POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

Tesi di Laurea Magistrale

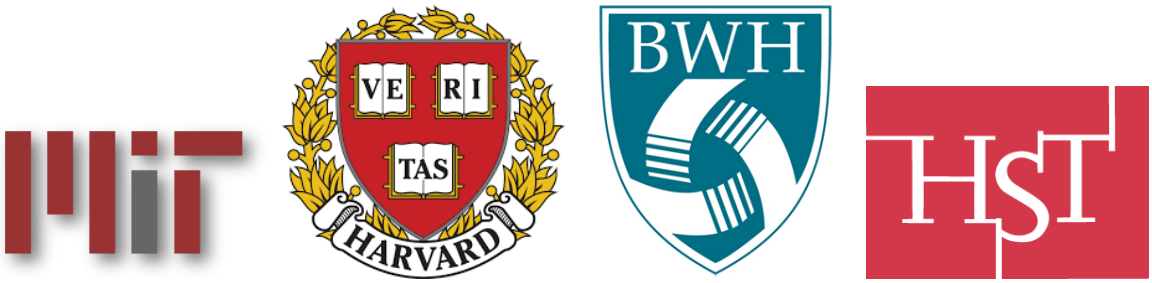
Google Glass Data Visualization and Monitoring for Organs-on-a-Chip and Biomedical Applications

Relatore
Prof. Danilo Demarchi

Candidato
Fabio Busignani s197883

Marzo 2015

ABSTRACT



The present scripture represent the master thesis of Fabio Busignani, and it has been carried out at *Khademhosseini Lab* (Cambdridge, MA, USA), Harvard-MIT Health Science and Technology, Brigham and Women's Hospital under the supervision of professor Ali Khademhosseini and Ph.D. Yu Shrike Zhang.
The design which is going to be described, has been inserted inside the context of a five years project (***XCEL*** grant), sponsored by the U.S. Defense Threat Reduction Agency (*DTRA*).

The aim of *XCEL* is to develop a *Body-On-A-Chip* microfluidic platform that is able to simulate multi-tissue interactions under physiological fluid flow conditions.

In this master thesis will focus on designing of a custom user interface on *Google Glass* for simultaneous recording of biosensing data such as temperature, pH, and microscopy images/videos as well as remote control of microfluidic valves and devices. The project involves all the hierarchical layers, starting from the physical one with the circuit in charge to acquire data from bio-sensors and drive the valves, up to the glasswear[1].

In the Introduction chapter, the main keys of the project are presented in detail as well as the final result from a user point of view.
After that a detailed description of each abstraction level which goes to build the entire systems is shown: starting from the bottom (Hardware) reaching the top (Google Glass Application) passing through the Firmware, that runs in an embedded *Linux* platform, and the Software, present on the *PC* and the *Google App Engine*.
The Experiments and Conclusion chapter ends this thesis, showing the obtained results with different experiments.

[1] Google Glass Application

CONTENTS

Abstract I

INTRODUCTION 1

The Glasswear . . . 3

The Board . . . 5

Video Storing . . . 6

i HARDWARE 9

1 CONDITIONING CIRCUIT AND ELECTROVALVES DRIVERS 11

1.1 The Sensors . . . 11

1.1.1 PH Sensor . . . 11

1.1.2 Temperature Sensor . . . 12

1.2 PH Conditioning . . . 14

1.3 Temperature Conditioning . . . 19

1.4 Electrovalves Driver . . . 20

1.5 DC-DC Converter . . . 21

1.5.1 Components Choice . . . 22

1.5.2 Relay . . . 24

2 PCB 26

ii FIRMWARE 31

3 INTRODUCTION 32

3.1 A Brief Introduction To IoT . . . 33

3.2 The Beaglebone Black . . . 34

4 EMBEDDED LINUX INSIDE THE PROJECT 36

4.1 Video Processing and Beating Plot . . . 38

4.2 Sensor Data Acquisition . . . 38

4.3 Electrovalves Updating Task . . . 39

iii SOFTWARE 41

5 GOOGLE APP ENGINE 43

5.1 The web application . . . 44

6 MICROSCOPE VIDEO STORING 47

iv GOOGLE GLASS APPLICATION 48

7 INTRODUCTION 49

v CONCLUSION AND APPENDIX 50

8 TEST AND PERFORMANCE 51

8.1 LED Experiments . . . 51

8.2 Electrovalves experiments . . . 52

8.2.1 Breadboard Phase . . . 52

8.2.2 PCB Phase 54
A CODE 55
A.1 Firmware 55
A.2 Video Storing Software 66
A.3 Google App Engine 71
A.4 Glassware 77
List of Figures 122
List of Listings 123
Bibliography 124

INTRODUCTION

This master thesis has been carried out at Khademhosseini laboratory, Harvard-MIT Health Science and Technology (Brigham and Women's Hospital), in Cambridge, MA. During my six-months of research I have joined ***XCEL*** grant project, a five years project sponsored by the U.S. Defense Threat Reduction Agency (*DTRA*).

The goal of this project is to develop a system, a microscale bioreactor containg four 3D fully-functional *organs-on-a-chip*.

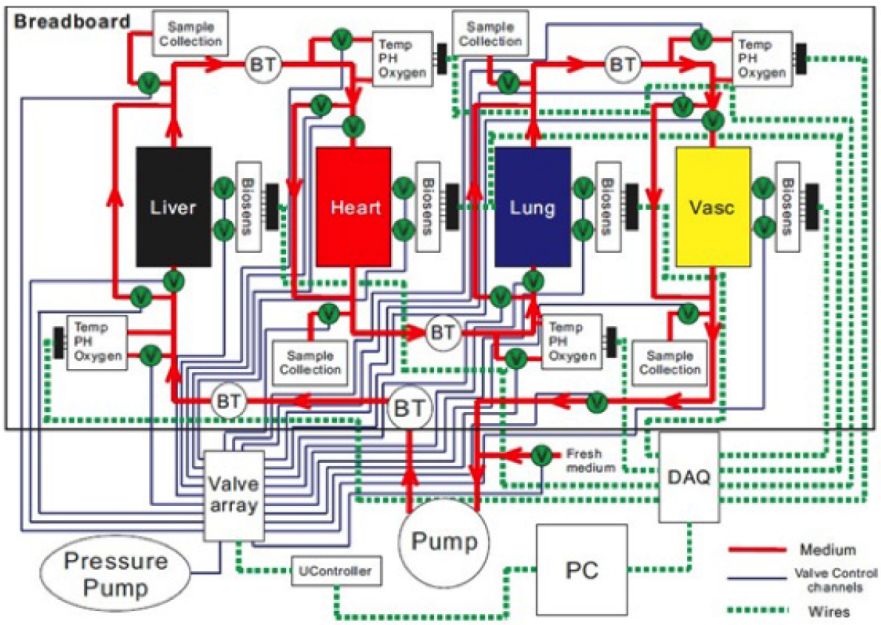


Figure 1: *XCEL* project (*Body-on-a-chip*)

The (Fig.1) shows, in a schematic representation, the design of the entire *XCEL* project. On the breadboard four organs are connected from each others: liver, heart, ling, and vascular system (*Vasc*). The medium used to connect them is using tubing circuit where the media flows. This tubing connections are driven by electrovalves.

My role in this project has been to create a custom user interface on Google Glass for simultaneous recording of biosensing data such as temperature, pH, and microscopy images/videos as well as remote control of the microfluidic valves previously introduced. In summary my aim was to design a Google Glass App for use in organs-on-a-chip platforms.

The *organs-on-a-chip* platforms contain interconnected microfluidic modular components including the bioreactors for hosting biomimicry human organ models, downstream biochemical sensors to continually monitor the levels of biomarkers secreted by the organs, and physical sensors to monitor the physical microenvironment of the circulatory system. Due to their extensive similarity with human organs, these miniature human models are finding widespread applications where the prediction of in vivo responses of

the human body is needed, including but not limited to drug screening, basic biomedical studies, and environmental safety assessment. Thus, the *organs-on-a-chip* platforms seek to recapitulate human organ function at micro-scale by integrating microfluidic networks with three-dimensional organ models, which are expected to provide robust and accurate predictions of drug/toxin effects in human bodies. In fulfilling this aim, a set of physical/chemical parameters need to be monitored and stored in order to capture such effects of drug/toxin administered into the system.

By precisely designing the Google Glass App for this organs-on-a-chip platform it allows convenient observation and control of the organ models, biosensors, and the microfluidic circuitry, which has been difficult to achieve previously.

The system designed and described in this thesis is illustrated in (Fig.2).

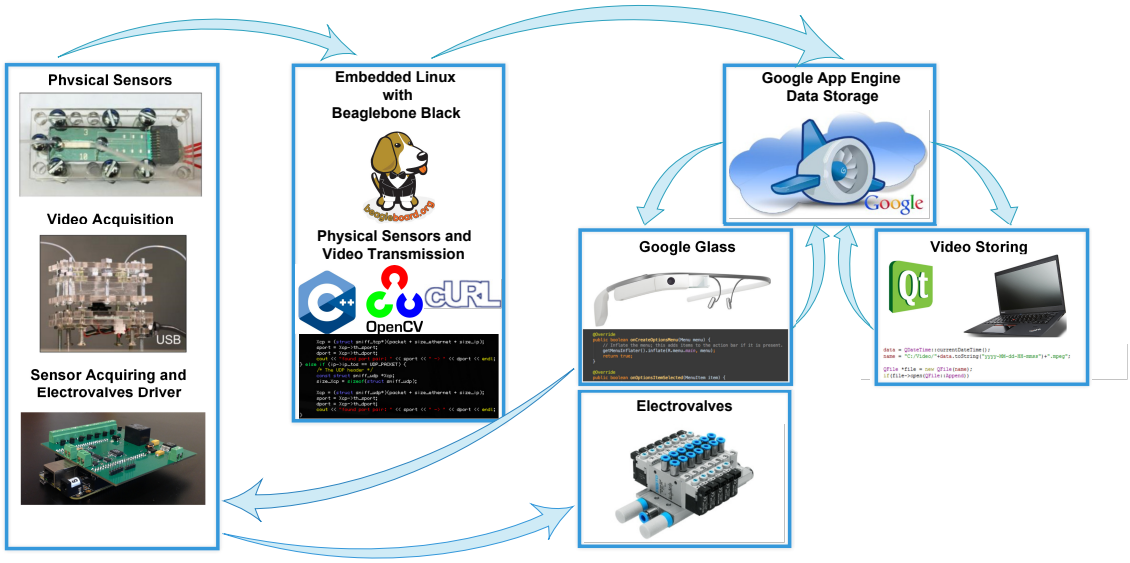


Figure 2: Block diagram of the system

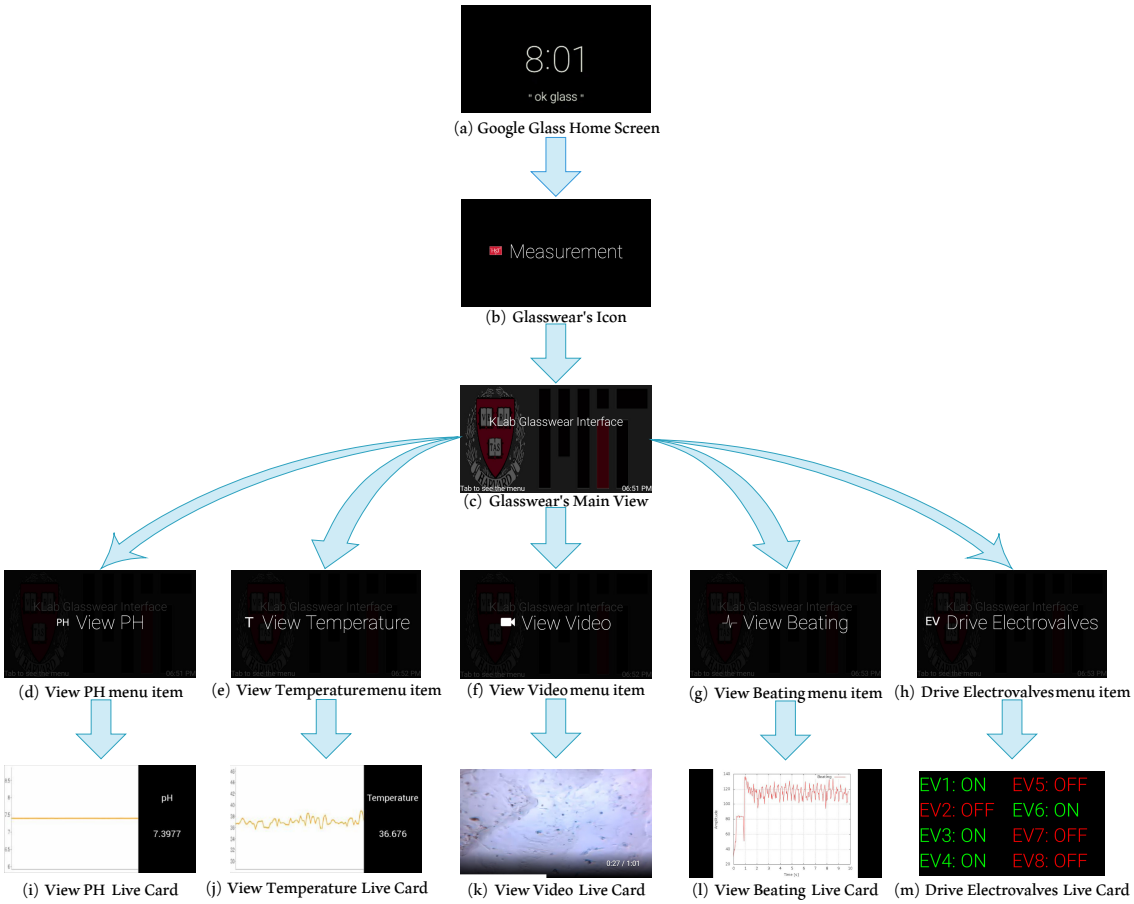
The (Fig.2) shows the principal steps of data transmission from physical and video sensors to the Google Glass via an *Embedded Linux System* performed using the **Beaglebone Black**.
The Beaglebone Black runs processes that are in charged to:

- acquire the sensors value and to store them onto *Google App Engine Data Storage*;
- acquire the video, perform the beating plot, and to store them onto *Google App Engine Data Storage*;
- get from the *Google App Engine Data Storage* the electrovalves status set from the user through the Google Glass and to drive the electrovalves.

The whole designed environment includes a program, written using the framework *Qt*, for storing the recorded video from microscope.

THE GLASSWEAR

The (Fig.3) shows the structure of the Glasswear. From the Home Screen (Fig.3a), using the voice trigger "*Show Measurement*" or tapping on the "*Measurement*" card (Fig.3b) user is allowed to enter in the application (Fig.3c). From this point, tapping and swiping, it's possible to navigate into the glasswear's menu (Fig.3d-h) and choose which card has to be shown. *View PH* (Fig.3i) and *View Temperature* (Fig.3j) cards plot on the card's left side the value of pH and temperature, respectively. While on the right side they show the average value. The microscope's video is shown by tapping on *View Video* (Fig.3k). The *View Beating* card shows the graph of the beating associated to the video.



(a) Google Glass Home Screen
(b) Glasswear's Icon
(c) Glasswear's Main View
(d) View PH menu item
(e) View Temperature menu item
(f) View Video menu item
(g) View Beating menu item
(h) Drive Electrovalves menu item
(i) View PH Live Card
(j) View Temperature Live Card
(k) View Video Live Card
(l) View Beating Live Card
(m) Drive Electrovalves Live Card

Figure 3: Glasswear's Block Diagram

From the *Drive Electrovalves* card (Fig.3m), the user can set the value of each electrovalve. The main view of this card shows the status of each electrovalve (written in green if it is on and in red if it is off).

(a) Elecrovalves Status View (b) Toggle EV1 menu item (c) Toggled EV1

Figure 4: Drive Electrovalves Steps

The (Fig.4) shows the steps to toggle the status of the first electrovalve:

1. (Fig.4a) shows the initial status of the whole electrovalves (all off);

2. tapping on the card and swiping the user is allowed to change the status of each electrovalve from the menu, as shown in (Fig.4b);

3. after that the electrovalve has been chosen, a toast message pops up (Fig.4c), and the new values of the electrovalves are shown.

To return on the main card of the glassware, the user has to tab on *Back* item (Fig.5) from every menu.

Figure 5: *Back* menu item

To terminate the glassware, from every menu, the user has to swipe up to the final item and tab on *Exit* item (Fig.6).

Figure 6: *Exit* menu item

THE BOARD

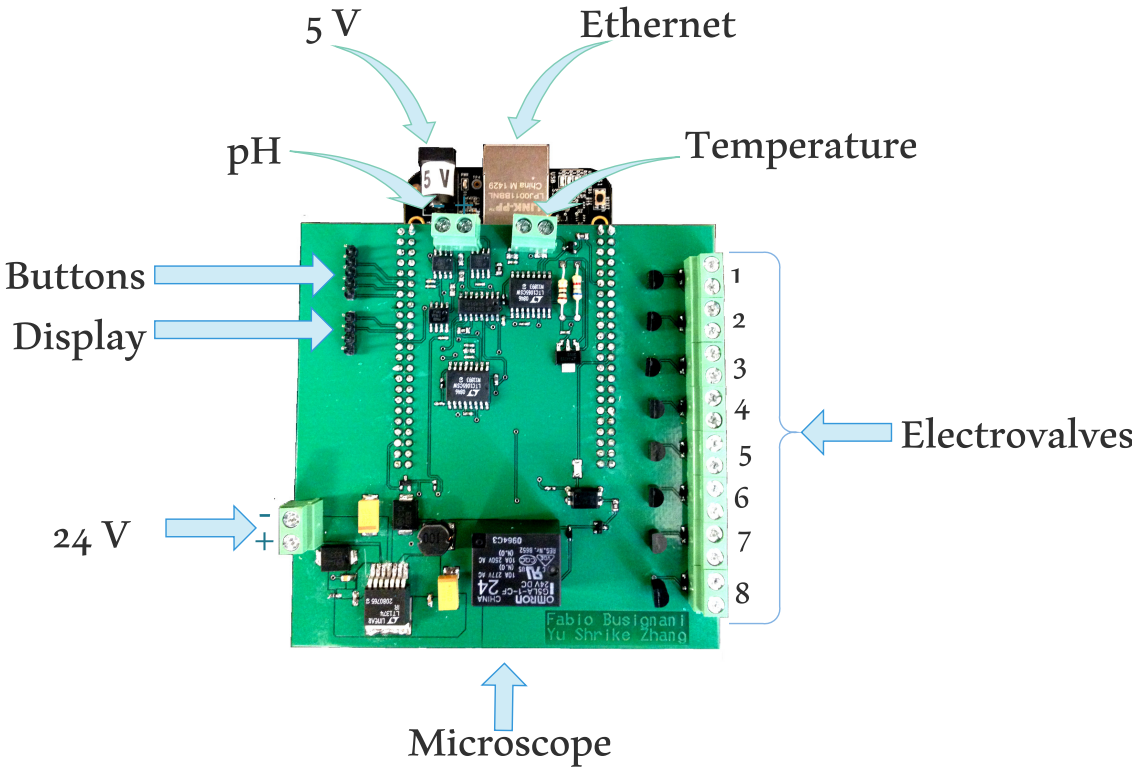


Figure 7: The Board

The (Fig.7) shows the top view of the system board. As can be seen it is composed by different interface and connections:

- $5\ V$ power supply, required by the microcomputer on the Baglebone Black and by the conditiong circuits on the *PCB*;
- $24\ V$ power supply, required by the electrovalves;
- *Ethernet*, to connect the board to the Internet;
- *USB* connection, for the microscope;
- *pH header*, to connect the pH sensor[2];
- *temperature header*, to connect the temperature sensor;
- *electrovalves header*, made by eight pairs of terminals, ordered as shown in (Fig.7), from the top to the bottom.

The remaining two headers, shown in the top left corner of (Fig.7) are thought for future application. In particular they are going to be useful for all those jobs that don't required the interaction with Google Glass, such as sensors calibration.

[2] **WARNING**: to ensure the correct functionality, user has to pay attention at this connection, since the pH sensor is a passive one, it has a polarity. The positive pin of the sensor has to be connected to the right terminal, looking fro the top (as shown in (Fig.7))

VIDEO STORING

The storing of the microscope video plays an important role of this system. It may be essential to review the recorded video during the experiment and in order to fulfill this aim a *Qt* program has been designed.
We chose *Qt* because in this way the program is available for different operating systems, *Linux*, *Windows*, and *MacOS*.

The program is very easy to use, the user just has to run the executable (Fig.8).

Figure 8: Storing Microscope Video Program's Icon

Once it has been launched, a console is opened (Fig.9). The program checks every 20 seconds if a new video has been uploaded on the server. If so, the new video will be stored inside the computer (directory `C:/Video`) with the current data and hour as name in the following form: $YYYY - MM - DD - HH - mmss$, as shown in (Fig.10).

As shown in (Fig.9) on the console the user can read all the information about what the program is doing.

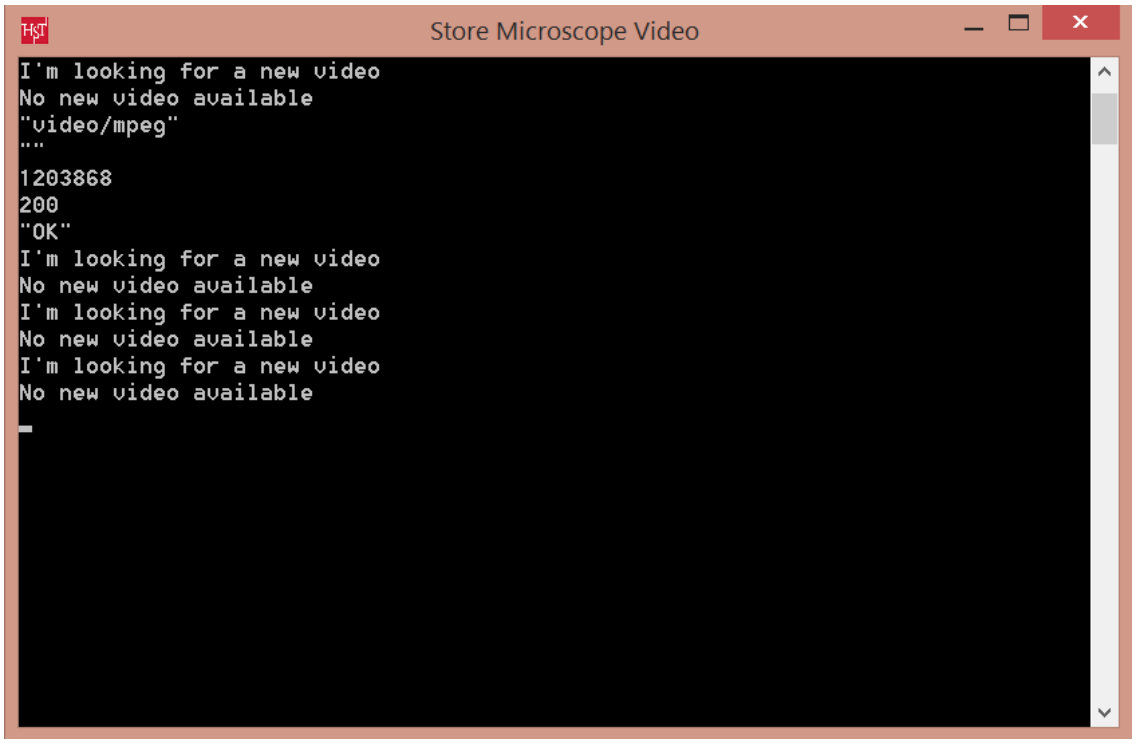


Figure 9: Storing Microscope Video Console

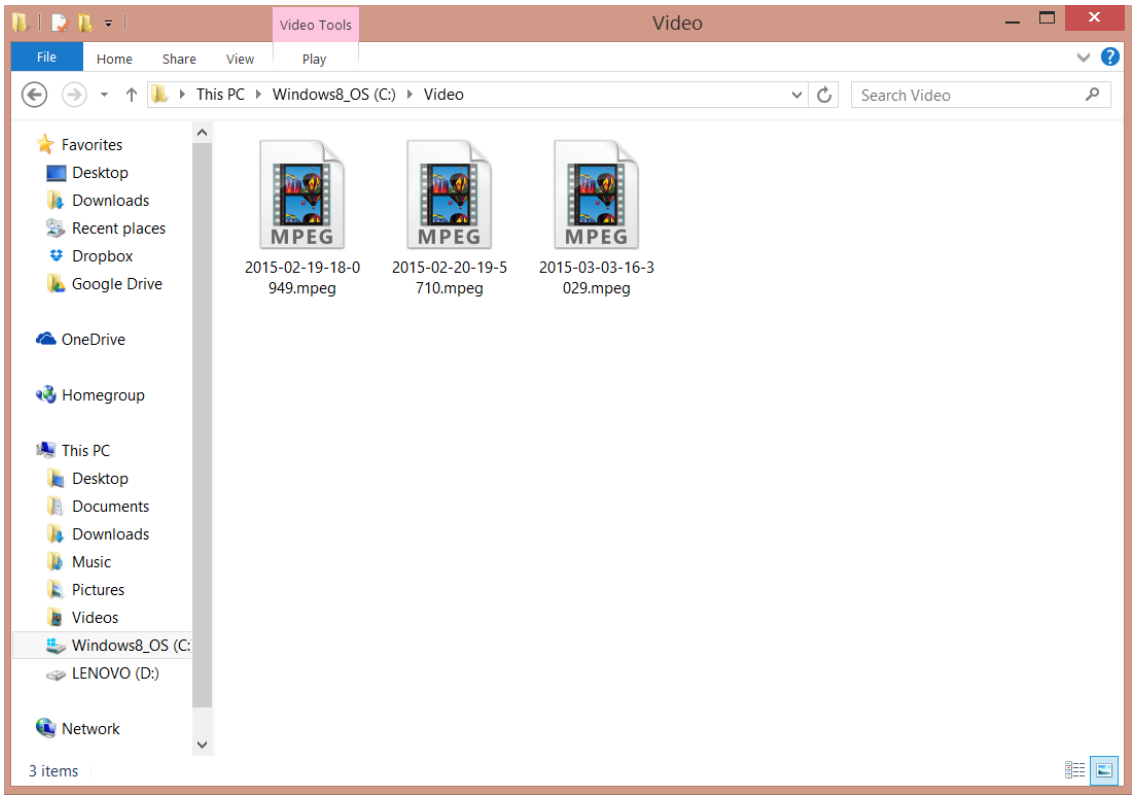


Figure 10: Video Stored in the Folder

Part I

HARDWARE

In this part of the thesis the hardware that has been designed in this system is explained. The design specification for this part are:

- make a sensor conditioning for pH and temperature sensors, see (Sec.1.2) and (Sec.1.3) for more details;
- make a driver for electrovalves which must be able to drives two different kinds of electrovalves, both of them require $80\ mA$ but one type at $24\ V$ while the other one at $12\ V$. To fulfill this aim a *DC-DC* converter has been designed, as well. See (Sec.1.4) for more details;
- make a *PCB* which supports and connects all the previous components and that is a *capes* for the Beaglebone Black, it has to be wedged on top of it, see (Chap.2) for more details.

