

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	35
iii SOFTWARE	36
5 GOOGLE APP ENGINE	37
6 MICROSCOPE VIDEO STORING	38
iv GOOGLE GLASS APPLICATION	39
7 INTRODUCTION	40
v CONCLUSION AND APPENDIX	41
8 TEST AND PERFORMANCE	42
8.1 LED Experiments	42
8.2 Electrovalves experiments	43
8.2.1 Breadboard Phase	43
8.2.2 PCB Phase	45

A CODE	46
A.1 Firmware	46
A.2 Video Storing Software	66
A.3 Google App Engine	72
A.4 Glassware	79
List of Figures	123
List of Listings	124
Bibliography	125

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*.

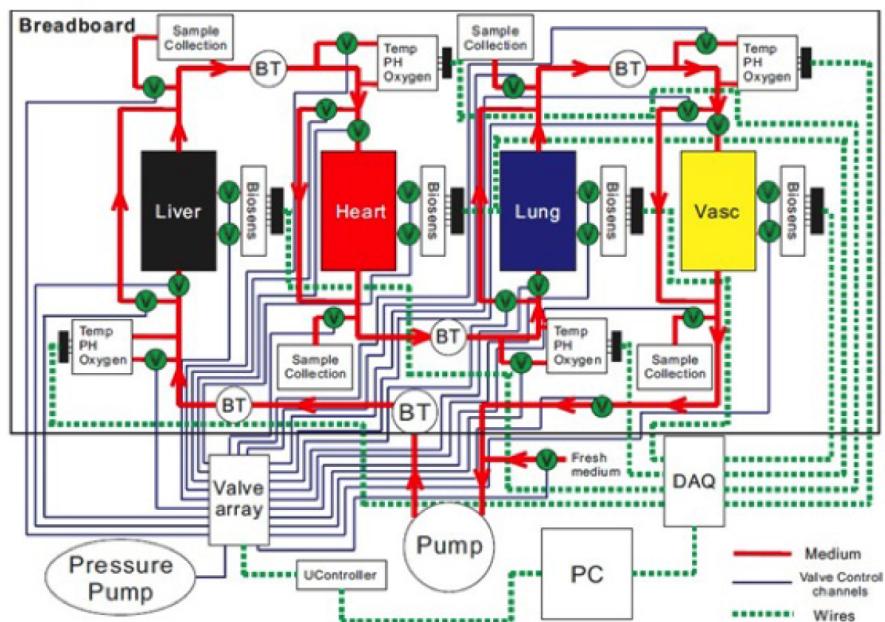


Figure 1: XCEL project (*Body-on-a-chip*)

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).

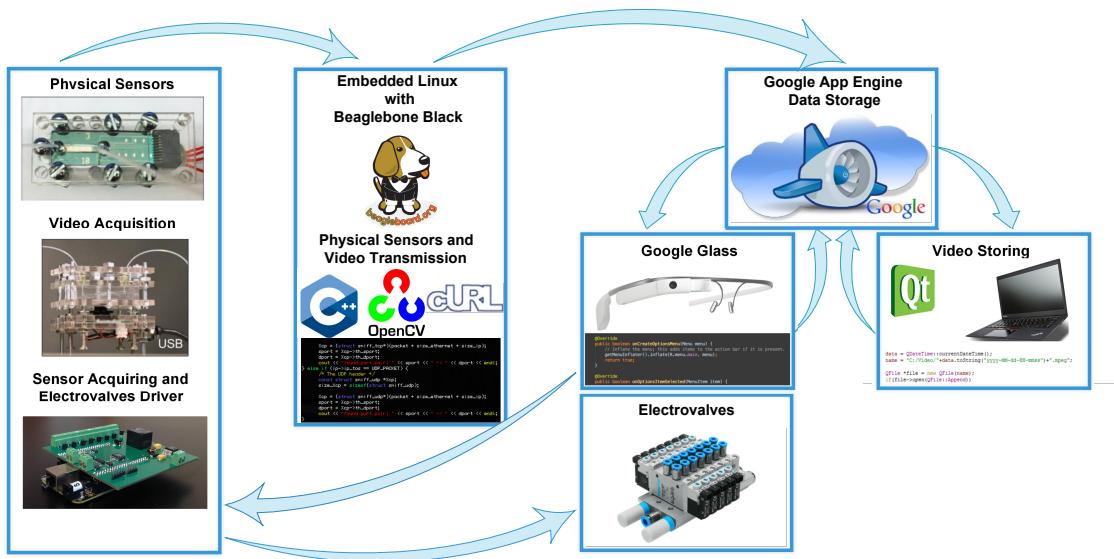


Figure 2: Block diagram of the system

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.

