most DIY cases this would be optional. However, due to active use of SMD components in GD system design solder mask is a necessity, it makes soldering much easier. For solder mask Dynamask was used. Basically clean once again the etched PCB with acetone and place the mask on top of the PCB. Then put it through laminator so the mask adheres to the board. Place pads on top of it, and expose in UV oven for 1:30 min. leave it for 1h in a dark place. Remove the second layer and place it in photonegative developer. After the pads are clean from solder mask, place the PCB



back in UV oven for 30 min. Finally place it in oven heated to 65°C to dry it off. See Figure A-2 for PCB board with solder mask applied.

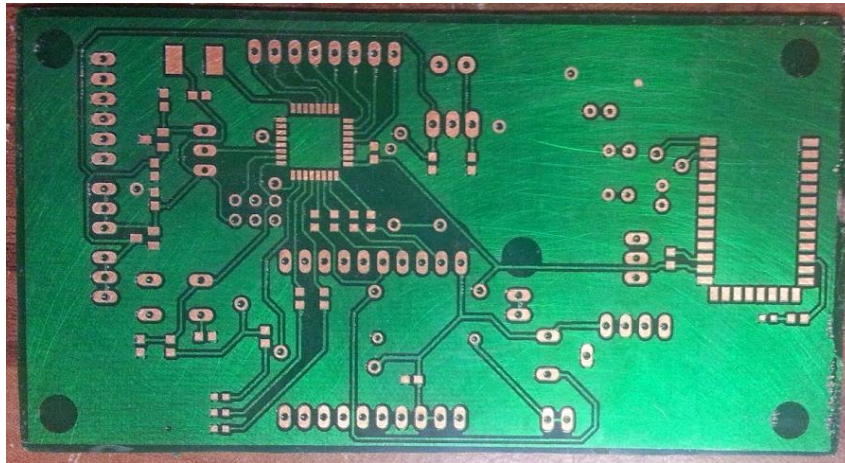


Figure A-2 PCB circuit with solder mask applied

7. Final step is just soldering.

IF any of the steps are unclear, internet is full of instruction on how to manufacture PCB in home environment. They may differ, because the technique which worked best was developed over course of 2 month.

#### 4.1.7.2 Final circuit and PCB design

As mentioned in introduction the GD system was developed to be expandable and provides another engineer with additional GPIO pins. See figure and figure for fully assembled PCB.

See Appendix C for full circuit diagram and PCB.

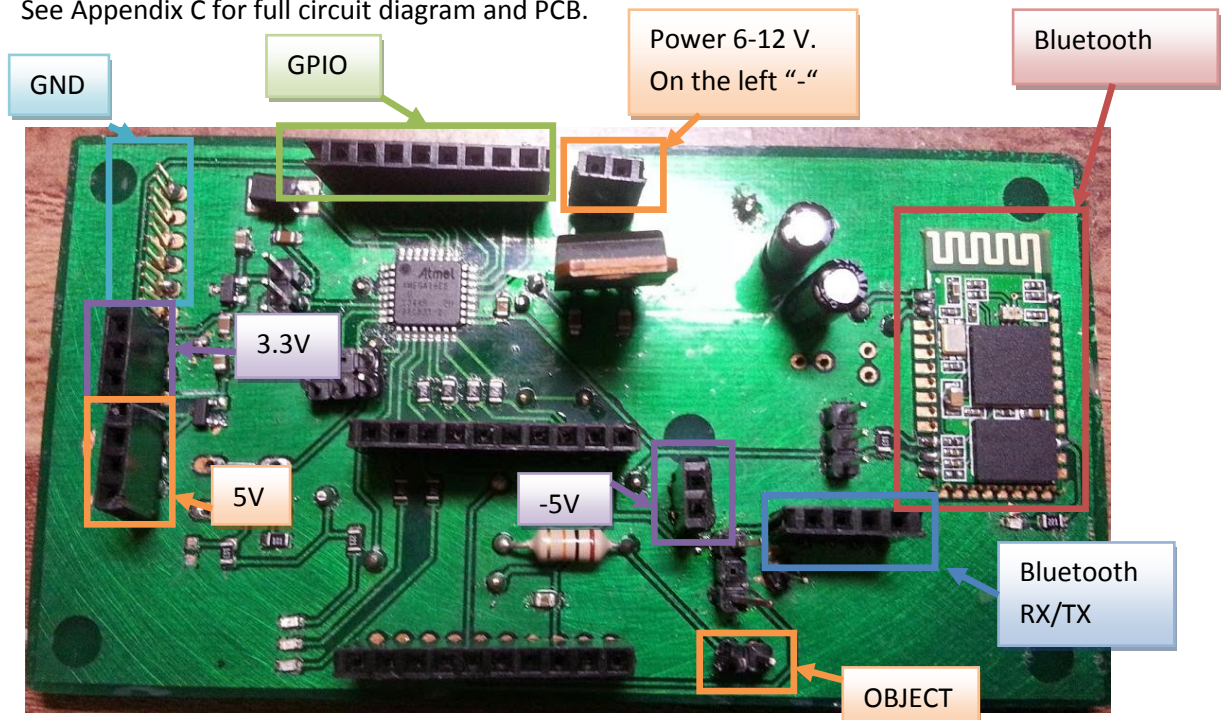


Figure A-1 fully assembled GD system board top side

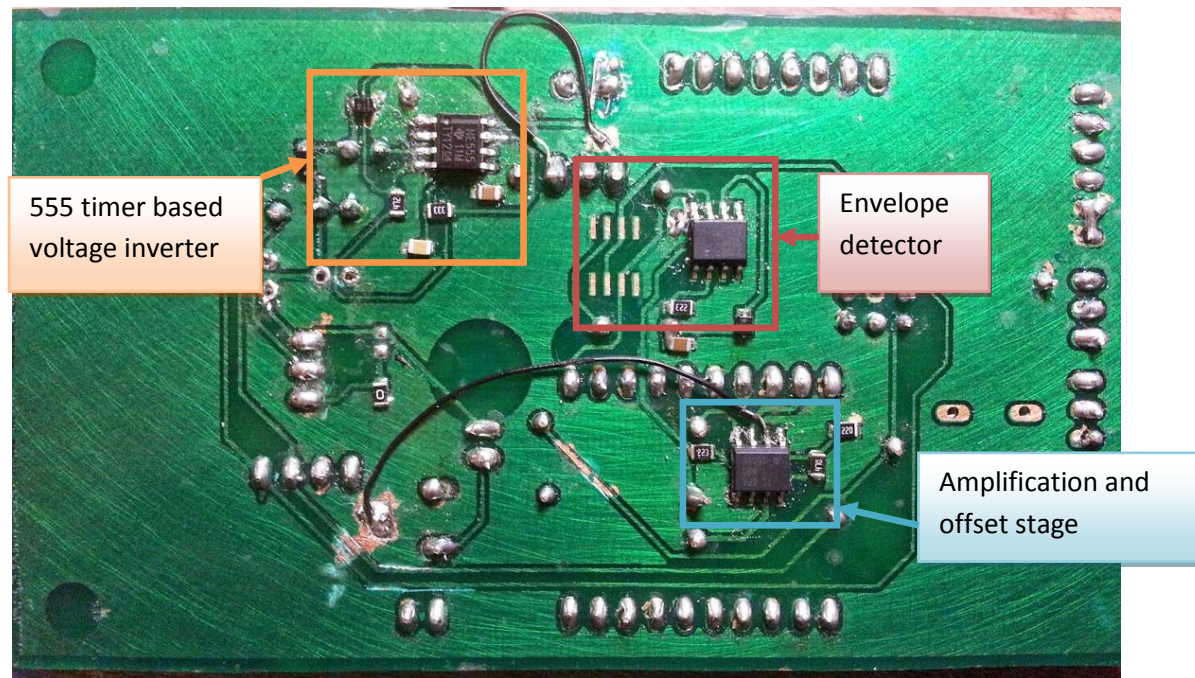


Figure A-2 fully assembled bottom side, note the black wires, which are fixes for incorrect circuit diagram.

#### 4.1.8 Conclusion

The hardware was probably the most complicated part of the project if the complexity is measured the time and effort put into it. Nonetheless it was valuable experience and skills which were learned over the course of developing such device will be useful throughout life. Especially, manufacturing PCBs at home. It might not be the cheapest method of manufacturing PCBs, but for sure the fastest. Please see appendix D for bill of materials.

## 4.2 Software

Final part of the GD system was to design user interface which can interface with the GD hardware and do pattern recognition. In the beginning it was intended to write an application for Android smart phone and PC. As mentioned in introduction only PC application was finished in time, due to unforeseen difficulties making the hardware.

### 4.2.1 Qt library

The application itself is written almost purely using Qt 5.2 library and C++. Qt is a massive cross-platform graphical framework designed to run on all devices and architectures. It supports Windows, Linux, Mac Android. Can be compiled for architectures as ARM, x86, x64 etc. Also it is LGPL licenses, which allows the usage of the library for free for commercial purposes, providing library isn't statically linked to the application. Basically, it was just matter of porting the PC application for Android, however you must write some Os and device specific code.

All Qt native classes start with letter "Q". If the class has extended Q\_OBJECT, it will most likely have signals. Signals are used in Qt as communication between classes. They can be viewed as events. A class which is parent of the particular class which sends the signal can connect to it and execute code like any other language would do for an event.

However, even though originally Qt library was designed only for GUI, over the decades it has grown to support many hardware functions, which are independent from the OS. Like Serial port, or Bluetooth, keyboard events etc. Obviously GD system extensively utilizes the QSerialPort class.

One more important thing to know about Qt to be able to understand the code is that, graphic forms are stored in UI files, which are basically auto generated code. Those UI files are loaded at the constructor in each of the corresponding classes. To change the graphical layout, it has to be done through Qt Creator to keep the consistency of the code. However, even if fixes are made by hand, providing no one opens it in Qt Creator, the changes will persist. More details about the structure of the actual application will be described later in the report.

## 4.2.2 Support Vector Machines (SVM)

### 4.2.2.1 Theory

Support vector machines (SVM) in machine learning are known as supervised learning algorithm that analyzes data and recognizes patterns. Mainly used for classification and regression. Basically by giving labeled data to the SVM learning algorithm, it will build a model that assigns the new examples in either one of the provided categories. SVM normally is represented as points in multi-dimensional space, and a straight line (in multiple-dimensions) are generated in a way so the provided learning data is separated as far as possible. Due to high dimensionality, it is very hard to imagine or debug an SVM. But for illustration purposes we can visualize 2D SVM, see figure 2.

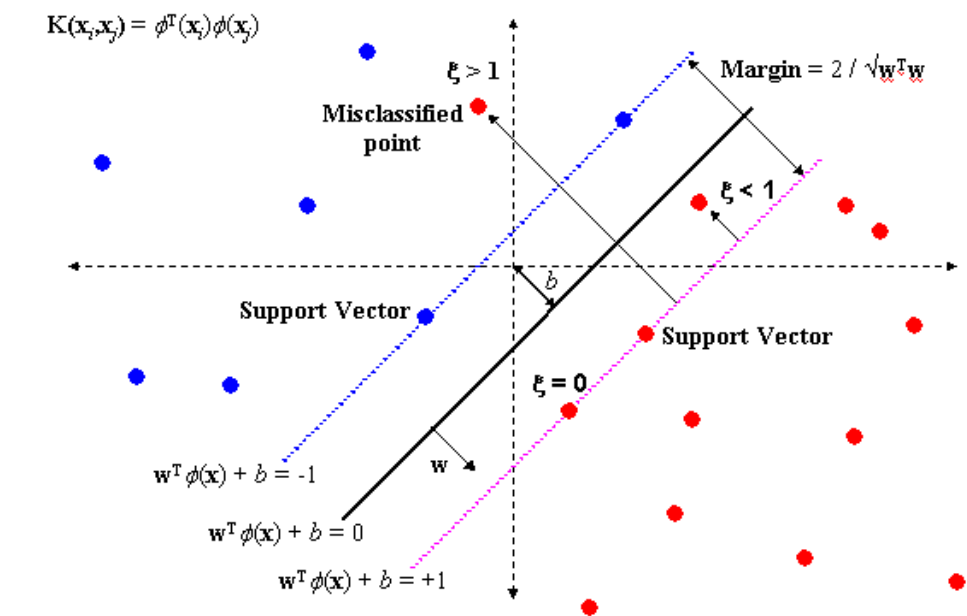


Figure A-1 SVM hyperplane example

Source: <http://research.microsoft.com/en-us/um/people/manik/projects/trade-off/svm.html>

As you can see in the Figure A-1, SVM has found the “most” optimal solution for classifying between two classes, in our case red and blue dots.



There are multiple techniques how to do multi-class classification. The one most commonly used is called One-Vs-All method. Were basically, for N amount of classes there would be N amount of boundary lines. In other words, each class will be tested separately against all the other classes as a whole. You can see in Figure A-2 and Figure A-3, how each class is checked against other classes, hence one-vs-all.

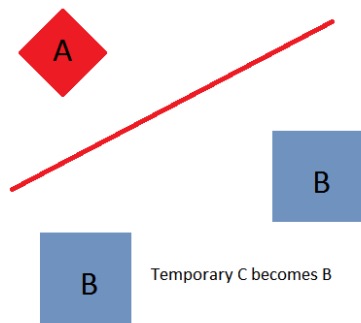


Figure A-2 Cluster A checked against cluster BC

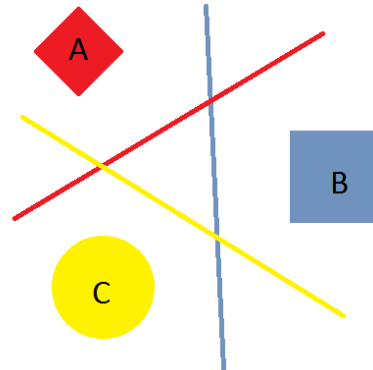


Figure A-3 Every cluster has a corresponding boundary line with all the other clusters.

Essentially, with one-vs-all method, we can construct very complicated boundary lines. For example the summation of the previous cluster example would produce a boundary line something like in Figure A-4

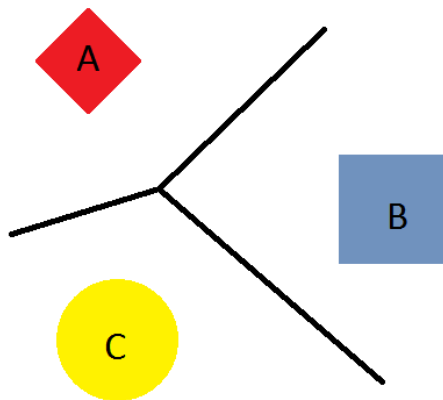


Figure A-4 constructing one boundary line by “summing” all individual one-vs-all SVM boundary lines

As mentioned before, normally SVM classification is done in multiple dimensions. Completely normal would be to have even over 160 dimensions for classifications. Due to human limitations it is impossible to perceive it, that’s why for illustrations purposes only 2 dimensions were described and illustrated. For example, GD system measures amplitude of the electrode in at least 160 frequencies, which means, that there will be at least 160 dimensions.

Also in the simplified example, only linearly separable examples were shown (except for Figure A-1), in real life there might be small amount of data overlapping, making clusters of data inseparable. That’s why cost margin parameter (C ) is used as control for the trade-off

between achieving low error on the training data. Large C values will “choose” small margins in hyperplane and small C values will do the exact opposite will make the SVM to look for larger margins. The C has to be chosen very carefully to avoid over fitting the data, which can lead to unreliable pattern recognition. Another important parameter to take into account is the kernel used for SVM classification, in the examples above linear kernel was assumed to be used. However, for more complicated data polynomial kernel would enable a more flexible decision boundary. The library, GD system is using will tend to use radial basis function kernel (RBF), which is essentially nonlinear kernel, where gamma can control the fitting of the decision boundary. The higher the gamma the more chance of over fitting the data, just like for small C.

The art of SVM is to carefully choose those two parameters, so the data is not over fitted and the decision boundary has the maximum distance between all data points.

#### 4.2.2.2 LibSVM

Since implementing SVM from scratch is a project on its own, GD system is using library called LibSVM. LibSVM library is written in pure C programming language, which gives the ability to be used in micro-controllers as well, providing enough memory is available.

LibSVM main features are:

- Different SVM formulations
- Efficient multi-class classification
- Cross validation for model selection
- Probability estimates
- Various kernels (including precomputed kernel matrix)
- Weighted SVM for unbalanced data
- Both C++ and Java sources
- GUI demonstrating SVM classification and regression
- Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, OCaml, LabVIEW, and PHP interfaces. C# .NET code and CUDA extension is available.
- It's also included in some data mining environments: RapidMiner, PCP, and LIONSolver.
- Automatic model selection which can generate contour of cross validation accuracy.

Source: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

The library itself comes with all the tools necessary for “teaching” the computer to recognize different touches. From programmer’s perspective, learning data has to be gathered and saved in specific way. Also the learned model has to be loaded in *smv\_model* structure before classification.

First of all, the structure of gathered data is as following

[id number] [data current position in case of GD]:[actual value] [data current position in case of GD]:[actual value] [data current position in case of GD]:[actual value] ...

Below you can see actual data stored in one of the SVM files gathered by GD system:

```
0 1:2.56434 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
0 1:2.56434 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
0 1:2.56434 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
0 1:2.56434 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
1 1:2.56434 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
1 1:2.56434 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
1 1:2.55808 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
1 1:2.56121 2:2.56434 3:2.56434 4:2.56434 5:2.56434 6:2.56434
```

As you can see the data consists of two gestures (ids 1 and 0) and each gesture contains 6 data points.

This data could be put through SVM learning algorithm to produce the model file. For that GD system is using a tool provided by the library, which is written in python. However, the user doesn't need to interact with the tool directly, because the python script is integrated in the application.

As mentioned earlier, the LibSVM expects from the programmer to load the model in its model structure. They do provide C code for constructing the model structure; GD application loads the file into memory and forwards the buffer to that part of the library. As you can see in the code below:

```
QDir dir;
//qDebug() << dir.absoluteFilePath("gesture.svm.model");
char buffer[1024];
strcpy(buffer, fileName.toStdString().c_str() );

if ( (model=svm_load_model(buffer)) == 0 )
{
    QMessageBox::critical(this, tr("Error"),
        tr("Can't open model file. "));
    exit(1);
}
```

*Model* is the variable is the svm\_model structure mentioned before.

The C value and gamma will be chosen by library automatically. However, those values can be provided manually in the library, but for the sake of simplicity, it was decided to trust the library for automatically calculating them. When the python script finishes the learning process, it will plot a graph for C vs gamma which were tested in the process this data can be used to analyze the accuracy of learned model. See Figure A-1 and Figure A-2.