1 CONDITIONING CIRCUIT AND ELECTROVALVES DRIVERS

1.1 THE SENSORS

The sensors used in this project are microfabricated on a single silicon chip, and kindly provided by professor Dr. Sandro Carrara research group from École Polytechnique Fédérale de Lausanne (*EPFL*), Switzerland.

The choice of using a microfabricated sensor instead of a commercial one (the number of commercial sensors is very high even for biomedical applications) has been taken because of a remarkably decrease in sensing occupied space. Indeed, in this way, the area consumption can be optimized.

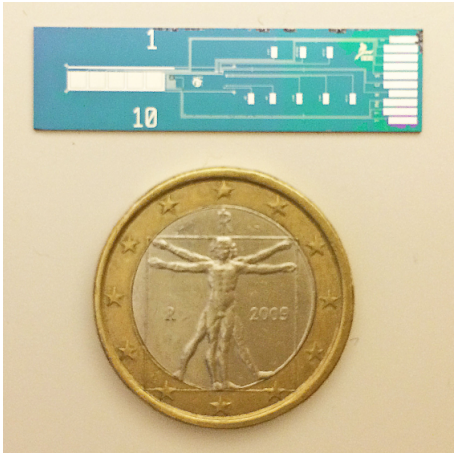


Figure 11: Multisensor - Size Comparison

The (Fig.11) shows the sensor and highlights its very small dimension ($35.1\ x\ 9.3\ mm$). This multisensor contains an Iridium Oxide based pH sensor and a platinum resistance temperature detector (*RTD*) [1].
The device also hosts five independent working electrodes (*WE*), with shared platinum reference electrode (*RE*) and platinum counter electrode (*CE*). But, at least for the time being, we are not going to use them.

The (Fig.12) shows the circuit schematic of multisensor. As can be seen, from the interface the pads that are going to be used in this thesis are the four on the bottom. In fact, the last pair of pins are the temperature ones, and the penultimate ones are tied to the pH sensor (negative pin on top).

1.1.1 PH Sensor

The embedded pH sensor has been made by platinum. Its electrical output is a voltage value which varies almost linearly with the pH.

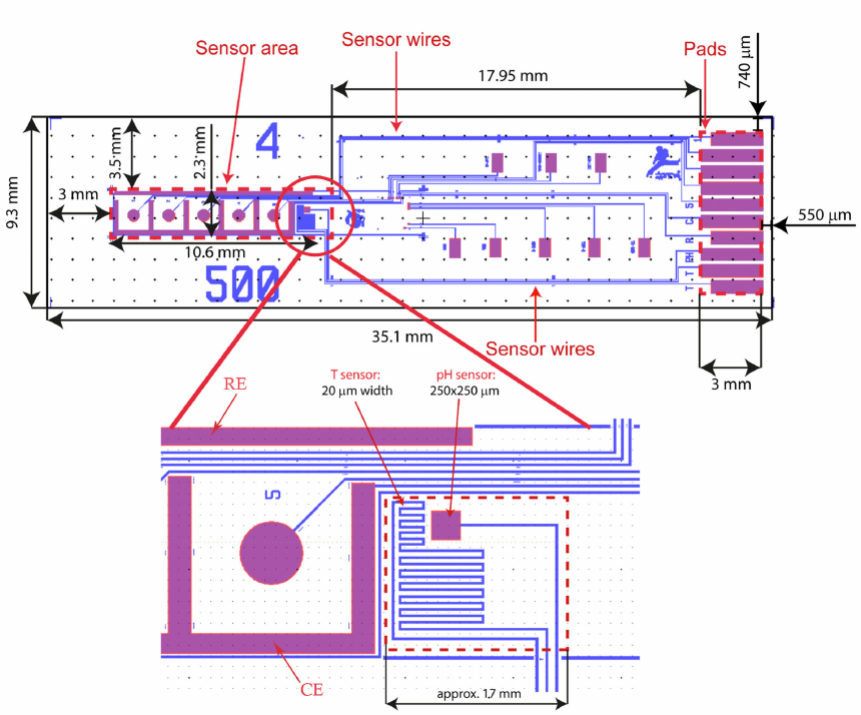


Figure 12: Multisensor - Scheme

The pH sensor needs to be calibrated before it can be used. The calibration and characterization of sensor are made in different environments and with different solutions. After this step the transfer function of the sensor is linear (Fig.14) with a slope of around $-100\ mV/pH$.

1.1.2 Temperature Sensor

For almost all the metal materials, electrical resistance is a parameter which varies with temperature. In order to create a resistance temperature detector (*RTD*), a coupled system of a metal wire and a resistance measurement device is needed. Inside the used device, the temperature sensor is made by a platinum wire. This metal is strongly used for temperature applications because it allows to obtain the most accurate measurements (up to $\pm 0.001\ ^{\circ}C$) with a linear output response.

The relationship between resistance and temperature of a platinum resistance thermometers is described by the ***Callendar-Van Dusen*** equation (Eq.1).

$$R(T) = R(0) \cdot [1 + A \cdot T + B \cdot T^2 + (T - 100) \cdot C \cdot T^3] \tag{1}$$

Where:

- $R(T)$, is the resistance at the T temperature;
- $R(0)$, is the resistance value at $0\ ^{\circ}C$;

- A, B, and C, are the *Callendar-Van Dusen constants* defined by the following:

$$A = \alpha + \frac{\alpha \cdot \delta}{100}, \tag{2a}$$

$$B = -\frac{\alpha \cdot \delta}{100^2}, \tag{2b}$$

$$C = -\frac{\alpha \cdot \beta}{100^4}. \tag{2c}$$

And $\alpha = 0.003921 \Omega/°C$, $\beta = 0$ for positive temperature and $\delta = 1.49$.

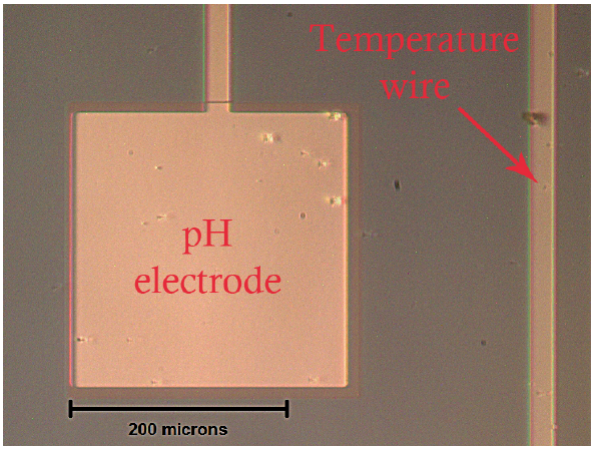


Figure 13: PH and Temperature Sensors from an Optical Image

The pH and temperature sensors are very close from each others (Fig.13), and this is very important because the pH electrode's sensitivity varies over temperature, thus: temperature affects the pH value, as can be seen from (Eq.3). So, we always need to know at what temperature we make the pH measure.

$$pH\,(X) = pH\,(S) + \frac{(E_S - E_X) \cdot F}{R \cdot T \cdot \ln(10)} \tag{3}$$

The (Eq.3) represents the transfer function of a pH sensor, where:

- $pH\,(X)$, is the pH value of unknown solution;
- $pH\,(S)$, is the pH value of standard solution (7);
- E_S, is the electric potential at standard electrode;
- E_X, is the electric potential at pH-measuring electrode;
- F, is the Faraday constant ($9.6485309 \cdot 10^4\ C\ mol^{-1}$);
- R, is the universal gas constant ($8.314510\ J\ K^{-1}\ mol^{-1}$);
- T, is the temperature in Kelvin.

1.2 PH CONDITIONING

As already explained in (Sec.1.1.1), the pH sensor is a passive sensor, which means no excitation source is required because the sensor itself generates its own electrical output signal. In particular any variation of pH in input is transduced in a voltage variation in output.

The pH sensor is also bipolar, this means the voltage output may be both positive and negative, as shown in (Fig.14).

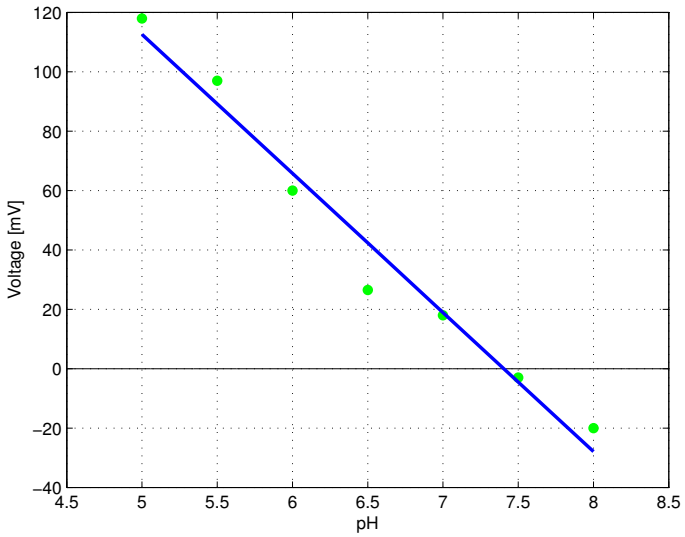


Figure 14: Typical pH-sensor transfer function

Resuming, it produce a voltage output that decreases linearly with pH of the solution being measured. The sensors give a sensitivity which ranges between 50 and $120\, mV/pH$ (it depends from sensor to sensor), this means that, in order to well observe this variation, an amplification stage may be required.

The (Fig.15) shows the adopted solution for conditioning the pH sensor.
Fist of all, since the pH sensor produces a bipolar signal and this application operates on a single voltage supply, the signal has been level shifted. To achieve this first challenge the operation amplifier *U1* forces an off-set of $512\ mV$ to the pH sensor. Indeed, the *LM4140A-1.0* is a high precision low noise *LDO* (Low Drop Out) voltage reference which provides an accurate $1.024\, V$. This voltage has been halved by the $10\ K\Omega$ resistor divider. The *U1* is in voltage follower configuration, thus its output should be equal to the input, and it biases the reference electrode of the pH sensor with $512\ mV$, at low impedance.
So, what the part of circuit made by *U1* and *LM4140A-1.0* does is to shift the bipolar pH sensor signal to an unipolar in order to be usable in the single-supply system.

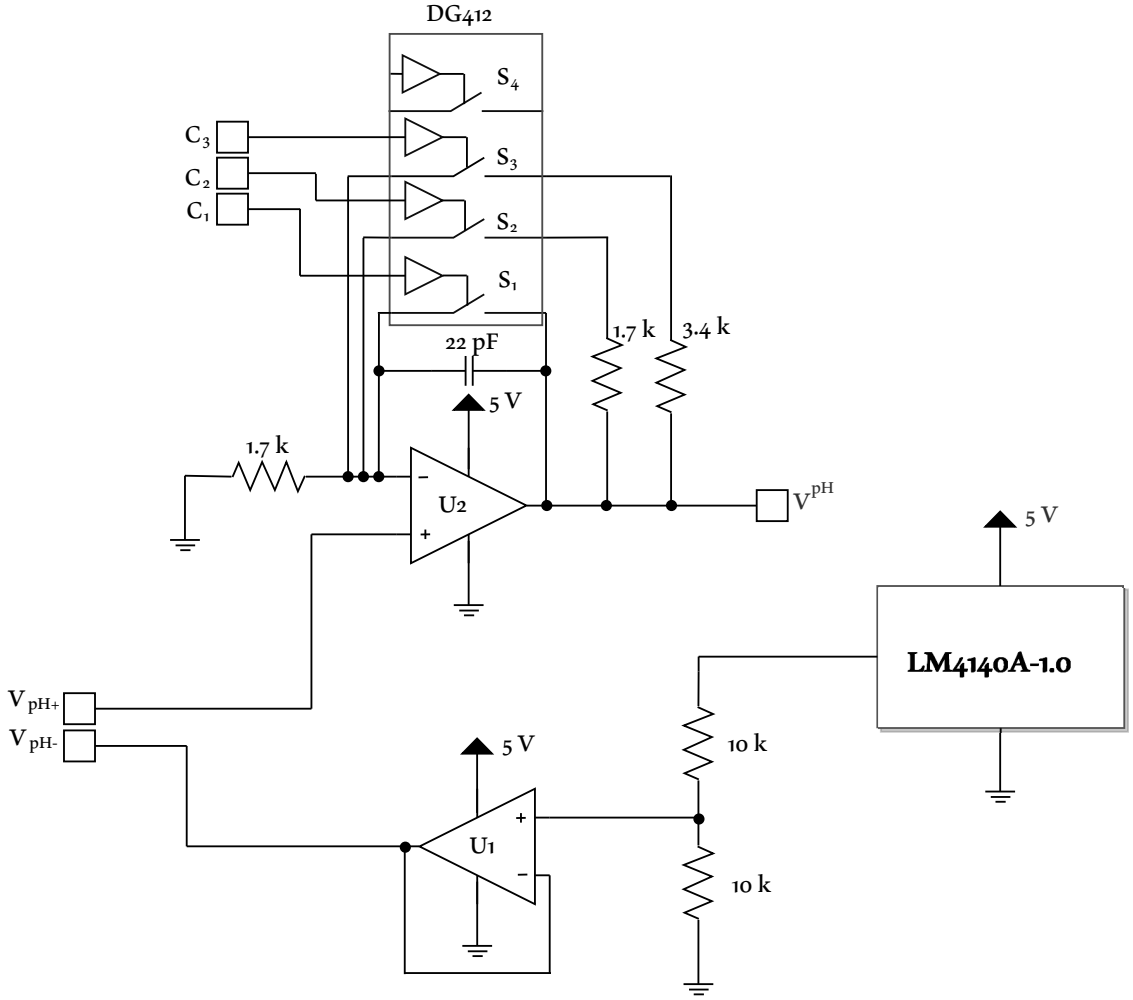


Figure 15: Conditioning circuit for pH sensor

Another challenge is given by the high impedance of the electrode. In fact the output impedance of the pH sensor is higher than $100\ M\Omega$. The circuit in (Fig.16) shows a typical connection of this sensor where the output voltage is given by:

$$V_{out} \simeq V_{in} = V_S - I_{bias} \cdot R_S \tag{4}$$

Thus, in order to reduce the error caused due to amplifier's input bias current a really low input bias current amplifier has to be chosen. For this reason, the *LMP7721* is used, it is has an ultra-low input bias current ($3 \pm 17\ fA$).

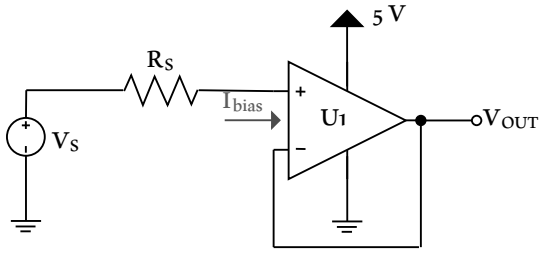


Figure 16: Error caused by Amplifier's Input Bias Current

In (Fig.15) both *U1* an *U2* are *LMP7721*. The second amplifier with the *DG412* represents a simple **PGA** (*Programmable Gain Amplifier*), where the resistors have been chosen to give the following gains:

- 1, when C_1 is asserted and the others are denied;
- 2, when C_2 is asserted and the others are denied;
- 4, when C_3 is asserted and the others are denied.

The feedback capacitor is used to ensure stability and holds the output voltage during the switching times. Indeed, in these slice of time the output node would be floated without the capacitor.
This PGA stage introduces an additional offset error of $75 \pm 470\ fV$, due to the bias current of the operational amplifier and R_{ON} of *DG412* ($25\ \Omega$). That voltage combined with the *LMP7721* offset (which is very higher than the first one) is approximately equal to $26\ \mu V$.

Since the environment in which the circuit is going to be used is a laboratory, so a really noisy place, the conditioning circuit for the sensor has to involve the design of a ***low-pass filter*** in order to reject the noise.

The circuit shown in (Fig.17) is a third order low-pass filter with a *Bessel* response. It has been designed in order to ensure a really flat-response in the pass band.
The parameter of the filter are:

1. *cutoff frequency* (f_c): $26.8\ Hz$, ;
2. *stop band attenuation*: $-28.2\ dB$ at *stop band frequency* (f_s) $60\ Hz$ (the line frequency in USA);
3. *Quality factor* (Q): 0.65;
4. *filter order*: third;
5. *filter response*: Bessel.

It's important that $Q < 0.707$ because otherwise would be some peaking in the filter response. While, in this case, as shown in (Fig.18), roll-off at the cutoff frequency is greater.
This filter also behaves as *anti-aliasing filter*, to prevent the aliasing components from being sampled during the analog to digital conversion.

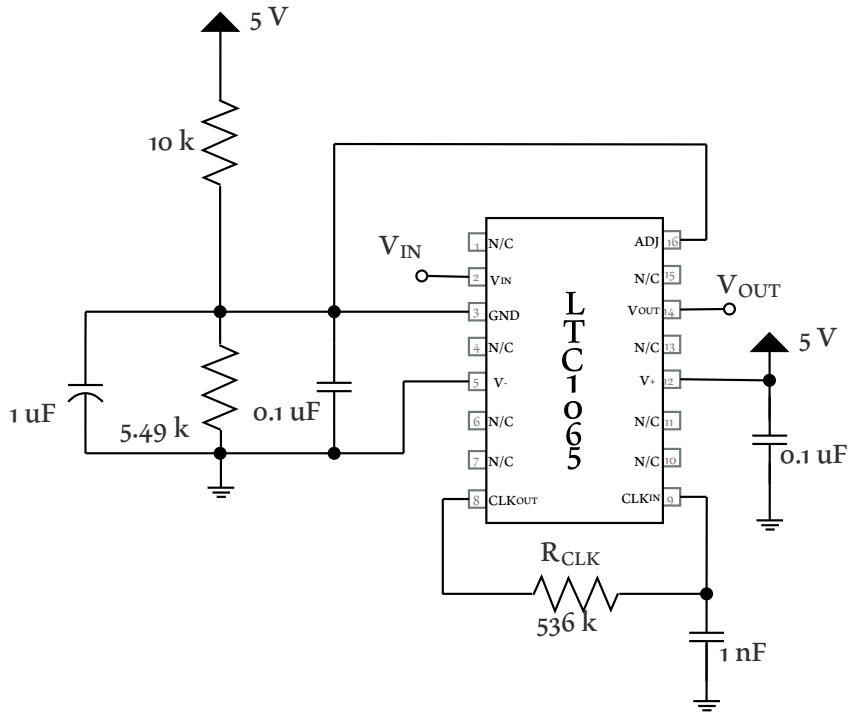


Figure 17: Low-Pass Filter Schematic

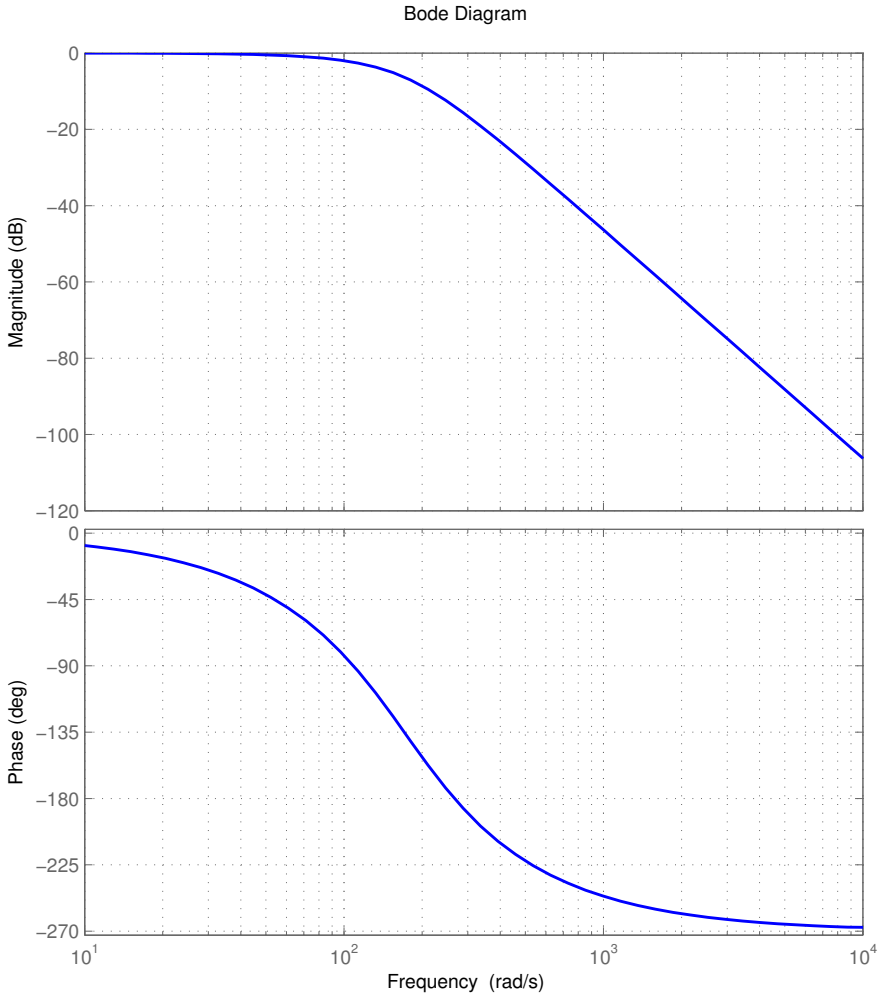


Figure 18: Low-Pass Filter Frequency Response

The output of this filter represent the input signal of the embedded ADC inside the Beaglebone Black.

Connecting together al this part we obtain the circuit in (Fig.19) where we can also see the general purpose input/output pin used to drive the *PGA* and the analog input used for the pH sensor. As it is explained in (Chap.4) `AN_2` is used to let the microcontroller know about the added offset.

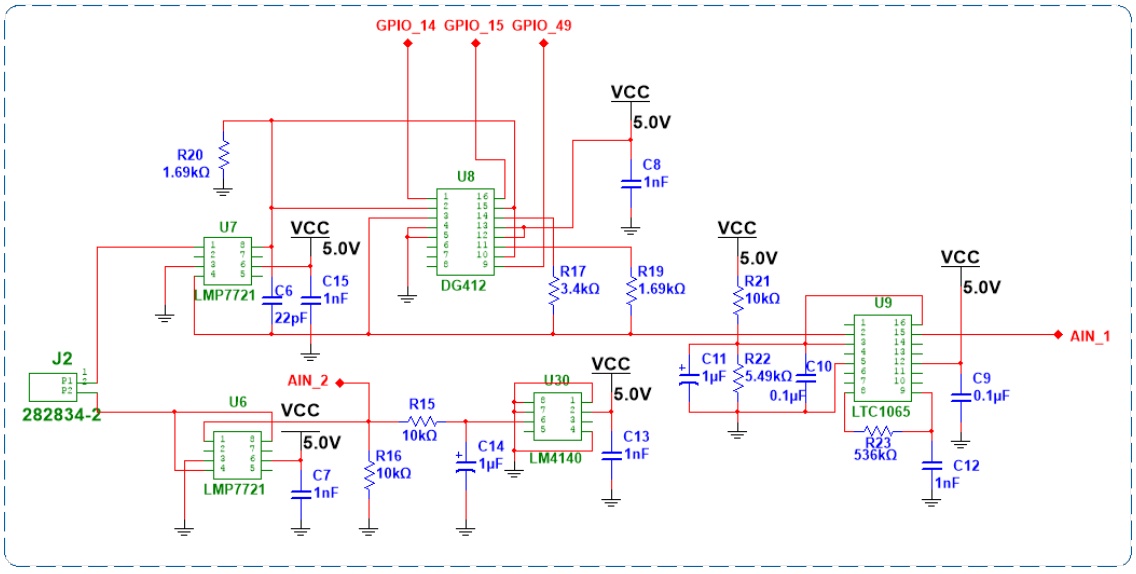


Figure 19: Acquisition Path for pH Sensor

1.3 TEMPERATURE CONDITIONING

As already explained in (Sec.1.1.2), the temperature sensor is a active sensor, which means excitation source is required because the sensor is resistor based so a current must be passed through it. Then, the corresponding voltage has to be measured in order to determine the temperature value.
So, any variation of temperature in input is transduced in a resistance variation in output.

The (Fig.20) shows the adopted solution for conditioning the temperature sensor.
In this connection the temperature sensor is excited by $680\ \mu A$, so the voltage V_T is given by the following equation:

$$V_T = 680\mu \cdot R_T \tag{5}$$

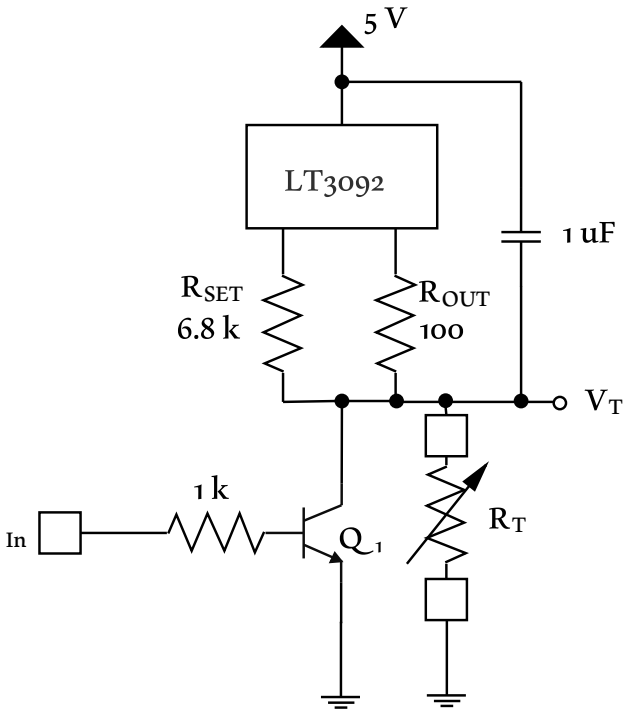


Figure 20: Conditioning Circuit for Temperature Sensor

In order to provide this amount of current a *LT3092* is used. It can supply an output current equal to:

$$I_{OUT} = 10\mu \cdot \frac{R_{SET}}{R_{OUT}} \tag{6}$$

To ensure the stability of the component, a feedback capacitor of $1\ \mu F$ is exploited. The transistor Q_1 is used to avoid the self-heating of the temperature sensor: when the temperature value has to be sampled In is denied, for the remaining time In is asserted, in this way the resistive sensor is by-passed, and the *Joule* effect is avoided.

