Dublin City University
School of Electronic Engineering

Coursework Submission Cover Sheet

Module	EE445 – Bioelectronics
Title	Group IoT Design & Make Project
Degree Scheme	MECEC
Year	C
Lecturer	Prof. Noel Murphy

Group 8 – Group Members	
Melroy Cordeiro	19210188
Prathosh Shastry	19211440
Victor Baulier	19214184
Mazhar Shaikh	19210936

We hereby declare that the attached submission is all our own work, that it has not previously been submitted for assessment, and that we have not knowingly allowed it to be used by another student/group. We understand that deceiving or attempting to deceive examiners by passing off the work of another as one's own is not permitted. We also understand that using another's student's work or knowingly allowing another student to use my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings.

Date: *23rd Apr 2020*

Note: Coursework examiners are entitled to reject any coursework which does not have a signed copy of this form attached.

For use by examiners only (students should not write below this line)

Comments:

Table of Contents

Abstract

This report outlines the approach taken for implementation of a cloud based IoT solution to measure and record the ECG signals captured and amplified from the person's body. It's a group project performed by four members and the report sums up the contribution of all the members located at distant places. Initially the team started by understanding the core concepts of the overall project. The report lists down the different areas of the project working and how the team achieved the results. Hardware designing section illustrates different components in the overall design. Each sub-section of the hardware design provides description of whole circuit part-by-part. Started with 'AD620' as instrumentation amplifier to amplify the bio-signals received at electrodes. All the difficulties related to the testing failures such getting 50Hz interference mixed with the bio-signals, etc. were handled and solved successfully. Later section illustrates the 'MSP432' micro-controller board implementation which receives the amplified bio-signals for processing. Energeia launchpad was successfully installed and compiled. Code to implement the processing of analog signals to digital equivalent was created. This section also illustrates the implementation of CC3100 Wi-fi board which is used to transmit the processed signal from controller to the cloud platform. For cloud platform the team has selected Ubidots services which implements MQTT protocol and has some advantages over Thingspeak, which has been illustrated in the section of Ubidots theory. For hardware simulation 'Multisim 14.2 Pro' edition software was used to test the output of the hardware design. Lots of challenges has been faced which has been listed out in detail in the observation section and the overall results and ECG signal output with simulation results are covered in results section. At last group work and co-ordination with workload division is described and how the team was successfully able to achieve the ECG output with IoT solution framework implementation.

Introduction

The main objective of this assignment is to develop an electrocardiograph (ECG) monitoring device that will be connected to a cloud platform. The ECG is read from a person using electrodes connected to the arms and the leg and then amplified using an instrumentation amplifier. It is then digitised using an Analog to Digital (ADC) converter and fed into a microcontroller. The microcontroller processes the signal and sends relevant information to a remote server.

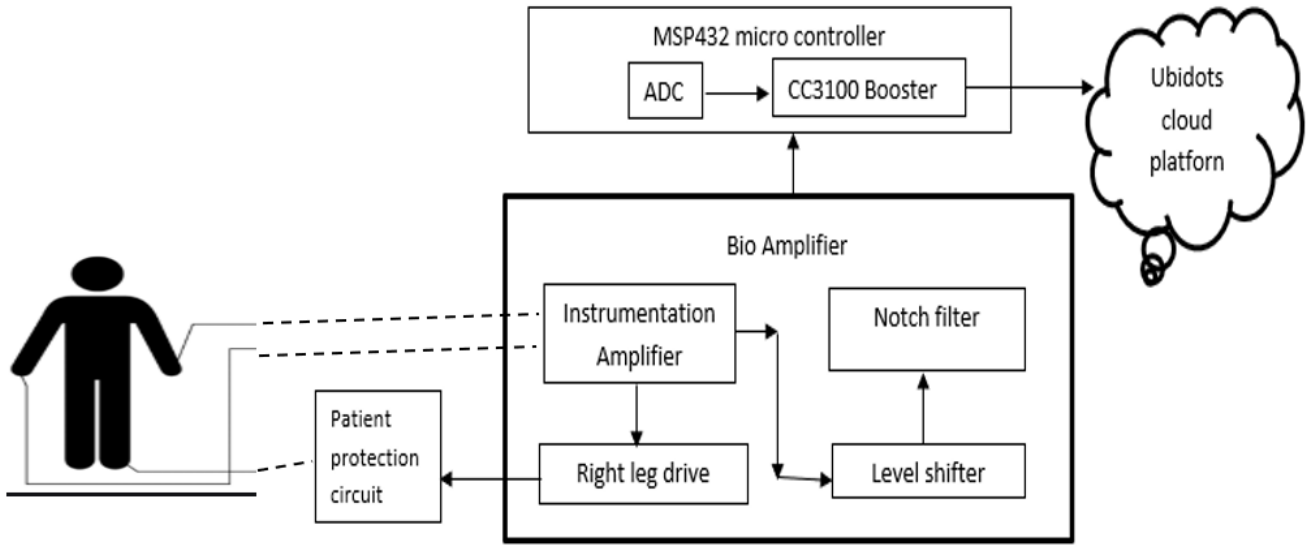
Hence, this project is complete since it involves developing a hardware architecture able to measure some ECG signals from a user while answering some constraints linked to the development of a medical device. It also involves the internet of things and embedded systems skills since the signal must be carried out from the analog circuit to the cloud platform where it can be stored and analysed. These develop technical skills and some organizational skills since most of the projects must be done remotely due to the covid-19 pandemic. The four of us are working remotely and only one device is built.

The following will give further information about the different technical and organizational issues that have been faced and the various approaches used to solve them.

Overview

An Electrocardiogram is a procedure where the electrical activity of the heart is captured by attaching a sensor over the skin. An AD620 is a highly accurate and low-cost instrumentational amplifier which requires only one external resistor to set a range of gain, so this is used to amplify the captured cardiac biopotential signals from the electrodes connected to the subject's left and right arm. To avoid the reverse flow of signal from the instrumentation amplifier, the patient protecting circuit is built with the help of TLC084. The patient protecting circuit will not allow the reverse signal from the instrumentational amplifier towards the subject. To isolate the common-mode from ECG signals, the subjects' right leg is connected to leg drive. Then the signal from AD640 is fed to a level shifter, where the signal can be shifted to a range of 0 to 3 V so that it is possible to observe the whole ECG signal. Then the signal from Shifter is

fed to notch filter where the 50Hz Ac frequency is filtered out. Lastly, the analog filtered data is read by the microcontroller and is uploaded to Ubidots where it is plotted for visualization.



Hardware Designing

ECG signal is the one that starts at the sinoatrial node of the heart and triggers the heart to beat and pump blood. These signals are very low in amplitude to be able to be detected from an external device using electrodes. This calls for a need of an instrumentation amplifier. An ECG instrumentation amplifier must be able to amplify low amplitudes signals (~ 0.5 and 5 mV) corresponding to the potential difference between electrodes while being immune to noise. However, due to the field of application of this type of device, the hardware design must respect the following guidelines:

1. The measurement must not affect the physiological process of the person under test.
2. The signal must not be distorted and be a true replica of the original signal, only amplified as features of the raw physiological signals are important for analysis of heart condition.
3. The signal to noise ratio (SNR) must be maximized.
4. The patient must be protected from any hazard of electrical shock. Hence the device should have isolation.
5. The device must be protected from high voltage inputs.

These constraints define and guide the hardware design process. The following sections explain the process of designing the device and mentions the challenges one faces.

Instrumentation amplifier (AD620)

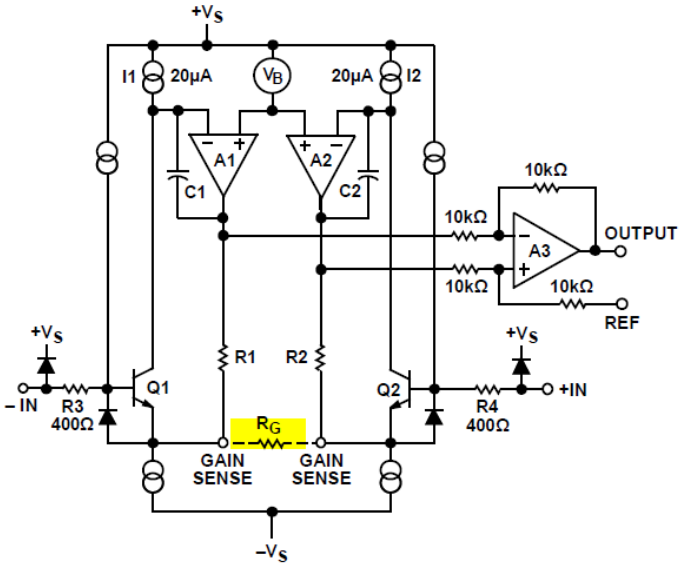
An instrumentation amplifier is based on the utilisation of a differential amplifier. This type of device, as its name indicates, amplifies the voltage difference between two input signals. The bio-signals coming from the two electrodes contain some noise, for example, main voltage noise(50hz). This noise is common to the two electrodes and hence is cancelled out at the differential stage. As exposed in the constraint number 3, the solution must maximize the SNR and then reduce this common mode noise.

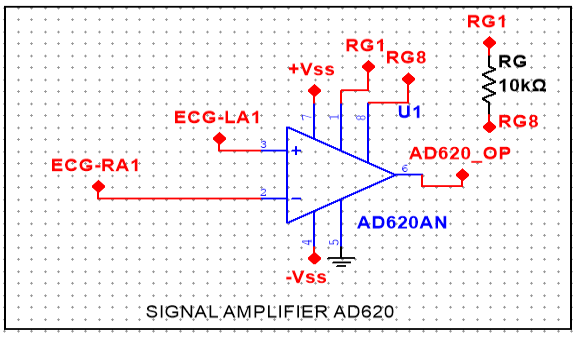
Another limitation of simply using a differential amplifier is the fact that the input impedance is low. The signal is not well conserved through the hardware because of the very low input voltage. Our final solution must have a high input impedance (a low input current must induce a decent voltage in the hardware). A solution is to use an instrumentation amplifier which is a differential amplifier with input buffers.

The AD620 is a monolithic instrumentation amplifier based on a modification of the classic three op amp approach. Absolute value trimming allows the user to program gain accurately (to 0.15% at G = 100) with only one

resistor. As show in the figure 2, the op-amps A1 and A2 are closed loop with negative feedback. It means that voltages at 'GAIN SENSE' points correspond to '-IN' and '+IN' and the voltage drop across 'Rg'. Hence, the common mode will be reduced.