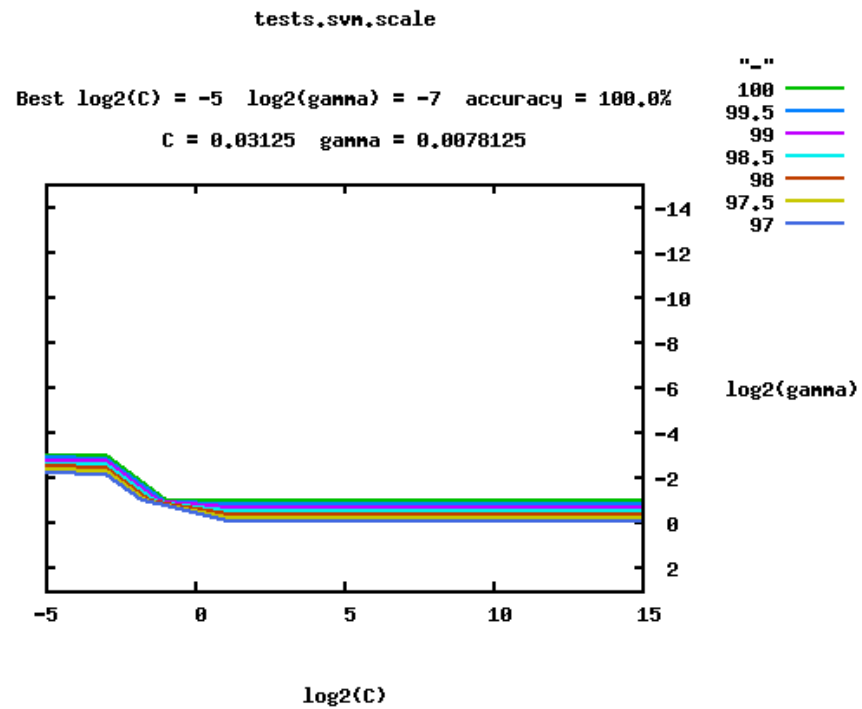


Figure A-1 Tested C vs gamma values. You can see that best C and gamma values produce 100% accuracy

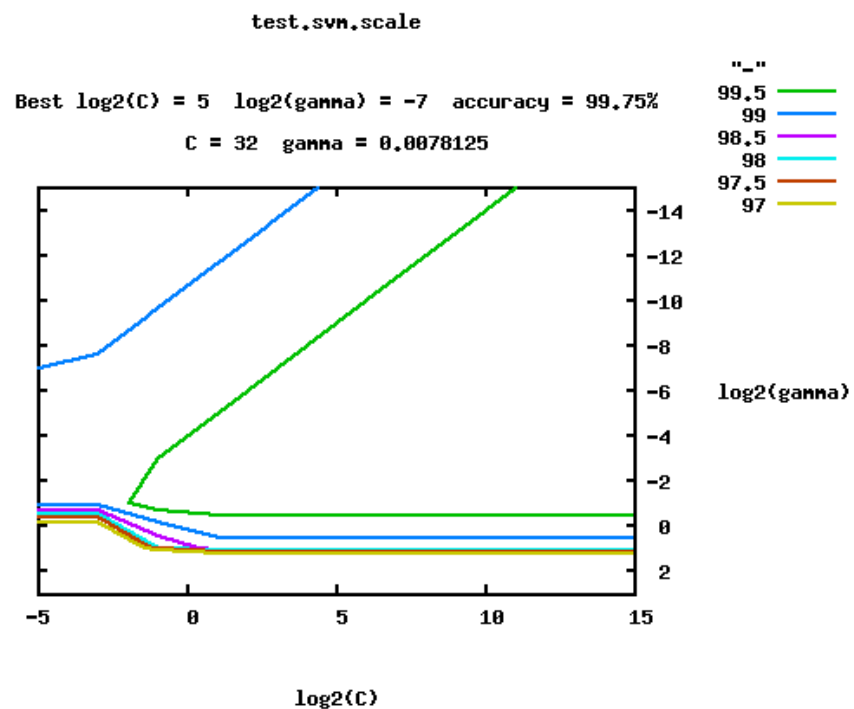


Figure A-2 Another plot C vs gamma, however, the best solution found was only 99.75% accurate. SO the data couldn't be fully fitted with the given maximum amount of iterations

After the model is loaded, by providing new data, which hasn't been taught, it should return the ID of the learned data closest to the new set of data. Also it is possible to invoke method which will return the confidence level of the classified set of data.

However before classifying the new set of data, they have to be properly stored in *svm\_node* structure and most importantly normalized between values -1 to 1. The normalization values (min and max for each X value) are given after learning has been finished in file ending *\*.svm.scale*.

After the all the necessary computation has been done on the new set of data, and then they can be passed to the LibSVM for classification, by using one of the functions below:

```
svm_predict_values(model, new_set_of_data, decVals);
```

```
svm_predict_probability(model, new_set_of_data, decVals);
```

The first function will return the ID of the closest cluster of data learned before. And the second function will basically return the probability of the classified data reassembling the closest data cluster. It can be viewed as a measure of confidence that the data were classified correctly, especially useful for assessing the performance of the system.

That mainly covers how LibSVM works and how it has been implemented in GD systems application. The GD application does create one more additional file with the same pattern: *\*.svm.names*. The names for the corresponding gesture are stored in this file. They are directly linked with the ID number. The first name in the file will correspond to data cluster with ID of 0, and so on.

### 4.2.3 Main application

The main PC application consists of 5 main classes which are not 3<sup>rd</sup> party libraries but written explicitly for GD system:

1. MainWindow
2. Learner
3. SerialTerminal
4. Settings
5. SimulateKeyboard

#### 4.2.3.1 MainWindow class

Just like in any Qt application everything begins at MainWindow, which is the first class constructed in the *main.cpp* file and executed. Afterwards MainWindow links to all the other classes necessary for the application – Settings, Learner, SerialTerminal, QSerialPort and SimulateKeyboard. The variables containing the pointer to instances for the classes listed before are stored as global variables in the MainWindow class. Also all the instances are created in MainWindow's constructor and signals connected to corresponding slots.

All pattern recognition is done inside the MainWindow class. Two methods are responsible for pattern recognition – *doSomeMagic()* and *enableMagic()*.



*enableMagic()* will open a file browser dialog, which will only show \*.model files. Basically, this method is responsible for loading the SVM model into the memory. There is nothing particularly special about the process itself, but in short it does the following functions:

1. Loads SVM model
2. Loads SVM range file for later data scaling
3. Loads SVM name file for id to gesture name mapping

However, *doSomeMagic()* method is responsible for actually classifying the new set of data and printing the output, that is the name of the recognized gesture. Also if *SimulateKeyboard* dialog is open, it will forward the gesture's ID which in turn will emulate a key press on the keyboard. Also this is the method, which will perform scaling of the new data from -1 to 1 and will forward it to the LibSVM function for prediction. This method is executed every time sufficient amount of data has been received from the serial port.

The serial communication in Qt is done first by constructing a *QSerialPort* object and then by opening user specified port with user specified baud rate. Those two properties are read from the *Settings* class. When serial object is created it can be opened by invoking *openSerialPort()* method in the *MainWindow* class. When a successful connection will be established with the GD hardware, first it will send reset command "q", wait for 150ms and send the new frequency sweep settings, which are specified by the user in *Settings* class. The settings are sent by sending "c" command to xmega. Exactly this is done by *sendDeviceSettings()* method in *MainWindow* class. Which utilizes the *sendSerialData()* method. That method just puts the data necessary for transmission on a buffer, which will be taken care of by *QSerialPort* class device write event.

Once successful connection is opened with GD hardware, an event, more precisely a signal will be listening to whether any data are available in the serial port's buffer. If so it will invoke *readMethod()*. This method will read all buffer, and extract data between two curly brackets "{ ... .. }". When the user specified amount of data is collected, the function will construct a vector of amplitude in V corresponding to the frequency. The voltage will be calculated at this stage, rather on the micro-controller. After vector of amplitudes vs frequency has been constructed it will be forwarded to plotting function. However, if *Learner* class has initiated data gathering, it will also store the vector into the file, corresponding the structure of SVM file described earlier. Finally, if a SVM model is loaded, it will forward the vector for classification, where it will be scaled and converted accordingly. When all of the above has been executed, the same method will measure how quick the data was collected.

The final method worth mentioning is the *plot()* method. For plotting GD application is using a free Qt library called *QCustomPlot*. It is a very basic plotting library with very limited functionality, there are better alternatives. But a better more functional library comes with the cost of complexity. In any case, the plotting library just needs to vectors to be forwarded – one for X axis and one for Y axis. The X vector in GD system corresponds to the frequency in kHz and Y vector to the amplitude in V. Also GD system provides some basic configuration, like recalculating the range of X and Y axis, so it doesn't change every time different scale of data is received. In other words, the range for X axis is fixed to the amount of frequencies

swept and Y axis to 5 V. When all properties are set and data forwarded to the plotting library, then you just need to invoke *replot()* method in the QCustomPlot class. Note, QCustomPlot library won't be described in more detailed than actually used, for more information refer to original author's homepage.

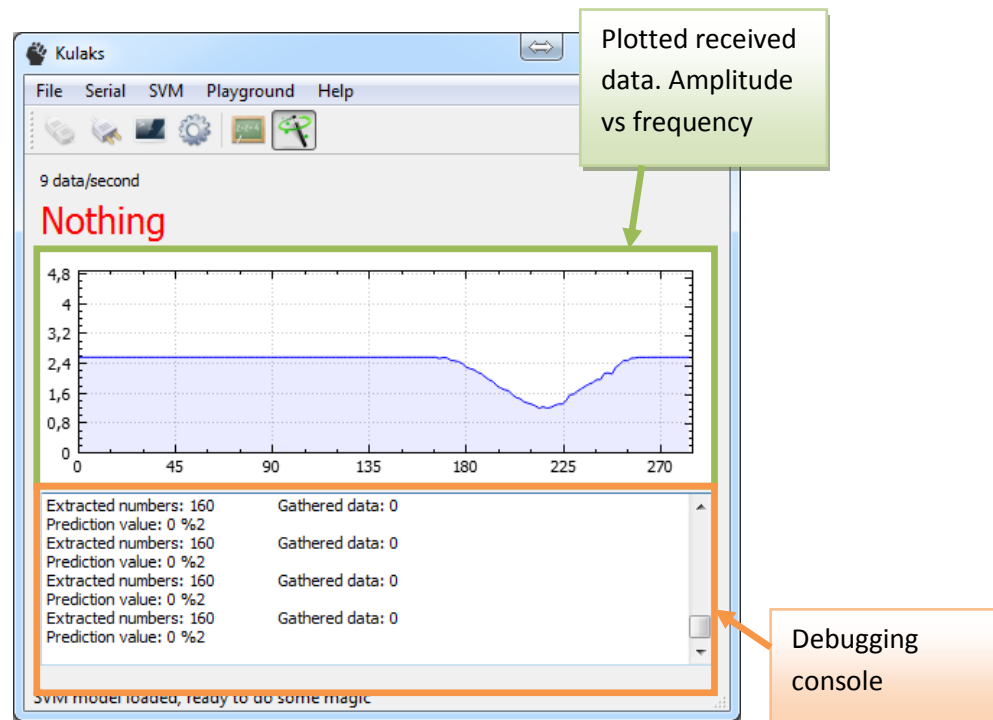


Figure A-1 Main application window, with plotting and recognized gesture in red font

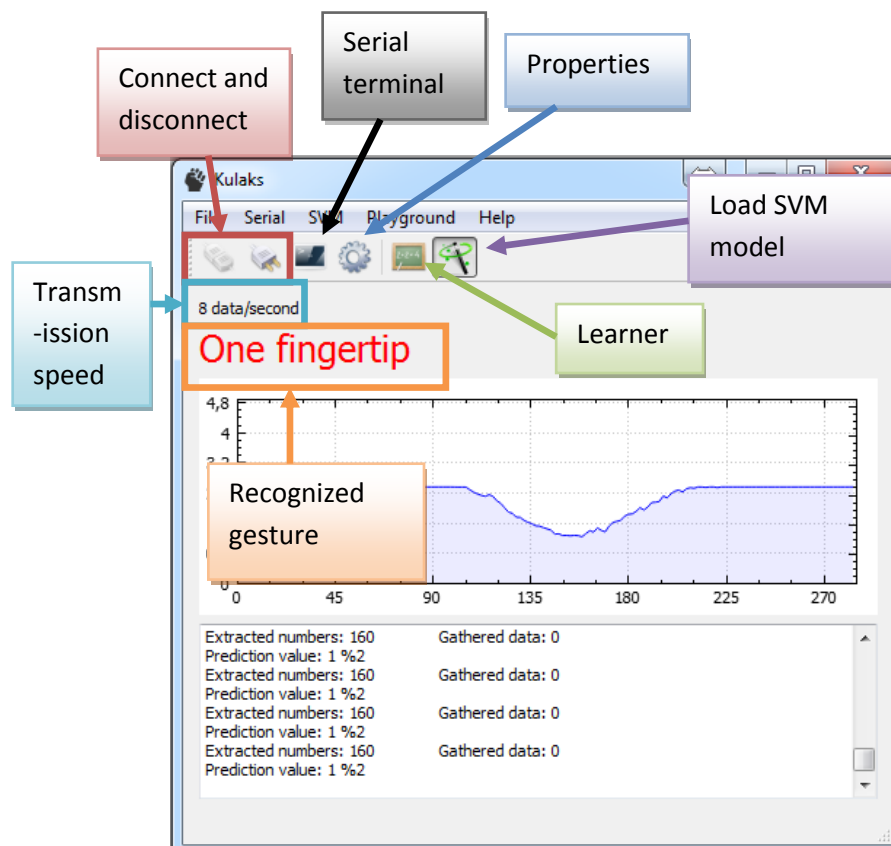


Figure A-2 Another gesture recognized by the system

#### 4.2.3.2 *Learner class*

This class is responsible for sending signal to the MainWindow class that user has initiated the data gathering process while specifying ID and name of the gesture. And also *Learner* class is responsible for executing the python script necessary for constructing the SVM model file.

When user presses button “Collect” *startGathering()* method will be invoked. Which will check whether a file for saving the gathered data is specified and then it will emit a custom signal *gatherData()* which will be received by MainWindow. The signal will also hold the value of the how many data needs to be gathered, the ID of the gesture and the name of the gesture.

When user presses the “Learn” button *learn()* method will be invoked. Which will first read from the Settings where python executable is located on the computer (must be provided in Settings class). Afterwards it will construct a QProcess to run the python script for learning on a separate thread, while listening when any data is returned from the process or it has finished executing. When execution is finished it will also check its exit code, if it’s bigger than 0, a message box will appear showing the error! See the Figure A-1 for this class GUI.

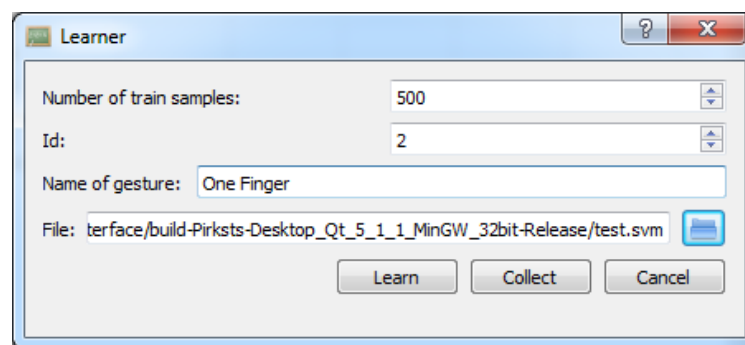


Figure A-1 Learner class dialog

#### 4.2.3.3 *Settings class*

Settings class is responsible for writing configuration into windows user registry. More specifically the settings will be stored in following location:

`"HKEY_CURRENT_USER\\Software\\Rastro\\Kulaks"`

Also that location will have sub folder of “Serial” and “Dev”. “Serial” will hold registry entries for port the GD device is connected to and baudrate. The “Dev” folder will hold information about the initial frequency the maximum frequency and amount of samples necessary to be measured. These settings will be sent to the GD device when the application first connects to it. In the main folder, there will be entry for “python” and “gnuplot” which are basically path to those two executables. See Figure A-1, for the Settings dialog.

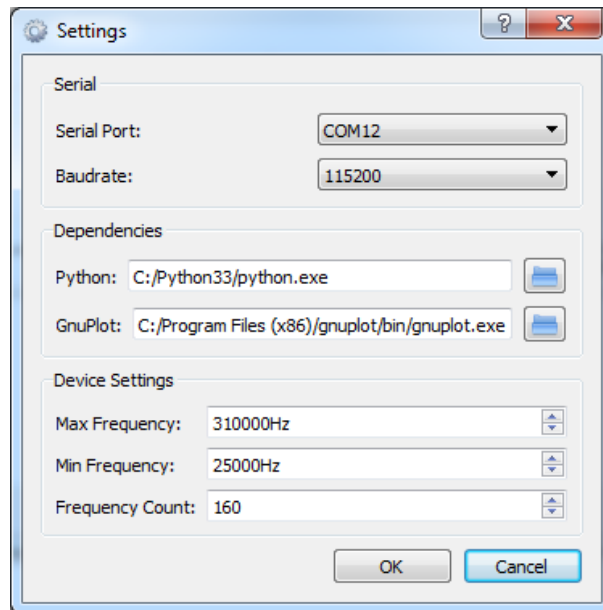


Figure A-1 Settings dialog

Every time the dialog will be opened it will load the settings from the registry. Also since settings are stored in user profile, multiple users can have different settings set up for GD system. When “OK” button is pressed, the main application will send the new device settings to the GD hardware and restart the frequency sweep sequence.

#### 4.2.3.4 *SerialTerminal class*

SerialTerminal class is basically a fully functional serial terminal, hence the name. Through this dialog it is possible to interact with the GD hardware manually, without the need for connecting to it with a different application like putty. The class itself is listening to key release event, if such event is detected it will send the character to the GD hardware. To write to the terminal window method *writeToTerminal(QString text)* must be invoked. This is done so by MainWindow class, if this dialog is visible, otherwise MainWindow won’t send any data to this class. See Figure A-1 for a terminal dialog screenshot.



```

    INPUT ip;
    // Set up a generic keyboard event.
    ip.type = INPUT_KEYBOARD;
    ip.ki.wScan = 0; // hardware scan code for key
    ip.ki.time = 0;
    ip.ki.dwExtraInfo = 0;

    // Press the "A" key
    ip.ki.wVk = ui->keyMap->topLevelItem(g)->text(1).toInt(); // virtual-key code from
settings
    ip.ki.dwFlags = 0; // 0 for key press
    SendInput(1, &ip, sizeof(INPUT));

    // Release the key
    ip.ki.dwFlags = KEYEVENTF_KEYUP; // KEYEVENTF_KEYUP for key release
    SendInput(1, &ip, sizeof(INPUT));

```

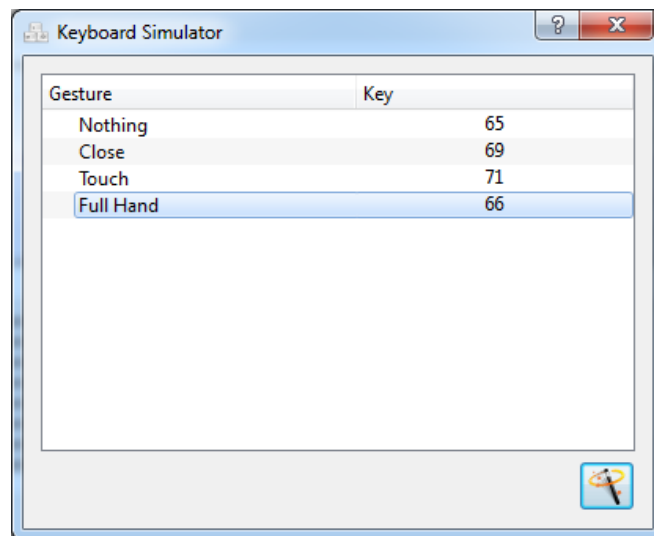


Figure A-1 Keyboard simulator configuration dialog, if this dialog is open while SVM is performing pattern recognition it will notify this class

If SVM is performing pattern recognition, the MainWindow class will invoke *gesture(...)* method. This method will check which key need to be emulated with the received gesture and perform such action.

For the full application code see Appendix E. Note to be able to run the Qt application, atleast Qt 5.2 redistributables must be installed on the system.

## 5 Testing

While testing the system, it was realized that the results should be more reliable more time is necessary, and further testing is necessary in the future. However the first results suggest that different environments produce different readings from the sensor. Even by unplugging the power supply from laptop will produce completely different frequency responses. See Figure A-1 and Figure A-2

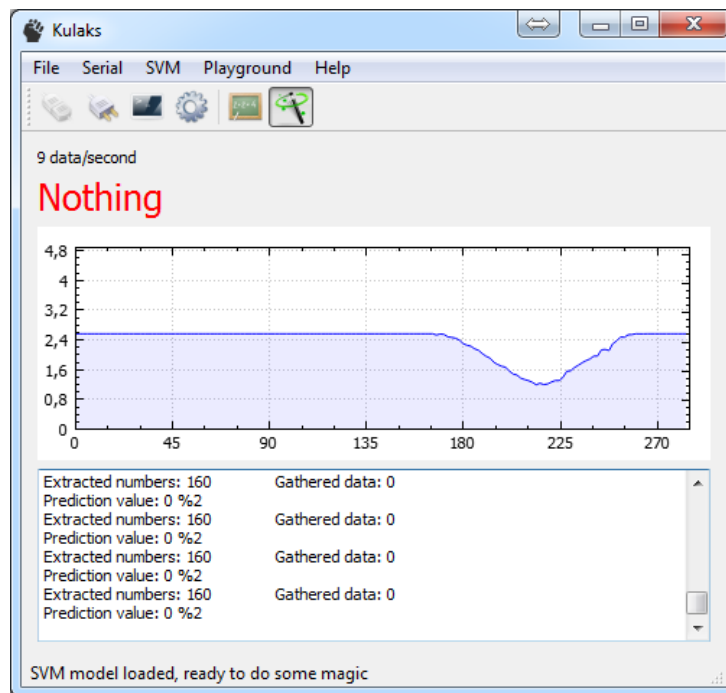


Figure A-1 Idle readings while laptop's power supply is plugged in

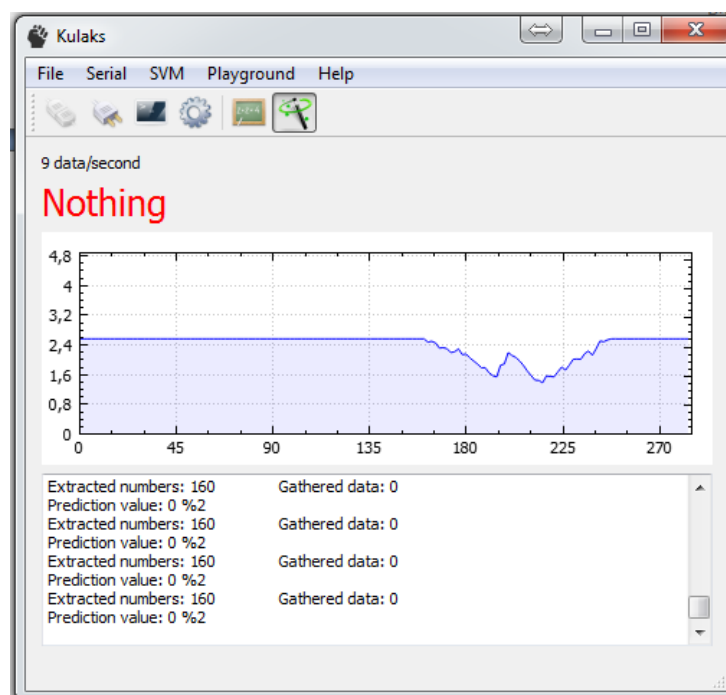


Figure A-2 Idle reading when laptop runs only on battery. The signal immediately becomes slightly altered.

