

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* (Cambridge, 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 glass-wear¹.

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 containing four 3D fully-functional *organs-on-a-chip*.

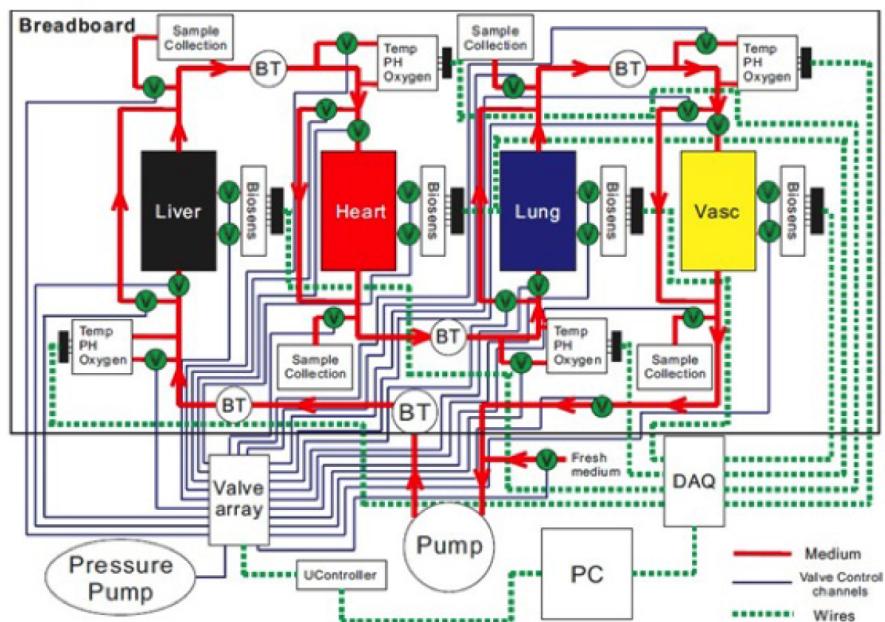


Figure 1: XCEL project (*Body-on-a-chip*)

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, lung, 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).

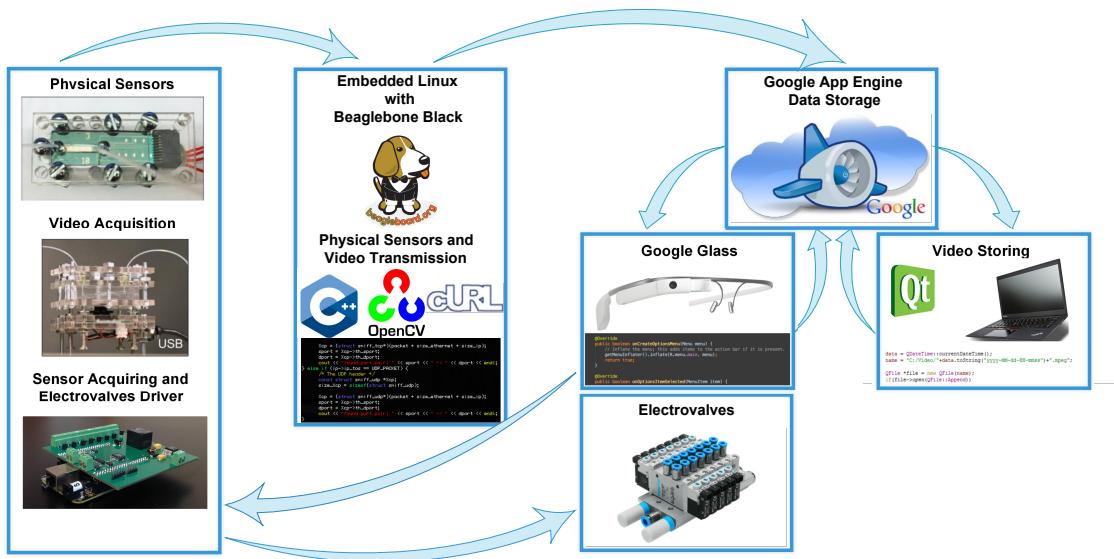


Figure 2: Block diagram of the system

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. The *View Beating* card shows the graph of the beating associated to the video.

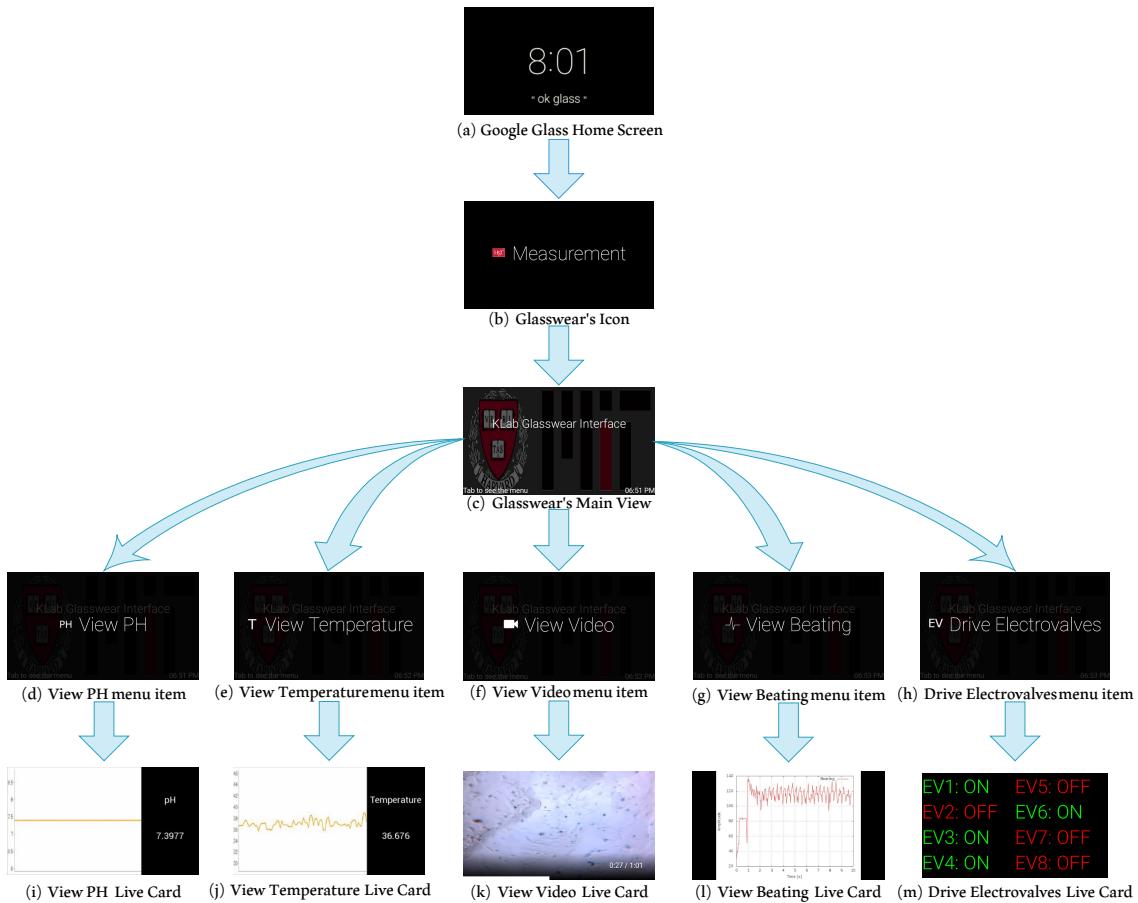


Figure 3: Glasswear's Block Diagram

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).



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.

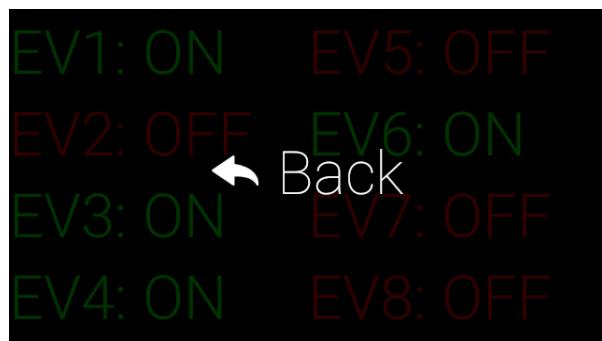


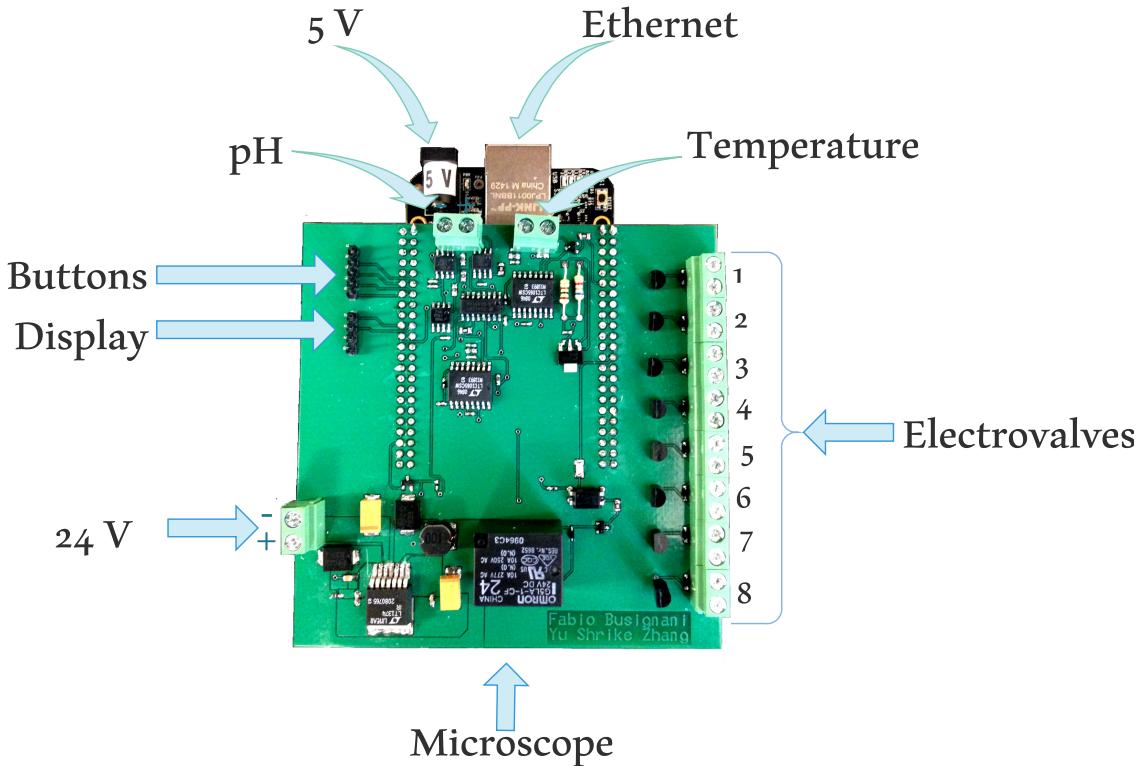
Figure 5: *Back* menu item

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 Beaglebone Black and by the conditioning 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 sensor²;
- *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 require 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 from 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*.

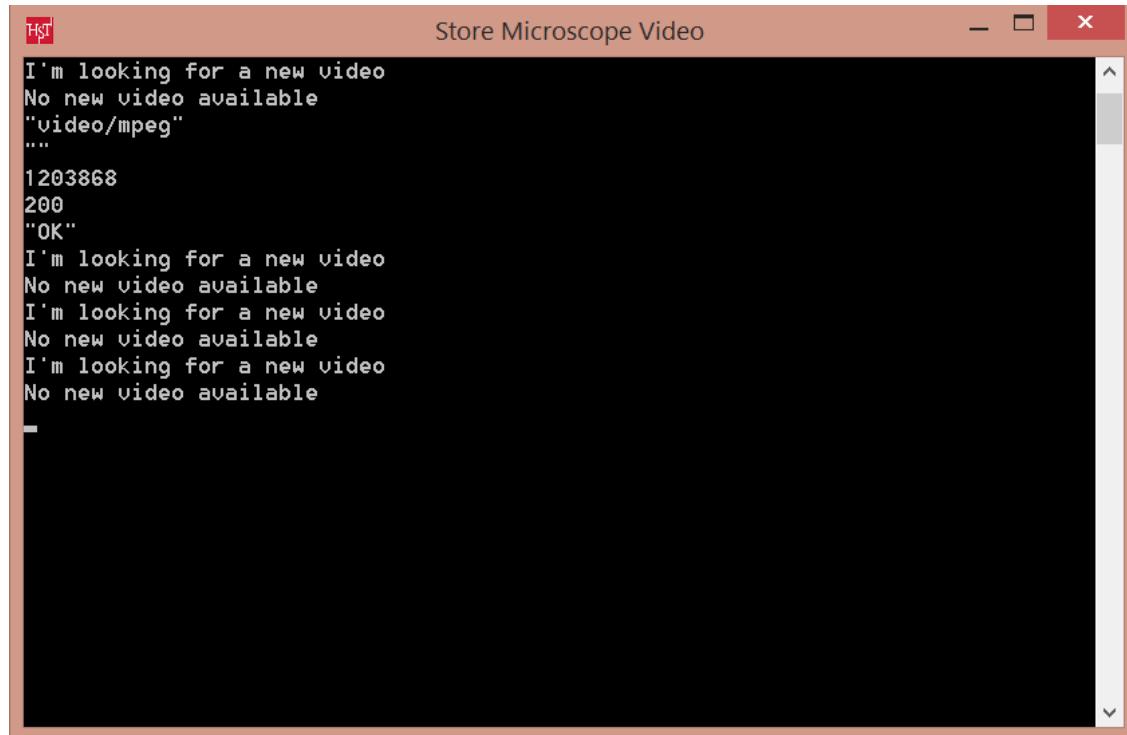
The program is very easy to use, the user just has to run the executable (Fig.8).



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 date 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.



```
I'm looking for a new video
No new video available
"video/mpeg"
...
1203868
200
"OK"
I'm looking for a new video
No new video available
I'm looking for a new video
No new video available
I'm looking for a new video
No new video available
```

Figure 9: Storing Microscope Video Console

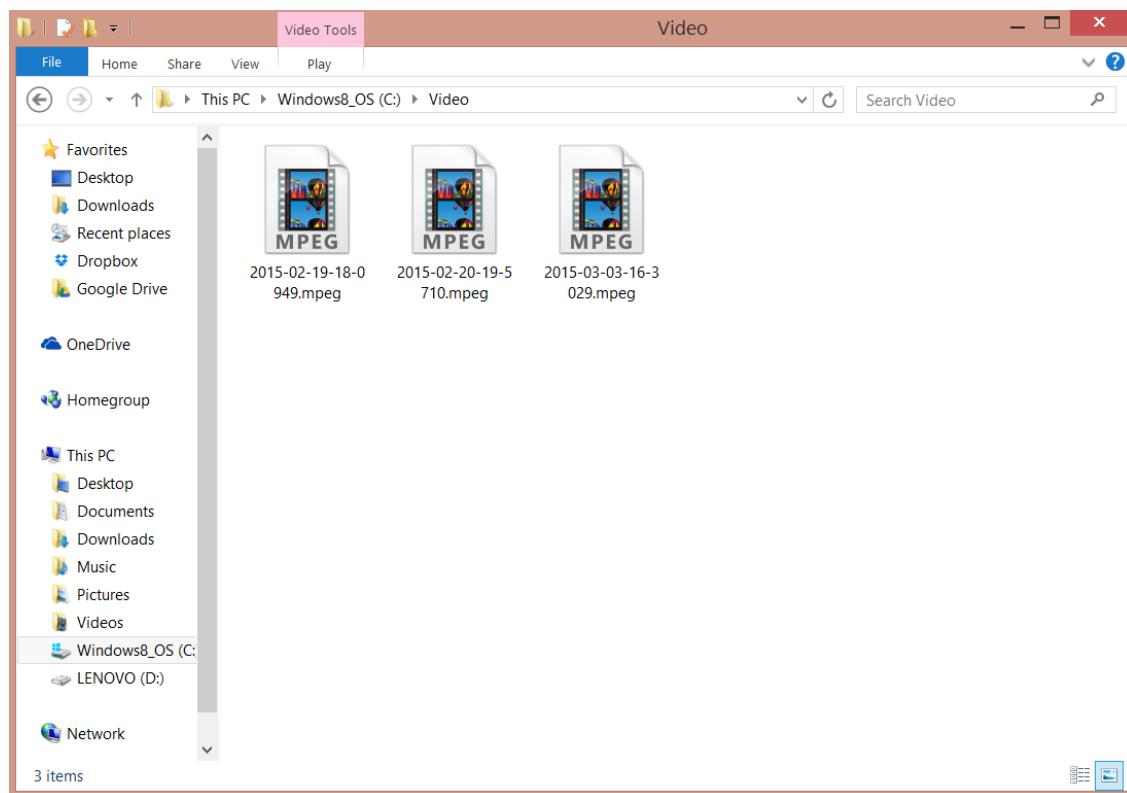


Figure 10: Video Stored in the Folder

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.

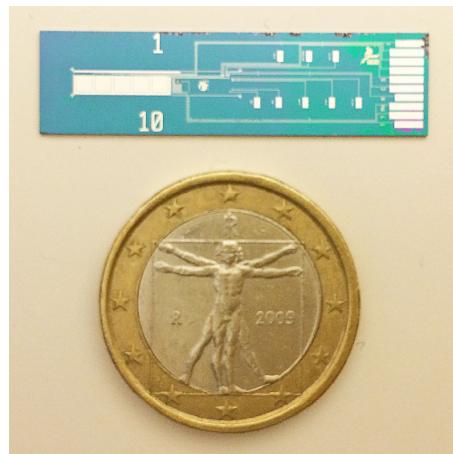


Figure 11: Multisensor - Size Comparison

The (Fig.11) shows the sensor and highlights its very small dimension ($35.1 \times 9.3 \text{ mm}$). 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.

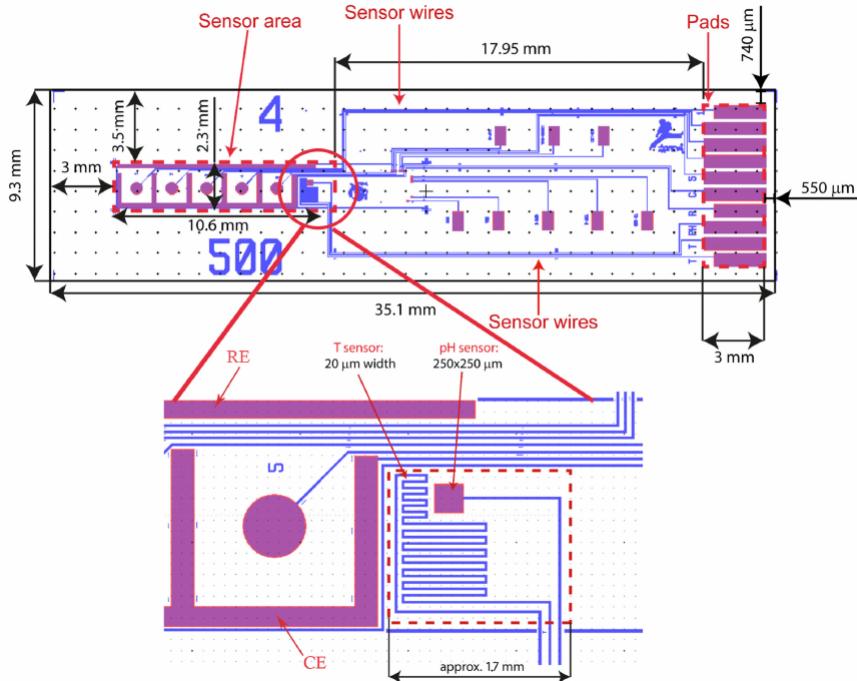


Figure 12: Multisensor - Scheme

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 $\pm 0.001 \text{ }^{\circ}\text{C}$) 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).

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

Where:

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

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

$$A = \alpha + \frac{\alpha \cdot \delta}{100}, \quad (2a)$$

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

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

And $\alpha = 0.003921 \Omega/\text{ }^\circ\text{C}$, $\beta = 0$ for positive temperature and $\delta = 1.49$.

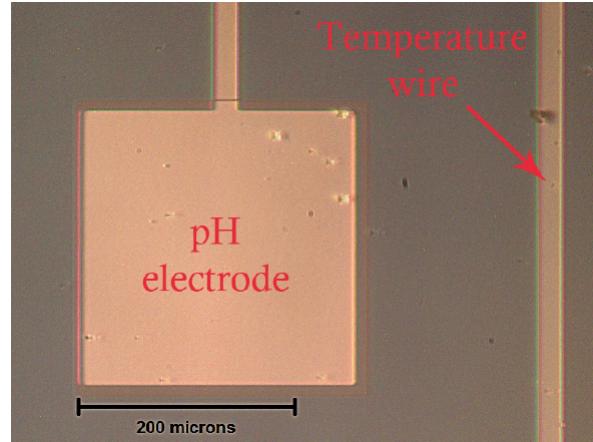


Figure 13: PH and Temperature Sensors from an Optical Image

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.

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

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

- $\text{pH}(X)$, is the pH value of unknown solution;
- $\text{pH}(S)$, 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 ($9.6485309 \cdot 10^4 \text{ C mol}^{-1}$);
- R , is the universal gas constant ($8.314510 \text{ J K}^{-1} \text{ mol}^{-1}$);
- 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).

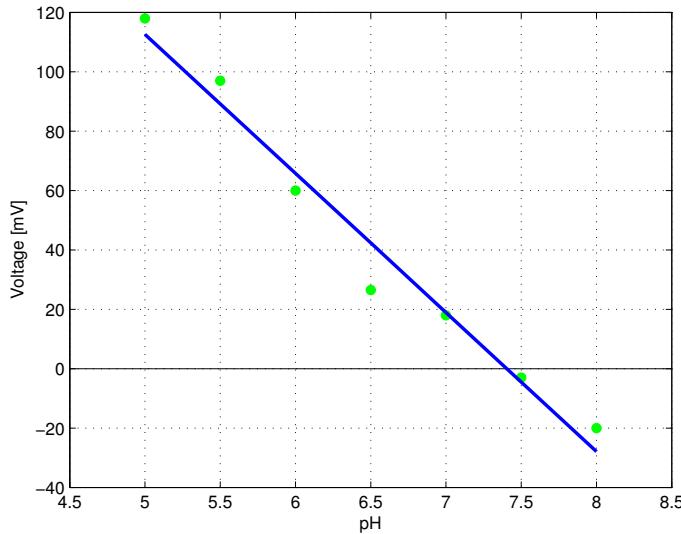


Figure 14: Typical pH-sensor transfer function

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. First 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 $10\text{ K}\Omega$ 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.

