

**Robert Corby, Adrian Kelly**

**Salesian Secondary College, Pallaskenry, Co.Limerick**

Pulse Oximeter

# Comments Page

**Judge’s Signatures: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Contents Page

[Comments Page 1](#_Toc376370750)

[Summary/Abstract 3](#_Toc376370751)

[Introduction 6](#_Toc376370752)

[History of Pulse Oximetry 8](#_Toc376370753)

[Principles of Pulse Oximetry 9](#_Toc376370754)

[Medical Use 12](#_Toc376370755)

[Technology 14](#_Toc376370756)

[Web Platform 15](#_Toc376370757)

[Microcontroller platform 18](#_Toc376370758)

[PSoC 19](#_Toc376370759)

[Pulse Oximeter 21](#_Toc376370760)

[ADC and amplifiers 21](#_Toc376370761)

[Filters 22](#_Toc376370762)

[Autocorrelation 23](#_Toc376370763)

[Probe 28](#_Toc376370764)

[Results and Conclusions 30](#_Toc376370765)

[Results 30](#_Toc376370766)

[Conclusions 35](#_Toc376370767)

[Project Management 35](#_Toc376370768)

[Challenges 36](#_Toc376370769)

[Acknowledgements 37](#_Toc376370770)

[Appendix: PSOC Code 38](#_Toc376370771)

[Appendices: 47](#_Toc376370772)

[Bibliography 52](#_Toc376370773)

# Summary/Abstract

A pulse oximeter measures oxygen saturation and blood flow. This works by using a technique called photoplethysmography.

Pulse oximetry is an important part of the WHO’s surgical safety checklist which has been shown to reduce complications and death by over 30% (Source WHO: [www.who.int](http://www.who.int)). In sub Saharan Africa up to 70% of operating theatres don't have access to pulse oximeters that could help minimise deaths by alerting doctors to the low oxygen levels.

In our project we set out to achieve the design and construction of a pulse oximeter that is very compact, non-invasive and easy to use.

Our Pulse Oximeter system consists of a probe, electronics and a local web server running on a laptop. The system uses Wi-Fi communication so that it is portable. By using the laptops battery power and a local area network, this makes it suitable for a region where electricity and internet access are limited. Our system is based on a Wi-Fi enabled microcontroller connected to a probe which we programmed to get the pulse and the oxygen levels.

We have a website which both doctors and patients can use to track their pulse and blood oxygen saturation levels easily. It includes a homepage, overview and login page. We have an accounts database so that both patients and doctors can log on and update t­­­­­heir status. It is programmed in HTML and PHP and it graphs and clearly shows oxygenated: deoxygenated haemoglobin ratio, bpm and plethysmograph.

We designed the housing for own probe using 123D Design software and then got it 3D printed.

The pulse oximeter works by calculating the level of oxygenation of blood (SaO2) based on the intensity of light that has been absorbed by body tissue. Human tissue absorbs different amounts of light depending on how much oxyhaemoglobin is present. Two different wavelengths of light are used. Red and infrared. The value relates to the SaO2 level and is obtained as a ratio of the AC red over the AC infrared. As the AC value can vary with brightness, we divide the AC value by its DC value to give us the final equation for .

To measure bpm for our project we need to get a signal whose period can be detected reliably even when there is a lot of noise. We used autocorrelation to do this. Autocorrelation processes the signal from the LEDs to highlight any repetitive pattern by producing peaks in the autocorrelation. We used a peak detect routine to measure the bpm from this.

Our project uses a client-server model to communicate between the PSoC and the web server. The graphs are plotted on the website by using a PHP charting library called JPGraph. When the user clicks a button on the account page, we get the data from the PSoC to the PHP page by getting the button to link to a PHP web crawler script which sends a GET request to the PSoC telling it to serve up a new page with specific data according to which button the user has pressed. This is done by using the REST protocol and a variable

We tested the probe on several subjects we discovered that the BPM readings are very reliable and accurate. The standard deviations we measured showed that the readings only really varied along with a person’s heartbeat.

We found that the R value measurements were more spread out compared to the bpm readings and the mean can vary from person to person even though their real SaO2 percentage is probably the same. We think this happened because of how the finger was placed in the probe. This made it difficult to find values of the line fit that would be the same for everyone.

We decided to redesign the probe to fix this. While we were doing this we added a space for the Spark core. The new probe arrived too late to take new measurements but can be seen on the display stand.

By doing this project we learned a lot about web development especially about databases and PHP. We also learned how signal processing and filters can be used in projects. Overall the scope of the project was too big, teamwork was challenging, and we didn’t get to finish every part of the project we started on, but we were pleased about how well the parts that we focused on worked such as the BPM readings, the web pages and the Wi-Fi communication.

# Introduction

Once a patient starts losing oxygen, a doctor has less than three minutes to prevent risk of brain damage, heart failure and death. In sub Saharan Africa up to 70% of operating theatres don't have access to pulse oximeters that could help minimise this by alerting doctors to the low oxygen levels.

Our project can help people in developing countries who can die from hypoxia (very low levels of oxygen, <90%) where hospitals and doctors often cannot afford proper monitoring equipment. A normal oxygen level is >95%.

In our project we plan to achieve the design and construction of a pulse oximeter that is very compact, non-invasive and easy to use. It is important that the Pulse Oximeter can be used in health centres and hospitals in developing regions such as sub-Saharan Africa. These are regions where access to electricity and internet and medical equipment are limited. So the pulse oximeter must be able to work on battery power and not depend on a constant internet connection.

Our Pulse Oximeter system consists of a probe, electronics and a local web server running on a laptop. By using the laptops battery power and a local area network, this makes it suitable for a region where electricity and internet access are limited.

Nellcor commercially design medical grade pulse oximeters for hospitals around the world an example of their present bedside oximeters is Nellcor™ Bedside SpO₂ Patient Monitoring System. It is 82 H x 255 W x 165 D (mm) and weighs 1.6 kg. These are very expensive ranging in the thousands of euro. This makes it unsuitable for remote medical centres to fund.

There are also Pulse-Oximeters for home use for an everyday patient. These can range from £12 to £80 but these just display oxy-levels and bpm. They do not record data and cannot connect to a network. That makes them unsuitable for clinical use.

The CMS-50E OLED Fingertip Pulse Oximeter is an example of a commercial pulse-oximeter which is very similar to our project but does not have wireless capability. Its lack of wireless capability and data collection make it only suitable for home use.

The leading edge in this technology is from The University of British Columbia who have invented their own Pulse-Oximeter which can connect to an iPhone. The phone instantly displays result on the phone accompanied by a graph. As this is the leading edge in research in the area this proves that our project is also cutting edge.

Our project was planned to be implemented on a low cost micro-controller called a Spark Core. As can be seen in the Spark Core is very small and because it also very low cost it makes an ideal platform for our project.



Figure Spark Core

The Spark Core was not available to us during the development through the project. Another very suitable platform although bigger is the PSoC from Cypress Semiconductor and we added a wireless board, making it very suitable for a prototyping platform. The Spark Core arrived after we completed the project but is available to be seen at the exhibition stand.

The signal processing of a Pulse-Oximeter is very dependent on the probe and its quality. To make our reading successful we would have had to buy a medical grade probe which are very expensive (for example: Nellcor OxiMax DS-100A Adult Finger Clip Spo2 Sensor Probe 3ft is £30).

We designed our own probe using 123D Design software and then got it 3D printed. Our website is programmed in HTML and PHP and can be used by healthcare professionals and everyday people. The Pulse Oximeter system includes a wireless lightweight probe which connects to a website and graphs the results.

We made this by using a microcontroller connected to a probe which we programmed to get the pulse and the oxygen levels. We have a website which both doctors and patients can use to track their pulse and blood saturation levels easily.

Our website includes a homepage, overview and login page. We have an accounts database so that both patients and doctors can log on and update t­­­­­heir status.

We have developed a system which graphs and clearly shows oxygenated: deoxygenated haemoglobin ratio.

We have used a custom built designed and 3D printed pulse probe connected to a microcontroller which is programmed to read the pulse and oxygen saturation levels from the LEDs (Light Emitting Diodes) and phototransistor in our probe. We have used the PSoC (Programmable System-On-Chip), mixed signal microprocessor chip from Cypress semiconductor to develop the system. It includes signal conditioning circuits and ARM Cortex-M0 microprocessor. The PSoC is a flexible prototyping board that allowed us to experiment and debug the hardware and software. Using signal processing we can determine the ratio of red to infrared light to calculate the oxygen levels. The ratios change when the SpO2 (peripheral oxygen saturation) level is higher or lower. This works because deoxygenated and oxygenated haemoglobin in the blood absorb light at different wavelengths.

# History of Pulse Oximetry

Before the Pulse-Oximeter the only way to measure SpO2 was by using a painful arterial blood gas which took a minimum of 20-30 minutes to view the result. In 1974 Takuo Aoyagi a Niigata University student disclosed his first Pulse-Oximeter for the use of pilots in World War 2. He set about searching for a non-invasive way of monitoring arterial blood oxygen saturation. Nellcor’s pulse oximeter was put on the market in 1983 and this used his theory developed in 1974.

By 1987 Pulse Oximetry became part of a standard procedure in administrating anaesthetic. The use of oximetry quickly spread to other hospital units, but unfortunately the use of pulse oximetry in developing nations is uncommon.

The WHO has started the pulse oximetry patient safety project in order to improve the use of pulse oximetry in developing countries. Their surgical safety checklist which pulse oximetry is an important part of has been shown to reduce complications and death by over 30% (Source WHO: www.who.int).

Even though Pulse-Oximetry came so far problems still occur such as noise rejection (movement) and accurate calibration. These are challenges were encountered in our project also.

# Principles of Pulse Oximetry

Beers law explains how light is absorbed my any physical object.

The equation of Beers Law: A=log10 ().

Where I0 is the intensity of the incident light and I is the transmitted light. This varies according to things such as the distance the light has to travel and physical properties of the object. In Pulse-Oximetry properties would depend on the finger and the amount of blood in the finger.