Also due to previous experience with making sensors it was tested whether mobile phone making a call will result in introducing noise in the system. The result was that mobile phone call doesn't have any impact on the sensor.

To measure the speed of the whole system every second number of conversions were collected over the course of 162 seconds, which in turn produced 162 data points. See Figure A-3

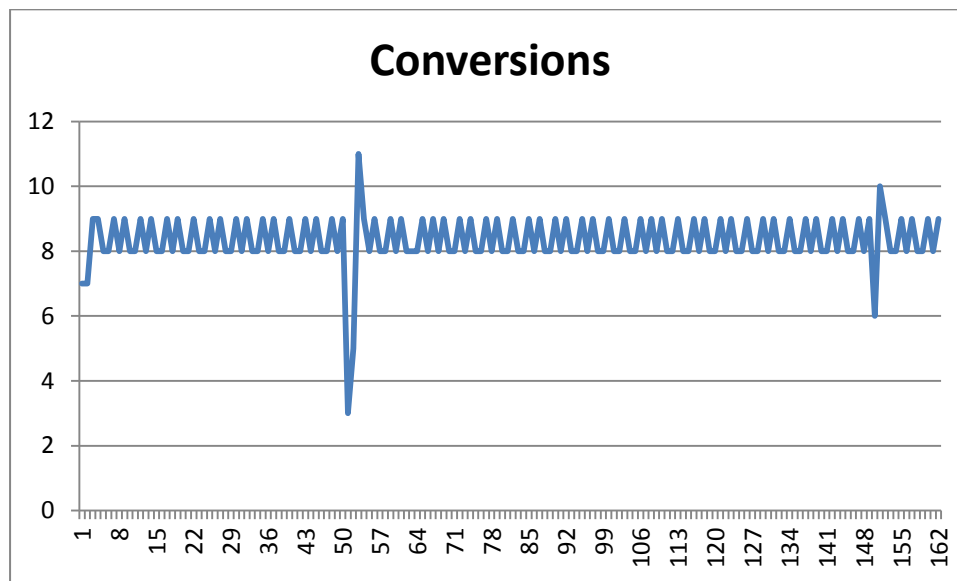


Figure A-3 Conversions a second over a course of 162 seconds

On average the GD system is able to perform 8.35 conversions a second. This is rather quick considering 160 data points are sent to the PC on every conversion.

More precisely,  $160 \cdot 6 + 159 + 2 = 1121$  bytes are sent on every reading at the worst case scenario. 160 data points are in floating format %0.2f. In worst case scenario it will be 3 digit number with two decimal points. + white spaces after each number except for the last digit 159 and finally we add 2 curly brackets.

The average was taken while SVM was performing pattern recognition, conclusion, it has little or no impact at all on the GD systems performance. The average ended up being 8.35 over a course of 197 seconds.

## 6 Conclusion

The GD system developed over the course of year was successfully able to increase the amount of gestures detected by human interaction with conductive materials. Normally, capacitive sensors can only detect whether an object is touched or not. GD system allows more complicated gestures to be detected by basically measuring the amount of skin touching it. From this information the sensor is able to distinguish whether one or two fingers are placed on the sensor, or if a hand is close to it.

In addition, the GD system hardware is easily expandable providing a workable platform for further improvements due to easy access for all power lines, fast micro-controller and 8 GPIO pins. The software and firmware is written by using OOP convention, which should make the code easier to read for another engineer. And more boards can be manufactured in home environment.



However, the GD system hasn't reached its full potential, especially regarding the sensitivity of the sensor. One hypothesis would be, if the current could be increased for the excitation AC signal, it may render the sensor to be more sensitive, because it would have more energy to propagate through internal body fluids. Probably amplitude of the signal could be increased as well to produce the same effect. Testing this hypothesis would be the next logical step to make.

Also further system testing is needed to obtain more reliable data, as mentioned earlier. Especially important would be to test the performance of the system with multiple people. With this test system's accuracy could be measured,

Overall, it was quite useful experience when working on this project. It allowed to further strengthen knowledge regarding machine learning and application developments. Also PCB designing, manufacturing and soldering are skills which will be useful skills for further work in this field. An important lesson learned, is that one thing is to construct a prototype on a bread board and completely another thing is to finalize and make a bug free device. The same principle applies for developing application. It is very important not to underestimate the complexity of the project and also be prepared for many failures until the device can be made working.

## **Appendices**

## A. Appendix

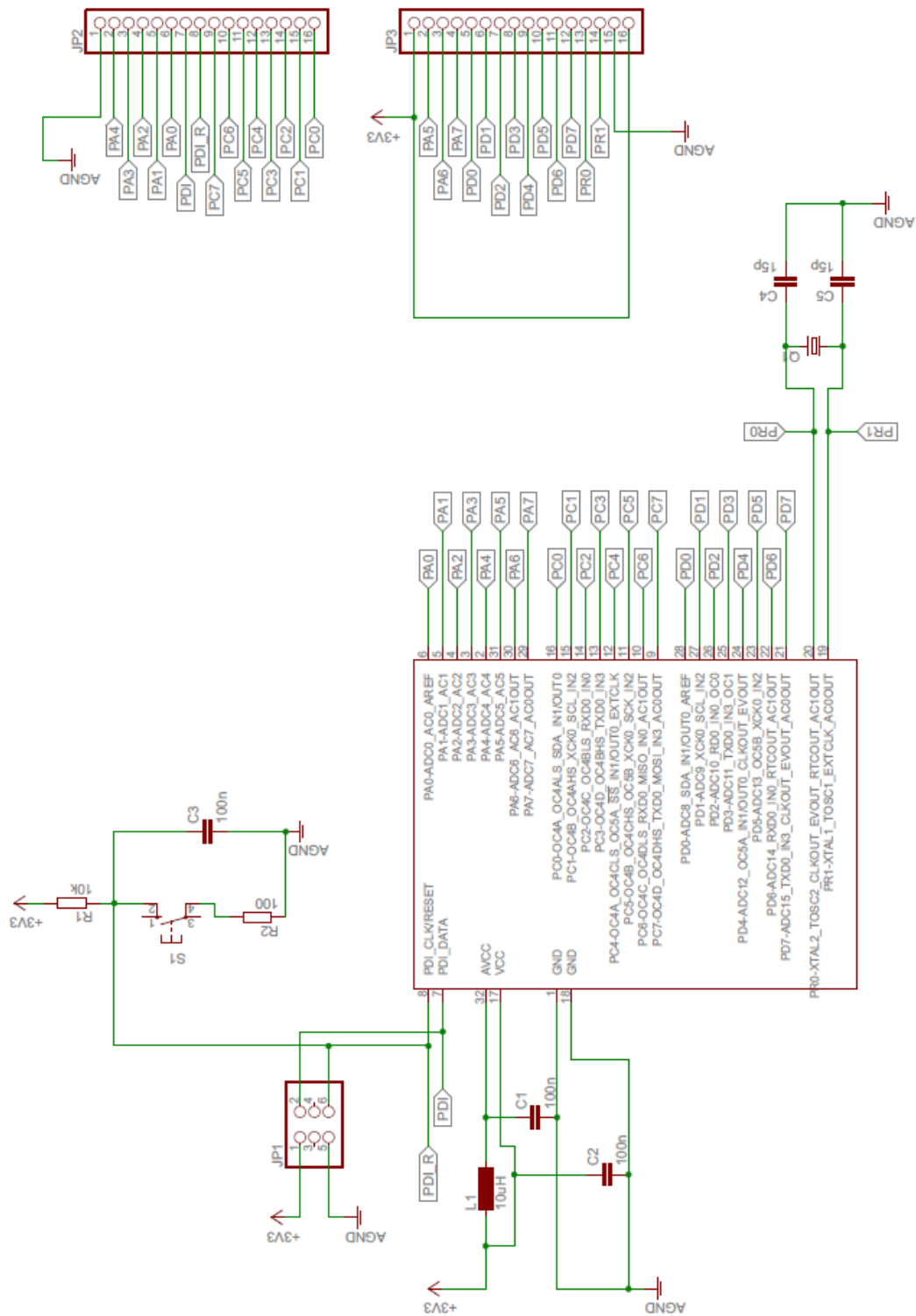
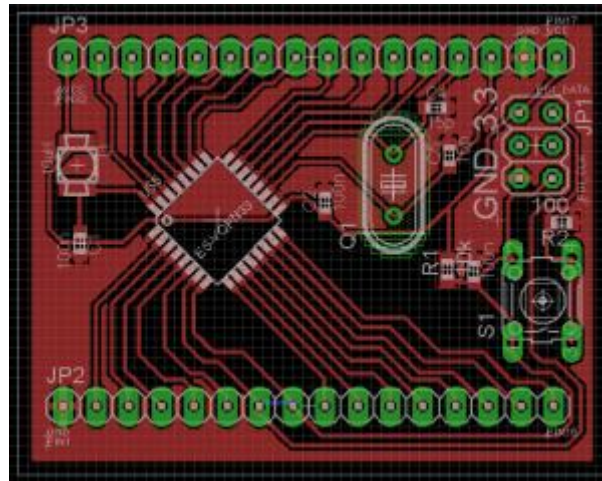


Figure A-1 Circuit diagram for the xmega16e5 breakout board



**Figure A-2 PCB design of the xmega16e5 breakout board**



**Figure A-3 Manufactured breakout board**

### *Kulaks\_firmware.cpp file*

```
/*
 * Kulaks_firmware.cpp
 *
 * Created: 2014.04.14. 18:19:53
 * Author: Raivis
 */

#define F_CPU 32000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include "SerialC.h"
#include "AD9850.h"

//For function generator
#define W_CLK 0
#define FQ_UD 1
#define DATA 2
#define RESET 3

//For sweeping DEFAULT VALUES, kulaks pc software can change it
#define N 160 //How many frequencies

#define MIN_FREQ 25000
#define MAX_FREQ 310000
#define SWEEP_STEP (MAX_FREQ-MIN_FREQ)/N

extern "C"{
    FILE * uart_str;
}

//Global variables
SerialC serial;
AD9850 funcGenerator(W_CLK, FQ_UD, DATA, RESET);

uint32_t singleStep;
uint32_t maxFreq;
uint32_t initialFreq;
uint32_t frequency = 1000;
uint16_t res = N;

float *results; //Filtered result buffer
float *freq; //Filtered result buffer

bool sweep;

//Prototype functions
void setUp32MhzInternalOsc();
static int uart_putchar (char c, FILE *stream);
int uart_getchar(FILE *stream);
void setUpSweep(uint32_t min, uint32_t max, uint16_t nRes);
void doSweep();
void setUpADC();
int readADC();

int main(void)
{
    setUp32MhzInternalOsc();

    //Overwrite standard stdout;
    uart_str = fdevopen(uart_putchar, uart_getchar); //send , receive functions
    stdout = stdin = uart_str;
```

```

    //Setup Function Generator
    PORTC_DIR |= ((1<<W_CLK) | (1<<FQ_UD) | (1<<DATA) | (1<<RESET)); //Set the direction
for function generator pins
    //PORTC_DIR = 0xff;
    //PORTC_DIR |= ((1<<4) | (1<<5));

    //printf("PORTA_DIR: %d\n", PORTA_DIR);
    funcGenerator.init();
    funcGenerator.doReset();

    funcGenerator.osc(2000,0);

    setUpSweep(MIN_FREQ, MAX_FREQ, N);

    //Setup ADC
    PORTA_OUT = 0x00;
    setUpADC();

    sweep = false;

    printf("Ready!\n");

    // PORTC_OUTSET = PIN4_bm;
while(1)
{
    //funcGenerator.osc(2000,0);
    //_delay_ms(200);
    //PORTC_OUT = ~PORTC_OUT;
    //_delay_ms(200);
    if(serial.available())
    {
        char inChar = serial.usart_receiveByte();
        if (inChar == 't')
        {
            char str [80];
            int i;
            printf ("Enter your family name: ");
            scanf ("%79s",str);
            printf ("Enter your age: ");
            scanf ("%d",&i);
            printf ("Mr. %s , %d years old.\n",str,i);
            printf ("Enter a hexadecimal number: ");
            scanf ("%x",&i);
            printf ("You have entered %#x (%d).\n",i,i);
        }
        else if(inChar == 'n')
        {
            frequency += (float)singleStep*XTAL_MHZ/FREQ_FACTOR;
            funcGenerator.osc(frequency, 0);
            printf("Frequency: %0.2f kHz\n", frequency/1000.0);
        }
        else if(inChar == 'b')
        {
            frequency -= (float)singleStep*XTAL_MHZ/FREQ_FACTOR;
            funcGenerator.osc(frequency, 0);
            printf("Frequency: %0.2f kHz\n", frequency/1000.0);
        }
        else if(inChar == 'r')
        {
            frequency = (float)initialFreq*XTAL_MHZ/FREQ_FACTOR;
            funcGenerator.osc(frequency, 0);
            printf("Frequency: %0.2f kHz\n", frequency/1000.0);
        }
        else if(inChar == 'c')
        {
            printf("New sweep [min,max,n]:");
            uint32_t min;
            uint32_t max;
            uint16_t _n;

            char line[100];
            uint16_t i = 0;

```

```

        &min, &max, &n)!=1 )

        //if( fgets(line,100,stdin) && sscanf(line,"%lu %lu %u",
        //{
            //min = 0;
            //max = 0;
            //_n = 0;
        //}
        scanf(" %lu %lu %u", &min, &max, &n);
        //scanf(" %lu",&min);

        printf("Received %lu %lu %u\n", min,max,_n);
        //printf("Received %s %d\n", line, i);
        setUpSweep(min, max, _n);
    }
    else if(inChar == 's') //Stop
    {
        sweep = false;
    }
    else if(inChar == 'd') //Do sweep
    {
        sweep = true;
    }
    else if(inChar == 'q')
    {
        //RESETS the whole AVR
        CCP = CCP_IOREG_gc; //Trigger protection mechanism
        RST_CTRL = RST_SWRST_bm;
    }
}

if(sweep == true)
    doSweep();

}

}

void doSweep()
{
    uint16_t d;
    frequency = initialFreq;
    funcGenerator.oscInt(frequency);

    serial.sendChar('{');

    for(d = 0; d<res; d++)
    {
        uint16_t v = readADC();
        frequency = frequency + singleStep;
        funcGenerator.oscInt(frequency);

        results[d] = results[d]*0.5+(float)(v)*0.5; //filter results

        printf("%0.2f ", results[d]);
    }

    serial.sendChar('}');
    serial.sendChar('\n');
}

void setUpADC()
{
    ADCA_CH0_CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc; //Configure ADC conversion as single
ended
    ADCA_CH0_MUXCTRL = ADC_CH_MUXPOS_PIN1_gc; // Set ADC to pin1, or PA1, because PA0 =
AREFA

    //Sample rate = 166666 samples.
    // 1000000/(0.5*(12+0)+0) = 166666
    // See page 351

```

```

        ADCA_CTRLB = ADC_CURRLIMIT_LOW_gc | ADC_RESOLUTION_12BIT_gc; //Low current limit, up
to 225ksp/s, and 12 bit resolution
        //ADCA_CTRLB = ADC_FREERUN_bm;
        //ADCA_REFCTRL = ADC_REFSEL_INTVCC_gc; // Internal reference VCC/1.6
        ADCA_REFCTRL = ADC_REFSEL_AREFA_gc; // External reference AVCC - 0.6
        ADCA_PRESCALER = ADC_PRESCALER_DIV32_gc; // Prescaler div 4

        ADCA_CTRLA = ADC_ENABLE_bm; //Enable ADC
    }

    int readADC()
    {
        ADCA_CTRLA |= ADC_START_bm;
        //ADCA_INTFLAGS = 0x01;
        while(!(ADCA_CH0_INTFLAGS & ADC_CH0IF_bm));
        return ADCA_CH0_RES;
    }

    void setUpSweep(uint32_t min, uint32_t max, uint16_t nRes)
    {
        res = nRes;

        maxFreq = (uint32_t)max*FREQ_FACTOR/XTAL_MHZ;
        initialFreq = (uint32_t)min*FREQ_FACTOR/XTAL_MHZ;
        singleStep = (maxFreq-initialFreq) / res;

        printf ("REs %d %d", res, nRes);

        free(results);
        results = (float*) malloc(res+1);
        free(freq);
        freq = (float*) malloc(res+1);

        printf("\nSweep initialized:\nMin: %0.2f kHz\nMax: %0.2f kHz\nStep:%0.2f
kHz\n\n",min/1000.0,max/1000.0,((max-min)/res)/1000.0);
    }

    void setUp32MhzInternalOsc()
    {
        OSC_CTRL |= OSC_RC32MEN_bm; //Setup 32Mhz crystal

        while(!(OSC_STATUS & OSC_RC32MRDY_bm));

        CCP = CCP_IOREG_gc; //Trigger protection mechanism
        CLK_CTRL = CLK_SCLKSEL_RC32M_gc; //Enable internal 32Mhz crystal
    }

    static int uart_putchar (char c, FILE *stream)
    {
        if (c == '\n')
            uart_putchar('\r', stream);

        serial.sendChar(c);

        return 0;
    }

    int uart_getchar(FILE *stream)
    {
        while( !serial.available() ); //Interesting DRIF didn't work.
        char data = serial.read();
        if(data == '\r')
            data = '\n';
        uart_putchar(data, stream);
        return data;
    }

```



### *SerialC.h cpp file*

```
#ifndef SERIALC_H
#define SERIALC_H

class SerialC
{
    public:
        SerialC(void);
        void setUpSerial();
        void sendChar(char c);
        void sendString(char *text);
        char usart_receiveByte();

        uint8_t available();
        char read();
};

#endif //SERIALC_H
```

### *SerialC.cpp file*

```
#ifndef F_CPU
#define F_CPU 32000000UL // Orangutans run at 20 MHz
#endif //!F_CPU

#include <avr/io.h>
#include <stdio.h>
#include "SerialC.h"

SerialC::SerialC(void)
{
    setUpSerial();
}

void SerialC::setUpSerial()
{
    //For the sake of example, I'll just REMAP the USART pins from PC3 and PC2 to PC7 and
    PC6
    PORTC_REMAP |= 0x16; //See page 152 in datasheet, remaps the USART0

    PORTC_OUTSET = PIN7_bm; //Let's make PC7 as TX
    PORTC_DIRSET = PIN7_bm; //TX pin as output

    PORTC_OUTCLR = PIN6_bm;
    PORTC_DIRCLR = PIN6_bm; //PC6 as RX

    // Baud rate selection
    // BSEL = (32000000 / (2^0 * 8*115200)) -1 = 34.7222 -> BSCALE = 0
    // FBAUD = ( (32000000)/(2^0*8(34+1)) = 114285.71 -> it's alright

    USARTC0_BAUDCTRLB = 0; //Just to be sure that BSCALE is 0
    USARTC0_BAUDCTRLA = 0x22; // 207

    //Disable interrupts
    USARTC0_CTRLA = 0;
    //8 data bits, no parity and 1 stop bit
    //USARTC0_CTRLC = USART_CMODE0_bm | USART_PMODE0_bm | USART_CHSIZE_8BIT_gc;
    USARTC0_CTRLC = USART_CHSIZE_8BIT_gc;

    //Enable receive and transmit
    USARTC0_CTRLB = USART_TXEN_bm | USART_CLK2X_bm | USART_RXEN_bm; // And enable high
    speed mode
}

void SerialC::sendChar(char c)
```

```

{
    while( !(USARTC0_STATUS & USART_DREIF_bm) ); //Wait until DATA buffer is empty
    USARTC0_DATA = c;
}

void SerialC::sendString(char *text)
{
    while(*text)
    {
        sendChar(*text++);
    }
}

char SerialC::usart_receiveByte()
{
    while( !(USARTC0_STATUS & USART_RXCIF_bm) ); //Interesting DRIF didn't work.
    return USARTC0_DATA;
}

uint8_t SerialC::available()
{
    return (USARTC0_STATUS & USART_RXCIF_bm);
}

char SerialC::read()
{
    return USARTC0_DATA;
}