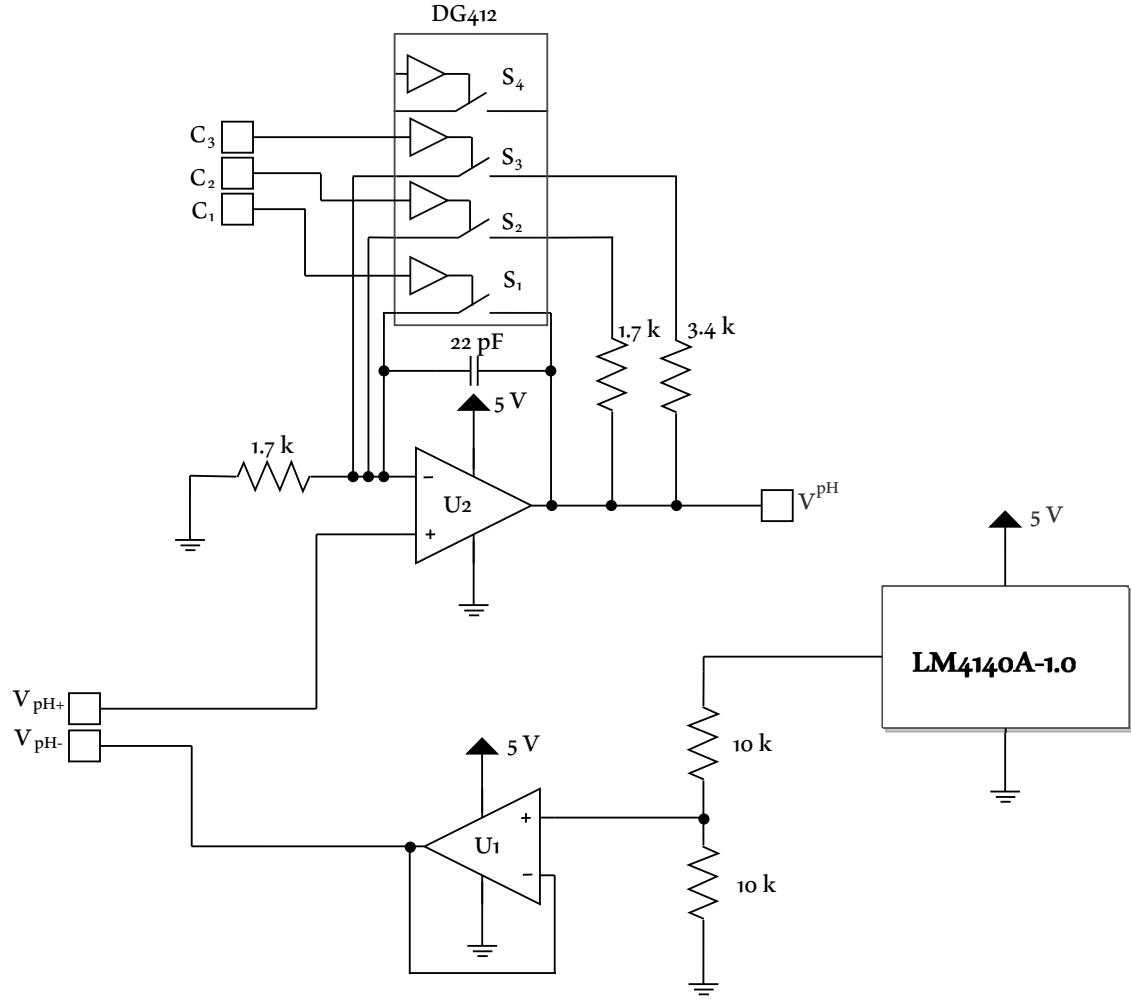


Figure 15: Conditioning circuit for pH sensor

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:

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

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 ($3 \pm 17 fA$).

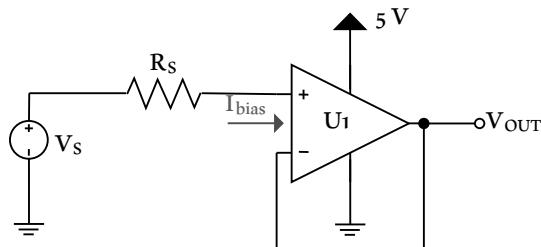


Figure 16: Error caused by Amplifier's Input Bias Current

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 $75 \pm 470 \text{ fV}$, due to the bias current of the operational amplifier and R_{ON} of *DG412* (25Ω). That voltage combined with the *LMP7721* offset (which is very higher than the first one) is approximately equal to $26 \mu\text{V}$.

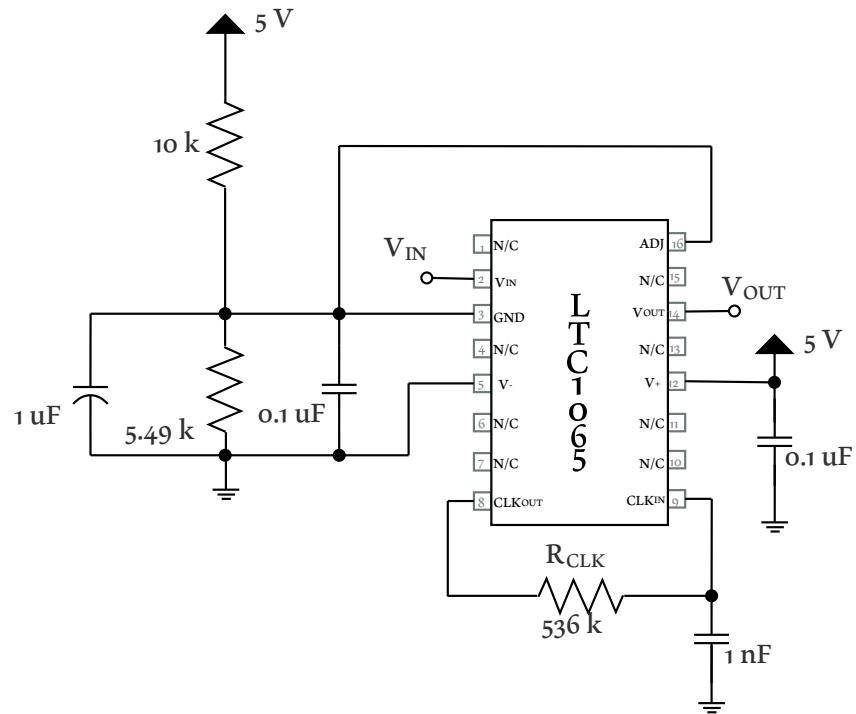
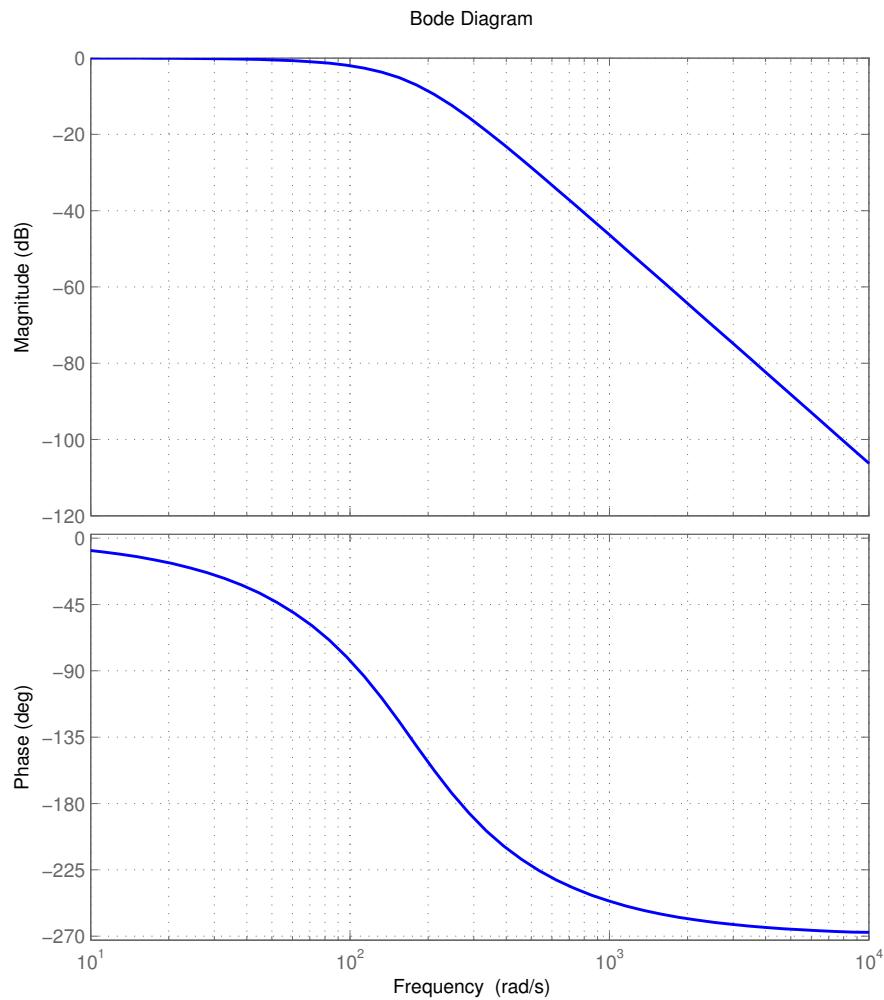
Since the environment in which the circuit is going to be used is a laboratory, so a really noisy place, the conditioning circuit for the sensor has to involve the design of a **low-pass filter** in order to reject the noise.

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 all 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.

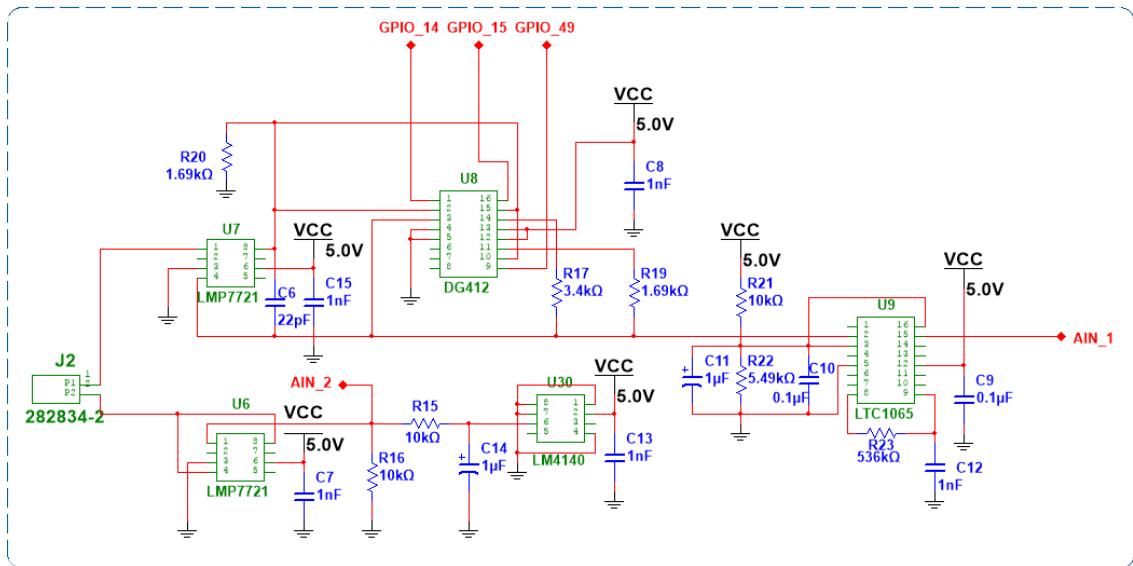


Figure 19: Acquisition Path for pH Sensor

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:

$$V_T = 680\mu \cdot R_T \quad (5)$$

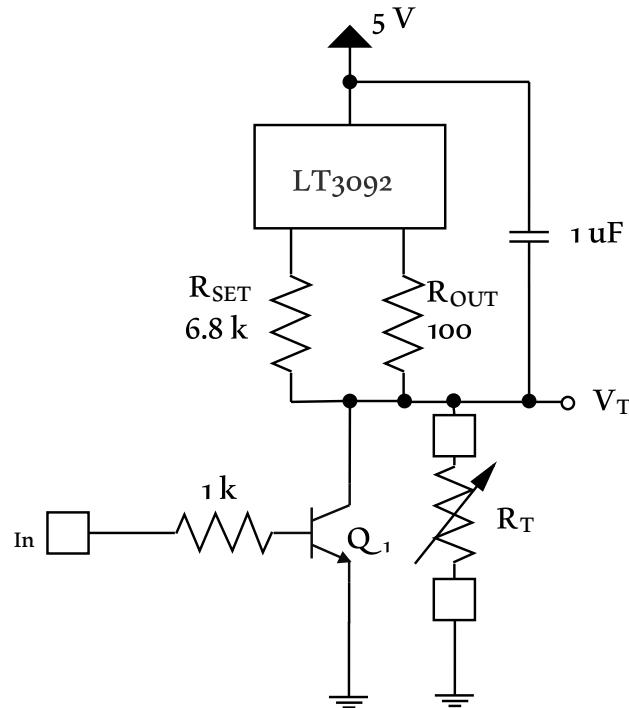


Figure 20: Conditioning Circuit for Temperature Sensor

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

$$I_{OUT} = 10\mu \cdot \frac{R_{SET}}{R_{OUT}} \quad (6)$$

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 one.

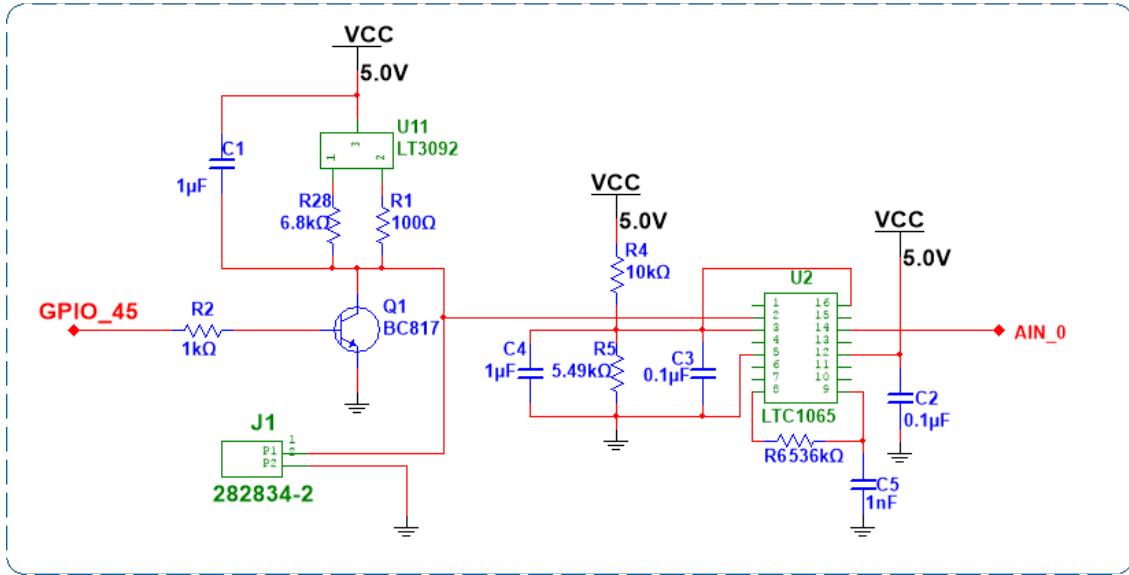


Figure 21: Acquisition Path for pH Sensor

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-Channel 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 IN4148 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

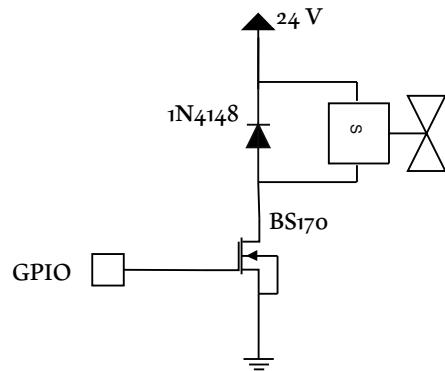


Figure 23: Driver for electrovalves

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.

Through 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.

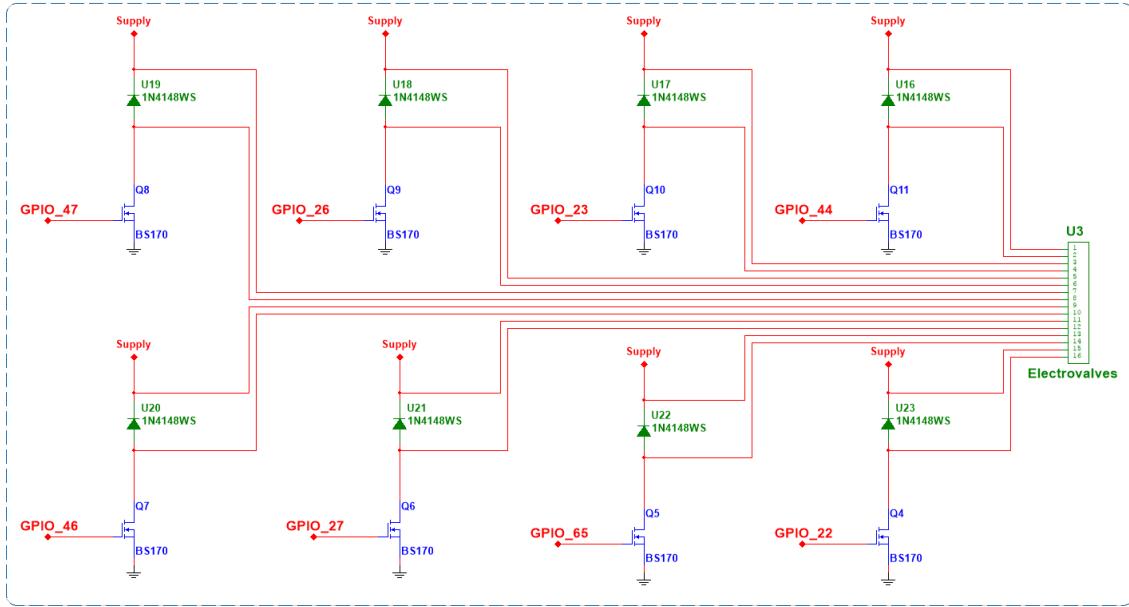


Figure 24: Electrovalves Driver Circuit

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:

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

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

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.

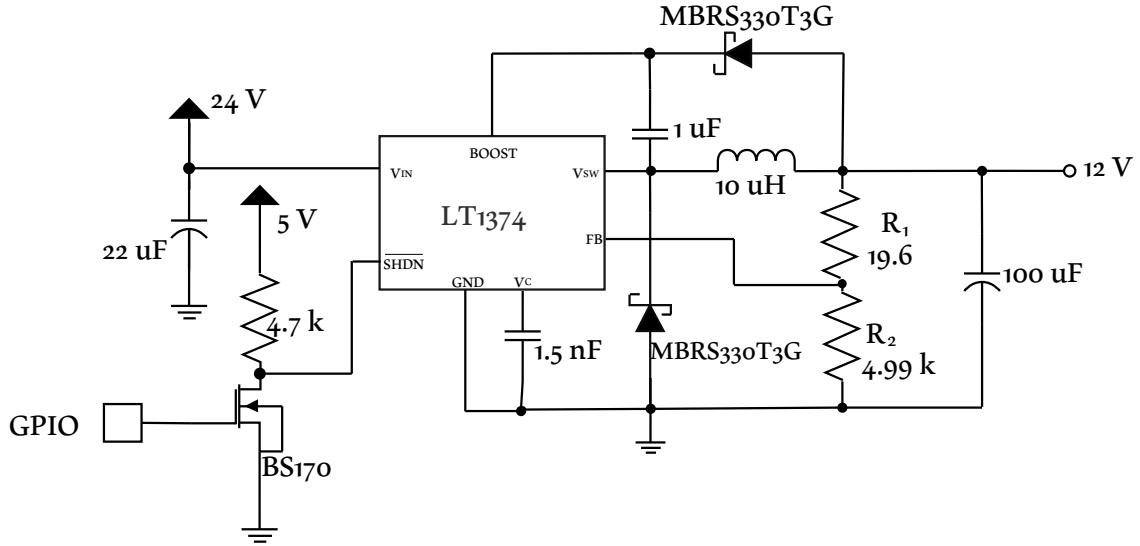


Figure 25: DC-DC circuit

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

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

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 0.05Ω to 0.2Ω , 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).

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

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 .

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

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 Beaglebone 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.

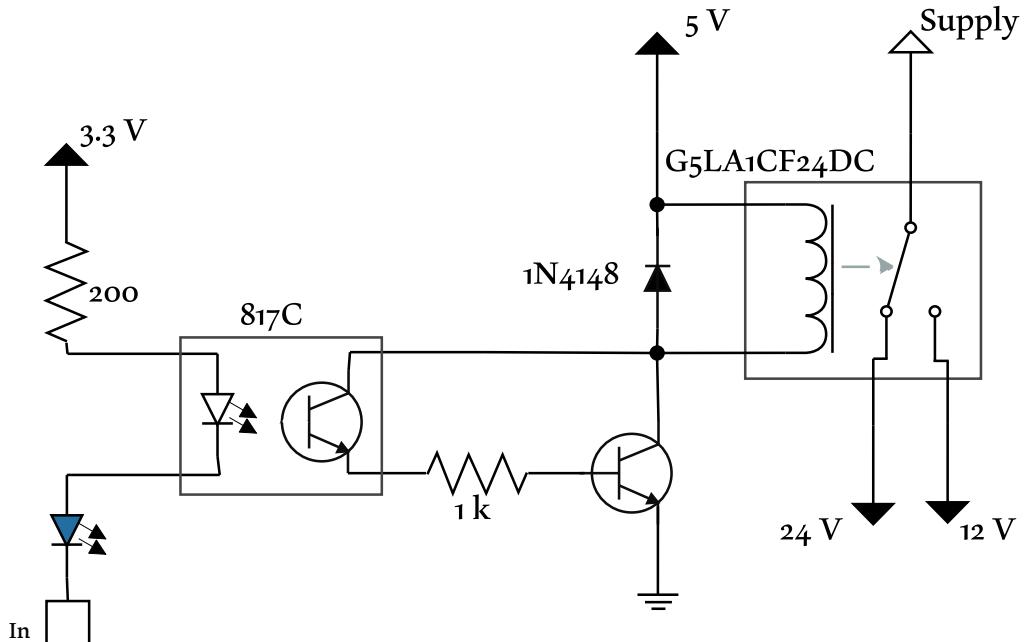


Figure 26: Relay circuit

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 .

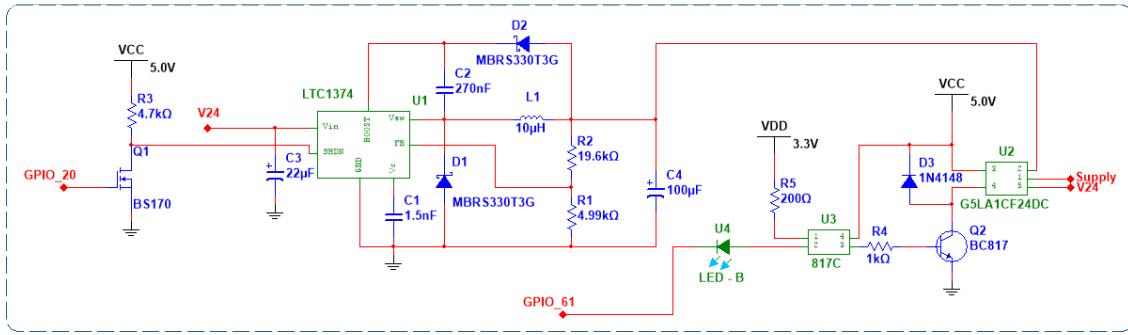


Figure 27: Electrovalves Supply Circuit

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

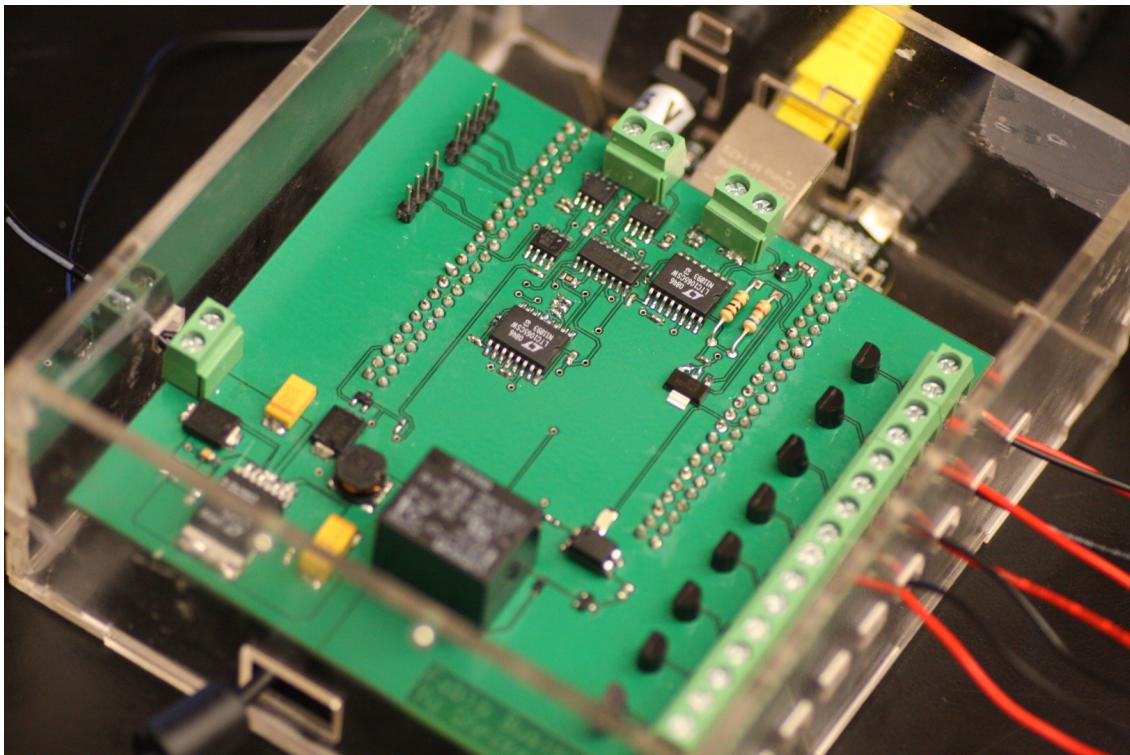


Figure 28: Final system mounted on a PCB

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 ($100 \times 100 \text{ mm}$) 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.