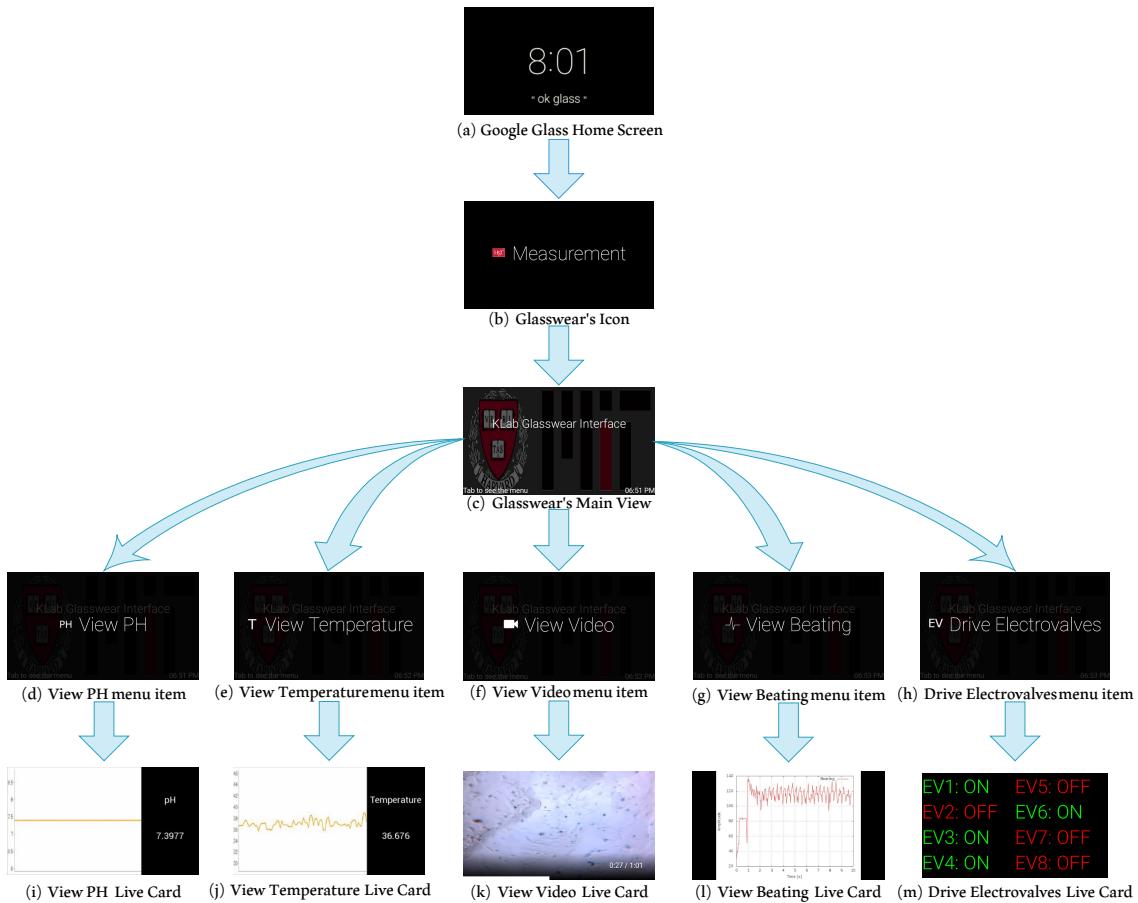


Figure 3: Glasswear's Block Diagram

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.

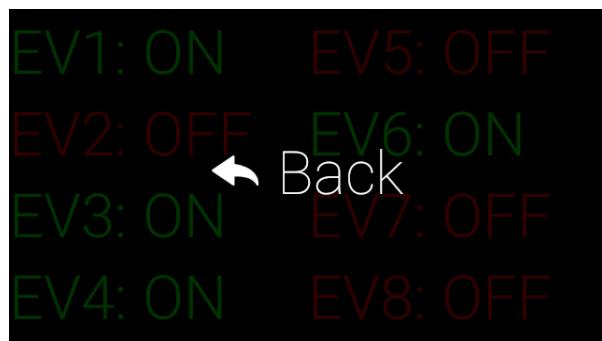


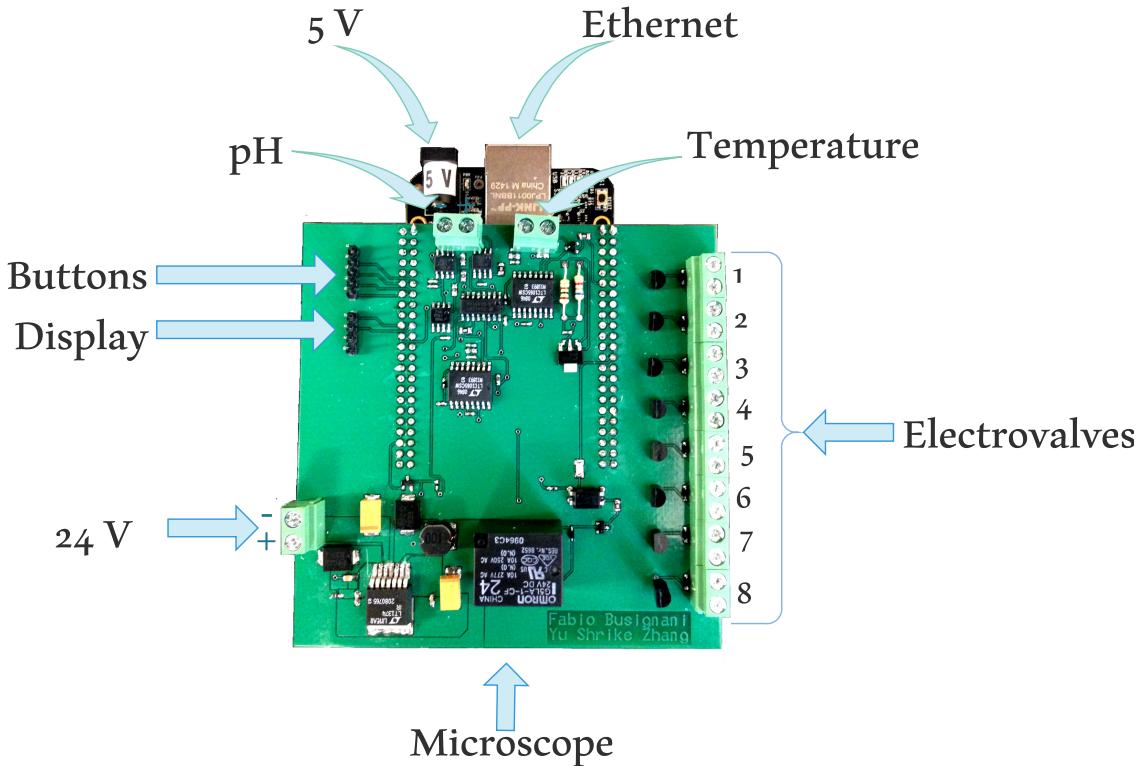
Figure 5: *Back* menu item

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 fro the top (as shown in (Fig.7))

VIDEO STORING

The storing of the microscope video plays an important role of this system. It may be essential to review the recorded video during the experiment and in order to fulfill this aim a *Qt* program has been designed.