```

## AD9850.h file

```
#ifndef AD9850_H
#define AD9850_H

//Frequency of your crystal oscillator (CLKIN input pin 9 in datasheet), measured in MHz.
// This reference frequency must be higher than 1MHz.
#define XTAL_MHZ 125

//Relationship value between actual frequency and 32-bit word sent in the serial streaming
#define FREQ_FACTOR 4294.967295

#define HIGH 10
#define LOW 0

#define _PORT_N PORTC_OUT

class AD9850
{
public:
    AD9850(uint8_t W_CLK, uint8_t FQ_UD, uint8_t DATA, uint8_t RESET);
    void init();
    void doReset();
    void osc(double frequency, double phase);
    void oscInt(unsigned long y);

private:
    uint8_t _W_CLK;
    uint8_t _FQ_UD;
    uint8_t _DATA;
    uint8_t _RESET;

    void digitalWrite(uint8_t pin, uint8_t state);
};

#endif //AD9850_H
```

## AD9850.cpp file

```
#ifndef F_CPU
#define F_CPU 32000000UL // Orangutans run at 20 MHz
#endif //F_CPU

#include <avr/io.h>
#include "AD9850.h"

AD9850::AD9850(uint8_t W_CLK, uint8_t FQ_UD, uint8_t DATA, uint8_t RESET)
{
    this->_W_CLK = W_CLK;
    this->_FQ_UD = FQ_UD;
    this->_DATA = DATA;
    this->_RESET = RESET;
}

void AD9850::init()
{
    digitalWrite(_W_CLK, LOW);
    digitalWrite(_FQ_UD, LOW);
    digitalWrite(_DATA, LOW);
    digitalWrite(_RESET, LOW);
}

void AD9850::doReset()
{
    digitalWrite(_RESET, LOW);
    digitalWrite(_RESET, HIGH);
    digitalWrite(_RESET, LOW);
}
```

```

        digitalWrite(_W_CLK, LOW);
        digitalWrite(_W_CLK, HIGH);
        digitalWrite(_W_CLK, LOW);

        digitalWrite(_FQ_UD, LOW);
        digitalWrite(_FQ_UD, HIGH);
        digitalWrite(_FQ_UD, LOW);

        osc(0,0);

        //digitalWrite(4, LOW);
    }

    void AD9850::osc(double frequency, double phase)
    {
        long y=(long)frequency*FREQ_FACTOR/XTAL_MHZ;
        while(phase>360)
            phase-=360;
        long z=phase/11.5;

        //    Serial.println("Frequency word:");
        //    Serial.println(y);

        int i;

        //Frequency 32-bit word
        for (i=0;i<32;i++){
            digitalWrite(_DATA, (y>>i) & 0x01);
            digitalWrite(_W_CLK, HIGH);
            digitalWrite(_W_CLK, LOW);
        }

        //control bit #1, control bit #2 and Power off, all to low
        digitalWrite(_DATA, LOW);
        digitalWrite(_W_CLK, HIGH);
        digitalWrite(_W_CLK, LOW);
        digitalWrite(_W_CLK, HIGH);
        digitalWrite(_W_CLK, LOW);
        digitalWrite(_W_CLK, HIGH);
        digitalWrite(_W_CLK, LOW);

        //phase 5-bit word
        for (i=0;i<5;i++){
            digitalWrite(_DATA, (z>>i) & 0x01);
            digitalWrite(_W_CLK, HIGH);
            digitalWrite(_W_CLK, LOW);
        }

        digitalWrite(_FQ_UD, HIGH);
        digitalWrite(_FQ_UD, LOW);

        //digitalWrite(4, HIGH);
    }

    void AD9850::oscInt(unsigned long y)
    {
        long z = 0;

        int i;

        //Frequency 32-bit word
        for (i=0;i<32;i++){
            digitalWrite(_DATA, (y>>i) & 0x01);
            digitalWrite(_W_CLK, HIGH);
            digitalWrite(_W_CLK, LOW);
        }

        //control bit #1, control bit #2 and Power off, all to low
        digitalWrite(_DATA, LOW);
        digitalWrite(_W_CLK, HIGH);
        digitalWrite(_W_CLK, LOW);
    }

```

```

digitalWrite(_W_CLK, HIGH);
digitalWrite(_W_CLK, LOW);
digitalWrite(_W_CLK, HIGH);
digitalWrite(_W_CLK, LOW);

//phase 5-bit word
for (i=0;i<5;i++){
    digitalWrite(_DATA, (z>>i) & 0x01);
    digitalWrite(_W_CLK, HIGH);
    digitalWrite(_W_CLK, LOW);
}

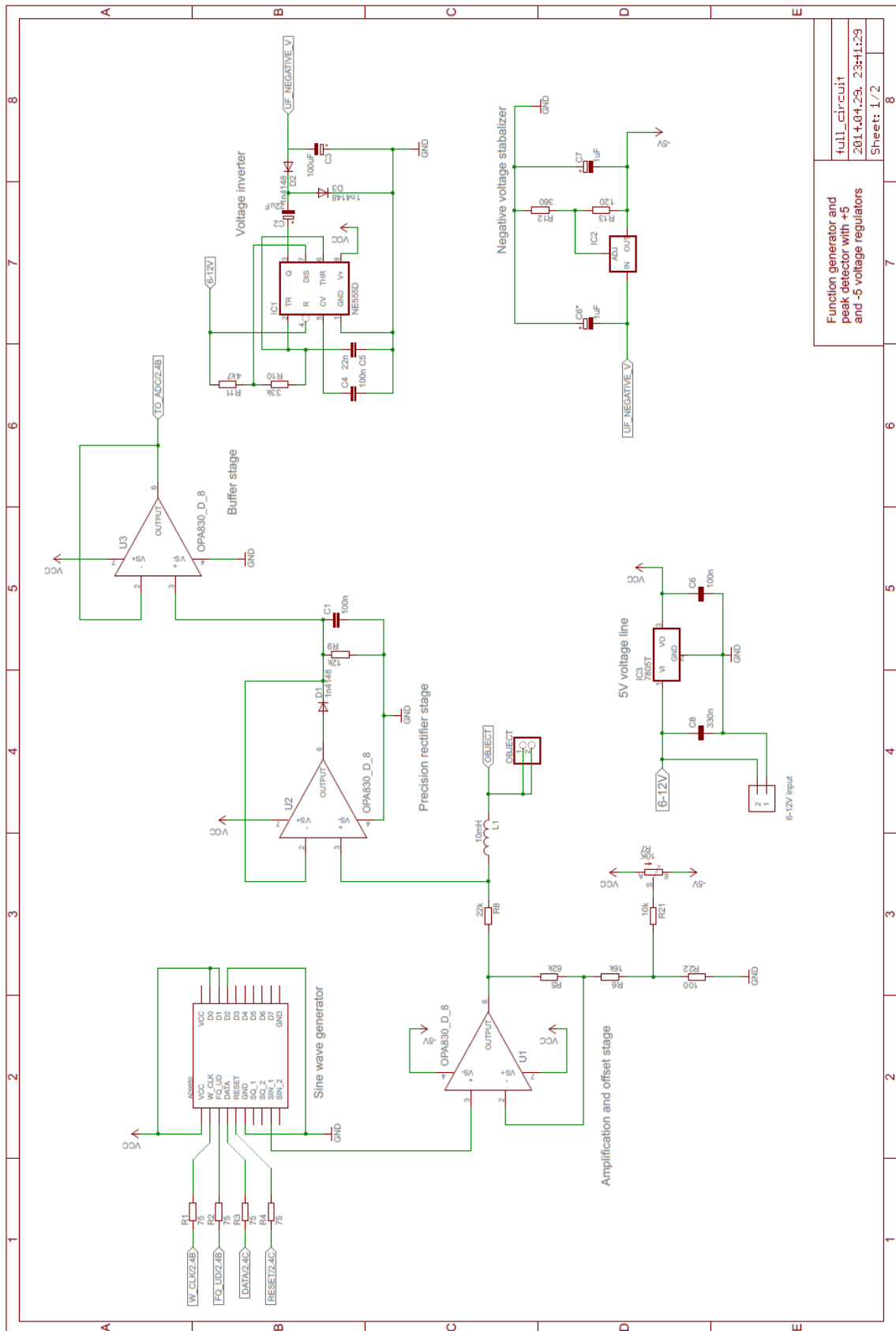
digitalWrite(_FQ_UD, HIGH);
digitalWrite(_FQ_UD, LOW);

//digitalWrite(5, HIGH);
}

void AD9850::digitalWrite(uint8_t pin, uint8_t state)
{
    if(state > 0)
        _PORT_N |= (1<<pin);
    else
        _PORT_N &=~(1<<pin);
}

```

## C. Appendix

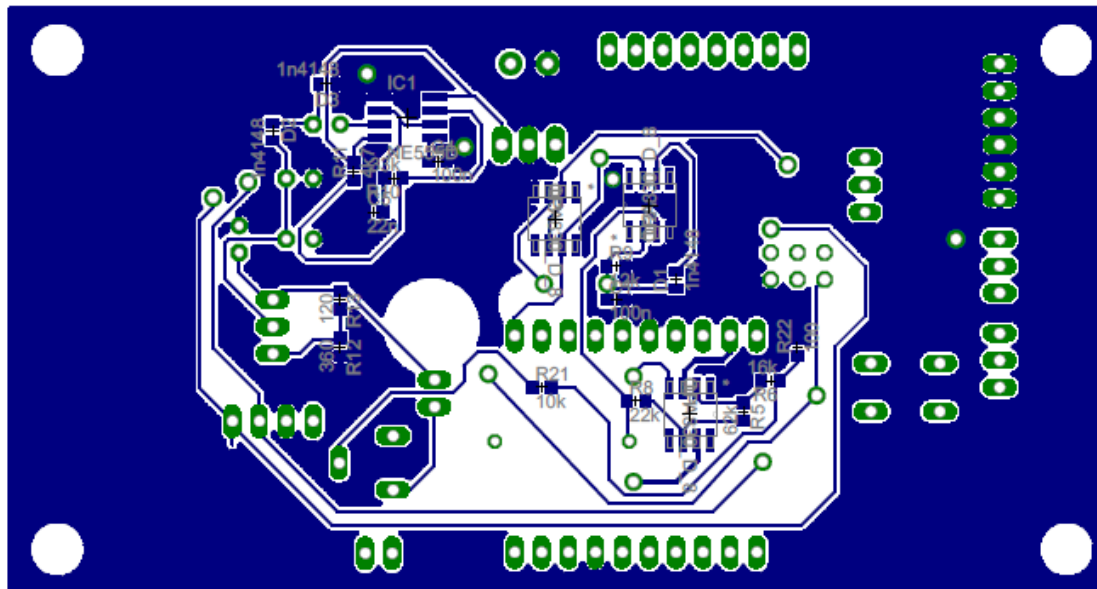


Function generator and  
peak detector with +5  
and -5 voltage regulators

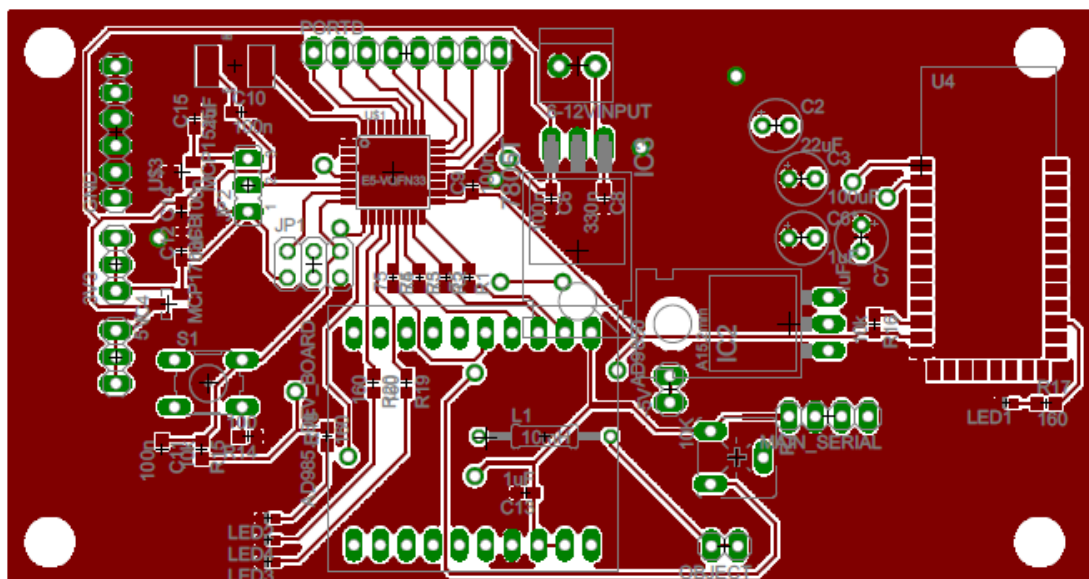
full\_circuit  
2014.04.29, 23:41:29  
Sheet: 1/2



## PCB design



**Figure C-1 Bottom part of PCB**



### Figure C-2 Top of the PCB



## D. Appendix

### Bill of material

Part	Value	Device	Package
-5V		PINHD-1X2	1X02
3V3		PINHD-1X3	1X03
5V		PINHD-1X3	1X03
6-12VINPUT	1X2-3.5MM	1X2-3.5MM	1X2-3.5MM
AD9850	AD985._DEV_BOARD	AD985._DEV_BOARD	PINS2X10
C1	100n	C-EUC0603	C0603
C2	22uF	CPOL-EUE2.5-5	E2,5-5
C3	100uF	CPOL-EUE2.5-5	E2,5-5
C4	100n	C-EUC0603	C0603
C5	22n	C-EUC0603	C0603
C6	100n	C-EUC0603	C0603
C6*	1uF	CPOL-EUE2.5-5	E2,5-5
C7	1uF	CPOL-EUE2.5-5	E2,5-5
C8	330n	C-EUC0603	C0603
C9	100n	C-EUC0603	C0603
C10	100n	C-EUC0603	C0603
C11	100n	C-EUC0603	C0603
C12	1uF	C-EUC0603	C0603
C13	1uF	C-EUC0603	C0603
C14	100n	C-EUC0603	C0603
C15	1uF	C-EUC0603	C0603
D1	1n4148	DIODE-MICROMELF-R	MICROMELF-R
D2	1n4148	DIODE-MICROMELF-R	MICROMELF-R
D3	1n4148	DIODE-MICROMELF-R	MICROMELF-R
GND		PINHD-1X6	1X06
IC1	NE555D	NE555D	SO08
IC2		LM337TL	337TL
IC3	7805T	7805T	TO220H
IC4	MCP1703CB	MCP1703CB	SOT23
JP1		PINHD-2X3	2X03
JP2		JP2E	JP2
L1	10mH	L-US0207/12	0207/12
LED1		LEDCHIP-LED0603	CHIP-LED0603
LED2		LEDCHIP-LED0603	CHIP-LED0603
LED3		LEDCHIP-LED0603	CHIP-LED0603
LED4		LEDCHIP-LED0603	CHIP-LED0603
MAIN_SERIAL		PINHD-1X4	1X04
OBJECT		PINHD-1X2	1X02
PORTD		PINHD-1X8	1X08
R1	75	R-EU_R0603	R0603

<b>R2</b>	75	R-EU_R0603	R0603
<b>R3</b>	75	R-EU_R0603	R0603
<b>R4</b>	75	R-EU_R0603	R0603
<b>R5</b>	62k	R-EU_R0603	R0603
<b>R6</b>	16k	R-EU_R0603	R0603
<b>R7</b>	10K	TRIM_EU-CA6V	CA6V
<b>R8</b>	22k	R-EU_R0603	R0603
<b>R9</b>	12k	R-EU_R0603	R0603
<b>R10</b>	33k	R-EU_R0603	R0603
<b>R11</b>	4k7	R-EU_R0603	R0603
<b>R12</b>	360	R-EU_R0603	R0603
<b>R13</b>	120	R-EU_R0603	R0603
<b>R14</b>	100	R-EU_R0603	R0603
<b>R15</b>	10k	R-EU_R0603	R0603
<b>R16</b>	10k	R-EU_R0603	R0603
<b>R17</b>	160	R-EU_R0603	R0603
<b>R18</b>	160	R-EU_R0603	R0603
<b>R19</b>	160	R-EU_R0603	R0603
<b>R20</b>	160	R-EU_R0603	R0603
<b>R21</b>	10k	R-EU_R0603	R0603
<b>R22</b>	100	R-EU_R0603	R0603
<b>S1</b>		10-XX	B3F-10XX
<b>U\$1</b>	E5-VQFN33	E5-VQFN33	TQFP32-08
<b>U\$2</b>	10uH	INDUCTOR1206H*	INDUCTOR-1206
<b>U\$3</b>	MCP1525	MCP1525	SOT23
<b>U1</b>	OPA830_D_8	OPA830_D_8	D8
<b>U2</b>	OPA830_D_8	OPA830_D_8	D8
<b>U3</b>	OPA830_D_8	OPA830_D_8	D8
<b>U4</b>	HC_05	HC_05	BLUETOOTH_SMD

### ***Main.cpp file***

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

### ***Learner.h file***

```
#ifndef LEARNER_H
#define LEARNER_H

#include <QDialog>
#include <QFileDialog>
#include <QMessageBox>
#include <QProcess>
#include <QSettings>
#include <QDebug>

namespace Ui {
class Learner;
}

class Learner : public QDialog
{
    Q_OBJECT

public:
    explicit Learner(QWidget *parent = 0);
    ~Learner();

signals:
    void gatherData(int amount, int id, QString file, QString name);

private slots:
    void startGathering();
    void openFileDialog();
    void learn();
    void enableButtons(int exitCode);
    void readProcess();

private:
    Ui::Learner *ui;

    QProcess *svmProcess;
};

#endif // LEARNER_H
```

## *Learner.cpp file*

```
#include "learner.h"
#include "ui_learner.h"

Learner::Learner(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Learner)
{
    ui->setupUi(this);

    connect(ui->collectBtn, SIGNAL(clicked()), this, SLOT(startGathering()));
    connect(ui->cancelBtn, SIGNAL(clicked()), this, SLOT(close()));
    connect(ui->browseBtn, SIGNAL(clicked()), this, SLOT(openFileDialog()));
    connect(ui->learnBtn, SIGNAL(clicked()), this, SLOT(learn()));

    svmProcess = new QProcess(this);

    ui->stopBtn->setVisible(false);

    connect(ui->stopBtn, SIGNAL(clicked()), svmProcess, SLOT(kill()));
}

void Learner::openFileDialog()
{
    QString fileName = QFileDialog::getSaveFileName(this,
        tr("Save SVM data"), "",
        tr("SVM data (*.svm);;All Files (*)"));

    if (fileName.isEmpty())
        return;
    else
    {
        //QFile file(fileName);
        ui->fileTxt->setText( fileName );
    }
}

void Learner::startGathering()
{
    if(ui->fileTxt->text().isEmpty())
    {
        QMessageBox::critical(this, tr("File not found"), tr("Please specify a
file"));
        return;
    }
    if(ui->gestureName->text().isEmpty())
    {
        QMessageBox::critical(this, tr("Empty name"), tr("Please specify the name of
gesture"));
        return;
    }

    //Send signal to MainWindow class, that I need Data
    emit gatherData(ui->spinTrain->value(), ui->spinId->value(), ui->fileTxt->text(),
ui->gestureName->text());
    this->close();
}

void Learner::learn()
{
    //initiate process to construct svm model file
    QSettings settings("HKEY_CURRENT_USER\\Software\\Rastro\\Kulaks",
        QSettings::NativeFormat);

    connect(svmProcess, SIGNAL(finished(int)), this, SLOT(enableButtons(int)));
    connect(svmProcess, SIGNAL(readyReadStandardOutput()), this, SLOT(readProcess()));

    QStringList arguments;
    QString scriptPath = QApplication::applicationDirPath() + "/libsvm/tools/";
    svmProcess->setWorkingDirectory(scriptPath);

    arguments << "easy.py" << ui->fileTxt->text() << ui->fileTxt->text();

    qDebug() << "Arguments: " << arguments;
    //    qDebug() << "scriptPath";
}
```

```

svmProcess->start( settings.value("python").toString() , arguments);

ui->learnBtn->setEnabled(false);
ui->cancelBtn->setEnabled(false);
ui->collectBtn->setEnabled(false);

ui->stopBtn->setVisible(true);
}

void Learner::readProcess()
{
    QByteArray bytes = svmProcess->readAllStandardOutput();
    qDebug() << bytes;
}

void Learner::enableButtons(int exitCode)
{
    ui->learnBtn->setEnabled(true);
    ui->cancelBtn->setEnabled(true);
    ui->collectBtn->setEnabled(true);

    ui->stopBtn->setVisible(false);

    qDebug() << "Process closed with: " << exitCode << svmProcess-
>readAllStandardError();

    if(exitCode > 0)
        QMessageBox::critical(this, "Something went wrong with QProcess", svmProcess-
>readAllStandardError());
}

Learner::~Learner()
{
    delete ui;
}