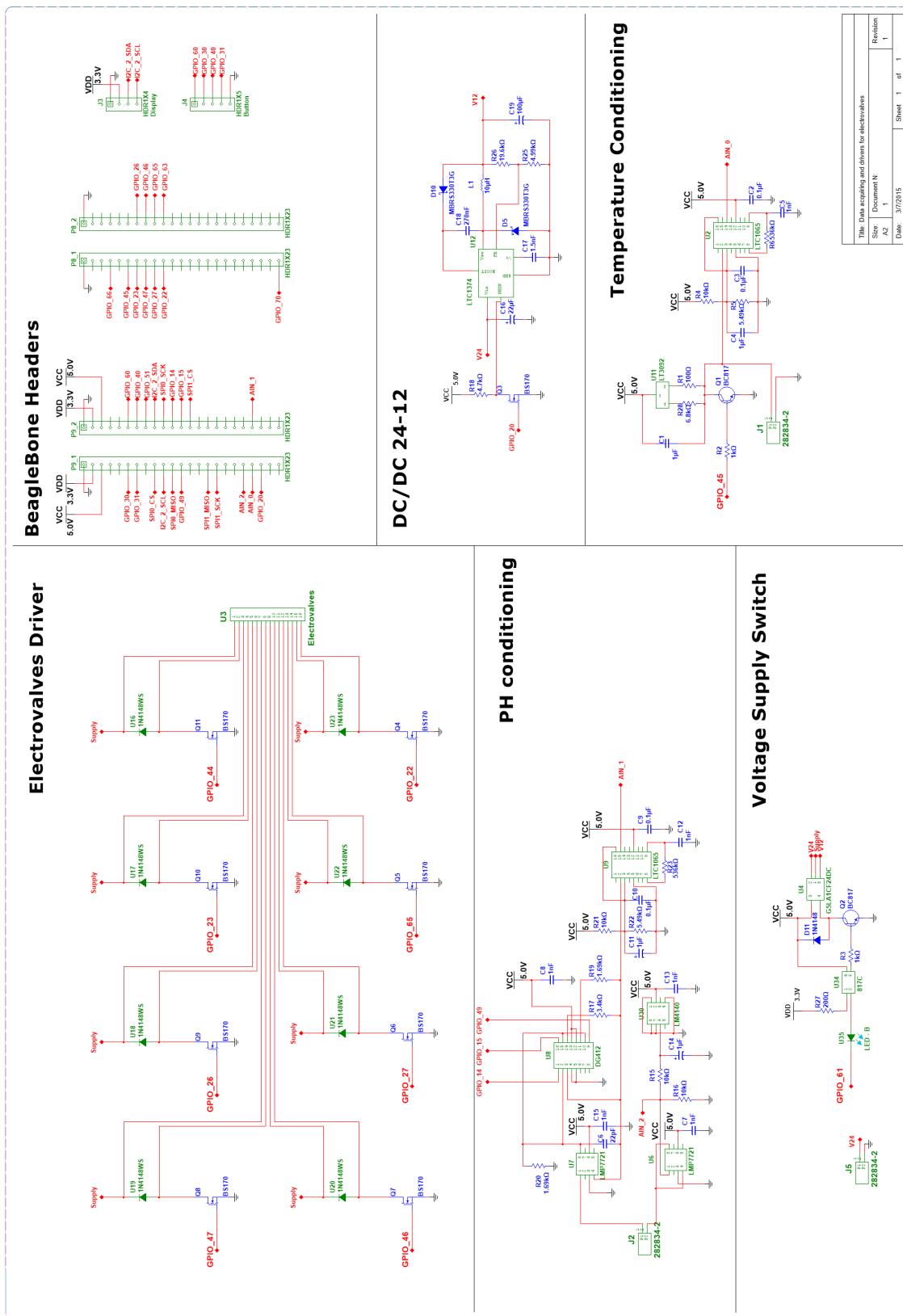


Figure 29: Schematic of Complete Circuit

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**.

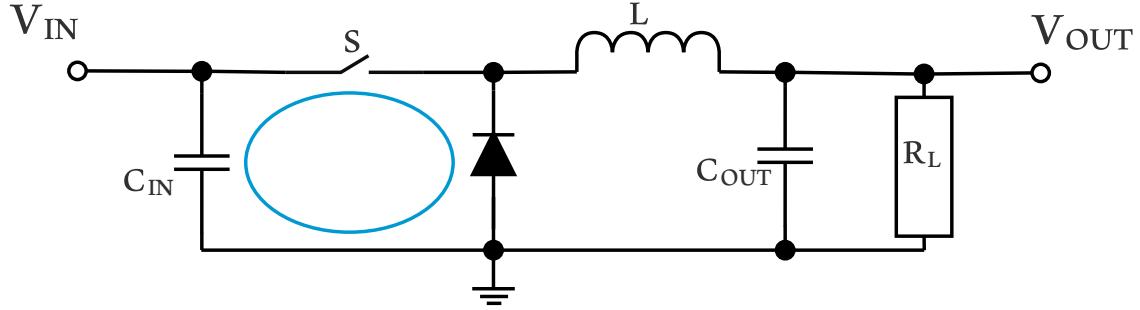


Figure 30: DC-DC High Speed Switching Path

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).

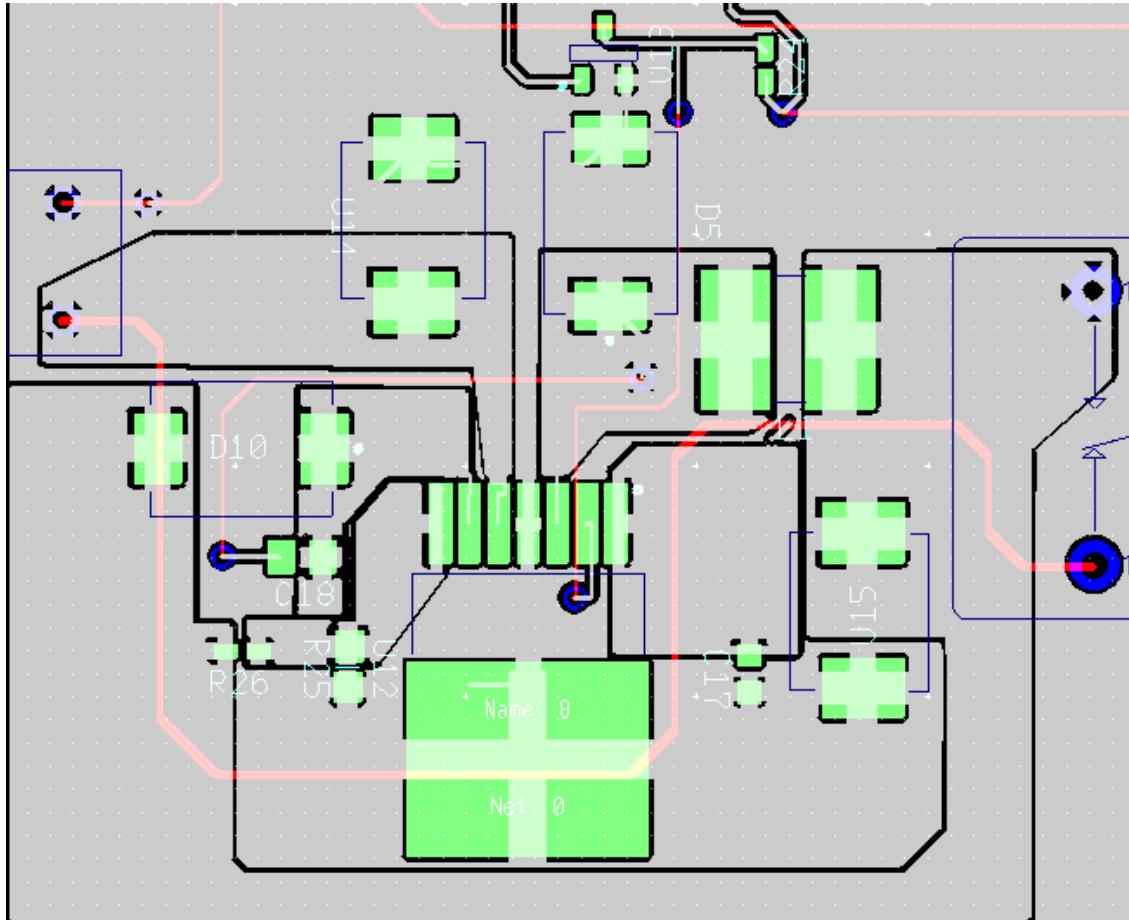


Figure 31: PCB Layout of DC-DC

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.

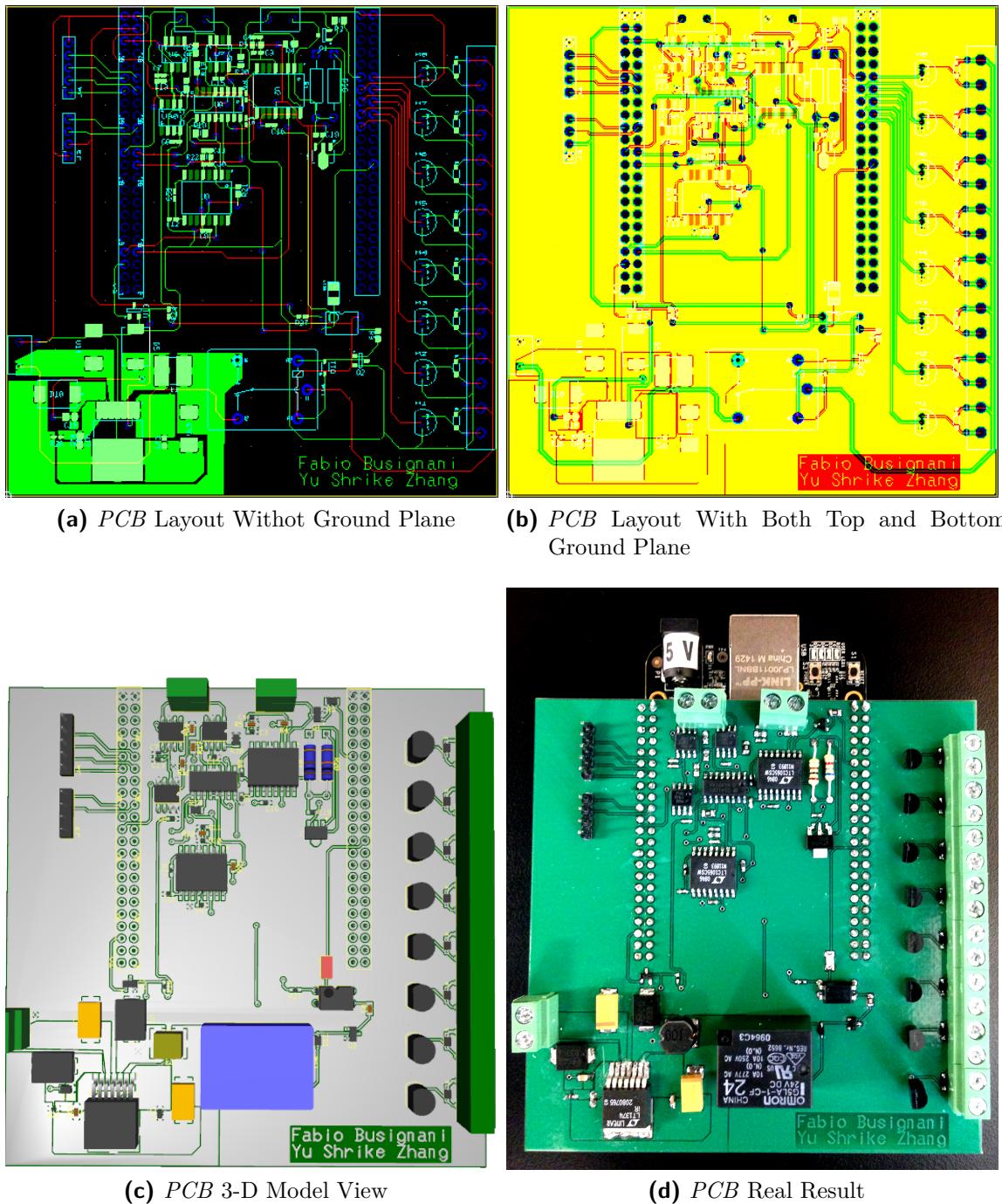


Figure 32: PCB of the System

Part II

FIRMWARE

3

INTRODUCTION

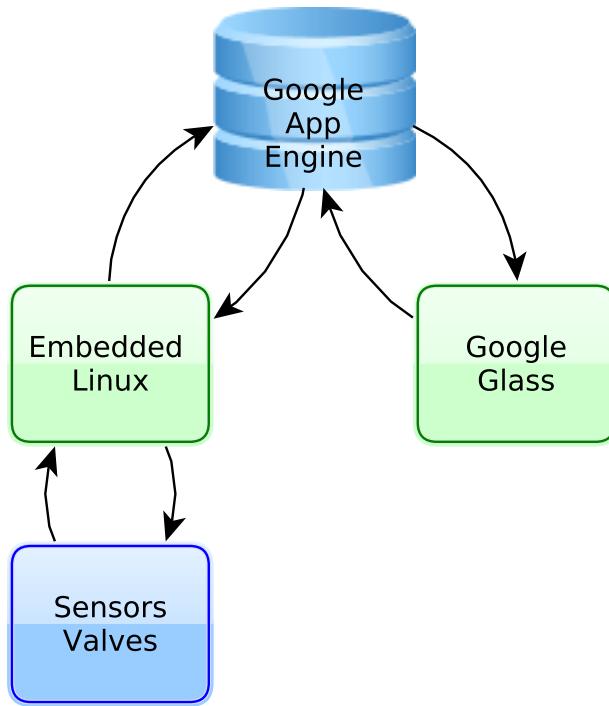


Figure 33: High Level System's Block Diagram

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 been 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..

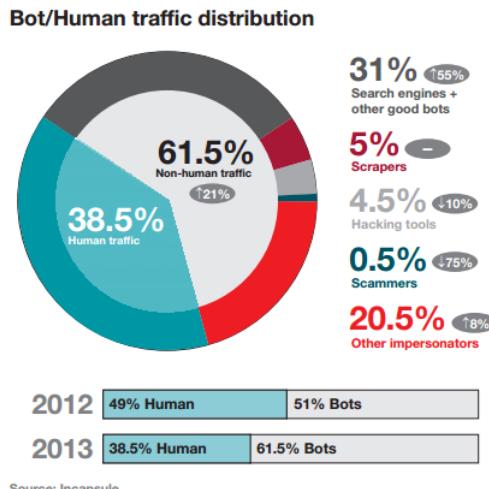


Figure 34: Distribution of Internet of Things

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

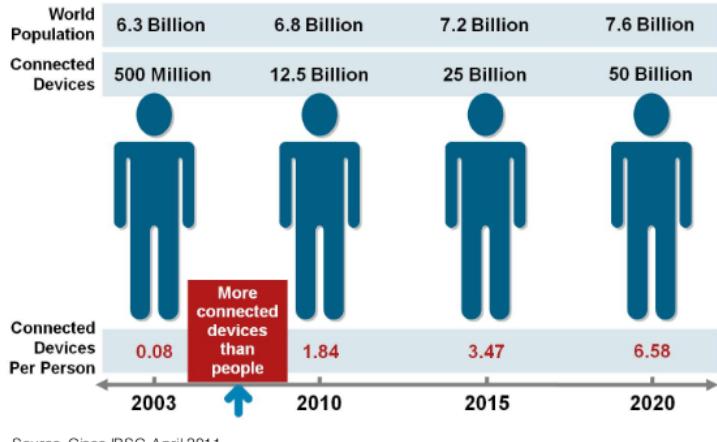


Figure 35: The *IoT* Evolution Prediction

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 download and use 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;
- 4 GB 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;
- 512 MB DDR3 of system memory;
- Ethernet processor, which supports *DHCP*, so it can be directly connected 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 to 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.

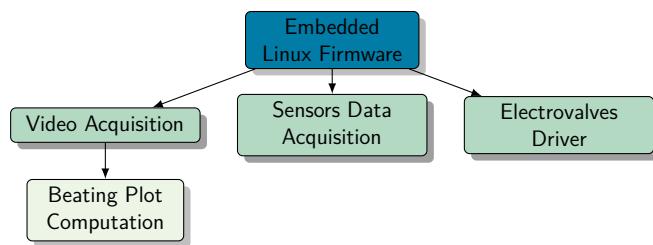


Figure 37: Embedded Linux Firmware - Blocks Subdivision

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™* micro-processor 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 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.

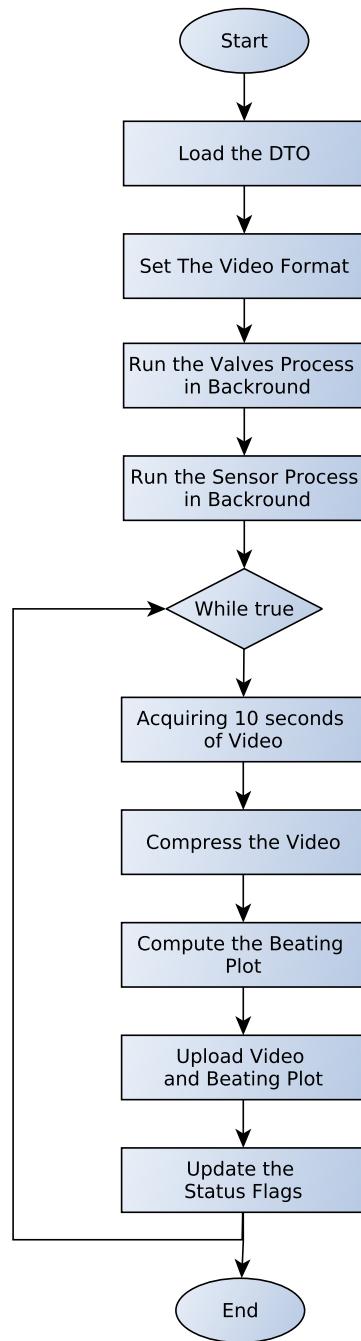


Figure 38: Bash Script Flow Chart

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 stores 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 have 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

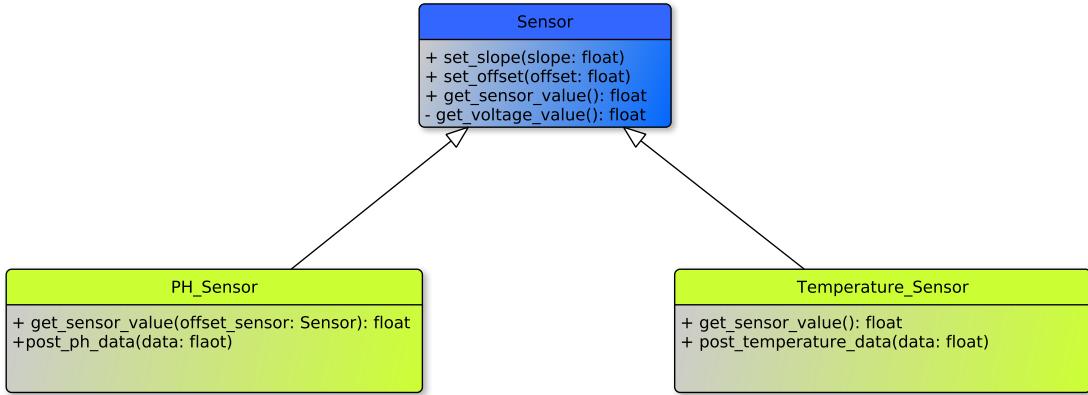
From the main *bash* script another one is launched for acquiring the sensors data. This script is shown in (list.A.2) and as can be seen it is an infinite loop where two operations are performed:

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 values on the Datastore of Google App Engine then it waits for 100 ms. Two important features have 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 cloud [6]. This solution is very useful because it allows everyone to write their 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 websites easily to manage. It has no database abstraction layer, indeed for this purpose, in this application, it supports the Google App Engine.

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

Using the Google App Engine means you do not have access to traditional databases 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 been passed as *POST* data, and named as "*Image.jpg*". While the 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.

Upload microscope file

No file chosen

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.

Upload microscope video

No file chosen

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" followed 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 needs 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 researchers, and it is very easy to encounter different operating systems.

In (App.A.2) the listings of this application are shown. As can be seen, the source codes are divided in such a way to ensure high hierarchical efficiency.

Indeed, the main (List.A.10) of this application just instantiates a *MyTimer* object. This *MyTimer* object (List.A.12) is in charge to generate an interrupt every 10 sec, and it starts from its creation. When this interrupt comes, the timer uses the *Downloader* object to check if a new microscope video has been uploaded, and if so, download it and reset the flag that points up the new video status. The code of *Downloader* is shown in (List.A.14). The videos are stored inside the directory *C:/Video* and their names correspond to the date and hour of download.

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).

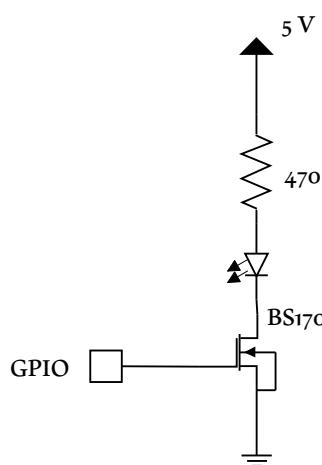


Figure 44: Driver for LED

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.

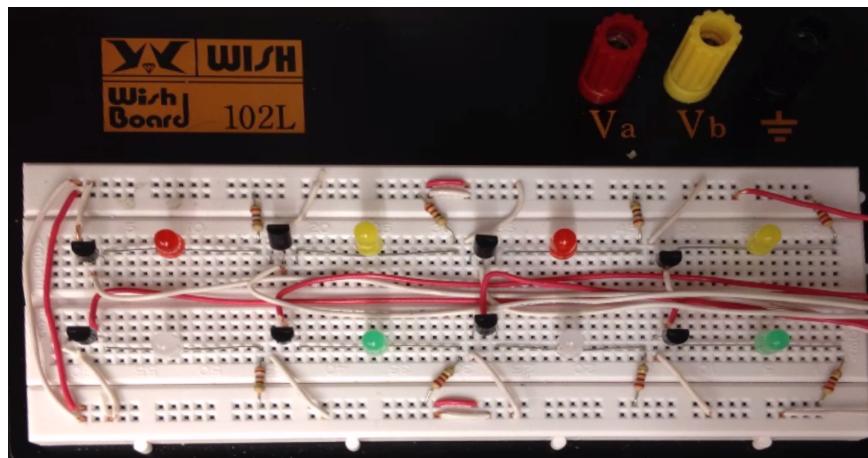


Figure 45: LEDs experiments board

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.