The internal gain resistors, R1 and R2, are trimmed to an absolute value of 24.7 kΩ, allowing the gain to be programmed accurately with a single external resistor. The gain equation is defined in the AD620 datasheet and is stated in the below figure. The value of the voltage gain must be maximized in order to be able to see as much details as possible in the output signal, as the constraints 2 highlights.





$$G = \frac{49.4k\Omega}{R_G} + 1$$

$$R_G = \frac{49.4k\Omega}{G-1}$$

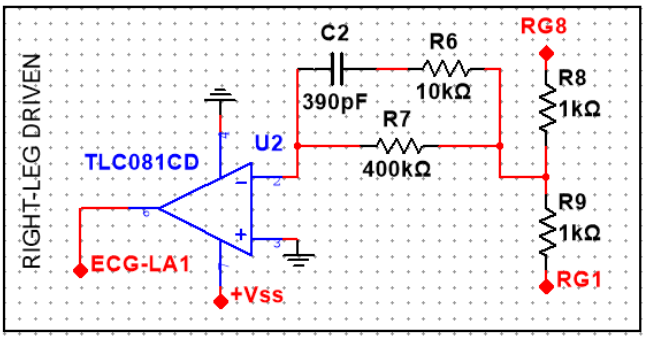
Band-Pass Filter

Average heart rate of a person lies around 1.1Hz and the signal we receive from the electrodes are very prone to interference. To avoid this band pass filter is designed and utilized in the bio-amplifier circuit, so that it passes frequencies only within the allowed range (0.05Hz to 150Hz). This can be achieved by using Op-amp (LM 324 / 741), capacitors to block/allow frequency component and resistors to unload the capacitor path. Due to limitation and constraints with component availability we have excluded the band-pass filter hardware in the final circuit, but has been tested and simulated in software.

$$\frac{1}{2\pi RC} = 0.05Hz \text{ and } \frac{1}{2\pi RC} = 150Hz$$

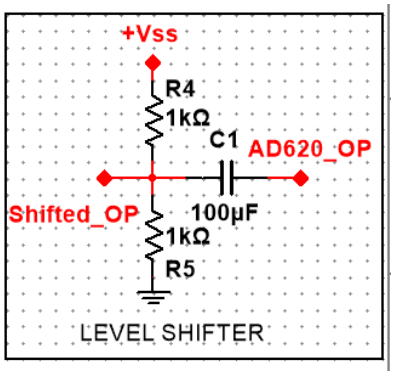
Right-Leg Drive Sub circuit

The instrumentation amplifier introduces a common mode rejection ratio (CMRR) that is not high enough to have a good SNR. In order to maximize this ratio, we added an active grounding to our design. The main idea is to use the common mode signal of the patient as ground reference. Hence, it is possible to isolate more accurately the common mode signals from the ECG signals. In order to do so, a third electrode is placed on the patient's right leg. The signal from the leg is free of any electrical signal for the heart and is then used to null out the common mode signal coming from the wrists. Having a high CMRR increases the performance of the measurements.



Output DC level shifting

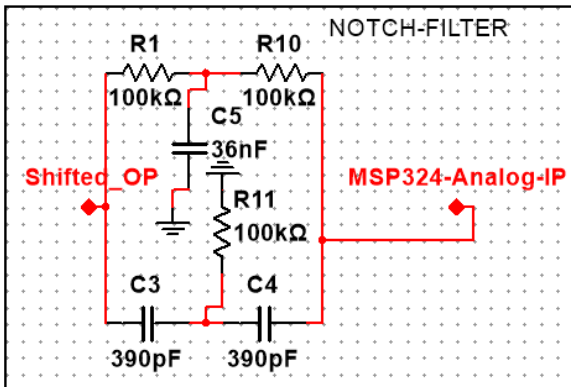
As the signal output from the AD620 swings from -3.3v to +3.3v, it cannot be directly fed into the ADC of the MSP432. This is because the ADC in the MSP432 has a range of 0v to +3.3v only and due to this the ADC would only detect the positive cycle of our signal while the negative voltage will be considered as 0v. To counteract this, a DC level shifter is being used where a DC shift is provided to the AC signal and the entire signal is moved in the range of our ADC. A bias of +1.5v is added by dividing the supply voltage using a voltage divider. The signal is introduced in the network using a 100uF capacitor which also acts as DC blocking. The signal output from this stage now swings from 0v to +3.3v and can be fed into our ADC.



Notch Filter Design

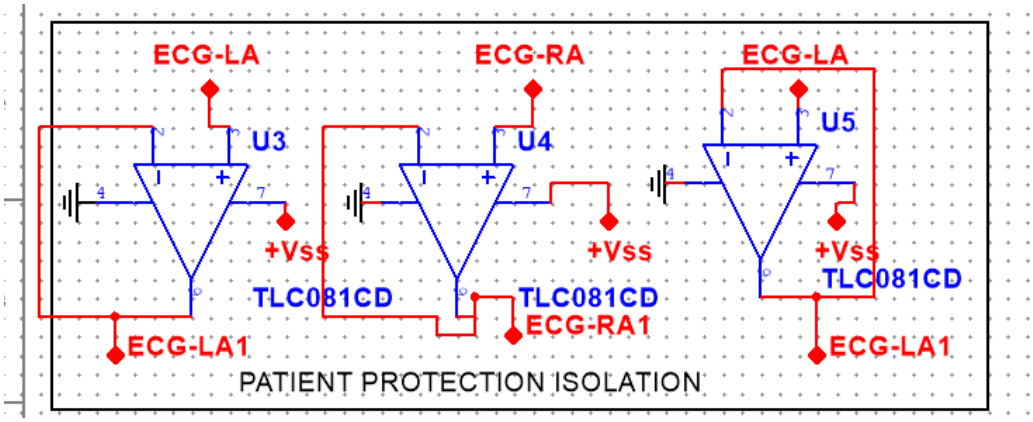
On looking at the AD620 output, the singnal seems to be coupled with 50Hz noise interference. As ADC is very sensitive and AD620 being differential amplifier amplifies very little signals in the range of ‘mV’. This same signal of 50Hz interference we received in the simulation in which the ECG signal was lost. So, as a solution to this we designed and implemented notch filter (narrowband twin-t notch filter) at the last stage of the circuit before sending it to micro-controller for processing.

$$\text{Notch Filter Frequency} = \frac{1}{4\pi RC}$$



Isolation for Patient Protection

One of the constraints concerns the safety of the patient. In order to avoid any risk of electrical shock to the patient (can lead to a cardiac arrest if avoided!), it is necessary to create an isolation between the patient and the bio-amplifier circuit. As we were having TLC084 single powered op-amp with us, which contains four op-amps inbuild in one IC; we made use of three op-amps as unity gain buffers feeding them with the RA, LA and RL electrode inputs. This input fed will be supplied to AD620 input pins unchanged and acts as isolation protection for patient safety.



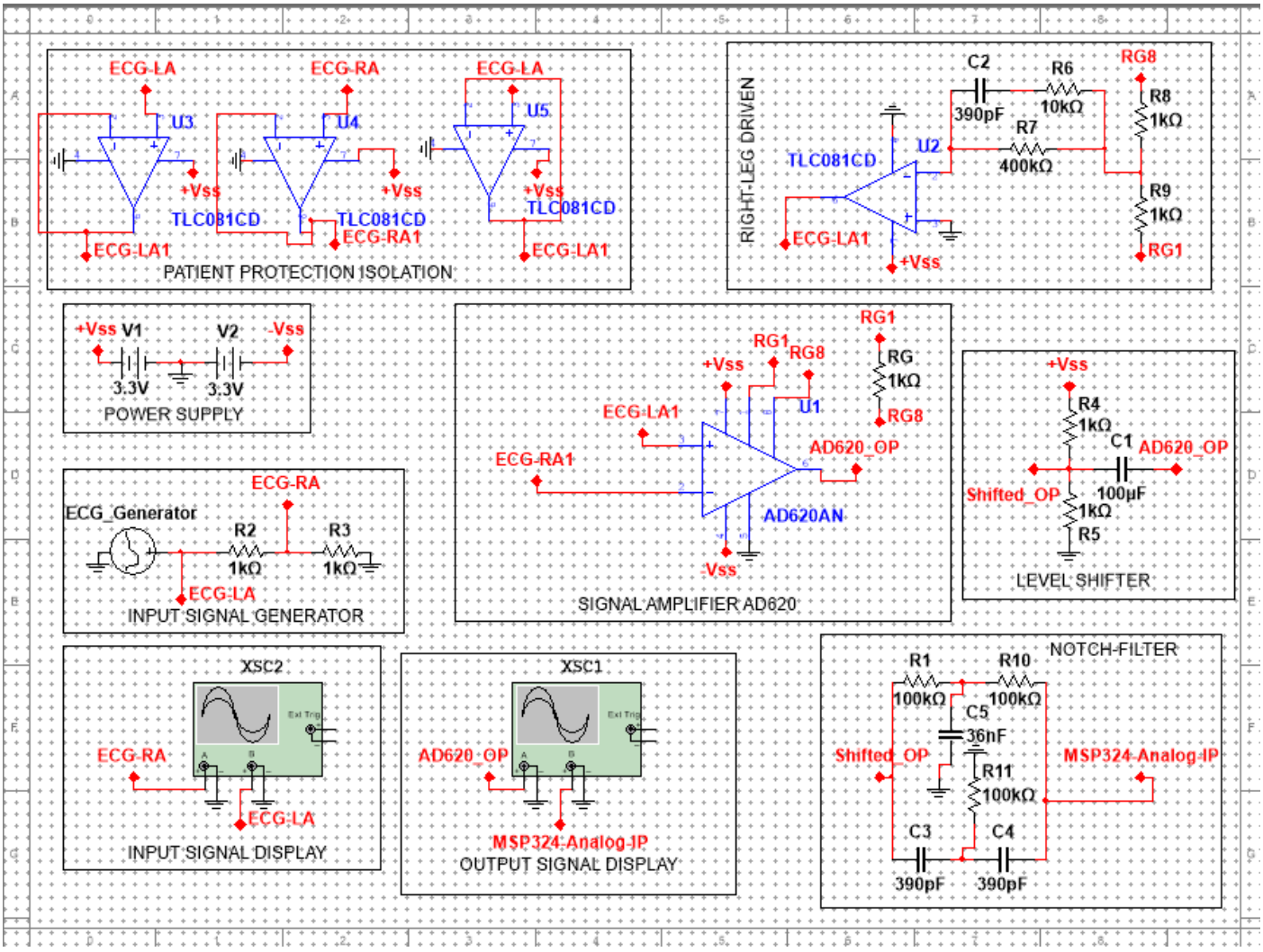
Final Complete Circuit Diagram

The simulations of the circuits have been done on Multisim 14.2 (by National Instrument) in order to test the hardware before implementation and to verify the theoretical design while allowing us to organise the design process. Due to covid-19 social distancing, the direct actions on the hardware were limited. Using this software gave us the opportunity to try different designs without physical actions on the hardware. We were able to find the best values for the resistors (mostly for choosing the gain) and the capacitors (notch filter). Once the simulation was done and the hardware chosen, we were able to implement the hardware.

The hardware has been set accordingly to the different literatures we had:

- The datasheet of the instrumentation gave us precious information since they had an example of a design of an ECG using their device.
- The notes from the module.
- Basic electronics understanding.

Note: The output of the simulations is shown and explained in 'Results' section of the report.



MSP432 microcontroller interfacing

The MSP432 microcontroller by Texas instruments is used along with the CC3100 WIFI booster pack used as a daughter board. The ADC on the MSP432 samples the ECG received from the analog circuitry and sends relevant information further to the server using the MQTT protocol and WIFI.

By default, the MSP432 is set to 10-bit when using the energia software. However, this can be reconfigured to 14 bit to make use of the full range of our ADC. This is particularly important to detect small changes in our ECG signal. The MSP432 also supports a differential mode where, the two inputs can be input to the ADC and then get the differential out from it while also performing any digital processing. But for this project, an AD620 bio-instrumentation amplifier is used which takes the differential signals from the left and right arms while also using the driven leg concept providing better noise immunity (this is not used here).

1. Installation of Energia challenges in enabling of boards

   The Energia IDE is used to program the MSP432. It provides a Arduino-like interface which enables easy shift from Arduino compatible microcontrollers which are usually 8/16-bit controllers to much powerful controllers like the MSP432 which is a 32 bit controller with support for RTOS.
   On succesfull installation of the necessary device drivers, the board should be detected and displayed once plugged into the computer. To program the board, we need to install the necessary board packages. These board packages enable the code to be compile based on the selected board (MSP432 in our case). The board RED LaunchPad w/msp432p401r EMT was installed using the board manager which enables us to compile code for the MSP432 and the CC3100 WIFI booster pack (refer to figure 1).

Note: One particular thing while installation of the board is, if the installation has failed once, uninstalling the software does not delete all the packages. Hence, any installation later on will always fail. The solution to this was to locate the board manager directory, deleting the installed board packages and then reinstalling the necessary boards.