```

## **MainWindow.h file**

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QMessageBox>
#include "qcustomplot.h"
#include "learner.h"
#include <QFile>
#include <QMessageBox>
#include "svm.h"
#include "settings.h"
#include <QSettings>
#include "serialterminal.h"
#include "playground/simulatekeyboard.h"
#include <QTimer>
#include <QThread>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    void writeToLogger(QString text);
    ~MainWindow();

private slots:
    void readData();
    void sendSerialData(QByteArray data);
    void openSerialPort();
    void closeSerialPort();
    void plot(QVector<double> values);
    void startGathering(int amount, int id, QString file, QString name);
    void openLearnerDialog();
    void doSomeMagic(QVector<double> samp);
    void openSettings();
    void openTerminal();
    void enableMagic(bool enable);
    void openSimulateKeyboard();
    void dataASecond();
    void sendDeviceSettings(int min, int max, int n);

private:
    Ui::MainWindow *ui;
    QSerialPort *serial;
    Learner *learner;
    SerialTerminal *terminal;
    Settings *settings;

    SimulateKeyboard *keyboardPlayground;

    QByteArray allData;
    QVector< QVector<double> > gatheredData;
    int numberOfDataStillNeeded;
    int numberOfDataNeeded;
    QString gestureName;
    QStringList gestureNameList;

    QTimer* dataASecondT;
    int collected;

    QString file;
    int id;

    int numberOfSamples;

    struct svm_model * model;
```

```
double rmin[160];  
double rmax[160];  
  
};  
  
#endif // MAINWINDOW_H
```



## MainWindow.cpp file

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <errno.h>

//Legacy constants, not used anymore
#define NOSAMPS 160
#define VOLTAGE 0.0048828125
#define FREQUENCY_STEP 1.781

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setUpUi(this);

    settings = new Settings(this);
    connect(settings, SIGNAL(deviceSettings(int,int,int)), this,
        SLOT(sendDeviceSettings(int,int,int)));

    serial = new QSerialPort(this); //Construct serial object

    //Connect correspnindg serial object signals to their slots
    connect(serial, SIGNAL(readyRead()), this, SLOT(readData()));
    connect(ui->actionConnect, SIGNAL(triggered()), this, SLOT(openSerialPort()));
    connect(ui->actionDisconnect, SIGNAL(triggered()), this, SLOT(closeSerialPort()));
    connect(ui->actionSettings, SIGNAL(triggered()), this, SLOT(openSettings()));

    learner = new Learner(this);
    connect(learner, SIGNAL(gatherData(int,int,QString,QString)), this,
        SLOT(startGathering(int,int,QString,QString)));
    connect(ui->actionGather_data, SIGNAL(triggered()), this,
        SLOT(openLearnerDialog()));
    connect(ui->actionDo_Magic, SIGNAL(triggered(bool)), this,
        SLOT(enableMagic(bool)));

    terminal = new SerialTerminal(this);
    connect(ui->actionTerminal, SIGNAL(triggered()), this, SLOT(openTerminal()));
    connect(terminal, SIGNAL(sendToSerial(QByteArray)), this,
        SLOT(sendSerialData(QByteArray)));

    connect(ui->actionExit, SIGNAL(triggered()), this, SLOT(close()));

    numberOfDataStillNeeded = 0;
    ui->progressBar->setVisible(false);

    connect(ui->actionSimulate_Keyboard, SIGNAL(triggered()), this,
        SLOT(openSimulateKeyboard()));
    keyboardPlayground = new SimulateKeyboard(this);

    numberOfSamples = NOSAMPS;

    dataASecondT = new QTimer(this);
    connect(dataASecondT, SIGNAL(timeout()), this, SLOT(dataASecond()));
    collected = 0;
}

void MainWindow::sendDeviceSettings(int min, int max, int n)
{
    qDebug() << "Sending 2: " << QString("%1 %2
%3").arg(min).arg(max).arg(n).toLatin1();
    this->sendSerialData( "s" );
    this->sendSerialData( "c" );
    QByteArray configData = QByteArray( QString("%1 %2
%3\n").arg(min).arg(max).arg(n).toLatin1() );
    sendSerialData( configData );
    //this->sendSerialData( "r" );
    this->sendSerialData( "d" );
}

void MainWindow::openSimulateKeyboard()
{
    if(gestureNameList.size() == 0)
    {
        QMessageBox::critical(this, "Error", "Before using keyboard playground you
must load model file.");
    }
}
```

```

        return;
    }
    keyboardPlayground->show();
    keyboardPlayground->setGestures( gestureNameList );
}

void MainWindow::enableMagic(bool enable)
{
    //Is pattern recognition enabled?
    if(enable)
    {
        gestureNameList.clear();
        QString fileName = QFileDialog::getOpenFileName(this,
            tr("Open SVM model"), "",
            tr("SVM model data (*.model);;All Files (*)"));

        if (fileName.isEmpty())
        {
            ui->actionDo_Magic->blockSignals(true);
            ui->actionDo_Magic->setChecked(false);
            ui->actionDo_Magic->blockSignals(false);
            return;
        }
        else
        {
            //QFile file(fileName);
            QDir dir;
            //qDebug() << dir.absoluteFilePath("gesture.svm.model");
            char buffer[1024];
            strcpy(buffer, fileName.toStdString().c_str());

            // Load SVM model in its data structure
            if((model=svm_load_model(buffer))==0)
            {
                QMessageBox::critical(this, tr("Error"), tr("Can't open model
file."));
                exit(1);
            }
            fileName.chop(5);
            QFile file( fileName.append("range") );
            if(!file.open(QIODevice::ReadOnly))
            {
                QMessageBox::critical(0, tr("Error"), file.errorString());
                return;
            }

            //Read the range, so we can perform scaling later
            QTextStream in(&file);
            in.readLine(); in.readLine();
            while(!in.atEnd())
            {
                QString line = in.readLine();
                QStringList fields = line.split(" ");

                int i = fields.at(0).toInt() - 1;
                double min = fields.at(1).toDouble();
                double max = fields.at(2).toDouble();

                rmin[i] = min;
                rmax[i] = max;

                //qDebug() << i << rmin[i] << rmax[i];
            }

            file.close();

            fileName.chop(5);
            //Read the names of gestures
            file.setFileName( fileName.append("name") );
            if(!file.open(QIODevice::ReadOnly))
            {
                QMessageBox::critical(0, tr("Error"), file.errorString());
                return;
            }

            in.setDevice(&file);
            while(!in.atEnd())

```

```

        {
            gestureNameList.push_back( in.readLine() );
        }
        file.close();
    }

    ui->statusBar->showMessage("SVM model loaded, ready to do some magic");
}

void MainWindow::openTerminal()
{
    terminal->show();
}

void MainWindow::openSettings()
{
    settings->exec();
}

void MainWindow::openLearnerDialog()
{
    learner->show();
}

void MainWindow::startGathering(int amount, int id, QString file, QString name)
{
    //initiates data gathering for learning
    numberOfDataStillNeeded = amount;
    numberOfDataNeeded = amount;
    ui->progressBar->setRange(0, amount);
    ui->progressBar->setVisible( true );
    this->id = id;
    this->file = file;
    this->gestureName = name;
    gatheredData.clear();
}

void MainWindow::doSomeMagic(QVector<double> samp)
{
    struct svm_node features[numberOfSamples+1];
    double *decVals;
    int x, noLabels, noDecVals;
    double r;

    noLabels = svm_get_nr_class( model );
    noDecVals=noLabels*(noLabels-1)/2;
    decVals=(double*)malloc(sizeof(double)*noDecVals);

    //Perform data scaling from -1 to 1
    for(x=0; x<numberOfSamples; x++)
    {
        features[x].index = x;

        r=(samp[x]-rmin[x+1]);
        r=r / (rmax[x+1]-rmin[x+1]);
        r=(r*2)-1;

        features[x+1].value = r;
    }

    features[x].index = -1;
    //Get the ID of closest data cluster
    r = svm_predict_values(model, features, decVals);
    writeToLogger( QString("Prediction value: %1 %2\n").arg(r) );

    /*switch((int)r)
    {
        case 1: ui->gestureLbl->setText("One finger"); break;
        case 2: ui->gestureLbl->setText("Four fingers"); break;
        case 4: ui->gestureLbl->setText("Two fingers"); break;
        case 3: ui->gestureLbl->setText("Hand near by"); break;
        default: ui->gestureLbl->setText("Nothing");
    }*/
}

```

```

        //Print the name of gesture
        if((int)r < gestureNameList.size())
            ui->gestureLbl->setText( gestureNameList.at( (int)r ) );

        //If key board simulator is open, emulate key press
        if(keyboardPlayground->isVisible())
            keyboardPlayground->gesture( (int)r );
    }

void MainWindow::plot(QVector<double> values)
{
    ui->plot->clearGraphs();
    ui->plot->addGraph();
    ui->plot->graph(0)->setPen(QPen(Qt::blue)); // line color blue for first graph
    ui->plot->graph(0)->setBrush(QBrush(QColor(0, 0, 255, 20))); // first graph will
    be filled with translucent blue

    QVector<double> x;
    for (int i=0; i<values.size(); ++i)
    {
        x.push_back( i*FREQUENCY_STEP );
    }
    // configure right and top axis to show ticks but no labels:
    // (see QCPAxisRect::setupFullAxesBox for a quicker method to do this)
    ui->plot->xAxis2->setVisible(true);
    ui->plot->xAxis2->setTickLabels(false);
    ui->plot->yAxis2->setVisible(true);
    ui->plot->yAxis2->setTickLabels(false);
    // make left and bottom axes always transfer their ranges to right and top axes:
    connect(ui->plot->xAxis, SIGNAL(rangeChanged(QCPRange)), ui->plot->xAxis2,
    SLOT(setRange(QCPRange)));
    //connect(ui->plot->yAxis, SIGNAL(rangeChanged(QCPRange)), ui->plot->yAxis2,
    SLOT(setRange(QCPRange)));
    // pass data points to graphs:
    ui->plot->graph(0)->setData(x, values);
    // let the ranges scale themselves so graph 0 fits perfectly in the visible area:
    ui->plot->graph(0)->rescaleAxes();

    ui->plot->yAxis->setRange(0, 1000*VOLTAGE);
    ui->plot->xAxis->setRange(0, values.size()*FREQUENCY_STEP);

    // Note: we could have also just called ui->plot->rescaleAxes(); instead
    // Allow user to drag axis ranges with mouse, zoom with mouse wheel and select
    graphs by clicking:
    ui->plot->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom |
    QCP::iSelectPlottables);

    ui->plot->replot();
}

void MainWindow::openSerialPort()
{
    //Read settings from registry
    QSettings settings("HKEY_CURRENT_USER\\Software\\Rastro\\Kulaks",
        QSettings::NativeFormat);
    serial->setPortName(settings.value("Serial/port").toString());
    //Try to open serial device
    if (serial->open(QIODevice::ReadWrite))
    {
        if(
            serial->setBaudRate(settings.value("Serial/baudrate").toInt()) &&
            serial->setParity( QSerialPort::NoParity ) &&
            serial->setStopBits( QSerialPort::OneStop ) &&
            serial->setFlowControl( QSerialPort::NoFlowControl )
        )
        {
            ui->statusBar->showMessage("Connected");
            ui->actionConnect->setEnabled(false);
            ui->actionDisconnect->setEnabled(true);
        }
        else
        {
            serial->close();
            QMessageBox::critical(this, tr("Error"), serial->errorString());

            ui->statusBar->showMessage(tr("Open error"));
        }
    }
}

```

```

    }
    else
    {
        QMessageBox::critical(this, tr("Error"), serial->errorString());

        ui->statusBar->showMessage(tr("Configure error"));
    }

    //sendSerialData(QByteArray( QString("Hello from Qt").toLatin1() ));
    sendSerialData( "q" ); //RESET the XMEGA
    this->thread()->msleep(150); //Wait for xmega to reinitiliaze
    //Send the configuration to the hardware
    this->sendDeviceSettings( settings.value("Dev/min").toInt(),
settings.value("Dev/max").toInt(), settings.value("Dev/n").toInt() );
}

void MainWindow::closeSerialPort()
{
    serial->close();
    ui->statusBar->showMessage(tr("Disconnected"));

    ui->actionConnect->setEnabled(true);
    ui->actionDisconnect->setEnabled(false);
}

void MainWindow::sendSerialData(QByteArray data)
{
    serial->write(data);
    serial->waitForBytesWritten(2000);
}

//reads data from serial
void MainWindow::readData()
{
    //QByteArray data = serial->readLine();

    QByteArray data = serial->readAll(); //Read all buffer

    if(terminal->isVisible())
        terminal->writeToTerminal(QString(data)); //If terminal is visible print
results

    allData.append(data); //Append to our own buffer
    //allData = data;

    //QDebug() << allData.size();
    if(allData.contains("{") ) //If buffer contains beining of data
    { //Make sure that unused data beforehand are deleted
        allData.remove(0, allData.indexOf("{"));

        if(!dataASecondT->isActive())
            dataASecondT->start(1000);
    }

    // If data contains both curly brackets, WE GOT DATA
    if(allData.indexOf("{") == 0 && allData.contains("}"))
    {
        QByteArray tempData = allData;

        //Extract the received data
        tempData = tempData.mid(allData.indexOf("{")+1, allData.indexOf("}")-1);
        allData.remove( 0, allData.indexOf( "}" )+1 );

        QString tempStr(tempData);

        QStringList tempList = tempStr.split(" ");

        QVector<double> numbers;

        //Convert extracted data to voltage
        foreach(QString str, tempList)
        {
            if(!str.isEmpty())
            {

```

```

        //numbers.push_back( str.toDouble()*VOLTAGE );
        numbers.push_back( (str.toDouble() * 2.7/ 4096.0f) - 0.135f );
    }

    numberOfSamples = numbers.size();

    writeToLogger( QString("Extracted numbers: %1\tGathered data:
%2\n").arg(numbers.size()).arg(gatheredData.size()) );

    plot(numbers); // <- plot the new results

    if(ui->actionDo_Magic->isChecked()) //If SVM classifier is enabled
    {
        doSomeMagic( numbers ); //Do pattern recognition
    }

    if(numberOfDataStillNeeded > 1) //Does SVM learner need more data?
    {
        gatheredData.push_back( numbers );
        if(gatheredData.size()%10 == 0)
            ui->progressBar->setValue(numberOfDataNeeded-numberOfDataStillNeeded);
    }
    else if(numberOfDataStillNeeded == 1) //If enough has gathered, save to SVM
file
    {
        ui->progressBar->setVisible(false);
        gatheredData.push_back( numbers );

        //time to save svm file
        QFile file(this->file);
        if (!file.open(QIODevice::Append))
        {
            QMessageBox::information(this, tr("Unable to open file"),
                file.errorString());
            return;
        }
        else
        {
            QTextStream out(&file);

            if(file.size() > 0)
                out << endl;

            for(int d=0; d<gatheredData.size(); d++)//QVector< double > data,
gatheredData)
            {
                QVector<double> data = gatheredData.at(d);
                out << this->id << " ";

                for(int i=0; i<data.size(); i++)
                    if(data.size()-1 != i || d == gatheredData.size()-1)
                        out << QString::number(i+1) << ":" << QString::number(
data[i] ) << " ";
                    else
                        out << QString::number(i+1) << ":" << QString::number(
data[i] ) << endl;
            }

            writeToLogger("Data saved to " + this->file);
            file.close();
        }

        //time to save name file
        this->file.append(".name");
        file.setFileName(this->file);
        if (!file.open(QIODevice::Append))
        {
            QMessageBox::information(this, tr("Unable to open file"),
                file.errorString());
            return;
        }
        else
        {
            QTextStream out(&file);
            out << this->gestureName << endl;

```

```

        }

        file.close();
    }

    if(numberOfDataStillNeeded != 0)
        numberOfDataStillNeeded--;

    collected++;
}

}

void MainWindow::dataASecond()
{
    //qDebug() << collected;
    ui->lblDataASecond->setText(QString::number(collected) + " data/second");

    //Code for measuring the performance of the device.
    //Every second stores how many data points were collected
    //    QFile file ("avg_conv_data.txt");
    //    if (!file.open(QIODevice::Append))
    //    {
    //        QMessageBox::information(this, tr("Unable to open file"),
    //            file.errorString());
    //        return;
    //    }
    //    else
    //    {
    //        QTextStream out(&file);
    //        out << QString::number(collected) << " ";
    //    }

    //    file.close();

    collected = 0;
}

void MainWindow::writeToLogger(QString text)
{
    //Writes text to bottom debugger text area

    if(ui->console->toPlainText().length() > 10000)
        ui->console->clear();
    ui->console->insertPlainText( text );
    QTextCursor c = ui->console->textCursor();
    c.movePosition(QTextCursor::End);
    ui->console->setTextCursor(c);
}

MainWindow::~MainWindow()
{
    delete ui;
}

```

### ***SerialTerminal.h file***

```
#ifndef SERIALTERMINAL_H
#define SERIALTERMINAL_H

#include <QMainWindow>
#include <QToolBar>
#include <QKeyEvent>
#include <QDebug>

namespace Ui {
class SerialTerminal;
}

class SerialTerminal : public QMainWindow
{
    Q_OBJECT

public:
    explicit SerialTerminal(QWidget *parent = 0);
    ~SerialTerminal();

    void writeToTerminal(QString text);

protected:
    void keyReleaseEvent(QKeyEvent *e);

private slots:
    void commandSend();

signals:
    void sendToSerial(QByteArray data);

private:
    Ui::SerialTerminal *ui;
};

#endif // SERIALTERMINAL_H
```



## ***SerialTerminal.cpp file***

```
#include "serialterminal.h"
#include "ui_serialterminal.h"

SerialTerminal::SerialTerminal(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::SerialTerminal)
{
    ui->setupUi(this);

    QToolBar *toolBar = new QToolBar("Terminal Tools", this);

    QAction *clearAction = new QAction(QIcon(":/images/edit-clear.png"),
"Clear",toolBar);
    toolBar->addAction(clearAction);
    connect(clearAction, SIGNAL(triggered()), ui->terminal, SLOT(clear()));
    connect(ui->sendBtn, SIGNAL(clicked()), this, SLOT(commandSend()));

    this->addToolBar(toolBar);
}

void SerialTerminal::keyReleaseEvent(QKeyEvent *e)
{
    if(e->key() == Qt::Key_Return)
        emit sendToSerial( QByteArray( QString("\n").toLatin1() ) );
    else
        emit sendToSerial( QByteArray( QString(e->text()).toLatin1() ) );
    //QMainWindow::keyReleaseEvent(e);
}

void SerialTerminal::commandSend()
{
    emit sendToSerial( QByteArray( QString(ui->cmdEdit->
text().append("\n")).toLatin1() ) );
}

void SerialTerminal::writeToTerminal(QString text)
{
    text.replace("\n\r", "\n");
    text.replace("\r\n", "\n");

    ui->terminal->insertPlainText( text );
    QTextCursor c = ui->terminal->textCursor();
    c.movePosition(QTextCursor::End);
    ui->terminal->setTextCursor(c);
}

SerialTerminal::~SerialTerminal()
{
    delete ui;
}
```

### ***Settings.h file***

```
#ifndef SETTINGS_H
#define SETTINGS_H

#include <QDialog>
#include <QDebug>
#include <QSerialPortInfo>
#include <QSettings>
#include <QFileDialog>

namespace Ui {
class Settings;
}

class Settings : public QDialog
{
    Q_OBJECT

public:
    explicit Settings(QWidget *parent = 0);
    ~Settings();

signals:
    void deviceSettings(int min, int max, int n);

private slots:
    void loadSettings();
    void saveSettings();

    void pythonPathBtn();
    void gnuplotPathBtn();

private:
    Ui::Settings *ui;
};

#endif // SETTINGS_H
```