For the same reason exposed in (Sec.1.2), also in this case a filter is needed, and, since the voltage value is almost the same, the used filter is equivalent to the previous on.

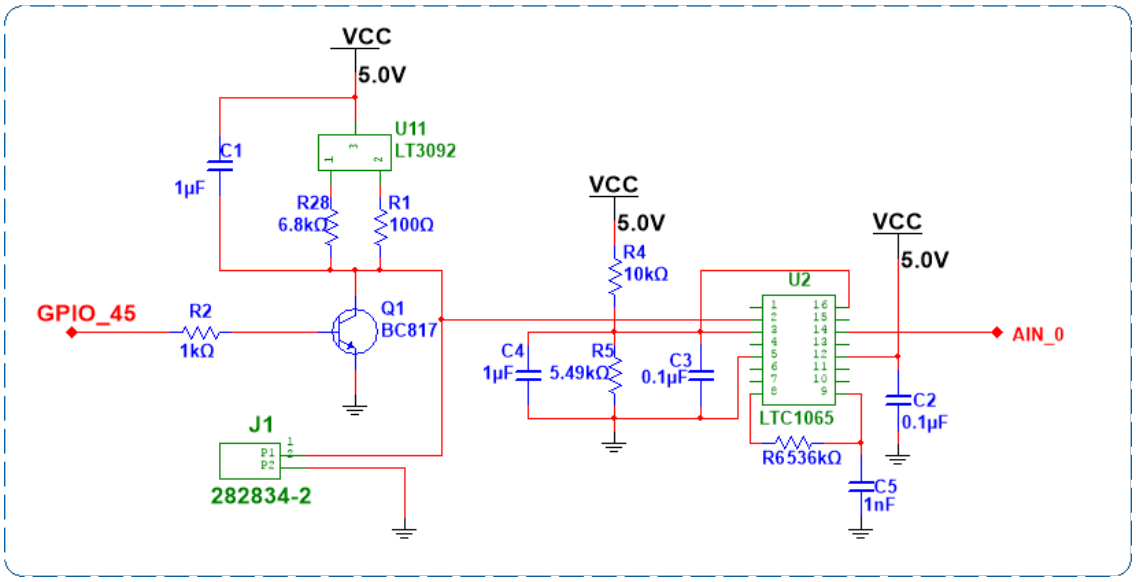


Figure 21: Acquisition Path for pH Sensor

1.4 ELECTROVALVES DRIVER

In order to allow the reverse control, from Google Glass to electrovalves, it is important to design a circuit that has to drive them from digital values provided by the *Beaglebone Black* (0 equal to 0 V, 1 equal to 3.3 V and in both cases the maximum suppliable current is only 6 mA).

The electrovalves used are FESTO solenoid valves MH1. They require a voltage of 24 V in DC and a current of 80 mA.

To fulfill this aim a low-side switch MOS has been used, as shown in (Fig.23).

The *Fairchild Semiconductor BS170 N-Chnnel MOS* has been chosen because of its low price and its capability of supporting a drain-source voltage of up to 60 V and supplying a continuous drain current of up to 500 mA.

To protect the *BS170* from reverse inductive current surges due to the solenoid of the electrovalve, a *Vishay Semiconductors 1N4148 Diode* is used. It is able to support a reverse voltage of up to 75 V and a continuous forward current of up to 150 mA.

As shown in (Fig.22) the number of electrovalves used is eight, so the previous driver has been replicated in order to obtain the circuit in (Fig.)

Figure 22: Electrovalves

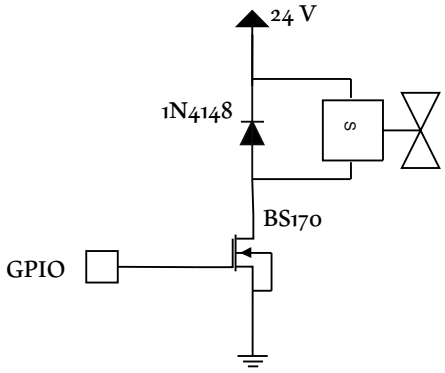


Figure 23: Driver for electrovalves

1.5 DC-DC CONVERTER

Since this system is supposed to drive different kind of electrovalves, which require a different value of voltage (but not in current) a *DC-DC converter* is used in order to convert the voltage supply from $24\ V$ to $12\ V$.

Using the *LT1374* the circuit in (Fig.25) has been designed, it is a constant frequency (equal to $500\ Hz$), current mode buck converter. It has an embedded clock and two feedback loops to control the duty cycle of power switch.

Trough a low-side switch, made with the *BS170*, it is possible to shutdown the converter from the software. When the *LT1374* is in shutdown, the supply current is reduced to $20\ \mu A$. As it is explained in (Chap.4) this is used to prevent regulator from operating when $24\ V$ are required.

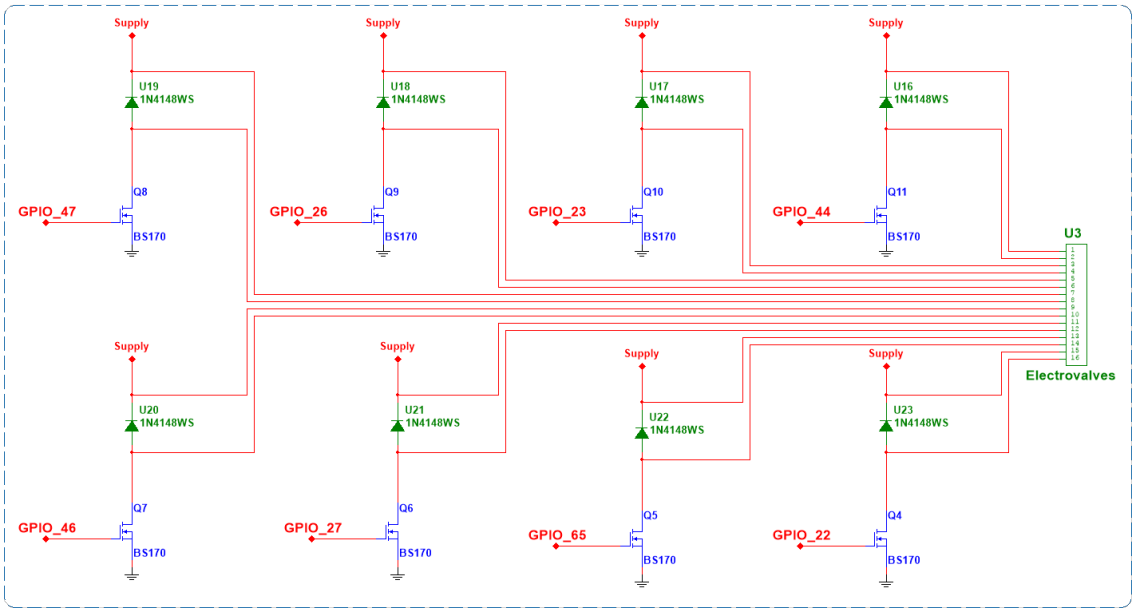


Figure 24: Electrovalves Driver Circuit

1.5.1 Components Choice

All the components have been chosen with attention and for some reasons that are going to be explained belong.

FEEDBACK RESISTORS

The main behavior of the feedback pin on the *LT1374* is to set the output voltage, and this deals with selecting the resistors R_1 and R_2. They are related from each others by the following equation:

$$R_1 \quad \frac{R_2 \cdot (V_{OUT} - 2.42)}{2.42} \tag{7}$$

As suggested on datasheet of the component, the resistor between feedback pin and ground is $4.99\ k\Omega$. So, from the (Eq.7) results that the value of R_1 is $19.6\ k\Omega$.

INDUCTOR

The choice of inductor is a trade-off among:

- *physical area*, lower values of inductor mean lower size;
- *output current*, higher values of inductor allow more output current because they reduce peak current ($I_{SW(PEAK)} \propto 1/L$)
- *ripple voltage*, higher values of inductor reduce the output ripple voltage.

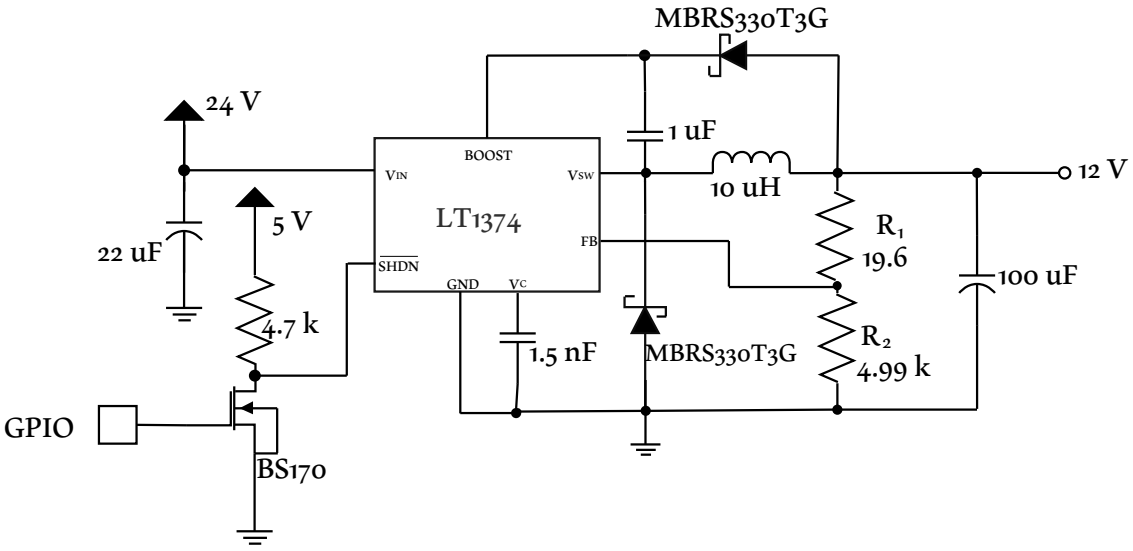


Figure 25: DC-DC circuit

A good choice is represented by $10\ \mu H$, with this inductor the maximum current peak (Eq.8) is equal to $1.24\ A$.

$$I_{SW(PEAK)} = I_{OUT}\ \frac{V_{OUT}\cdot(V_{IN}-V_{OUT})}{2\cdot f\cdot L\cdot V_{IN}} \tag{8}$$

OUTPUT CAPACITOR

The output capacitor determines the output ripple voltage, for this reason a small *Effective Series Resistance* (ESR) is required.
The frequency operation of *LT1374*, as already said, is equal to $500\ Hz$ and at this frequency any polarized capacitor is essentially resistive. As suggested from datasheet, for typical *LT1374* application the ESR has to range from $0.05\ \Omega$ to $0.2\ \Omega$, for this reason the output capacitor is a *solid tantalum capacitor*. The choice of $100\ \mu F$ is a good trade-off between output ripple voltage and physical area.

SCHOTTKY DIODE

The chosen diode is *On Semiconductor MBR330* because of its capability of supporting a $3\ A$ average forward current and $30\ V$ reverse voltage.
Indeed, the reverse voltage is approximately $12\ V$ (the output voltage), while the average forward current is given by the (Eq.9).

$$I_{D(AVG)} = \frac{I_{OUT}\cdot(V_{IN}-V_{OUT})}{V_{IN}} \tag{9}$$

The (Eq.9) will never yield values higher than $3\ A$, neither in worst-case scenario.

BOOST CAPACITOR

The boost capacitor has been chosen based on the voltage that has to support, which is basically equal to the output voltage ($12\ V$) and the (Eq.10) provided by *LT* on datasheet of the component. In this application result that its minimum value is equal to $1.5\ nF$.

$$C_{MIN} = \frac{(I_{OUT}/50) \cdot (V_{OUT}/V_{IN})}{f \cdot (V_{OUT} - 3)} \tag{10}$$

1.5.2 Relay

Since the electrovalves may need $12\ V$ or $24\ V$ a way to switch between this two voltages supplies is needed. The circuit shown in (Fig.26) has been used for this purpose.
It allows the switching trough the firmware.The circuit uses a optocoupler, the DPC-817C, to isolate the Beaglebone Black to the relay, and prevent in this way that pounces produced by magnetic part of relay reach the general purpose pin of the Beglebone itself. The relay used is the G5LA1CF24DC and, as happened in (Sec.1.4), a diode has been exploited to preserve the transistor used as voltage controlled switch from broking.

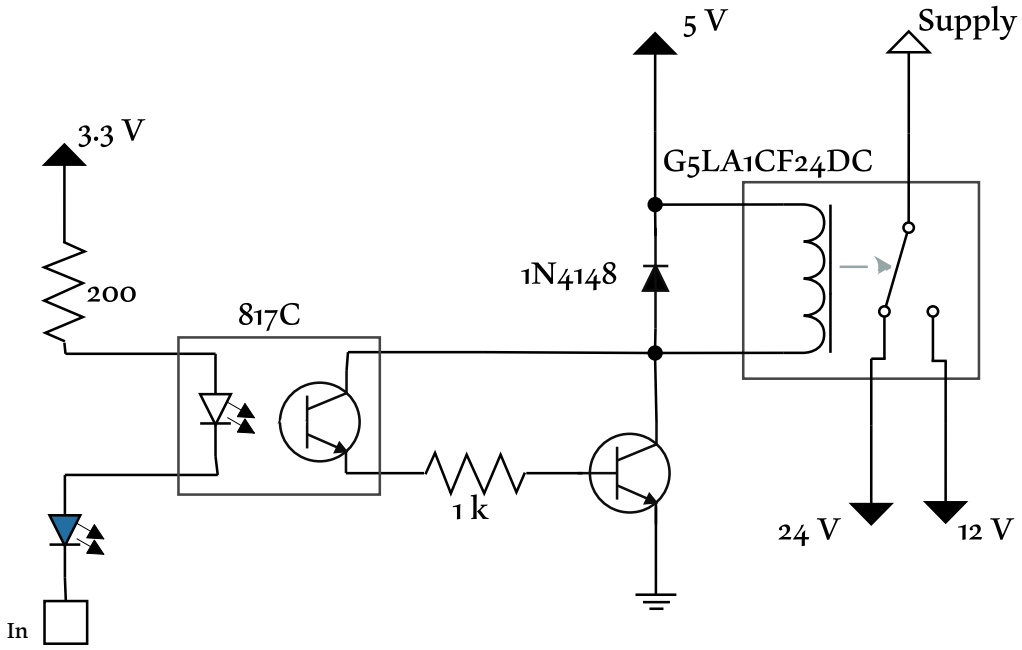


Figure 26: Relay circuit

Summary, in the circuit represented in (Fig.26) the *Supply* is the voltage which is going to be provided to the electrovalves. When the *In* signal is asserted, the relay is in its normal connection, son *Supply* is tied to $24\ V$. On the other hand, when *In* is denied relay is excited and *Supply* is tied to $12\ V$.

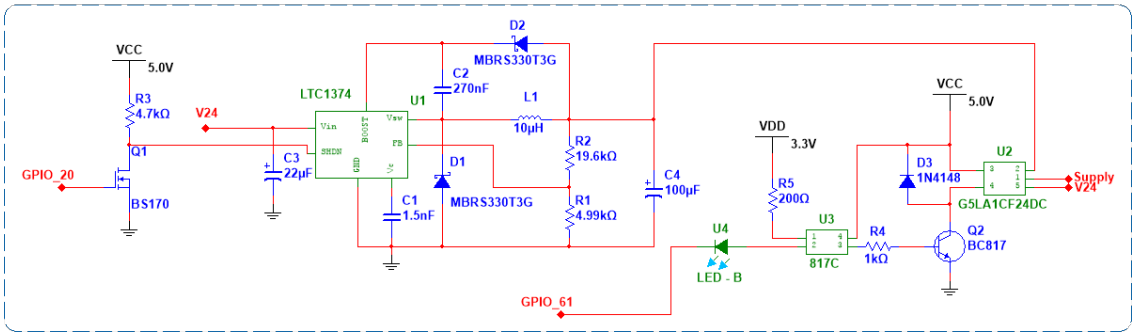


Figure 27: Electrovalves Supply Circuit

The circuit in (Fig.27) shows the connection between *DC-DC* circuit and the switching one, all of this is necessary to correctly supply the different kind of electrovalves.

2 PCB

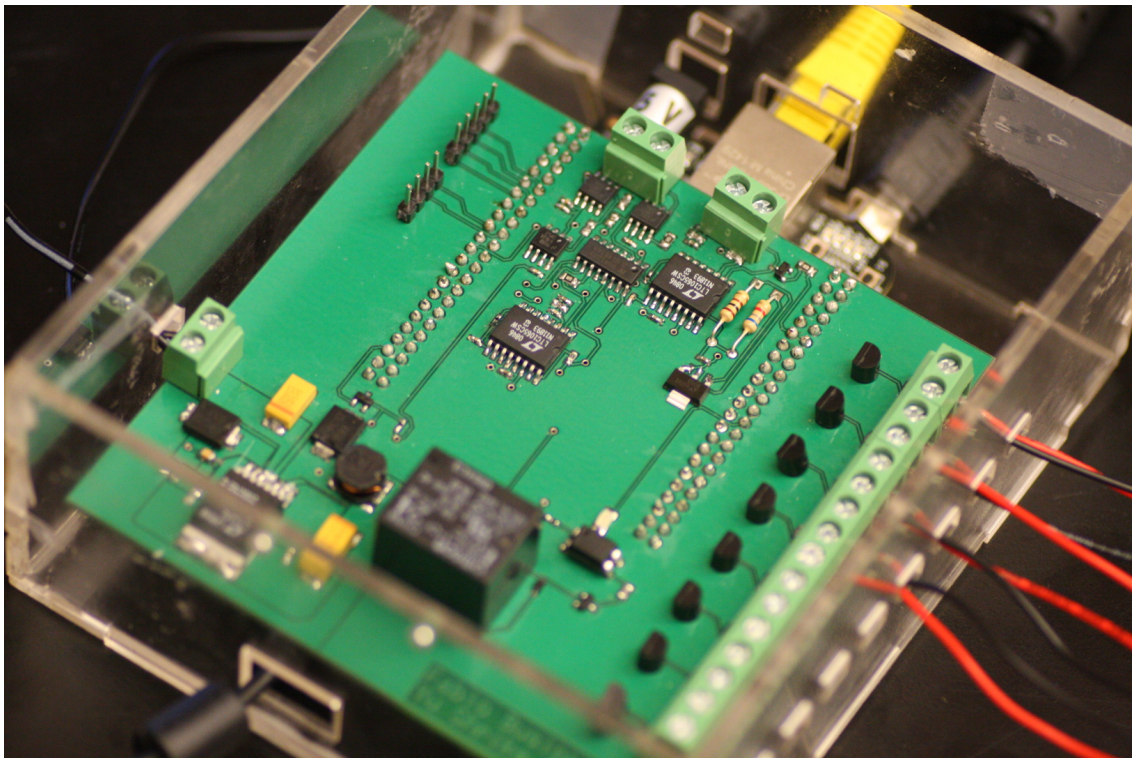


Figure 28: Final system mounted on a PCB

All the hardware and the circuit described so far has been mounted on a Printed Circuit Board (*PCB*) that is supposed to be a *cape* of the Beaglebone Black. This means that the whole physical structure has been made to be attached on the headers of the board. In (Fig.29) all the circuit that has to be made on the *PCB* is shown.

The PCB has been designed in a two-layer board ($100\ x\ 100\ mm$) using the *National Instruments Circuit Design Suite*, in particular *Multisim* to carry out the schematic and *Ultiboard* for what concern the PCB.
The design has followed the guidelines for reduced electromagnetic interface (*EMI*). First of all, since Surface-mount devices (*SMD*) are better than Through-hole components (*THD*) in dealing with RF energy, because of reduced inductances and closer component placements available ([2]), where there was a choice, *SMD* components have been used.

Particular attention has been paid for the *DC-DC* converter layout, because a wrong *PCB* design for switching power supply often means failure. Moreover, in these terms, what is good for *EMI* is also good in terms of functional stability for the regulator.

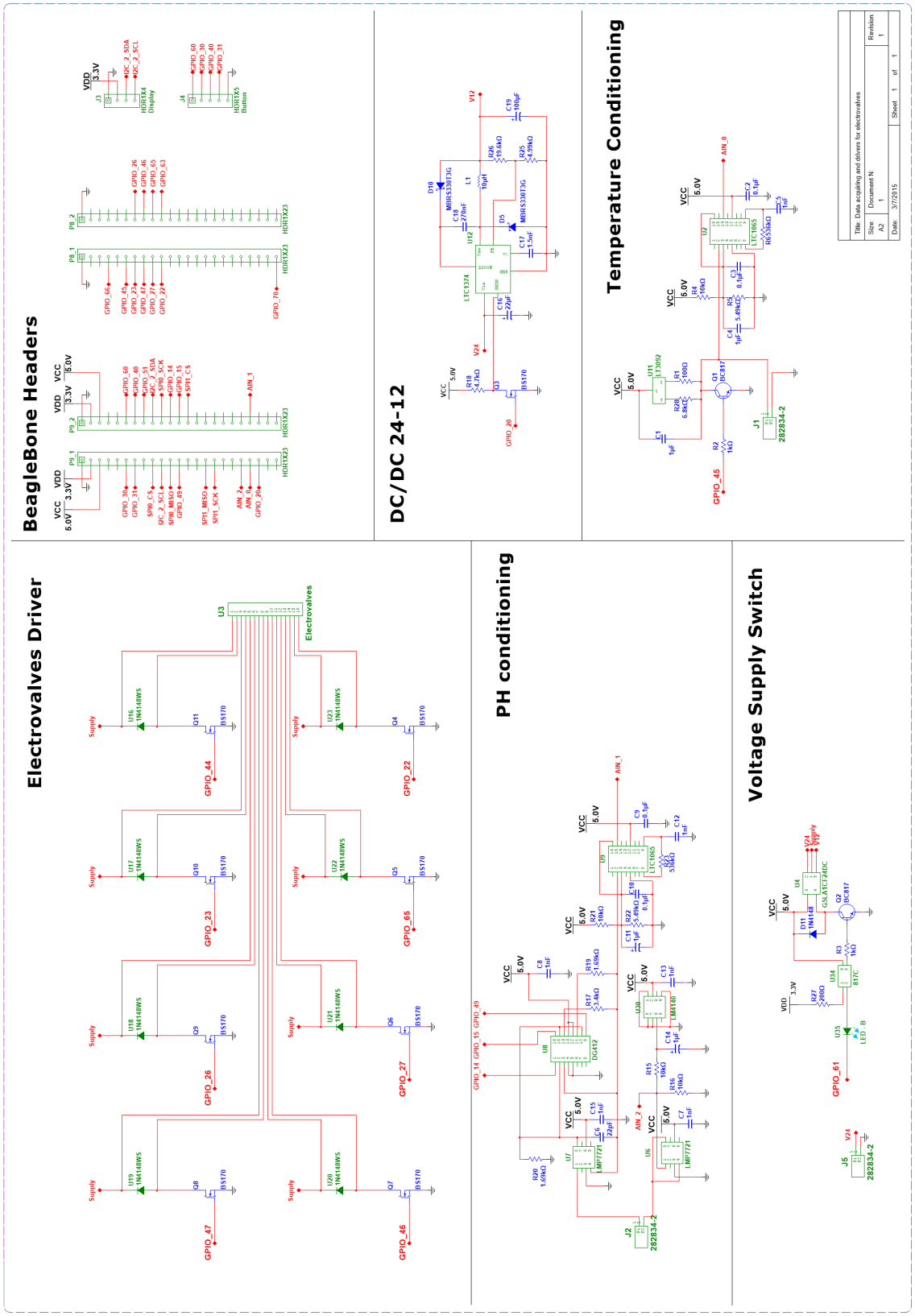


Figure 29: Schematic of Complete Circuit

The circuit in (Fig.30) is schematically a common buck regulator. In that figure, with a blue circle, is highlighted the high speed switching current path: the loop which produces the highest *EMI*. Indeed, in the blue loop flows a fully switched alternated current, and for this reason it is also referred as **hot loop**.

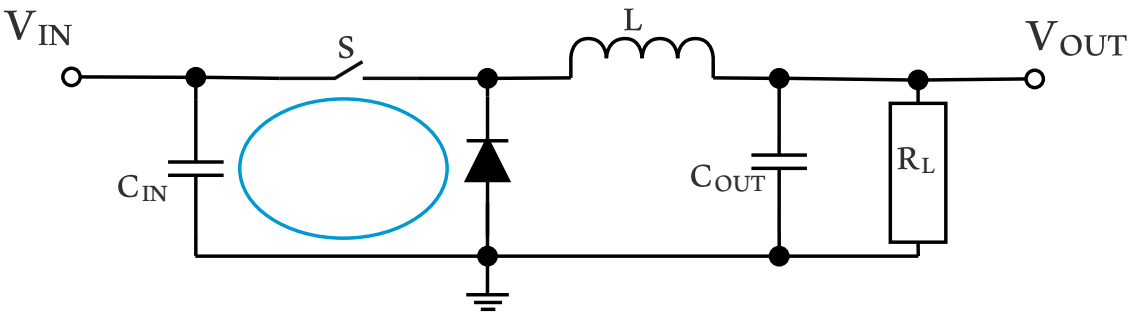


Figure 30: *DC-DC* High Speed Switching Path

In order to ensure clean switching and reduce *EMI* the minimum lead length is required for reducing the radiating effect of the hot loop as much as possible: and so it has been done, as can be seen from (Fig.31).