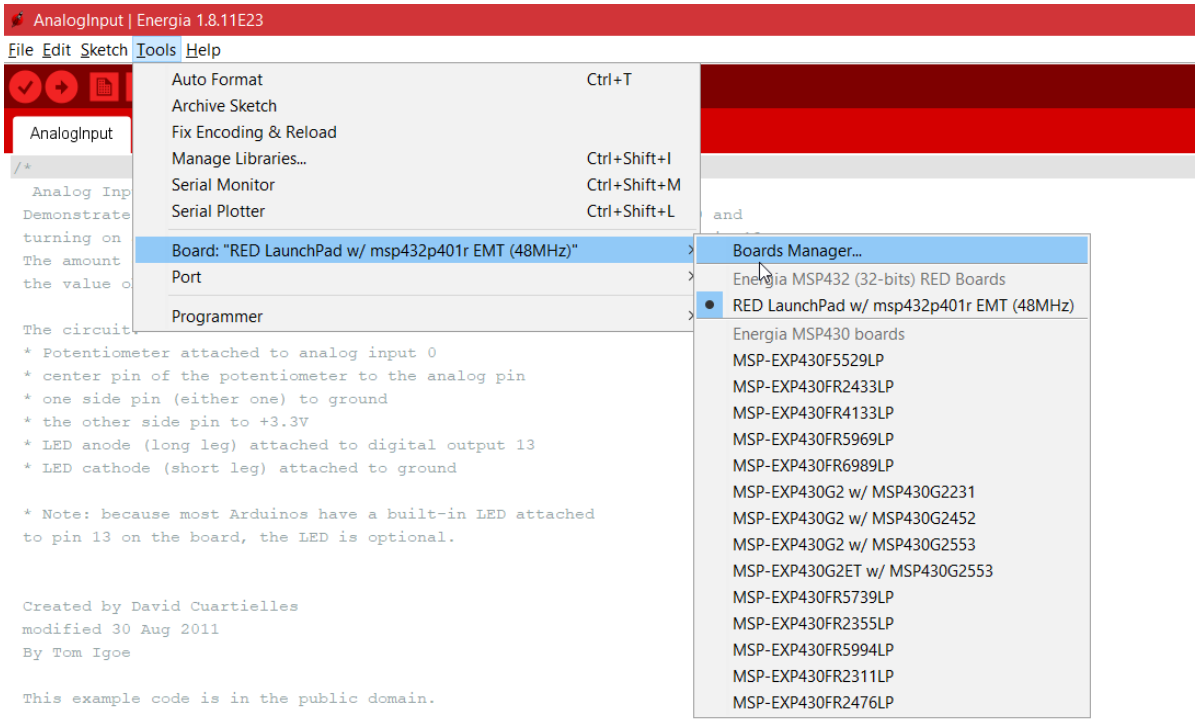


Figure 1: Board Installation in Energia

2. Connecting AD620 output to the ADC Input

As configured, our reference ADC voltage is 3.3v and the resolution is 14 bit. This means we can sense a voltage value of $\frac{3.3}{2^{14}}$. Hence the minimum voltage we can mesure is 0.201 milli volt of a chang in input voltage.

As the supply voltages to the AD620 is between -3.3 to +3.3 volts, the maximum output voltage from the AD620 remains capped at the VSS and would get clipped above that. This ensures the voltage to our ADC would never exceed 3.3v and does not require any further circuitry for input protection. Furthermore, our ADC only works on the range of 0 to 3.3 volts, but, an ECG wave consists of both positive and negative sections of voltages. If the signal output from the AD620 is fed directly into out MSP432, the negative voltages of the ECG wave will not be detected by our ADC and will be only seen as a 0 volts by the microcontroller. To counteract this, a DC bias(explained above in the level shifter circuit) has to be provided to the signal. This means, instead of the signal sweeping from -3.3v to +3.3v, it will now sweep within 0 to +3.3v, making sure that our ADC can sense the wave form to its true shape. The bias provided is such that the 0 now exists at 1.5v and any voltage greater than +1.5v is considered positive while, any signal lesses than +1.5v is seen as negative.

3. Sampling of ECG signal

An ECG signal has an approximate frequenccy of 1.1 hz. Hence to be able to sample efficiently and without distortion, we must sample our signal at atleast 2hz or above according to nyquist's criteria. Hence we measure a sample of the AD620 every 30 millisecond giving us a sampling frequency of approximately 3.3hz.

4. Detecting of Heart Beat

The ECG obtained from the AD620 output is sampled and gives us a raw ECG singal at the microcontroller to be processed. We have used the signal to detect the heart beat of the patient. The QRS complex in the ECG signal can be used in order to estimate the heart beat of the user. The peaks of the QRS complex in the ECG

signal is detected by the microcontroller using a set threshold. When the signal crosses this threshold, a peak is detected and out heart beat count is increased. The peaks are detected for one minute and sent to the server and a new reading is started again.

Note: Thresholding is not the best method for peak detection and has been done for demonstration purposes and to keep things simple. This is because the ECG singnal is affected by artefacts and therefore fixed thresholding does not work effectively.

5. Sending data to the MQTT broker

The Ubidots MQTT broker is used (why ubi dots is explaied in next section). The MQTT protocol is used where, the MSP432 acts as a publisher. The MSP432 publishes our heart beat data in a JSON format which is then plotted out at the Server.

Reconnection:

```
if (!client.connected() && TimeSpent(lastConnTime, 5000)) { //check if we're connected, if not try to reconnect
  //timespent does a reconnection every 5000 if we're disconnected to take care that the reconnection is not done continously
   reconnect();
   lastConnTime = millis();//save last connection time
 }
```

The above code connectes to the server and takes care of reconnection. It uses a hardware timer while reconnection to the server to make a connection attempt is not made while a previous connection is in process.

Forming of JSON:

```
sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);//forming data JSON
 sprintf(payload, "%s", ""); // Cleans the payload
 sprintf(payload, "{\"%s\":", VARIABLE_LABEL); // Adds the variable label
```

The above code forms the JSON data to be sent to the ubidots server. And appends the device label to the topic while forming the payload.

Reading of analog data and thresholding:

```
 float myecg = analogRead(SENSOR);//reading input from AD620 into our adc
 //   Serial.println(myecg);//plots out the ecg signal received on our adc pin
if (myecg > 732) { //thresholding to measure a peak in the QRS complex
   beats += 1; //if peak detected, count it as a beat
   Serial.println("Peak");
 }
```

The above code reads the analog pin using the analogRead() function. This data is stored in the variable myecg. The value is compared to the threshold value of 732, if it is higher than the threshold, the value of beats variable is incremented to count a beat.

Publishing heart beat to the server:

```
if (TimeSpent(heartRateTime, 60000)) { //if 1 minute is passed, publish the value of heart beat and reset our variable to 0. This helps us get the BPM
   Serial.println("Publishing data to Ubidots Cloud");
   Serial.println(beats);
   /* 4 is mininum width, 2 is precision; float value is copied onto str_sensor*/
   dtostrf(beats, 4, 2, str_sensor);
```

```
  sprintf(payload, "%s {\"value\": %s}}", payload, str_sensor); // Adds heart beat data in our JSON payload to be sent to the server
  client.publish(topic, payload);//publish heart beat data to the server
  beats = 0;
  heartRateTime = millis();//saving time of last sent heart beat
 }
```

The above code publishes one data packet to the Ubidots broker every one minute. It sends the value of our heart beat variable hence it is the BPM of the person under test.

Note: Network connections require dedicated processor time and is blocking piece of code. Hence it blocks our microcontroller from taking a sample while it is trying to publish data or reconnect to the server. Hence this affects our sampling rate. To counteract this, an RTOS must be used where one task with higher priority is dedicated to network connections and an other task takes care of sampling of data. Once data has been sampled, a semaphore is passed on to a task to process our signal. This takes care that the sampling rate is not affected in any way. This work can be considered as future work to perfect the device.

Ubidots Cloud platform

Ubidots is a self-designed IoT cloud application that performs specific functions intending to receive certain results and can be used from any internet-enabled device. This cloud platform is a combination of digital data such as real-time stock or gas prices and physical sensor data. The IoT system can be described as the digitalization of our physical environment, where most of the actions are captured using input/output devices and sensors, for any Internet of Things system, the IoT dashboard is the key Human- Machine interface component, which represents the digital data in the form of understandable visualization, by populated the platform with charts, graphs, switched and other widgets. This IoT dashboard can be used to monitor and control specific assets and processes from anywhere in the world.

Components of Ubidots

1. Creating new devices

   Ubidots can be used in almost all internet-enabled devices. The library for ubidots should be installed in device IDE. We can add devices to ubidots in three different ways.
   - Firstly, whenever the cloud receives the dot for the first time from the user's private token, the device will be created automatically.
   - Secondly, manually we can add a device by clicking the icon “+” which will be on the right corner of the screen.
   - Lastly, we can create a new device based on the device type with their predetermined properties such as Ethernet, WiFi, cellular, etc as shown in fig 1.

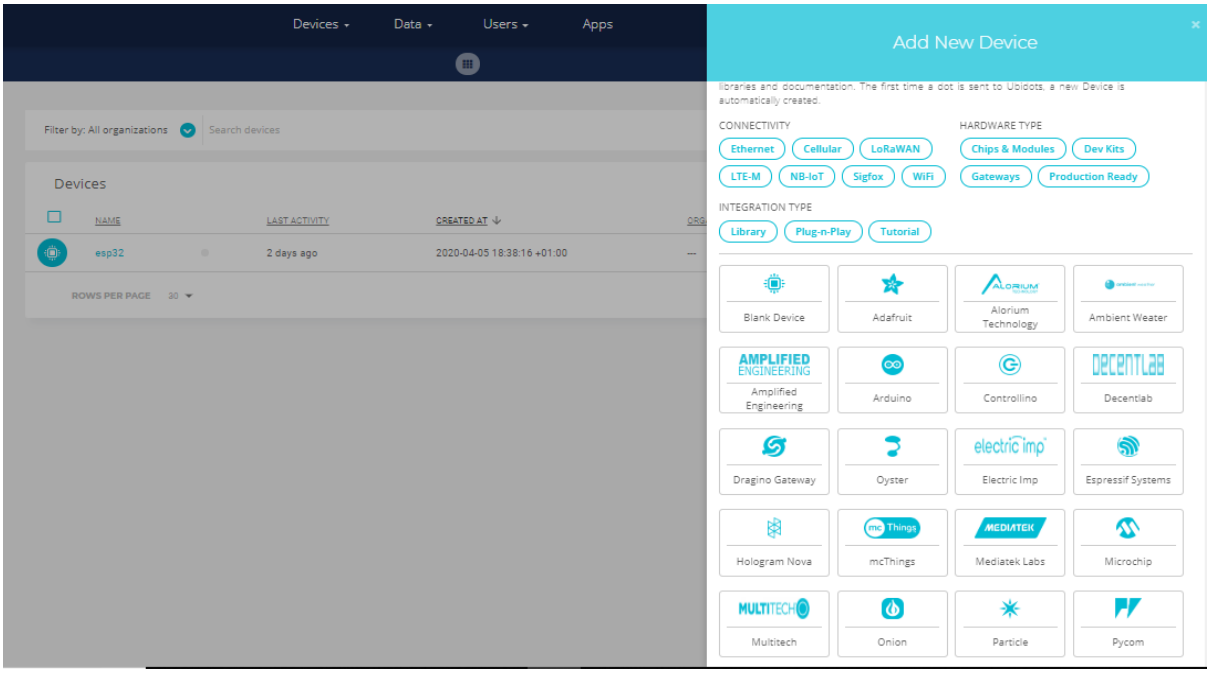


Fig 1

2. Variables to the newly added device

After devices are created, if the sensed data are received from controllers, it will be represented as a variable in its raw or calculated form. The variables are said to be the default if it is assigned a raw data and it is said to be a synthetic variable if it is assigned a corresponding statistical or arithmetical raw data. There are two ways to create a variable.