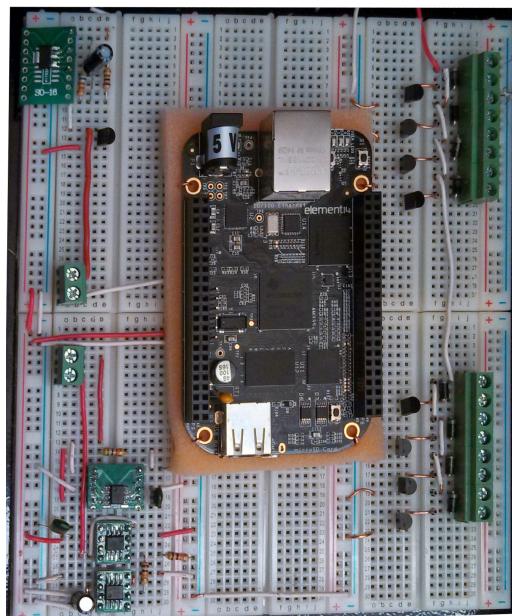


Figure 46: Circuit mounted on breadboard top view

The result of the experiments with electrovalves in a real microfluidic case can bee watched in [this](#) video.

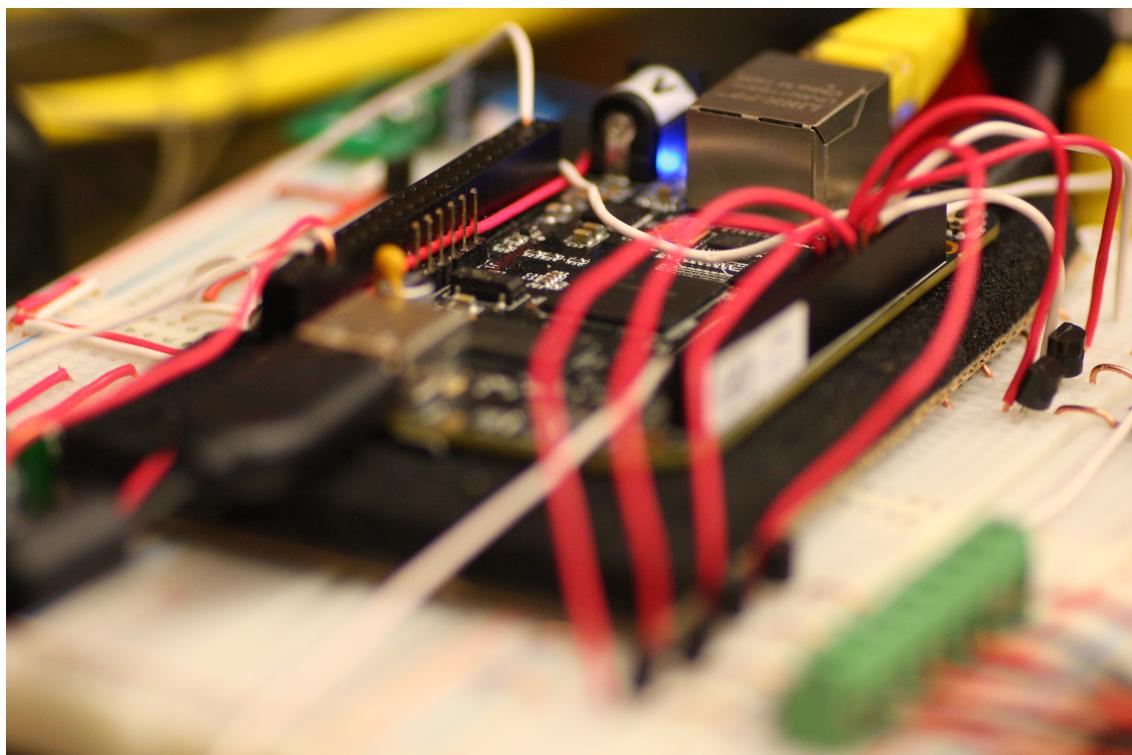


Figure 47: Circuit mounted on breadboard side view

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 previous step.

A | CODE

A.1 FIRMWARE

Listing A.1: KlabFirware (bash)

```
1 #!/bin/bash
2 echo "Load the DTO"
3 echo GlassProject-GPIO > $SLOTS
4
5 echo "Setting the video format as MJPG"
6 v4l2-ctl --set-fmt-video=width=320,height=240,pixelformat=1 -d 0
7 v4l2-ctl --set-parm=30
8
9 echo "Running the Electrovalve status updating task"
10 ./Electrovalves/ValvesUpdating </dev/null &>/dev/null &
11 ./Sensors/sensorAcquiring </dev/null &>/dev/null &
12
13 while true; do
14     echo "Recording ten second of Video"
15     ./Video/capture -d /dev/video0 -F -o -c 300 > output.raw -y
16     avconv -f mjpeg -i output.raw -vcodec copy output.mp4 -y
17     echo "Compressing Video"
18     ffmpeg -i output.mp4 -acodec mp2 Video.mp4
19     echo "Uploading the Video to the server"
20     curl -include --form Video.mp4=@Video.mp4 -A "National Instruments LabVIEW"
21         http://planar-contact-601.appspot.com/video/submit -0
22     echo "Updating the flag for video"
23     curl --data "name=video&status=on"
24         http://planar-contact-601.appspot.com/add/electrovalve
25     curl --data "name=glass&status=on"
26         http://planar-contact-601.appspot.com/add/electrovalve
27     echo "Ploting the beating rate"
28     ./Video/compute_beating
29     echo "Uploading the beating plot"
30     curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW"
31         http://planar-contact-601.appspot.com/picture/submit -0
32     echo "finish"
33 done
```

Listing A.2: SensorAcquiring (bash)

```

1#!/bin/bash

3while true; do
4    echo "Clearing previous data on server"
5    curl http://planar-contact-601.appspot.com/clear
6    echo "Acquiring and uploading new sensors data"
7    ./sensor_acquiring

9done

```

Listing A.3: Pins Setting

```

1/*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
3 *
4 * This program is free software; you can redistribute it and/or modify
5 * it under the terms of the GNU General Public License Version 2 as
6 * published by the Free Software Foundation
7 *
8 * Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
9 *
10 * Modified by Fabio Busignani fbusigna@mit.edu
11 *
12 */
13
14/dts-v1/;
15/plugin/;

17/{
18    compatible = "ti,beaglebone", "ti,beaglebone-black";
19    part-number = "KLab";
20    version = "00A0";
21
22    fragment@0 {
23        target = <&am33xx_pinmux>;
24
25        __overlay__ {
26            ebb_example: KLab {
27                pinctrl-single,pins = <
28                    0x04c 0x07 // P9_16 - T - Output Mode7 pulldown
29                    0x024 0x07 // P8_13 - EV1 - Output Mode7 pulldown
30                    0x028 0x07 // P8_14 - EV4 - Output Mode7 pulldown
31                    0x03c 0x07 // P8_15 - EV3 - Output Mode7 pulldown
32                    0x038 0x07 // P8_16 - EV6 - Output Mode7 pulldown
33                    0x02c 0x07 // P8_17 - EV5 - Output Mode7 pulldown
34                    0x08c 0x07 // P8_18 - EV8 - Output Mode7 pulldown
35                    0x020 0x07 // P8_19 - EV7 - Output Mode7 pulldown
36                    0x030 0x07 // P8_44 - EV2 -Output Mode7 pulldown
37                >;
38            };
39        };
40    };
41}

```

```

37                         0x0a0 0x07 // P8_46 - PowerSupply

39
40         >;
41     };
42   };
43
44   fragment@1 {
45     target = <&ocp>;
46     __overlay__ {
47       gpio_helper {
48         compatible = "gpio-of-helper";
49         status = "okay";
50         pinctrl-names = "default";
51         pinctrl-0 = <&eklab>;
52       };
53     };
54   };
55 }
```

Listing A.4: ElectrovalvesDriver.cpp

```

1 #include <stdio.h>
2 #include <string.h>
3 #include <iostream>
4 #include <sstream>
5 #include <curl/curl.h>
6 #include <GPIO/GPIO.h>

8
9 #define NUMBER_ELECTROVALVES 8

10
11 using namespace std;
12
13 std::string const ELECTROVALVES_LINK =
14   "http://planar-contact-601.appspot.com/show/";
15 std::string const TRUE_VALUE = " True ";
16 std::string const FALSE_VALUE = " False ";

18 static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
19 {
20   ((std::string*)userp)->append((char*)contents, size * nmemb);
21   return size * nmemb;
22 }

24 int main(void)
25 {
26   // constructor of pins
27   GPIO EV1(44), EV2(23), EV3(26), EV4(47), EV5(46), EV6(27), EV7(65), EV8(22);
28   GPIO RELAY(61);
```

```

29     CURL *curl;
30     CURLcode res;
31
32     GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];
33
34     //Set output pins
35     EV1.setDirection(OUTPUT);
36     EV2.setDirection(OUTPUT);
37     EV3.setDirection(OUTPUT);
38     EV4.setDirection(OUTPUT);
39     EV5.setDirection(OUTPUT);
40     EV6.setDirection(OUTPUT);
41     EV7.setDirection(OUTPUT);
42     EV8.setDirection(OUTPUT);
43
44     RELAY.setDirection(OUTPUT);
45
46     RELAY.setValue(HIGH);
47
48     while(1)
49     {
50         //acquire the values of electrovalves status and store them inside
51         //output_vector
52         int i;
53         for(i=0;i<NUMBER_ELECTROVALVES; i++)
54         {
55             curl = curl_easy_init();
56             if(curl)
57             {
58                 std::string readBuffer;
59                 std::ostringstream ss;
60                 ss << i+1;
61                 std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
62                 curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
63                 /* example.com is redirected, so we tell libcurl to follow redirection */
64                 curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
65                 curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
66                 /* Perform the request, res will get the return code */
67                 res = curl_easy_perform(curl);
68                 /* always cleanup */
69                 curl_easy_cleanup(curl);
70
71                 if(readBuffer.compare(TRUE_VALUE) == 0)
72                 {
73                     std::cout << "EV" + ss.str() + ": ON" << std::endl;
74                     output_vector[i] = HIGH;
75                 }
76                 else if(readBuffer.compare(FALSE_VALUE) == 0)
77                 {
78                     std::cout << "EV" + ss.str() + ": OFF" << std::endl;
79                     output_vector[i] = LOW;
80                 }
81             }
82         }
83     }
84 }
```

```

77     {
78         std::cout << "EV" + ss.str() + ": OFF" << std::endl;
79         output_vector[i] = LOW;
80     }
81 }
82
83 //now I'm ready to write the output pins:
84 EV1.setValue(output_vector[0]);
85 EV2.setValue(output_vector[1]);
86 EV3.setValue(output_vector[2]);
87 EV4.setValue(output_vector[3]);
88 EV5.setValue(output_vector[4]);
89 EV6.setValue(output_vector[5]);
90 EV7.setValue(output_vector[6]);
91 EV8.setValue(output_vector[7]);
92 }
93 return 0;
94 }
```

Listing A.5: sensor_aquiring.cpp

```

1 #include<iostream>
2 #include<fstream>
3 #include<unistd.h>
4 #include<string>
5 #include<sstream>
6 #include<stdlib.h>
7 #include<curl/curl.h>
8 #include"../GPIO.h"
9 #include"../http.h"
10 using namespace std;
11
12 #define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
13 #define GPIO_PATH "/sys/class/gpio/"
14 #define OUTPUT false
15 #define INPUT true
16 #define HIGH true
17 #define LOW false
18
19 float const CURRENT = 0.00068;
20 float const SLOPE_TEMPERATURE = 2;
21 float const OFFSET_TEMPERATURE = 1870;
22 float const SLOPE_PH = -0.070;
23 float const OFFSET_PH = 0.6;
24
25 class Sensor
26 {
27     int number;
28 }
```

```

public:
30   float m;
31   float q;
32 Sensor(int x): number(x){}
33   void set_slope(float slope){
34     m = slope;
35   }
36   void set_offset(float offset){
37     q = offset;
38   }
39   float get_sensor_value(){
40     float voltage_value = this->get_voltage_value();
41     float sensor_value = (voltage_value - q)/m;
42     return sensor_value;
43   }
44 private:
45   float get_voltage_value(){
46     stringstream ss;
47     ss << LDR_PATH << number << "_raw";
48     fstream fs;
49     int adc_value;
50     fs.open(ss.str().c_str(), fstream::in);
51     fs >> adc_value;
52     fs.close();
53     float cur_voltage = adc_value * (1.80f/4096.0f);
54     float actual_value;
55     return cur_voltage;
56   }
57 };
58

60 class Temperature_Sensor : public Sensor{
61 private:
62
63 public:
64   Temperature_Sensor(int x) : Sensor(x) {}
65   float get_sensor_value(){
66     float voltage_value = this->get_voltage_value();
67     float resistor_value = voltage_value / CURRENT;
68     float sensor_value = (resistor_value - q)/m;
69     return sensor_value;
70   }
71   void post_temperature_data(float data){
72     post_data_sensor(0, data);
73   }
74 };
75

76 class PH_Sensor : public Sensor{

```

```
78 public:
79     PH_Sensor(int x) : Sensor(x) {}
80     float get_sensor_value(Sensor offset_sensor){
81         //Sensor_offset_sensor(2);
82         float voltage_value = this->get_voltage_value();
83         q += offset_sensor.get_voltage_value();
84         float sensor_value = (voltage_value - q)/m;
85         return sensor_value;
86     }
87
88     void post_ph_data(float data){
89         post_data_sensor(1, data);
90     }
91 };
92
93 int main(int argc, char* argv[])
94 {
95     GPIO temperature_disable(45);
96     float slope_temperature;
97     float slope_ph;
98     float offset_temperature;
99     float offset_ph;
100
101     if (argc == 5)
102     {
103         slope_temperature = atoi(argv[1]);
104         offset_temperature = atoi(argv[2]);
105         slope_ph = atoi(argv[3]);
106         offset_ph = atoi(argv[4]);
107     }
108     else
109     {
110         slope_temperature = SLOPE_TEMPERATURE;
111         offset_temperature = OFFSET_TEMPERATURE;
112         slope_ph = SLOPE_PH;
113         offset_ph = OFFSET_PH;
114     }
115
116     //set the pin GPIO_45 as output
117     temperature_disable.setDirection(OUTPUT);
118     //Temporary disable the temperature
119     temperature_disable.setValue(HIGH);
120
121     Temperature_Sensor temperature_sensor(2);
122     PH_Sensor ph_sensor(1);
123     Sensor offset_sensor(2);
124
125     temperature_sensor.set_slope(slope_temperature);
126     temperature_sensor.set_offset(offset_temperature);
```

```

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

130 int i= 0;
131 for(i=0; i<100; i++)
132 {
133     temperature_disable.setValue(LOW);
134     float ph = ph_sensor.get_sensor_value(offset_sensor);
135     float temperature = temperature_sensor.get_sensor_value();
136     temperature_disable.setValue(HIGH);
137     cout << "The voltage value is: " << temperature << " V." << endl;
138     temperature_sensor.post_temperature_data(temperature);
139     ph_sensor.post_ph_data(ph);
140     usleep(100000);
141 }
142 float temperature = temperature_sensor.get_voltage_value();
143 cout << "The voltage value is: " << temperature << " V." << endl;
144 return 0;
145 }
```

Listing A.6: compute_beating.cpp

```

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

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

19     // set any properties in the VideoCapture object
20     capture.set(CV_CAP_PROP_FRAME_WIDTH,320); // width pixels
21     capture.set(CV_CAP_PROP_FRAME_HEIGHT,240); // height pixels
22

23     if(!capture.isOpened())
24     {
25         cout << "Capture not open." << endl;
26     }
27 }
```

```

29     Mat frame, firstFrame, diffFrame;

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

37     int i;
38     for(i=1; i<capture.get(CV_CAP_PROP_FRAME_COUNT);i++)
39     {
40         capture >> frame;           // capture the image to the frame
41         if(frame.empty())
42         {
43             cout << "Failed to capture an image" << endl;
44             return -1;
45         }
46         absdiff(frame, firstFrame, diffFrame);
47         color_information = mean(frame);
48         float red = color_information[0];
49         float green = color_information[1];
50         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
54         float temp = (i-1);
55         temp *= 0.033;

57         point_file << temp << '\t';
58         point_file << mean_color << '\n';

59     }

61 }

63
64     point_file.close();
65     return 0;
66 }
```

Listing A.7: Pins Setting

```

/*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
3 *
4 * This program is free software; you can redistribute it and/or modify
5 * it under the terms of the GNU General Public License Version 2 as
6 * published by the Free Software Foundation
7 *
```

```

8 * Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
9 *
10 * Modified by Fabio Busignani fbusigna@mit.edu
11 *
12 */
13
14 /dts-v1/;
15 /plugin/;
16
17 /{
18     compatible = "ti,beaglebone", "ti,beaglebone-black";
19     part-number = "KLab";
20     version = "00AO";
21
22     fragment@0 {
23         target = <&am33xx_pinmux>;
24
25         __overlay__ {
26             ebb_example: KLab {
27                 pinctrl-single,pins = <
28                     0x04c 0x07 // P9_16 - T - Output Mode7 pulldown
29                     0x024 0x07 // P8_13 - EV1 - Output Mode7 pulldown
30                     0x028 0x07 // P8_14 - EV4 - Output Mode7 pulldown
31                     0x03c 0x07 // P8_15 - EV3 - Output Mode7 pulldown
32                     0x038 0x07 // P8_16 - EV6 - Output Mode7 pulldown
33                     0x02c 0x07 // P8_17 - EV5 - Output Mode7 pulldown
34                     0x08c 0x07 // P8_18 - EV8 - Output Mode7 pulldown
35                     0x020 0x07 // P8_19 - EV7 - Output Mode7 pulldown
36                     0x030 0x07 // P8_44 - EV2 -Output Mode7 pulldown
37                     0x0a0 0x07 // P8_46 - PowerSupply
38
39                 >;
40             };
41         };
42     };
43
44     fragment@1 {
45         target = <&ocp>;
46         __overlay__ {
47             gpio_helper {
48                 compatible = "gpio-of-helper";
49                 status = "okay";
50                 pinctrl-names = "default";
51                 pinctrl-0 = <&eklab>;
52             };
53         };
54     };
55 };

```

Listing A.8: HTTP.h

```
1 #ifndef HTTP
2 #define HTTP
3
4 #include<iostream>
5 #include<fstream>
6 #include<unistd.h>
7 #include<string>
8 #include<sstream>
9 #include<stdlib.h>
10
11 #include<curl/curl.h>
12
13 std::string const URL_POST = "http://planar-contact-601.appspot.com/sensor_values";
14 std::string const POST_DATA_TEMPERATURE = "sensor=temperature&value=";
15 std::string const POST_DATA_PH = "sensor=pH&value=";
16 std::string const POST_DATA_BEATING = "sensor=beating&value=";
17
18 void post_data_sensor(int type, float value);
19
20 #endif // HTTP
```

Listing A.9: HTTP.cpp

```
#include "http.h"

2

void post_data_sensor(int type, float value){
4
    CURL *curl;
    CURLcode res;
6
    /* In windows, this will init the winsock stuff */
    curl_global_init(CURL_GLOBAL_ALL);
8
10
    /* get a curl handle */
    curl = curl_easy_init();
12
    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. */
14
        curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
16
        /* Now specify the POST data */
18
        std::ostringstream ss;
19
        ss << value;
20
        std::string s;
21
        if (type == 0)
22
            s = POST_DATA_TEMPERATURE +(ss.str());
23
        else if (type == 1)
24
            s = POST_DATA_PH +(ss.str());
25
        else
26
            s = POST_DATA_BEATING + (ss.str());
27
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
28
    }
29
}
```

```

28     /* Perform the request, res will get the return code */
29     res = curl_easy_perform(curl);
30     /* Check for errors */
31     if(res != CURLE_OK)
32         fprintf(stderr, "curl_easy_perform() failed: %s\n",
33                 curl_easy_strerror(res));
34     /* always cleanup */
35     curl_easy_cleanup(curl);
36 }
37 curl_global_cleanup();
38 }
```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```

1 #include <QCoreApplication>
2 #include "downloader.h"
3 #include "mytimer.h"
4 #include <QDebug>
5
6 int main(int argc, char *argv[])
7 {
8     QCoreApplication a(argc, argv);
9     MyTimer t;
10    return a.exec();
11 }
```

Listing A.11: VideoStoring.pro

```

1 -----
2 #
3 # Project created by QtCreator 2015-02-17T21:56:37
4 #
5 -----
6
7 QT      += core
8 QT      += network
9
10 QT      -= gui
11
12 #RC_FILE = myapp.rc
13
14 TARGET = VideoStoring
15 CONFIG  += console
16 CONFIG  -= app_bundle
```

```

17 TEMPLATE = app
19
21 SOURCES += main.cpp \
22     downloader.cpp \
23     mytimer.cpp
25 HEADERS += \
26     downloader.h \
27     mytimer.h

```

Listing A.12: mytimer.h

```

1 #ifndef MYTIMER_H
2 #define MYTIMER_H
3
4 #include <QObject>
5 #include <QtCore>
6 #include <downloader.h>
7
8 class MyTimer : public QObject
9 {
10     Q_OBJECT
11 public:
12     MyTimer();
13     ~MyTimer();
14     QTimer *timer;
15
16 public slots:
17     void timerSlot();
18 private:
19     Downloader d;
20 };
21
22#endif // MYTIMER_H

```

Listing A.13: mytimer.cpp

```

1 #include "mytimer.h"
2 #include <QtCore>
3 #include <QDebug>
4 MyTimer::MyTimer()
5 {
6     timer = new QTimer(this);
7     connect(timer, SIGNAL(timeout()), this, SLOT(timerSlot()));
8
9     timer->start(10000);

```

```

11 }

13 MyTimer::~MyTimer()
{
15 }

17 void MyTimer::timerSlot(){
18     d.checkFlag();
19     if( d.getStatus() )
20     {
21         d.resetFlag();
22         d.doDownload();
23     }
24 }
25 }
```

Listing A.14: downloader.h

```

1 #ifndef DOWNLOADER_H
2 #define DOWNLOADER_H

4 #include <QObject>
5 #include <QNetworkAccessManager>
6 #include <QNetworkRequest>
7 #include <QNetworkReply>
8 #include <QUrl>
9 #include <QDateTime>
10 #include <QFile>
11 #include <QDebug>
12 #include <QEventLoop>

14 class Downloader : public QObject
15 {
16     Q_OBJECT
17
18     public:
19         explicit Downloader(QObject *parent = 0);
20
21         void doDownload();
22         void checkFlag();
23         bool getStatus();
24         void resetFlag();
25
26     signals:
27
28     public slots:
29         void replyFinished (QNetworkReply *reply);
30         void checkStatus(QNetworkReply *replay);
```

```

private:
32   QNetworkAccessManager *manager;
33   QNetworkAccessManager *manager2;
34   QDateTime data;
35   QString name;
36   bool flag_ready;
37   bool status;
38
39 };
40
41 #endif // DOWNLOADER_H

```

Listing A.15: downloader.cpp

```

#include "downloader.h"
2 const QString FLAG_ON = " True ";
3
4 Downloader::Downloader(QObject *parent) :
5     QObject(parent)
6 {
7 }
8
9 void Downloader::doDownload()
10 {
11     manager = new QNetworkAccessManager(this);
12
13     connect(manager, SIGNAL(finished(QNetworkReply*)),
14             this, SLOT(replyFinished(QNetworkReply*)));
15
16     manager->get(QNetworkRequest(
17         QUrl("http://planar-contact-601.appspot.com/video/view")));
18 }
19
20 void Downloader::replyFinished (QNetworkReply *reply)
21 {
22     if(reply->error())
23     {
24         qDebug() << "ERROR!";
25         qDebug() << reply->errorString();
26     }
27     else
28     {
29         qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
30         qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).
31             toDateTime().toString();
32         qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).
33             toULongLong();
34         qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).

```