## Settings.cpp file

```
#include "settings.h"
#include "ui_settings.h"

Settings::Settings(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Settings)
{
    ui->setupUi(this);

    QList<QSerialPortInfo> ports = QSerialPortInfo::availablePorts();

    foreach(QSerialPortInfo port, ports)
        ui->serialPort->addItem(port.portName());

    foreach(qint32 baudrate, QSerialPortInfo::standardBaudRates())
        ui->baudrate->addItem(QString::number( baudrate ));

    connect(ui->buttonBox, SIGNAL(accepted()), this, SLOT(saveSettings()));
    connect(ui->pythonBtn, SIGNAL(clicked()), this, SLOT(pythonPathBtn()));
    connect(ui->gnuPlotBtn, SIGNAL(clicked()), this, SLOT(gnuplotPathBtn()));

    loadSettings();
}

void Settings::gnuplotPathBtn()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("Select Gnuplot"),
                                                    "",
                                                    tr("Files (*.*)"));

    ui->gnuPlotPath->setText(fileName);
    saveSettings();
}

void Settings::pythonPathBtn()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("Select Python"),
                                                    "",
                                                    tr("Files (*.*)"));

    ui->pythonPath->setText(fileName);
    saveSettings();
}

//Saves settings in the registry
void Settings::saveSettings()
{
    QSettings settings("HKEY_CURRENT_USER\\Software\\Rastro\\Kulaks",
                      QSettings::NativeFormat);
    settings.setValue("Serial/port", ui->serialPort->currentText());
    settings.setValue("Serial/baudrate", ui->baudrate->currentText());

    //Send new settings to the hardware
    emit deviceSettings(ui->minFreqSpinBox->value(), ui->maxFreqSpinBox->value(), ui->frequencyCountSpinBox->value());

    settings.setValue("Dev/min", ui->minFreqSpinBox->value());
    settings.setValue("Dev/max", ui->maxFreqSpinBox->value());
    settings.setValue("Dev/n", ui->frequencyCountSpinBox->value());

    settings.setValue("python", ui->pythonPath->text());
    settings.setValue("gnuplot", ui->gnuPlotPath->text());
}

//loads settings from the registry
void Settings::loadSettings()
{
    QSettings settings("HKEY_CURRENT_USER\\Software\\Rastro\\Kulaks",
                      QSettings::NativeFormat);

    ui->serialPort->setCurrentText( settings.value("Serial/port").toString() );
    ui->baudrate->setCurrentText( settings.value("Serial/baudrate").toString() );
}
```

```
ui->minFreqSpinBox->setValue( settings.value("Dev/min").toInt() );
ui->maxFreqSpinBox->setValue( settings.value("Dev/max").toInt() );
ui->frequencyCountSpinBox->setValue( settings.value("Dev/n").toInt() );

ui->pythonPath->setText( settings.value("python").toString() );
ui->gnuPlotPath->setText( settings.value("gnuplot").toString() );
}

Settings::~Settings()
{
    delete ui;
}
```

### ***CustomTreeWidget.h file***

```
#ifndef CUSTOMTREEWIDGET_H
#define CUSTOMTREEWIDGET_H

#include <QTreeWidget>

class CustomTreeWidget : public QTreeWidget
{
    Q_OBJECT
public:
    explicit CustomTreeWidget(QWidget *parent = 0);

protected:
    void keyReleaseEvent(QKeyEvent *e);
    void keyPressEvent(QKeyEvent *event);

signals:

public slots:

};

#endif // CUSTOMTREEWIDGET_H
```

### ***CustomTreeWidget.cpp file***

```
#include "customtreewidget.h"

CustomTreeWidget::CustomTreeWidget(QWidget *parent) :
    QTreeWidget(parent)
{
}

void CustomTreeWidget::keyPressEvent(QKeyEvent *event)
{
    return;
}

void CustomTreeWidget::keyReleaseEvent(QKeyEvent *e)
{
    return;
}
```

### ***KeyBinder.h file***

```
#ifndef KEYBINDER_H
#define KEYBINDER_H

#include <QLineEdit>
#include <QDebug>
#include <QKeyEvent>

class KeyBinder : public QLineEdit
{
    Q_OBJECT
public:
    explicit KeyBinder(QWidget *parent = 0);

protected:
    void keyReleaseEvent(QKeyEvent *e);

signals:

public slots:

};

#endif // KEYBINDER_H
```

### ***KeyBinder.cpp file***

```
#include "keybinder.h"

KeyBinder::KeyBinder(QWidget *parent) :
    QLineEdit(parent)
{
    this->setReadOnly(true);
    this->setFocusPolicy(Qt::StrongFocus);
}

void KeyBinder::keyReleaseEvent(QKeyEvent *e)
{
    qDebug() << "Key released:" << e->key() << Qt::Key_Up;

    /*if(e->key() == Qt::Key_Up)
        this->setText("Up");
    else if(e->key() == Qt::Key_Down)
        this->setText("Down");
    else if(e->key() == Qt::Key_Left)
        this->setText("Left");
    else if(e->key() == Qt::Key_Right)
        this->setText("Right");
    else if(e->key() == Qt::Key_Space)
        this->setText("Space");
    else
        this->setText(e->text());*/

    this->setText( QString::number(e->key()) );
}
```

### ***KeyDelegate.h file***

```
#ifndef KEYDELEGATE_H
#define KEYDELEGATE_H

#include <QStyledItemDelegate>
#include "keybinder.h"

class KeyDelegate : public QStyledItemDelegate
{
    Q_OBJECT
public:
    explicit KeyDelegate(QObject *parent = 0);

    QWidget *createEditor(QWidget *parent, const QStyleOptionViewItem &option,
                          const QModelIndex &index) const;

    void setEditorData(QWidget *editor, const QModelIndex &index) const;
    void setModelData(QWidget *editor, QAbstractItemModel *model,
                      const QModelIndex &index) const;

    void updateEditorGeometry(QWidget *editor,
                              const QStyleOptionViewItem &option, const QModelIndex &index) const;

signals:

public slots:

};

#endif // KEYDELEGATE_H
```

### ***KeyDelegate.cpp file***

```
#include "keydelegate.h"

KeyDelegate::KeyDelegate(QObject *parent) :
    QStyledItemDelegate(parent)
{
}

QWidget *KeyDelegate::createEditor(QWidget *parent,
    const QStyleOptionViewItem & /* option */,
    const QModelIndex & /* index */) const
{
    KeyBinder *editor = new KeyBinder(parent);

    return editor;
}

void KeyDelegate::setEditorData(QWidget *editor,
    const QModelIndex &index) const
{
    //If tree widget double clicked, it will initiate this method
    QString value = index.model()->data(index, Qt::EditRole).toString();

    KeyBinder *spinBox = static_cast<KeyBinder*>(editor);
    spinBox->setText(value);
}

void KeyDelegate::setModelData(QWidget *editor, QAbstractItemModel *model,
    const QModelIndex &index) const
{
    //First lets set known data from tree widget
    KeyBinder *spinBox = static_cast<KeyBinder*>(editor);
    //QLineEdit->interpretText();
    QString value = spinBox->text();

    model->setData(index, value, Qt::EditRole);
    spinBox->releaseKeyboard();
}

void KeyDelegate::updateEditorGeometry(QWidget *editor,
    const QStyleOptionViewItem &option, const QModelIndex & /* index */) const
{
    //Perform widget scaling to fill the tree widget's field
    editor->setGeometry(option.rect);
}
```



### ***SimulateKeyboard.h file***

```
#ifndef SIMULATEKEYBOARD_H
#define SIMULATEKEYBOARD_H

#include <QDialog>
#include <QDebug>
#include "keydelegate.h"
#include "customtreewidget.h"
#include <windows.h>

namespace Ui {
class SimulateKeyboard;
}

class SimulateKeyboard : public QDialog
{
    Q_OBJECT

public:
    explicit SimulateKeyboard(QWidget *parent = 0);
    ~SimulateKeyboard();
    void setGestures(QStringList gestures);
    void gesture(int g);

private slots:
    void sendKeys(bool enabled);

private:
    Ui::SimulateKeyboard *ui;
    QStringList availableGestures;
};

#endif // SIMULATEKEYBOARD_H
```

## *SimulateKeyboard.cpp file*

```
#include "simulatekeyboard.h"
#include "ui_simulatekeyboard.h"

SimulateKeyboard::SimulateKeyboard(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SimulateKeyboard)
{
    ui->setupUi(this);
    ui->keyMap->setItemDelegate(new KeyDelegate(this));
    ui->keyMap->setFocusPolicy(Qt::NoFocus);

    connect(ui->simulateBtn, SIGNAL(clicked(bool)), this, SLOT(sendKeys(bool)));
}

void SimulateKeyboard::gesture(int g)
{
    // if(ui->keyMap->topLevelItem(g)->text(1).isEmpty())
    //     return;

    INPUT ip;
    // Set up a generic keyboard event.
    ip.type = INPUT_KEYBOARD;
    ip.ki.wScan = 0; // hardware scan code for key
    ip.ki.time = 0;
    ip.ki.dwExtraInfo = 0;

    // Press the "A" key
    ip.ki.wVk = ui->keyMap->topLevelItem(g)->text(1).toInt(); // virtual-key code for
the "a" key
    ip.ki.dwFlags = 0; // 0 for key press
    SendInput(1, &ip, sizeof(INPUT));

    // Release the "A" key
    ip.ki.dwFlags = KEYEVENTF_KEYUP; // KEYEVENTF_KEYUP for key release
    SendInput(1, &ip, sizeof(INPUT));
}

void SimulateKeyboard::sendKeys(bool enabled)
{
    if(enabled)
    {
        qDebug() << "Starting keyboard simulation";
        INPUT ip;

        Sleep(1000);
        // Set up a generic keyboard event.
        ip.type = INPUT_KEYBOARD;
        ip.ki.wScan = 0; // hardware scan code for key
        ip.ki.time = 0;
        ip.ki.dwExtraInfo = 0;

        // Press the "A" key
        ip.ki.wVk = ui->keyMap->topLevelItem(0)->text(1).toInt(); // virtual-key code
for the "a" key
        ip.ki.dwFlags = 0; // 0 for key press
        SendInput(1, &ip, sizeof(INPUT));
        Sleep(200);
        // Release the "A" key
        ip.ki.dwFlags = KEYEVENTF_KEYUP; // KEYEVENTF_KEYUP for key release
        SendInput(1, &ip, sizeof(INPUT));

        qDebug() << "key press simulated";
    }
}

void SimulateKeyboard::setGestures(QStringList gestures)
{
    this->availableGestures = gestures;

    // ui->keyMap->clear();

    foreach(QString gesture, availableGestures)
    {
        QTreeWidgetItem *newItem = new QTreeWidgetItem(ui->keyMap);
        newItem->setText(0,gesture);
    }
}
```

```
        newItem->setTextAlignment(1, Qt::AlignHCenter);
        newItem->setFlags(newItem->flags() | Qt::ItemIsEditable);
    }
}

SimulateKeyboard::~SimulateKeyboard()
{
    delete ui;
}
```

## F. Appendix

```
#define SET(x,y) (x |= (1<<y))           //-Bit set/clear macros
#define CLR(x,y) (x &= (~(1<<y)))        // |
#define CHK(x,y) (x & (1<<y))           // |
#define TOG(x,y) (x^=(1<<y))            //-+

#define N 160    //-How many frequencies

float results[N];           //-Filtered result buffer
float freq[N];              //-Filtered result buffer
int sizeofArray = N;

#include <SoftwareSerial.h>

SoftwareSerial mySerial(3, 4); // RX, TX

void setup()
{

    TCCR1A=0b10000010;      //-Set up frequency generator
    TCCR1B=0b00011001;      //-+
    ICR1=125;
    OCR1A=62;

    pinMode(9,OUTPUT);       //-Signal generator pin
    pinMode(8,OUTPUT);       //-Sync (test) pin
    pinMode(2, OUTPUT);

    Serial.begin(57600);     //-115200

    Serial.println("Goodnight moon!");

    // set the data rate for the SoftwareSerial port
    mySerial.begin(9600);
    mySerial.println("AT");

    for(int i=0;i<N;i++)     //-Preset results
        results[i]=0;       //-+
}

void loop()
{
    if (mySerial.available())
        Serial.write(mySerial.read());
    if (Serial.available())
        mySerial.write(Serial.read());
}

void loop2()
{
    unsigned int d;

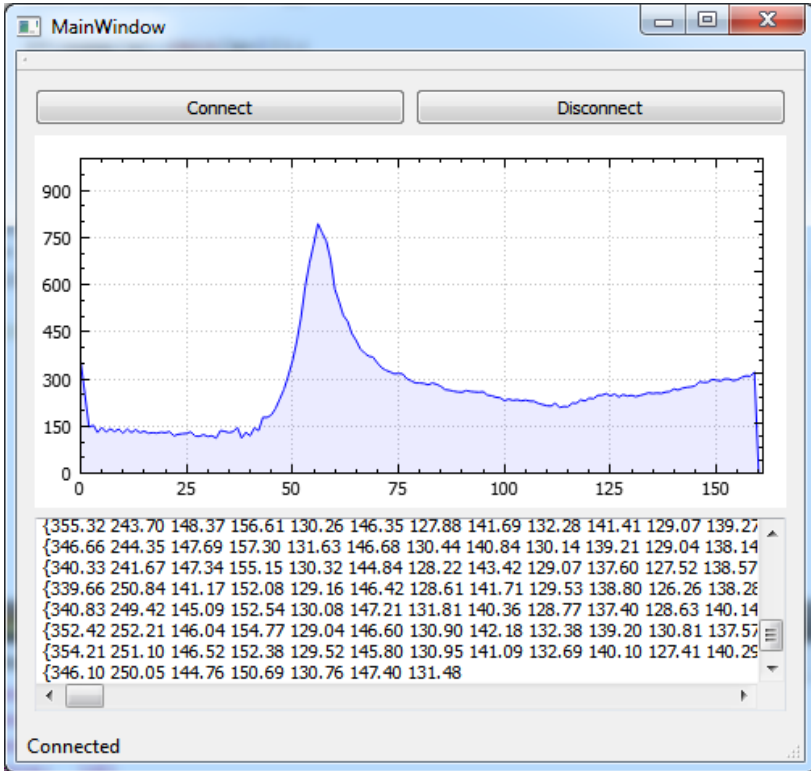
    int counter = 0;
    for(unsigned int d=0;d<N;d++)
    {
        int v=analogRead(0);   //-Read response signal
        CLR(TCCR1B,0);         //-Stop generator
        TCNT1=0;               //-Reload new frequency
        ICR1=d;                // |
        OCR1A=d/2;             //-+
        SET(TCCR1B,0);         //-Restart generator

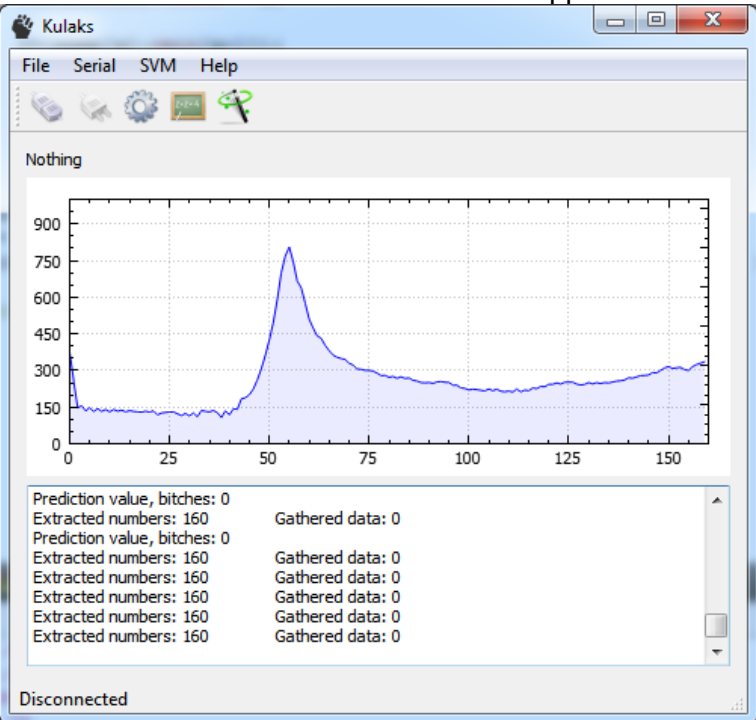
        results[d]=results[d]*0.5+(float)(v)*0.5; //-Filter results

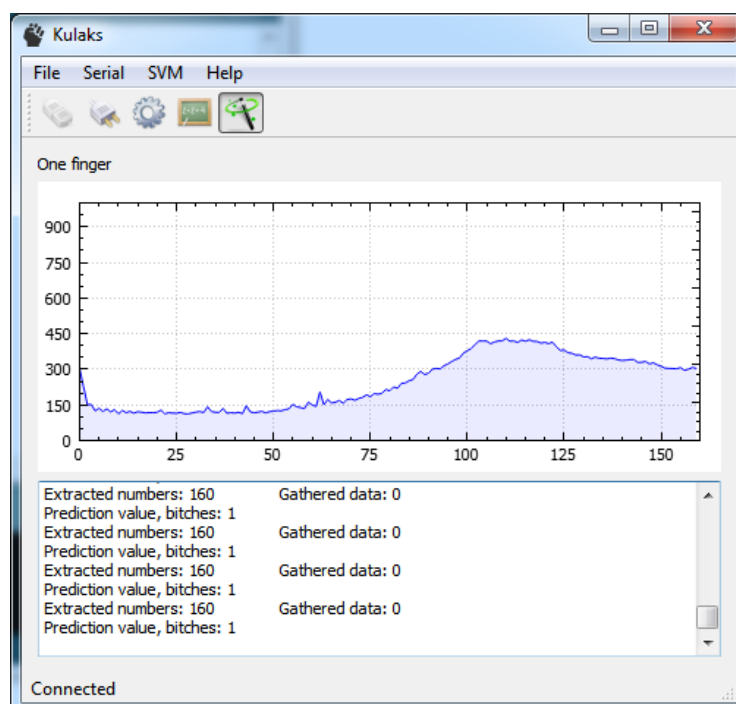
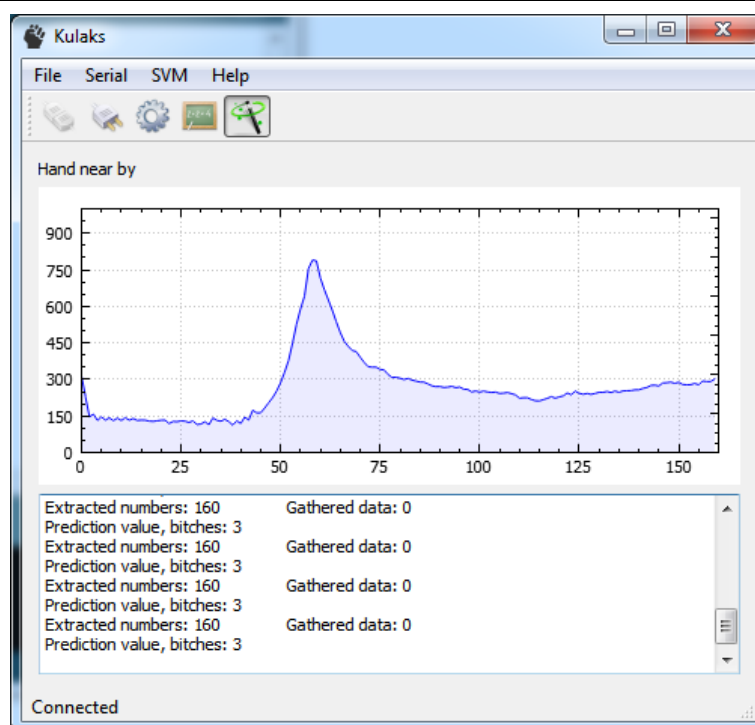
        freq[d] = d;
    }
}
```

```
Serial.print("{");  
  for (unsigned int d=0; d<N; d++) {  
    Serial.print(results[d]);  
    Serial.print(" ");  
  }  
  Serial.print("}");  
  Serial.println();  
  
}
```