- Firstly, it can be done by assigning a label as a variable in our hardware code.
- Lastly, it can be created by clicking the "+" icon in the device screen as shown in fig 2 and a name should be assigned which should be the same as the variables label.

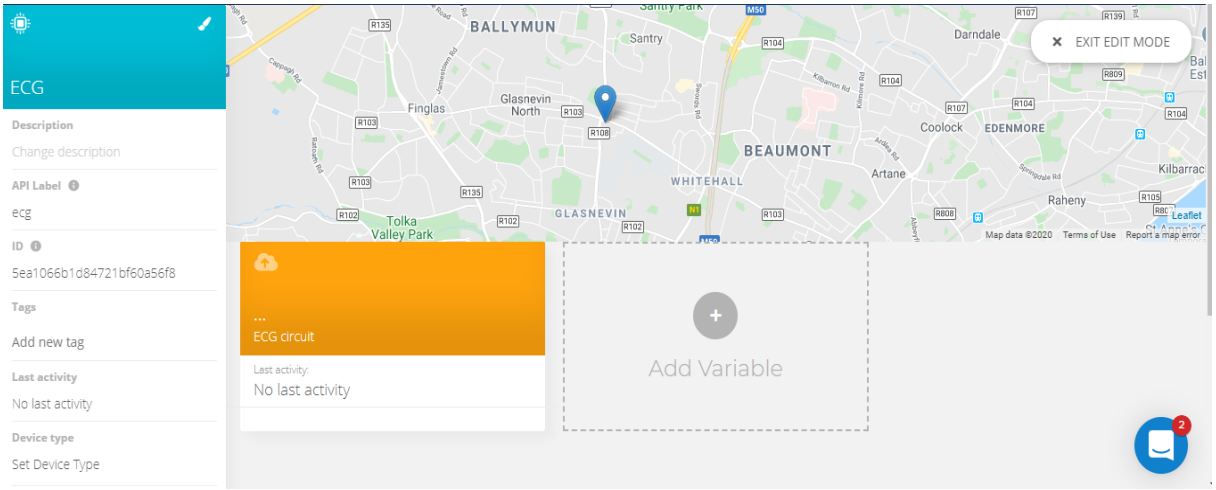


Fig 2

3. Selecting widget

To plot a graph, we need to select the appropriate widget from the dashboard drop-down and is shown in fig 3. As we are plotting the ECG signal, the line graph is selected from the widget dropdown.

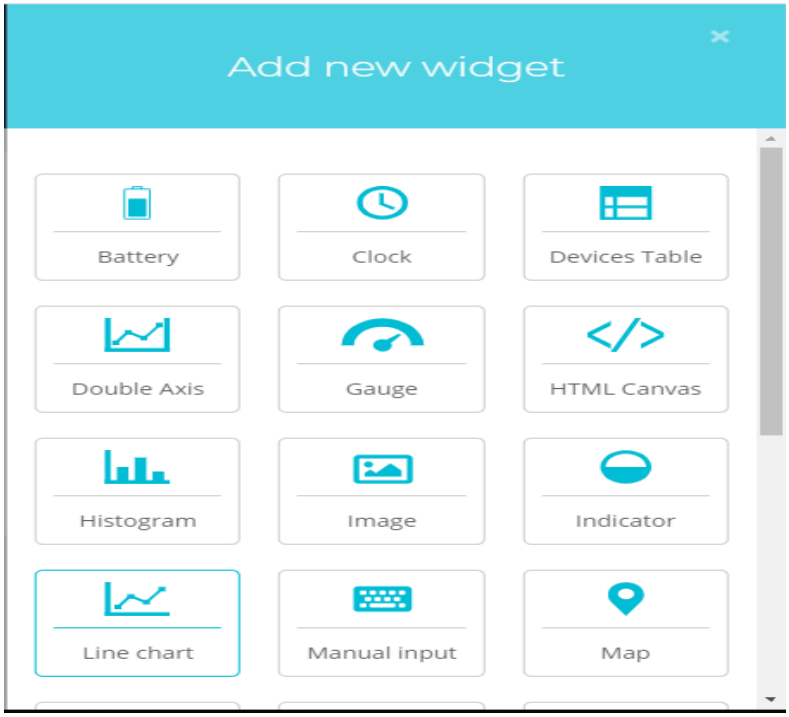


Fig 3

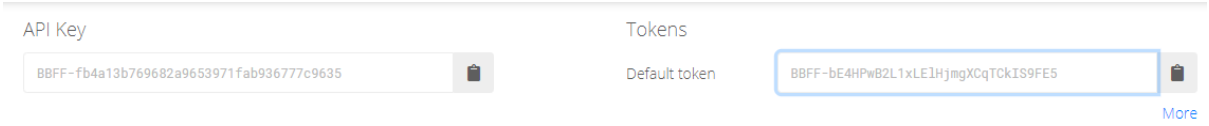
4. Connecting the widget to a variable

Now we have to connect the selected line graph widget to the variable created in the above step.

5. Authentication

Every packet requires a token, which can be obtained by clicking on “API Credentials” under the profile tab. In API credentials two types of keys can be found.

- API key: To generate our account token, a unique key will be used called a master API key.
- Tokens: Tokens are momentary and revocable key, which is to be embedded in all our API requests.



6. Connecting to Ubidots

We need to install PubSubClient to our device, which provides support for MQTT and the ubidots credentials to be used to push sensor data to the cloud.

The devices communicating over the internet have different challenges over other communication. The HTTP always remains idle for requesting data from the known devices and pushing data over an unreliable network using HTTP is not dependable. So we have decided to use Message Queuing Telemetry Transport protocol. The MQTT is a lightweight messaging protocol used in the machine to machine publish/subscribe communication. This protocol is majorly used in low level embedded devices because it provides decoupling between the publisher and subscribers, as a single publisher can publish data to multiple subscribers.

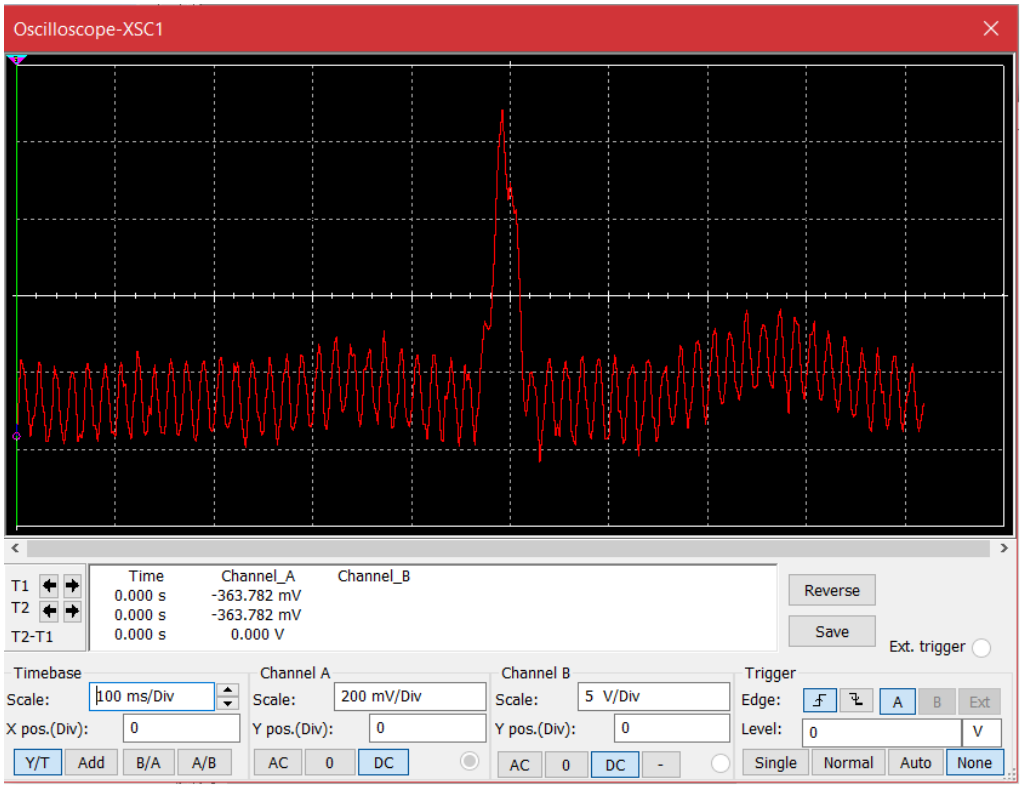
Benefits of MQTT over HTTP

MQTT allows us to bring event-oriented architecture to the web by providing low latency communication between servers and clients. The MQTT provides different Quality of Service levels which provides reliability in the delicate environment. The higher QoS of MQTT level provides the guarantee of messages by a four-part handshake between publisher and subscriber and also provides an option for a retained message and persistent connection. In MQTT an average of 21,000 messages per hour can be sent over MQTT, the only average of 1,926 messages per hour can be over HTTP. The MQTT consumes an average of 0.00082% of battery per message at the time of messaging publishing but on the other hand, HTTP will consume 0.00975% of battery per message at the time of message publishing. As we have to send real-time electrical activity of the heart, it is not possible to send a high sampling rate over HTTP. So we decided to use Ubidots instead of Thingspeak as Ubidots provides support for MQTT and has better Graphic User Interface.

Results

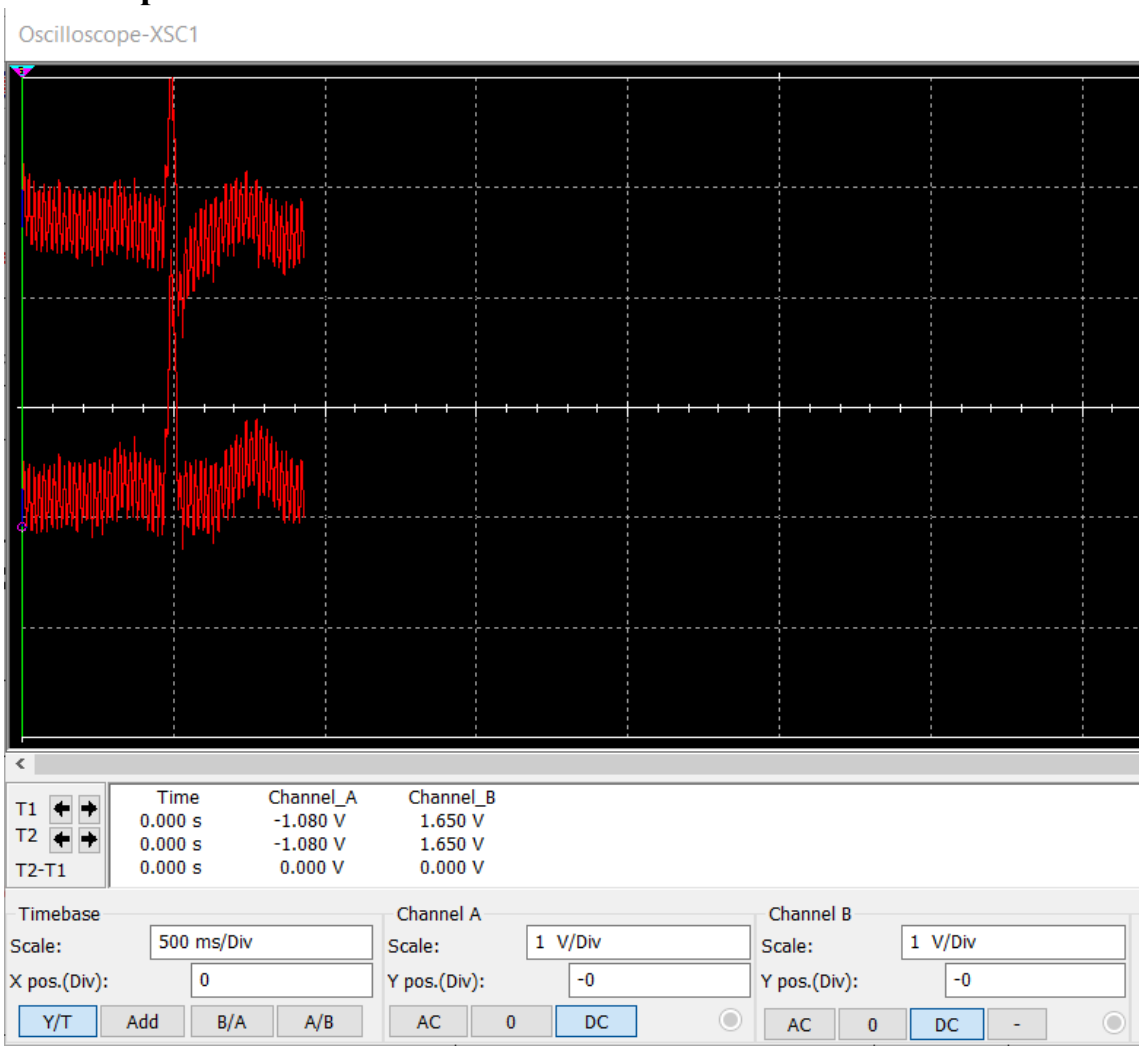
Simulation:

- **Ecg Signal**



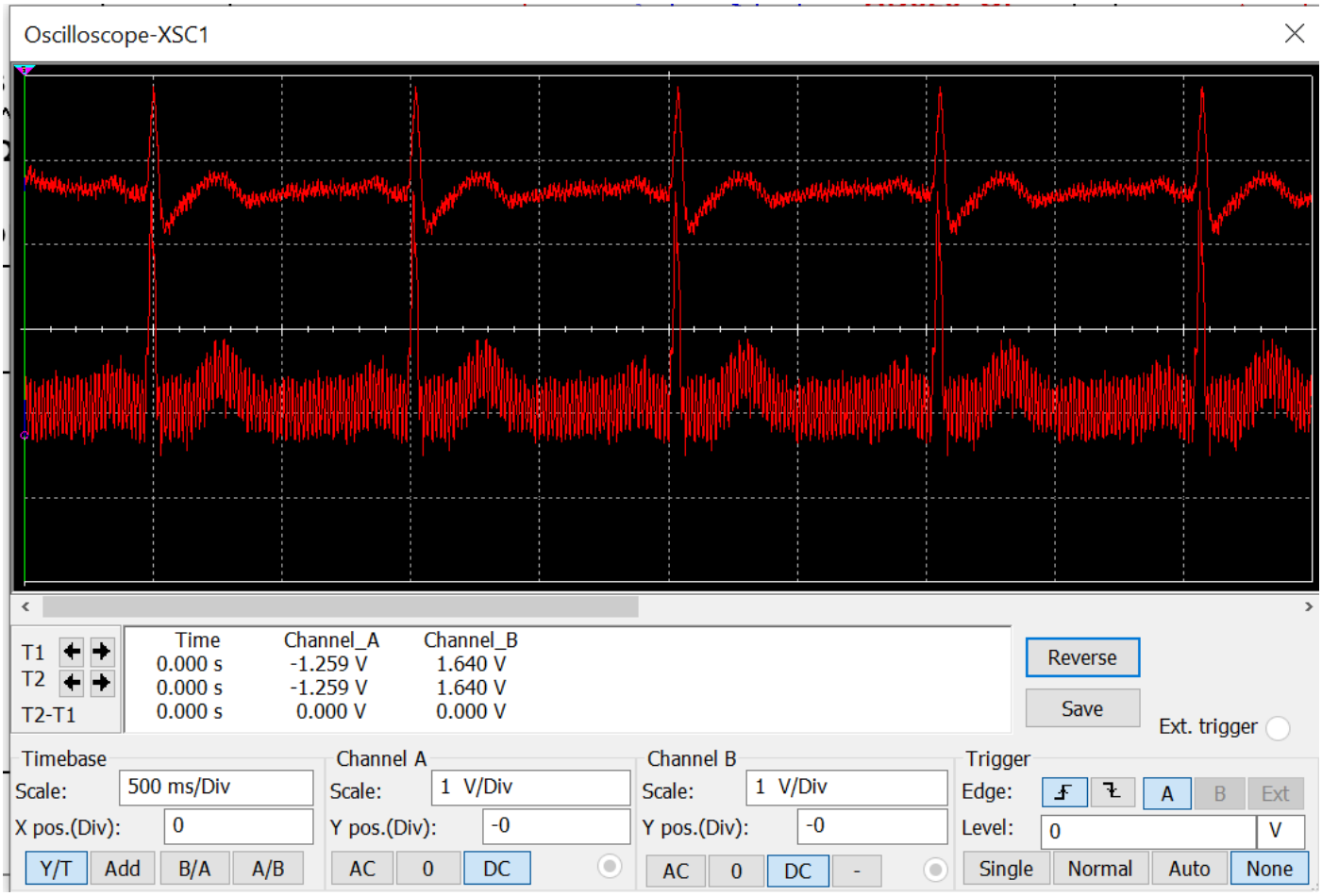
This is the input raw singal fed to our AD620 in the simulation. It also shows the 50hz noise coupled in it.

- **AD620 output without notch filter**



The above image shows the ecg amplified signal by the AD620. The first signal is the level shifted signal and the second signal is the raw signal from the AD620. Note the scale is set to 1v/Div and is the amplified signal.

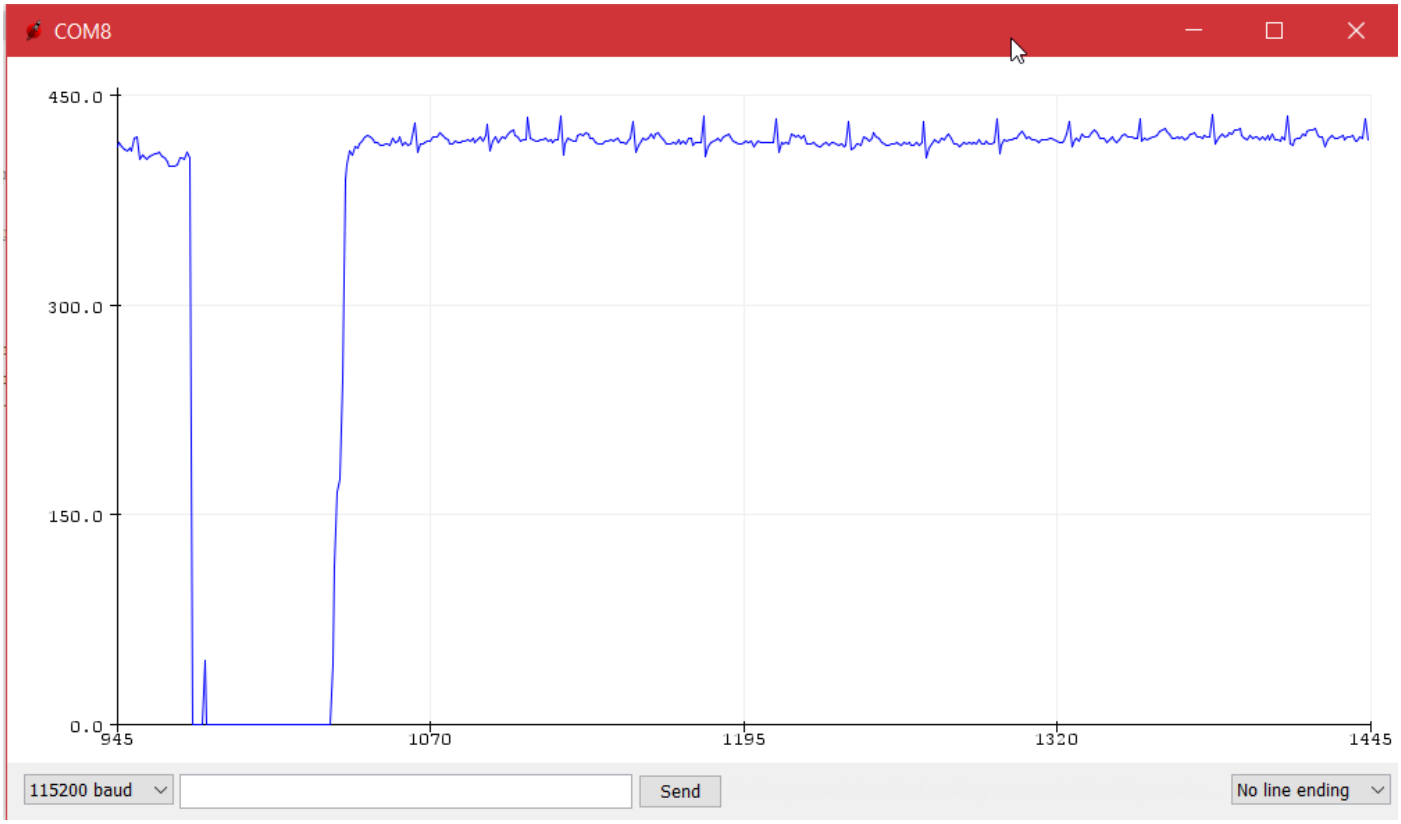
- **AD620 output without notch filter**



The above signal is the raw and and notch filtered output. The 50hz noise is filtered and is a clean ECG signal with very low noise.

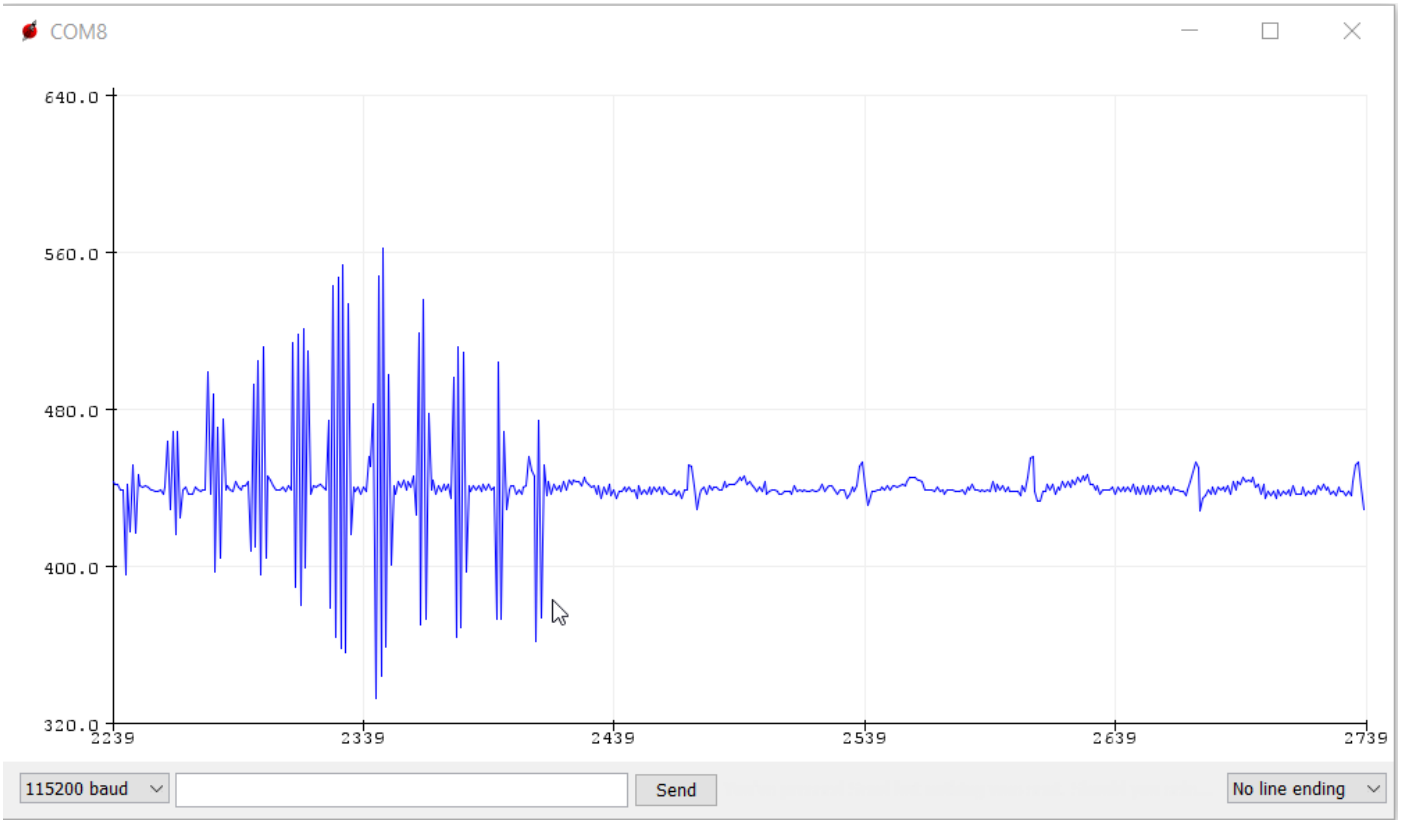
MSP432 output:

- **ADC detected ECG data without subject movement:**



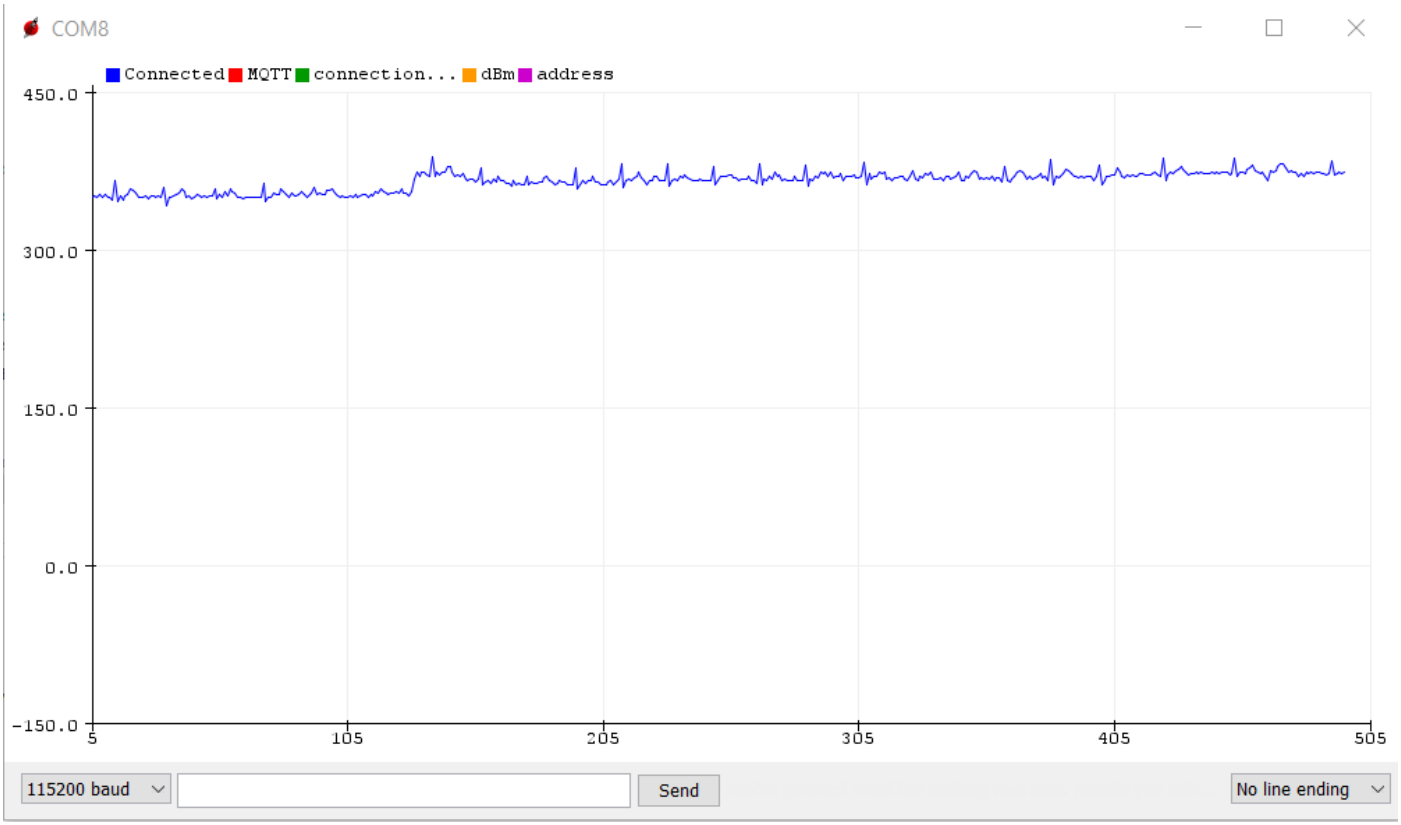
The above image shows the ECG data sampled and plotted by the MSP432. The the dip in signal shows the electrodes being removed and inserted again to detect the ECG signal again.

- **ADC detected ECG data with subject movement:**



The above image shows the ECG signal with artefacts. The subject's arm was moved which causes artefacts in our signal.

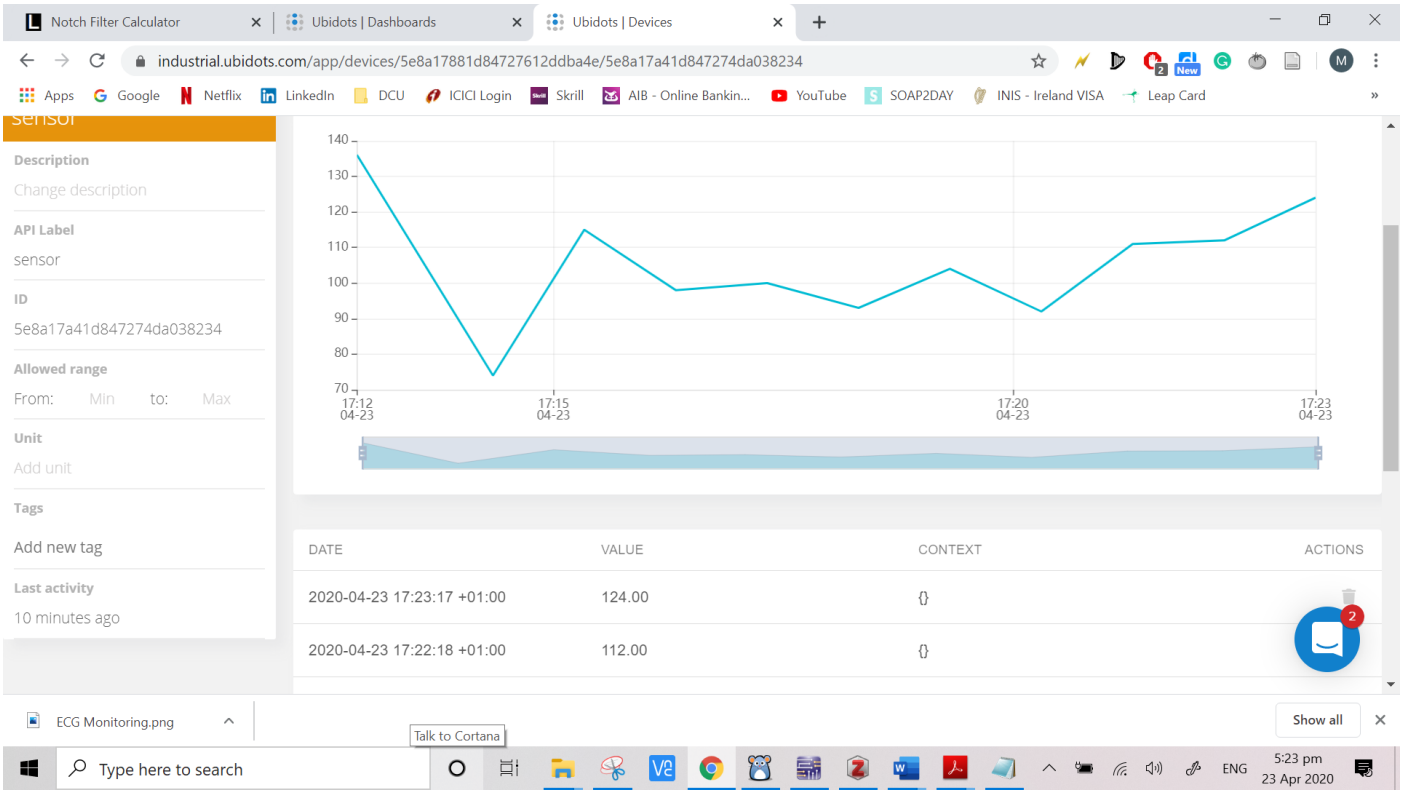
- **Publishing of ADC data**



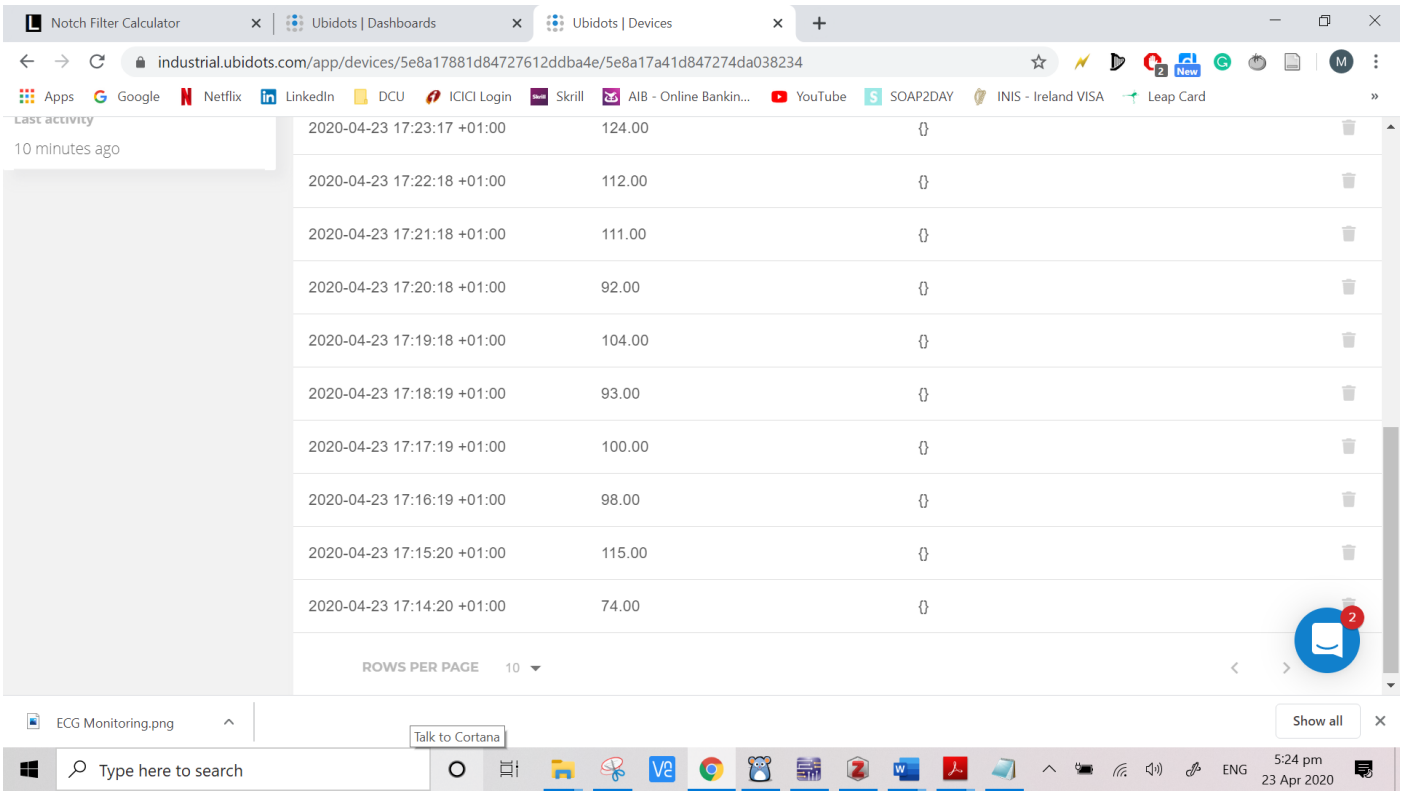
The above image shows the data being published to the Ubidots MQTT broker. The flags on the top of the graph show the status of connection to the server and publishing state.