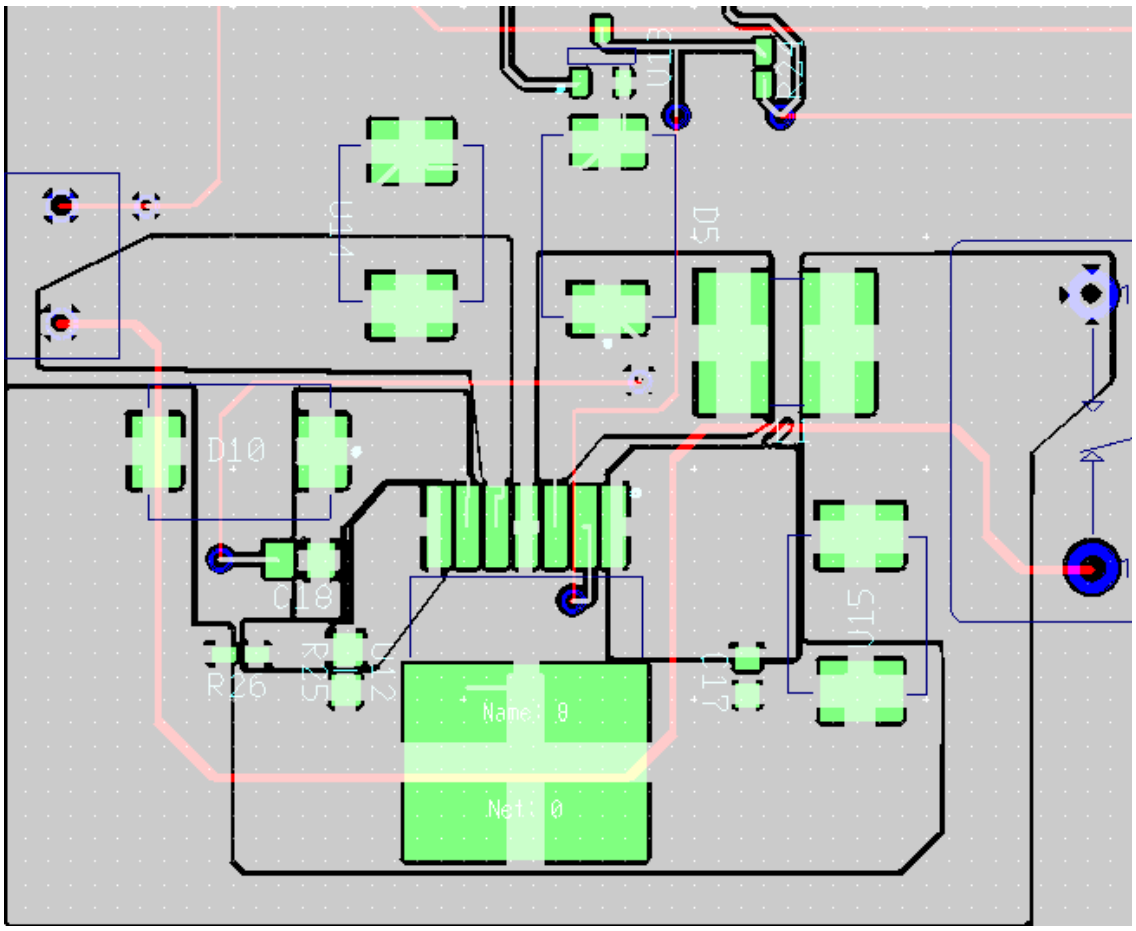


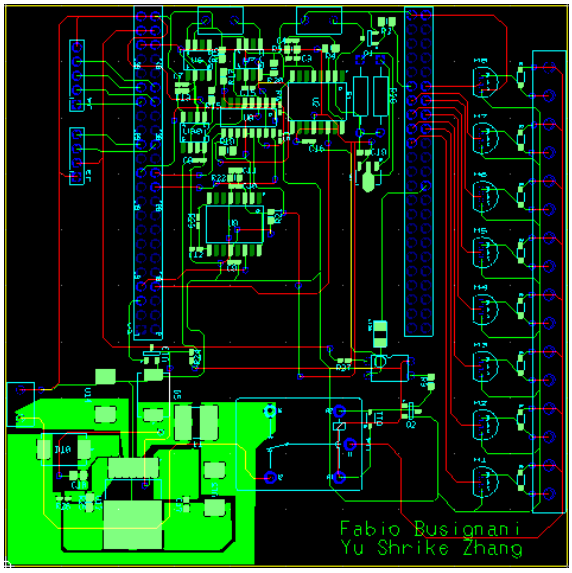
Figure 31: *PCB* Layout of *DC-DC*

In (Fig.31) the *PCB* layout for *DC-DC* converter has been shown, in which we can see that:

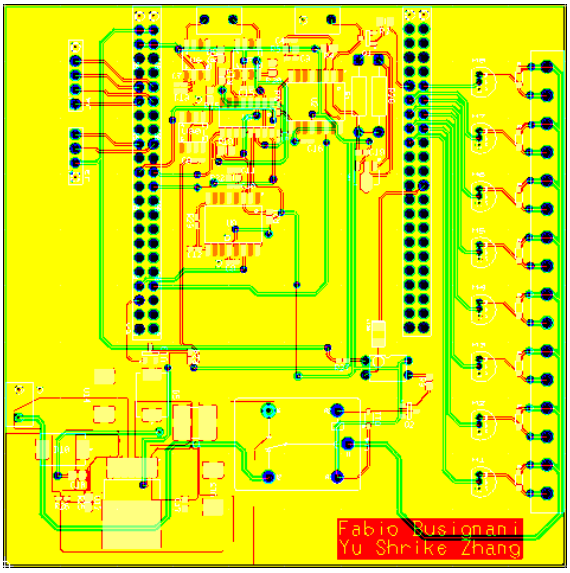
- the magnetic radiation is minimized by keeping by keeping catch diode and the input capacitor leads as short as possible;
- the electric radiation is minimized by reducing the area and length of all traces connected to the switch and boost pins.

Moreover, in order to reduce the noise on the feedback, that is translated in error on output voltage, the switch node and the feedback resistors are kept as far as possible from each others.

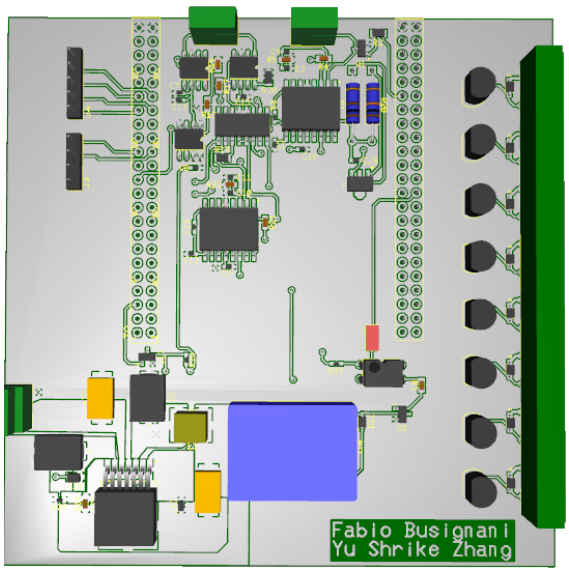
The (Fig.32) shows the *PCB* layout without ground plane (Fig.32a), then with both top and bottom ground plane (Fig.32b) which help with heat dissipation and *EMI* reduction. In (Fig.32c) and (Fig.32d) the 3-D model generated using *Ultiboard* and the final real result are compared.



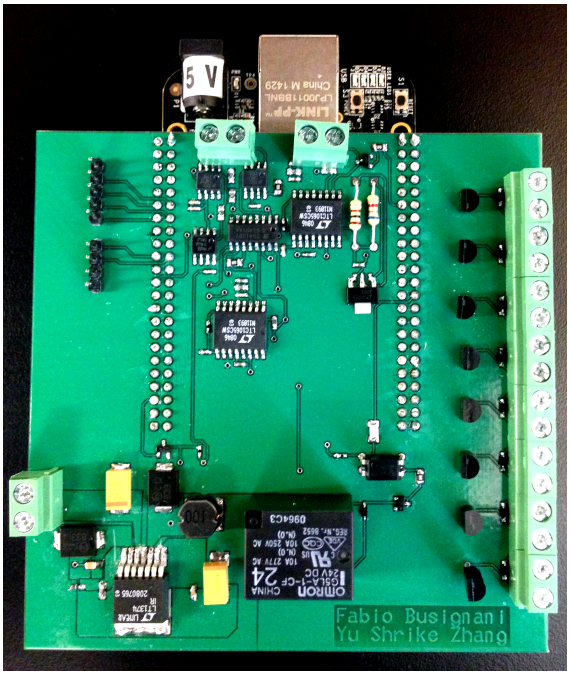
(a) *PCB* Layout Withot Ground Plane



(b) *PCB* Layout With Both Top and Bottom Ground Plane



(c) *PCB* 3-D Model View



(d) *PCB* Real Result

Figure 32: PCB of the System

Part II

FIRMWARE

3 INTRODUCTION

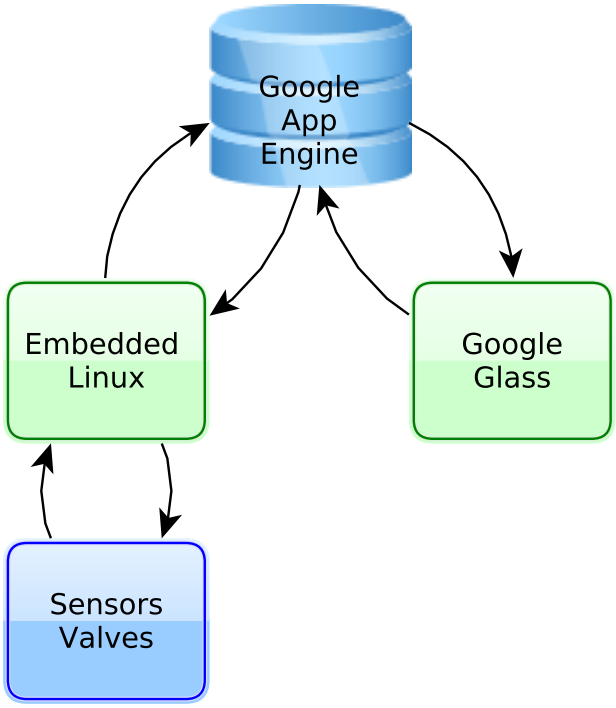


Figure 33: High Level System's Block Diagram

The term *Embedded Systems* is widely used nowadays, because its definition is very generic: any device that includes a programmable computer, but is not itself a general-purpose computer is referred as Embedded Systems.

Indeed this definition covers a huge number of systems, from a vending machine to a smartphone, form a electric toothbrush to a body computer of a car.
Peculiarity of an Embedded System is that it has hardware and software parts, takes advantage of application characteristics to optimize the design, interacts with the external environment using sensors and actuators.

Embedded Linux systems are a subset of embedded systems that are enhanced by a Linux operating system (*OS*), here the integration of high-level Linux software and low-level electronics represents a paradigm shift in embedded systems development [3]. It allows an easy way to design systems that can meet future challenges in smart buildings, the Internet of Things (*IoT*), human-computer interaction(*HCI*), robotics, and many other applications.

In this thesis project, as can be seen from (Fig.33), the system may be intended as an *IoT* one. Indeed, in order to allow the interfacing between sensors and electrovalves with the Google Glass, I had to use an Internet connection, represented by the Google App

Engine. And, while the Google Glass can interact directly with this server, the hardware described in (Part.i) needs a micro computer to be connected with that server.

3.1 A BRIEF INTRODUCTION TO IOT

The term Internet of Things is broadly used to describe the extension of the web and the Internet for the physical objects or "*things*". It is important to realize that the physical connection of the human Internet and the ones of the *IoT* are the same. And even if the *IoT* term has be coined recently, the majority of web traffic nowadays is non-human [4].

Indeed, the massive usage of online services, in part due to the smartphone revolution, has increased the interactions between servers, machine-to-machine (*M2M*) communications. This trend can only increase since it is expected an explosion of wearable systems, such as smartwatch, smartglass and so on..

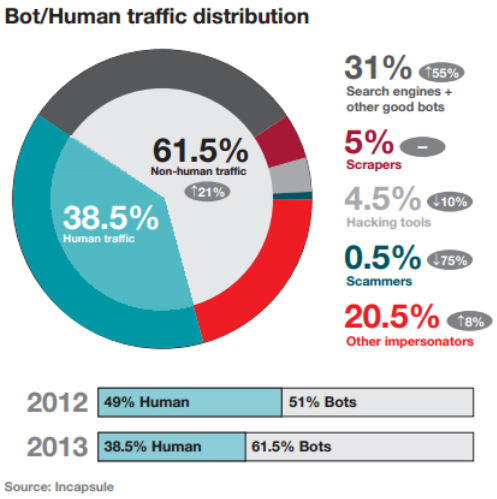


Figure 34: Distribution of Internet of Things

In the near future, *Cisco IBSG* predicts there will be 25 billion devices connected to the Internet by 2015 and this number is destinate to reach 50 billion by 2020, see (Fig.35)[5]. It is also important to see that these number are based on the grown of 2011 and do not consider the rapid advances in device technology. So the number shown in (Fig.35) have to be considered as minimum.

The IoT concept, that has been exploited even in this thesis, is that if physical sensors and actuators can be linked to the Internet , then a lot of new applications and services are possible [3].

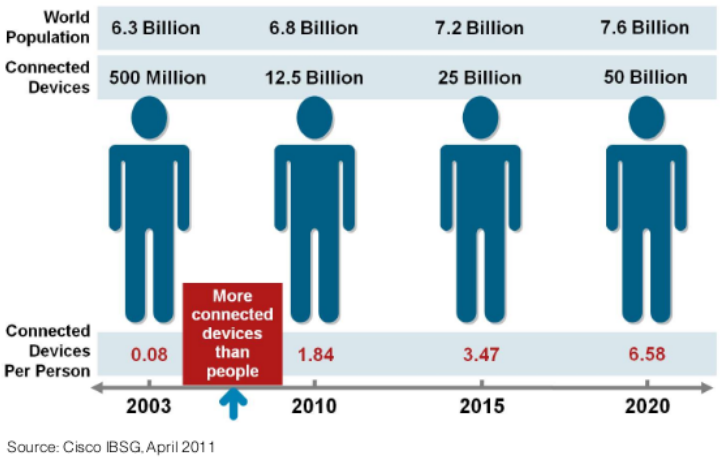


Figure 35: The *IoT* Evolution Prediction

3.2 THE BEAGLEBONE BLACK

For this thesis, the Embedded Linux system has been developed using the *Beaglebone Black*.

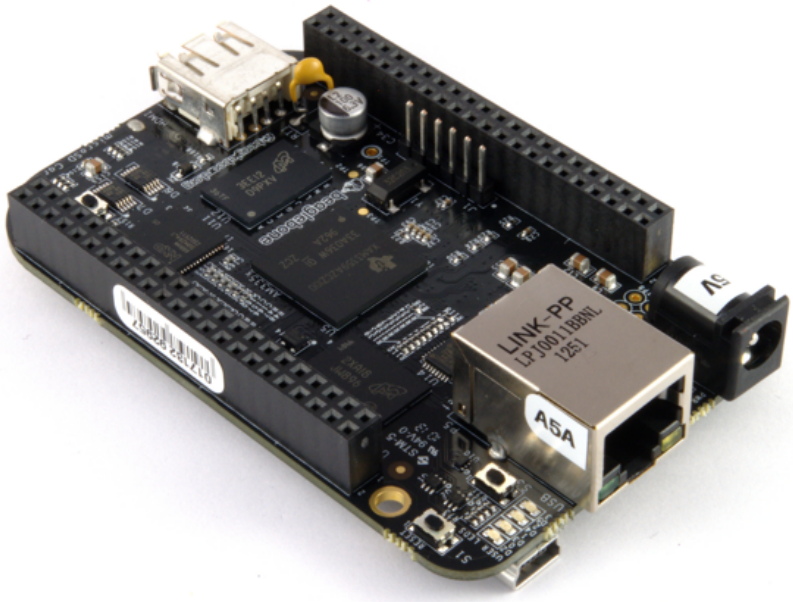


Figure 36: The *Beaglebone Black*

The *Beaglebone Black* is a low-cost, fully open-source (that means everyone can downloads and uses the hardware schematics and layout in order to build up a complete product design), single board micro computer (*SBC*) powered by Linux.

There are a lot of *SBC* on the market, I chose the Beaglebone Black for the following reasons:

- low price, it costs $\$50$;

- powerful, the *Sitara™AM335x ARM®Cortex™-A8* processor from Texas Instruments (*TI*) can run up to $1\ GHz$, allowing up to 2 billion instructions per second;
- $4GB$ on-board embedded multi-media card (*eMMC*), this means the Beaglebone can boot without a *SD* card, unlike other boards like Raspberry PI. It allows faster boot, approximately 10 seconds;
- $512MB$ *DDR3* of system memory;
- Ethernet processor, which supports *DHCP*, so it can be directly conncted to a network
- **expansion headers**, 92 pins with 65 *GPIOs*, 8 analog output, 7 analog input, 3 different voltage supplies ($5\ V$, $3.3\ V$, and $1.8\ V$), and the whole most used digital interface. All these make the Beaglebone Black built to be interfaced to.

Over these, the board has other important feature that are not used in this thesis, but may be used in the future to enhance it. An example is given by the 2 Programmable Real-time Units (*PRU*) that, unlike the definitions of Embedded Linux, make this board usable for real-time applications.

As already said in (Chap.1), this board has been connected to a custom *capes*, a daughterboard that can be attached to the two headers on the Beaglebone itself.

4 EMBEDDED LINUX INSIDE THE PROJECT

In this section the different tasks that Beaglebone has o perform are described in detail. In this project the *Linux distro* used is *Debian*. It has been chosen among the wide possibilities because ensure a good level of stability, and moreover it has a very good repository.

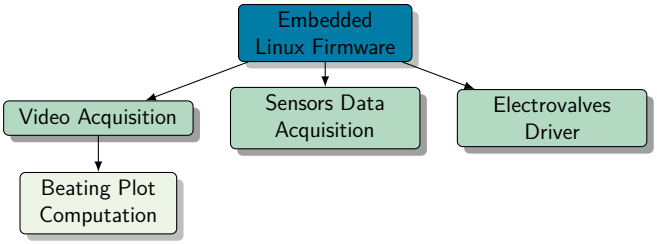


Figure 37: Embedded Linux Firmware - Blocks Subdivision

In (Fig.37) are shown the three principle subsection of the firmware. For each of them a *C++* program has been written, and their executable files are going to be run through a *bash* script, which is supposed to be launched at the system boot. The (List.A.1) shown this bash script that behaves as shown in (Fig.38).

First of all the *bash* script loads the *Device Tree Overlay*. The pins of *Sitara™* microprocessor are *multiplexed*, it means that each pin of the two header may assume different behavior (*GPIO*, *SPI*, *I2C*, and so on..). During the booting a default value for this mux is loaded, and for certain pins, this default value differs from the actual value that the system needs. In order to change those values I wrote a Device Tree Overlays (*DTO*) which has to run at the beginning, immediately after the booting, in order to explain how is it work the Flattened Device Tree (*FDT*) has to be introduced.

The last releases of Linux kernels for ARM boards use *FDT*, it is a data structure that describes the hardware on board. Thus, it describes every component presents on the board, from the user *LEDs* to the *CPU*. Using this *FDT* it is possible to write a *device tree overlay* in order to change to mode of use of each pin. The (List.A.7) changes the status of 10 pins from their default value, in particular sets the pins used to drive the valves, to allow the current on resistor sensor, and to change the power supply voltage on the valves to be digital output pins, with a pulldown resistor.

To record the video from the microscope I used the program ***Video for Linux v.2*** (*V4L2*), before start record the *bash* script sets the video format as *MJPG* and the dimension as $320\ x\ 240\ @30\ fps$. After that it runs in background the program to update the electrovalves status and to acquire the sensors value. As it is explained in (Sec.4.3), these programs run out of the infinite loop because they have an infinite loop inside.

Inside the infinite loop the *bash* script launch the recording of 10 seconds microscope video, compresses the latter and elaborate it in order to compute an image with the beating plot. Finally, it uploads (using the library *CURL*) video and image and update the flags

that in order to indicate to Google Glass App and Video Storing software that a new video has been updated.

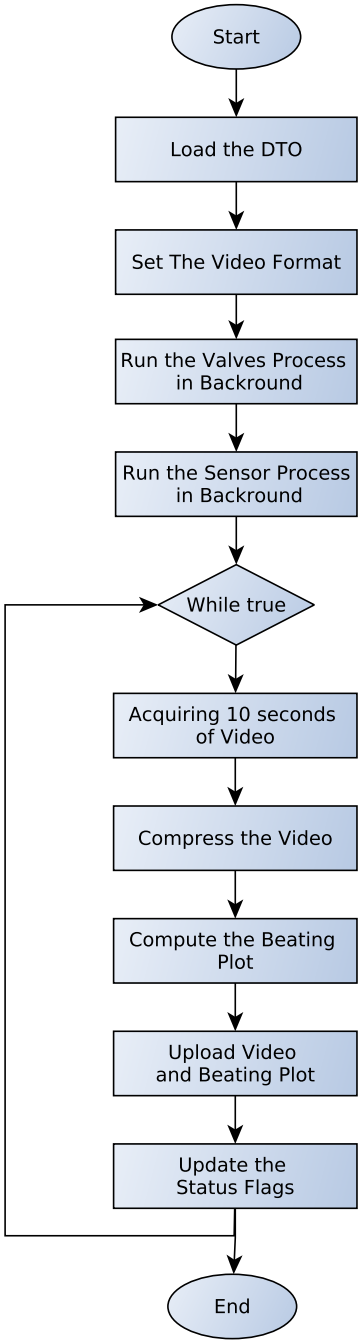


Figure 38: Bash Script Flow Chart

4.1 VIDEO PROCESSING AND BEATING PLOT

As already introduced once the microscope video has been recorded in a *MP4* format it has to be processed in order to compute the beating rate. The code described in this section is shown in (List.A.6).

To achieve this goal, the *OpenCV* library has been used. It is a specific library for computer vision.

As can be seen the program captures the video from the file and, elaborating frame by frame it computes the point to be plotted. In particular it sores the first frame of video, then each point is given by the mean different color density between the under study frame and the first one.

The results has been stored in file called *video_data.dat*. This file is going to be the input for the *gnuplot* task that is in charge to plot the beating data.

4.2 SENSOR DATA ACQUISITION

From the main *bash* script another one is launched for acquiring the sensors data. This script is shown in (list.A.2) and as can be seen it is an infinite loop where two operations are performed:

1. cleaning the previous data from the datastore of Google App Engine server, otherwise plots on the Google Glass are impossible to be viewed;

2. acquiring and uploading 100 new sensors data.

This second step is performed by another task shown in (List.A.5), in which three objects have been used (Fig.39).

How it works: using a for loop, program runs 100 times the reading of pH and temperature values, after that the reading step has been accomplished, the program is in charge to store these value on the Datastore of Google App Engine then it waits for $100\ ms$. Two important features has to be highlighted:

1. as already introduced in (Sec.1.3), there is a *BJT* inside the temperature conditioning circuit that acts as software-controlled switch. This is necessary to avoid the self-heating of temperature sensor and ensure a correct measure. This software-controlling may be seen from the code, indeed there is a digital output called *temperature_disable* that is normally denied, and it is asserted only when a temperature measure has to be carried out;

2. as already introduced in (Sec.1.2), the pH measuring process need to know exactly how much is the amount of added offset. Indeed, in order to make the pH sensor response unipolar, the circuit polarizes the negative pin of pH sensor at approximately $512\ mV$. Instead of using the theoretical value it is better to measure it just before every pH measure process. This is easily viewable from the code.

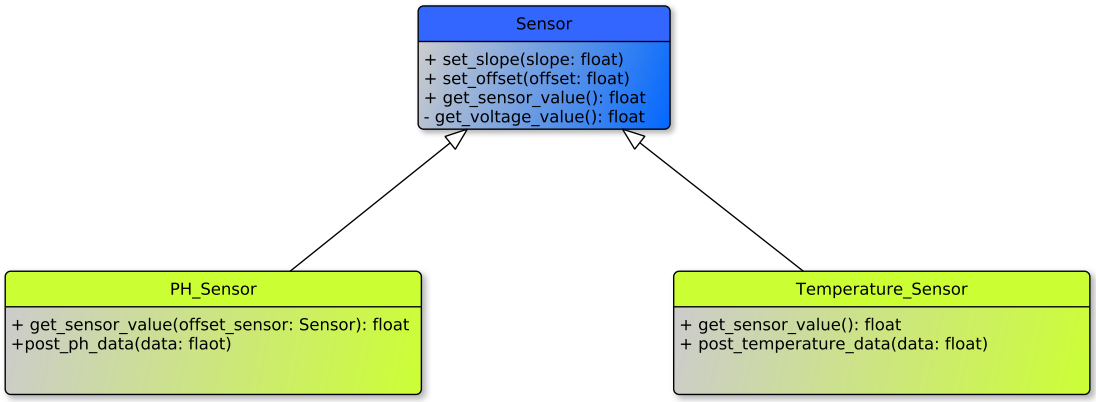


Figure 39: Sensor UML Description

4.3 ELECTROVALVES UPDATING TASK

The code of task which performs the driving of the valves is shown in (List.A.4). Also this program uses the *curl* library to communicate with the Google App Engine, and also this program has been built with an infinite loop.

What the task does is to download from the Google App Engine the status of each electrovalve that has been set by the user through the Google Glass. These status value are then stored onto a vector, to be written in a second step onto the output pins. As explained in (Sec.1.4) each valve is connected to a *MOS* transistor that acts as voltage-controlled switch. The voltage which drives the transistor is the digital value written onto the output pins.

Part III

SOFTWARE

Rising in the logical level we find the *software*. In this part of thesis the python app which controls the server and the video storing program are going to be explained.

The first plays the important role to link the circuit, the embedded Linux system, with the Google Glass App, whenever the user is. While the second is in charge to store every new video that has been recorded, inside an hard drive of a generic computer, in order to be watched in a second time.