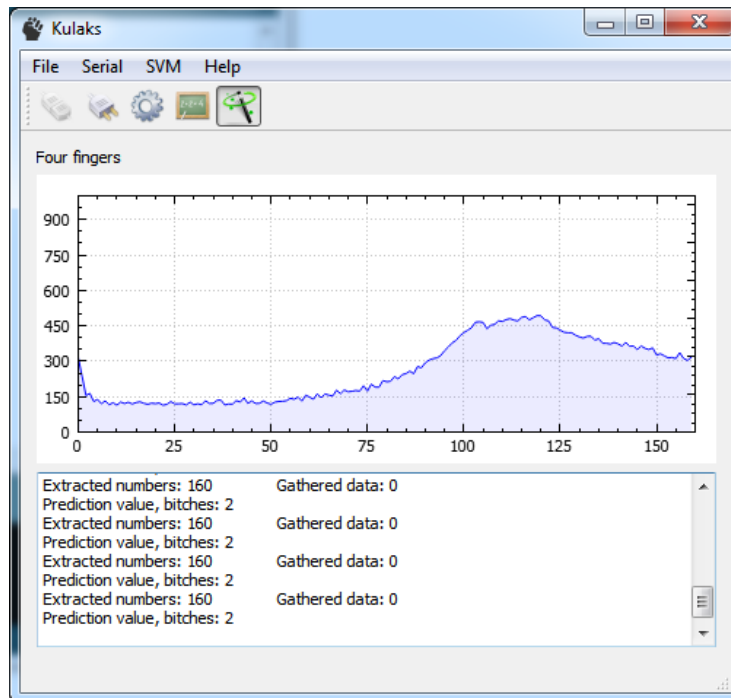
## 7 Logbooks

Date	Work done and/or meeting minutes.
24rd Sep 2013	Just discussed, the plan of "attack". In other words, how we will meet, and what info I've got to provide for my lecturer. And signed the project allocation form.
1 Oct 2013	<p>Nothing happened.</p> <p>Started gathering theoretical materials for my project. Found the original paper which describes in high detail how they achieved their goal.  <a href="http://www.disneyresearch.com/wp-content/uploads/touchechi2012.pdf">http://www.disneyresearch.com/wp-content/uploads/touchechi2012.pdf</a>  <a href="http://spritesmods.com/?art=engarde&amp;page=1">http://spritesmods.com/?art=engarde&amp;page=1</a></p> <p>Tried to implement first prototype in arduino, and wrote small plotting tool on Qt for it.</p>  <p>(note: not the actual plotted results from back then)</p> <p>Conclusion: Rate of change was barely noticeable with the schematics I found on the second link. It might be due to fact, that I'm using a LED instead of high frequency diode in my schematic. Just didn't have one, so I thought I would use LED. In any case, ordered some 1n4147 diodes and couple 10mH coils, for noise filtering.</p>
8 Oct 2013	<p>Project manager was busy.</p> <p>Still waiting for my diodes.</p>
18 Oct	My diodes and coils have arrived. Now I can reconstruct the schematics like

2013	<p>provided in this website:  <a href="http://www.instructables.com/id/Touche-for-Arduino-Advanced-touch-sensing/?ALLSTEPS">http://www.instructables.com/id/Touche-for-Arduino-Advanced-touch-sensing/?ALLSTEPS</a></p> <p>Now the rate of change is visible much more. I increased from 32 data points to 160 frequencies now. It is easy to spot the difference when multiple gestures are applied on my test platform. Now it's time to revise SVM and continue writing the PC side of application.</p> <p>Fixed a bug in my plotting tool. It wasn't consistently reading 160 values from rs232. It was due to the fact I was using buffer to store my data, and I didn't design correct algorithm to extract exactly 160 numbers from it. I forgot to take into account that data in buffer arrives in different rates. Now it's fixed.</p>
20 Oct 2013	<p>After some frustration with LIBSVM library managed to understand how to interact with it. Most of the instructions are how to use pre-built tools, not how to implement in your application. In any case over the weekend constructed a working prototype which can differentiate between fours gestures. There is one small issue when trying to differentiate between two or four fingers, the data is very similar and SVM sometimes fails to give the correct result.</p> <p>Here are some screenshots from current application:</p> 







After weekend of work, I've managed to make pattern recognition for these gestures

- Nothing
- Hand near by
- One finger
- Two fingers
- Four fingers

However, two fingers can't be distinguished easily from one or four, and a lot of times gives false positives. I could try to make sinusoidal waveform generator and sweep more frequencies, theoretically, it should improve the response.

Here you can see a video of first prototype in action:

<http://www.youtube.com/watch?v=xGhG-vS4PJw>

Things still to do:

- Improve the response of the gestures.
- Completely integrate libsvm in my application
- Fix a bug: after 3 min of data gathering, my app just starts flickering
- Design a PCB

29 Oct  
2013

Had a quick meeting with supervisor about progress, tried to show the project, didn't work as expected, apparently different environments have different levels of noise

2 Oct - 4  
Oct 2013

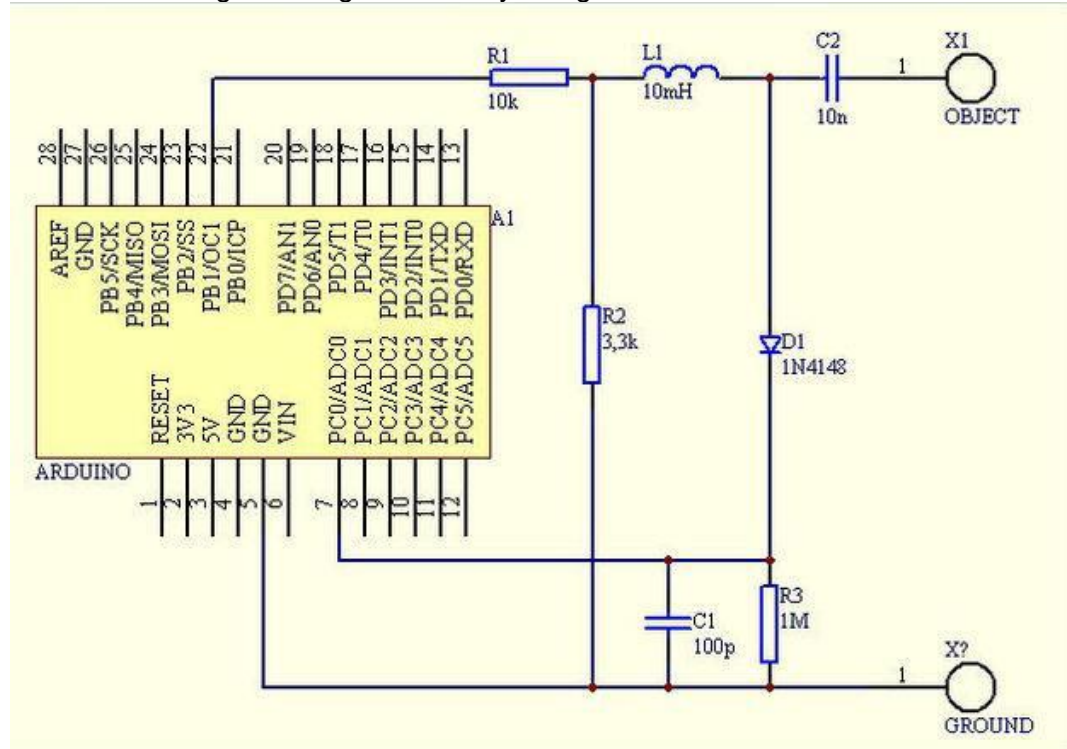
Improved the application, before it was crashing after a while, apparently I made a mistake when plotting a graph. Basically I was making new graph object on top of previous one, and eventually, constructed thousands of them, which crashed the application. Now I'm reusing the previous one.

Improved the gesture teaching tool. Now you can teach the gestures without

existing application, and no alteration needed afterwards. The files it will generate can be automatically run through the SVM.

Added SVM model loader, so for different devices or environments I can load the corresponding model quickly.

Circuit used for generating so far in my design:



Source: <http://www.instructables.com/file/FR73R4DH2MYISBD>

It is not the final circuit I'll be using, since eventually I will be switching to AC generator. So it will be slightly different.

As far I understand the circuit it consists of RLC resonant circuit.

source: <http://www.electronics-tutorials.ws/accircuits/series-circuit.html>

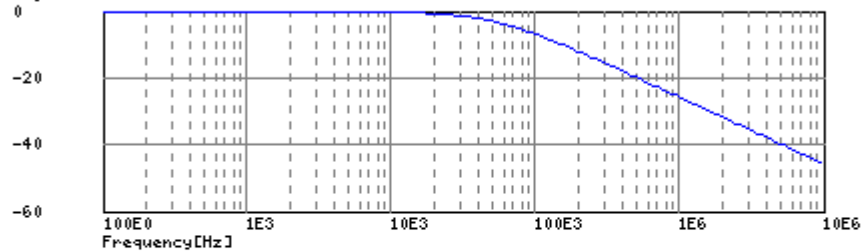
Which is a second-order filter, since I'm using two active components for energy storage, the inductor and capacitor. When the finger is placed on the electrode, the capacitance is increased, and a different frequency has the highest pass-through.

With resonant frequency  $f_c = 52521.1312203[\text{Hz}]$  (with no touch)

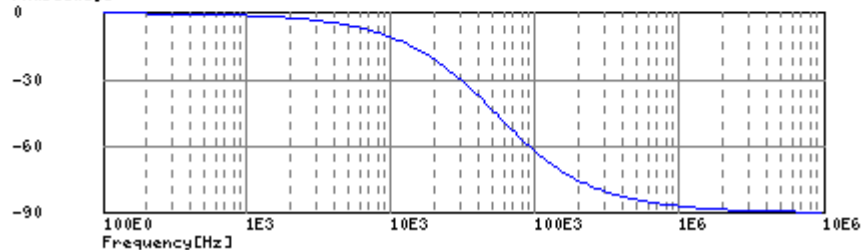
## Frequency analysis

BodeDiagram

Magnitude[dB]



Phase[deg]



(c)okawa-denshi.jp

This is how I've set up the wave generator in AVR

```
TCCR1A=0b10000010;    //-Set up frequency generator
TCCR1B=0b00011001;    //-+
```

Basically it is set in Fast PWM mode, with TOP being ICR1A register

And compare mode is set to:

*Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM, (non-inverting mode)*

And clock selected without pre-scaler.

For AVR it means that the timer works with FCPU frequency, which in my case is 16 MHz. Also I'm taking 160 different frequencies. That means  $16\text{MHz} / 160 = 100\text{ kHz}$  as one step frequency. Hence, I'm sweeping from 0Hz up to 16MHz

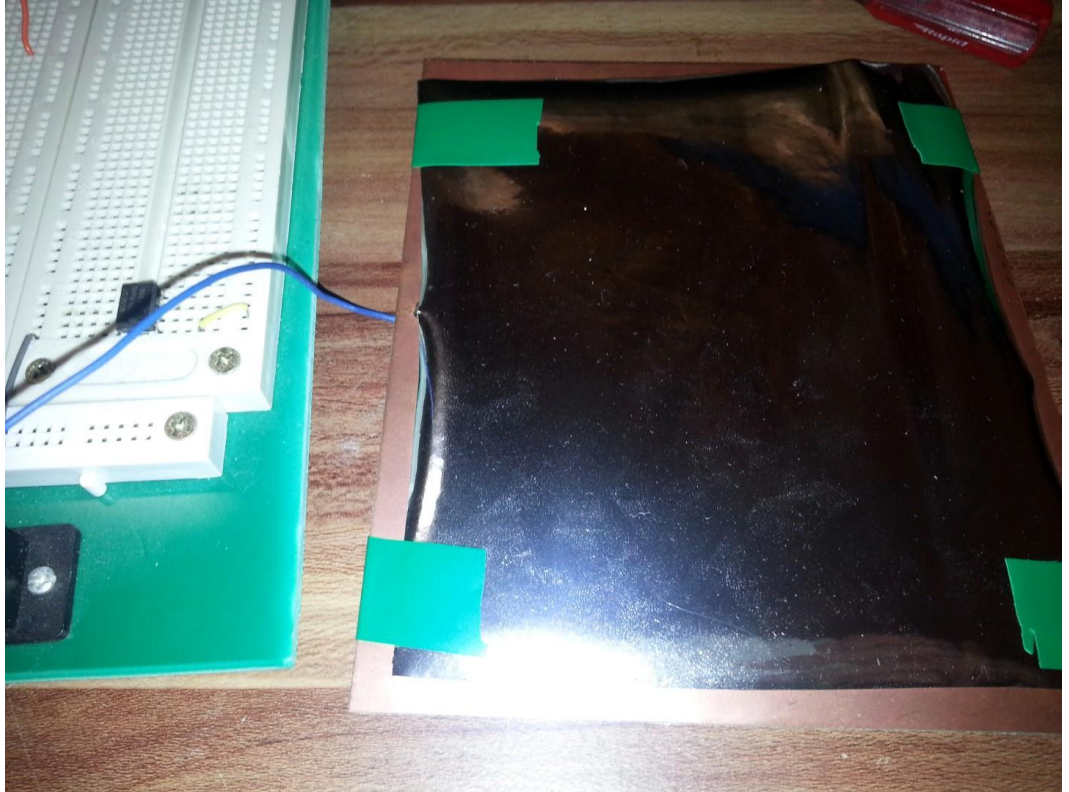
5 Nov  
2013

Had a quick discussion with lecturer what is the next step. We discussed that it would be nice to have a FFT analysis. which I will do at the beginning of next week. As well over the weekend I did some improvement to my "kulaks" application.

Basically I added new module "playground" where I intend to make sample application for the gesture control. As of now, I've made "keyboard simulator" which doesn't work as good as I expected. Apparently there are limitations when it comes to simulating a keyboard on windows. As far as I know the default library only allows to simulate keyboard stroke event, but I wanted a keyboard press event. I already lost more time on this than I wanted, so I will be coming back to this a little bit later, since this isn't an integral part of my project. At least now. Still waiting for my AC signal generator, should arrive next week, and then I'll be

able to experiment with touch sensing on human body.

I did an experiment how the touch sensing will change when a glass of water is used for touch sensing. I noticed that the response significantly increased if a dielectric material is between the finger and water, in my case glass. Fortunately for me, one of my coursemates, was experimenting with mylar aka. *BoPET polyester film*, which has high dielectric properties and is commonly used in foil capacitors. So I borrowed some of it and placed on top of my PCB aka. touch sensing platform, and observed the effects.



### Results:

The response, just like for glass significantly increased, and even with a square wave (PWM) frequency sweep I could detect these gestures: hand near by, one finger touch, two finger touch, three finger touch, four/five finger touch and a palm touch. Before I could barely detect one, two or four fingers. Now with ease those gesture can be distinguished.

As far as I understand why does the performance increases, is that the mylar shields the user from the PCB plate, and literally makes the PCB plate and user act as two capacitor plates. Before when user was touching the PCB plate it was basically grounding the "plate". However, regarding this article:

<http://hyperphysics.phy-astr.gsu.edu/hbase/tables/diel.html>

Air has much better dielectric constant, but as far as I know it is very hard to place a layer of air between the PCB and my finger. Teflon could be used instead of mylar and could have better performance.

For experimentation purpose I'll buy some sheets of teflon, just to see how the capacitive fingerprint will change.

Further reading about electromagnetism and capacitance:

	<a href="http://www.lightandmatter.com/html_books/0sn/ch11/ch11.html">http://www.lightandmatter.com/html_books/0sn/ch11/ch11.html</a>																																																																								
12 Nov 2013	Just a quick chat about the next step. And REPORT.																																																																								
19 Nov 2013	<p>Got my ethical review signed.</p> <p>Received bluetooth for my project.</p> <p>Basically I bought HC-06 bluetooth. <a href="http://mcuoneclipse.com/2013/06/19/using-the-hc-06-bluetooth-module/">http://mcuoneclipse.com/2013/06/19/using-the-hc-06-bluetooth-module/</a> <a href="http://www.exp-tech.de/service/datasheet/HC-Serial-Bluetooth-Products.pdf">http://www.exp-tech.de/service/datasheet/HC-Serial-Bluetooth-Products.pdf</a></p> <p>It's a very cheap bluetooth with UART interface with SPP (serial port profile) support. It enables me to have a serial port via bluetooth on my computer. Just like connecting the micro-controller to the PC with USB.</p> <p>I decided to go for bluetooth, so I can make sample application on the smart-phones as well. Obviously the "best" way is to use some sort of wireless technology to do that. Just a subjective opinion, who would want more wires sticking from their phone anyway? Inevitably, I chose bluetooth.</p> <p>By default the HC-06 works in baudrate 9600. Which is very slow! Too slow for my project, since I'm sending 160 numbers through it. More precisely <math>160+2+159=321</math> bytes</p> <p>Roughly baud rate of 9600 can transmit 1200 bytes/s That means <math>1200 / 321 = \sim 3.71</math> refreshes a second. Basically unusable for my application.</p> <p>So I investigated whether it is possible to reprogram the HC-06 bluetooth adapter. And lucky for me, it is.</p> <p>Basically, before connecting any bluetooth device to the adapter, from micro-controller I have to send the following command "AT" to enter program mode. Notice, there are no carriage return and new line character. Then you can all sorts of things, change the baudrate of the device, the name or the PIN, and the it will be saved in internal eeprom memory of the bluetooth dongle. So it has to be done only once.</p> <p>Here is list of commands you can send to the HC-06:</p> <table><tr><th></th><th>A</th><th>B</th><th>C</th></tr><tr><th>1</th><th>Command</th><th>Response</th><th>Comment</th></tr><tr><td>2</td><td>AT</td><td>OK</td><td>Used to verify communication</td></tr><tr><td>3</td><td>AT+VERSION</td><td>OKlinvorV1.8</td><td>The firmware version (version might depend on firmware)</td></tr><tr><td>4</td><td>AT+NAMExyz</td><td>OKsetname</td><td>Sets the module name to "xyz"</td></tr><tr><td>5</td><td>AT+PIN1234</td><td>OKsetPIN</td><td>Sets the module PIN to 1234</td></tr><tr><td>6</td><td>AT+BAUD1</td><td>OK1200</td><td>Sets the baud rate to 1200</td></tr><tr><td>7</td><td>AT+BAUD2</td><td>OK2400</td><td>Sets the baud rate to 2400</td></tr><tr><td>8</td><td>AT+BAUD3</td><td>OK4800</td><td>Sets the baud rate to 4800</td></tr><tr><td>9</td><td>AT+BAUD4</td><td>OK9600</td><td>Sets the baud rate to 9600</td></tr><tr><td>10</td><td>AT+BAUD5</td><td>OK19200</td><td>Sets the baud rate to 19200</td></tr><tr><td>11</td><td>AT+BAUD6</td><td>OK38400</td><td>Sets the baud rate to 38400</td></tr><tr><td>12</td><td>AT+BAUD7</td><td>OK57600</td><td>Sets the baud rate to 57600</td></tr><tr><td>13</td><td>AT+BAUD8</td><td>OK115200</td><td>Sets the baud rate to 115200</td></tr><tr><td>14</td><td>AT+BAUD9</td><td>OK230400</td><td>Sets the baud rate to 230400</td></tr><tr><td>15</td><td>AT+BAUDA</td><td>OK460800</td><td>Sets the baud rate to 460800</td></tr><tr><td>16</td><td>AT+BAUDB</td><td>OK921600</td><td>Sets the baud rate to 921600</td></tr><tr><td>17</td><td>AT+BAUDC</td><td>OK1382400</td><td>Sets the baud rate to 1382400</td></tr></table>		A	B	C	1	Command	Response	Comment	2	AT	OK	Used to verify communication	3	AT+VERSION	OKlinvorV1.8	The firmware version (version might depend on firmware)	4	AT+NAMExyz	OKsetname	Sets the module name to "xyz"	5	AT+PIN1234	OKsetPIN	Sets the module PIN to 1234	6	AT+BAUD1	OK1200	Sets the baud rate to 1200	7	AT+BAUD2	OK2400	Sets the baud rate to 2400	8	AT+BAUD3	OK4800	Sets the baud rate to 4800	9	AT+BAUD4	OK9600	Sets the baud rate to 9600	10	AT+BAUD5	OK19200	Sets the baud rate to 19200	11	AT+BAUD6	OK38400	Sets the baud rate to 38400	12	AT+BAUD7	OK57600	Sets the baud rate to 57600	13	AT+BAUD8	OK115200	Sets the baud rate to 115200	14	AT+BAUD9	OK230400	Sets the baud rate to 230400	15	AT+BAUDA	OK460800	Sets the baud rate to 460800	16	AT+BAUDB	OK921600	Sets the baud rate to 921600	17	AT+BAUDC	OK1382400	Sets the baud rate to 1382400
	A	B	C																																																																						
1	Command	Response	Comment																																																																						
2	AT	OK	Used to verify communication																																																																						
3	AT+VERSION	OKlinvorV1.8	The firmware version (version might depend on firmware)																																																																						
4	AT+NAMExyz	OKsetname	Sets the module name to "xyz"																																																																						
5	AT+PIN1234	OKsetPIN	Sets the module PIN to 1234																																																																						
6	AT+BAUD1	OK1200	Sets the baud rate to 1200																																																																						
7	AT+BAUD2	OK2400	Sets the baud rate to 2400																																																																						
8	AT+BAUD3	OK4800	Sets the baud rate to 4800																																																																						
9	AT+BAUD4	OK9600	Sets the baud rate to 9600																																																																						
10	AT+BAUD5	OK19200	Sets the baud rate to 19200																																																																						
11	AT+BAUD6	OK38400	Sets the baud rate to 38400																																																																						
12	AT+BAUD7	OK57600	Sets the baud rate to 57600																																																																						
13	AT+BAUD8	OK115200	Sets the baud rate to 115200																																																																						
14	AT+BAUD9	OK230400	Sets the baud rate to 230400																																																																						
15	AT+BAUDA	OK460800	Sets the baud rate to 460800																																																																						
16	AT+BAUDB	OK921600	Sets the baud rate to 921600																																																																						
17	AT+BAUDC	OK1382400	Sets the baud rate to 1382400																																																																						

source: <http://mcuoneclipse.com/2013/06/19/using-the-hc-06-bluetooth-module/>

To do the configuration of the HC-06 I wrote a quick arduino code, to utilize the SoftwareSerial library. Basically through normal serial I can do HC-06 programming. It all worked perfectly

Now I can communicate with bluetooth serial port with baud rate of 115000 Which means I can send 14375 bytes/s and my packet can be sent  $321/14375 = 0.0223304348 \text{ s} = 22\text{ms}$ . It is better, but still rather slow.