We chose *Qt* because in this way the program is available for different operating systems, *Linux*, *Windows*, and *MacOS*.

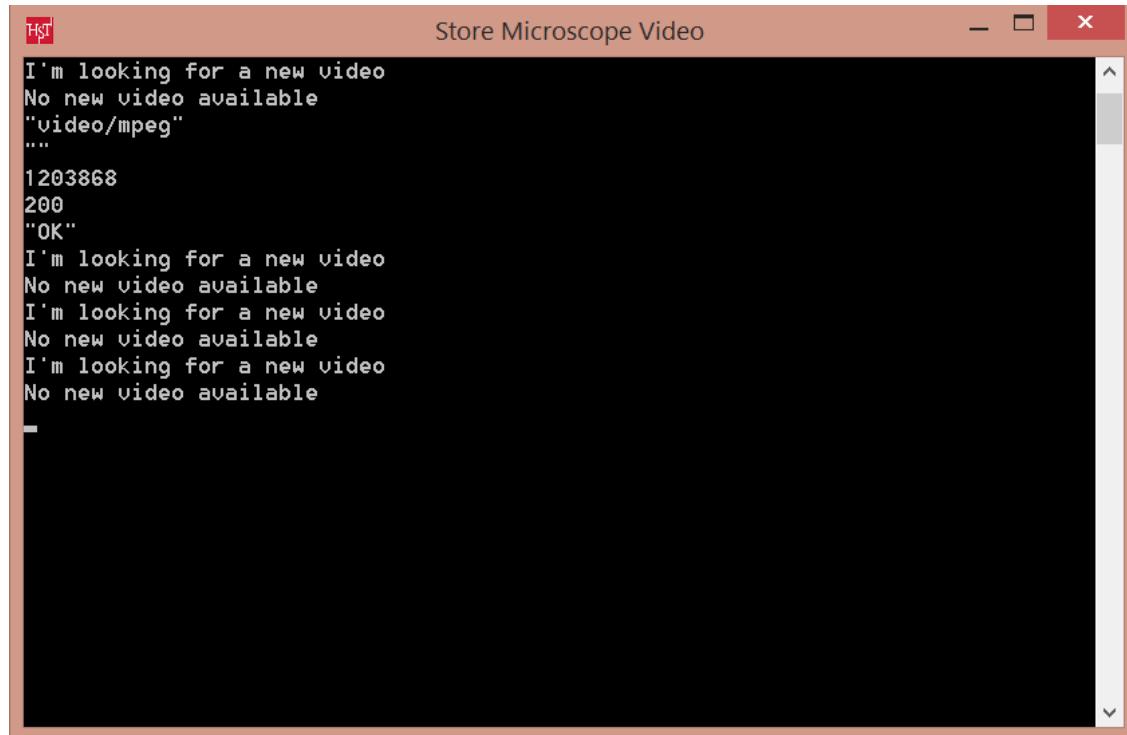
The program is very easy to use, the user just has to run the executable (Fig.8).



Figure 8: Storing Microscope Video Program's Icon

Once it has been launched, a console is opened (Fig.9). The program checks every 20 seconds if a new video has been uploaded on the server. If so, the new video will be stored inside the computer (directory C:/Video) with the current 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

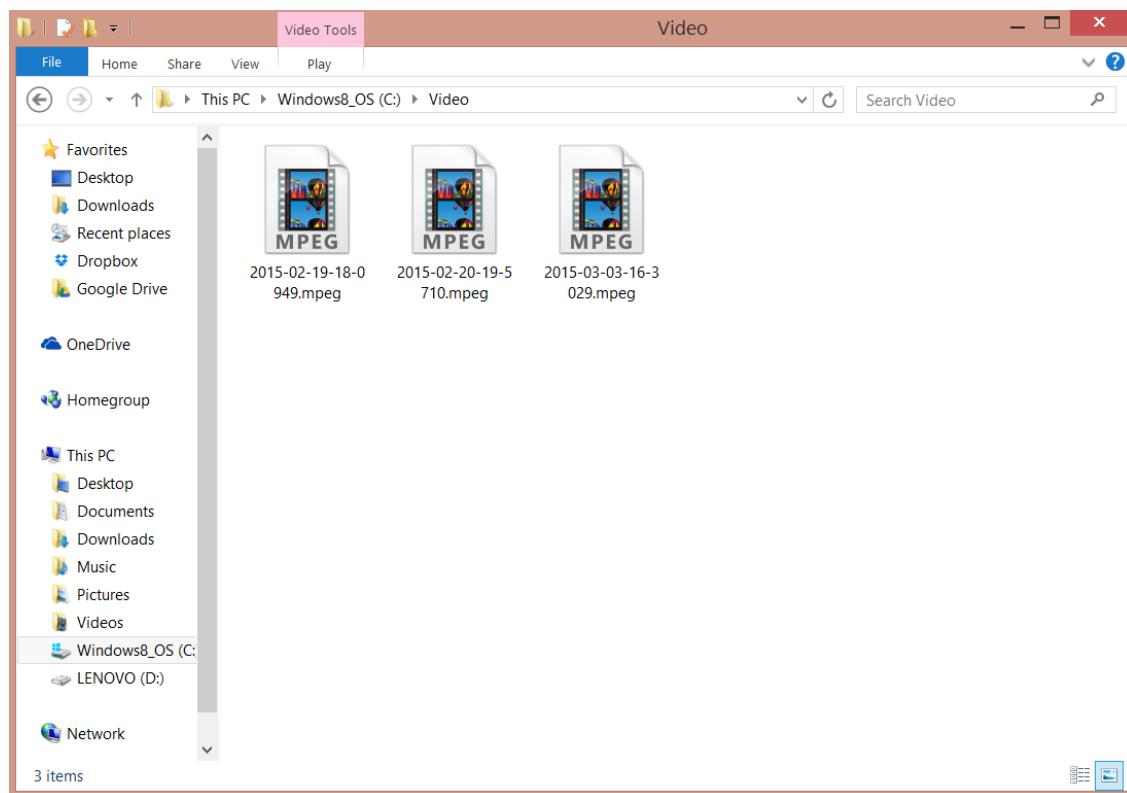


Figure 10: Video Stored in the Folder

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.