```

        toInt();

36    qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).
        toString();

38
39    data = QDateTime::currentDateTime();
40    name = "C:/Video/" + data.toString("yyyy-MM-dd-HH-mmss") + ".mpeg";

42    QFile *file = new QFile(name);
43    if(file->open(QFile::Append))
44    {
45        file->write(reply->readAll());
46        file->flush();
47        file->close();
48    }
49    delete file;
50 }

52    reply->deleteLater();
53}

54

56 void Downloader::checkStatus(QNetworkReply *replay)
{
57
58    QString result(replay->readAll());
59    int compare = QString::compare(result, FLAG_ON);
60    if(compare == 0)
61    {
62        status = true;
63    }
64    else
65    {
66        status = false;
67    }
68    qDebug() << status;
69    flag_ready = true;
70}
71

72 void Downloader::checkFlag()
{
73
74    manager = new QNetworkAccessManager(this);
75    QNetworkReply *reply = manager->get(QNetworkRequest
76
77        (QUrl("http://planar-contact-601.appspot.com/electrovalves/video")));
78
79    QEventLoop loop;
80    connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));

81
82    loop.exec();
83    qDebug() << "I'm looking for a new video";
84    QString result(reply->readAll());
85    int compare = QString::compare(result, FLAG_ON);

```

```

84     if(compare == 0)
85     {
86         status = true;
87         qDebug() << "Found it";
88     }
89     else
90     {
91         status = false;
92         qDebug() << "No new video available";
93     }
94 }
95
96     bool Downloader::getStatus()
97 {
98     return status;
99 }
100
101 void Downloader::resetFlag()
102 {
103     status = false;
104     manager2 = new QNetworkAccessManager(this);
105
106     QUrl flag_url = QUrl("http://planar-contact-601.appspot.com/add/electrovalve");
107     QByteArray postData;
108     postData.append("name=video&status=off");
109     manager2->post(QNetworkRequest(flag_url), postData);
110 }
```

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```

1 from flask import Flask
2
3 app = Flask(__name__)
4
5 from flask import request
6 from flask import make_response
7
8 from models import MESSAGES
9 from google.appengine.ext import ndb
10 from google.appengine.api import memcache
11 import logging
12 import datetime
13 import cloudstorage
```

```
import json

15
SECONDS_BETWEEN_UPDATES = 60
17 SENSOR_TIMEOUT_IN_SECONDS = 300
ELECTROVALVES_TIME_OUT = 1000
19 BUCKET_NAME = "/planar-contact-601.appspot.com/"
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
21 UPDATE_FLAG = 'database_updated_recently'

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

29
    @classmethod
31         # returns the list of DataPoint associated with sensor_key
    def points_for_sensor(cls, sensor_key):
        33             return cls.query(ancestor=sensor_key).order(-cls.date)

35
    @classmethod
37         #returns the last DataPoint inserted
    def oldest_points(cls):
        39             return cls.query().order(+cls.date)

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

47
def update():
    49         # Check to see if we have updated recently
    update_flag = memcache.get(UPDATE_FLAG)
    51         if update_flag is not None:
        53             return
    memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
    55         # Delete old data points
    now = datetime.datetime.now()
    count = 0
    57         for point in DataPoint.oldest_points().iter():
        59             delta = (now - point.date).total_seconds()
            if delta > 3600:
                point.key.delete()
                count += 1
            else:
```

```

63         break
64     logging.info("Deleted {} old data points".format(count))
65     # Loop through all of the sensors
66     sensor_names = []
67     for sensor in Sensor.sensor_list().iter():
68         # Store current values of each sensor
69         sensor_value = memcache.get(sensor.key.id() + "_value")
70         if sensor_value is not None:
71             point = DataPoint(parent=sensor.key, value=sensor_value)
72             point.put()
73         # Delete sensors that don't have any data
74         curr_points = DataPoint.points_for_sensor(sensor.key)
75         if curr_points.count() == 0 and sensor_value is None:
76             logging.info("Deleting sensor {}".format(sensor.key.id()))
77             sensor.key.delete()
78         else:
79             sensor_names.append(sensor.key.id())
80         # Store a list of sensor names in memcache
81         memcache.set("sensor_names", ",".join(sensor_names))
82
83     # this method is used to obtain the json format containg the list of the
84     # 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
86     # 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
88     # float number of the sensor's value.
89 @app.route('/', methods=['POST'])
90 @app.route('/sensor_values', methods=['GET', 'POST'])
91 def process_values():
92     if request.method == 'GET':
93         sensor_names = memcache.get('sensor_names')
94         if sensor_names is None:
95             sensor_names = ",".join([sensor.key.id() for sensor in
96                                     Sensor.sensor_list()])
97         memcache.set('sensor_names', sensor_names)
98         sensor_names = sensor_names.split(",")
99         sensor_values = {}
100        for sensor_name in sensor_names:
101            sensor_value = memcache.get(sensor_name + '_value')
102            if sensor_value is not None:
103                sensor_values[sensor_name] = sensor_value
104        return json.dumps(sensor_values)
105    elif request.method == 'POST':
106        update()
107        logging.info(request.values)
108        sensor_name = request.form.get('sensor')
109        sensor_value = float(request.form.get('value'))
110        sensor_names = memcache.get('sensor_names')
111        if sensor_names is not None and sensor_name not in sensor_names:

```

```

    sensor = Sensor()
113    sensor.key = ndb.Key('Sensor', sensor_name)
    sensor.put()

115    sensor_names = sensor_name if sensor_names == "" else sensor_names +
                    ", " + sensor_name
117    memcache.set('sensor_names', sensor_names)
    memcache.set(sensor_name + '_value', sensor_value,
                 SENSOR_TIMEOUT_IN_SECONDS)
119    return "Success\n"
121

122    # this function is accessible only using a http GET method, it returns a
123    # json object with the sensor's names list
124    @app.route('/sensor_names', methods=['GET'])
125    def print_names():
        return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])
126

127    # this function is accessible only through GET http method. This GET request
128    # has to contain two parameters one is "sensor", the sensor name, and the
129    # other one is "first_timestamp", the timestamp of the first point that has
130    # to be plotted (this last parameter is not mandatory, if missed it is assumed
131    # equal to 0.
132    @app.route('/graphing_data', methods=['GET'])
133    def graphing_data():
        first_timestamp = request.args.get('first_timestamp')
        sensor_name = request.args.get('sensor')
134        if first_timestamp is None:
            first_timestamp = 0
135        else:
            first_timestamp = float(first_timestamp)
136        points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
        epoch = datetime.datetime(1970, 1, 1)
137        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
138        res.reverse()
139        return json.dumps(res)
140

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

```

```
161     for sensor in Sensor.sensor_list():
162         sensor.key.delete()
163     return "Success"
164
165 # this function is accessible through both GET and POST http methods and it
166 # 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
168 # "Image.jpg" field. Using a PC is also possible to update an image
169 # just using a browser, thank to the GET method implementation
170 @app.route('/picture/submit', methods=['GET', 'POST'])
171 def set_picture():
172     if request.method == 'POST':
173
174         _file = request.files['Image.jpg']
175
176         if _file:
177             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image."
178                                            + _file.filename.rsplit('.', 1)[1],
179                                            mode='w', content_type="image/jpeg")
180             _file.save(cloud_file)
181             cloud_file.close()
182             return "Success"
183
184     else:
185         return ''
186
187         <!doctype html>
188         <title>Upload microscope file V.1</title>
189         <h1>Upload microscope file</h1>
190         <form method="POST">
191             action=""
192             role="form"
193             enctype="multipart/form-data">
194
195             <p><input type=file name=Image.jpg>
196                 <input type=submit value=Upload>
197             </form>
198             ...
199
200         # this function is accessible through a GET http method and it returns the image
201         # stored inside the datastore
202         @app.route('/picture/view', methods=['GET'])
203         def view_picture():
204             for _file in cloudstorage.listbucket(BUCKET_NAME):
205                 if "microscope_image" in _file.filename:
206                     _file = cloudstorage.stat(_file.filename)
207                     logging.info(_file.filename)
208                     logging.info(_file.content_type)
209                     cloud_file = cloudstorage.open(_file.filename, mode='r')
210                     response = make_response(cloud_file.read())
211                     cloud_file.close()
```

```

    response.mimetype = _file.content_type
211     return response
212
213     return "No file found"
214
215 # this function is accessible through both GET and POST http methods and it
216 # is in charge to store the microscope video. This function is normally used
217 # with the POST method where the image is passed through "Video.mp4" field.
218 # Using a PC is also possible to update a video just using a browser,
219 # thank to the GET method implementation
220 @app.route('/video/submit', methods=['GET', 'POST'])
221 def set_video():
222     if request.method == 'GET':
223         return ''
224
225         <!doctype html>
226         <title>Upload microscope video</title>
227         <h1>Upload microscope video</h1>
228         <form method="POST">
229             action=""
230             role="form"
231             enctype="multipart/form-data">
232             <p><input type=file name=Video.mp4>
233             <input type=submit value=Upload>
234             </form>
235         ''
236
237     else:
238         _file = request.files['Video.mp4']
239
240         if _file:
241             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_video." +
242                                         _file.filename.rsplit('.', 1)[1],
243                                         mode='w', content_type="video/mpeg")
244
245             _file.save(cloud_file)
246             cloud_file.close()
247
248             return "Success"
249
250
251 # this function is accessible through a GET http method and it returns the
252 # microscope video stored inside the datastore
253 @app.route('/video/view', methods=['GET'])
254 def view_video():
255     for _file in cloudstorage.listbucket(BUCKET_NAME):
256         if "microscope_video" in _file.filename:
257             _file = cloudstorage.stat(_file.filename)
258             logging.info(_file.filename)
259             logging.info(_file.content_type)
260             cloud_file = cloudstorage.open(_file.filename, mode='r')
261             response = make_response(cloud_file.read())
262             cloud_file.close()
263             response.mimetype = _file.content_type
264
265             return response

```

```

259     return "No file found"
260
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
262 # POST has to have the following parameters: "name", the name that
# identifies the valve, "status", the status of electrovalve (on, off)
263 @app.route('/add/electrovalve', methods=['POST'])
264 def add_electrovalve():
265     key = request.form.get('name')
266     message = request.form.get('status')
267     if key and message:
268         if message == 'on' or message == 'On' or message == 'ON':
269             MESSAGES[key] = True
270             memcache.set(key, True)
271
272     else:
273         MESSAGES[key] = False
274         memcache.set(key, False)
275
276     # ev.put()
277
278     return '%r' % memcache.get(key)
279
280
281 # this function accessible through a GET method returns the value of the
# valve identified through <name> field in the URL
282 @app.route('/electrovalves/<name>', methods=['GET'])
283 def get_electrovalve(name):
284     ev = memcache.get(name);
285     if ev is None:
286         return '%r' % MESSAGES[name] or "%s not found!" % name
287     else:
288         return '%r' % memcache.get(name)
289
290 @app.errorhandler(404)
291 def page_not_found(e):
292     """Return a custom 404 error."""
293     return 'Sorry, nothing at this URL.', 404
294
295
296
297 if __name__ == "__main__":
298     app.run()

```

A.4 GLASSWARE

Listing A.17: MainService.java

```
1 package com.google.android.glass.sample.klabinterface;
2
3 import com.google.android.glass.timeline.LiveCard;
4 import com.google.android.glass.timeline.LiveCard.PublishMode;
5
6 import android.app.PendingIntent;
7 import android.app.Service;
8 import android.content.Context;
9 import android.content.Intent;
10 import android.graphics.Bitmap;
11 import android.graphics.BitmapFactory;
12 import android.net.ConnectivityManager;
13 import android.net.NetworkInfo;
14 import android.os.AsyncTask;
15 import android.os.Environment;
16 import android.os.Handler;
17 import android.os.HandlerThread;
18 import android.os.IBinder;
19 import android.util.Log;
20 import android.view.animation.Animation;
21
22 import com.google.android.glass.timeline.LiveCard;
23 import com.googlecode.charts4j.AxisLabelsFactory;
24 import com.googlecode.charts4j.Data;
25 import com.googlecode.charts4j.GCharts;
26 import com.googlecode.charts4j.LineChart;
27 import com.googlecode.charts4j.Plot;
28 import com.googlecode.charts4j.Plots;
29 import com.jjoe64.graphview.GraphView;
30 import com.jjoe64.graphview.GraphViewSeries;
31
32 import org.json.JSONArray;
33 import org.json.JSONException;
34 import org.json.JSONObject;
35 import org.json.JSONTokener;
36
37 import com.jjoe64.graphview.GraphView;
38 import com.jjoe64.graphview.LineGraphView;
39
40 import java.io.BufferedInputStream;
41 import java.io.File;
42 import java.io.FileOutputStream;
43 import java.io.IOException;
44 import java.io.InputStream;
45 import java.net.HttpURLConnection;
46 import java.net.URL;
47 import java.net.URLConnection;
```

```
import java.nio.channels.FileLock;
50 import java.util.ArrayList;
51 import java.util.Collections;
52 import java.util.HashMap;
53 import java.util.Map;
54
55 /**
56  * Service owning the LiveCard living in the timeline.
57 */
58
59 public class MainService extends Service {
60     /** TAG associated to the LiveCard */
61     private static final String LIVE_CARD_TAG = "BWH interface";
62     /** TAG associated to the Menu view */
63     private static final String MENU_TAG = "Menu";
64     /** TAG associated to the Temperature view */
65     private static final String TEMPERATURE_TAG = "Temperature";
66     /** TAG associated to the PH view */
67     private static final String PH_TAG = "pH";
68     /** TAG associated to the Video view */
69     private static final String VIDEO_TAG = "Video";
70     /** TAG associated to the Beating view */
71     private static final String BEATING_TAG = "Beating";
72
73     private Bitmap bmp;
74
75     /** URL where the image is stored */
76     private static final String URL_IMAGE = "http://planar-contact-601.appspot.com/picture/view";
77
78
79     /** Runnable which describes the task to download the Beating's graph */
80     private final ImageDownloader mImageDownloader = new ImageDownloader(URL_IMAGE, this);
81     /** Action is an enumerate used to implement switch-case for the extra text appended to the intent */
82     private static enum Action
83     {
84         Menu, Temperature, pH, Video, Beating
85     }
86
87
88     private AppDrawer mCallback;
89
90     private LiveCard mLiveCard;
91
92     /** HandlerThread used to launch a background thread that manages the Data updating */
93     private HandlerThread mHandlerThread;
94     /** Handler used to launch a background thread that manages the Data updating */
95     private Handler mHandler;
```

```

98     /** INT associated to the Menu view request */
100    private static final int MENU = 0;
100
101    /** INT associated to the PH view request */
102    private static final int PH = 1;
102
103    /** INT associated to the Menu view request */
104    private static final int TEMPERATURE = 2;
104
105    /** INT associated to the Video view request */
106    private static final int VIDEO = 3;
106
107    /** INT associated to the Beating view request */
108    private static final int BEATING = 4;
108
109
110    /** updating data period (in ms) */
111    private static final long DATA_UPDATE_DELAY_MILLIS = 500;
111
112    /** updating graph period (in ms) */
113    private static final long GRAPH_UPDATE_DELAY_MILLIS = 500;
113
114    /** updating video period (in ms) */
115    private static final long FRAME_TIME_MILLIS = 100;
115
116    private static final long VIDEO_UPDATE_DELAY_MILLIS = 60*1000; //time for video
116
117
118    /** Runnable which describes the task to update the pH and Temperature sensors values */
119    private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable = new UpdateSensorValuesRunn
119
120    /** Runnable which describes the task to compute the pH and Temperature graph (starting from thei
120    private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable = new UpdateSensorGraphsRunn
120
121
122    private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable = new UpdateMicroscope
122
123
124    private static final String VIDEO_FILE_NAME = Environment.getExternalStorageDirectory()+"/microscop
124    private static final String TEMP_VIDEO_FILE_NAME = Environment.getExternalStorageDirectory()+"/temp
124
125
126    /** String array for the sensors (PH and Temperature) */
127    private String[] mSensors = new String[]{PH_TAG, TEMPERATURE_TAG};
127
128
129    /*
129     * Hash table used for creating the graphs
130     */
130
131    /** Hash table in which the sensors values are stored */
132    private Map<String,Double> mCurrentSensorValues;
132
133    /** Hash table in which the graphs points are stored */
134    private Map<String, ArrayList<DataPoint>> mSensorGraphData;
134
135    /** Hash table in which the sensors graphs are stored */
136    private Map<String, Bitmap> mCurrentSensorGraphs;
136
137
138    private Map<String, Double> mSensorAverage;
138
139
140    // private ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_S
140
141
142    @Override
143
144    public IBinder onBind(Intent intent) {
144        return null;
144    }

```

```
148  
149     @Override  
150     public int onStartCommand(Intent intent, int flags, int startId) {  
151         if (mLiveCard == null) {  
152             AppManager.getInstance().setState(MENU);  
153             mLiveCard = new LiveCard(this, LIVE_CARD_TAG);  
154  
155             // Keep track of the callback to remove it before unpublishing.  
156             mCallback = new AppDrawer(this);  
157             mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder().addCallback(mCallback);  
158  
159             Intent menuIntent = new Intent(this, MenuActivity.class);  
160             menuIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);  
161             mLiveCard.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));  
162             mLiveCard.attach(this);  
163             mLiveCard.publish(PublishMode.REVEAL);  
164  
165             /*  
166                 Launch the task to update the data in another thread  
167             mHandlerThread = new HandlerThread("myHandlerThread");  
168             mHandlerThread.start();  
169             mHandler = new Handler(mHandlerThread.getLooper());  
170             DataTask task = new DataTask();  
171             task.execute(this);  
172         } else {  
173             if(intent.getStringExtra(Intent.EXTRA_TEXT)!= null) {  
174                 Action action = Action.valueOf(intent.getStringExtra(Intent.EXTRA_TEXT));  
175                 switch (action) {  
176                     case Menu:  
177                         Log.i(LIVE_CARD_TAG, "State = Menu");  
178                         AppManager.getInstance().setState(MENU);  
179                         break;  
180                     case Beating:  
181                         Log.i(LIVE_CARD_TAG, "State = Beating");  
182                         AppManager.getInstance().setState(BEATING);  
183                         break;  
184                     case pH:  
185                         Log.i(LIVE_CARD_TAG, "State = pH");  
186                         AppManager.getInstance().setState(PH);  
187                         break;  
188                     case Temperature:  
189                         Log.i(LIVE_CARD_TAG, "State = Temperature");  
190                         AppManager.getInstance().setState(TEMPERATURE);  
191                         break;  
192                     case Video:  
193                         Log.i(LIVE_CARD_TAG, "State = Video");  
194                         AppManager.getInstance().setState(VIDEO);  
195                         break;  
196                 default:
```

```

196                     mLIVECARD.navigate();
197                     break;
198                 }
199             }
200         else
201         {
202             mLIVECARD.navigate();
203         }
204     }

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

212 @Override
213 public void onDestroy() {
214     // Stop the Task which update the beating image
215     if(!mImageDownloader.isStopped())
216     {
217         mImageDownloader.setIsStopped(true);
218         Log.i(BEATING_TAG , "Removed Task");
219     }
220     // Stop the Task which update the sensors Graphs
221     if(!mUpdateSensorGraphsRunnable.isStopped())
222     {
223         mUpdateSensorGraphsRunnable.setStop(true);
224         Log.i(PH_TAG + " " + TEMPERATURE_TAG , "Removed Task of Graphs");
225     }
226     // Stop the Task which update the sensors value
227     if(!mUpdateSensorValuesRunnable.isStopped())
228     {
229         mUpdateSensorValuesRunnable.setStop(true);
230         Log.i(PH_TAG + " " + TEMPERATURE_TAG , "Removed Task of Values");
231     }
232     if (!mUpdateMicroscopeVideoRunnable.isStopped())
233     {
234         mUpdateMicroscopeVideoRunnable.setStop(true);
235         Log.i(VIDEO_TAG , "Removed Task of Uploading values");
236     }
237     if (mLiveCard != null && mLIVECARD.isPublished()) {
238         mLIVECARD.unpublish();
239         mLIVECARD = null;
240     }
241     super.onDestroy();
242 }

244 /**

```

```
246     * Asynchronous Task for downloading the graphs and video in background
247     */
248
249     private class DataTask extends AsyncTask<MainService,Void(Void>
250     {
251
252         @Override
253         protected Void doInBackground(MainService... params) {
254
255             Log.i(LIVE_CARD_TAG, "Loading initial data");
256
257             // create the three hash tables
258             mCurrentSensorValues = new HashMap<String, Double>();
259             mSensorGraphData = new HashMap<String, ArrayList<DataPoint>>();
260             mCurrentSensorGraphs = new HashMap<String, Bitmap>();
261             mSensorAverage = new HashMap<String, Double>();
262
263             // initializes each hash map with dummy values
264             for (String mSensor : mSensors)
265             {
266
267                 mCurrentSensorValues.put(mSensor, 0.0);
268                 mSensorAverage.put(mSensor, 0.0);
269                 mSensorGraphData.put(mSensor, new ArrayList<DataPoint>());
270                 mCurrentSensorGraphs.put(mSensor, null);
271             }
272
273             Log.i(LIVE_CARD_TAG, "Download the data");
274
275             //uploads the value captured by PH and Temperature sensors
276             //NetworkInfo ni = cm.getActiveNetworkInfo();
277             //if(ni!= null) {
278
279
280                 // if (ni.isConnected()) {
281                     mUpdateSensorValuesRunnable.run();
282                     // give the hash table to the Callback
283                     // mCallback.setSensorValues(mCurrentSensorValues);
284                     // compute the graphs with previous values and give them to the AppDrawer
285                     mUpdateSensorGraphsRunnable.run();
286
287                     // give the hash table to the Callback
288                     // mCallback.setSensorGraphs(mCurrentSensorGraphs);
289                     // download the beating image and give this to the AppDrawer
290                     mImageDownloader.run();
291
292                     mUpdateMicroscopeVideoRunnable.run();
293                 //}
294             //}
295         }
296     }
```

```

294         return null;
295     }
296 }
297
298 /**
299  * This runnable updates the sensors values taking them from the google engine
300  */
301 private class UpdateSensorValuesRunnable implements Runnable
302 {
303     private boolean mIsStopped = false;
304
305     /** It implements the task of runnable
306      *
307      * @see UpdateSensorValuesRunnable
308      */
309     @Override
310     public void run()
311     {
312         if (!isStopped())
313         {
314             /** JavaScript object in which the sensor values are temporary stored */
315             JSONObject values = getSensorValues();
316             if (values != null)
317             {
318                 for (String mSensor : mSensors)
319                 {
320                     try
321                     {
322                         // update the hash table of sensor values
323                         mCurrentSensorValues.put(mSensor, values.getDouble(mSensor));
324
325                     }
326                     catch (JSONException ignored)
327                     {}
328                 }
329             }
330         }
331
332         // restart the Runnable after a given amount of time
333         mHandler.postDelayed(mUpdateSensorValuesRunnable, DATA_UPDATE_DELAY_MILLIS);
334     }
335 }
336
337
338 /**
339  * It is the getter function that shows the status of runnable
340  *
341  * @return mIsStopped, true if the runnable is stopped, false otherwise
342  */