The Pulse Oximeter system consists of the probe that senses the pulse, together with the circuitry required to analyse the measurements. A core part of the pulse oximeter is a measure of how much of the haemoglobin in blood is carrying oxygen (oxygen saturation). As you inhale oxygen enters the lungs and then it gets passed into the blood which is full of haemoglobin. Haemoglobin takes the oxygen to the organs of the human body. Haemoglobin with oxygen is called oxygenated haemoglobin. Haemoglobin without oxygen is called deoxygenated haemoglobin.

One molecule of haemoglobin can carry up to four molecules of oxygen – this is described as oxygen saturation.

According to Texas Instruments: “*In a pulse oximeter, the calculation of the level of oxygenation of blood (SaO2) is based on measuring the intensity of light that has been attenuated by body tissue*.”

The tissue inside a human’s body absorbs different amounts of light depending on how much oxyhaemoglobin is passing through it.

The SpO2 ratio is the ratio of oxygenated haemoglobin to total haemoglobin:

SpO2 =

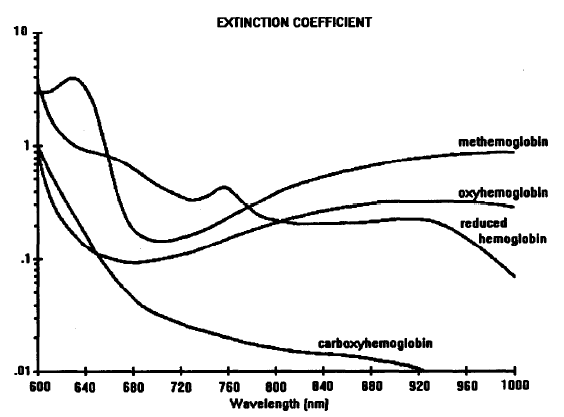


Figure Absorption curves for haemoglobin (adapted from Rusch et.al[[1]](#endnote-1))

All light is made of waves. The distance between the "tips" of the waves is equal to the wavelength. The unit of measurement is nanometre.

Two different wavelengths of light are used where λ1 and λ2 are the different wavelengths of the light. The red LED has a wavelength of 650 nm and the Infrared LED has a longer wavelength of 950 nm. By using two different wavelengths, the absorption in arterial blood will vary according to how much oxygen is present. Figure 2 shows this with the different absorption curves for haemoglobin. We can see that oxyhaemoglobin absorbs wavelengths of light differently compared to the curve for reduced haemoglobin, for example.

If a finger is placed between the LEDs and the Photodiode, the light will now have to pass through the finger to reach the detector as shown in Figure 3.

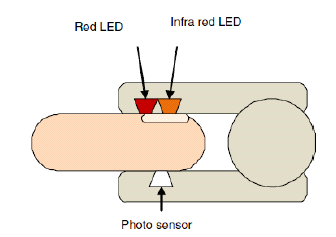


Figure A finger in a probe

Oxyhaemoglobin absorbs more infrared light than red light & deoxyhaemoglobin absorbs more red light than infrared light. This simple principle will help us in our project. In the Pulse Oximeter there are two LEDs: red & Infrared.

Light emitting diodes are ideal for pulse oximeters as they: Are cheap (so they can be used even in disposable probes), are very compact (can fit into very small probes), emit light in accurate wavelengths, do not heat up much during use.

Using the principle above (Oxyhaemoglobin absorbs more infrared light than red light & deoxyhaemoglobin absorbs more red light than infrared light.) we will get the ratio between oxyhaemoglobin and deoxyhaemoglobin. Using this ratio, the pulse oximeter can then work out the oxygen saturation.

We work out the ratio of the AC value as shown in the equation below.

The equation we used to calculate the oxygen saturation:

The value is obtained as a ratio of the AC red over the AC infrared. As the AC value can vary with brightness, we divide the AC value by its DC value to give us the final equation for .

Rearranging the equation makes it clear to see how the value is corrected for the brightness of the LEDs.

Figure 4 shows how the values have a linear relationship to the SaO2. Accurate line fitting was carried out by experiment and is covered in the experimental methods section.

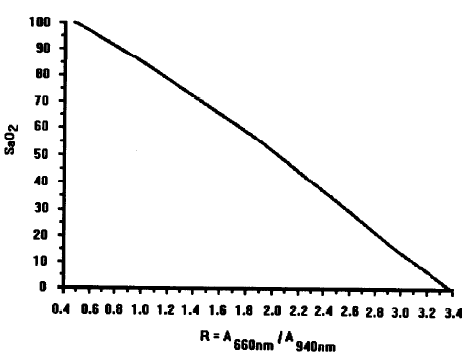


Figure SaO2 vs R

## Medical Use

A pulse oximeter measures oxygen saturation and blood flow. This works by using a technique called photoplethysmography. Figure 5 shows the wave from of the blood flow and provides useful information regarding a heart condition. This is useful in detecting an irregular heartbeat also known as arrhythmia.

Figure 6 shows how a photoplethysmograph (PPG) like the one in our project is used to detect premature ventricular contraction (PVC). This can be compared to the blood pressure BP and electro cardiogram EKG techniques.

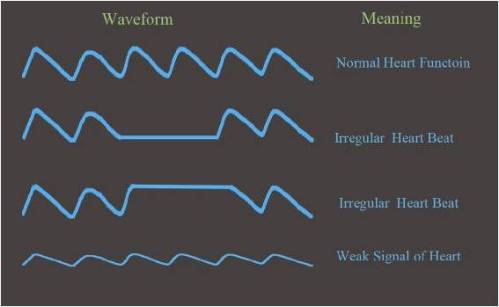


Figure 5 Different Waveforms

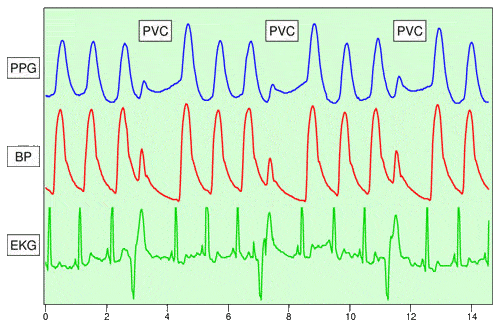


Figure 6 Photoplethysmography (PPG) can be used to detect Premature Ventricular Contraction (PVC)

# Technology



Figure everything that’s required to make the system work

All of the components needed to use the project are shown in . The laptop is used to host the web platform locally on an XAMPP web server running PHP and a MySQL database. This allows communication between the user and the PSoC, allowing commands to be sent to the PSoC by the user and to display graphs of data collected by the PSoC. The laptop is battery powered allowing it to be used temporarily without power. The PSoC also can be powered by the laptop through USB. The LAN router allows wireless communication between the PSoC Wi-Fi shield and the web server without internet access. This works because data only passes through the local network from the PSoC to the laptop and back. The probe uses a phototransistor and 2 LEDs of different wavelengths connected to the PSoC to get the beats per minute and the oxygen saturation.

## Web Platform

Our project uses a client-server model to communicate between the PSoC and the web server. According to Wikipedia “*The* *client-server relationship* *describes the relation between the client and how it makes a service request from the server, and how the server can accept these requests, process them, and return the requested information to the client.*”

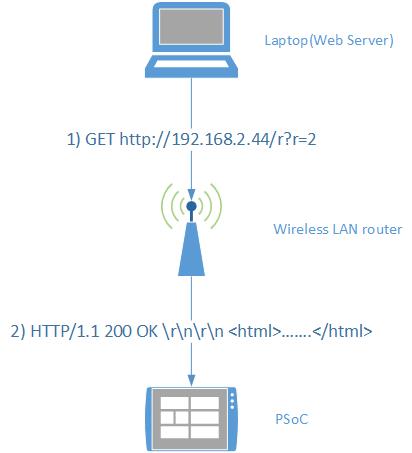


Figure 8 Client-Server model

The server and the client depend on the situation. E.g. when the web browser is connecting to the website the web browser is the client and the machine hosting the website is the server. But when the website is taking the data from the PSoC the PSoC acts as the server and waits for a request from the client and sends back the data to be plotted on a graph by website, the client in this context is the web server.

The finished website runs on an XAMPP web server stack, consisting of an Apache HTTP server, a MySQL database, and support for the PHP and Perl programming languages. The static pages of the website were made in HTML E.g. The home page, overview page. The dynamic pages were done in PHP E.g. Plotting the graphs on the account page.

While we were developing the static pages of the website we hosted it on amazon web services (AWS) EC2 platform. We used a LAMP stack (Linux Apache MySQL PHP) running on a virtual amazon computer to host our website. This was useful because we didn’t have to host on our local computers and it was accessible at any time. The server was controlled by an SSH client (we used PUTTY), and files could then be uploaded using an FTP client (we used Win SCP).

We used the Bootstrap front-end framework to make the visual layout of the website. It gave us basic templates to work with instead of designing it from scratch. It also provides fonts, buttons, widgets etc. that we used. A login system was implemented in PHP using a MySQL database to hold a table of users. The passwords were encrypted using an md5 hash so they weren’t stored in plain text. The login system works by using PHP session cookies. Each time the user logs in the cookie would be set to say the user had logged in and would allow them to access any of the restricted pages. When the user logs out the cookies are unset.

There are 3 main pages on our website: the home page, the overview page and the account page. On the home page there is a brief explanation and introduction to the website. On the Overview page there is a full description of what the website can be used for. On the account page it states your rank i.e. doctor/patient and it also displays a graph of your pulse and blood saturation levels.

Table The users table used in the login system

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | EMAIL | PASSWORD | GROUP |
| 1 | Adrian | [test@example.com](mailto:test@example.com) | 0cbc6611f5540bd0809a388dc95a615b | Doctor |
| 2 | Robert | [test2@example.com](mailto:test2@example.com) | c454552d52d55d3ef56408742887362b | User |

We tried to find a way to store the health data on our website and we found 2 methods of doing this: using the EAV (Entity Attribute Value) table, and the conventional database table. We then compared both of these methods.