Ubidots output:

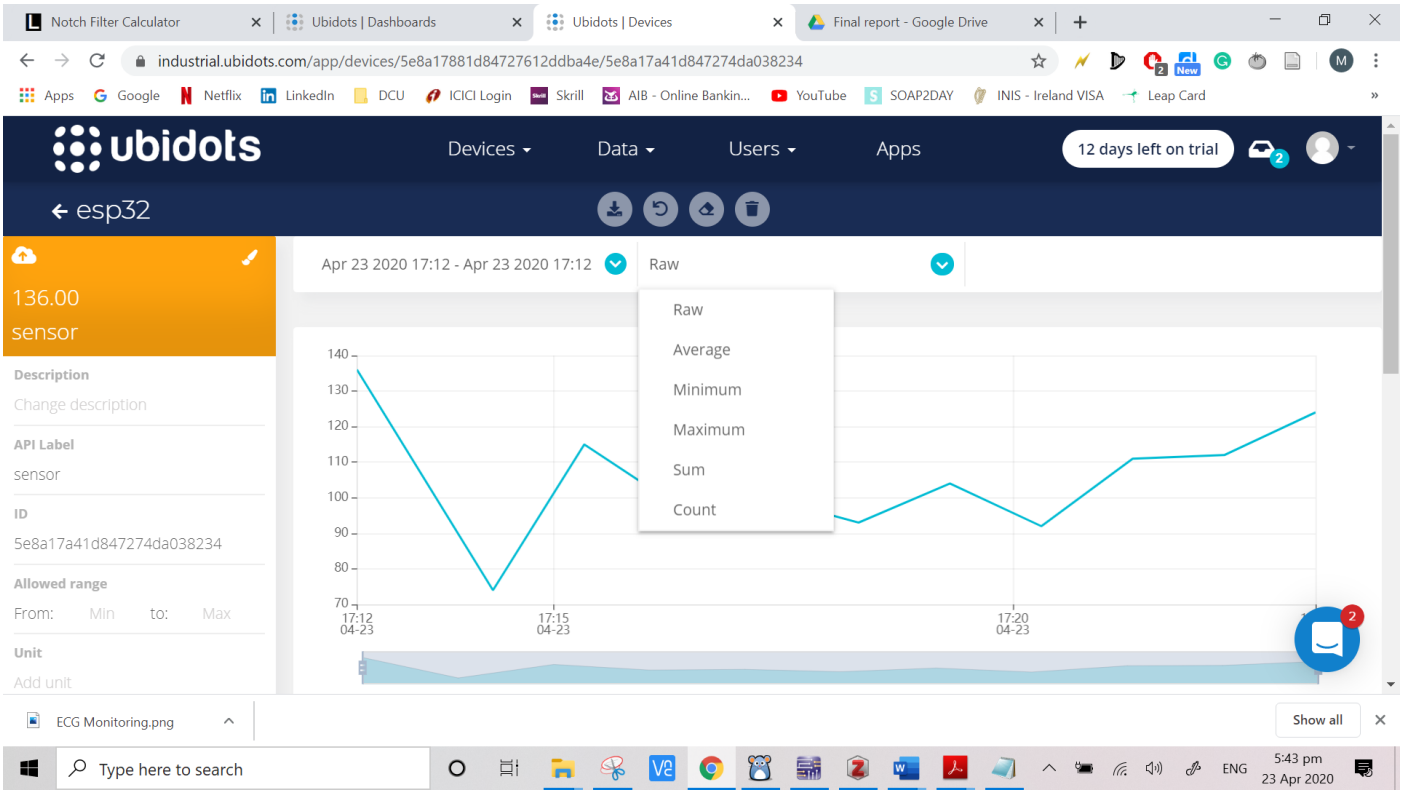
- **Heart Rate plotted at Ubidots**



The above image shows the heart rate being plotted at the server. The dip in data shows the electrodes being disconnected and then connected again. The user's heart rate is in the range of 80-120 bpm.

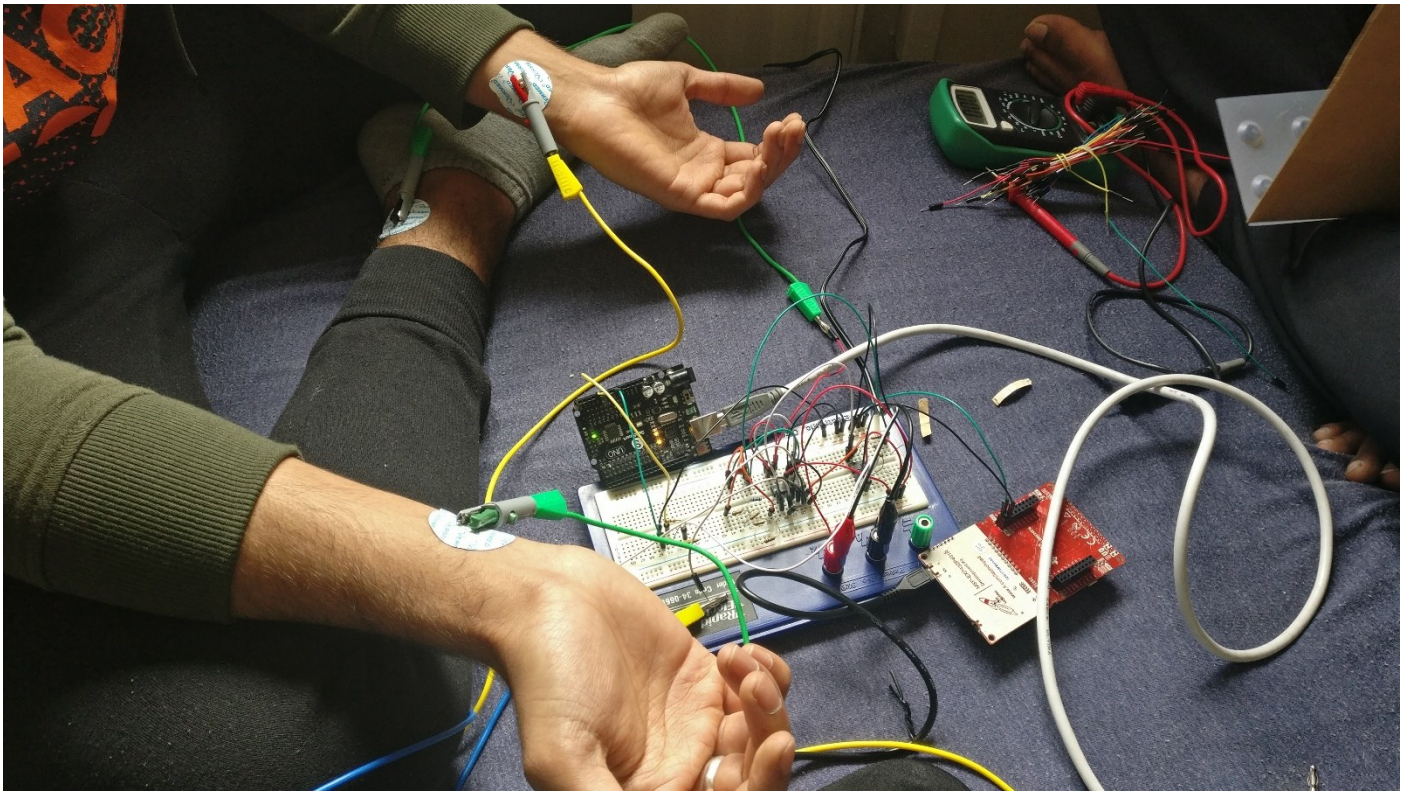


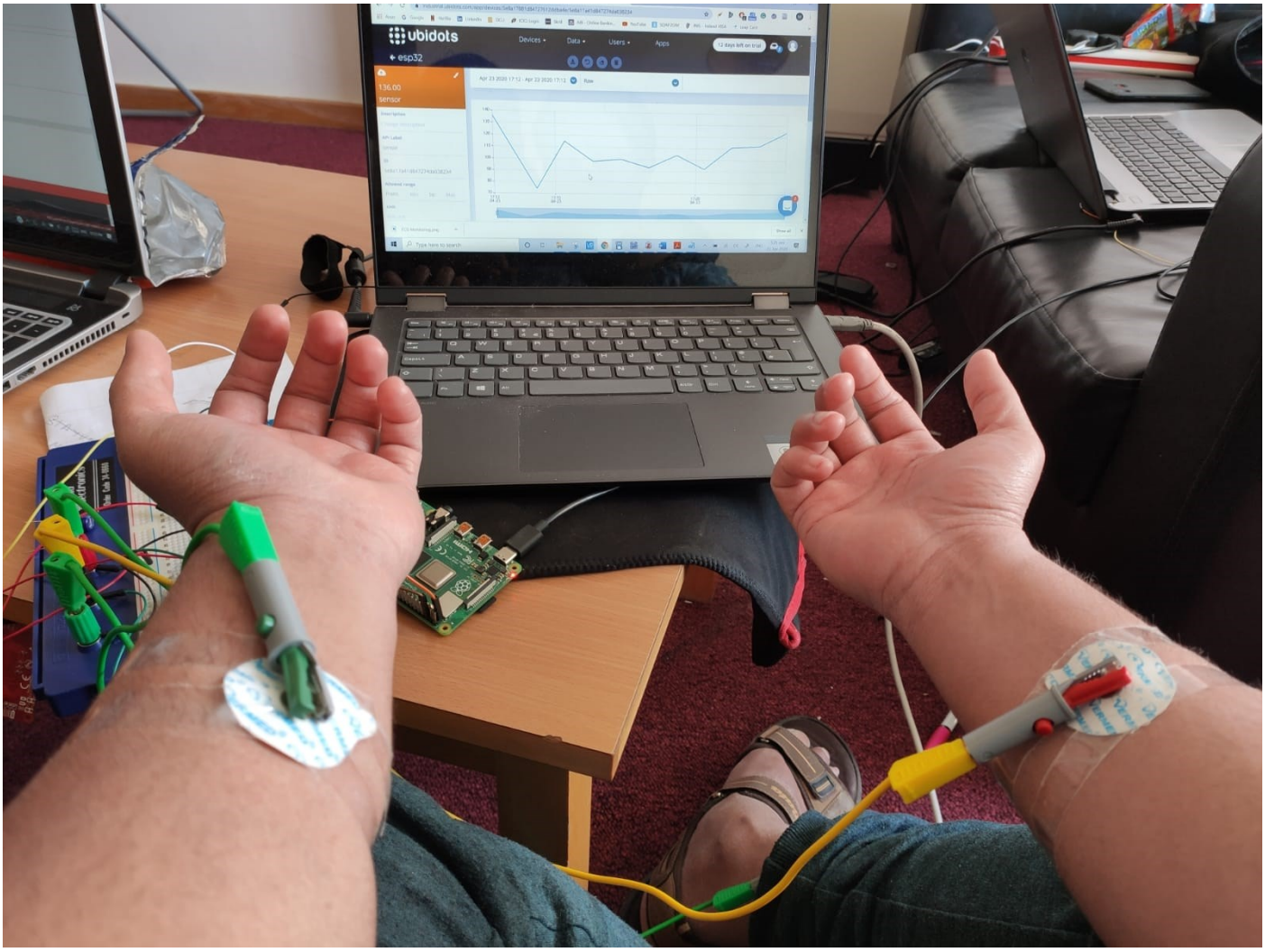
The above image shows the data that was published to the server after which it is been plotted.