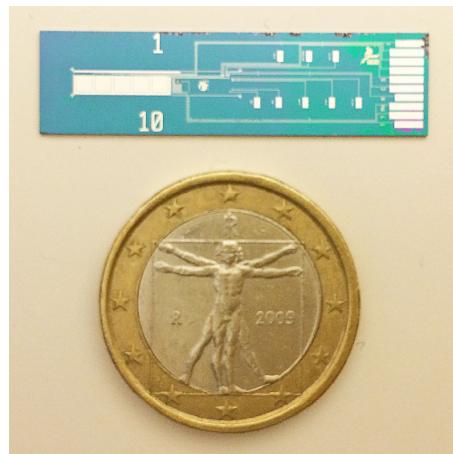


Figure 11: Multisensor - Size Comparison

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.

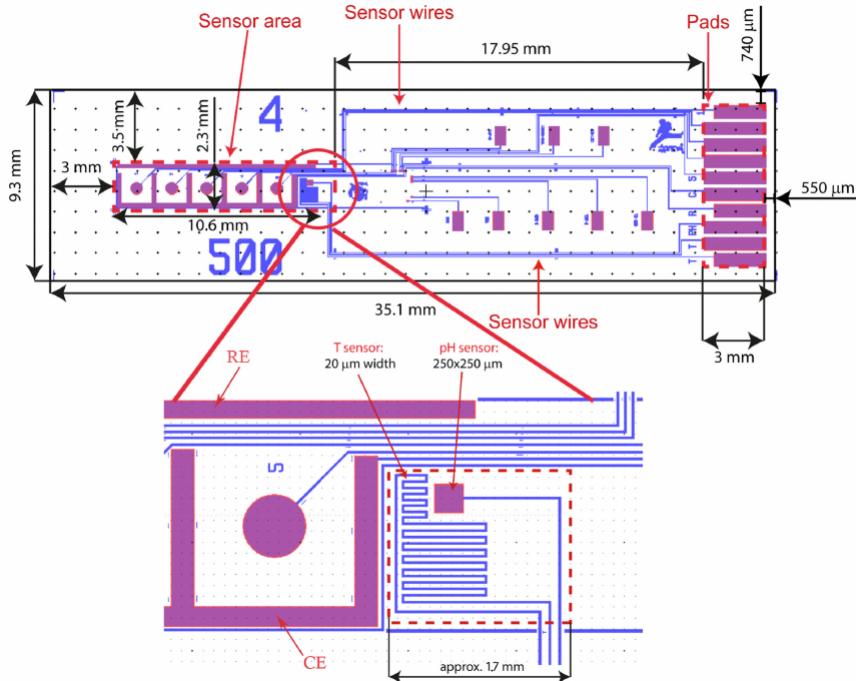


Figure 12: Multisensor - Scheme

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 $\delta = 1.49$.

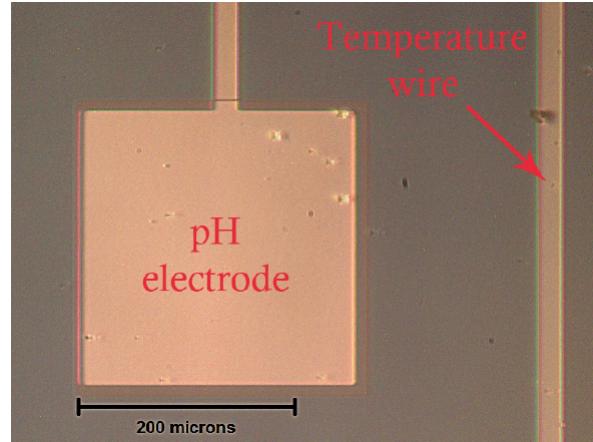


Figure 13: PH and Temperature Sensors from an Optical Image

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).

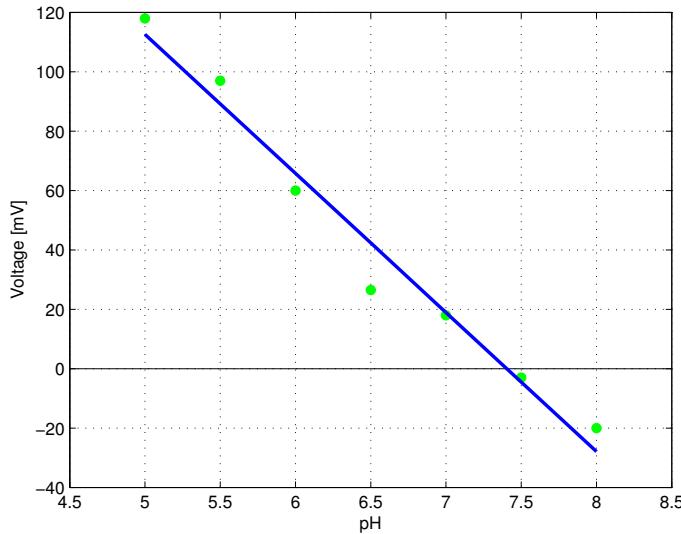


Figure 14: Typical pH-sensor transfer function

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.