Table A conventional table

|  |  |  |  |
| --- | --- | --- | --- |
| ID | NAME | DATE OF BIRTH | MEDICATION |
| 1 | Adrian Kelly | 8-10-1998 | None |
| 2 | Robert Corby | 29-01-1999 | Nurofen |

Table An Entity Attribute Value table

|  |  |  |
| --- | --- | --- |
| ID | Attribute | Value |
| 1 | Name | Adrian Kelly |
| 1 | DOB | 8-10-1998 |
| 1 | Medication | None |
| 2 | Name | Robert Corby |
| 2 | DOB | 29-01-1999 |
| 2 | Medication | None |

The EAV model (Table 3) was better for our project because there are no limits to the number of attributes per entity. Also in health databases there could be several thousand patients and many might not need every attribute, on a conventional table this would lead to NULL fields, but on the EAV table it wouldn’t need to reserve space leading to the table taking up less storage space.

The graphs are plotted on the account page of the website by using a PHP charting library called JPGraph. When the user clicks a button on the account page, we get the data from the PSoC to the PHP page by getting the button to link to a PHP web crawler script which sends a GET request to the PSoC telling it to serve up a new page with specific data according to which button the user has pressed. This is done by using the REST protocol and a variable r. R changes according to which button was pressed. Depending on which value of r the PSoC detects it sends out a different set of data. The web crawler script then uses the simpleHTMLdom library to take the tags out of the PSoCs page and saves them into a file. We then load in these files and used JpGraph to plot them on a line plot. The complete table of valid GET request parameters is shown in Table 4

Table Valid GET parameters

|  |  |
| --- | --- |
| Valid r value format | Response |
| GET 192.168.2.44/r?r=0 | Bpm, ratio, period |
| GET 192.168.2.44/r?r=1 | Red data |
| GET 192.168.2.44/r?r=2 | Autocorrelation data |
| GET 192.168.2.44/r?r=3 | Red data and autocorrelation data |
| GET 192.168.2.44/r?r=4 | Interleaved buffering of red and infrared channels preparing for 5 |
| GET 192.168.2.44/r?r=5 | Red and infrared data |
| GET 192.168.2.44/r?r=8 | Collect experimental data |

## Microcontroller platform

Our project was planned to be implemented on a low cost micro-controller called the Spark Core. The Spark Core was not available to us during the development through the project. We considered using 2 other microcontrollers: the Arduino Yún and the PSoC4 Pioneer kit. At first we tried the Arduino Yún as it has far better community libraries and support and a wider user base. It also has built in Wi-Fi support and an Atheros AR9331 processor running a Linux distribution called Linino. This made the Yún seem like the better option but when we tried to get pulse readings with it we found that the ADC wasn’t fast enough at only 500 Sps (samples per second) and only had 10 bits of resolution. A better alternative was the PSoC from Cypress semiconductor as its ADC ran at 100kSps (kilosamples per second) and had a resolution of 12 bits. The problems with the PSoC are that it is a commercial closed source platform and doesn’t have as much documentation or library support. We also have to program the PSoC in pure C, whereas in the Yún we could program it in a C based programming language called Sketch. The PSoC doesn’t have Wi-Fi support by default so we had to connect it to an Arduino Wi-Fi shield. We could do this because the PSoC4 pioneer kit supports Arduino shields, however the driver libraries still had to be ported to the PSoC. One of the main reasons we used the PSoC as well as for the better ADC was that it supported reconfigurable analogue circuits such as opamps and ADCs that are required to process the tiny currents coming from the phototransistor (light sensor).

### C:\Users\Adrian\Desktop\2013-12-29 16_47_48-www.cypress.com__docID=47035.pngPSoC

Figure The PSoC4 Pioneer kit

The PSoC pioneer kit is shown in Figure 9.

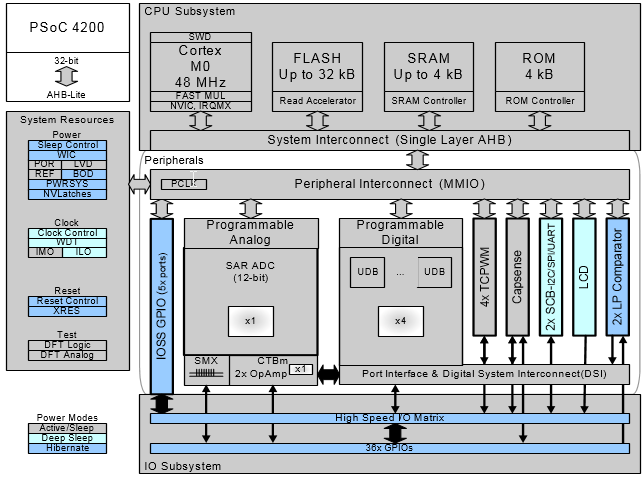


Figure Block diagram of the PSoC

A block diagram of the PSoC 4 chip is shown in . As shown in the diagram the SRAM is only up to 4 kB, this means that our buffer sizes for holding the data have to be kept small and that our program has to be optimised for size as it uses ~95% of the available SRAM. The diagram also shows an ARM Cortex M0 processor running at 48MHz and a programmable analogue and a programmable digital. We use the programmable analogue to make an opamp to boost the signal coming into the program.

## Pulse Oximeter



Figure A block diagram of our program. A full size version can be seen in the appendix

### ADC and amplifiers

The infrared and red LEDS in the probe are pulsed on alternately every 2.5ms so they both pulse on every 5ms. The LEDs shine through the finger on to the photo sensor. The signal from the photo sensor is very small and needs to be amplified by the amplifier. The ADC samples the signal from the amplifier at a rate of 200 samples per second (SPS). This was carefully timed and adjusted to be as close to 200.0Hz as possible so that the 100Hz interference from mains lighting would be rejected accurately by the band pass filters which have a notch at 100Hz. The ADC output red data first then infrared and so on. The red and infra data are separated out in to two data streams to be processed separately. The data from the ADC is very noisy especially when there is a lot of artificial light. Figure 12 shows the plethysmograph reading with no filtering.

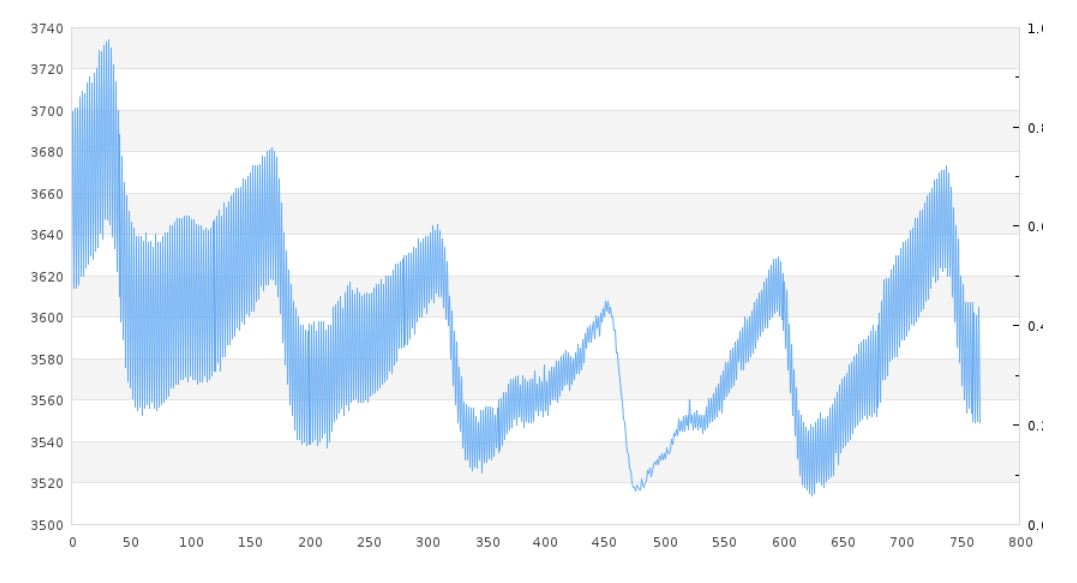


Figure A plethysmograph reading without filtering

### Filters

The band pass filters only pass signals with frequencies between 0.5Hz and 5Hz as shown in . This covers heart rates of 30bpm to 300bpm and allows enough frequencies through so the plethysmograph is not disturbed but the DC and high frequency noise is still filtered out.

The low pass filters remove frequencies above 0.5Hz to make the DC value of the signals without interference from the AC. The AC and DC values are used in the calculation of the ratio R which represents SaO2 saturation.