The above image shows that the plotted data can now be shown as average, minimum, maximum etc. by selecting at the server. If average is selected, it will display the average data for the given time interval.

Physical Connections on the Bread board:





Group Work and Issues Faced

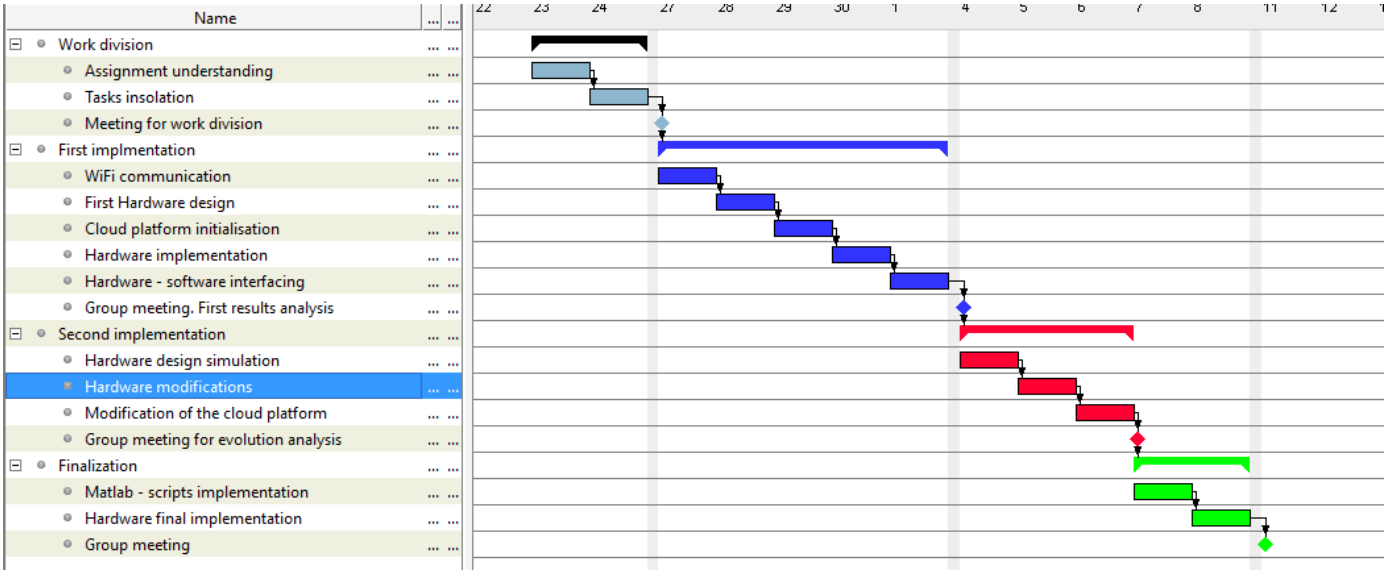
Due to the covid-19 scenario, the group was divided in 4 with impossibility to physically meet. Three of us were in Ireland and one was in France where one person had the hardware components.

Work division

The work has been divided in two main parts: the hardware and the software development. Two people have been allocated to each part. In order to be able to divide the work, the first step was to analyse the subject in depth. The main idea was to be sure to understand the stakes of the assignment and hence, to be able to isolate each of the tasks involved. This step was individual and was followed by a group meeting to share our ideas and to assign the roles.

Timeline

The image below presents the timeline of our project work. It does not respect the time allocated to each task but is a representation of the succession of actions we have done during this assignment. Each milestone represents a group meeting using the Zoom platform.



Difficultes faced

Being far from each other and not having the possibility to all have a physical interaction with the hardware were the main difficulties of remotely working together. It implied that our work was more sequential than continuous: we all had to work on our subject and had to be ready to release the new version during the group meetings. Hence, everything had to be well prepared in order to avoid losing time. Each version (i.e. group meeting) had to represent a step forward in the assignment. The length of the step had to be well thought because meeting all together takes time and represents a loss of productivity, however, we had to meet regularly in order to spot issues and cut them before they turned into bigger issues.

We did well with those aspects since we were able to react quite fast to the different difficulties such as the choice of the cloud platform or the modifications of the hardware design.

Some main problems:

- No access to lab equipment for debugging of hardware, mainly no access to an oscilloscope
- Limited electronic components like resistors and capacitors, leads of potentiometers breaking etc.
- Shortage of electrodes as they are single use.

Our workarounds:

- The Arduino was used as a plotter to plot our signal to replace the oscilloscope
- Electrodes were reused, the conductive gel had gone off so we added some high salt content water to act as highly conductive liquid and used adhesive tape to strap them on.

Conclusion

In conclusion, successfully achieved in understanding various steps from electrode selection, cloud platform selection to processing, and publishing sensor data to the cloud storage. Firstly, we have effectively designed a suitable bio amplifier to capture the cardiac signal along with appropriate measures to reduce noise and interferences to increase the signal to noise ratio. Lastly, we have carried out the required signal processing such as sampling or digitizing before interfacing MSP432 to the ubidots cloud platform and was able to send cardiac signals to a cloud platform via MQTT protocol. In the overall design, we were also able to provide essential patient safety by an isolation circuit design.

References

[1] "Differential Amplifier - The Voltage Subtractor," *Basic Electronics Tutorials*, 22-Aug-2013. [Online]. Available: https://www.electronics-tutorials.ws/opamp/opamp_5.html. [Accessed: 23-Apr-2020]

[2] "ECG.pdf." [Online]. Available: http://yagizmungan.com/ECG.pdf. [Accessed: 23-Apr-2020]

[3] "EE445 Notes - CMRR Examples and Derivations.pdf." [Online]. Available: https://loop.dcu.ie/pluginfile.php/3062719/mod_resource/content/0/EE445%20Notes%20-%20CMRR%20Examples%20and%20Derivations.pdf. [Accessed: 23-Apr-2020]

[4] "Ubidots Basics: Devices, Variables, Dashboards, and Alerts." [Online]. Available: http://help.ubidots.com/en/articles/854333-ubidots-basics-devices-variables-dashboards-and-alerts. [Accessed: 23-Apr-2020]

Appendix

Source Code:

```
/*
 *  The following code uses Analog input A1 of the MSP432 to read the output from the AD620 instrumentation amplifier
 *  The data is sampled at approimately 33.33Hz and thresholding is done to detect peaks in the QRS complex of the ecg signal
 *  Each detected peak is couunted as a beat
 *  The detected beats are sent to the ubidots mqtt broker where a graph displays the heart rate of the person
 */

#include <SPI.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include<stdio.h>

// your network name also called SSID
char ssid[] = "BsDx-bWF6enk";
// your network password
char password[] = "mazzy0593";
#define TOKEN "BBFF-bE4HPwB2L1xLElHjmgXCqTCkIS9FE5"          // Put your Ubidots' TOKEN
#define MQTT_CLIENT_NAME "myecgsensor"                        // MQTT client Name, please enter your own 8-12 alphanumeric character ASCII string;
// MQTTServer to use
char server[] = "industrial.api.ubidots.com";

/****************************************
   Define Constants
 ****************************************/
#define VARIABLE_LABEL "sensor" // Assing the variable label
#define DEVICE_LABEL "esp32" // Assig the device label


#define SENSOR A1 // Set the A1 pin as ecg input to adc
char payload[100];//holds our data to be published
char topic[150];//topic data is published
// Space to store values to send
char str_sensor[10]; //buffer to append our data to be sent to server
unsigned long heartRateTime = 0;//timer for calculation of heart beat

WiFiClient wifiClient;//instance of class wifiClient
PubSubClient client(server, 1883, callback, wifiClient);//defines our mqtt connection
```

```
void callback(char* topic, byte* payload, unsigned int length) {//callback function when data
is received
  char p[length + 1];
  memcpy(p, payload, length);
  p[length] = NULL;
  Serial.write(payload, length);
  Serial.println(topic);
}

void setup()
{
  Serial.begin(115200);//enabling serial communication at 115200 baud rate
  pinMode(SENSOR, INPUT);//defines our ecg pin as input

  // attempt to connect to Wifi network:
  Serial.print("Attempting to connect to Network named: ");
  // print the network name (SSID);
  Serial.println(ssid);
  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  WiFi.begin(ssid, password); //connect to wifi router
  while ( WiFi.status() != WL_CONNECTED) {//waits till we're connected
    // print dots while we wait to connect
    Serial.print(".");
    delay(300);
  }

  Serial.println("\nYou're connected to the network");
  Serial.println("Waiting for an ip address");

  while (WiFi.localIP() == INADDR_NONE) {
    // print dots while we wait for an ip addresss
    Serial.print(".");
    delay(300);
  }

  Serial.println("\nIP Address obtained");
  // We are connected and have an IP address.
  // Print the WiFi status.
  printWifiStatus();//prints out IP address allocated and signal strength
}


void reconnect() {//manages reconnection/connection to server if we're disconnected
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.println("Attempting MQTT connection...");

    // Attemp to connect
    if (client.connect(MQTT_CLIENT_NAME, TOKEN, "")) {//sends MQTT connect message
      Serial.println("Connected");
    } else {
      Serial.print("Failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 2 seconds");
      // Wait 2 seconds before retrying
      delay(2000);
    }
  }
}

unsigned long lastConnTime = 0; // Track the last update time
```

```
bool TimeSpent(unsigned long prevTime, unsigned long value) {//returns true when the comparision between current time is greater that time to be checked
  bool returnVal = false;
  returnVal = ((millis() - prevTime) > value) ? true : false;//use of a ternery operator
  return returnVal;
}
float beats = 0;
void loop()
{
  //   Reconnect if the connection was lost
  if (!client.connected() && TimeSpent(lastConnTime, 5000)) { //check if we're connected, if not try to reconnect
    //timespent does a reconnection every 5000 if we're disconnected to take care that the reconnection is not done continously
    reconnect();
    lastConnTime = millis();//save last connection time
  }
  sprintf(topic, "%s%s", "/v1.6/devices/", DEVICE_LABEL);//forming data JSON
  sprintf(payload, "%s", ""); // Cleans the payload
  sprintf(payload, "{\"%s\":", VARIABLE_LABEL); // Adds the variable label
  float myecg = analogRead(SENSOR);//reading input from AD620 into our adc
  //   Serial.println(myecg);//plots out the ecg signal received on our adc pin

  if (myecg > 732) { //thresholding to measure a peak in the QRS complex
    beats += 1; //if peak detected, count it as a beat
    Serial.println("Peak");
  }
  if (TimeSpent(heartRateTime, 60000)) { //if 1 minute is passed, publish the value of heart beat and reset our variable to 0. This helps us get the BPM
    Serial.println("Publishing data to Ubidots Cloud");
    Serial.println(beats);
    /* 4 is mininum width, 2 is precision; float value is copied onto str_sensor*/
    dtostrf(beats, 4, 2, str_sensor);

    sprintf(payload, "%s {\"value\": %s}}", payload, str_sensor); // Adds heart beat data in our JSON payload to be sent to the server
    client.publish(topic, payload);//publish heart beat data to the server
    beats = 0;
    heartRateTime = millis();//saving time of last sent heart beat
  }
  delay(30);//sample at approximately every 30ms approx 33.33Hz sampling frequency
  //note this is approximate as the functions for maintaing server connection are non-blocking and take processor time
  //also, the delay function is not very accurate and tends to have a small drift, the timer can be used instead
}

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your WiFi IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.print(rssi);
  Serial.println(" dBm");
}
```