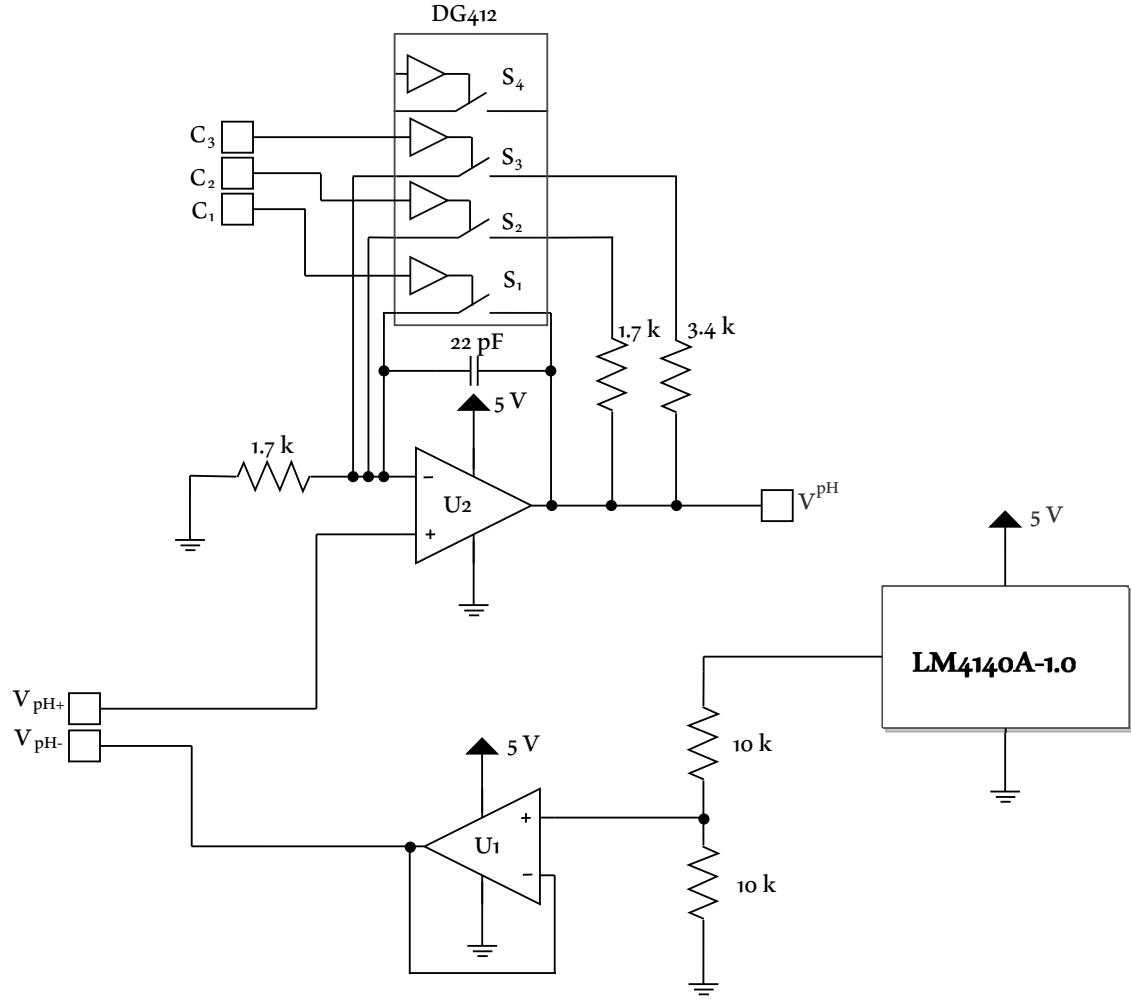


Figure 15: Conditioning circuit for pH sensor

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 ($3 \pm 17 fA$).

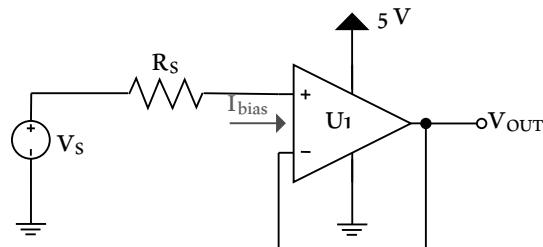


Figure 16: Error caused by Amplifier's Input Bias Current

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 $26 \mu\text{V}$.