The filters are made by adding the weighted input values and output values including past values. The low pass filter heavily weights the last output value, it puts a low weight on new input data. This is how it forms a low pass filter.

dc\_n = (1 - 0.984) \* (float)u\_n + 0.984 \* dc\_nm1;

The band pass filter is more complicated but works on the same principles.

v\_n = 0.07295\*(float)u\_n - 0.07295\*(float)u\_nm2 + 1.853\*v\_nm1 - 0.8541\*v\_nm2;

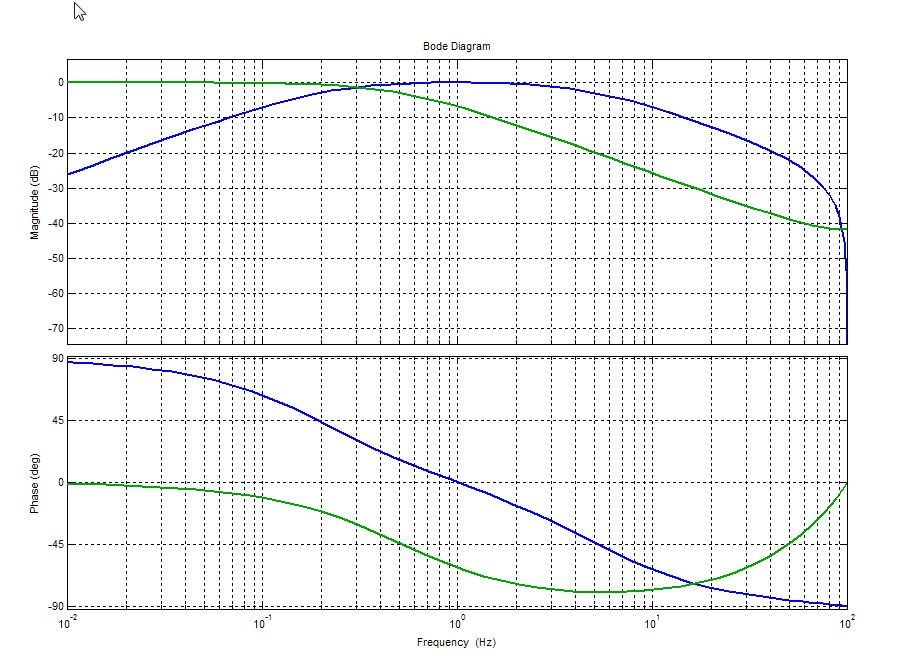


Figure Bode diagram of the filters (green: low pass, blue: band pass)

### Autocorrelation

To measure bpm for our project we need to get a signal whose period can be detected reliably even when there is a lot of noise. We filtered the signal and the zero crossings to get the period of each beat. shows a typical plethysmograph notice how noisy it is which makes it difficult to detect the period reliably.

To get a more reliable period detection we needed to change the original signal into another one that highlights the periodicity of the original signal. So if it is indeed periodic, then that will stand out in the new signal and then we can measure that in the usual way using peak detect or zero crossing detect.

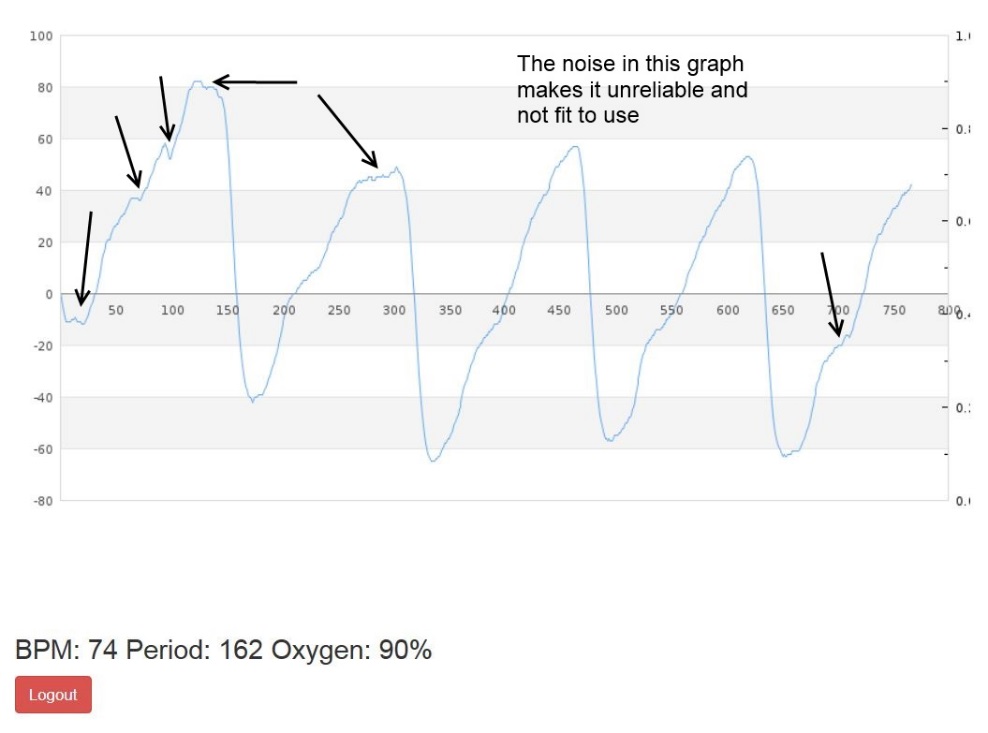
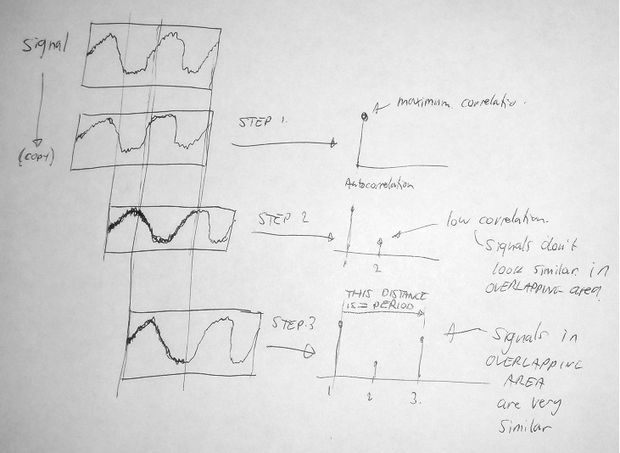


Figure Raw plethysmograph data

 Figure 15 Diagram explaining autocorrelation

To understand autocorrelation imagine your signal is contained in a window or buffer. Now image you have an exact copy of that window or buffer with a time delay.

What Autocorrelation does is to measure the correlation (or similarity) between the signal and its delayed copy each time the copy is delayed by a sample period.

Referring to Figure 15. When the signal and the copy have no delay they are very similar (i.e. highly correlated) as shown in step 1, and therefore the autocorrelation value for delay = 0 is maximum.

Step 2 shows that when the copy is delayed significantly it doesn't look similar to the original in the overlapping area. Therefore the autocorrelation value for this delay is small.

Step 3 shows that when the copy is delayed even more the signal in the overlapping area is very similar to the original because the signal is periodic. Therefore the autocorrelation value for this delay shows a peak.

We can see that the distance in time between the maximum peak at the beginning and the first peak afterwards must be equal to the fundamental period of the waveform.

Technically the "similarity" or correlation between the signal and its delayed copy is the **sum of the product** of the two signals.

The core of the Autocorrelation Code is very short:

**for** **(**i **=** 0**;** i **<** len**;** i**++)**

**{**

sum **=** 0**;**

**for** **(**k **=** 0**;** k **<** len **-** i**;** k**++)** sum **+=** **(**rawData**[**k**]** **-** 128**)\*(**rawData**[**k **+** i**]** **-** 128**)** **/** 256**;**

**}**

The data is in the rawData[] array. We subtract 128 from each value because it’s 8bit unsigned and we require signed values.

The sum value is the result of each autocorrelation calculation, i.e. each point of the function. In order to save memory we don't save the output to an array. We're going to work on the individual sum values to find the first peak and therefore calculate the period.



Figure Correlated graph of bpm

shows an autocorrelation in red and a filtered autocorrelation function in blue. shows data in detail. Notice that in the raw autocorrelation values has a false peak at -465. On the Filtered autocorrelation side you can see that it has been filtered out.

|  |  |
| --- | --- |
| Filtered ACorr Values: [80 to 95] | Raw ACorr Values: [80 to 95] |
| -439 | -452 |
| -445 | -455 |
| -452 | -462 |
| -457 | -467 |
| -461 | -470 |
| -464 <=== False peak removed by filter | -465 <=== Value causes false peak detect |
| -465 | -467 |
| -464 | -460 |
| -462 | -460 |
| -461 | -463 |
| -459 | -459 |
| -457 | -455 |
| -452 | -443 |
| -447 | -442 |
| -443 | -442 |
| -437 | -428 |

Table Comparing correlated and uncorrelated data

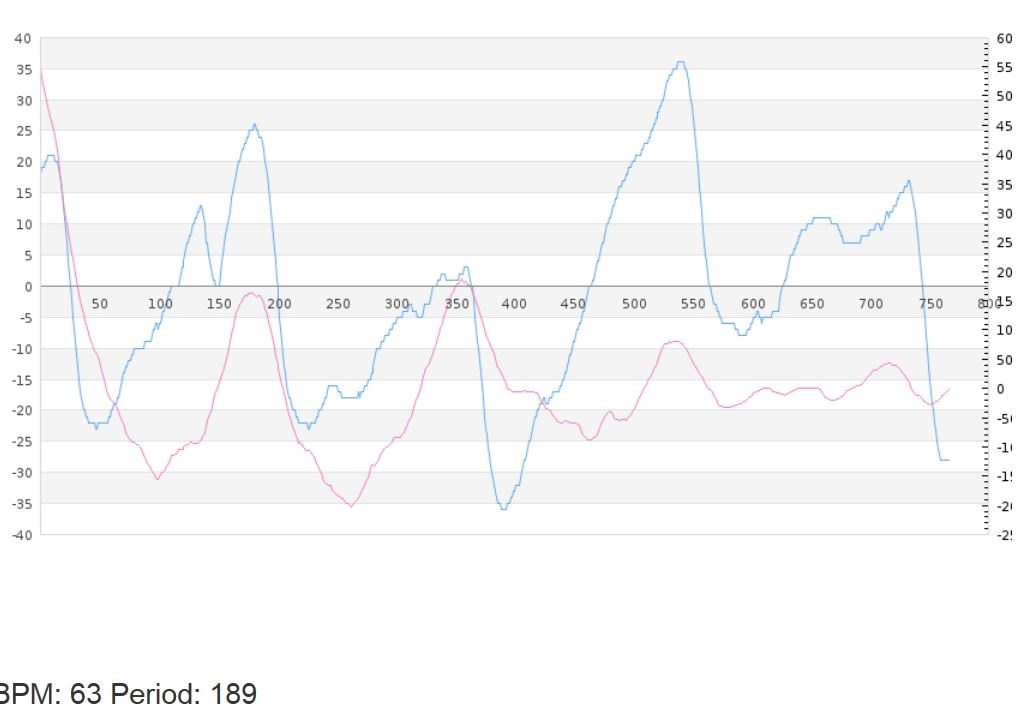


Figure Comparing acorr vs. acorr filtered

In the graph shows the plethysmograph waveform in blue and its corresponding autocorrelation function in red. Even though the plethysmograph is very noisy, it can be seen that the period corresponds to the peak and the BPM is correct.