More details of each of two are shown in the following sections.

5 GOOGLE APP ENGINE

Figure 40: Google App Engine Logo

Google App Engine is a Platform as a Service (*PaaS*) which allows users to build and run web applications on Google infrastructure.

Thus, when a programmer writes a web application that runs on App Engine, the software is going to run on the Google servers, somewhere in the Google cloude [6]. This solution is very useful because allows everyone to write the own web program without purchasing and maintaining the needed hardware and network.

For the time being Google App Engine supports web applications written in a variety of the main programming languages for the web: *Java*, *Python*, *PHP*, and *GO*. Since the App Engine was first built in 2008 for Python, this is the language used to carry out the web application for this thesis.

Over the possibility to create our own web application without dealing with hardware, the Google App Engine has been chosen because of its datastore and memcache, which allow a simple memorization of data, automatic scaling and load balancing, and more others important features that are going to be shown below.

To create a more scalable and hierarchical web application the *Flask framework* has been used. Flask is a lightweight web application framework written in Python. It is based on the *WSGI* toolkit and Jinja2 template engine and it is distributed with a *BSD* license.

Flask provides an easy way to organize a web application which allows to build up complex website easy to manage. It has no database abstraction layer, indeed for this purpose, in this application, it supports the Google App Engine.

Before to explain how the code is made, the App Engine Datastore and Memcache are going to be introduced.

Using the Google App Engine means do not have access to traditional database like Oracle and MySQL. In fact, App Engine uses ***Google Datastore***, which is easier to use because it takes more of a hierarchical object-oriented storage approach. That approach

Figure 41: Flask Logo

allows to ensure efficient application scalability. Thus, the Datastore holds data objects known as *entities*. An entity is composed by one or more *properties*. Making a comparison with the object oriented it is possible to say that the properties are the fields of objects. So, likely the field, a property may support different data types such as integer, float, string and so on... Any entity holds its *kind* and *key*, which categorize and uniquely identify it, respectively.

The *memcache* results very useful to speed up common Datastore queries, indeed inside the Google Datastore, distributed memory cache are used for storage. For this reason memcache is used to store the status of the electrovalves, in fact in this way the delay caused by the server is highly reduced.

5.1 THE WEB APPLICATION

The main code which runs on server is shown in (List.A.16). It uses two models to describe the data supposed to be stored:

1. *Sensor*, contains the list of sensors used from the system, through the method *sensor_list* it returns that list;

2. *DataPoint*, contains all the points that have to be plotted, the field value represent the $y - axes$ while time the $x - axes$. While the first field has to be passed at the moment in which the class is constructed, the *date* field is auto-filled with the current data. This class has two public methods to interact with:

   - *point_for_sensor*, that with the name of sensor as parameter returns the list of *DataPoint* associated;
   - *oldest_point*, that returns the last *DataPoint* inserted.

Below the different services offered by this web application are listed:

- *process_values*, available at the path `/sensor_values`. User can access to this service using both *HTTP GET* and *POST* methods. With the first one, it returns the *JSON* (*JavaScript Object Notation*) containing all the data regarding the whole sensors stored inside the Datastore, in this format: { value, timestamp, sensor_name }. On the other hand, when the used method is *POST* the application takes from the POST parameters those named *"sensor"*, indicating the sensor's name, and *"value"*, which is the sensor's value that has to be plotted. Thus, in this last way what the

application does is to add a new data point for a sensor and store it inside the Datastore.

- *print_names*, available at the path `/sensor_names`. This function is accessible only through HTTP *GET* method and returns a *JSON* object with the content of sensor's name list.

- *graphing_data*, available at the path `/graphing_data`. To access to this function a HTTP *GET* method is required. This *GET* request must contain two parameters: *"sensor"*, the sensor name, and *"first_timestamp"*, which is the timestamp of the first point that has to be plotted. This last parameter is not mandatory and if it is missed it is assumed equal to 0. What this function does is to return a *JSON* object containing the whole DataPoint for the given sensor that have timestamp higher than *"first_timestamp"*.

- *clear_data*, available at the path `/clear`. This function is in charge to deletes all the points for each sensor.

- *set_picture*, available at path `/picture/submit`. This function is accessible with both HTTP *POST* and *GET* methods, and in both cases its behavior is to load a new image, containing the beating plot, onto Datastore. In the first case the image has be passed ad *POST* data, and named as *"Image.jpg"*. While th second case is supposed to be a browser way to upload the image. In fact, browsing to that *URL* what the user sees is shown in (Fig.42), a view where it is possible to select the picture file and submit it.

Figure 42: Picture Submitting from the Browser

- *view_picture*, available at the path `/picture/view`. Using this function, with a HTTP *GET* method causes the access to the image previously stored inside the Datastore.

- *set_video*, available at the path `/video/submit`. As happened to the beating plot image, this function is accessible with both HTTP *POST* and *GET* methods. And, as before, it is used to load onto Datastore the microscope video. Using the *POST* method means storing the data passed as parameter and named *"Video.mp4"*. On the other hand, in case of GET method what the user sees on the browser is shown in (Fig.43). As before, through the two buttons, user can choose the video to store and submit the request.

- *view_video*, available at the path `/video/view`. This function, accessible with HTTP *GET* method, accesses to the microscope's video on the Datastore and returns it.

Figure 43: Video Submitting from the Browser

- *add_electrovalve*, available at the path `/add/electrovalve`. This function is in charge to store the valve's status inside the memcache. It is accessible through HTTP *POST* method, which must have these two following parameters: *"name"*, that uniquely identifies the valves (it is *"EV"* follows by the number of the valve, for example the first one is *"EV1"*), and *"status"*, which indicates the actual status of the valve (it may be *on* or *off*).
- *get_electrovalve*, available at the path `/electrovalves/<name>`. It is accessible through HTTP *GET* method and returns the value of the valve identified with <*name*> field inside the *URL* (for example, if someone wants to read the status of the first valve, has to use the path `/electrovalves/EV1`).

6 MICROSCOPE VIDEO STORING

An important role of an experiment in organ-on-a-chip applications, as well as the whole biomedical field, is given by the video analysis. In fact, analyze the video just one time in real-time, during the experiment is running is never sufficient. So, what the system described in this thesis need to be usable is a way to memorize the video in order to be used, watched, in a second time.

To fulfill this aim I designed a console application using the *Qt* framework. The motivation that brought me to use this framework is that the same code can run in different platform. In other words, this application can run on Linux, Windows, and MacOS indistinctly. This is a big advantage, because in a laboratory environment there are many researcher, and it is very easy to encounter different operating systems.

Part IV

GOOGLE GLASS APPLICATION

7 INTRODUCTION

Part V

CONCLUSION AND APPENDIX

8 TEST AND PERFORMANCE

In order to try the reverse control from the Google Glass to the electrovalves we made different kind of experiments.

8.1 LED EXPERIMENTS

First of all we tried the circuit on a breadboard using LEDs instead of electrovalves. The aim of this step is to demonstrate that the firmware running on the Beaglebone Black, the Java code running on the Google Glass and the Python code running on the Google App Engine (used to store the information about the electrovalves status) work well.
Moreover the LED and the electrovalve have basically the same behavior so, if everything works well with the LEDs, there are all the reasons to believe that everything is going to work well with the electrovalves, too.

The circuit that actually drives the LEDs is very simple, and it is based on a MOS transistor (*BS170*) used as a switch voltage-controlled, as shown in (Fig.44).

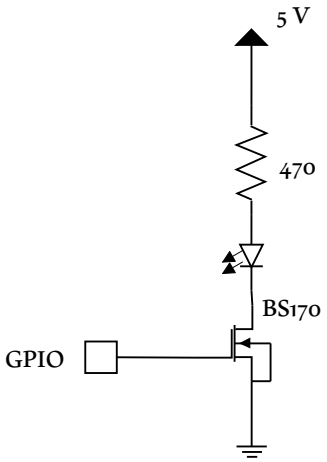


Figure 44: Driver for LED

The (Fig.45) shows the circuit used for this step of testing. As can be seen the number of LEDs used is eight, the same number of electrovalves that can be driven from this system.
In order to test all of them we made 2 different kind of trials:

1. *In order turning on&off*, first all the LEDs are turned on starting from the first one (on the top right corner) to the last one (on the bottom left corner). Then the LEDs are turned off following the same order.
   The result of this can bee watched in **this** video.

2. *Out of order turning on&off*, in this trial, like before, all the LEDs start from a condition where all of them are off and then we turned on and off all the LEDs, but in this case following a random order.
   The result of this can bee watched in **this** video.

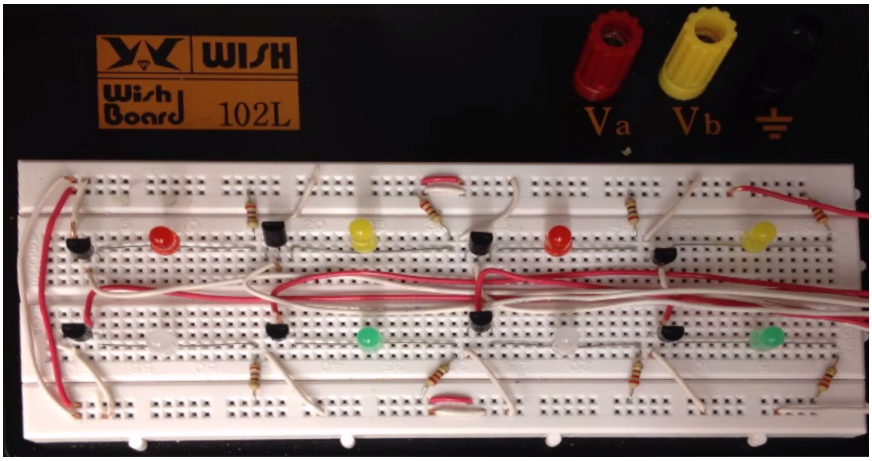


Figure 45: LEDs experiments board

8.2 ELECTROVALVES EXPERIMENTS

8.2.1 Breadboard Phase

The (Fig.46) shows from the top view the circuit used during the second phase of experiment, the one where we started using electrovalves in a real microfluidic application.
On the left side of the figure we can see the conditioning circuits for the temperature sensor (on the top) and pH sensor (on the bottom). While, on the other side, we can see the part of circuit in charge to drive the electrovalves.
In this last one we are going to focus for now. Each electrovalve is driven by the circuit shown in (Fig.23).

As you can see, this circuit is pretty close to the one of (Fig.44), indeed the only difference is given by freewheeling diode, mandatory because of inductive behavior of electrovalve's solenoid.

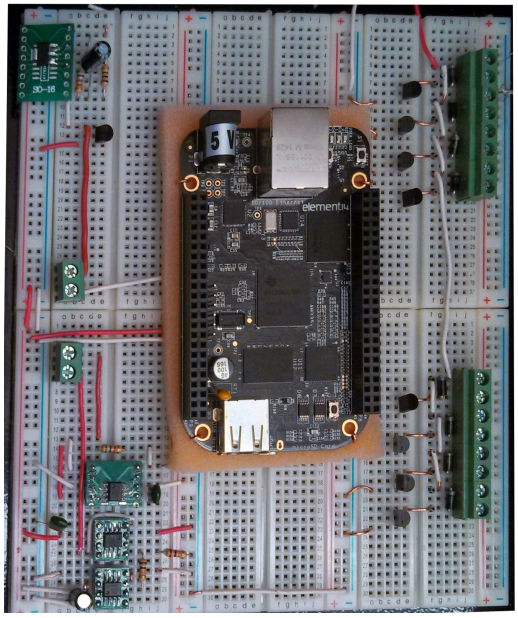


Figure 46: Circuit mounted on breadboard top view

The result of the experiments with electrovalves in a real microfluidic case can bee watched in **this** video.

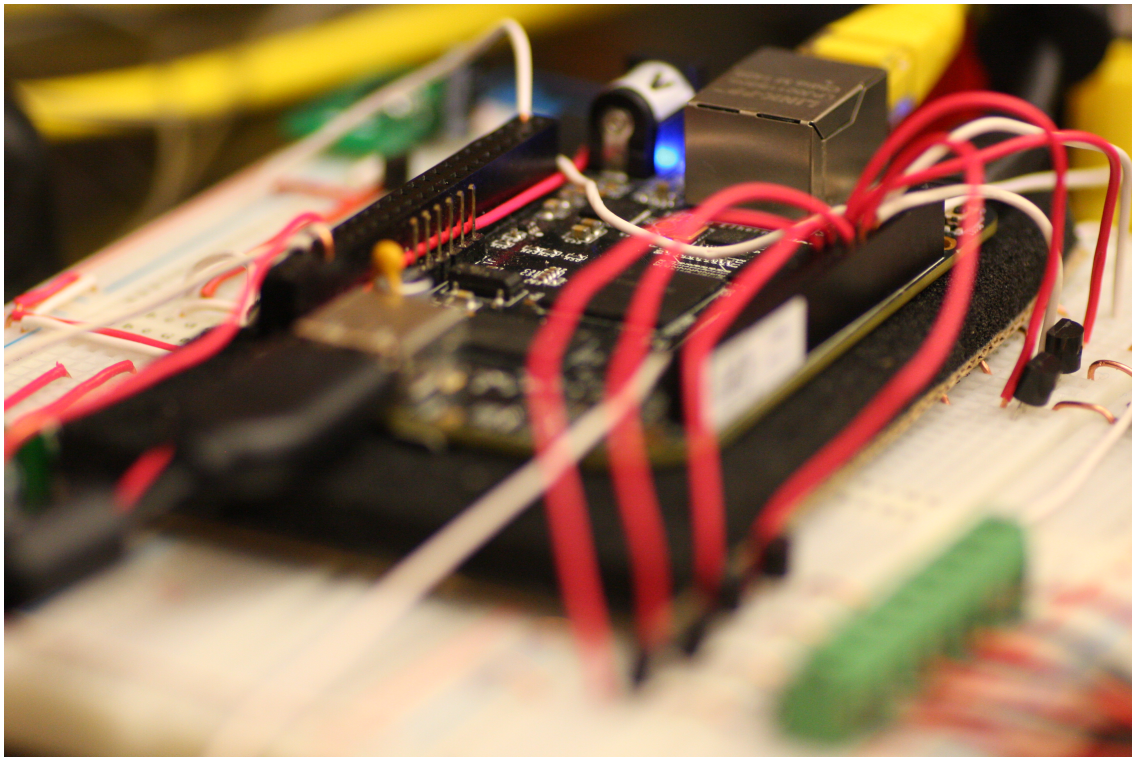


Figure 47: Circuit mounted on breadboard side view

8.2.2 PCB Phase

Finally we replied the last experiment using a PCB (Fig.32), designed for this system.
As expected the result of this experiment is the same of the previus step.

A | CODE

A.1 FIRMWARE

Listing A.1: KlabFirware (bash)

```bash
#!/bin/bash
echo "Load the DTO"
echo GlassProject-GPIO > $SLOTS

echo "Setting the video format as MJPG"
v4l2-ctl --set-fmt-video=width=320,height=240,pixelformat=1 -d 0
v4l2-ctl --set-parm=30

echo "Running the Electrovalve status updating task"
./Electrovalves/ValvesUpdating </dev/null &>/dev/null &
./Sensors/sensorAcquiring </dev/null &>/dev/null &

while true; do
  echo "Recording ten second of Video"
  ./Video/capture -d /dev/video0 -F -o -c 300 > output.raw -y
  avconv -f mjpeg -i output.raw -vcodec copy output.mp4 -y
  echo "Compressing Video"
  ffmpeg -i output.mp4 -acodec mp2 Video.mp4
  echo "Uploading the Video to the server"
  curl -include --form Video.mp4 =@Video.mp4 -A "National Instruments LabVIEW"
      http://planar-contact-601.appspot.com/video/submit -0
  echo "Updating the flag for video"
  curl  --data "name=video&status=on"
      http://planar-contact-601.appspot.com/add/electrovalve
  curl  --data "name=glass&status=on"
      http://planar-contact-601.appspot.com/add/electrovalve
  echo "Ploting the beating rate"
  ./Video/compute_beating
  echo "Uploading the beating plot"
  curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW"
      http://planar-contact-601.appspot.com/picture/submit -0
  echo "finish"
done
```

Listing A.2: SensorAcquiring (bash)

```
#!/bin/bash

while true; do
  echo "Clearing previus data on server"
  curl http://planar-contact-601.appspot.com/clear
  echo "Acquiring and uploading new sensors data"
  ./sensor_acquiring

done
```

Listing A.3: Pins Setting

```
/*
* Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Purpose License Version 2 as
* published by the Free Software Foundation
*
* Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
*
* Modified by Fabio Busignani fbusigna@mit.edu
*
*/

/dts-v1/;
/plugin/;

/{
        compatible = "ti,beaglebone", "ti,beaglebone-black";
        part-number = "KLab";
        version = "00A0";

        fragment@0 {
              target = <&am33xx_pinmux>;

              __overlay__ {
                   ebb_example: KLab {
                          pinctrl-single,pins = <
                                0x04c 0x07  // P9_16 - T - Output Mode7 pulldown
                                0x024 0x07  // P8_13 - EV1 - Output Mode7 pulldown
                                0x028 0x07  // P8_14 - EV4 - Output Mode7 pulldown
                                0x03c 0x07  // P8_15 - EV3 - Output Mode7 pulldown
                                0x038 0x07  // P8_16 - EV6 - Output Mode7 pulldown
                                0x02c 0x07  // P8_17 - EV5 - Output Mode7 pulldown
                                0x08c 0x07  // P8_18 - EV8 - Output Mode7 pulldown
                                0x020 0x07  // P8_19 - EV7 - Output Mode7 pulldown
                                0x030 0x07  // P8_44 - EV2 -Output Mode7 pulldown
```

```
                                0x0a0 0x07  // P8_46 - PowerSupply


                        >;
                };
            };
        };

        fragment@1 {
    target = <&ocp>;
    __overlay__ {
      gpio_helper {
        compatible = "gpio-of-helper";
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&eklab>;
      };
    };
  };
};
```

Listing A.4: ElectrovalvesDriver.cpp

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <sstream>
#include <curl/curl.h>
#include <GPIO/GPIO.h>

#define NUMBER_ELECTROVALVES 8

using namespace std;

std::string const ELECTROVALVES_LINK =
    "http://planar-contact-601.appspot.com/show/";
std::string const TRUE_VALUE = " True ";
std::string const FALSE_VALUE = " False ";

static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
  ((std::string*)userp)->append((char*)contents, size * nmemb);
  return size * nmemb;
}

int main(void)
{
  // constructor of pins
  GPIO EV1(44), EV2(23), EV3(26), EV4(47), EV5(46), EV6(27), EV7(65), EV8(22);
        GPIO RELAY(61);
```

```
29  CURL *curl;
    CURLcode res;
31
    GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];
33
    //Set output pins
35  EV1.setDirection(OUTPUT);
    EV2.setDirection(OUTPUT);
37  EV3.setDirection(OUTPUT);
    EV4.setDirection(OUTPUT);
39  EV5.setDirection(OUTPUT);
    EV6.setDirection(OUTPUT);
41  EV7.setDirection(OUTPUT);
    EV8.setDirection(OUTPUT);
43
    RELAY.setDirection(OUTPUT);
45
    RELAY.setValue(HIGH);
47
    while(1)
49  {
      //acquire the values of electrovalves status and store them inside
51    //output_vector
      int i;
53    for(i=0;i<NUMBER_ELECTROVALVES; i++)
      {
55      curl = curl_easy_init();
        if(curl)
57      {
          std::string readBuffer;
59        std::ostringstream ss;
          ss << i+1;
61        std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
          curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
63        /* example.com is redirected, so we tell libcurl to follow redirection */
          curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
65        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
          /* Perform the request, res will get the return code */
67        res = curl_easy_perform(curl);
          /* always cleanup */
69        curl_easy_cleanup(curl);

71        if(readBuffer.compare(TRUE_VALUE) == 0)
          {
73          std::cout << "EV" + ss.str()+": ON" << std::endl;
            output_vector[i] = HIGH;
75        }
          else if(readBuffer.compare(FALSE_VALUE) == 0)
```

```
77          {
              std::cout << "EV" + ss.str()+": OFF" << std::endl;
79            output_vector[i] = LOW;
            }
81        }
      }
83    //now I'm ready to write the output pins:
      EV1.setValue(output_vector[0]);
85    EV2.setValue(output_vector[1]);
      EV3.setValue(output_vector[2]);
87    EV4.setValue(output_vector[3]);
      EV5.setValue(output_vector[4]);
89    EV6.setValue(output_vector[5]);
      EV7.setValue(output_vector[6]);
91    EV8.setValue(output_vector[7]);
  }
93  return 0;
}
```

Listing A.5: sensor_aquiring.cpp

```
#include<iostream>
2 #include<fstream>
#include<unistd.h>
4 #include<string>
#include<sstream>
6 #include<stdlib.h>
#include<curl/curl.h>
8 #include"../GPIO.h"
#include"../http.h"
10 using namespace std;
#define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
12 #define GPIO_PATH "/sys/class/gpio/"
#define OUTPUT false
14 #define INPUT true
#define HIGH true
16 #define LOW false

18 float const CURRENT = 0.00068;
float const SLOPE_TEMPERATURE = 2;
20 float const OFFSET_TEMPERATURE = 1870;
float const SLOPE_PH = -0.070;
22 float const OFFSET_PH = 0.6;

24 class Sensor
{
26   int number;


28
```

```
public:
  float m;
  float q;
  Sensor(int x): number(x){}
  void set_slope(float slope){
    m = slope;
  }
  void set_offset(float offset){
    q = offset;
  }
  float get_sensor_value(){
    float voltage_value = this->get_voltage_value();
    float sensor_value = (voltage_value - q)/m;
    return sensor_value;
  }
private:
  float get_voltage_value(){
    stringstream ss;
    ss << LDR_PATH << number << "_raw";
    fstream fs;
    int adc_value;
    fs.open(ss.str().c_str(), fstream::in);
    fs >> adc_value;
    fs.close();
    float cur_voltage = adc_value * (1.80f/4096.0f);
    float actual_value;
    return cur_voltage;

  }
};

class Temperature_Sensor : public Sensor{
private:

public:
  Temperature_Sensor(int x) : Sensor(x) {}
  float get_sensor_value(){
    float voltage_value = this->get_voltage_value();
    float resistor_value = voltage_value / CURRENT;
    float sensor_value = (resistor_value - q)/m;
    return sensor_value;
  }
  void post_temperature_data(float data){
    post_data_sensor(0, data);
  }
};

class PH_Sensor : public Sensor{
```

```
78 public:
     PH_Sensor(int x) : Sensor(x) {}
80   float get_sensor_value(Sensor offset_sensor){
         //Sensor offset_sensor(2);
82       float voltage_value = this->get_voltage_value();
         q += offset_sensor.get_voltage_value();
84       float sensor_value = (voltage_value - q)/m;
         return sensor_value;
86     }

88   void post_ph_data(float data){
       post_data_sensor(1, data);
90   }
   };
92
   int main(int argc, char* argv[])
94 {
     GPIO temperature_disable(45);
96   float slope_temperature;
     float slope_ph;
98   float offset_temperature;
     float offset_ph;
100
     if (argc == 5)
102  {
       slope_temperature = atoi(argv[1]);
104    offset_temperature = atoi(argv[2]);
       slope_ph = atoi(argv[3]);
106    offset_ph = atoi(argv[4]);
     }
108  else
     {
110    slope_temperature = SLOPE_TEMPERATURE;
       offset_temperature = OFFSET_TEMPERATURE;
112    slope_ph = SLOPE_PH;
       offset_ph = OFFSET_PH;
114  }

116  //set the pin GPIO_45 as output
     temperature_disable.setDirection(OUTPUT);
118  //Temporary disable the temperature
     temperature_disable.setValue(HIGH);
120
     Temperature_Sensor temperature_sensor(2);
122  PH_Sensor ph_sensor(1);
     Sensor offset_sensor(2);
124
     temperature_sensor.set_slope(slope_temperature);
126  temperature_sensor.set_offset(offset_temperature);
```

```
  ph_sensor.set_slope(slope_ph);
  ph_sensor.set_offset(offset_ph);

  int i= 0;
  for(i=0; i<100; i++)
  {
    temperature_disable.setValue(LOW);
    float ph = ph_sensor.get_sensor_value(offset_sensor);
    float temperature = temperature_sensor.get_sensor_value();
    temperature_disable.setValue(HIGH);
    cout << "The voltage value is: " << temperature << " V." << endl;
    temperature_sensor.post_temperature_data(temperature);
    ph_sensor.post_ph_data(ph);
    usleep(100000);
  }
  float temperature = temperature_sensor.get_voltage_value();
  cout << "The voltage value is: " << temperature << " V." << endl;
  return 0;
}
```

Listing A.6: compute_beating.cpp

```

#include<iostream>
#include<fstream>
#include<string>
#include <curl/curl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include<sstream>
#include<opencv2/opencv.hpp>   // C++ OpenCV include file
using namespace std;
using namespace cv;            // using the cv namespace too

int main()
{
    ofstream point_file;
    point_file.open("video_data.dat");
    VideoCapture capture; // capturing from file
    capture.open("output.mp4");

    // set any  properties in the VideoCapture object
    capture.set(CV_CAP_PROP_FRAME_WIDTH,320);   // width pixels
    capture.set(CV_CAP_PROP_FRAME_HEIGHT,240);   // height pixels

    if(!capture.isOpened())
    {
        cout << "Capture not open." << endl;
    }
```

```
29      Mat frame, firstFrame, diffFrame;

31      capture >> frame; //save the first frame
        firstFrame = frame;
33      Scalar color_information = mean(firstFrame);
        float initial_point = (color_information[0] +color_information[1]
35        +color_information[2])/3;

37      int i;
        for(i=1; i<capture.get(CV_CAP_PROP_FRAME_COUNT);i++)
39      {
            capture >> frame;           // capture the image to the frame
41          if(frame.empty())
            {
43              cout << "Failed to capture an image" << endl;
                return -1;
45          }
            absdiff(frame, firstFrame, diffFrame);
47          color_information = mean(frame);
            float red = color_information[0];
49          float green = color_information[1];
            float blue = color_information[2];
51
            float mean_color = (red+green+blue)/3;
53
             //qui poi chiamo la funzione che mi posta il punto nel db
55           float temp = (i-1);
            temp *= 0.033;
57
    point_file << temp << '\t';
59          point_file << mean_color << '\n';