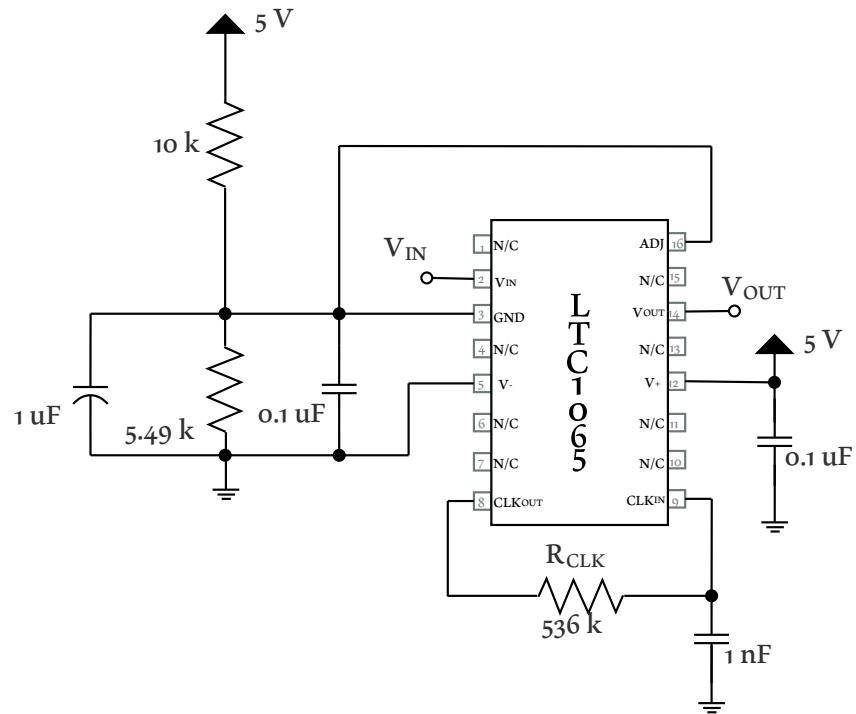
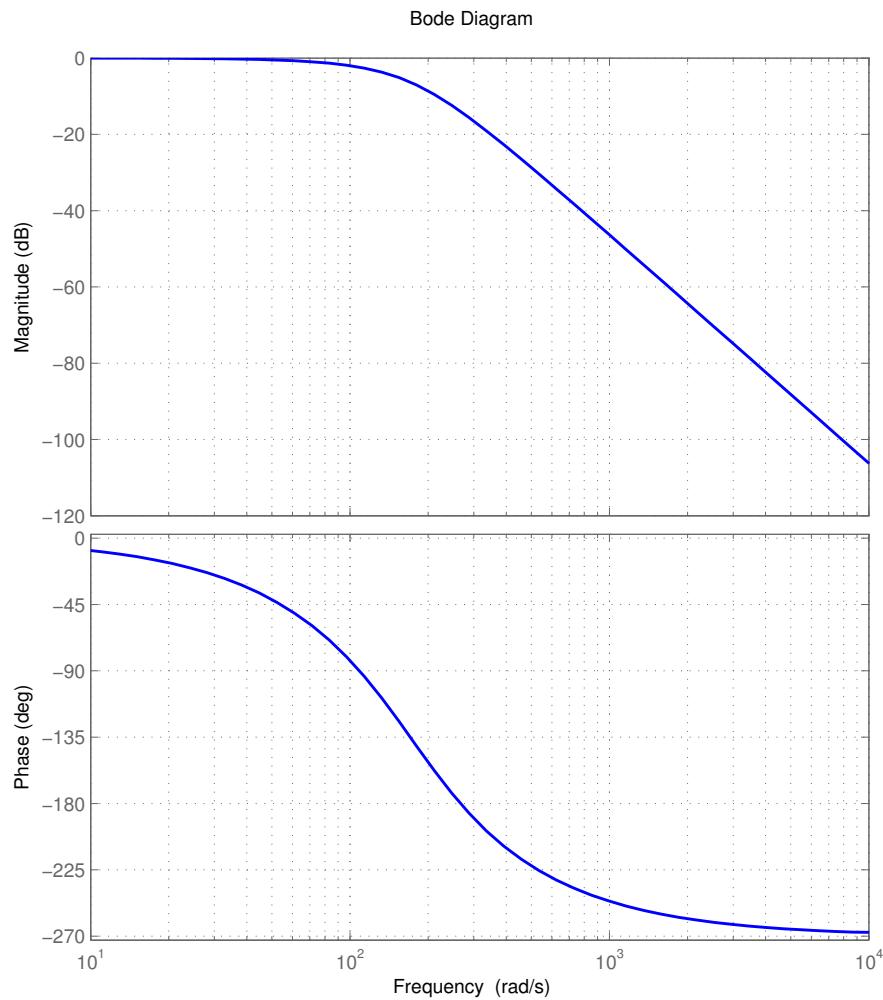
Since the environment in which the circuit is going to be used is a laboratory, so a really noisy place, the conditioning circuit for the sensor has to involve the design of a **low-pass filter** in order to reject the noise.

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.

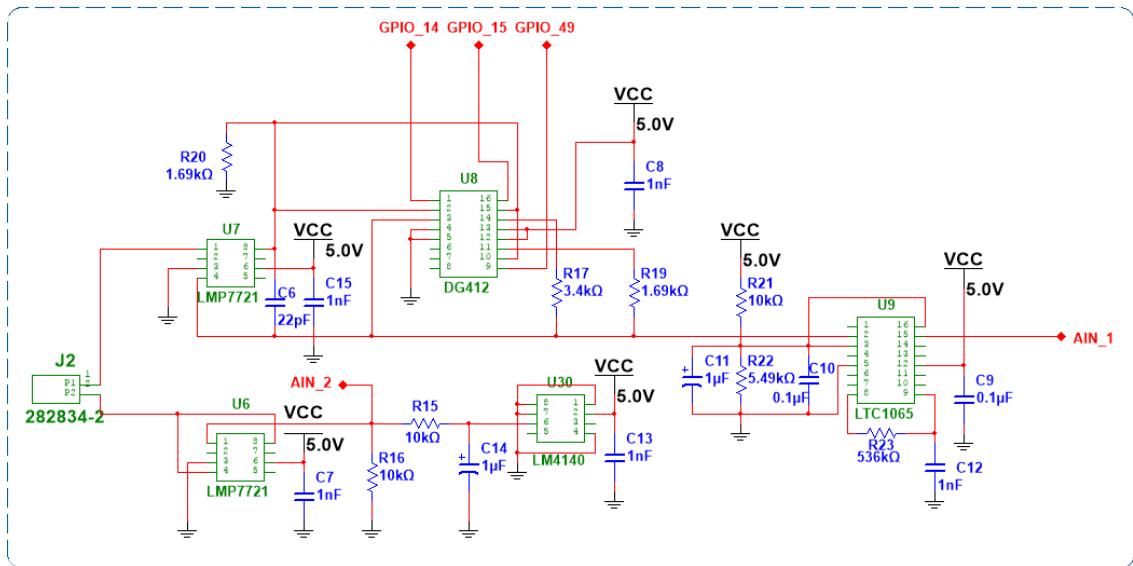


Figure 19: Acquisition Path for pH Sensor

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:

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

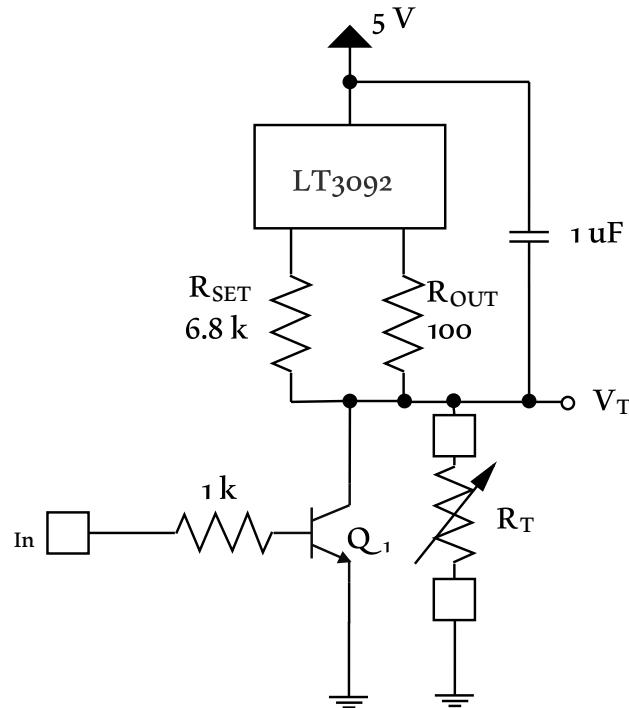


Figure 20: Conditioning Circuit for Temperature Sensor

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.

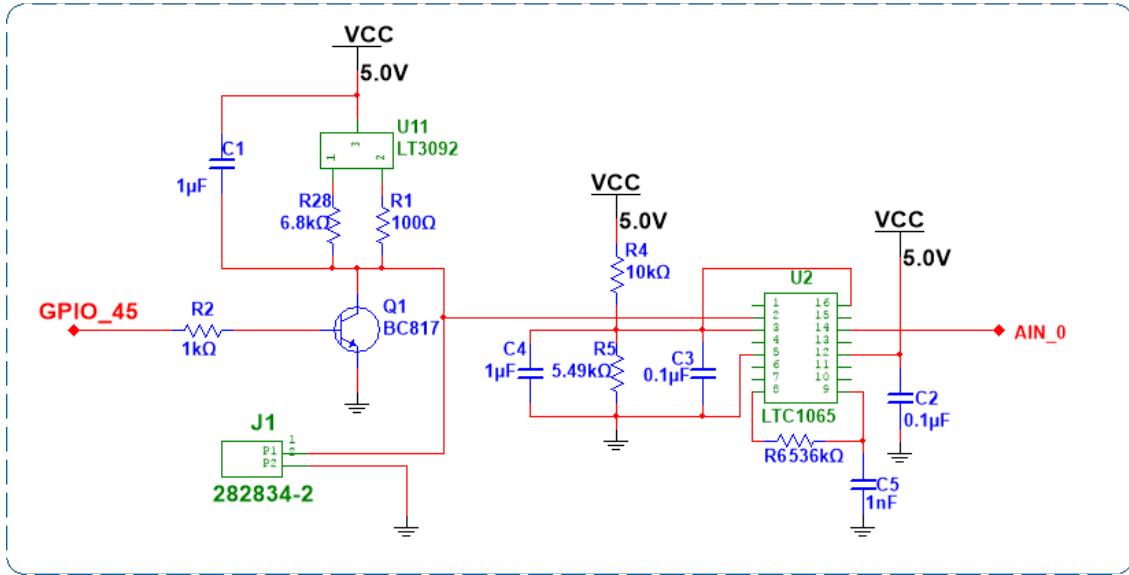


Figure 21: Acquisition Path for pH Sensor

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

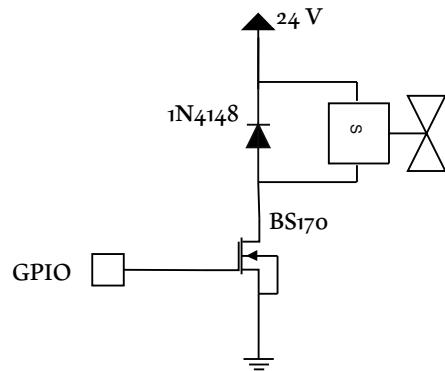


Figure 23: Driver for electrovalves

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.

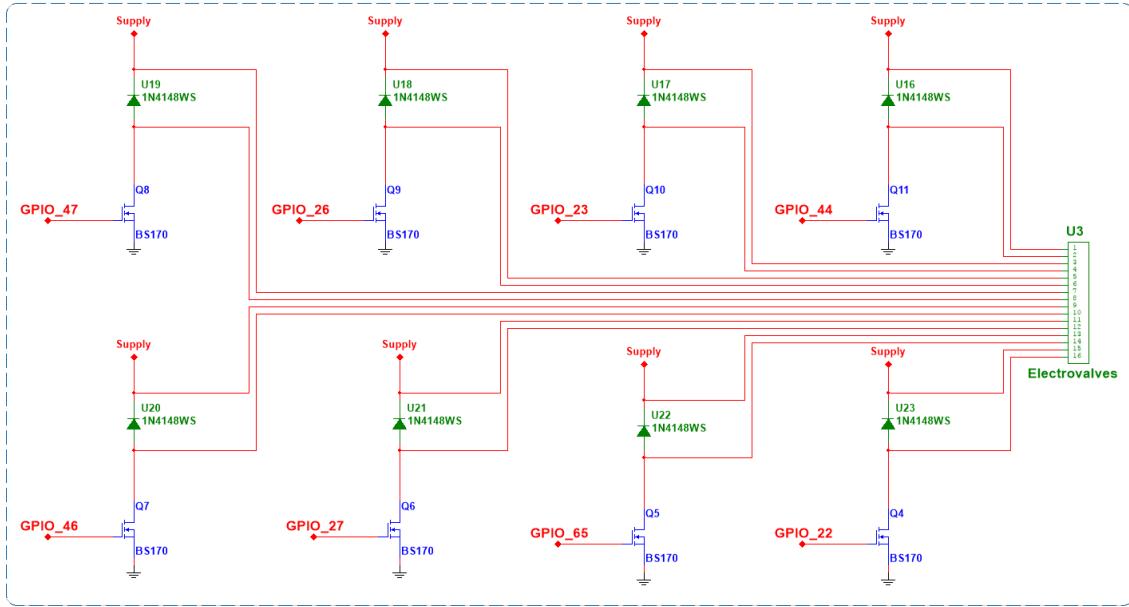


Figure 24: Electrovalves Driver Circuit

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.