## Probe

We used 123D Design to design our probe. We first decided the dimensions of the probe. We made a 70mm space for the finger and an extra compartment for a battery and a Spark Core.

A Spark Core is a tiny (2cmx3.3cm) wireless programmable board. It has the simplicity of an Arduino and wirelessly programmable.

We made three holes on the probe. There are two holes on the top for the infrared LED and the red LED. There is one hole on the bottom for the photodiode. We first started with a 120mm x 40mm rectangle. We extruded this by 60mm. On the long side we made a cut through 60mm and then made an elliptical curve and cut it through. This would be the hole from which the person would insert their finger. We then cut the hole for the battery and Spark Core. We made this hole at the back end and made an in cut so that the battery and Spark Core could sit in properly. We then filleted all the edges. This made all the edges curved.

We got this 3D printed in Polyamide (A strong and flexible material with a high level of detail).

After receiving the probe we tested it using the PSoC. Shortly after this the Spark Core arrived. We could make further improvements on the probe to accommodate the Spark Core and tidy up wiring.

A new design was made for the probe in which there were two slots for adjustable finger thickness and a compartment in which the Spark Core could sit.

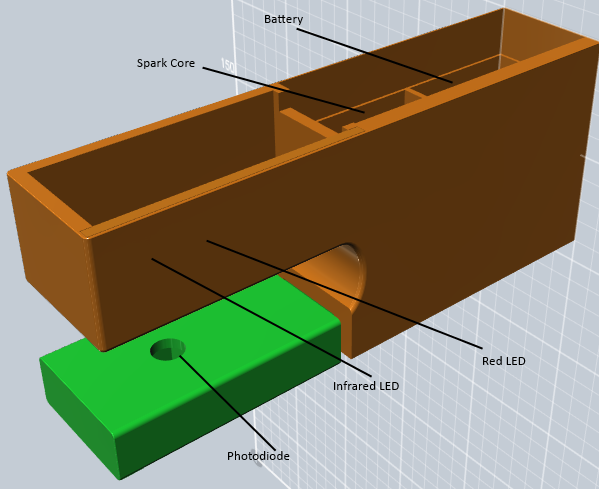


Figure Initial 123D Design of Probe

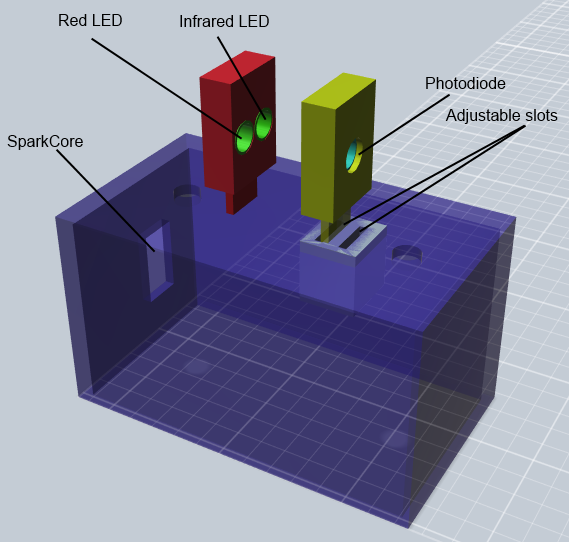


Figure Improved 123D Design of Probe

# Results and Conclusions

We tested our Pulse-Oximeter for accuracy on several subjects. We Tested bpm, redac, reddc, irfrared ac, infrared dc, R% and S%. We tested about 60 times on each subject. Subject A was tested in daylight and in the evening under artificial lighting. Subject B was tested in daylight.

## Results

Subject A (Daylight): Over forty results were taken. The average of the BPM results was 86.87. The standard deviation was 5.13 as shown in . The results showed the readings were measured successfully as we found consistent results because the standard deviation was small compared to the mean. There are also outliers present shown at point 70. This would be caused by the change in heart rate of a person within the five minutes of testing. There were also other outliers at the values such as 187, 187, 193, and 44. This could have been caused by a person moving their finger in the probe.

Figure 20 Histogram showing Subject A bpm (daylight)

Figure Histogram showing Subject A R value (daylight)

shows that the R data is more spread out compared to the BPM readings. The mean was 1.255 and the standard deviation was 0.174. There is a clear peak in the frequency of the data at 1.2 but there are several higher readings also. This shows the readings are correctly measured but are sensitive to external factors causing higher values sometimes.

Subject B (Daylight): For subject B we also obtained BPM results. The results were very consistent and had a low standard deviation of 5.47. The mean of the data was 64.88. From these results we only had one outlier which was 96. This was due to a change in heart rate of the subject over the period of time testing. In we see that the results were consistent around the values 60 and 70.

Figure 22 Histogram showing Subject B (daylight)

The R values are shown in . The mean was 0.187 and the standard deviation was 0.041. The data has a higher frequency around 0.2 and is somewhat spread out as shown by the standard deviation. The mean is much lower than the mean for Subject A. This is because the redac value is much lower. This was to do with how Subject B put their finger into the probe.

Figure Histogram showing R values of Subject B (daylight)

Subject A (Artificial Light): The BPM results under artificial light were the same consistency as the daylight. This shows that interference from the artificial lights has been filtered out. We had a low standard deviation in this test also at 4.23. The mean was 87.95. In this test there was only one outlier as well, this was 181. This was due to different positioning of the finger from previous readings. From we see that was high consistency around the value of 90. We have just one value of 102 which has affected the histogram.

The R value results had a mean of 0.924 and a standard deviation of 0.296. The data showed higher frequencies around 0.75 but was more spread out compared to the bpm. This can be seen in .

Figure 24 Histogram showing Subject A bpm (artificial light)

Figure Histogram showing R value of Subject A (artificial light)

## Conclusions

From the tests we made on several subjects we discovered that the BPM readings are very reliable and accurate. The standard deviations we measured showed that the readings only really varied along with a person’s heartbeat. There we very few outliers in our measurements.

The R value is the ratio of the redac signal to the infrared ac reading. This is related to the SaO2 oxygen value in the blood. We found that the R value measurements were more spread out compared to the bpm readings and the mean can vary from person to person even though their real SaO2 percentage is probably the same. We think this happened because of how the finger was placed in the probe. This made it difficult to find values of the line fit that would be the same for everyone.

We decided to redesign the probe to fix this. While we were doing this we added a space for the Spark core. The new probe arrived too late to take new measurements but can be seen on the display stand.

By doing this project we learned a lot about web development especially about databases and PHP. We also found how signal processing and filters can be used in projects. Overall the scope of the project was too big, teamwork was challenging, and we didn’t get to finish every part of the project we started on, but we were pleased about how well the parts that we focused on worked such as the BPM readings, the web pages and the Wi-Fi communication.

# Project Management

Documentation involved a project diary, project schedule, poster and a documented report.

We developed a project management plan in order to schedule the tasks to be completed. We used Atlassian JIRA as an agile project management tool. We listed the tasks to be completed. There were three epics, 1) Pulse Oximeter 2) Website 3) Documentation. The epics were broken down into stories which were further broken down into tasks in keeping with the AGILE project management technique. The tasks were assigned to each member of the group and when the tasks were completed, they logged their hours of work and uploaded to the Bitbucket repository. The tasks were marked as completed and this was a good way of sharing documentation and code files.

We completed one week sprints in which we added stories from the backlog to the sprint list and completed these tasks over the course of the week. We had 15 minute sprint meetings to understand what needed to be done, how long it would take and what were the challenges to completing the tasks.

Using Atlassian JIRA helped us to plan the tasks for the project and to keep to the time schedule set out at the start of the project. We used Bit bucket to manage the code repository so we could upload the code and update the project status at any time.

# Challenges

The spark core was an important factor in the low cost and small size of our project. As the spark core was a Kickstarter project delays were expected, so we decided to prototype the project on PSoC. We initially attempted to add Wi-Fi to the PSoC by using the cc3000 Wi-Fi module from Texas Instruments. This is the same module that is used on the spark core so it is small and is easy to configure, but because of limited documentation and support we found that porting the cc3000 drivers to PSoC was too big of a task. Instead we used an Arduino Wi-Fi shield with open source libraries provided by other PSoC users.

Measuring the BPM reliably was a big challenge because of noisy a plethysmograph and the environment e.g. artificial lights. Looking at plethysmograph data, is not always clear that it is periodic. But by using the autocorrelation function and the peak detect state machine we have found that it is very good at reliably getting the BPM.

This was a very big project. Managing the scope of the project so we didn’t take on too much was a big challenge. We spent many hours developing an android app which we later discarded because we didn’t have enough time to finish and debug it. This was the same with the cc3000 described above. We think that we were right to focus on the core part of the project which was measuring BPM and SPo2. The website is at a demo stage but is still not fully functional.

# Acknowledgements

We would like to thank mentors of CoderDoJo Limerick for their help.

Salesian Secondary College for supporting our project.

Science teachers – Ms.Sheehy & Mr. O’Donovan for their support.