```

```

    public boolean isStopped()
344    {
345        return mIsStopped;
346    }
347
348    /**
349     * It is the setter that allows to stop the runnable
350     *
351     * @param isStopped, true if the user wishes to stop the runnable, false otherwise
352     */
353    public void setStop(boolean isStopped)
354    {
355        this.mIsStopped = isStopped;
356    }
357
358    /**
359     * It gets the values of all of the sensors
360     *
361     * @return JSONObject which contains all the values of sensors
362     */
363    private JSONObject getSensorValues()
364    {
365        try
366        {
367            String values = getURL("http://planar-contact-601.appspot.com/sensor_values");
368            // return a JSONObject constructed by JSONTokener, which takes a source string and extra
369            return new JSONObject(new JSONTokener(values));
370        }
371        catch (Exception e)
372        {
373            Log.e(LIVE_CARD_TAG,"Failed to get sensor values",e);
374            return null;
375        }
376    }
377
378    /**
379     * Get the new data points that we will need to graph
380
381     */
382    /**
383     * @param sensor , the name of sensor (pH or Temperature)
384     * @param last_timestamp , the previous value of timestamp
385     * @return JSONArray, return a list of values that corresponds to the points that have to be plotted
386     */
387    private JSONArray getDataPoints(String sensor, double last_timestamp)
388    {
389        try
390        {
391            String url = "http://planar-contact-601.appspot.com/graphing_data?sensor=" + sensor + "&last_timestamp=" + last_timestamp;
392            String values = getURL(url);
393        }
394    }

```

```

392         return new JSONArray(new JSONTokener(values));
393     }
394     catch (Exception e)
395     {
396         Log.e(LIVE_CARD_TAG , "Failed to get data points" ,e);
397         return null;
398     }
399 }
400
401 /**
402 * This method gets data from the given url website
403 *
404 * @param _url, url in which the data are contained
405 * @return String, contained data from the given url
406 * @throws IOException if an IO exception occurred during the download
407 */
408 private String getURL(String _url) throws IOException
409 {
410     URL url = new URL(_url);
411     InputStream is = url.openStream();
412     int ptr;
413     StringBuffer buffer = new StringBuffer();
414     while ((ptr = is.read()) != -1)
415     {
416         buffer.append((char)ptr);
417     }
418     return buffer.toString();
419 }
420
421 /**
422 * This runnable updates the graphs of pH and Temperature
423 */
424 private class UpdateSensorGraphsRunnable implements Runnable {
425     private boolean mIsStopped = false;
426     public void run()
427     {
428         if (!mIsStopped())
429         {
430             GraphViewSeries exampleSeries = new GraphViewSeries(new GraphView.GraphViewData[] +
431             {
432                 new GraphView.GraphViewData(1, 2.0d),
433                 new GraphView.GraphViewData(2, 1.5d),
434                 new GraphView.GraphViewData(3, 2.5d),
435                 new GraphView.GraphViewData(4, 1.0d)
436             });
437
438             GraphView graphView = new LineGraphView( MainService.this , "GraphViewDemo");
439             graphView.addSeries(exampleSeries);
440             mCallback.setPHGraphViewer(graphView);
441         }
442     }
443 }

```

```

442

444     // Loop through each of the sensors
445     for (String curr_sensor : mSensors) {
446         ArrayList<DataPoint> curr_data = mSensorGraphData.get(curr_sensor);
447         double lastTimestamp = curr_data.size() > 0 ? curr_data.get(curr_data.size() - 1).getTimestamp() : 0;
448         // Get a list of the new data points for this sensor
449         JSONArray newPoints = getDataPoints(curr_sensor, lastTimestamp);
450         for (int j = 0; j < newPoints.length(); j++) {
451             // Save each data point
452             try {
453                 JSONObject point = newPoints.getJSONObject(j);
454                 double timestamp = (Double) point.get("timestamp");
455                 double value = (Double) point.get("value");
456                 curr_data.add(new DataPoint(timestamp, value));
457             } catch (JSONException e) {
458                 Log.e(LIVE_CARD_TAG, "JSON error", e);
459                 //continue;
460             }
461         }
462         // Clear out points that are over an hour old
463         while (true) {
464             if (curr_data.size() > 0 && curr_data.get(0).getTimestamp() < (curr_data.get(0).getTimestamp() - 3600000)) {
465                 Log.i(LIVE_CARD_TAG, "Deleting old data point");
466                 curr_data.remove(0);
467             } else {
468                 break;
469             }
470         }
471         // If there are no points don't show a graph
472         if (curr_data.size() == 0) {
473             continue;
474         }
475         // Store the timestamps and values in separate arrays for graphing
476         ArrayList<Double> timestamps = new ArrayList<Double>();
477         ArrayList<Double> values = new ArrayList<Double>();
478         for (DataPoint curr_point : curr_data) {
479             timestamps.add(curr_point.getTimestamp());
480             values.add(curr_point.getValue());
481             //System.out.println("Value = " + values.get(j));
482         }

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

```

```

490     double intervalSize = maxTimestamp - minTimestamp;
491     for (int i1 = 0; i1 < timestamps.size(); i1++) {
492         double currTimestamp = timestamps.get(i1);
493         currTimestamp -= minTimestamp;
494         currTimestamp /= intervalSize;
495         currTimestamp *= 100;
496         timestamps.set(i1, currTimestamp);
497     }
498     // Scale the value data
499     double maxVal = Collections.max(values);
500     double minVal = Collections.min(values);
501     maxVal *= 1.2;
502     minVal *= 0.8;
503     intervalSize = maxVal - minVal;
504     for (int i1 = 0; i1 < values.size(); i1++) {
505         double currVal = values.get(i1);
506         currVal -= minVal;
507         currVal /= intervalSize;
508         currVal *= 100;
509         values.set(i1, currVal);
510     }
511
512     // Data xData = Data.newData(timestamps);
513     Data yData = Data.newData(values);
514
515
516     Plot plot = Plots.newPlot(yData);
517     LineChart lineChart = GCharts.newLineChart(plot);
518     lineChart.setSize(400, 200);
519     lineChart.addYAxisLabels(AxisLabelsFactory.newNumericRangeAxisLabels(minVal, maxVal));
520
521
522     mCurrentSensorGraphs.put(curr_sensor, getBitmapFromURL(lineChart.to urlString()));
523 }
524
525
526     mCallback.setSensorAvg(mSensorAverage);
527     mCallback.setSensorGraphs(mCurrentSensorGraphs);
528     Log.i("Main Service", "Graphs updated");
529
530     // restart it after a given amount of time
531     mHandler.postDelayed(mUpdateSensorGraphsRunnable, GRAPH_UPDATE_DELAY_MILLIS);
532 }
533
534
535
536     private double computeAverage(ArrayList<Double> values) {
537         double sum = 0.0;
538         if (!values.isEmpty()) {

```

```
540         for(Double value : values){
541             sum += value;
542         }
543         return sum/values.size();
544     }
545 }
546
547 /**
548 * @return mIsStopped, true if the runnable is stopped, false otherwise
549 */
550 public boolean isStopped()
551 {
552     return mIsStopped;
553 }
554
555 /**
556 * It is the setter that allows to stop the runnable
557 *
558 * @param isStopped, true if the user wishes to stop the runnable, false otherwise
559 */
560 public void setStop(boolean isStopped)
561 {
562     this.mIsStopped = isStopped;
563 }
564 }
565
566 /**
567 */
568 class DataPoint {
569     private final double mTimestamp;
570     private final double mValue;
571     DataPoint(double timestamp, double value) {
572         mTimestamp = timestamp;
573         mValue = value;
574     }
575     public double getTimestamp() {
576         return mTimestamp;
577     }
578     public double getValue() {
579         return mValue;
580     }
581 }
582
583 /**
584 * Runnable that implements the task for downloading beating image */
585 public class ImageDownloader implements Runnable {
586     private String url;
587     private Context c;
588     private boolean mIsStopped = false;
```

```
588     /** Class constructor of ImageDownloader runnable
589      *
590      * @param url, url link of image
591      * @param c, is the context in which the request of downloading image has been sent
592      * @see
593      */
594     public ImageDownloader(String url, Context c)
595     {
596         this.url = url;
597         this.c = c;
598     }
599
600     /** It implements the task of ImageDownloader runnable
601      *
602      * @see
603      */
604     @Override
605     public void run()
606     {
607         if(!isStopped())
608         {
609             bmp = getBitmapFromURL(url);
610
611             mCallback.setBMP(bmp);
612             Log.i(BEATING_TAG, "bmp settato");
613         }
614     }
615
616     /** It is the getter function that shows the status of ImageDownloader runnable
617      *
618      * @return mIsStopped, true if the runnable is stopped, false otherwise
619      */
620     public boolean isStopped()
621     {
622         return mIsStopped;
623     }
624
625
626     /** It is the setter that allows to stop the runnable
627      *
628      * @param isStopped, true if the user wishes to stop the runnable, false otherwise
629      */
630     public void setIsStopped(boolean isStopped)
631     {
632         this.mIsStopped = isStopped;
633     }
634
635     /** This method pulls an image from the given url
636      *
```

```
638     * @param urlLink where the image is stored
639     * @return Bitmap which contains the image of the graph
640     * @throws java.io.IOException if an IO exception occurred during the download
641     */
642     public static Bitmap getBitmapFromURL(String urlLink){
643         try{
644             Log.i(BEATING_TAG, "start downloading image ");
645             long startTime = System.currentTimeMillis();
646             URL url = new URL(urlLink);
647             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
648             connection.setDoInput(true);
649             connection.connect();
650             InputStream inputStream = connection.getInputStream();
651             Log.i(BEATING_TAG, "download completed in "
652                   + ((System.currentTimeMillis() - startTime) / 1000)
653                   + " sec");
654             return BitmapFactory.decodeStream(inputStream);
655         } catch (IOException e) {
656             e.printStackTrace();
657             Log.e(BEATING_TAG, e.getMessage());
658             return null;
659         }
660     }
661
662 //=====
663 // =====
664
665 // Runnable that updates the microscope video
666 private class UpdateMicroscopeVideoRunnable implements Runnable {
667     private boolean mIsStopped = false;
668     public void run() {
669         if (!mIsStopped()) {
670             getMicroscopeVideo();
671             mHandler.postDelayed(mUpdateMicroscopeVideoRunnable, VIDEO_UPDATE_DELAY_MILLIS);
672         }
673     }
674     public boolean isStopped() {
675         return mIsStopped;
676     }
677     public void setStop(boolean isStopped) {
678         this.mIsStopped = isStopped;
679     }
680 }
681
682 // Pull the microscope video from a URL
683 private void getMicroscopeVideo() {
684     try {
685         URL url = new URL("http://planar-contact-601.appspot.com/video/view");
```

```

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

695
696     FileOutputStream outStream = new FileOutputStream(file);

697
698     FileLock lock = outStream.getChannel().lock();
699     byte[] buff = new byte[1024*5];
700     // Read bytes (and store them) until there is nothing more to read(-1)
701     int len;
702     while ((len = inStream.read(buff)) != -1) {
703         outStream.write(buff, 0, len);
704     }
705     // Clean up
706
707     outStream.flush();
708     lock.release();
709     outStream.close();
710     inStream.close();
711     Log.i(VIDEO_TAG, "download completed in "
712           + ((System.currentTimeMillis() - startTime) / 1000)
713           + " sec");
714 }
715 catch (IOException e) {
716     Log.e(VIDEO_TAG, "Failed to download microscope video", e);
717 }
718 }
719
720 }
```

Listing A.18: AppDrawer.java

```

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

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

```
11 import android.os.Handler;
12 import android.os.SystemClock;
13 import android.util.Log;
14 import android.view.SurfaceHolder;
15 import android.view.View;
16
17 import java.util.Map;
18
19 /**
20 * {@link DirectRenderingCallback} used to draw the chronometer on the timeline {@link com.google.android.glass.timeline.DirectRenderingCallback}.
21 * Rendering requires that:
22 * <ol>
23 * <li>a {@link SurfaceHolder} has been created through monitoring the
24 *      {@link SurfaceHolder.Callback#(SurfaceHolder)} and
25 *      {@link SurfaceHolder.Callback#(SurfaceHolder)} callbacks.
26 * <li>rendering has not been paused (defaults to rendering) through monitoring the
27 *      {@link com.google.android.glass.timeline.DirectRenderingCallback#renderingPaused(SurfaceHolder)}.
28 * </ol>
29 * As this class uses an inflated {@link View} to draw on the {@link SurfaceHolder}'s
30 * {@link Canvas}, monitoring the
31 * {@link SurfaceHolder.Callback#(SurfaceHolder, int, int, int)}} callback is also
32 * required to properly measure and layout the {@link View}'s dimension.
33 */
34
35 public class AppDrawer implements DirectRenderingCallback {
36     /** INT associated to the Menu view request */
37     private static final int MENU = 0;
38
39     /** INT associated to the PH view request */
40     private static final int PH = 1;
41
42     /** INT associated to the Menu view request */
43     private static final int TEMPERATURE = 2;
44
45     /** INT associated to the Video view request */
46     private static final int VIDEO = 3;
47
48     /** INT associated to the Beating view request */
49     private static final int BEATING = 4;
50
51
52     private VideoThread mRenderThread;
53
54     /** Bitmap in which the beating image is stored */
55     private Bitmap bmp;
56
57     private boolean mReady;
58
59     private static final String TAG = AppDrawer.class.getSimpleName();
60
61
62     private final MainView mMainView;
63     private final BeatingView mBeatingView;
64     /** View object of the GlassWear pH window */
65     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
76
        @Override
        public void onChange() {
78            // mMainDone = true;
79            //mBeatingView.setBaseMillis(0);
80            updateRenderingState();
81        }
82    };
83
84
    /** Defines the Listener of pH viewer, it is used to communicate with
     * that viewer and allowing its viewing */
85
    private final PHViewer.Listener mPhListener = new PHViewer.Listener(){
86        @Override
87        /* This function is used when the {@link com.example.alik.bwhglass.PHViewer} Class wants
         * to change the view object.
88        */
89        public void onChange(){
90            //state = PH;
91            updateRenderingState();
92        }
93
94    };
95
96
97
98    private final BeatingView.Listener mBeatingListener = new BeatingView.Listener() {
99
100        @Override
101        public void onChange() {
102            updateRenderingState();
103        }
104    };
105
106
107    private final TemperatureView.Listener mTemperatureListener = new TemperatureView.Listener() {
```

```
109
110     @Override
111     public void onChange() {
112         updateRenderingState();
113     }
114
115 /**
116  * Defines the ManageBitmap of Beating viewer, it is used to communicate with
117  * that viewer in order to update the bitmap to be displayed*/
118 private final BeatingView.ManageBitmap mBeatingBitmap = new BeatingView.ManageBitmap(){
119
120     @Override
121     /* Getter that returns the bitmap of the beating image to be displayed
122      *
123      * @return Bitmap, sensor values with the values of pH and Temperature
124      */
125     public Bitmap getBitmap() {
126         return bmp;
127     }
128
129     public boolean isReady(){
130         return mReady;
131     }
132
133 /**
134  * Defines the ManageDataGraph of PH viewer, it is used to communicate with
135  * that viewer in order to update the hash map which contains the graph to be displayed*/
136 private final PHViewer.ManageBitmap mPHManageDataGraph = new PHViewer.ManageBitmap(){
137
138     public Bitmap getBitmap() {
139         return mCurrentSensorGraphs.get("pH");
140     }
141
142     public double getAvg(){return mSensorAverage.get("pH");}
143
144     public boolean isReady(){
145         return mReady;
146     }
147
148 /**
149  * Defines the ManageDataGraph of PH viewer, it is used to communicate with
150  * that viewer in order to update the hash map which contains the graph to be displayed*/
151 private final TemperatureView.ManageBitmap mTemperatureManageDataGraph = new TemperatureView.ManageBitmap(){
152
153     public Bitmap getBitmap() {
154         return mCurrentSensorGraphs.get("pH");
155     }
156
157     public double getAvg(){return mSensorAverage.get("Temperature");}
```

```
159     public boolean isReady(){
160         return mReady;
161     }
162
163     public AppDrawer.(Context context) {
164         this(new MainView(context), new BeatingView(context), new PHViewer(context), new TemperatureView(context));
165     }
166
167     public AppDrawer.(MainView countDownView, BeatingView chronometerView, PHViewer phViewer, TemperatureView temperatureView) {
168
169         mMainView = countDownView;
170         mMainView.setListener(mMainListener);
171
172         mBeatingView = chronometerView;
173         mBeatingView.setListener(mBeatingListener);
174         mBeatingView.setManageBPM(mBeatingBitmap);
175
176         mPhViewer = phViewer;
177         mPhViewer.setListener(mPhListener);
178         mPhViewer.setManageBPM(mPHManageDataGraph);
179
180         mTemperatureView = temperatureView;
181         mTemperatureView.setListener(mTemperatureListener);
182         mTemperatureView.setManageBPM(mTemperatureManageDataGraph);
183
184         mRenderThread = new VideoThread(mHolder);
185         mReady = false;
186     }
187
188
189 /**
190 * Uses the provided {@code width} and {@code height} to measure and layout the inflated
191 * {@link MainView} and {@link BeatingView}.
192 */
193 @Override
194 public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
195     // Measure and layout the view with the canvas dimensions.
196     int measuredWidth = View.MeasureSpec.makeMeasureSpec(width, View.MeasureSpec.EXACTLY);
197     int measuredHeight = View.MeasureSpec.makeMeasureSpec(height, View.MeasureSpec.EXACTLY);
198
199     mMainView.measure(measuredWidth, measuredHeight);
200     mMainView.layout(
201         0, 0, mMainView.getMeasuredWidth(), mMainView.getMeasuredHeight());
202
203     mBeatingView.measure(measuredWidth, measuredHeight);
204     mBeatingView.layout(
205         0, 0, mBeatingView.getMeasuredWidth(), mBeatingView.getMeasuredHeight());
```

```
207         mPhViewer.measure(measuredWidth, measuredHeight);
209         mPhViewer.layout(0, 0, mPhViewer.getMeasuredWidth(), mPhViewer.getMeasuredHeight());
211
212         mTemperatureView.measure(measuredWidth, measuredHeight);
213         mTemperatureView.layout(
214             0, 0, mTemperatureView.getMeasuredWidth(), mTemperatureView.getMeasuredHeight());
215     }
216
217     /**
218      * Keeps the created {@link SurfaceHolder} and updates this class' rendering state.
219      */
220
221     @Override
222     public void surfaceCreated(SurfaceHolder holder) {
223         // The creation of a new Surface implicitly resumes the rendering.
224         mRenderingPaused = false;
225         mHolder = holder;
226         updateRenderingState();
227     }
228
229     /**
230      * Removes the {@link SurfaceHolder} used for drawing and stops rendering.
231      */
232
233     @Override
234     public void surfaceDestroyed(SurfaceHolder holder) {
235         mHolder = null;
236         updateRenderingState();
237     }
238
239     /**
240      * Updates this class' rendering state according to the provided {@code paused} flag.
241      */
242
243     @Override
244     public void renderingPaused(SurfaceHolder holder, boolean paused) {
245         mRenderingPaused = paused;
246         updateRenderingState();
247     }
248
249     /**
250      * Starts or stops rendering according to the {@link com.google.android.glass.timeline.LiveCard}
251      */
252
253     private void updateRenderingState() {
254         if (mHolder != null && !mRenderingPaused) {
255             switch (AppManager.getInstance().getState())
256             {
257                 case MENU:
258                     mRenderThread.quit();
259                     // mMediaPlayer.setDisplay(null);
260                     draw(mMainView);
261             }
262         }
263     }
```

```
257         mBeatingView.stop();
258         mPhViewer.stop();
259         mTemperatureView.stop();
260         mMainView.start();
261         break;
262
263     case BEATING:
264         //  mMediaPlayer.setDisplay(null);
265         draw(mBeatingView);
266         mMainView.stop();
267         mPhViewer.stop();
268         mTemperatureView.stop();
269         mBeatingView.start();
270         break;
271
272     case PH:
273         //  mMediaPlayer.setDisplay(null);
274         draw(mPhViewer);
275         mMainView.stop();
276         mBeatingView.stop();
277         mTemperatureView.stop();
278         mPhViewer.start();
279         break;
280
281     case TEMPERATURE:
282         //  mMediaPlayer.setDisplay(null);
283         draw(mTemperatureView);
284         mMainView.stop();
285         mBeatingView.stop();
286         mPhViewer.stop();
287         mTemperatureView.start();
288         break;
289
290     case VIDEO:
291         //  mMediaPlayer.setDisplay(mHolder);
292         mRenderThread.setShouldRun(true);
293         mRenderThread.start();
294         //  mRenderThread.run();
295         mTemperatureView.stop();
296         mBeatingView.stop();
297         mMainView.stop();
298         mPhViewer.stop();
299
300         break;
301
302     default:
303         //  mMediaPlayer.setDisplay(null);
304         draw(mMainView);
305         mPhViewer.stop();
306         mBeatingView.stop();
307         mTemperatureView.stop();
308         mMainView.start();
309         break;
```

```
305         }
306
307     } else {
308         mMainView.stop();
309         mBeatingView.stop();
310         mPhViewer.stop();
311         mTemperatureView.stop();
312     }
313 }
314
315 /**
316 * Draws the view in the SurfaceHolder's canvas.
317 */
318 private void draw(View view) {
319
320     Canvas canvas;
321
322     try {
323         canvas = mHolder.lockCanvas();
324     } catch (Exception e) {
325         Log.e(TAG, "Unable to lock canvas: " + e);
326         return;
327     }
328     if (canvas != null) {
329
330         view.draw(canvas);
331         mHolder.unlockCanvasAndPost(canvas);
332     }
333 }
334
335 /**
336 * Setter for the beating graph
337 *
338 * @param bmp , Bitmap which contains the beating graph
339 */
340 public void setBMP(Bitmap bmp){
341
342     this.bmp = bmp;
343     this.mReady = true;
344     Log.i("DRAWER", "bmp settato");
345 }
346
347 public void setSensorGraphs( Map<String, Bitmap> sensorGraphs ){
348     this.mCurrentSensorGraphs = sensorGraphs;
349 }
350
351 public void setPHGraphViewer( GraphView graphView ){
352     this.mGraphView = graphView;
353 }
```

```

355     /** Setter for the sensors value hash map
356      *
357      * @param currentSensorValues , Map<String,Double> currentSensorValues which contains the hash
358      *                                 map with the current value of the sensors
359      */
360
361     public void setSensorValues( Map<String,Double> currentSensorValues ){
362         this.mCurrentSensorValues = currentSensorValues;
363     }
364
365
366     public void SetSensorAvg (Map<String,Double> sensorAvg){
367         this.mSensorAverage = sensorAvg;
368     }
369
370 }
371 }
```

Listing A.19: MainView.java

```

1 /*
2  * Copyright (C) 2013 The Android Open Source Project
3  *
4  * Licensed under the Apache License, Version 2.0 (the "License");
5  * you may not use this file except in compliance with the License.
6  * You may obtain a copy of the License at
7  *
8  *     http://www.apache.org/licenses/LICENSE-2.0
9  *
10 * Unless required by applicable law or agreed to in writing, software
11 * distributed under the License is distributed on an "AS IS" BASIS,
12 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * See the License for the specific language governing permissions and
14 * limitations under the License.
15 */
16
17 package com.google.android.glass.sample.klabinterface;
18
19 import android.content.Context;
20 import android.media.AudioManager;
21 import android.media.SoundPool;
22 import android.os.Handler;
23 import android.os.SystemClock;
24 import android.util.AttributeSet;
25 import android.view.LayoutInflater;
26 import android.widget.FrameLayout;
27 import android.widget.TextView;
```

```
29 import java.text.SimpleDateFormat;
30 import java.util.Date;
31 import java.util.concurrent.TimeUnit;
32
33 /**
34  * Animated countdown going from {@code mTimeSeconds} to 0.
35  *
36  * 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
38  *     {@code 0} to {@code ALPHA_DELIMITER}.
39  * 2. From 500ms to 1000ms, update the TextView's alpha from {@code ALPHA_DELIMITER} to {@code 1}.
40  * At each second change, update the TextView text.
41 */
42
43 public class MainView extends FrameLayout {
44
45     /**
46      * Interface to listen for changes in the countdown.
47      */
48
49     public interface Listener {
50
51         /**
52          * Notified when the countdown is finished.
53          */
54         public void onChange();
55     }
56
57
58     /** Time delimiter specifying when the second component is fully shown. */
59     private static final long DELAY_MILLIS = 40;
60
61
62     private final TextView timeText;
63
64
65     private final Handler mHandler = new Handler();
66     private final Runnable mUpdateViewRunnable = new Runnable() {
67
68         @Override
69         public void run() {
70             if (mRunning) {
71                 updateView();
72                 postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
73             }
74         }
75     };
76
77
78     private Listener mListener;
79     private boolean mRunning = false;
```