61       }


63
        point_file.close();
65      return 0;
}
```

Listing A.7: Pins Setting

```
/*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
4 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Purpose License Version 2 as
6 * published by the Free Software Foundation
 *
```

```
* Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
*
* Modified by Fabio Busignani fbusigna@mit.edu
*
*/

/dts-v1/;
/plugin/;

/{
        compatible = "ti,beaglebone", "ti,beaglebone-black";
        part-number = "KLab";
        version = "00A0";

        fragment@0 {
              target = <&am33xx_pinmux>;

              __overlay__ {
                   ebb_example: KLab {
                           pinctrl-single,pins = <
                                 0x04c 0x07  // P9_16 - T - Output Mode7 pulldown
                                 0x024 0x07  // P8_13 - EV1 - Output Mode7 pulldown
                                 0x028 0x07  // P8_14 - EV4 - Output Mode7 pulldown
                                 0x03c 0x07  // P8_15 - EV3 - Output Mode7 pulldown
                                 0x038 0x07  // P8_16 - EV6 - Output Mode7 pulldown
                                 0x02c 0x07  // P8_17 - EV5 - Output Mode7 pulldown
                                 0x08c 0x07  // P8_18 - EV8 - Output Mode7 pulldown
                                 0x020 0x07  // P8_19 - EV7 - Output Mode7 pulldown
                                 0x030 0x07  // P8_44 - EV2 -Output Mode7 pulldown
                                 0x0a0 0x07  // P8_46 - PowerSupply

                           >;
                   };
              };
        };

        fragment@1 {
    target = <&ocp>;
    __overlay__ {
      gpio_helper {
        compatible = "gpio-of-helper";
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&eklab>;
      };
    };
  };
};
```

Listing A.8: HTTP.h

```
#ifndef HTTP
#define HTTP
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<string>
#include<sstream>
#include<stdlib.h>
#include<curl/curl.h>

std::string const URL_POST = "http://planar-contact-601.appspot.com/sensor_values";
std::string const POST_DATA_TEMPERATURE =  "sensor=temperature&value=";
std::string const POST_DATA_PH =  "sensor=pH&value=";
std::string const POST_DATA_BEATING =  "sensor=beating&value=";


void post_data_sensor(int type, float value);

#endif // HTTP
```

Listing A.9: HTTP.cpp

```
#include "http.h"

void post_data_sensor(int type, float value){
        CURL *curl;
        CURLcode res;
        /* In windows, this will init the winsock stuff */
        curl_global_init(CURL_GLOBAL_ALL);

        /* get a curl handle */
        curl = curl_easy_init();
        if(curl) {
          /* First set the URL that is about to receive our POST. This URL can
             just as well be a https:// URL if that is what should receive the
             data. */
          curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
          /* Now specify the POST data */
          std::ostringstream ss;
          ss << value;
          std::string s;
          if ( type == 0)
                  s= POST_DATA_TEMPERATURE +(ss.str());
          else if (type == 1)
                  s= POST_DATA_PH +(ss.str());
          else
              s= POST_DATA_BEATING + (ss.str());
          curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
```

```
    /* Perform the request, res will get the return code */
28          res = curl_easy_perform(curl);
            /* Check for errors */
30          if(res != CURLE_OK)
              fprintf(stderr, "curl_easy_perform() failed: %s\n",
32                    curl_easy_strerror(res));
    /* always cleanup */
34          curl_easy_cleanup(curl);
        }
36      curl_global_cleanup();
}
```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```
1 #include <QCoreApplication>
  #include "downloader.h"
3 #include "mytimer.h"
  #include <QDebug>
5 #include <windows.h> // for Sleep


7
  int main(int argc, char *argv[])
9 {
      QCoreApplication a(argc, argv);
11    MyTimer t;
      return a.exec();
13 }
```

Listing A.11: VideoStoring.pro

```
1 #-------------------------------------------------
  #
3 # Project created by QtCreator 2015-02-17T21:56:37
  #
5 #-------------------------------------------------

7 QT       += core
  QT       += network
9
  QT       -= gui
11
  #RC_FILE = myapp.rc
13
  TARGET = VideoStoring
```

```
CONFIG   += console
CONFIG   -= app_bundle

TEMPLATE = app


SOURCES += main.cpp \
    downloader.cpp \
    mytimer.cpp

HEADERS += \
    downloader.h \
    mytimer.h
```

Listing A.12: mytimer.h

```
#ifndef MYTIMER_H
#define MYTIMER_H

#include <QObject>
#include <QtCore>
#include <downloader.h>

class MyTimer : public QObject
{
    Q_OBJECT
public:
    MyTimer();
    ~MyTimer();
    QTimer *timer;


public slots:
    void timerSlot();
private:
    Downloader d;
};

#endif // MYTIMER_H
```

Listing A.13: mytimer.cpp

```
#include "mytimer.h"
#include <QtCore>
#include <QDebug>
MyTimer::MyTimer()
{
    timer = new QTimer(this);
    connect(timer,SIGNAL(timeout()), this, SLOT(timerSlot()));
```

```
    timer->start(10000);

}

MyTimer::~MyTimer()
{

}




void MyTimer::timerSlot(){
    d.checkFlag();
    if( d.getStatus())
    {
        d.resetFlag();
        d.doDownload();
    }

}
```

Listing A.14: downloader.h

```
#ifndef DOWNLOADER_H
#define DOWNLOADER_H

#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QUrl>
#include <QDateTime>
#include <QFile>
#include <QDebug>
#include <QEventLoop>

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

    void doDownload();
    void checkFlag();
    bool getStatus();
    void resetFlag();

```

```
signals:

public slots:
    void replyFinished (QNetworkReply *reply);
    void checkStatus(QNetworkReply *replay);

private:
   QNetworkAccessManager *manager;
   QNetworkAccessManager *manager2;
   QDateTime data;
   QString name;
   bool flag_ready;
   bool status;


};

#endif // DOWNLOADER_H
```

Listing A.15: downloader.cpp

```
#include "downloader.h"
const QString FLAG_ON = " True ";

Downloader::Downloader(QObject *parent) :
    QObject(parent)
{
}

void Downloader::doDownload()
{
    manager = new QNetworkAccessManager(this);

    connect(manager, SIGNAL(finished(QNetworkReply*)),
            this, SLOT(replyFinished(QNetworkReply*)));

    manager->get(QNetworkRequest(QUrl("http://planar-contact-601.appspot.com/video/view")));
}

void Downloader::replyFinished (QNetworkReply *reply)
{
    if(reply->error())
    {
        qDebug() << "ERROR!";
        qDebug() << reply->errorString();
    }
    else
    {
        qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
```

```
        qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).toDateTime().toString();
        qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).toULongLong();
        qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt();
        qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).toString();

        data = QDateTime::currentDateTime();
        name = "C:/Video/"+data.toString("yyyy-MM-dd-HH-mmss")+".mpeg";

        QFile *file = new QFile(name);
        if(file->open(QFile::Append))
        {
            file->write(reply->readAll());
            file->flush();
            file->close();
        }
        delete file;
    }

    reply->deleteLater();
}


void Downloader::checkStatus(QNetworkReply *replay)
{
    QString result(replay->readAll());
    int compare = QString::compare(result, FLAG_ON);
    if(compare == 0)
    {
        status = true;
    }
    else
    {
        status = false;
    }
    qDebug() << status;
    flag_ready = true;

}
void Downloader::checkFlag()
{
    manager = new QNetworkAccessManager(this);
    QNetworkReply *reply = manager->get(QNetworkRequest(QUrl("http://planar-contact-601.appspot.com/e
    QEventLoop loop;
    connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));

    loop.exec();
    qDebug() << "I'm looking for a new video";
    QString result(reply->readAll());
    int compare = QString::compare(result, FLAG_ON);
```

```
78      if(compare == 0)
        {
80          status = true;
            qDebug() << "Found it";
82      }
        else
84      {
            status = false;
86          qDebug() << "No new video available";
        }
88
}
90
bool Downloader::getStatus()
92{
    return status;
94}

96void Downloader::resetFlag()
{
98    status = false;
    manager2 = new QNetworkAccessManager(this);
100
    QUrl flag_url = QUrl("http://planar-contact-601.appspot.com/add/electrovalve");
102   QByteArray postData;
    postData.append("name=video&status=off");
104   manager2->post(QNetworkRequest(flag_url), postData);
}
```

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```
1 from flask import Flask

3 app = Flask(__name__)

5 from flask import request
  from flask import make_response
7
  from models import MESSAGES
9 from google.appengine.ext import ndb
  from google.appengine.api import memcache
11 import logging
  import datetime
13 import cloudstorage
```

```
import json

SECONDS_BETWEEN_UPDATES = 60
SENSOR_TIMEOUT_IN_SECONDS = 300
ELECTROVALVES_TIME_OUT = 1000
BUCKET_NAME = "/planar-contact-601.appspot.com/"
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
UPDATE_FLAG = 'database_updated_recently'

#DataPoint contains all the points that have to be plotted
class DataPoint(ndb.Model):
    #the y-axes value
    value = ndb.FloatProperty()
    #the x-axes value (autofilled with the current data)
    date = ndb.DateTimeProperty(auto_now_add=True)

    @classmethod
    # returns the list of DataPoint associated with sensor_key
    def points_for_sensor(cls, sensor_key):
        return cls.query(ancestor=sensor_key).order(-cls.date)

    @classmethod
    #returns the last DataPoint inserted
    def oldest_points(cls):
        return cls.query().order(+cls.date)

#Sensor class contains the list of used sensors
class Sensor(ndb.Model):
    @classmethod
    # returns that list
    def sensor_list(cls):
        return cls.query()


def update():
    # Check to see if we have updated recently
    update_flag = memcache.get(UPDATE_FLAG)
    if update_flag is not None:
        return
    memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
    # Delete old data points
    now = datetime.datetime.now()
    count = 0
    for point in DataPoint.oldest_points().iter():
        delta = (now - point.date).total_seconds()
        if delta > 3600:
            point.key.delete()
            count += 1
        else:
```

```
63              break
    logging.info("Deleted {} old data points".format(count))
65  # Loop through all of the sensors
    sensor_names = []
67  for sensor in Sensor.sensor_list().iter():
        # Store current values of each sensor
69      sensor_value = memcache.get(sensor.key.id() + "_value")
        if sensor_value is not None:
71          point = DataPoint(parent=sensor.key, value=sensor_value)
            point.put()
73      # Delete sensors that don't have any data
        curr_points = DataPoint.points_for_sensor(sensor.key)
75      if curr_points.count() == 0 and sensor_value is None:
            logging.info("Deleting sensor {}".format(sensor.key.id()))
77          sensor.key.delete()
        else:
79          sensor_names.append(sensor.key.id())
    # Store a list of sensor names in memcache
81  memcache.set("sensor_names", ",".join(sensor_names))

83# this method is used to obtain the json format containg the list of the
# entire sensor list made by {value, time, sensor_name} when the http
85# method is GET or to load a new value for a given sensor using the POST
# http method. In this last case the data to be passed with the POST are
87# "sensor", which indicates the sensor name, and "value", which is the
# float number of the sensor's value.
89@app.route('/', methods=['POST'])
@app.route('/sensor_values', methods=['GET', 'POST'])
91def process_values():
    if request.method == 'GET':
93      sensor_names = memcache.get('sensor_names')
        if sensor_names is None:
95          sensor_names = ",".join([sensor.key.id() for sensor in
             Sensor.sensor_list()])
97          memcache.set('sensor_names', sensor_names)
        sensor_names = sensor_names.split(",")
99      sensor_values = {}
        for sensor_name in sensor_names:
101         sensor_value = memcache.get(sensor_name + '_value')
            if sensor_value is not None:
103             sensor_values[sensor_name] = sensor_value
        return json.dumps(sensor_values)
105 elif request.method == 'POST':
        update()
107     logging.info(request.values)
        sensor_name = request.form.get('sensor')
109     sensor_value = float(request.form.get('value'))
        sensor_names = memcache.get('sensor_names')
111     if sensor_names is not None and sensor_name not in sensor_names:
```

```
            sensor = Sensor()
            sensor.key = ndb.Key('Sensor', sensor_name)
            sensor.put()
            sensor_names = sensor_name if sensor_names == "" else sensor_names +
                           "," + sensor_name
            memcache.set('sensor_names', sensor_names)
        memcache.set(sensor_name + '_value', sensor_value,
                     SENSOR_TIMEOUT_IN_SECONDS)
        return "Success\n"

# this function is accessible only using a http GET method, it returns a
# json object with the sensor's names list
@app.route('/sensor_names', methods=['GET'])
def print_names():
    return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])

# this function is accessible only through GET http method. This GET request
# has to contain two parameters one is  "sensor", the sensor name, and the
# other one is "first_timestamp", the timestamp of the first  point that has
# to be plotted (this last parameter in not mandatory, if missed it is assumed
# equal to 0.
@app.route('/graphing_data', methods=['GET'])
def graphing_data():
    first_timestamp = request.args.get('first_timestamp')
    sensor_name = request.args.get('sensor')
    if first_timestamp is None:
        first_timestamp = 0
    else:
        first_timestamp = float(first_timestamp)
    points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
    epoch = datetime.datetime(1970, 1, 1)
    res = []
    for point in points.iter():
        timestamp = (point.date - epoch).total_seconds()
        if timestamp > first_timestamp:
            _res = {"timestamp": timestamp, "value": point.value}
            res.append(_res)
        else:
            break
    res.reverse()
    return json.dumps(res)

# this function is accessible through GET http method and deletes all the
# point of sensors
@app.route('/clear', methods=['GET'])
def clear_data():
    points = DataPoint.query()
    for point in points:
        point.key.delete()
```

```
161     for sensor in Sensor.sensor_list():
            sensor.key.delete()
163     return "Success"

165 # this function is accessible through both GET and POST http methods and it
    # is in charge to store the picture of the beating plot. This function is
167 # normally used with the POST method where the image is passed through
    # "Image.jpg" field. Using a PC is also possible to update an image
169 # just using a browser, thank to the GET method implementation
    @app.route('/picture/submit', methods=['GET', 'POST'])
171 def set_picture():
        if request.method == 'POST':
173
            _file = request.files['Image.jpg']
175
            if _file:
177             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image."
                                               + _file.filename.rsplit('.', 1)[1],
179                                            mode='w', content_type="image/jpeg")
                _file.save(cloud_file)
181             cloud_file.close()
                return "Success"
183     else:
            return '''
185         <!doctype html>
            <title>Upload microscope file V.1</title>
187         <h1>Upload microscope file</h1>
            <form method="POST"
189             action=""
                role="form"
191             enctype="multipart/form-data">

193         <p><input type=file name=Image.jpg>
               <input type=submit value=Upload>
195         </form>
            '''
197
    # this function is accessible through a GET http method and it returns the image
199 # stored inside the datastore
    @app.route('/picture/view', methods=['GET'])
201 def view_picture():
        for _file in cloudstorage.listbucket(BUCKET_NAME):
203         if "microscope_image" in _file.filename:
                _file = cloudstorage.stat(_file.filename)
205             logging.info(_file.filename)
                logging.info(_file.content_type)
207             cloud_file = cloudstorage.open(_file.filename, mode='r')
                response = make_response(cloud_file.read())
209             cloud_file.close()
```

```
            response.mimetype = _file.content_type
            return response
    return "No file found"

# this function is accessible through both GET and POST http methods and it
# is in charge to store the microscope video. This function is normally used
# with the POST method where the image is passed through "Video.mp4" field.
# Using a PC is also possible to update a video just using a browser,
# thank to the GET method implementation
@app.route('/video/submit', methods=['GET', 'POST'])
def set_video():
    if request.method == 'GET':
        return '''
        <!doctype html>
        <title>Upload microscope video</title>
        <h1>Upload microscope video</h1>
        <form method="POST"
             action=""
             role="form"
             enctype="multipart/form-data">
         <p><input type=file name=Video.mp4>
         <input type=submit value=Upload>
         </form>
         '''
    else:
        _file = request.files['Video.mp4']
        if _file:
            cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_video." +
                                           _file.filename.rsplit('.', 1)[1],
                                           mode='w', content_type="video/mpeg")
            _file.save(cloud_file)
            cloud_file.close()
        return "Success"


# this function is accessible through a GET http method and it returns the
# microscope video stored inside the datastore
@app.route('/video/view', methods=['GET'])
def view_video():
    for _file in cloudstorage.listbucket(BUCKET_NAME):
        if "microscope_video" in _file.filename:
            _file = cloudstorage.stat(_file.filename)
            logging.info(_file.filename)
            logging.info(_file.content_type)
            cloud_file = cloudstorage.open(_file.filename, mode='r')
            response = make_response(cloud_file.read())
            cloud_file.close()
            response.mimetype = _file.content_type
            return response
```

```
259     return "No file found"

261 # this function is accessible through a POST http method and it is in
    # charge to store the electrovalve status inside the memcache. This http
263 # POST has to have the following parameters: "name", the name that
    # identifies the valve, "status", the status of electrovalve (on, off)
265 @app.route('/add/electrovalve', methods=['POST'])
    def add_electrovalve():
267     key = request.form.get('name')
        message = request.form.get('status')
269     if key and message:
            if message == 'on' or message == 'On' or message == 'ON':
271             MESSAGES[key] = True
                memcache.set(key, True)
273
            else:
275             MESSAGES[key] = False
                memcache.set(key, False)
277
           # ev.put()
279     return '%r' % memcache.get(key)

281 # this function accessible through a GET method returns the value of the
    # valve identified through <name> field in the URL
283 @app.route('/electrovalves/<name>', methods=['GET'])
    def get_electrovalve(name):
285     ev = memcache.get(name);
        if ev is None:
287         return ' %r ' % MESSAGES[name] or "%s not found!" % name
        else:
289         return ' %r ' % memcache.get(name)

291 @app.errorhandler(404)
    def page_not_found(e):
293     """Return a custom 404 error."""
        return 'Sorry, nothing at this URL.', 404
295



297
    if __name__ == "__main__":
299     app.run()
```

A.4 GLASSWARE

Listing A.17: MainService.java

```
package com.google.android.glass.sample.klabinterface;

import com.google.android.glass.timeline.LiveCard;
import com.google.android.glass.timeline.LiveCard.PublishMode;

import android.app.PendingIntent;
import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.os.Environment;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.IBinder;
import android.util.Log;
import android.view.animation.Animation;

import com.google.android.glass.timeline.LiveCard;
import com.googlecode.charts4j.AxisLabelsFactory;
import com.googlecode.charts4j.Data;
import com.googlecode.charts4j.GCharts;
import com.googlecode.charts4j.LineChart;
import com.googlecode.charts4j.Plot;
import com.googlecode.charts4j.Plots;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphViewSeries;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.json.JSONTokener;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.LineGraphView;


import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
```

```
import java.nio.channels.FileLock;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;


/**
 * Service owning the LiveCard living in the timeline.
 */
public class MainService extends Service {
    /** TAG associated to the LiveCard */
    private static final String LIVE_CARD_TAG = "BWH interface";
    /** TAG associated to the Menu view   */
    private static final String MENU_TAG = "Menu";
    /** TAG associated to the Temperature view  */
    private static final String TEMPERATURE_TAG = "Temperature";
    /** TAG associated to the PH view  */
    private static final String PH_TAG = "pH";
    /** TAG associated to the Video view   */
    private static final String VIDEO_TAG = "Video";
    /** TAG associated to the Beating view   */
    private static final String BEATING_TAG = "Beating";

    private Bitmap bmp;

    /** URL where the image is stored   */
    private static final String URL_IMAGE = "http://planar-contact-601.appspot.com/picture/view";



    /** Runnable which describes the task to download the Beating's graph */
    private final ImageDownloader mImageDownloader = new ImageDownloader(URL_IMAGE, this);
    /** Action is an enumerate used to implement switch-case for the extra text appended to the inte
    private static enum Action
    {
        Menu, Temperature, pH, Video, Beating
    }


    private AppDrawer mCallback;

    private LiveCard mLiveCard;

    /** HandlerThread used to launch a background thread that manages the Data updating   */
    private HandlerThread mHandlerThread;
    /** Handler used to launch a background thread that manages the Data updating   */
    private Handler mHandler;
```

```
98      /** INT associated to the Menu view request  */
        private static final int MENU = 0;
100     /** INT associated to the PH view request  */
        private static final int PH = 1;
102     /** INT associated to the Menu view request  */
        private static final int TEMPERATURE = 2;
104     /** INT associated to the Video view request  */
        private static final int VIDEO = 3;
106     /** INT associated to the Beating view request  */
        private static final int BEATING = 4;
108
        /** updating data period (in ms)  */
110     private static final long DATA_UPDATE_DELAY_MILLIS = 500;
        /** updating graph period (in ms)  */
112     private static final long GRAPH_UPDATE_DELAY_MILLIS = 500;
        /** updating video period (in ms)  */
114     private static final long FRAME_TIME_MILLIS = 100;
        private static final long VIDEO_UPDATE_DELAY_MILLIS = 60*1000; //time for video
116
        /** Runnable which describes the task to update the pH and Temperature sensors values   */
118     private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable = new UpdateSensorValuesRunn
        /** Runnable which describes the task to compute the pH and Temperature graph (starting from thei
120     private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable = new UpdateSensorGraphsRunn

122     private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable = new UpdateMicroscope

124     private static final String VIDEO_FILE_NAME = Environment.getExternalStorageDirectory()+"/microsc
        private static final String TEMP_VIDEO_FILE_NAME = Environment.getExternalStorageDirectory()+"/te
126
        /** String array for the sensors (PH and Temperature) */
128     private String[] mSensors = new String[]{PH_TAG, TEMPERATURE_TAG};

130     /*                          Hash table used for creating the graphs                              *
        /** Hash table in which the sensors values are stored  */
132     private Map<String,Double> mCurrentSensorValues;
        /** Hash table in which the graphs points are stored  */
134     private Map<String, ArrayList<DataPoint>> mSensorGraphData;
        /** Hash table in which the sensors graphs are stored */
136     private Map<String,Bitmap> mCurrentSensorGraphs;

138     private Map<String,Double> mSensorAverage;

140   //  private ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SE

142
        @Override
144     public IBinder onBind(Intent intent) {
            return null;
146     }
```

```
148     @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
150         if (mLiveCard == null) {
                AppManager.getInstance().setState(MENU);
152             mLiveCard = new LiveCard(this, LIVE_CARD_TAG);

154             // Keep track of the callback to remove it before unpublishing.
                mCallback = new AppDrawer(this);
156             mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder().addCallback(mCallback);

158             Intent menuIntent = new Intent(this, MenuActivity.class);
                menuIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
160             mLiveCard.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));
                mLiveCard.attach(this);
162             mLiveCard.publish(PublishMode.REVEAL);


164
                 /*                  Launch the task to update the data in another thread
166             mHandlerThread = new HandlerThread("myHandlerThread");
                mHandlerThread.start();
168             mHandler = new Handler(mHandlerThread.getLooper());
                DataTask task = new DataTask();
170             task.execute(this);
            } else {
172             if(intent.getStringExtra(Intent.EXTRA_TEXT)!= null) {
                    Action action = Action.valueOf(intent.getStringExtra(Intent.EXTRA_TEXT));
174                 switch (action) {
                        case Menu:
176                         Log.i(LIVE_CARD_TAG, "State = Menu");
                            AppManager.getInstance().setState(MENU);
178                         break;
                        case Beating:
180                         Log.i(LIVE_CARD_TAG, "State = Beating");
                            AppManager.getInstance().setState(BEATING);
182                         break;
                        case pH:
184                         Log.i(LIVE_CARD_TAG, "State = pH");
                            AppManager.getInstance().setState(PH);
186                         break;
                        case Temperature:
188                         Log.i(LIVE_CARD_TAG, "State = Temperature");
                            AppManager.getInstance().setState(TEMPERATURE);
190                         break;
                        case Video:
192                         Log.i(LIVE_CARD_TAG, "State = Video");
                            AppManager.getInstance().setState(VIDEO);
194                         break;
                        default:
```

```
196                         mLiveCard.navigate();
                            break;
198                 }
                }
200             else
                {
202                 mLiveCard.navigate();
                }
204         }

206         // Return START_NOT_STICKY to prevent the system from restarting the service if it is killed
            // (e.g., due to an error). It doesn't make sense to restart automatically because the
208         // stopwatch state will have been lost.
            return START_NOT_STICKY;
210     }

212     @Override
        public void onDestroy() {
214         // Stop the Task which update the beating image
            if(!mImageDownloader.isStopped())
216         {
                mImageDownloader.setIsStopped(true);
218             Log.i(BEATING_TAG,"Removed Task");
            }
220         // Stop the Task which update the sensors Graphs
            if(!mUpdateSensorGraphsRunnable.isStopped())
222         {
                mUpdateSensorGraphsRunnable.setStop(true);
224             Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Graphs");
            }
226         // Stop the Task which update the sensors value
            if(!mUpdateSensorValuesRunnable.isStopped())
228         {
                mUpdateSensorValuesRunnable.setStop(true);
230             Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Values");
            }
232         if (!mUpdateMicroscopeVideoRunnable.isStopped())
            {
234             mUpdateMicroscopeVideoRunnable.setStop(true);
                Log.i(VIDEO_TAG, "Removed Task of Uploading values");
236         }
            if (mLiveCard != null && mLiveCard.isPublished()) {
238             mLiveCard.unpublish();
                mLiveCard = null;
240         }
            super.onDestroy();
242     }

244     /**
```

```
     * Asynchronous Task for downloading the graphs and video in background
246   */
    private class DataTask extends AsyncTask<MainService,Void,Void>
248   {

250       @Override
          protected Void doInBackground(MainService... params) {
252
              Log.i(LIVE_CARD_TAG, "Loading initial data");
254
              // create the three hash tables
256           mCurrentSensorValues = new HashMap<String, Double>();
              mSensorGraphData = new HashMap<String, ArrayList<DataPoint>>();
258           mCurrentSensorGraphs = new HashMap<String, Bitmap>();
              mSensorAverage = new HashMap<String, Double>();
260
              // initializes each hash map with dummy values
262           for (String mSensor : mSensors)
              {
264               mCurrentSensorValues.put(mSensor, 0.0);
                  mSensorAverage.put(mSensor, 0.0);
266               mSensorGraphData.put(mSensor, new ArrayList<DataPoint>());
                  mCurrentSensorGraphs.put(mSensor, null);
268           }