# Appendix: PSOC Code

/\* Copyright 2013 Adrian Kelly

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

#include <device.h>

#define LENGTH 768

// AC scaling factor

#define ACSCALE 256

#include <WiFiClient.h>

#include <WiFi.h>

#include <server\_drv.h>

#include <html.h>

#define SOCKET 0

#define HTTP\_PORT 80

#define NULL\_CHAR '\0'

#define NEWLINE '\n'

#define LINEFEED '\r'

#define GET\_BUFFER\_SIZE 50

#define RESTART\_VAL 50

void WifiInit**();**

void WifiServe**(**uint8**);**

// Keeps the server up and running

uint8 ServerAv**(**void**);**

// Used to parse the request

// returns 1 on a valid get request

uint8 ParseData**(**char **\***getData**);**

// Helper function for parsing the request

// Returns 0-99 or 255 if the request is invalid

uint8 ReadUint8Val**(**char **\*\***getData**,** char delim**,** char field**,** uint8 def**);**

// Helper function, takes value and writes ascii to the spi interface

// takes 0-99 only.

void WriteUint8Val**(**uint8 val**);**

void WriteInt16Val**(**int16 val**);**

/\* GLOBALS \*/

int16 sample\_period **=** 5**;** // milliseconds

int16 buffer**[**LENGTH**];**

char string**[**16**];**

uint8\_t ind**;**

int16 data\_select**;**

int16 data\_select\_old**;**

float irac**;**

float redac**;**

float irdc**;**

float reddc**;**

float ratio**;**

float R**;**

uint8 bpm **=** 0**;**

int period **=** 0**;**

void getOxygen**()**

**{**

float Rred **=** redac **/** reddc**;**

float Rir **=** irac **/** irdc**;**

R **=** 100**\*(**Rred **/** Rir**);**

ratio **=** 2**\***R**/**100 **+** 95**;**

**if** **(**ratio**>**100**)** ratio **=** 100**;**

**}**

int autocorr**()**

**{**

int length **=** LENGTH**;**

int i**,**k**;**

long sum\_av**,** sum\_av\_d1**,** sum\_d1**,** sum\_d2**,** sum\_d3**;**

long sumRed**;**

int pd\_state **=**0**;**

int thresh **=** 0**;**

static int bpm**;**

sum\_av **=** 0**;**

sum\_av\_d1 **=** 0**;**

sumRed **=** 0**;**

sum\_d1 **=** 0**;**

sum\_d2 **=** 0**;**

period **=** 0**;**

pd\_state **=**0**;**

**for(**i**=**0**;** i **<** length**;** i**++)**

**{**

sum\_d3 **=** sum\_d2**;**

sum\_d2 **=** sum\_d1**;**

sum\_d1 **=** sumRed**;**

sumRed **=** 0**;**

**for(**k**=**0**;** k **<** **(**length**-**i**);** k**++)**

**{**

// Only does overlapping area

sumRed **+=** buffer**[**k**]\***buffer**[**k**+**i**]/**ACSCALE**;**

**}**

// calculate AC value

**if(**i**==**0**)**

redac **=** **(**float**)**sumRed**;**

// Peak Detect State Machine

sum\_av\_d1 **=** sum\_av**;**

// Average 4 ACORR values to remove noise for more reliable peak detect

sum\_av **=** sumRed**/**4 **+** sum\_d1**/**4 **+** sum\_d2**/**4 **+** sum\_d3**/**4**;**

**if** **(**pd\_state **==** 2 **&&** **(**sum\_av**-**sum\_av\_d1**)** **<=-**4**)**

**{**

period **=** i**;**

pd\_state **=** 3**;**

**}**

**if** **(**i**>**4 **&&** pd\_state **==** 1 **&&** **(**sum\_av **>** thresh**)** **&&** **(**sum\_av**-**sum\_av\_d1**)** **>** 4**)** pd\_state **=** 2**;**

**if** **(**i**==**0 **&&** pd\_state **==** 0**)** **{**

thresh **=** sumRed **\*** 0.1**;**

pd\_state **=** 1**;**

**}**

// Only sent if requested by 'GET /r?r=1' or 'GET /r?r=3'

**if** **(**data\_select **==** 1 **||** data\_select **==** 3**)**

**{**

// Write Data from Red Channel

WiFiClient\_WriteChunk**(**"<p>"**);**

sprintf**(**string**,**"%i"**,**buffer**[**i**]);**

strcat**(**string**,**"</p>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

**}**

// Only sent if requested by 'GET /r?r=2' or 'GET /r?r=3'

**if** **(**data\_select **==** 2 **||** data\_select **==** 3**)**

**{**

// Write ACorr Data

WiFiClient\_WriteChunk**(**"<a>"**);**

sprintf**(**string**,**"%i"**,**sumRed**);**

strcat**(**string**,**"</a>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

**}**

// Only sent if requested by 'GET /r?r=5'

// and 'GET /r?r=4' was sent just before

**if** **(**data\_select\_old **==** 4 **&&** data\_select **==** 5**)**

**{**

// Write Data from Red Channel

WiFiClient\_WriteChunk**(**"<p>"**);**

sprintf**(**string**,**"%i"**,**buffer**[**i**]);**

strcat**(**string**,**"</p>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

i**++;**

// Write Data from InfraRed Channel

WiFiClient\_WriteChunk**(**"<i>"**);**

sprintf**(**string**,**"%i"**,**buffer**[**i**]);**

strcat**(**string**,**"</i>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

**}**

/\*UART\_UartPutChar('C');

UART\_UartPutChar(IRbuffer[i]>>8);

UART\_UartPutChar(IRbuffer[i]&0xff);

UART\_UartPutChar(buffer[i]>>8);

UART\_UartPutChar(buffer[i]&0xff);

UART\_UartPutChar('C');

UART\_UartPutChar(sum>>8);

UART\_UartPutChar(sum&0xff);

UART\_UartPutChar(sumRed>>8);

UART\_UartPutChar(sumRed&0xff); \*/

**}**

// bpm = 60\*sample Frequency/period

**if** **(**period **>** 0**)** bpm **=** 60000**/**period**/**sample\_period**;**

**return** bpm**;**

**}**

int16 Filter**(**int16 input**)**

**{**

// Static automatically sets to 0 and doesnt get reset

static int16 u\_n**;**

static int16 u\_nm1**;**

static int16 u\_nm2**;;**

static float v\_n**;**

static float v\_nm1**;**

static float v\_nm2**;**

static float dc\_n**;**

static float dc\_nm1**;**

u\_nm2 **=** u\_nm1**;**

u\_nm1 **=** u\_n**;**

u\_n **=** input**;**

dc\_nm1 **=** dc\_n**;**

dc\_n **=** **(**1 **-** 0.984**)** **\*** **(**float**)**u\_n **+** 0.984 **\*** dc\_nm1**;**

reddc **=** dc\_n**;**

v\_nm2 **=** v\_nm1**;**

v\_nm1 **=** v\_n**;**

v\_n **=** 0.07295**\*(**float**)**u\_n **-** 0.07295**\*(**float**)**u\_nm2 **+** 1.853**\***v\_nm1 **-** 0.8541**\***v\_nm2**;**

int16 result **=** **(**int16**)**v\_n**;**

**return** result**;**

**}**

int16 IRFilter**(**int16 input**)**

**{**

// Static automatically sets to 0 and doesnt get reset

static int16 u\_n**;**

static int16 u\_nm1**;**

static int16 u\_nm2**;;**

static float v\_n**;**

static float v\_nm1**;**

static float v\_nm2**;**

static float dc\_n**;**

static float dc\_nm1**;**

u\_nm2 **=** u\_nm1**;**

u\_nm1 **=** u\_n**;**

u\_n **=** input**;**

dc\_nm1 **=** dc\_n**;**

dc\_n **=** **(**1 **-** 0.984**)** **\*** u\_n **+** 0.984 **\*** dc\_nm1**;**

irdc **=** dc\_n**;**

v\_nm2 **=** v\_nm1**;**

v\_nm1 **=** v\_n**;**

v\_n **=** 0.07295**\*(**float**)**u\_n **-** 0.07295**\*(**float**)**u\_nm2 **+** 1.853**\***v\_nm1 **-** 0.8541**\***v\_nm2**;**

int16 result **=** **(**int16**)**v\_n**;**

**return** result**;**

**}**

int main**()**

**{**

Opamp\_Start**();**

ADC\_Start**();**

ADC\_StartConvert**();**

WifiInit**();**

int count **=** 0**;**

int led **=** 0**;**

Pin\_Red\_Write**(**0**);**

Pin\_IR\_Write**(**0**);**

**for(;;)**

**{**

irac **=** 0**;**

// A pre-count run-in (count <0) to flush out the filters

**for(**count**=-**20**;** count **<** LENGTH**;** count**++)**

**{**

**for** **(**led**=**0**;**led**<**2**;**led**++)**

**{**

Test\_Write**(**led**);**

ADC\_StartConvert**();**

ADC\_IsEndConversion**(**ADC\_WAIT\_FOR\_RESULT**);**

int16 reading **=** ADC\_GetResult16**(**0**);**

**if(**led **==** 1**)**

**{**

int16 reading\_filtered **=** Filter**(**reading**);**

**if** **(**count **>=**0 **)** buffer**[**count**]=**reading\_filtered**;**

Pin\_Red\_Write**(**0**);**

Pin\_IR\_Write**(**1**);**

**}**

**else** **if(**led **==** 0**)**

**{**

int16 IRreading\_filtered **=** IRFilter**(**reading**);**

// calculate AC value

**if** **(**count **>=**0 **)** irac **+=** IRreading\_filtered**\***IRreading\_filtered**/**ACSCALE**;**

**if** **(**count **>=**0 **&&** data\_select **==** 4**)**

**{**

buffer**[**count**]=**IRreading\_filtered**;**

count**++;**

**}**

Pin\_Red\_Write**(**1**);**

Pin\_IR\_Write**(**0**);**

**}**

/\* Tweak the delay for desired sample rate \*/

// CyDelayUs(10);