```
    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    /**
94     * Sets a {@link Listener}.
95     */
96    public void setListener(Listener listener) {
97        mListener = listener;
    }

99    /**
100     * Returns the set {@link Listener}.
101     */
102    public Listener getListener() {
103        return mListener;
    }

104    @Override
105    public boolean postDelayed(Runnable action, long delayMillis) {
106        return mHandler.postDelayed(action, delayMillis);
    }

107    /**
108     * Starts the countdown animation if not yet started.
109     */
110    public void start() {
111        if (!mRunning) {
112            postDelayed(mUpdateViewRunnable, 0);
        }
113        mRunning = true;
    }

114    /**
115     * Stops the chronometer.
116     */
117    public void stop() {
118        if (mRunning) {
```

```

127         removeCallbacks(mUpdateViewRunnable);
128         // mStarted = false;
129     }
130
131     mRunning = false;
132
133
134
135     /**
136      * Updates the view to reflect the current state of animation, visible for testing.
137      *
138      * @return whether or not the count down is finished.
139      */
140
141     void updateView() {
142         //if (mRunning) {
143             timeText.setText( new SimpleDateFormat("hh:mm a").format(new Date()) );
144             // updateView(millisLeft);
145             if (mListener != null) {
146                 mListener.onChange();
147             }
148         //}
149     }
150 }
```

Listing A.20: BeatingView.java

```

package com.google.android.glass.sample.klabinterface;
2
3 import android.content.Context;
4 import android.graphics.Bitmap;
5 import android.graphics.drawable.Drawable;
6 import android.os.Handler;
7 import android.text.Layout;
8 import android.util.AttributeSet;
9 import android.util.Log;
10 import android.view.LayoutInflater;
11 import android.widget.FrameLayout;
12 import android.widget.ImageView;
13 import android.widget.TextView;
14
15
16 /**
17  * View used to display draw a running Chronometer.
18  *
19  * This code is greatly inspired by the Android's Chronometer widget.
20  */
21
22 public class BeatingView extends FrameLayout {
```

```
24     boolean mState = false;
25     private Bitmap bmp;
26
27     /**
28      * Interface to listen for changes on the view layout.
29     */
30     public interface Listener {
31         /** Notified of a change in the view. */
32         public void onChange();
33     }
34
35     /** About 24 FPS, visible for testing. */
36     static final long DELAY_MILLIS = 41;
37
38     private ImageView beatingImage;
39     private final TextView mTitle;
40
41     private final Handler mHandler = new Handler();
42     private final Runnable mUpdateTextRunnable = new Runnable() {
43
44         @Override
45         public void run() {
46             if (mRunning) {
47                 updateText();
48                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
49             }
50         }
51     };
52
53     private boolean mRunning;
54
55     public interface ManageBitmap{
56         public Bitmap getBitmap();
57
58         public boolean isReady();
59     }
60
61     private Listener mChangeListener;
62
63     private ManageBitmap mManage;
64
65     public BeatingView(Context context) {
66         this(context, null, 0);
67     }
68
69     public BeatingView(Context context, AttributeSet attrs) {
70         this(context, attrs, 0);
71     }
```

```
72     public BeatingView(Context context, AttributeSet attrs, int style) {
73         super(context, attrs, style);
74         LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
75         mTitle = (TextView) findViewById(R.id.message);
76         beatingImage = (ImageView) findViewById(R.id.image_left);
77         mTitle.setText("Beating");
78         int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/la
79         beatingImage.setImageResource(id);
80
82     }
84
86 /**
87 * Sets a {@link Listener}.
88 */
89 public void setListener(Listener listener) {
90     mChangeListener = listener;
91 }
92
93 /**
94 * Returns the set {@link Listener}.
95 */
96 public Listener getListener() {
97     return mChangeListener;
98 }
99
100 /**
101 * Starts the chronometer.
102 */
103 public void start() {
104     if (!mRunning) {
105         postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
106     }
107     mRunning = true;
108 }
109
110 /**
111 * Stops the chronometer.
112 */
113 public void stop() {
114     if (mRunning) {
115         removeCallbacks(mUpdateTextRunnable);
116     }
117     mRunning = false;
118 }
119
120 @Override
```

```

122     public boolean postDelayed(Runnable action, long delayMillis) {
123         return mHandler.postDelayed(action, delayMillis);
124     }
125
126     @Override
127     public boolean removeCallbacks(Runnable action) {
128         mHandler.removeCallbacks(action);
129         return true;
130     }
131
132     /**
133      * Sets a {@link Listener}.
134     */
135
136     public void setManageBPM(ManageBitmap manager) {
137         mManage = manager;
138     }
139
140
141     /**
142      * Updates the value of the chronometer, visible for testing.
143     */
144     void updateText() {
145         if(mManage.isReady())
146         {
147             Log.i("viewer", "ce prova");
148             bmp = mManage.getBitmap();
149             beatingImage.setImageBitmap(bmp);
150         }
151         // else
152         //{
153             // int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/beating");
154             // beatingImage.setImageResource(id);
155         //}
156         if (mChangeListener != null) {
157             mChangeListener.onChange();
158         }
159     }
160 }

```

Listing A.21: PHViewer.java

```

1 package com.google.android.glass.sample.klabinterface;
2
3 import android.content.Context;
4 import android.graphics.Bitmap;
5 import android.graphics.drawable.Drawable;

```

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

15
16 import com.jjoe64.graphview.GraphView;
17 import com.jjoe64.graphview.GraphViewSeries;
import com.jjoe64.graphview.LineGraphView;
18
19
21 /**
22 * View used to display draw a running Chronometer.
23 *
24 * This code is greatly inspired by the Android's Chronometer widget.
25 */
26
27 public class PHViewer extends FrameLayout {
28
29     private Context context;
30     private Bitmap bmp;
31
32     /**
33      * Interface to listen for changes on the view layout.
34      */
35
36     public interface Listener {
37         /** Notified of a change in the view. */
38         public void onChange();
39     }
40
41     /**
42      * About 24 FPS, visible for testing.
43      */
44     static final long DELAY_MILLIS = 41;
45
46     private ImageView beatingImage;
47     private final TextView mTextTitle;
48     private TextView AvgView;
49
50     private final Handler mHandler = new Handler();
51     private final Runnable mUpdateTextRunnable = new Runnable() {
52
53         @Override
54         public void run() {
55             if (mRunning) {
56                 updateText();
57                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
58             }
59         }
60     }
61 }
```

```
55         }
56     };
57
58     private boolean mRunning;
59
60     public interface ManageBitmap{
61         public Bitmap getBitmap();
62
63         public boolean isReady();
64
65         public double getAvg();
66     }
67
68
69     private Listener mChangeListener;
70
71     private ManageBitmap mManage;
72
73     public PHViewer(Context context) {
74         this(context, null, 0);
75     }
76
77     public PHViewer(Context context, AttributeSet attrs) {
78         this(context, attrs, 0);
79     }
80
81     public PHViewer(Context context, AttributeSet attrs, int style) {
82         super(context, attrs, style);
83         LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
84         beatingImage = (ImageView) findViewById(R.id.image_left);
85         mTextTitle = (TextView) findViewById(R.id.message);
86         AvgView = (TextView) findViewById(R.id.avg);
87         mTextTitle.setText("PH");
88         int id = getResources().getIdentifier("com.google.android.sample.stopwatch:drawable/layer1");
89         beatingImage.setImageResource(id);
90         //        int id = getResources().getIdentifier("com.google.android.sample.stopwatch:drawable/layer2");
91         //        beatingImage.setImageResource(id);
92
93     }
94
95
96     /**
97      * Sets a {@link Listener}.
98      */
99
100    public void setListener(Listener listener) {
101        mChangeListener = listener;
102    }
103}
```

```
105 /**
106  * Returns the set {@link Listener}.
107 */
108 public Listener getListener() {
109     return mChangeListener;
110 }
111 /**
112  * Starts the chronometer.
113 */
114 public void start() {
115     if (!mRunning) {
116         postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
117     }
118     mRunning = true;
119 }
120 /**
121  * Stops the chronometer.
122 */
123 public void stop() {
124     if (mRunning) {
125         removeCallbacks(mUpdateTextRunnable);
126     }
127     mRunning = false;
128 }
129
130 @Override
131 public boolean postDelayed(Runnable action, long delayMillis) {
132     return mHandler.postDelayed(action, delayMillis);
133 }
134
135 @Override
136 public boolean removeCallbacks(Runnable action) {
137     mHandler.removeCallbacks(action);
138     return true;
139 }
140 /**
141  * Sets a {@link Listener}.
142 */
143 public void setManageBPM(ManageBitmap manager) {
144     mManage = manager;
145 }
146
147 /**
148  * Updates the value of the chronometer, visible for testing.
149 */
150 void updateText() {
```

```

153     if (mManage.isReady())
154     {
155         Log.i("PH", "ce prova");
156         bmp = mManage.getBitmap();
157         AvgView.setText(Double.toString(mManage.getAvg()));
158         beatingImage.setImageBitmap(bmp);
159
160     }
161     // else
162     //{
163     //    int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/beatingImage");
164     //    beatingImage.setImageResource(id);
165     //}
166     if (mChangeListener != null) {
167         mChangeListener.onChange();
168     }
169 }
170
171
172
173
174
175 }
```

Listing A.22: TemperatureView.java

```

package com.google.android.glass.sample.klabinterface;
2
3     import android.content.Context;
4     import android.graphics.Bitmap;
5     import android.graphics.drawable.Drawable;
6     import android.os.Handler;
7     import android.text.Layout;
8     import android.util.AttributeSet;
9     import android.util.Log;
10    import android.view.LayoutInflater;
11    import android.widget.FrameLayout;
12    import android.widget.ImageView;
13    import android.widget.TextView;
14
15
16 /**
17 * View used to display draw a running Chronometer.
18 *
19 * This code is greatly inspired by the Android's Chronometer widget.
20 */
21
22 public class TemperatureView extends FrameLayout {
23
24     private Bitmap bmp;
```

```
24     private TextView AvgView;  
  
26     /**  
27      * Interface to listen for changes on the view layout.  
28      */  
29     public interface Listener {  
30         /** Notified of a change in the view. */  
31         public void onChange();  
32     }  
  
34     /** About 24 FPS, visible for testing. */  
35     static final long DELAY_MILLIS = 41;  
  
36  
37     private ImageView beatingImage;  
38     private final TextView mTitle;  
  
39  
40     private final Handler mHandler = new Handler();  
41     private final Runnable mUpdateTextRunnable = new Runnable() {  
42  
43         @Override  
44         public void run() {  
45             if (mRunning) {  
46                 updateText();  
47                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);  
48             }  
49         }  
50     };  
  
51  
52     private boolean mRunning;  
  
53  
54     public interface ManageBitmap{  
55         public Bitmap getBitmap();  
56  
57         public boolean isReady();  
58  
59         public double getAvg();  
60     }  
  
61  
62     private Listener mChangeListener;  
63  
64     private ManageBitmap mManage;  
65  
66     public TemperatureView(Context context) {  
67         this(context, null, 0);  
68     }  
69  
70     public TemperatureView(Context context, AttributeSet attrs) {  
71         this(context, attrs, 0);  
72     }
```

```
    }

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

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

86    }

88

90    /**
91     * Sets a {@link Listener}.
92     */
93    public void setListener(Listener listener) {
94        mChangeListener = listener;
95    }

96

97    /**
98     * Returns the set {@link Listener}.
99     */
100   public Listener getListener() {
101       return mChangeListener;
102   }

104

105    /**
106     * Starts the chronometer.
107     */
108    public void start() {
109        if (!mRunning) {
110            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
111        }
112        mRunning = true;
113    }

114

115    /**
116     * Stops the chronometer.
117     */
118    public void stop() {
119        if (mRunning) {
120            removeCallbacks(mUpdateTextRunnable);
121        }
122        mRunning = false;
123    }
```

```
122    }
124
125    @Override
126    public boolean postDelayed(Runnable action, long delayMillis) {
127        return mHandler.postDelayed(action, delayMillis);
128    }
129
130    @Override
131    public boolean removeCallbacks(Runnable action) {
132        mHandler.removeCallbacks(action);
133        return true;
134    }
135
136    /**
137     * Sets a {@link Listener}.
138     */
139    public void setManageBPM(ManageBitmap manager) {
140        mManage = manager;
141    }
142
143
144    /**
145     * Updates the value of the chronometer, visible for testing.
146     */
147    void updateText() {
148        if (mManage.isReady())
149        {
150            Log.i("Temperature", "ce prova");
151            bmp = mManage.getBitmap();
152            AvgView.setText(Double.toString(mManage.getAvg()));
153            beatingImage.setImageBitmap(bmp);
154
155        }
156        // else
157        //{
158        //    int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/beat");
159        //    beatingImage.setImageResource(id);
160        //}
161        if (mChangeListener != null) {
162            mChangeListener.onChange();
163        }
164    }
165
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  
21 public class VideoPlayerActivity extends Activity {  
22     private static final int VIDEO_PLAY_REQUEST_CODE = 200;  
23     private static final String TAG = "VIDEO_TAG";  
24     public void onCreate(Bundle savedInstanceState) {  
25         super.onCreate(savedInstanceState);  
26         String filepath;  
27         Bundle extras = getIntent().getExtras();  
28         if (extras != null)  
29             filepath = extras.getString("filepath");  
30         else {  
31             filepath = copyAsset(VIDEO_FILE_NAME);  
32         }  
33  
34         Intent i = new Intent();  
35         i.setAction("com.google.glass.action.VIDEOPLAYER");  
36         i.putExtra("video_url", filepath);  
37         startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);  
38     }  
39     protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
40         if (requestCode == VIDEO_PLAY_REQUEST_CODE)  
41             finish();  
42     }  
43     String copyAsset(String filename) {  
44         final String PATH = Environment.getExternalStorageDirectory().toString() + "/myvideoapps/";  
45         File dir = new File(PATH);  
46         if (!dir.exists()) {  
47             if (!dir.mkdirs()) {  
48                 Log.v(TAG, "ERROR: Creation of directory " + PATH + " on sdcard failed");  
49                 return null;  
50             } else {  
51             }  
52         }  
53     }  
54 }
```

```

        Log.v(TAG, "Created directory " + PATH + " on sdcard");
51    }
52}
53    if (!(new File( PATH + filename).exists())) {
54        try {
55            AssetManager assetManager = getAssets();
56            InputStream in = assetManager.open(filename);
57            OutputStream out = new FileOutputStream(PATH + filename);
58            byte[] buf = new byte[1024];
59            int len;
60            while ((len = in.read(buf)) > 0) {
61                out.write(buf, 0, len);
62            }
63            in.close();
64            out.close();
65        } catch (IOException e) {
66            Log.e(TAG, "Was unable to copy " + filename + e.toString());
67            return null;
68        }
69    }
70    return PATH + filename;
71}
72}

```

Listing A.24: MenuActivity.java

```

package com.google.android.glass.sample.klabinterface;
2
import android.app.Activity;
4 import android.content.Intent;
5 import android.os.Bundle;
6 import android.os.Handler;
7 import android.view.Menu;
8 import android.view.MenuInflater;
9 import android.view.MenuItem;
10 import android.view.View;
11 import android.widget.Toast;
12
13 import java.lang.Runnable;
14
15 /**
16 * Activity showing the stopwatch options menu.
17 */
18 public class MenuActivity extends Activity {
19
20     /** INT associated to the Menu view request */
21     private static final int MENU = 0;
22     /** INT associated to the PH view request */
23     private static final int PH = 1;