So I decided to increase the speed even more. To a baud rate of 230400. I have never seen such a baudrate in my life, probably it isn't a standard one.

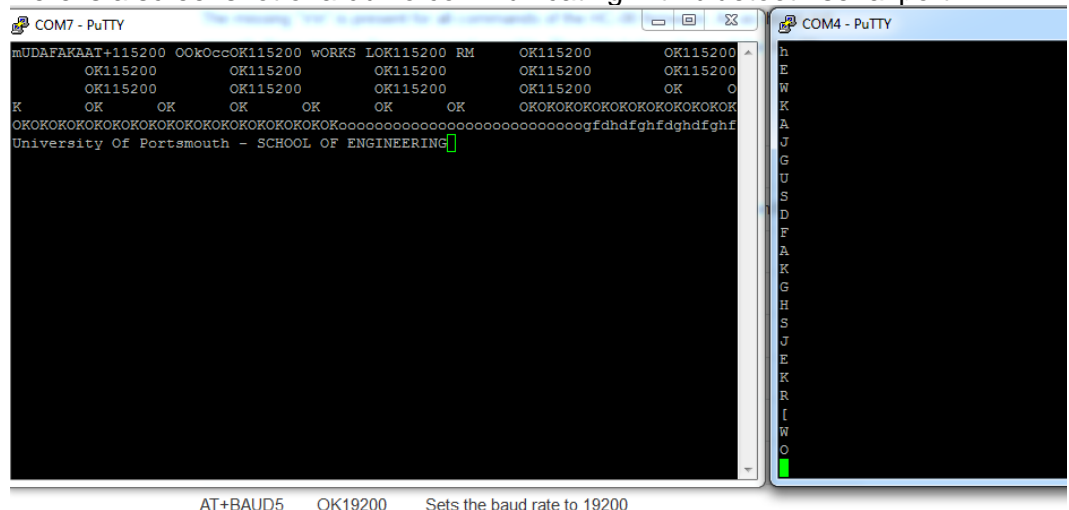
Well I was correct, as soon as I changed it I couldn't communicate with the HC-06 anymore through arduino. I should have been more careful, because then I found this chart: <http://www.wormfood.net/avrbaudcalc.php> Which says that for 16MHz clock that baudrate has really high error rate of 7.8%, and it is considered unusable.

So I tried resetting the bluetooth, didn't WORK. By triggering Key pin, apparently it's not connected. So device goes in 230400 baud rate mode.

Got an Idea how to fix it. I have a FTDI chip laying around, I think I can interface it directly with the bluetooth adapter and from putty I can set whatever baudrate I like. Chance of fixing it.

Yup, I was correct, by simply interfacing the FTDI232L with the HC-06 I was able to communicate in that bizarre baud rate and set it back to 115200. For a split second I thought I'll need a new bluetooth. Just have to sit down and think a bit, and anything can be solved.

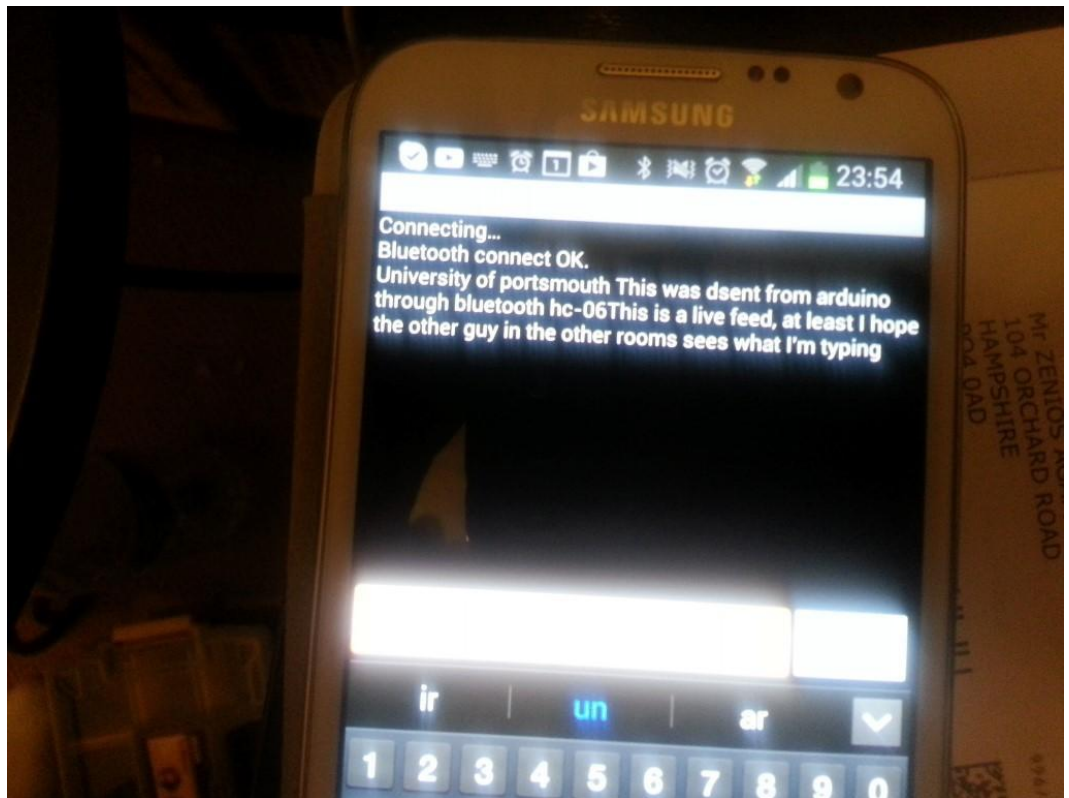
Here is a screenshot of arduino communicating with bluetooth serial port:



Lesson learned, be careful when experimenting with baudrates, better check whether the baudrate has error rate in acceptable norms.

And I tested the SPP mode on android phone as well.  
Here is picture of data being received from arduino:





you will just have to take my word that I sent it from arduino.

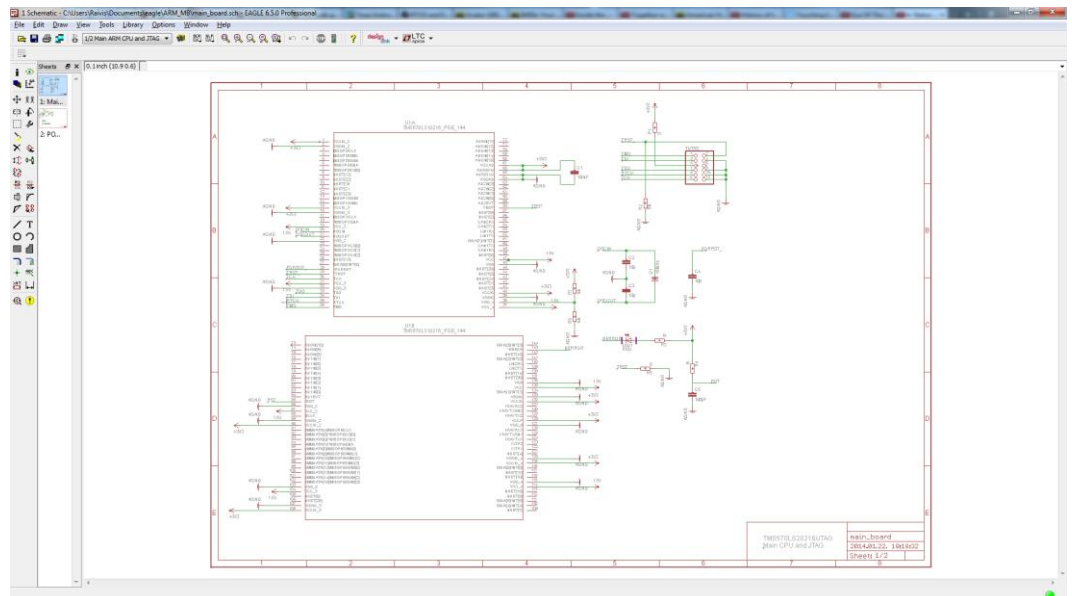
At least I've proved that the concept works, the next step would be to start putting the big pieces together. Unfortunately iPhone asks for a licence to RCOMM protocol. So for prototyping reasons I will be avoiding "money hungry" apple products. However, if I eventually use bluetooth with HID profile, it could be connected without to any smart phone with bluetooth and use basic phone functions instead of writing a specific interface application.

Date	Work done and/or meeting minutes.
13/12/13-01/01/14	<p>Successfully interfaced the sine wave function generator with my arduino. The function generator basically receives the inputs through something similar to SPI protocol. The function generator I'm using is called AD9850 it can generate pure sine wave up to 50 Mhz.</p> <p>Obviously I wrote a simple library for interfacing the AD9850 function generator.</p> <p>However, by using a function generator I will need to change my circuit a little bit, because when the user touches the micro-controller receives a sine wave back into the adc. The solution is to use envelope detection to only send amplitude of the sine wave to the micro-controllers adc.</p>
02/01/14-25/01/14	<p>I've been working on designing the main motherboard for my project which will consist of ARM TMS570 cortex r4 32 bit microprocessor. I intend to do pattern recognition on the main board itself instead of the computer or smartphone. To do so I acquired a development board from texas instruments for that particular</p>

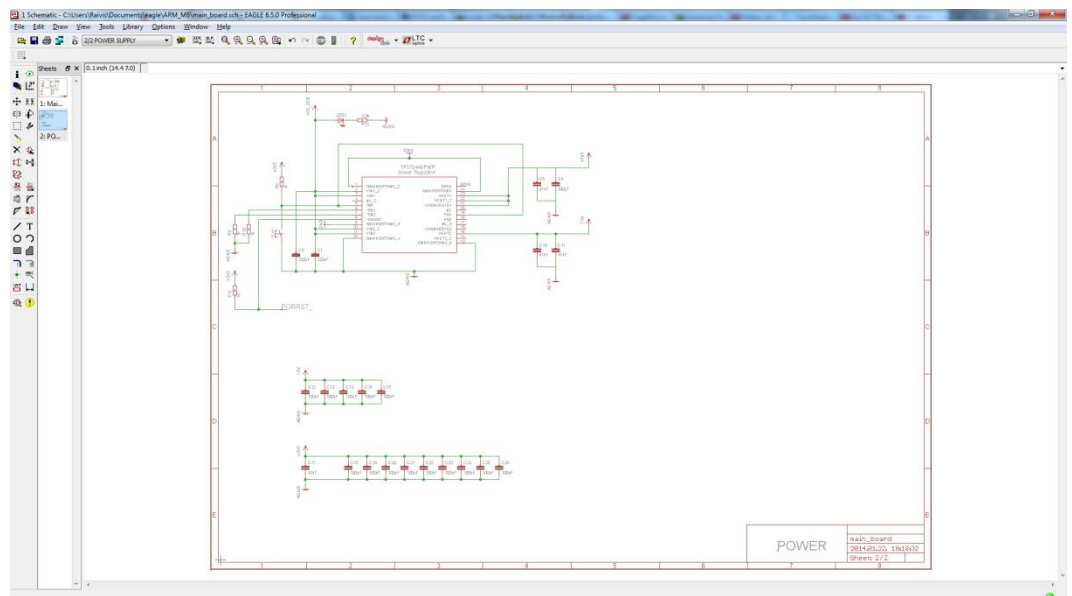
CPU.

I've successfully managed to interface it with the function generator and bluetooth module. I still need to interface a RAM module with the cpu so the board has enough memory to do the pattern recognition otherwise it only has 160kb of memory available, which is not nearly enough.

Parallel to experimenting with the development kit, I've been designing the circuit diagram on eagle. So far I've finished the the basic interfacing with the ARM CPU and power supply.



Main cpu circuit



Power supply circuit

The circuit design is mainly based on the development board I acquired from TI



	<p>(texas instruments). They were kind enough to release the full circuit diagram for the development kit.</p> <p>As of now I've been struggling a lit bit to get ERIF module to work on the CPU, which is responsible for interfacing with SDRAM + RTOS operating system.</p> <p>Even though an operating system in my opinion is not an essential part of the design of the touch sensor, it will ease the development considerably.</p>
--	--

## 8 Bibliography

- 1 admin. (2007, 3 29). *Clocking ARM with Crystal oscillator and PLL* . Retrieved 1 2014, from ARM GCC Tutorial: <http://winarm.scienceprog.com/arm-mcu-types/clocking-arm-with-crystal-oscillator-and-pll.html>
- 2 Admin. (2011, 3 10). *Using standard IO stream in AVR GCC*. Retrieved 3 2014, from EMBEDS: <http://www.embedds.com/using-standard-io-streams-in-avr-gcc/>
- 3 Aisen, B. (2006, 12 15). *A Comparison of Multiclass SVM Methods*. Retrieved 09 28, 2013, from MIT: <http://courses.media.mit.edu/2006fall/mas622j/Projects/aisen-project/>
- 4 Atmel. (2013). *8/16-bit Atmel AVR xmega microcontrollers*. Retrieved 02 05, 2014, from Atmel: [http://www.atmel.com/Images/Atmel-8153-8-and-16-bit-AVR-Microcontroller-XMEGA-E-ATxmega8E5-ATxmega16E5-ATxmega32E5\\_Datasheet.pdf](http://www.atmel.com/Images/Atmel-8153-8-and-16-bit-AVR-Microcontroller-XMEGA-E-ATxmega8E5-ATxmega16E5-ATxmega32E5_Datasheet.pdf)
- 5 Ben-Hur, A., & Weston, J. (NA). *A User's guide to support vector machines*. Colorado State University.
- 6 Camera, D. (2009, 12 16). *Open Source XMEGA PDI Programmer!* Retrieved 2 2014, from Memos From The Cube: <http://fourwalledcubicle.com/blog/2009/12/open-source-xmega-pdi-programmer/>
- 7 Composition of the ESPEN Working Group. (2004). *Bioelectrical impedance analysis - part1: review of principles and methods*. Clinical Nutrition.
- 8 Crowell, B. (n.d.). *Light And Matter*. Retrieved 11 12, 2013, from Electromagnetism: [http://www.lightandmatter.com/html\\_books/Osn/ch11/ch11.html](http://www.lightandmatter.com/html_books/Osn/ch11/ch11.html)
- 9 Disney. (2013). *Touché: Enhancing Touch Interaction on Humans, Screens, Liquids, and Everyday Objects*. Retrieved 09 29, 2013, from Disney: <http://www.disneyresearch.com/wp-content/uploads/touchechi2012.pdf>
- 10 GeekPhysical. (2012, 06 24). *Arduino do the Touché dance* . Retrieved 10 2013, from Dzl's Evil Genius Lair: <http://dzlsevilgeniuslair.blogspot.se/2012/05/arduino-do-touche-dance.html>
- 11 Google. (2013, 11 15). *BluetoothSocket*. Retrieved 11 15, 2013, from Developers: <http://developer.android.com/reference/android/bluetooth/BluetoothSocket.html>
- 12 Goudge, L. (2013, 9 28). *AD9850 function generator SPI driver*. Retrieved 11 2013, from mbed: <http://mbed.org/users/liamg/code/AD9850-function-generator-SPI-driver/>
- 13 Iwata, T., Houlshby, N., & Ghahramani, Z. (NA). *Active Learning for Interactive Visualization*. Cambridge: University of Cambridge.
- 14 Lin, C.-C. C.-J. (2013, 04 01). *A Library for Support Vector Machines*. Retrieved 09 29, 2013, from LIBSVM: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

- 15 Maxim Integrated. (2001, 9 7). *EPOT Applications: Offset Adjustment in Op-Amp Circuits*. Retrieved 4 2014, from Maxim Integrated:  
<http://www.maximintegrated.com/app-notes/index.mvp/id/803>
- 16 Microsoft. (n.d.). *The Standard SVM Formulation*. Retrieved 09 28, 2013, from Microsoft: <http://research.microsoft.com/en-us/um/people/manik/projects/trade-off/svm.html>
- 17 Mitchel, C. (n.d.). *50 - 555 Circuits*. Retrieved 3 2014, from talking electronics:  
<http://www.talkingelectronics.com/projects/50%20-%20555%20Circuits/50%20-%20555%20Circuits.html>
- 18 NA. (n.d.). *AD9850 Library*. Retrieved 11 2013, from GRSynth:  
<http://grsynth.com/ad9850-library/>
- 19 NA. (n.d.). *Envelope Detector*. Retrieved 3 2014, from  
<http://seniord.ee.iastate.edu/SSOL/RADAR/prjpln99/detector3.html>
- 20 NA. (2012, 10 13). *Float/Double printf(), scanf() in ATME Studio 6*. Retrieved 3 2014, from Rohde Engineering Blog: [http://blog.ib-rohde.de/printf\\_float\\_atmelstudio6/](http://blog.ib-rohde.de/printf_float_atmelstudio6/)
- 21 NA. (n.d.). *Passive Band Pass Filter*. Retrieved 11 2013, from Electronics Tutorial:  
[http://www.electronics-tutorials.ws/filter/filter\\_4.html](http://www.electronics-tutorials.ws/filter/filter_4.html)
- 22 NA. (n.d.). *Peak detector circuit*. Retrieved 1 2014, from Electronics:  
<http://www.proelectron.com/elect/>
- 23 palowireless. (2007, 01 05). *Bluetooth Tutorial - Profiles*. Retrieved 11 9, 2013, from palowireless - Bluetooth Resource Center:  
<http://www.palowireless.com/infotooth/tutorial/profiles.asp>
- 24 Pratt, S. (2006, 10). *Capacitance Sensors for Human Interfaces to Electronic Equipment*. Retrieved 10 2013, from Analog Dialogue:  
[http://www.analog.com/library/analogdialogue/archives/40-10/cap\\_sensors.html](http://www.analog.com/library/analogdialogue/archives/40-10/cap_sensors.html)
- 25 Quispe-Ayala, M. R., Asalde-Alvarez, K., & Roman-Gonzalez, A. (2010). *Image Classification Using Data Compression*. Paris: Universidad Nacional San Antonio Abad del Cusco – UNSAAC.
- 26 Rekimoto, J. *SmartSkin: An Infrastructure for Freehand Manipulation on Interactive Surfaces*. Tokyo: Sony Computer Science Laboratories, Inc.
- 27 Sprite. (2013). *En Garde, a classifying capacitive touch sensor - Intro*. Retrieved 11 5, 2013, from SpritesMods: <http://spritesmods.com/?art=engarde>
- 28 Styger, E. (2013, 7 19). *Using the HC-06 Bluetooth Module*. Retrieved 11 2013, from mcuoneclipse: <http://mcuoneclipse.com/2013/06/19/using-the-hc-06-bluetooth-module/>

- 29 SZu. (2012, 8 15). *ATxmega programmer for \$0.50* . Retrieved 2 2014, from ATxmega and company: <http://szulat.blogspot.co.uk/2012/08/atxmega-programmer-for-050.html>
- 30 Texas Instruments . (n.d.). *TMS570LS Microcontrollers:Blinky example*. Retrieved 1 2014, from TI:  
[ftp://entc.tamu.edu/ENTC369/Laboratory/Reference/TMS570/TMS570LS2x\\_Blinky\\_Example.pdf](ftp://entc.tamu.edu/ENTC369/Laboratory/Reference/TMS570/TMS570LS2x_Blinky_Example.pdf)
- 31 Texas Instruments. (2011, 9). *Configuring the Hercules™ ARM® Safety MCU SCI/LIN Module for UART Communication*. Retrieved 1 2014, from TI:  
<http://www.ti.com/lit/an/spna124a/spna124a.pdf>
- 32 tobias. (2006, 11 11). *AD9833 / AD5932 Interface*. Retrieved 11 2013, from Tobiscorner:  
<http://tobiscorner.floery.net/ad9833-ad5932-interface/>
- 33 USB Implementers' Forum. (2001, 06 27). *Device Class Definition for Human Interface Devices (HID)*. Retrieved 11 12, 2013, from USB:  
[http://www.usb.org/developers/devclass\\_docs/HID1\\_11.pdf](http://www.usb.org/developers/devclass_docs/HID1_11.pdf)
- 34 Viktoriia Sharmanska, Novi Quadrianto, & Lampert, C. H. (2010). *Augmented Attribute Representations*. Cambridge: University of Cambridge.