270           Log.i(LIVE_CARD_TAG, "Download the data");

272           //uploads the value captured by PH and Temperature sensors
              //NetworkInfo ni = cm.getActiveNetworkInfo();
274           //if(ni!= null) {

276              // if (ni.isConnected()) {
                      mUpdateSensorValuesRunnable.run();
278                   // give the hash table to the Callback
                      // mCallback.setSensorValues(mCurrentSensorValues);
280                   // compute the graphs with previous values and give them to the AppDrawer
                      mUpdateSensorGraphsRunnable.run();
282
                      // give the hash table to the Callback
284                   // mCallback.setSensorGraphs(mCurrentSensorGraphs);
                      // download the beating image and give this to the AppDrawer
286                   mImageDownloader.run();

288           mUpdateMicroscopeVideoRunnable.run();
              //    }
290   //        }

292
```

```
294
            return null;
296         }
        }
298
        /**
300      * This runnable updates the sensors values taking them from the google engine
         */
302     private class UpdateSensorValuesRunnable implements Runnable
        {
304         private boolean mIsStopped = false;

306         /** It implements the task of runnable
             *
308          * @see UpdateSensorValuesRunnable
             */
310         @Override
            public void run()
312         {
                if (!isStopped())
314             {
                    /** JavaScript object in which the sensor values are temporary stored */
316                 JSONObject values = getSensorValues();
                    if (values != null)
318                 {
                        for (String mSensor : mSensors)
320                     {
                            try
322                         {
                                // update the hash table of sensor values
324                             mCurrentSensorValues.put(mSensor, values.getDouble(mSensor));

326                         }
                            catch (JSONException ignored)
328                         {}
                        }
330                 }

332                 // restart the Runnable after a given amount of time
                    mHandler.postDelayed(mUpdateSensorValuesRunnable, DATA_UPDATE_DELAY_MILLIS);
334             }
            }
336


338
            /** It is the getter function that shows the status of runnable
340          *
             * @return mIsStopped, true if the runnable is stopped, false otherwise
342          */
```

```
        public boolean isStopped()
        {
            return mIsStopped;
        }

        /** It is the setter that allows to stop the runnable
         *
         * @param isStopped, true if the user wishes to stop the runnable, false otherwise
         */
        public void setStop(boolean isStopped)
        {
            this.mIsStopped = isStopped;
        }
    }

    /** It gets the values of all of the sensors
     *
     * @return JSONOObject which contains all the values of sensors
     */
    private JSONObject getSensorValues()
    {
        try
        {
            String values = getURL("http://planar-contact-601.appspot.com/sensor_values");
            // return a JSONObject constructed by JSONTokener, which takes a source string and extra
            return new JSONObject(new JSONTokener(values));
        }
        catch (Exception e)
        {
            Log.e(LIVE_CARD_TAG,"Failed to get sensor values",e);
            return null;
        }
    }


    // Get the new data points that we will need to graph

    /** This function computes the points that have to be plotted
     *
     * @param sensor , the name of sensor (pH or Temperature)
     * @param last_timestamp , the previous value of timestamp
     * @return JSONArray, return a list of values that corresponds to the points that have to be plo
     */
    private JSONArray getDataPoints(String sensor, double last_timestamp)
    {
        try
        {
            String url = "http://planar-contact-601.appspot.com/graphing_data?sensor=" + sensor + "&
            String values = getURL(url);
```

```
392             return new JSONArray(new JSONTokener(values));
        }
394         catch (Exception e)
        {
396             Log.e(LIVE_CARD_TAG,"Failed to get data points",e);
            return null;
398         }
    }
400
    /** This method gets data from the given url website
402  *
     * @param _url, url in which the data are contained
404  * @return String, contained data from the given url
     * @throws IOException if an IO exception occurred during the download
406  */
    private String getURL(String _url) throws IOException
408 {
        URL url = new URL(_url);
410     InputStream is = url.openStream();
        int ptr;
412     StringBuffer buffer = new StringBuffer();
        while ((ptr = is.read()) != -1)
414     {
            buffer.append((char)ptr);
416     }
        return buffer.toString();
418 }

420 /** This runnable updates the graphs of pH and Temperature    */
    private class UpdateSensorGraphsRunnable implements Runnable {
422     private boolean mIsStopped = false;
        public void run()
424     {
            if (!isStopped())
426         {
//              GraphViewSeries exampleSeries = new GraphViewSeries(new GraphView.GraphViewData[] {
428 //                  new GraphView.GraphViewData(1, 2.0d)
//                  , new GraphView.GraphViewData(2, 1.5d)
430 //                  , new GraphView.GraphViewData(3, 2.5d)
//                  , new GraphView.GraphViewData(4, 1.0d)
432 //              });
//
434 //              GraphView graphView = new LineGraphView( MainService.this, "GraphViewDemo");
//
436 //              graphView.addSeries(exampleSeries);
//
438 //              mCallback.setPHGraphViewer(graphView);


440
```

```
442
444                 // Loop through each of the sensors
                    for (String curr_sensor : mSensors) {
446                     ArrayList<DataPoint> curr_data = mSensorGraphData.get(curr_sensor);
                        double lastTimestamp = curr_data.size() > 0 ? curr_data.get(curr_data.size() - 1
448                     // Get a list of the new data points for this sensor
                        JSONArray newPoints = getDataPoints(curr_sensor, lastTimestamp);
450                     for (int j = 0; j < newPoints.length(); j++) {
                            // Save each data point
452                         try {
                                JSONObject point = newPoints.getJSONObject(j);
454                             double timestamp = (Double) point.get("timestamp");
                                double value = (Double) point.get("value");
456                             curr_data.add(new DataPoint(timestamp, value));
                            } catch (JSONException e) {
458                             Log.e(LIVE_CARD_TAG, "JSON error", e);
                                //continue;
460                         }
                        }
462                     // Clear out points that are over an hour old
                        while (true) {
464                         if (curr_data.size() > 0 && curr_data.get(0).getTimestamp() < (curr_data.get
                                Log.i(LIVE_CARD_TAG, "Deleting old data point");
466                             curr_data.remove(0);
                            } else {
468                             break;
                            }
470                     }
                        // If there are no points don't show a graph
472                     if (curr_data.size() == 0) {
                            continue;
474                     }
                        // Store the timestamps and values in separate arrays for graphing
476                     ArrayList<Double> timestamps = new ArrayList<Double>();
                        ArrayList<Double> values = new ArrayList<Double>();
478                     for (DataPoint curr_point : curr_data) {
                            timestamps.add(curr_point.getTimestamp());
480                         values.add(curr_point.getValue());
                            //System.out.println("Value = " + values.get(j));
482                     }

484                     mSensorAverage.put(curr_sensor,computeAverage(values));
                        // Scale the timestamp data
486                     double maxTimestamp = Collections.max(timestamps);
                        double minTimestamp = Collections.min(timestamps);
488                     maxTimestamp *= 1.2;
                        minTimestamp *= 0.8;
```

```
490                     double intervalSize = maxTimestamp - minTimestamp;
                        for (int i1 = 0; i1 < timestamps.size(); i1++) {
492                         double currTimestamp = timestamps.get(i1);
                            currTimestamp -= minTimestamp;
494                         currTimestamp /= intervalSize;
                            currTimestamp *= 100;
496                         timestamps.set(i1, currTimestamp);
                        }
498                     // Scale the value data
                        double maxVal = Collections.max(values);
500                     double minVal = Collections.min(values);
                        maxVal *= 1.2;
502                     minVal *= 0.8;
                        intervalSize = maxVal - minVal;
504                     for (int i1 = 0; i1 < values.size(); i1++) {
                            double currVal = values.get(i1);
506                         currVal -= minVal;
                            currVal /= intervalSize;
508                         currVal *= 100;
                            values.set(i1, currVal);
510
                        }
512
                        // Data xData = Data.newData(timestamps);
514                     Data yData = Data.newData(values);


516
                        Plot plot = Plots.newPlot(yData);
518                     LineChart lineChart = GCharts.newLineChart(plot);
                        lineChart.setSize(400, 200);
520                     lineChart.addYAxisLabels(AxisLabelsFactory.newNumericRangeAxisLabels(minVal, maxV


522
                        mCurrentSensorGraphs.put(curr_sensor, getBitmapFromURL(lineChart.toURLString()));
524
                    }
526
                    mCallback.SetSensorAvg(mSensorAverage);
528                 mCallback.setSensorGraphs(mCurrentSensorGraphs);
                    Log.i("Main Service", "Graphs updated");
530
                    // restart it after a given amount of time
532                 mHandler.postDelayed(mUpdateSensorGraphsRunnable,GRAPH_UPDATE_DELAY_MILLIS);
                }
534         }

536         private double computeAverage(ArrayList<Double> values) {
                double sum = 0.0;
538             if(!values.isEmpty()){
```

```
                for(Double value : values){
                    sum += value;
                }
                return sum/values.size();
            }
            return sum;
        }

        /** It is the getter function that shows the status of runnable
         *
         * @return mIsStopped, true if the runnable is stopped, false otherwise
         */
        public boolean isStopped()
        {
            return mIsStopped;
        }

        /** It is the setter that allows to stop the runnable
         *
         * @param isStopped, true if the user wishes to stop the runnable, false otherwise
         */
        public void setStop(boolean isStopped)
        {
            this.mIsStopped = isStopped;
        }
    }
//
//
//    class DataPoint {
//        private final double mTimestamp;
//        private final double mValue;
//        DataPoint(double timestamp, double value) {
//            mTimestamp = timestamp;
//            mValue = value;
//        }
//        public double getTimestamp() {
//            return mTimestamp;
//        }
//        public double getValue() {
//            return mValue;
//        }
//    }


    /** Runnable that implements the task for downloading beating image   */
    public class ImageDownloader implements Runnable {
        private String url;
        private Context c;
        private boolean mIsStopped = false;
```

```
588
        /** Class constructor of ImageDownloader runnable
590      *
         * @param url, url link of image
592      * @param c, is the context in which the request of downloading image has been sent
         * @see
594      */
        public ImageDownloader(String url,Context c )
596     {
            this.url = url;
598         this.c = c;
        }
600
        /** It implements the task of ImageDownloader runnable
602      *
         * @see
604      */
        @Override
606     public void run()
        {
608         if(!isStopped())
            {
610             bmp = getBitmapFromURL(url);

612             mCallback.setBMP(bmp);
                Log.i(BEATING_TAG,"bmp settato");
614         }
        }
616
        /** It is the getter function that shows the status of ImageDownloader runnable
618      *
         * @return mIsStopped, true if the runnable is stopped, false otherwise
620      */
        public boolean isStopped()
622     {
            return mIsStopped;
624     }

626     /** It is the setter that allows to stop the runnable
         *
628      * @param isStopped, true if the user wishes to stop the runnable, false otherwise
         */
630     public void setIsStopped(boolean isStopped)
        {
632         this.mIsStopped = isStopped;
        }
634 }
    /** This method pulls an image from the given url
636  *
```

```
     * @param urlLink where the image is stored
     * @return Bitmap which contains the image of the graph
     * @throws java.io.IOException if an IO exception occurred during the download
     */
    public static Bitmap getBitmapFromURL(String urlLink){
        try{
            Log.i(BEATING_TAG, "start downloading image ");
            long startTime = System.currentTimeMillis();
            URL url = new URL(urlLink);
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setDoInput(true);
            connection.connect();
            InputStream inputStream = connection.getInputStream();
            Log.i(BEATING_TAG,"download completed in "
                    + ((System.currentTimeMillis() - startTime) / 1000)
                    + " sec");
            return BitmapFactory.decodeStream(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
            Log.e(BEATING_TAG, e.getMessage());
            return null;
        }
    }

    //================================================================================================

    // ================================================================================================ /

    // Runnable that updates the microscope video
    private class UpdateMicroscopeVideoRunnable implements Runnable {
        private boolean mIsStopped = false;
        public void run() {
            if (!isStopped()) {
                getMicroscopeVideo();
                mHandler.postDelayed(mUpdateMicroscopeVideoRunnable, VIDEO_UPDATE_DELAY_MILLIS);
            }
        }
        public boolean isStopped() {
            return mIsStopped;
        }
        public void setStop(boolean isStopped) {
            this.mIsStopped = isStopped;
        }
    }

    // Pull the microscope video from a URL
    private void getMicroscopeVideo() {
        try {
            URL url = new URL("http://planar-contact-601.appspot.com/video/view");
```

```
686             long startTime = System.currentTimeMillis();
                Log.i(VIDEO_TAG, "video download beginning: "+url);
688             URLConnection ucon = url.openConnection();
                ucon.setReadTimeout(0);
690             ucon.setConnectTimeout(0);
                // Define InputStreams to read from the URLConnection.
692             InputStream is = ucon.getInputStream();
                BufferedInputStream inStream = new BufferedInputStream(is, 1024*5);
694             File file = new File(TEMP_VIDEO_FILE_NAME);

696             FileOutputStream outStream = new FileOutputStream(file);

698             FileLock lock = outStream.getChannel().lock();
                byte[] buff = new byte[1024*5];
700             // Read bytes (and store them) until there is nothing more to read(-1)
                int len;
702             while ((len = inStream.read(buff)) != -1) {
                    outStream.write(buff,0,len);
704             }
                // Clean up
706
                outStream.flush();
708             lock.release();
                outStream.close();
710             inStream.close();
                Log.i(VIDEO_TAG, "download completed in "
712                     + ((System.currentTimeMillis() - startTime) / 1000)
                        + " sec");
714         }
            catch (IOException e) {
716             Log.e(VIDEO_TAG, "Failed to download microscope video", e);
            }
718     }


720
}
```

Listing A.18: AppDrawer.java

```
1  package com.google.android.glass.sample.klabinterface;

3  import com.google.android.glass.timeline.DirectRenderingCallback;
   import com.jjoe64.graphview.GraphView;
5
   import android.content.Context;
7  import android.content.Intent;
   import android.graphics.Bitmap;
9  import android.graphics.Canvas;
   import android.os.Environment;
```

```
11 import android.os.Handler;
   import android.os.SystemClock;
13 import android.util.Log;
   import android.view.SurfaceHolder;
15 import android.view.View;

17 import java.util.Map;

19 /**
    * {@link DirectRenderingCallback} used to draw the chronometer on the timeline {@link com.google.an
21  * Rendering requires that:
    * <ol>
23  * <li>a {@link SurfaceHolder} has been created through monitoring the
    *     {@link SurfaceHolder.Callback#(SurfaceHolder)} and
25  *     {@link SurfaceHolder.Callback#(SurfaceHolder)} callbacks.
    * <li>rendering has not been paused (defaults to rendering) through monitoring the
27  *     {@link com.google.android.glass.timeline.DirectRenderingCallback#renderingPaused(SurfaceHolde
    * </ol>
29  * As this class uses an inflated {@link View} to draw on the {@link SurfaceHolder}'s
    * {@link Canvas}, monitoring the
31  * {@link SurfaceHolder.Callback#(SurfaceHolder, int, int, int)} callback is also
    * required to properly measure and layout the {@link View}'s dimension.
33  */
   public class AppDrawer. implements DirectRenderingCallback {
35      /** INT associated to the Menu view request  */
        private static final int MENU = 0;
37      /** INT associated to the PH view request  */
        private static final int PH = 1;
39      /** INT associated to the Menu view request  */
        private static final int TEMPERATURE = 2;
41      /** INT associated to the Video view request  */
        private static final int VIDEO = 3;
43      /** INT associated to the Beating view request  */
        private static final int BEATING = 4;
45


47      private VideoThread mRenderThread;

49      /** Bitmap in which the beating image is stored */
        private Bitmap bmp;
51      private boolean mReady;

53      private static final String TAG = AppDrawer..class.getSimpleName();


55
        private final MainView mMainView;
57      private final BeatingView mBeatingView;
        /** View object of the GlassWear pH window */
59      private final PHViewer mPhViewer;
```

```
    private final TemperatureView mTemperatureView;
61
    private SurfaceHolder mHolder;
63  private boolean mRenderingPaused;

65  /** Hash table in which the sensors graphs are stored */
    private Map<String,Bitmap> mCurrentSensorGraphs;
67  /** Hash table in which the sensors values are stored  */
    private Map<String,Double> mCurrentSensorValues;
69  private Map<String,Double> mSensorAverage;
    private GraphView mGraphView;
71


73
    private final MainView.Listener mMainListener = new MainView.Listener() {
75


77      @Override
        public void onChange() {
79          // mMainDone = true;
             //mBeatingView.setBaseMillis(0);
81           updateRenderingState();
        }
83  };

85  /** Defines the Listener of pH viewer, it is used to communicate with
     * that viewer and allowing its viewing */
87  private final PHViewer.Listener mPhListener = new PHViewer.Listener(){
        @Override
89      /* This function is used when the {@link com.example.alik.bwhglass.PHViewer} Class wants
         * to change the view object.
91       */
        public void onChange(){
93              //state = PH;
                updateRenderingState();
95      }


97
    };
99
    private final BeatingView.Listener mBeatingListener = new BeatingView.Listener() {
101

        @Override
103     public void onChange() {
            updateRenderingState();
105     }
    };
107

    private final TemperatureView.Listener mTemperatureListener = new TemperatureView.Listener() {
```

```
109
        @Override
111     public void onChange() {
            updateRenderingState();
113     }
    };
115
    /** Defines the ManageBitmap of Beating viewer, it is used to communicate with
117  * that viewer in order to update the bitmap to be displayed*/
    private final BeatingView.ManageBitmap mBeatingBitmap = new BeatingView.ManageBitmap(){
119     @Override
        /* Getter that returns the bitmap of the beating image to be displayed
121      *
         * @return Bitmap, sensor values with the values of pH and Temperature
123      */
        public Bitmap getBitmap() {
125         return bmp;
        }
127
        public boolean isReady(){
129         return mReady;
        }
131 };

133 /** Defines the ManageDataGraph of PH viewer, it is used to communicate with
     * that viewer in order to update the hash map which contains the graph to be displayed*/
135 private final PHViewer.ManageBitmap mPHManageDataGraph = new PHViewer.ManageBitmap(){

137     public Bitmap getBitmap() {
            return mCurrentSensorGraphs.get("pH");
139     }

141     public double getAvg(){return mSensorAverage.get("pH");}

143     public boolean isReady(){
            return mReady;
145     }
    };
147
    /** Defines the ManageDataGraph of PH viewer, it is used to communicate with
149  * that viewer in order to update the hash map which contains the graph to be displayed*/
    private final TemperatureView.ManageBitmap mTemperatureManageDataGraph = new TemperatureView.Man
151
        public Bitmap getBitmap() {
153         return mCurrentSensorGraphs.get("pH");
        }
155
        public double getAvg(){return mSensorAverage.get("Temperature");}
157
```

```
        public boolean isReady(){
159         return mReady;
        }
161 };


163

    public AppDrawer.(Context context) {
165     this(new MainView(context), new BeatingView(context), new PHViewer(context), new TemperatureV
    }
167

    public AppDrawer.(MainView countDownView, BeatingView chronometerView, PHViewer phViewer, Tempera
169     mMainView = countDownView;
        mMainView.setListener(mMainListener);
171

        mBeatingView = chronometerView;
173     mBeatingView.setListener(mBeatingListener);
        mBeatingView.setManageBPM(mBeatingBitmap);
175

        mPhViewer = phViewer;
177     mPhViewer.setListener(mPhListener);
        mPhViewer.setManageBPM(mPHManageDataGraph);
179

        mTemperatureView = temperatureView;
181     mTemperatureView.setListener(mTemperatureListener);
        mTemperatureView.setManageBPM(mTemperatureManageDataGraph);
183

        mRenderThread = new VideoThread(mHolder);
185     mReady = false;
    }
187



189

    /**
191  * Uses the provided {@code width} and {@code height} to measure and layout the inflated
     * {@link MainView} and {@link BeatingView}.
193  */
    @Override
195 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
        // Measure and layout the view with the canvas dimensions.
197     int measuredWidth = View.MeasureSpec.makeMeasureSpec(width, View.MeasureSpec.EXACTLY);
        int measuredHeight = View.MeasureSpec.makeMeasureSpec(height, View.MeasureSpec.EXACTLY);
199

        mMainView.measure(measuredWidth, measuredHeight);
201     mMainView.layout(
                0, 0, mMainView.getMeasuredWidth(), mMainView.getMeasuredHeight());
203

        mBeatingView.measure(measuredWidth, measuredHeight);
205     mBeatingView.layout(
                0, 0, mBeatingView.getMeasuredWidth(), mBeatingView.getMeasuredHeight());
```

```
207
        mPhViewer.measure(measuredWidth, measuredHeight);
209     mPhViewer.layout(0, 0, mPhViewer.getMeasuredWidth(), mPhViewer.getMeasuredHeight());

211     mTemperatureView.measure(measuredWidth, measuredHeight);
        mTemperatureView.layout(
213             0, 0, mTemperatureView.getMeasuredWidth(), mTemperatureView.getMeasuredHeight());
    }
215

    /**
217  * Keeps the created {@link SurfaceHolder} and updates this class' rendering state.
     */
219 @Override
    public void surfaceCreated(SurfaceHolder holder) {
221     // The creation of a new Surface implicitly resumes the rendering.
        mRenderingPaused = false;
223     mHolder = holder;
        updateRenderingState();
225 }

227 /**
     * Removes the {@link SurfaceHolder} used for drawing and stops rendering.
229  */
    @Override
231 public void surfaceDestroyed(SurfaceHolder holder) {
        mHolder = null;
233     updateRenderingState();
    }
235

    /**
237  * Updates this class' rendering state according to the provided {@code paused} flag.
     */
239 @Override
    public void renderingPaused(SurfaceHolder holder, boolean paused) {
241     mRenderingPaused = paused;
        updateRenderingState();
243 }

245 /**
     * Starts or stops rendering according to the {@link com.google.android.glass.timeline.LiveCard}
247  */
    private void updateRenderingState() {
249     if (mHolder != null && !mRenderingPaused) {
            switch (AppManager.getInstance().getState())
251         {
                case MENU:
253                 mRenderThread.quit();
                  //  mMediaPlayer.setDisplay(null);
255                 draw(mMainView);
```

```
                mBeatingView.stop();
257             mPhViewer.stop();
                mTemperatureView.stop();
259             mMainView.start();
                break;
261         case BEATING:
              //  mMediaPlayer.setDisplay(null);
263             draw(mBeatingView);
                mMainView.stop();
265             mPhViewer.stop();
                mTemperatureView.stop();
267             mBeatingView.start();
                break;
269         case PH:
               // mMediaPlayer.setDisplay(null);
271             draw(mPhViewer);
                mMainView.stop();
273             mBeatingView.stop();
                mTemperatureView.stop();
275             mPhViewer.start();
                break;
277         case TEMPERATURE:
              //  mMediaPlayer.setDisplay(null);
279             draw(mTemperatureView);
                mMainView.stop();
281             mBeatingView.stop();
                mPhViewer.stop();
283             mTemperatureView.start();
                break;
285         case VIDEO:
              //  mMediaPlayer.setDisplay(mHolder);
287             mRenderThread.setShouldRun(true);
                mRenderThread.start();
289            // mRenderThread.run();
                mTemperatureView.stop();
291             mBeatingView.stop();
                mMainView.stop();
293             mPhViewer.stop();

295             break;
            default:
297           //  mMediaPlayer.setDisplay(null);
                draw(mMainView);
299             mPhViewer.stop();
                mBeatingView.stop();
301             mTemperatureView.stop();
                mMainView.start();
303             break;
```

```
305             }

307         } else {
                mMainView.stop();
309             mBeatingView.stop();
                mPhViewer.stop();
311             mTemperatureView.stop();
            }
313
    }
315
    /**
317  * Draws the view in the SurfaceHolder's canvas.
     */
319 private void draw(View view) {

321     Canvas canvas;

323     try {
            canvas = mHolder.lockCanvas();
325     } catch (Exception e) {
            Log.e(TAG, "Unable to lock canvas: " + e);
327         return;
        }
329     if (canvas != null) {

331         view.draw(canvas);
            mHolder.unlockCanvasAndPost(canvas);
333     }
    }
335 /** Setter for the beating graph
     *
337  * @param bmp ,  Bitmap which contains the beating graph
     */
339 public void setBMP(Bitmap bmp){

341     this.bmp = bmp;
        this.mReady = true;
343     Log.i("DRAWER", "bmp settato");
    }
345
    public void setSensorGraphs(  Map<String,Bitmap> sensorGraphs ){
347     this.mCurrentSensorGraphs = sensorGraphs;
    }
349
    public void setPHGraphViewer(  GraphView graphView ){
351     this.mGraphView = graphView;
    }
353
```

```
355     /** Setter for the sensors value hash map
        *
357     * @param currentSensorValues ,  Map<String,Double> currentSensorValues which contains the hash
        *                              map with the current value of the sensors
359     */
        public void setSensorValues(  Map<String,Double> currentSensorValues ){
361         this.mCurrentSensorValues = currentSensorValues;
        }
363


365     public  void SetSensorAvg (Map<String,Double> sensorAvg){
            this.mSensorAverage = sensorAvg;
367     }


369


371 }
```

Listing A.19: MainView.java

```
1  /*
    * Copyright (C) 2013 The Android Open Source Project
3   *
    * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
    * You may obtain a copy of the License at
7   *
    *      http://www.apache.org/licenses/LICENSE-2.0
9   *
    * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
    * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
    * limitations under the License.
15  */

17 package com.google.android.glass.sample.klabinterface;

19 import android.content.Context;
   import android.media.AudioManager;
21 import android.media.SoundPool;
   import android.os.Handler;
23 import android.os.SystemClock;
   import android.util.AttributeSet;
25 import android.view.LayoutInflater;
   import android.widget.FrameLayout;
27 import android.widget.TextView;
```

```
29 import java.text.SimpleDateFormat;
   import java.util.Date;
31 import java.util.concurrent.TimeUnit;

33 /**
    * Animated countdown going from {@code mTimeSeconds} to 0.
35  *
    * The current animation for each second is as follow:
37  *   1. From 0 to 500ms, move the TextView from {@code MAX_TRANSLATION_Y} to 0 and its alpha from
    *      {@code 0} to {@code ALPHA_DELIMITER}.
39  *   2. From 500ms to 1000ms, update the TextView's alpha from {@code ALPHA_DELIMITER} to {@code 1}.
    * At each second change, update the TextView text.
41  */
   public class MainView extends FrameLayout {