```

```
24     /** INT associated to the Menu view request */
25     private static final int TEMPERATURE = 2;
26     /** INT associated to the Video view request */
27     private static final int VIDEO = 3;
28     /** INT associated to the Beating view request */
29     private static final int BEATING = 4;
30
31
32     private final Handler mHandler = new Handler();
33     private int state = 0;
34
35
36     @Override
37     public void onAttachedToWindow() {
38         super.onAttachedToWindow();
39         openOptionsMenu();
40     }
41
42
43     @Override
44     public boolean onCreateOptionsMenu(Menu menu) {
45         MenuInflater inflater = getMenuInflater();
46         inflater.inflate(R.menu.stopwatch, menu);
47         return true;
48     }
49
50
51     @Override
52     public boolean onPreparePanel(int featureId, View view, Menu menu) {
53         AppManager appManager = AppManager.getInstance();
54
55         boolean initialView = appManager.getState() == MENU;
56         boolean imageView = appManager.getState() == PH ||
57             appManager.getState() == TEMPERATURE ||
58             appManager.getState() == BEATING ||
59             appManager.getState() == VIDEO;
60
61         setOptionsMenuState(menu, R.id.action_back, imageView);
62         setOptionsMenuState(menu, R.id.action_view_ph, initialView);
63         setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
64         setOptionsMenuState(menu, R.id.action_view_video, initialView);
65         setOptionsMenuState(menu, R.id.action_view_beating, initialView);
66         setOptionsMenuState(menu, R.id.action_stop, true);
67
68         return true;
69     }
70
71
72     @Override
73     public boolean onOptionsItemSelected(MenuItem item) {
74         // Handle item selection.
75         switch (item.getItemId()) {
76             case R.id.action_stop:
77                 // Stop the service at the end of the message queue for proper options menu
78         }
79     }
80
81
82     private void setOptionsMenuState(Menu menu, int itemId, boolean checked) {
83         MenuItem item = menu.findItem(itemId);
84         if (item != null) {
85             item.setChecked(checked);
86         }
87     }
88
89
90     private void openOptionsMenu() {
91         if (mHandler != null) {
92             mHandler.post(new Runnable() {
93                 @Override
94                 public void run() {
95                     if (mHandler != null) {
96                         mService.startService();
97                     }
98                 }
99             });
100        }
101    }
102
103
104    private void closeOptionsMenu() {
105        if (mHandler != null) {
106            mHandler.post(new Runnable() {
107                @Override
108                public void run() {
109                    if (mHandler != null) {
110                        mService.stopService();
111                    }
112                }
113            });
114        }
115    }
116
117
118    private void startService() {
119        Intent intent = new Intent(this, Service.class);
120        startService(intent);
121    }
122
123
124    private void stopService() {
125        Intent intent = new Intent(this, Service.class);
126        stopService(intent);
127    }
128
129
130    private void updateUI() {
131        if (mHandler != null) {
132            mHandler.post(new Runnable() {
133                @Override
134                public void run() {
135                    if (mHandler != null) {
136                        update();
137                    }
138                }
139            });
140        }
141    }
142
143
144    private void update() {
145        if (mHandler != null) {
146            mHandler.post(new Runnable() {
147                @Override
148                public void run() {
149                    if (mHandler != null) {
150                        update();
151                    }
152                }
153            });
154        }
155    }
156
157
158    private void update() {
159        if (mHandler != null) {
160            mHandler.post(new Runnable() {
161                @Override
162                public void run() {
163                    if (mHandler != null) {
164                        update();
165                    }
166                }
167            });
168        }
169    }
170
171
172    private void update() {
173        if (mHandler != null) {
174            mHandler.post(new Runnable() {
175                @Override
176                public void run() {
177                    if (mHandler != null) {
178                        update();
179                    }
180                }
181            });
182        }
183    }
184
185
186    private void update() {
187        if (mHandler != null) {
188            mHandler.post(new Runnable() {
189                @Override
190                public void run() {
191                    if (mHandler != null) {
192                        update();
193                    }
194                }
195            });
196        }
197    }
198
199
200    private void update() {
201        if (mHandler != null) {
202            mHandler.post(new Runnable() {
203                @Override
204                public void run() {
205                    if (mHandler != null) {
206                        update();
207                    }
208                }
209            });
210        }
211    }
212
213
214    private void update() {
215        if (mHandler != null) {
216            mHandler.post(new Runnable() {
217                @Override
218                public void run() {
219                    if (mHandler != null) {
220                        update();
221                    }
222                }
223            });
224        }
225    }
226
227
228    private void update() {
229        if (mHandler != null) {
230            mHandler.post(new Runnable() {
231                @Override
232                public void run() {
233                    if (mHandler != null) {
234                        update();
235                    }
236                }
237            });
238        }
239    }
240
241
242    private void update() {
243        if (mHandler != null) {
244            mHandler.post(new Runnable() {
245                @Override
246                public void run() {
247                    if (mHandler != null) {
248                        update();
249                    }
250                }
251            });
252        }
253    }
254
255
256    private void update() {
257        if (mHandler != null) {
258            mHandler.post(new Runnable() {
259                @Override
260                public void run() {
261                    if (mHandler != null) {
262                        update();
263                    }
264                }
265            });
266        }
267    }
268
269
270    private void update() {
271        if (mHandler != null) {
272            mHandler.post(new Runnable() {
273                @Override
274                public void run() {
275                    if (mHandler != null) {
276                        update();
277                    }
278                }
279            });
280        }
281    }
282
283
284    private void update() {
285        if (mHandler != null) {
286            mHandler.post(new Runnable() {
287                @Override
288                public void run() {
289                    if (mHandler != null) {
290                        update();
291                    }
292                }
293            });
294        }
295    }
296
297
298    private void update() {
299        if (mHandler != null) {
300            mHandler.post(new Runnable() {
301                @Override
302                public void run() {
303                    if (mHandler != null) {
304                        update();
305                    }
306                }
307            });
308        }
309    }
310
311
312    private void update() {
313        if (mHandler != null) {
314            mHandler.post(new Runnable() {
315                @Override
316                public void run() {
317                    if (mHandler != null) {
318                        update();
319                    }
320                }
321            });
322        }
323    }
324
325
326    private void update() {
327        if (mHandler != null) {
328            mHandler.post(new Runnable() {
329                @Override
330                public void run() {
331                    if (mHandler != null) {
332                        update();
333                    }
334                }
335            });
336        }
337    }
338
339
340    private void update() {
341        if (mHandler != null) {
342            mHandler.post(new Runnable() {
343                @Override
344                public void run() {
345                    if (mHandler != null) {
346                        update();
347                    }
348                }
349            });
350        }
351    }
352
353
354    private void update() {
355        if (mHandler != null) {
356            mHandler.post(new Runnable() {
357                @Override
358                public void run() {
359                    if (mHandler != null) {
360                        update();
361                    }
362                }
363            });
364        }
365    }
366
367
368    private void update() {
369        if (mHandler != null) {
370            mHandler.post(new Runnable() {
371                @Override
372                public void run() {
373                    if (mHandler != null) {
374                        update();
375                    }
376                }
377            });
378        }
379    }
380
381
382    private void update() {
383        if (mHandler != null) {
384            mHandler.post(new Runnable() {
385                @Override
386                public void run() {
387                    if (mHandler != null) {
388                        update();
389                    }
390                }
391            });
392        }
393    }
394
395
396    private void update() {
397        if (mHandler != null) {
398            mHandler.post(new Runnable() {
399                @Override
400                public void run() {
401                    if (mHandler != null) {
402                        update();
403                    }
404                }
405            });
406        }
407    }
408
409
410    private void update() {
411        if (mHandler != null) {
412            mHandler.post(new Runnable() {
413                @Override
414                public void run() {
415                    if (mHandler != null) {
416                        update();
417                    }
418                }
419            });
420        }
421    }
422
423
424    private void update() {
425        if (mHandler != null) {
426            mHandler.post(new Runnable() {
427                @Override
428                public void run() {
429                    if (mHandler != null) {
430                        update();
431                    }
432                }
433            });
434        }
435    }
436
437
438    private void update() {
439        if (mHandler != null) {
440            mHandler.post(new Runnable() {
441                @Override
442                public void run() {
443                    if (mHandler != null) {
444                        update();
445                    }
446                }
447            });
448        }
449    }
450
451
452    private void update() {
453        if (mHandler != null) {
454            mHandler.post(new Runnable() {
455                @Override
456                public void run() {
457                    if (mHandler != null) {
458                        update();
459                    }
460                }
461            });
462        }
463    }
464
465
466    private void update() {
467        if (mHandler != null) {
468            mHandler.post(new Runnable() {
469                @Override
470                public void run() {
471                    if (mHandler != null) {
472                        update();
473                    }
474                }
475            });
476        }
477    }
478
479
480    private void update() {
481        if (mHandler != null) {
482            mHandler.post(new Runnable() {
483                @Override
484                public void run() {
485                    if (mHandler != null) {
486                        update();
487                    }
488                }
489            });
490        }
491    }
492
493
494    private void update() {
495        if (mHandler != null) {
496            mHandler.post(new Runnable() {
497                @Override
498                public void run() {
499                    if (mHandler != null) {
500                        update();
501                    }
502                }
503            });
504        }
505    }
506
507
508    private void update() {
509        if (mHandler != null) {
510            mHandler.post(new Runnable() {
511                @Override
512                public void run() {
513                    if (mHandler != null) {
514                        update();
515                    }
516                }
517            });
518        }
519    }
520
521
522    private void update() {
523        if (mHandler != null) {
524            mHandler.post(new Runnable() {
525                @Override
526                public void run() {
527                    if (mHandler != null) {
528                        update();
529                    }
530                }
531            });
532        }
533    }
534
535
536    private void update() {
537        if (mHandler != null) {
538            mHandler.post(new Runnable() {
539                @Override
540                public void run() {
541                    if (mHandler != null) {
542                        update();
543                    }
544                }
545            });
546        }
547    }
548
549
550    private void update() {
551        if (mHandler != null) {
552            mHandler.post(new Runnable() {
553                @Override
554                public void run() {
555                    if (mHandler != null) {
556                        update();
557                    }
558                }
559            });
560        }
561    }
562
563
564    private void update() {
565        if (mHandler != null) {
566            mHandler.post(new Runnable() {
567                @Override
568                public void run() {
569                    if (mHandler != null) {
570                        update();
571                    }
572                }
573            });
574        }
575    }
576
577
578    private void update() {
579        if (mHandler != null) {
580            mHandler.post(new Runnable() {
581                @Override
582                public void run() {
583                    if (mHandler != null) {
584                        update();
585                    }
586                }
587            });
588        }
589    }
590
591
592    private void update() {
593        if (mHandler != null) {
594            mHandler.post(new Runnable() {
595                @Override
596                public void run() {
597                    if (mHandler != null) {
598                        update();
599                    }
600                }
601            });
602        }
603    }
604
605
606    private void update() {
607        if (mHandler != null) {
608            mHandler.post(new Runnable() {
609                @Override
610                public void run() {
611                    if (mHandler != null) {
612                        update();
613                    }
614                }
615            });
616        }
617    }
618
619
620    private void update() {
621        if (mHandler != null) {
622            mHandler.post(new Runnable() {
623                @Override
624                public void run() {
625                    if (mHandler != null) {
626                        update();
627                    }
628                }
629            });
630        }
631    }
632
633
634    private void update() {
635        if (mHandler != null) {
636            mHandler.post(new Runnable() {
637                @Override
638                public void run() {
639                    if (mHandler != null) {
640                        update();
641                    }
642                }
643            });
644        }
645    }
646
647
648    private void update() {
649        if (mHandler != null) {
650            mHandler.post(new Runnable() {
651                @Override
652                public void run() {
653                    if (mHandler != null) {
654                        update();
655                    }
656                }
657            });
658        }
659    }
660
661
662    private void update() {
663        if (mHandler != null) {
664            mHandler.post(new Runnable() {
665                @Override
666                public void run() {
667                    if (mHandler != null) {
668                        update();
669                    }
670                }
671            });
672        }
673    }
674
675
676    private void update() {
677        if (mHandler != null) {
678            mHandler.post(new Runnable() {
679                @Override
680                public void run() {
681                    if (mHandler != null) {
682                        update();
683                    }
684                }
685            });
686        }
687    }
688
689
690    private void update() {
691        if (mHandler != null) {
692            mHandler.post(new Runnable() {
693                @Override
694                public void run() {
695                    if (mHandler != null) {
696                        update();
697                    }
698                }
699            });
700        }
701    }
702
703
704    private void update() {
705        if (mHandler != null) {
706            mHandler.post(new Runnable() {
707                @Override
708                public void run() {
709                    if (mHandler != null) {
710                        update();
711                    }
712                }
713            });
714        }
715    }
716
717
718    private void update() {
719        if (mHandler != null) {
720            mHandler.post(new Runnable() {
721                @Override
722                public void run() {
723                    if (mHandler != null) {
724                        update();
725                    }
726                }
727            });
728        }
729    }
730
731
732    private void update() {
733        if (mHandler != null) {
734            mHandler.post(new Runnable() {
735                @Override
736                public void run() {
737                    if (mHandler != null) {
738                        update();
739                    }
740                }
741            });
742        }
743    }
744
745
746    private void update() {
747        if (mHandler != null) {
748            mHandler.post(new Runnable() {
749                @Override
750                public void run() {
751                    if (mHandler != null) {
752                        update();
753                    }
754                }
755            });
756        }
757    }
758
759
760    private void update() {
761        if (mHandler != null) {
762            mHandler.post(new Runnable() {
763                @Override
764                public void run() {
765                    if (mHandler != null) {
766                        update();
767                    }
768                }
769            });
770        }
771    }
772
773
774    private void update() {
775        if (mHandler != null) {
776            mHandler.post(new Runnable() {
777                @Override
778                public void run() {
779                    if (mHandler != null) {
780                        update();
781                    }
782                }
783            });
784        }
785    }
786
787
788    private void update() {
789        if (mHandler != null) {
790            mHandler.post(new Runnable() {
791                @Override
792                public void run() {
793                    if (mHandler != null) {
794                        update();
795                    }
796                }
797            });
798        }
799    }
800
801
802    private void update() {
803        if (mHandler != null) {
804            mHandler.post(new Runnable() {
805                @Override
806                public void run() {
807                    if (mHandler != null) {
808                        update();
809                    }
810                }
811            });
812        }
813    }
814
815
816    private void update() {
817        if (mHandler != null) {
818            mHandler.post(new Runnable() {
819                @Override
820                public void run() {
821                    if (mHandler != null) {
822                        update();
823                    }
824                }
825            });
826        }
827    }
828
829
830    private void update() {
831        if (mHandler != null) {
832            mHandler.post(new Runnable() {
833                @Override
834                public void run() {
835                    if (mHandler != null) {
836                        update();
837                    }
838                }
839            });
840        }
841    }
842
843
844    private void update() {
845        if (mHandler != null) {
846            mHandler.post(new Runnable() {
847                @Override
848                public void run() {
849                    if (mHandler != null) {
850                        update();
851                    }
852                }
853            });
854        }
855    }
856
857
858    private void update() {
859        if (mHandler != null) {
860            mHandler.post(new Runnable() {
861                @Override
862                public void run() {
863                    if (mHandler != null) {
864                        update();
865                    }
866                }
867            });
868        }
869    }
870
871
872    private void update() {
873        if (mHandler != null) {
874            mHandler.post(new Runnable() {
875                @Override
876                public void run() {
877                    if (mHandler != null) {
878                        update();
879                    }
880                }
881            });
882        }
883    }
884
885
886    private void update() {
887        if (mHandler != null) {
888            mHandler.post(new Runnable() {
889                @Override
890                public void run() {
891                    if (mHandler != null) {
892                        update();
893                    }
894                }
895            });
896        }
897    }
898
899
900    private void update() {
901        if (mHandler != null) {
902            mHandler.post(new Runnable() {
903                @Override
904                public void run() {
905                    if (mHandler != null) {
906                        update();
907                    }
908                }
909            });
910        }
911    }
912
913
914    private void update() {
915        if (mHandler != null) {
916            mHandler.post(new Runnable() {
917                @Override
918                public void run() {
919                    if (mHandler != null) {
920                        update();
921                    }
922                }
923            });
924        }
925    }
926
927
928    private void update() {
929        if (mHandler != null) {
930            mHandler.post(new Runnable() {
931                @Override
932                public void run() {
933                    if (mHandler != null) {
934                        update();
935                    }
936                }
937            });
938        }
939    }
940
941
942    private void update() {
943        if (mHandler != null) {
944            mHandler.post(new Runnable() {
945                @Override
946                public void run() {
947                    if (mHandler != null) {
948                        update();
949                    }
950                }
951            });
952        }
953    }
954
955
956    private void update() {
957        if (mHandler != null) {
958            mHandler.post(new Runnable() {
959                @Override
960                public void run() {
961                    if (mHandler != null) {
962                        update();
963                    }
964                }
965            });
966        }
967    }
968
969
970    private void update() {
971        if (mHandler != null) {
972            mHandler.post(new Runnable() {
973                @Override
974                public void run() {
975                    if (mHandler != null) {
976                        update();
977                    }
978                }
979            });
980        }
981    }
982
983
984    private void update() {
985        if (mHandler != null) {
986            mHandler.post(new Runnable() {
987                @Override
988                public void run() {
989                    if (mHandler != null) {
990                        update();
991                    }
992                }
993            });
994        }
995    }
996
997
998    private void update() {
999        if (mHandler != null) {
1000            mHandler.post(new Runnable() {
1001                @Override
1002                public void run() {
1003                    if (mHandler != null) {
1004                        update();
1005                    }
1006                }
1007            });
1008        }
1009    }
1010
1011
1012    private void update() {
1013        if (mHandler != null) {
1014            mHandler.post(new Runnable() {
1015                @Override
1016                public void run() {
1017                    if (mHandler != null) {
1018                        update();
1019                    }
1020                }
1021            });
1022        }
1023    }
1024
1025
1026    private void update() {
1027        if (mHandler != null) {
1028            mHandler.post(new Runnable() {
1029                @Override
1030                public void run() {
1031                    if (mHandler != null) {
1032                        update();
1033                    }
1034                }
1035            });
1036        }
1037    }
1038
1039
1040    private void update() {
1041        if (mHandler != null) {
1042            mHandler.post(new Runnable() {
1043                @Override
1044                public void run() {
1045                    if (mHandler != null) {
1046                        update();
1047                    }
1048                }
1049            });
1050        }
1051    }
1052
1053
1054    private void update() {
1055        if (mHandler != null) {
1056            mHandler.post(new Runnable() {
1057                @Override
1058                public void run() {
1059                    if (mHandler != null) {
1060                        update();
1061                    }
1062                }
1063            });
1064        }
1065    }
1066
1067
1068    private void update() {
1069        if (mHandler != null) {
1070            mHandler.post(new Runnable() {
1071                @Override
1072                public void run() {
1073                    if (mHandler != null) {
1074                        update();
1075                    }
1076                }
1077            });
1078        }
1079    }
1080
1081
1082    private void update() {
1083        if (mHandler != null) {
1084            mHandler.post(new Runnable() {
1085                @Override
1086                public void run() {
1087                    if (mHandler != null) {
1088                        update();
1089                    }
1090                }
1091            });
1092        }
1093    }
1094
1095
1096    private void update() {
1097        if (mHandler != null) {
1098            mHandler.post(new Runnable() {
1099                @Override
1100                public void run() {
1101                    if (mHandler != null) {
1102                        update();
1103                    }
1104                }
1105            });
1106        }
1107    }
1108
1109
1110    private void update() {
1111        if (mHandler != null) {
1112            mHandler.post(new Runnable() {
1113                @Override
1114                public void run() {
1115                    if (mHandler != null) {
1116                        update();
1117                    }
1118                }
1119            });
1120        }
1121    }
1122
1123
1124    private void update() {
1125        if (mHandler != null) {
1126            mHandler.post(new Runnable() {
1127                @Override
1128                public void run() {
1129                    if (mHandler != null) {
1130                        update();
1131                    }
1132                }
1133            });
1134        }
1135    }
1136
1137
1138    private void update() {
1139        if (mHandler != null) {
1140            mHandler.post(new Runnable() {
1141                @Override
1142                public void run() {
1143                    if (mHandler != null) {
1144                        update();
1145                    }
1146                }
1147            });
1148        }
1149    }
1150
1151
1152    private void update() {
1153        if (mHandler != null) {
1154            mHandler.post(new Runnable() {
1155                @Override
1156                public void run() {
1157                    if (mHandler != null) {
1158                        update();
1159                    }
1160                }
1161            });
1162        }
1163    }
1164
1165
1166    private void update() {
1167        if (mHandler != null) {
1168            mHandler.post(new Runnable() {
1169                @Override
1170                public void run() {
1171                    if (mHandler != null) {
1172                        update();
1173                    }
1174                }
1175            });
1176        }
1177    }
1178
1179
1180    private void update() {
1181        if (mHandler != null) {
1182            mHandler.post(new Runnable() {
1183                @Override
1184                public void run() {
1185                    if (mHandler != null) {
1186                        update();
1187                    }
1188                }
1189            });
1190        }
1191    }
1192
1193
1194    private void update() {
1195        if (mHandler != null) {
1196            mHandler.post(new Runnable() {
1197                @Override
1198                public void run() {
1199                    if (mHandler != null) {
1200                        update();
1201                    }
1202                }
1203            });
1204        }
1205    }
1206
1207
1208    private void update() {
1209        if (mHandler != null) {
1210            mHandler.post(new Runnable() {
1211                @Override
1212                public void run() {
1213                    if (mHandler != null) {
1214                        update();
1215                    }
1216                }
1217            });
1218        }
1219    }
1220
1221
1222    private void update() {
1223        if (mHandler != null) {
1224            mHandler.post(new Runnable() {
1225                @Override
1226                public void run() {
1227                    if (mHandler != null) {
1228                        update();
1229                    }
1230                }
1231            });
1232        }
1233    }
1234
1235
1236    private void update() {
1237        if (mHandler != null) {
1238            mHandler.post(new Runnable() {
1239                @Override
1240                public void run() {
1241                    if (mHandler != null) {
1242                        update();
1243                    }
1244                }
1245            });
1246        }
1247    }
1248
1249
1250    private void update() {
1251        if (mHandler != null) {
1252            mHandler.post(new Runnable() {
1253                @Override
1254                public void run() {
1255                    if (mHandler != null) {
1256                        update();
1257                    }
1258                }
1259            });
1260        }
1261    }
1262
1263
1264    private void update() {
1265        if (mHandler != null) {
1266            mHandler.post(new Runnable() {
1267                @Override
1268                public void run() {
1269                    if (mHandler != null) {
1270                        update();
1271                    }
1272                }
1273            });
1274        }
1275    }
1276
1277
1278    private void update() {
1279        if (mHandler != null) {
1280            mHandler.post(new Runnable() {
1281                @Override
1282                public void run() {
1283                    if (mHandler != null) {
1284                        update();
1285                    }
1286                }
1287            });
1288        }
1289    }
1290
1291
1292    private void update() {
1293        if (mHandler != null) {
1294            mHandler.post(new Runnable() {
1295                @Override
1296                public void run() {
1297                    if (mHandler != null) {
1298                        update();
1299                    }
1300                }
1301            });
1302        }
1303    }
1304
1305
1306    private void update() {
1307        if (mHandler != null) {
1308            mHandler.post(new Runnable() {
1309                @Override
1310                public void run() {
1311                    if (mHandler != null) {
1312                        update();
1313                    }
1314                }
1315            });
1316        }
1317    }
1318
1319
1320    private void update() {
1321        if (mHandler != null) {
1322            mHandler.post(new Runnable() {
1323                @Override
1324                public void run() {
1325                    if (mHandler != null) {
1326                        update();
1327                    }
1328                }
1329            });
1330        }
1331    }
1332
1333
1334    private void update() {
1335        if (mHandler != null) {
1336            mHandler.post(new Runnable() {
1337                @Override
1338                public void run() {
1339                    if (mHandler != null) {
1340                        update();
1341                    }
1342                }
1343            });
1344        }
1345    }
1346
1347
1348    private void update() {
1349        if (mHandler != null) {
1350            mHandler.post(new Runnable() {
1351                @Override
1352                public void run() {
1353                    if (mHandler != null) {
1354                        update();
1355                    }
1356                }
1357            });
1358        }
1359    }
1360
1361
1362    private void update() {
1363        if (mHandler != null) {
1364            mHandler.post(new Runnable() {
1365                @Override
1366                public void run() {
1367                    if (mHandler != null) {
1368                        update();
1369                    }
1370                }
1371            });
1372        }
1373    }
1374
1375
1376    private void update() {
1377        if (mHandler != null) {
1378            mHandler.post(new Runnable() {
1379                @Override
1380                public void run() {
1381                    if (mHandler != null) {
1382                        update();
1383                    }
1384                }
1385            });
1386        }
1387    }
1388
1389
1390    private void update() {
1391        if (mHandler != null) {
1392            mHandler.post(new Runnable() {
1393                @Override
1394                public void run() {
1395                    if (mHandler != null) {
1396                        update();
1397                    }
1398                }
1399            });
1400        }
1401    }
1402
1403
1404    private void update() {
1405        if (mHandler != null) {
1406            mHandler.post(new Runnable() {
1407                @Override
1408                public void run() {
1409                    if (mHandler != null) {
1410                        update();
1411                    }
1412                }
1413            });
1414        }
1415    }
1416
1417
1418    private void update() {
1419        if (mHandler != null) {
1420            mHandler.post(new Runnable() {
1421                @Override
1422                public void run() {
1423                    if (mHandler != null) {
1424                        update();
1425                    }
1426                }
1427            });
1428        }
1429    }
1430
1431
1432    private void update() {
1433        if (mHandler != null) {
1434            mHandler.post(new Runnable() {
1435                @Override
1436                public void run() {
1437                    if (mHandler != null) {
1438                        update();
1439                    }
1440                }
1441            });
1442        }
1443    }
1444
1445
1446    private void update() {
1447        if (mHandler != null) {
1448            mHandler.post(new Runnable() {
1449                @Override
1450                public void run() {
1451                    if (mHandler != null) {
1452                        update();
1453                    }
1454                }
1455            });
1456        }
1457    }
1458
1459
1460    private void update() {
1461        if (mHandler != null) {
1462            mHandler.post(new Runnable() {
1463                @Override
1464                public void run() {
1465                    if (mHandler != null) {
1466                        update();
1467                    }
1468                }
1469            });
1470        }
1471    }
1472
1473
1474    private void update() {
1475        if (mHandler != null) {
1476            mHandler.post(new Runnable() {
1477                @Override
1478                public void run() {
1479                    if (mHandler != null) {
1480                        update();
1481                    }
1482                }
1483            });
1484        }
1485    }
1486
1487
1488    private void update() {
1489        if (mHandler != null) {
1490            mHandler.post(new Runnable() {
1491                @Override
1492                public void run() {
1493                    if (mHandler != null) {
1494                        update();
1495                    }
1496                }
1497            });
1498        }
1499    }
1500
1501
1502    private void update() {
1503        if (mHandler != null) {
1504            mHandler.post(new Runnable() {
1505                @Override
1506                public void run() {
1507                    if (mHandler != null) {
1508                        update();
1509                    }
1510                }
1511            });
1512        }
1513    }
1514
1515
1516    private void update() {
1517        if (mHandler != null) {
1518            mHandler.post(new
```

```
// animation. This is only needed when starting a new Activity or stopping a Service
74    // that published a LiveCard.
75    post(new Runnable() {
76
76        @Override
77        public void run() {
78            stopService(new Intent(MenuActivity.this, StopwatchService.class));
79        }
80    );
81    return true;
82
83    case R.id.action_view_beating:
84        handleViewBeating();
85        return true;
86
87    case R.id.action_back:
88        handleViewBack();
89        return true;
90
91    case R.id.action_view_temperature:
92        handleViewTemperature();
93        return true;
94
95    case R.id.action_view_ph:
96        handleViewPh();
97        return true;
98
99    case R.id.action_view_video:
100       handleViewVideo();
101       return true;
102
103    default:
104        return super.onOptionsItemSelected(item);
105    }
106}
107
108
109 private void handleViewVideo() {
110     Toast.makeText(this, "Video", Toast.LENGTH_SHORT).show();
111     // launch a new thread for starting a new live card
112     mHandler.post(new Runnable() {
113
114         @Override
115         public void run() {
116             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
117             intent.putExtra(Intent.EXTRA_TEXT, "Video");
118             startService(intent);
119         }
120     );
121 }
122
123
124 private void handleViewTemperature() {
125     Toast.makeText(this, "Temperature", Toast.LENGTH_SHORT).show();
126     // launch a new thread for starting a new live card
127     mHandler.post(new Runnable() {
128
129         @Override
130         public void run() {
```

```

122         Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
123         intent.putExtra(Intent.EXTRA_TEXT, "Temperature");
124         startService(intent);
125     }
126 }
127
128 private void handleViewPh() {
129     Toast.makeText(this, "Ph", Toast.LENGTH_SHORT).show();
130     // launch a new thread for starting a new live card
131     mHandler.post(new Runnable() {
132         @Override
133         public void run() {
134             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
135             intent.putExtra(Intent.EXTRA_TEXT, "pH");
136             startService(intent);
137         }
138     });
139 }
140
141
142 private void handleViewBack() {
143     Toast.makeText(this, "Menu", Toast.LENGTH_SHORT).show();
144     // launch a new thread for starting a new live card
145     mHandler.post(new Runnable() {
146         @Override
147         public void run() {
148             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
149             intent.putExtra(Intent.EXTRA_TEXT, "Menu");
150             startService(intent);
151         }
152     });
153 }
154
155 @Override
156 public void onOptionsMenuClosed(Menu menu) {
157     // Nothing else to do, closing the Activity.
158     finish();
159 }
160
161 /**
162 * Posts a {@link Runnable} at the end of the message loop, overridable for testing.
163 */
164 protected void post(Runnable runnable) {
165     mHandler.post(runnable);
166 }
167
168 /**
169 * The function handle the request to view the beating plot
170 */

```

```

172     private void handleViewBeating() {
173         Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
174         // launch a new thread for starting a new live card
175         mHandler.post(new Runnable() {
176             @Override
177             public void run() {
178                 Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
179                 intent.putExtra(Intent.EXTRA_TEXT, "Beating");
180                 startService(intent);
181             }
182         });
183     }
184
185     private static void setOptionsMenuState(Menu menu, int menuItemId, boolean enabled){
186         MenuItem menuItem = menu.findItem(menuItemId);
187         menuItem.setVisible(enabled);
188         menuItem.setEnabled(enabled);
189     }
190 }
```

Listing A.25: AppManager.java

```

package com.google.android.glass.sample.klabinterface;
2
3 /**
4  * Created by alik on 12/2/2014.
5  */
6 public class AppManager {
7
8     private int state;
9
10    private static AppManager instance = new AppManager();
11
12    public static AppManager getInstance(){
13        return instance;
14    }
15
16    public void setState(int value){
17        state = value;
18    }
19
20    public int getState(){
21        return state;
22    }
23
24 }
```

Listing A.26: DataPoint.java

```
1 package com.google.android.glass.sample.klabinterface;  
2  
3 /**  
4  * Created by Buso on 28/10/2014.  
5 */  
6 class DataPoint {  
7     private final double mTimestamp;  
8     private final double mValue;  
9     DataPoint(double timestamp, double value) {  
10         mTimestamp = timestamp;  
11         mValue = value;  
12     }  
13     public double getTimestamp() {  
14         return mTimestamp;  
15     }  
16     public double getValue() {  
17         return mValue;  
18     }  
19 }
```

[7]

LIST OF FIGURES

Figure 1	<i>XCEL</i> project (<i>Body-on-a-chip</i>)	1
Figure 2	Block diagram of the system	2
Figure 3	Glasswear's Block Diagram	3
Figure 4	Drive Electrovalves Steps	4
Figure 5	<i>Back</i> menu item	4
Figure 6	<i>Exit</i> 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	<i>DC-DC</i> High Speed Switching Path	28
Figure 31	<i>PCB</i> Layout of <i>DC-DC</i>	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 <i>IoT</i> Evolution Prediction	34
Figure 36	The <i>Beaglebone Black</i>	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 Brower	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>.