**}**

**}**

// Check for a Client Connection

WifiServe**(**bpm**);**

**}**

**}**

void WifiInit**()**

**{**

uint8 localIP**[]** **=** **{**255**,**255**,**255**,**255**};**

// Give the WiFi chip a chance to init

CyDelay**(**1000**);**

WiFi\_Init**();**

// SSID and Password

WiFi\_Begin**(**WPA**,** "HomerWAP"**,** "2Ashfield"**);**

// Give time to connect

**do{**

CyDelay**(**100**);**

// Grab the IP Address, look in the debugger if needed

WiFi\_LocalIPGet**((**uint8\_t**\*)(&**localIP**));**

/\* // This IP is assigned by DHCP on my router, turn on the L9 LED to indicate we're ready

localIP[0] = 192;

localIP[1] = 168;

localIP[2] = 1;

localIP[3] = 206; \*/

**}** **while(**localIP**[**0**]** **==** 0**);**

ServerDrv\_StartServer**(**HTTP\_PORT**,** SOCKET**);**

**}**

void WifiServe**(**uint8 bpm**)**

**{**

char getData**[**GET\_BUFFER\_SIZE**];**

int i **=** 0**;**

**if(**ServerAv**())**

**{**

// used to save the previous charcter to check for \n\n

uint8 lastVal **=** 0**;**

// used to keep our position in the request array

uint8 pos**;**

// Clear out old data

**for(**pos **=** GET\_BUFFER\_SIZE**;** pos**>**0**;)**

**{**

getData**[--**pos**]=**NULL\_CHAR**;**

**}**

**while(**WiFiClient\_Connected**())**

**{**

**if(**WiFiClient\_Available**())**

**{**

uint8 curVal **=** WiFiClient\_ReadByte**();**

// if we get two \n's indicating the client is done with the request

// send our response

**if(**lastVal **==** NEWLINE **&&** curVal**==**NEWLINE**)**

**{**

// Parse the response

ParseData**(**getData**);**

WiFiClient\_WriteChunk**(**"HTTP/1.1 200 OK\r\n"**);**

WiFiClient\_WriteChunk**(**"Connection: Keep-Alive\r\n"**);**

WiFiClient\_WriteChunk**(**"Content-Type: text/html\r\n"**);**

WiFiClient\_WriteChunk**(**"X-Pad: avoid browser bug\r\n"**);**

WiFiClient\_WriteChunk**(**"\r\n"**);**

WiFiClient\_WriteChunk**(**"<html><body>"**);**

// Server Connected so Calculate AutoCorr and BPM

bpm **=** autocorr**();**

// Calculate SaO2% (R)

getOxygen**();**

WiFiClient\_WriteChunk**(**"<b>"**);**

sprintf**(**string**,**"%i"**,**bpm**);**

strcat**(**string**,**"</b>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

WiFiClient\_WriteChunk**(**"<o>"**);**

sprintf**(**string**,**"%i"**,** **(**int**)**ratio**);**

strcat**(**string**,**"</o>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

WiFiClient\_WriteChunk**(**"<z>"**);**

sprintf**(**string**,**"%i"**,** period**);**

strcat**(**string**,**"</z>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

// Collect Experimental Data

**if** **(**data\_select **==** 8**)**

**{**

WiFiClient\_WriteChunk**(**"<r>"**);**

sprintf**(**string**,**"%i"**,** **(**int**)(**R**));**

strcat**(**string**,**"</r>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

WiFiClient\_WriteChunk**(**"<rac>"**);**

sprintf**(**string**,**"%i"**,** **(**int**)**redac**);**

strcat**(**string**,**"</rac>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

WiFiClient\_WriteChunk**(**"<rdc>"**);**

sprintf**(**string**,**"%i"**,** **(**int**)**reddc**);**

strcat**(**string**,**"</rdc>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

WiFiClient\_WriteChunk**(**"<irac>"**);**

sprintf**(**string**,**"%i"**,** **(**int**)**irac**);**

strcat**(**string**,**"</irac>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

WiFiClient\_WriteChunk**(**"<irdc>"**);**

sprintf**(**string**,**"%i"**,** **(**int**)**irdc**);**

strcat**(**string**,**"</irdc>\n"**);**

WiFiClient\_WriteChunk**(**string**);**

**}**

WiFiClient\_WriteChunk**(**"</body></html>\r\n\r\n"**);**

// Clear out our buffer, we're done processing

**for(**pos **=** GET\_BUFFER\_SIZE**;** pos**>**0**;)**

**{**

getData**[--**pos**]=**NULL\_CHAR**;**

**}**

CyDelay**(**10**);**

// Close the connection

// Failure to do so will cause the browser to sit for much longer

WiFiClient\_Stop**();**

**}**

// ignore LINEFEED

**if(**curVal **!=** LINEFEED**)**

**{**

// add the first GET\_BUFFER\_SIZE bytes to our buffer for GET processing

**if(**pos **<** GET\_BUFFER\_SIZE**)**

getData**[**pos**++]=**curVal**;**

lastVal**=**curVal**;**

**}**

**}**

**}**

WiFiClient\_Stop**();**

**}**

**}**

// Starts up the server, checks that it continues to run

uint8 ServerAv**()**

**{**

static uint8 restartCnt **=** 0**;**

**if((**ServerDrv\_GetServerState**(**SOCKET**)** **==** CLOSED**)** **&&** **(**restartCnt**++>**RESTART\_VAL**))**

**{**

restartCnt **=** 0**;**

ServerDrv\_StartServer**(**HTTP\_PORT**,** SOCKET**);**

**}**

WiFiClient\_Init**(**SOCKET**);**

**if(**WiFiClient\_Status**()** **==** ESTABLISHED**)**

**{**

**return** 1**;**

**}**

**return** 0**;**

**}**

// format will be "GET /r?r=# "

// ## can be 0, 1 or 2 digits

uint8 ParseData**(**char**\*** getData**)**

**{**

uint8 vals**[]={**0**,**0**,**0**};**

**if(**strncmp**(**getData**,** "GET /r?"**,**7**))**

**return** 0**;**

getData **+=** 6**;**

vals**[**0**]** **=** ReadUint8Val**(&**getData**,** '?'**,** 'r'**,** data\_select**);**

**if(**getData**[**0**]==**' '**){**

// End of GET request

data\_select\_old **=** data\_select**;**

data\_select **=** vals**[**0**];**

**return** 1**;**

**}**

**return** 0**;**

**}**

// Helper function, make sure the string is <delim><field>=<0, 1, or 2 ints>

// Has the side effect of incrementing pos

// returns 255 on failure

uint8 ReadUint8Val**(**char**\*\*** getData**,** char delim**,** char field**,** uint8 def**)**

**{**

// Expect at most two ASCII chars between 0-9.

uint8 ret **=** def**;**

**if((\***getData**)[**0**]==**delim **&&** **(\***getData**)[**1**]==**field **&&** **(\***getData**)[**2**]** **==** '='**){**