43
       /**
45      * Interface to listen for changes in the countdown.
        */
47     public interface Listener {

49         /**
           * Notified when the countdown is finished.
51         */
           public void onChange();
53     }

55     /** Time delimiter specifying when the second component is fully shown. */
       private static final long DELAY_MILLIS = 40;
57


59     private final TextView timeText;


61
       private final Handler mHandler = new Handler();
63     private final Runnable mUpdateViewRunnable = new Runnable() {

65         @Override
           public void run() {
67             if (mRunning) {
                   updateView();
69                 postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
               }
71         }
       };
73


75     private Listener mListener;
       private boolean mRunning = false;
77
```

```
    public MainView(Context context) {
79      this(context, null, 0);
    }
81
    public MainView(Context context, AttributeSet attrs) {
83      this(context, attrs, 0);
    }
85
    public MainView(Context context, AttributeSet attrs, int style) {
87      super(context, attrs, style);
        LayoutInflater.from(context).inflate(R.layout.live_card_layout, this);
89      timeText = (TextView) findViewById(R.id.timestamp);

91  }

93  /**
     * Sets a {@link Listener}.
95   */
    public void setListener(Listener listener) {
97      mListener = listener;
    }
99
    /**
101  * Returns the set {@link Listener}.
     */
103 public Listener getListener() {
        return mListener;
105 }

107 @Override
    public boolean postDelayed(Runnable action, long delayMillis) {
109     return mHandler.postDelayed(action, delayMillis);
    }
111
    /**
113  * Starts the countdown animation if not yet started.
     */
115 public void start() {
        if (!mRunning) {
117         postDelayed(mUpdateViewRunnable, 0);
        }
119     mRunning = true;
    }
121
    /**
123  * Stops the chronometer.
     */
125 public void stop() {
        if (mRunning) {
```

```
127             removeCallbacks(mUpdateViewRunnable);
               // mStarted = false;
129         }
            mRunning = false;
131     }


133


135     /**
         * Updates the view to reflect the current state of animation, visible for testing.
137      *
         * @return whether or not the count down is finished.
139      */
        void updateView() {
141         //if (mRunning) {
            timeText.setText( new SimpleDateFormat("hh:mm a").format(new Date()));
143             // updateView(millisLeft);
                if (mListener != null) {
145                 mListener.onChange();
                }
147

            //}
149     }
}
```

Listing A.20: BeatingView.java

```
    package com.google.android.glass.sample.klabinterface;
2

    import android.content.Context;
4   import android.graphics.Bitmap;
    import android.graphics.drawable.Drawable;
6   import android.os.Handler;
    import android.text.Layout;
8   import android.util.AttributeSet;
    import android.util.Log;
10  import android.view.LayoutInflater;
    import android.widget.FrameLayout;
12  import android.widget.ImageView;
    import android.widget.TextView;
14


16  /**
     * View used to display draw a running Chronometer.
18   *
     * This code is greatly inspired by the Android's Chronometer widget.
20   */
    public class BeatingView extends FrameLayout {
22
```

```
    boolean mState = false;
    private Bitmap bmp;

    /**
     * Interface to listen for changes on the view layout.
     */
    public interface Listener {
        /** Notified of a change in the view. */
        public void onChange();
    }

    /** About 24 FPS, visible for testing. */
    static final long DELAY_MILLIS = 41;

    private ImageView beatingImage;
    private final TextView mTitle;

    private final Handler mHandler = new Handler();
    private final Runnable mUpdateTextRunnable = new Runnable() {

        @Override
        public void run() {
            if (mRunning) {
                updateText();
                postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
            }
        }
    };

    private boolean mRunning;

    public interface ManageBitmap{
        public Bitmap getBitmap();

        public boolean isReady();
    }


    private Listener mChangeListener;

    private ManageBitmap mManage;

    public BeatingView(Context context) {
        this(context, null, 0);
    }

    public BeatingView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }
```

```
72
    public BeatingView(Context context, AttributeSet attrs, int style) {
74      super(context, attrs, style);
        LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
76      mTitle = (TextView) findViewById(R.id.message);
        beatingImage = (ImageView) findViewById(R.id.image_left);
78      mTitle.setText("Beating");
        int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/la
80      beatingImage.setImageResource(id);

82

    }
84

86  /**
     * Sets a {@link Listener}.
88   */
    public void setListener(Listener listener) {
90      mChangeListener = listener;
    }
92

    /**
94   * Returns the set {@link Listener}.
     */
96  public Listener getListener() {
        return mChangeListener;
98  }

100 /**
     * Starts the chronometer.
102  */
    public void start() {
104     if (!mRunning) {
            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
106     }
        mRunning = true;
108 }

110 /**
     * Stops the chronometer.
112  */
    public void stop() {
114     if (mRunning) {
            removeCallbacks(mUpdateTextRunnable);
116     }
        mRunning = false;
118 }

120 @Override
```

```
    public boolean postDelayed(Runnable action, long delayMillis) {
        return mHandler.postDelayed(action, delayMillis);
    }

    @Override
    public boolean removeCallbacks(Runnable action) {
        mHandler.removeCallbacks(action);
        return true;
    }
    /**
     * Sets a {@link Listener}.
     */
    public void setManageBPM(ManageBitmap manager) {
        mManage = manager;
    }


    /**
     * Updates the value of the chronometer, visible for testing.
     */
    void updateText() {
        if(mManage.isReady())
        {
            Log.i("viewer", "ce prova");
            bmp = mManage.getBitmap();
            beatingImage.setImageBitmap(bmp);
        }
      //  else
        //{
          //  int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawab
           // beatingImage.setImageResource(id);
        //}
        if (mChangeListener != null) {
            mChangeListener.onChange();
        }
    }



}
```

Listing A.21: PHViewer.java

```
package com.google.android.glass.sample.klabinterface;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.drawable.Drawable;
```

```
import android.os.Handler;
import android.text.Layout;
import android.util.AttributeSet;
import android.util.Log;
import android.view.LayoutInflater;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;


/**
 * View used to display draw a running Chronometer.
 *
 * This code is greatly inspired by the Android's Chronometer widget.
 */
public class PHViewer extends FrameLayout {

    private Context context;
    private Bitmap bmp;

    /**
     * Interface to listen for changes on the view layout.
     */
    public interface Listener {
        /** Notified of a change in the view. */
        public void onChange();
    }

    /** About 24 FPS, visible for testing. */
    static final long DELAY_MILLIS = 41;

    private ImageView beatingImage;
    private final TextView mTextTitle;
    private TextView AvgView;

    private final Handler mHandler = new Handler();
    private final Runnable mUpdateTextRunnable = new Runnable() {

        @Override
        public void run() {
            if (mRunning) {
                updateText();
                postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
            }
```

```
55          }
    };
57
    private boolean mRunning;
59
    public interface ManageBitmap{
61          public Bitmap getBitmap();

63          public boolean isReady();

65          public double getAvg();
    }
67


69      private Listener mChangeListener;

71      private ManageBitmap mManage;

73      public PHViewer(Context context) {
            this(context, null, 0);
75      }

77      public PHViewer(Context context, AttributeSet attrs) {
            this(context, attrs, 0);
79      }

81      public PHViewer(Context context, AttributeSet attrs, int style) {
            super(context, attrs, style);
83          LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
            beatingImage = (ImageView) findViewById(R.id.image_left);
85          mTextTitle = (TextView) findViewById(R.id.message);
            AvgView = (TextView) findViewById(R.id.avg);
87          mTextTitle.setText("PH");
            int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/lay
89          beatingImage.setImageResource(id);
//          int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/l
91  //          beatingImage.setImageResource(id);


93
    }
95


97      /**
         * Sets a {@link Listener}.
99       */
    public void setListener(Listener listener) {
101         mChangeListener = listener;
    }
103
```

```
    /**
105  * Returns the set {@link Listener}.
     */
107 public Listener getListener() {
        return mChangeListener;
109 }

111 /**
     * Starts the chronometer.
113  */
    public void start() {
115     if (!mRunning) {
            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
117     }
        mRunning = true;
119 }

121 /**
     * Stops the chronometer.
123  */
    public void stop() {
125     if (mRunning) {
            removeCallbacks(mUpdateTextRunnable);
127     }
        mRunning = false;
129 }

131 @Override
    public boolean postDelayed(Runnable action, long delayMillis) {
133     return mHandler.postDelayed(action, delayMillis);
    }
135
    @Override
137 public boolean removeCallbacks(Runnable action) {
        mHandler.removeCallbacks(action);
139     return true;
    }
141 /**
     * Sets a {@link Listener}.
143  */
    public void setManageBPM(ManageBitmap manager) {
145     mManage = manager;
    }
147


149 /**
     * Updates the value of the chronometer, visible for testing.
151  */
    void updateText() {
```

```
153         if(mManage.isReady())
            {
155             Log.i("PH", "ce prova");
                bmp = mManage.getBitmap();
157             AvgView.setText(Double.toString(mManage.getAvg()));
                beatingImage.setImageBitmap(bmp);
159


161         }
            //  else
163         //{
            //  int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable
165         // beatingImage.setImageResource(id);
            //}
167         if (mChangeListener != null) {
                mChangeListener.onChange();
169         }
        }
171


173


175 }
```

Listing A.22: TemperatureView.java

```
    package com.google.android.glass.sample.klabinterface;
2

        import android.content.Context;
4       import android.graphics.Bitmap;
        import android.graphics.drawable.Drawable;
6       import android.os.Handler;
        import android.text.Layout;
8       import android.util.AttributeSet;
        import android.util.Log;
10      import android.view.LayoutInflater;
        import android.widget.FrameLayout;
12      import android.widget.ImageView;
        import android.widget.TextView;
14


16  /**
     * View used to display draw a running Chronometer.
18   *
     * This code is greatly inspired by the Android's Chronometer widget.
20   */
    public class TemperatureView extends FrameLayout {
22

        private Bitmap bmp;
```

```
24      private TextView AvgView;

26      /**
         * Interface to listen for changes on the view layout.
28       */
        public interface Listener {
30          /** Notified of a change in the view. */
            public void onChange();
32      }

34      /** About 24 FPS, visible for testing. */
        static final long DELAY_MILLIS = 41;
36
        private ImageView beatingImage;
38      private final TextView mTitle;

40      private final Handler mHandler = new Handler();
        private final Runnable mUpdateTextRunnable = new Runnable() {
42
            @Override
44          public void run() {
                if (mRunning) {
46                  updateText();
                    postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
48              }
            }
50      };

52      private boolean mRunning;

54      public interface ManageBitmap{
            public Bitmap getBitmap();
56
            public boolean isReady();
58
            public double getAvg();
60      }


62
        private Listener mChangeListener;
64
        private ManageBitmap mManage;
66
        public TemperatureView(Context context) {
68          this(context, null, 0);
        }
70
        public TemperatureView(Context context, AttributeSet attrs) {
72          this(context, attrs, 0);
```

```
    }

    public TemperatureView(Context context, AttributeSet attrs, int style) {
        super(context, attrs, style);
        LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
        beatingImage = (ImageView) findViewById(R.id.image_left);
        mTitle = (TextView) findViewById(R.id.message);
        AvgView = (TextView) findViewById(R.id.avg);

        mTitle.setText("Temperature");
        int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/lay
        beatingImage.setImageResource(id);


    }


    /**
     * Sets a {@link Listener}.
     */
    public void setListener(Listener listener) {
        mChangeListener = listener;
    }

    /**
     * Returns the set {@link Listener}.
     */
    public Listener getListener() {
        return mChangeListener;
    }

    /**
     * Starts the chronometer.
     */
    public void start() {
        if (!mRunning) {
            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
        }
        mRunning = true;
    }

    /**
     * Stops the chronometer.
     */
    public void stop() {
        if (mRunning) {
            removeCallbacks(mUpdateTextRunnable);
        }
        mRunning = false;
```

```
122     }
124     @Override
        public boolean postDelayed(Runnable action, long delayMillis) {
126         return mHandler.postDelayed(action, delayMillis);
        }
128
        @Override
130     public boolean removeCallbacks(Runnable action) {
            mHandler.removeCallbacks(action);
132         return true;
        }
134     /**
         * Sets a {@link Listener}.
136      */
        public void setManageBPM(ManageBitmap manager) {
138         mManage = manager;
        }
140


142     /**
         * Updates the value of the chronometer, visible for testing.
144      */
        void updateText() {
146         if(mManage.isReady())
            {
148             Log.i("Temperature", "ce prova");
                bmp = mManage.getBitmap();
150             AvgView.setText(Double.toString(mManage.getAvg()));
                beatingImage.setImageBitmap(bmp);
152

            }
154         //  else
            //{
156         //  int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawabl
            // beatingImage.setImageResource(id);
158         //}
            if (mChangeListener != null) {
160             mChangeListener.onChange();
            }
162     }


164


166
}
```

Listing A.23: VideoPlayerActivity.java

```
1  package com.google.android.glass.sample.klabinterface;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.content.res.AssetManager;
6  import android.net.Uri;
7  import android.os.Bundle;
8  import android.os.Environment;
9  import android.util.Log;
10 import android.view.Menu;
11 import android.view.MenuItem;
12 import android.widget.MediaController;
13 import android.widget.VideoView;
14
15 import java.io.File;
16 import java.io.FileOutputStream;
17 import java.io.IOException;
18 import java.io.InputStream;
19 import java.io.OutputStream;
20 public class VideoPlayerActivity extends Activity {
21    private static final int VIDEO_PLAY_REQUEST_CODE = 200;
22    private static final String TAG = "VIDEO_TAG";
23    public void onCreate(Bundle savedInstanceState) {
24        super.onCreate(savedInstanceState);
25        String filepath;
26        Bundle extras = getIntent().getExtras();
27        if (extras != null)
28            filepath = extras.getString("filepath");
29        else {
30            filepath = copyAsset(VIDEO_FILE_NAME);
31        }
32
33        Intent i = new Intent();
34        i.setAction("com.google.glass.action.VIDEOPLAYER");
35        i.putExtra("video_url", filepath);
36        startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);
37    }
38    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
39        if (requestCode == VIDEO_PLAY_REQUEST_CODE )
40            finish();
41    }
42    String copyAsset(String filename) {
43        final String PATH = Environment.getExternalStorageDirectory().toString() + "/myvideoapps/";
44        File dir = new File(PATH);
45        if (!dir.exists()) {
46            if (!dir.mkdirs()) {
47                Log.v(TAG, "ERROR: Creation of directory " + PATH + " on sdcard failed");
48                return null;
49            } else {
```

```
                Log.v(TAG, "Created directory " + PATH + " on sdcard");
            }
        }
        if (!(new File( PATH + filename).exists())) {
            try {
                AssetManager assetManager = getAssets();
                InputStream in = assetManager.open(filename);
                OutputStream out = new FileOutputStream(PATH + filename);
                byte[] buf = new byte[1024];
                int len;
                while ((len = in.read(buf)) > 0) {
                    out.write(buf, 0, len);
                }
                in.close();
                out.close();
            } catch (IOException e) {
                Log.e(TAG, "Was unable to copy " + filename + e.toString());
                return null;
            }
        }
        return PATH + filename;
    }
}
```

Listing A.24: MenuActivity.java

```
package com.google.android.glass.sample.klabinterface;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

import java.lang.Runnable;

/**
 * Activity showing the stopwatch options menu.
 */
public class MenuActivity extends Activity {

    /** INT associated to the Menu view request  */
    private static final int MENU = 0;
    /** INT associated to the PH view request  */
    private static final int PH = 1;
```

```
24      /** INT associated to the Menu view request  */
        private static final int TEMPERATURE = 2;
26      /** INT associated to the Video view request  */
        private static final int VIDEO = 3;
28      /** INT associated to the Beating view request  */
        private static final int BEATING = 4;
30
        private final Handler mHandler = new Handler();
32      private int state = 0;

34      @Override
        public void onAttachedToWindow() {
36          super.onAttachedToWindow();
            openOptionsMenu();
38      }

40      @Override
        public boolean onCreateOptionsMenu(Menu menu) {
42          MenuInflater inflater = getMenuInflater();
            inflater.inflate(R.menu.stopwatch, menu);
44          return true;
        }
46
        @Override
48      public boolean onPreparePanel(int featureId, View view, Menu menu) {
            AppManager appManager = AppManager.getInstance();
50
            boolean initialView = appManager.getState() == MENU;
52          boolean imageView = appManager.getState() == PH ||
                    appManager.getState() == TEMPERATURE ||
54                  appManager.getState() == BEATING ||
                    appManager.getState() == VIDEO;
56
            setOptionsMenuState(menu, R.id.action_back, imageView);
58          setOptionsMenuState(menu, R.id.action_view_ph, initialView);
            setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
60          setOptionsMenuState(menu, R.id.action_view_video, initialView);
            setOptionsMenuState(menu, R.id.action_view_beating, initialView);
62          setOptionsMenuState(menu, R.id.action_stop, true);

64          return true;
        }
66
        @Override
68      public boolean onOptionsItemSelected(MenuItem item) {
            // Handle item selection.
70          switch (item.getItemId()) {
                case R.id.action_stop:
72                  // Stop the service at the end of the message queue for proper options menu
```

```
                // animation. This is only needed when starting a new Activity or stopping a Service
74              // that published a LiveCard.
                post(new Runnable() {
76
                    @Override
78                  public void run() {
                        stopService(new Intent(MenuActivity.this, StopwatchService.class));
80                  }
                });
82              return true;
            case R.id.action_view_beating:
84              handleViewBeating();
                return true;
86          case R.id.action_back:
                handleViewBack();
88              return true;
            case R.id.action_view_temperature:
90              handleViewTemperature();
                return true;
92          case R.id.action_view_ph:
                handleViewPh();
94              return true;
            case R.id.action_view_video:
96              handleViewVideo();
                return true;
98          default:
                return super.onOptionsItemSelected(item);
100     }
    }
102
    private void handleViewVideo() {
104     Toast.makeText(this, "Video", Toast.LENGTH_SHORT).show();
        // launch a new thread for starting a new live card
106     mHandler.post(new Runnable() {
            @Override
108         public void run() {
                Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
110             intent.putExtra(Intent.EXTRA_TEXT, "Video");
                startService(intent);
112         }
        });
114 }

116 private void handleViewTemperature() {
        Toast.makeText(this, "Temperature", Toast.LENGTH_SHORT).show();
118     // launch a new thread for starting a new live card
        mHandler.post(new Runnable() {
120         @Override
            public void run() {
```

```
122                 Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
                    intent.putExtra(Intent.EXTRA_TEXT, "Temperature");
124                 startService(intent);
                }
126         });
    }
128
    private void handleViewPh() {
130         Toast.makeText(this, "Ph", Toast.LENGTH_SHORT).show();
            // launch a new thread for starting a new live card
132         mHandler.post(new Runnable() {
                @Override
134             public void run() {
                    Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
136                 intent.putExtra(Intent.EXTRA_TEXT, "pH");
                    startService(intent);
138             }
            });
140 }

142 private void handleViewBack() {
        Toast.makeText(this, "Menu", Toast.LENGTH_SHORT).show();
144     // launch a new thread for starting a new live card
        mHandler.post(new Runnable() {
146         @Override
            public void run() {
148             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
                intent.putExtra(Intent.EXTRA_TEXT, "Menu");
150             startService(intent);
            }
152     });
    }
154
    @Override
156 public void onOptionsMenuClosed(Menu menu) {
        // Nothing else to do, closing the Activity.
158     finish();
    }
160
    /**
162  * Posts a {@link Runnable} at the end of the message loop, overridable for testing.
     */
164 protected void post(Runnable runnable) {
        mHandler.post(runnable);
166 }

168 /**
     * The function handle the request to view the beating plot
170  */
```

```
    private void handleViewBeating() {
        Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
        // launch a new thread for starting a new live card
        mHandler.post(new Runnable() {
            @Override
            public void run() {
                Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
                intent.putExtra(Intent.EXTRA_TEXT, "Beating");
                startService(intent);
            }
        });
    }

    private static void setOptionsMenuState(Menu menu, int menuItemId, boolean enabled){
        MenuItem menuItem = menu.findItem(menuItemId);
        menuItem.setVisible(enabled);
        menuItem.setEnabled(enabled);
    }

}
```

Listing A.25: AppManager.java

```
package com.google.android.glass.sample.klabinterface;

/**
 * Created by alik on 12/2/2014.
 */
public class AppManager {

    private int state;

    private static  AppManager instance = new AppManager();

    public static AppManager getInstance(){
        return instance;
    }

    public void setState(int value){
        state = value;
    }

    public int getState(){
        return state;
    }

}
```

Listing A.26: DataPoint.java

```
package com.google.android.glass.sample.klabinterface;

/**
 * Created by Buso on 28/10/2014.
 */
class DataPoint {
    private final double mTimestamp;
    private final double mValue;
    DataPoint(double timestamp, double value) {
        mTimestamp = timestamp;
        mValue = value;
    }
    public double getTimestamp() {
        return mTimestamp;
    }
    public double getValue() {
        return mValue;
    }
}
```

[7]

LIST OF FIGURES

Figure 1 *XCEL* project (*Body-on-a-chip*) . . . 1
Figure 2 Block diagram of the system . . . 2
Figure 3 Glasswear's Block Diagram . . . 3
Figure 4 Drive Electrovalves Steps . . . 4
Figure 5 *Back* menu item . . . 4
Figure 6 *Exit* menu item . . . 4
Figure 7 The Board . . . 5
Figure 8 Storing Microscope Video Program's Icon . . . 6
Figure 9 Storing Microscope Video Console . . . 7
Figure 10 Video Stored in the Folder . . . 7
Figure 11 Multisensor - Size Comparison . . . 11
Figure 12 Multisensor - Scheme . . . 12
Figure 13 PH and Temperature Sensors from an Optical Image . . . 13
Figure 14 Typical pH-sensor transfer function . . . 14
Figure 15 Conditioning circuit for pH sensor . . . 15
Figure 16 Error caused by Amplifier's Input Bias Current . . . 15
Figure 17 Low-Pass Filter Schematic . . . 17
Figure 18 Low-Pass Filter Frequency Response . . . 17
Figure 19 Acquisition Path for pH Sensor . . . 18
Figure 20 Conditioning Circuit for Temperature Sensor . . . 19
Figure 21 Acquisition Path for pH Sensor . . . 20
Figure 22 Electrovalves . . . 21
Figure 23 Driver for electrovalves . . . 21
Figure 24 Electrovalves Driver Circuit . . . 22
Figure 25 DC-DC circuit . . . 23
Figure 26 Relay circuit . . . 24
Figure 27 Electrovalves Supply Circuit . . . 25
Figure 28 Final system mounted on a PCB . . . 26
Figure 29 Schematic of Complete Circuit . . . 27
Figure 30 *DC-DC* High Speed Switching Path . . . 28
Figure 31 *PCB* Layout of *DC-DC* . . . 28
Figure 32 PCB of the System . . . 30
Figure 33 High Level System's Block Diagram . . . 32
Figure 34 Distribution of Internet of Things . . . 33
Figure 35 The *IoT* Evolution Prediction . . . 34
Figure 36 The *Beaglebone Black* . . . 34
Figure 37 Embedded Linux Firmware - Blocks Subdivision . . . 36
Figure 38 Bash Script Flow Chart . . . 37
Figure 39 Sensor UML Description . . . 39

Figure 40 Google App Engine Logo 43
Figure 41 Flask Logo 44
Figure 42 Picture Submitting from the Browser 45
Figure 43 Video Submitting from the Browser 46
Figure 44 Driver for LED 51
Figure 45 LEDs experiments board 52
Figure 46 Circuit mounted on breadboard top view 53
Figure 47 Circuit mounted on breadboard side view 54

LIST OF LISTINGS

A.1 KlabFirware (bash) . . . 55
A.2 SensorAcquiring (bash) . . . 56
A.3 Pins Setting . . . 56
A.4 ElectrovalvesDriver.cpp . . . 57
A.5 sensor_aquiring.cpp . . . 59
A.6 compute_beating.cpp . . . 62
A.7 Pins Setting . . . 63
A.8 HTTP.h . . . 64
A.9 HTTP.cpp . . . 65
A.10 main.cpp . . . 66
A.11 VideoStoring.pro . . . 66
A.12 mytimer.h . . . 67
A.13 mytimer.cpp . . . 67
A.14 downloader.h . . . 68
A.15 downloader.cpp . . . 69
A.16 Main Script of Server . . . 71
A.17 MainService.java . . . 77
A.18 AppDrawer.java . . . 92
A.19 MainView.java . . . 100
A.20 BeatingView.java . . . 103
A.21 PHViewer.java . . . 106
A.22 TemperatureView.java . . . 110
A.23 VideoPlayerActivity.java . . . 113
A.24 MenuActivity.java . . . 115
A.25 AppManager.java . . . 119
A.26 DataPoint.java . . . 119

BIBLIOGRAPHY

[1] Andrea Cavallini Camilla Baj-Rossi Sara Ghoreishizadeh Giovanni De Micheli Sandro Carrara. Design, fabrication, and test of a sensor array for perspective biosensing in chronic pathologies. *IEEE Biomedical Circuits and System Conference*, 2012. URL http://si2.epfl.ch/~demichel/publications/archive/2012/BioCAS_2012_Andrea.pdf.

[2] Texas Instruments. Pcb design guidelines for reduced emi. *Application Note*, November 1999. URL http://www.ti.com/lit/an/szza009/szza009.pdf.

[3] Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux*. Wiley, 2014. ISBN 1118935128. URL http://www.exploringbeaglebone.com/.

[4] Avi Baum. An-1852 designing with ph electrodes. *Application Note, Texas Instruments*, July 2014. URL http://www.ti.com/lit/wp/swry009/swry009.pdf.

[5] Dave Evans. The internet of things, how the next evolution of the internet is changing everything. *Cisco Systems*, April 2011. URL http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[6] Charles Severance. *Using Google App Engine*. O'Reilly Media, May 2009. ISBN 0596555806.

[7] Texas Instruments. An-1852 designing with ph electrodes. *Application Note*, September 2008–Revised April 2013. URL http://www.ti.com/lit/an/snoa529a/snoa529a.pdf.