**(\***getData**)+=**3**;**

**if((\***getData**)[**0**]** **>=** '0' **&&** **(\***getData**)[**0**]** **<=**'9'**)**

**{**

ret **=** 0**;**

ret **+=** **(\*((\***getData**)++)** **-** '0'**);**

**if((\***getData**)[**0**]** **>=** '0' **&&** **(\***getData**)[**0**]** **<=**'9'**)**

**{**

ret **\*=** 10**;**

ret **+=** **(\*((\***getData**)++)** **-** '0'**);**

**}**

**}**

**}else{**

// Error, undefined format of get request

**return** 0xff**;**

**}**

**return** ret**;**

**}**

/\* [] END OF FILE \*/

# Appendices:

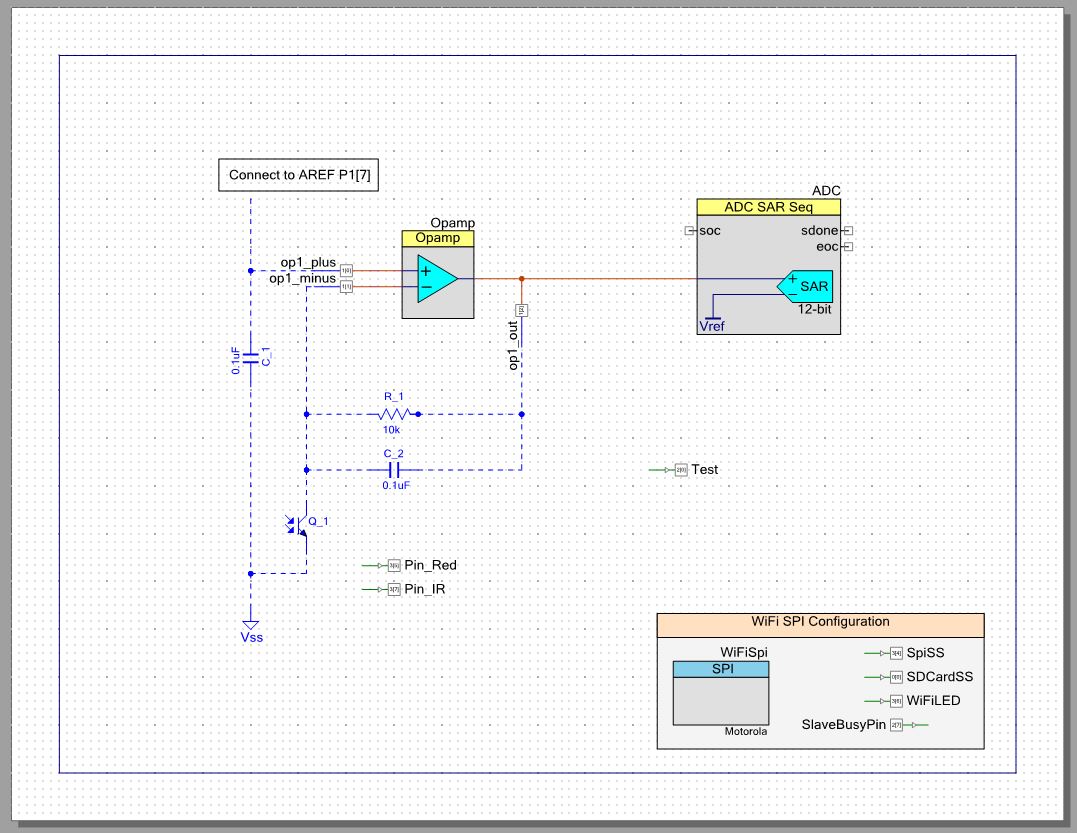


Figure PSOC system Schematic



Figure Homepage of website

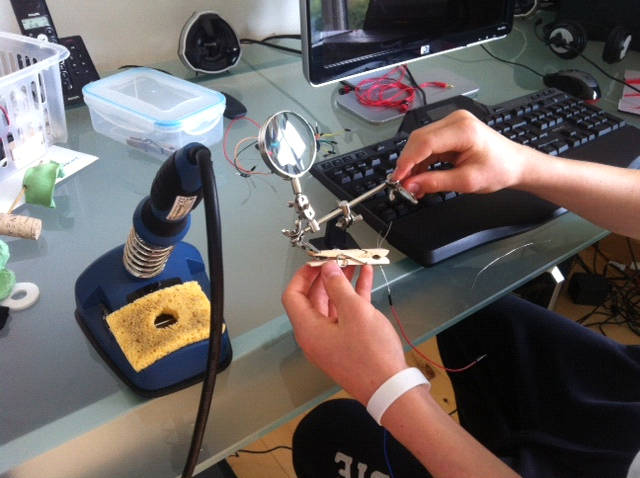


Figure 28 First Prototype



Figure 29 PSoC Pulse-Ox

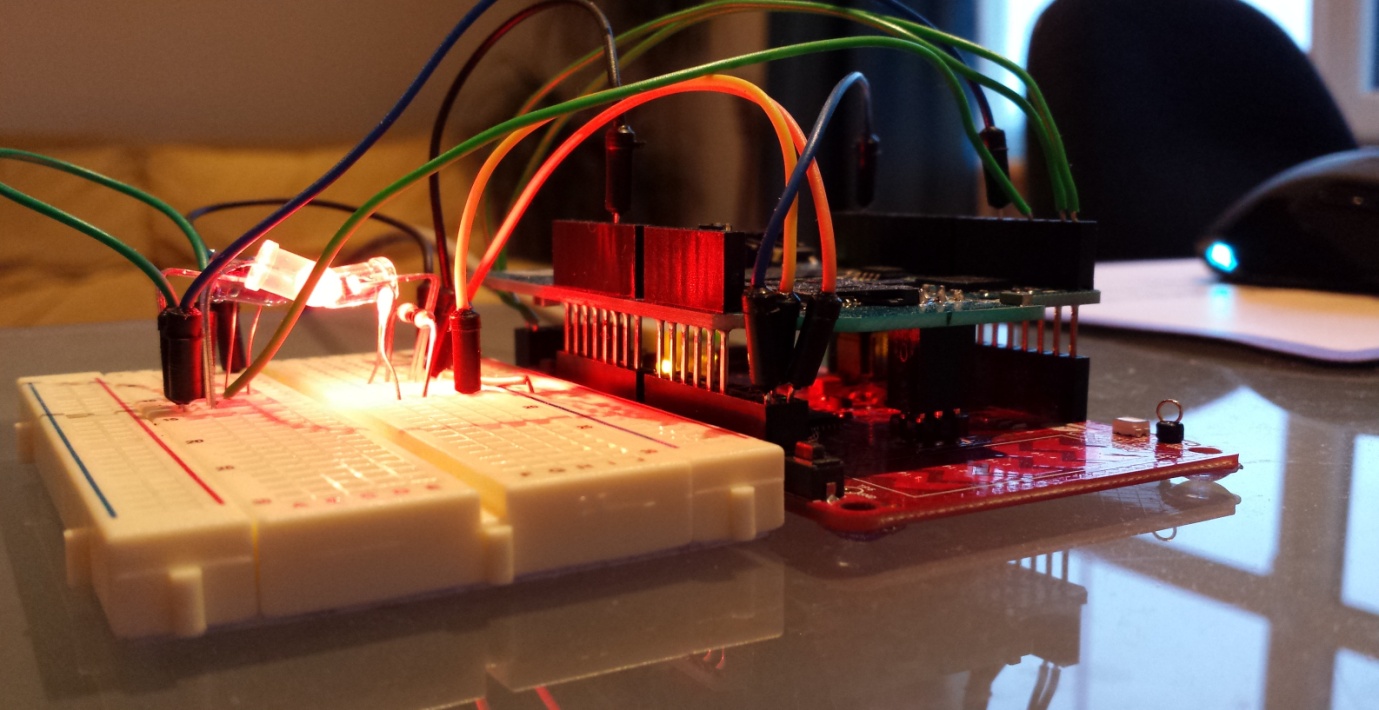
****

Figure 30 PSoC Pulse Oximeter

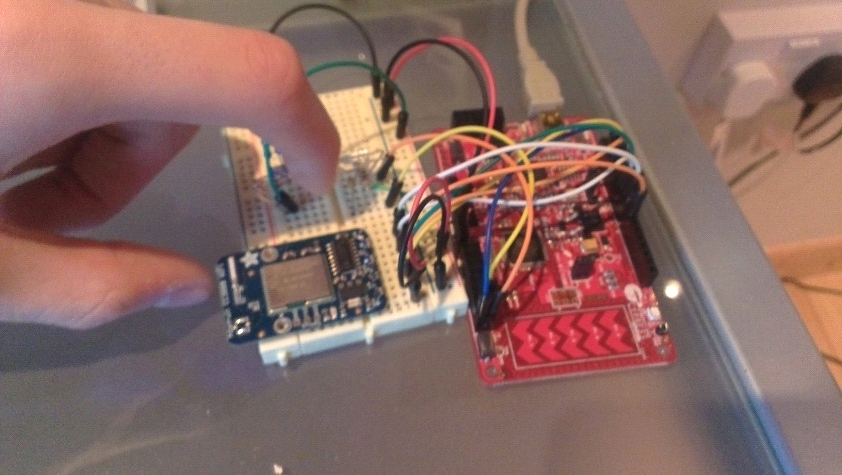


Figure 31 PSoC

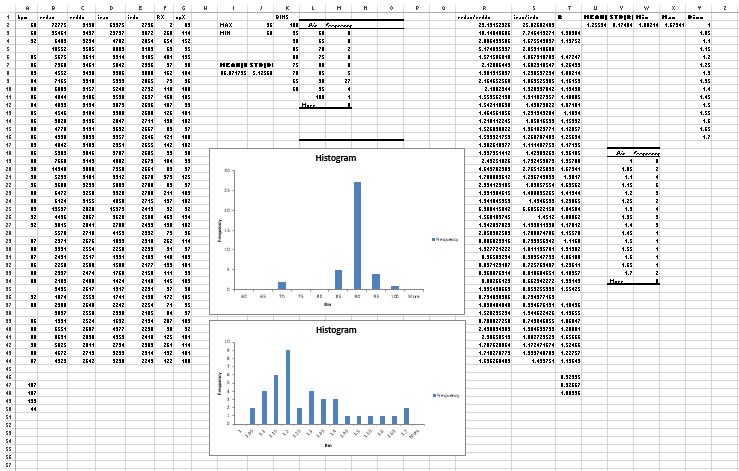


Figure 32 Subject A Daylight Data

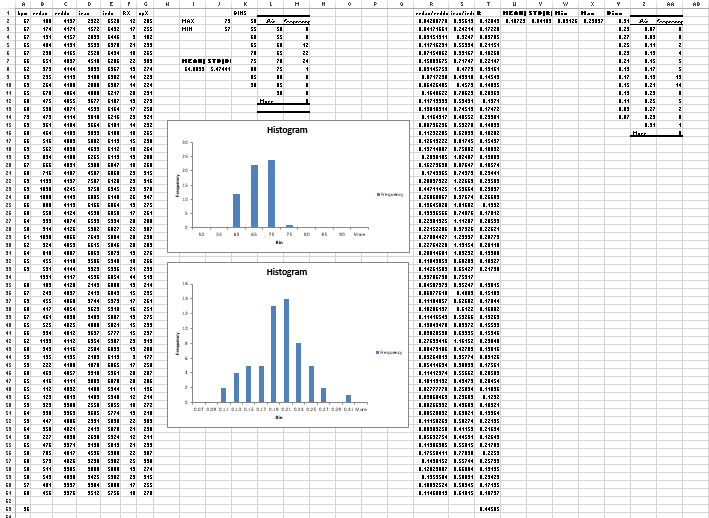


Figure 33 Subject B Daylight Data

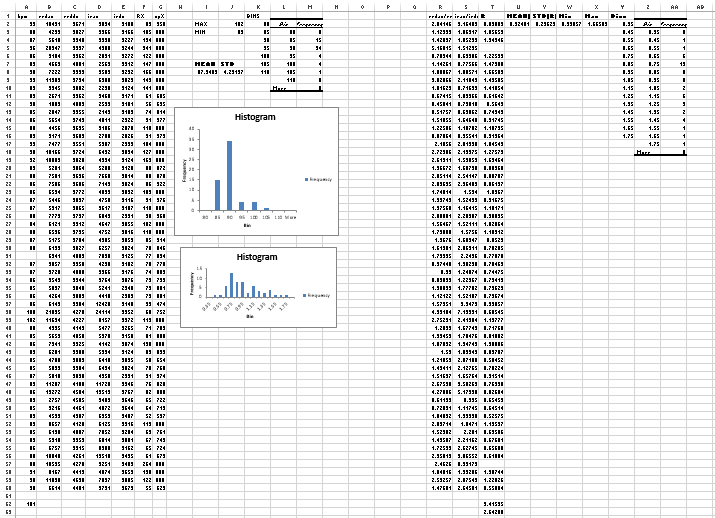


Figure 34 Subject A Artificial Light Data

# Bibliography

Throughout this project we gained information from lots of sources which are listed throughout the document.

Some other useful sources of information were:

<http://www.lifebox.org/>

University of Limerick Library

CoderDoJo Limerick

Texas Instruments Website

Cypress Semiconductor Website

123D Design

howequipmentworks.com

http://www.atlassian.com/software/jira

1. T.L. Rusch , R. Sankart and J.E. Scharf

   Signal processing methods for pulse

   Oximetry

   Comput. Biol. Med. Vol. 26, No. 2, pp. 143-159, 1996 [↑](#endnote-ref-1)