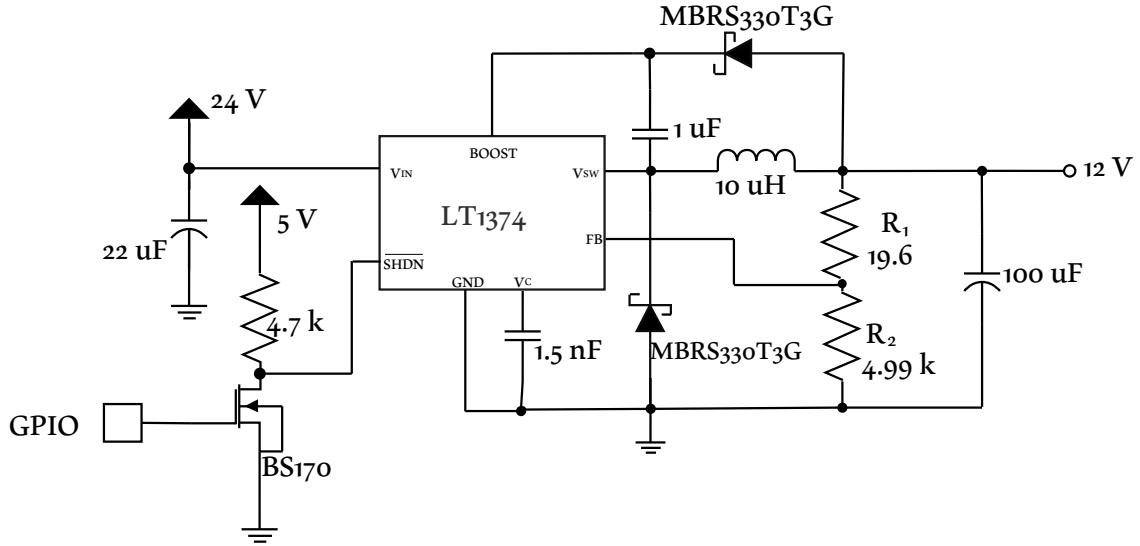


Figure 25: DC-DC circuit

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 these two voltage supplies is needed. The circuit shown in (Fig.26) has been used for this purpose.

It allows the switching through 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.

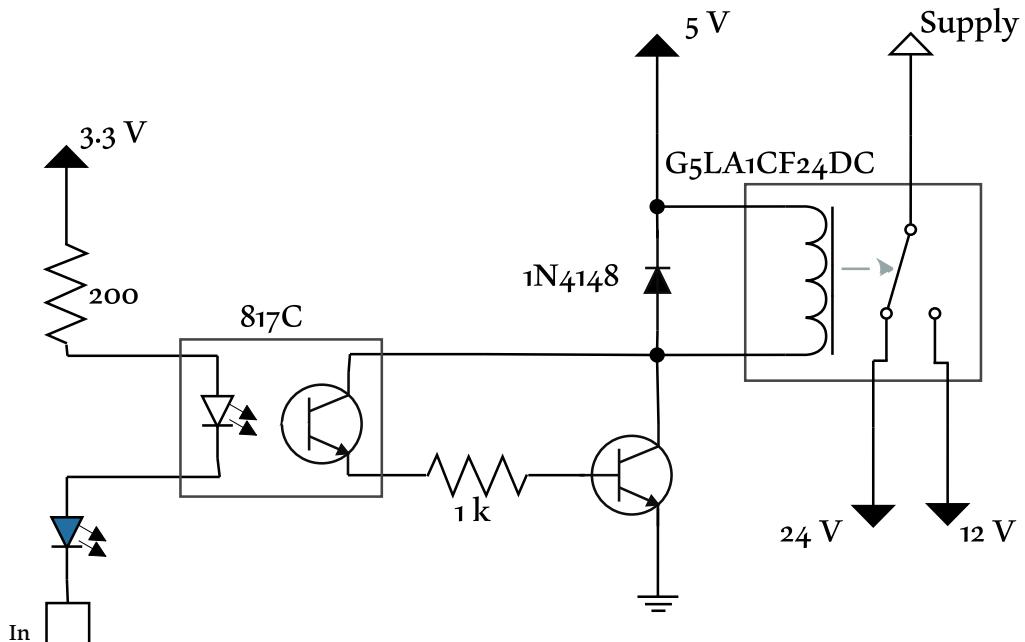


Figure 26: Relay circuit

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, so *Supply* is tied to 24 V. On the other hand, when *In* is denied relay is excited and *Supply* is tied to 12 V.

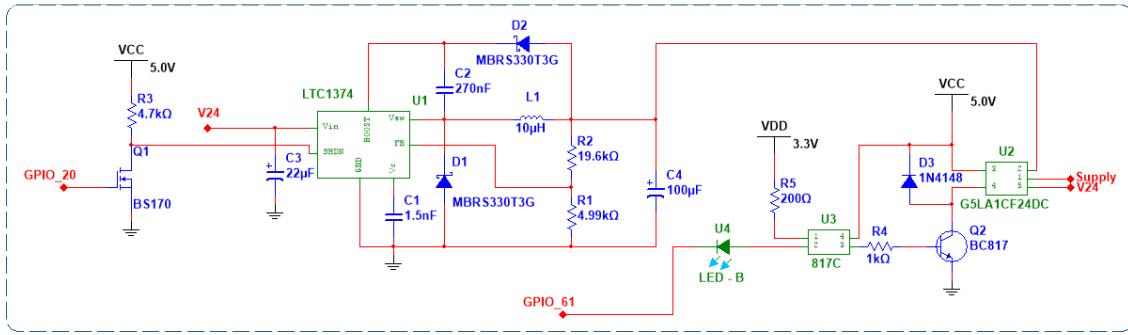


Figure 27: Electrovalves Supply Circuit

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

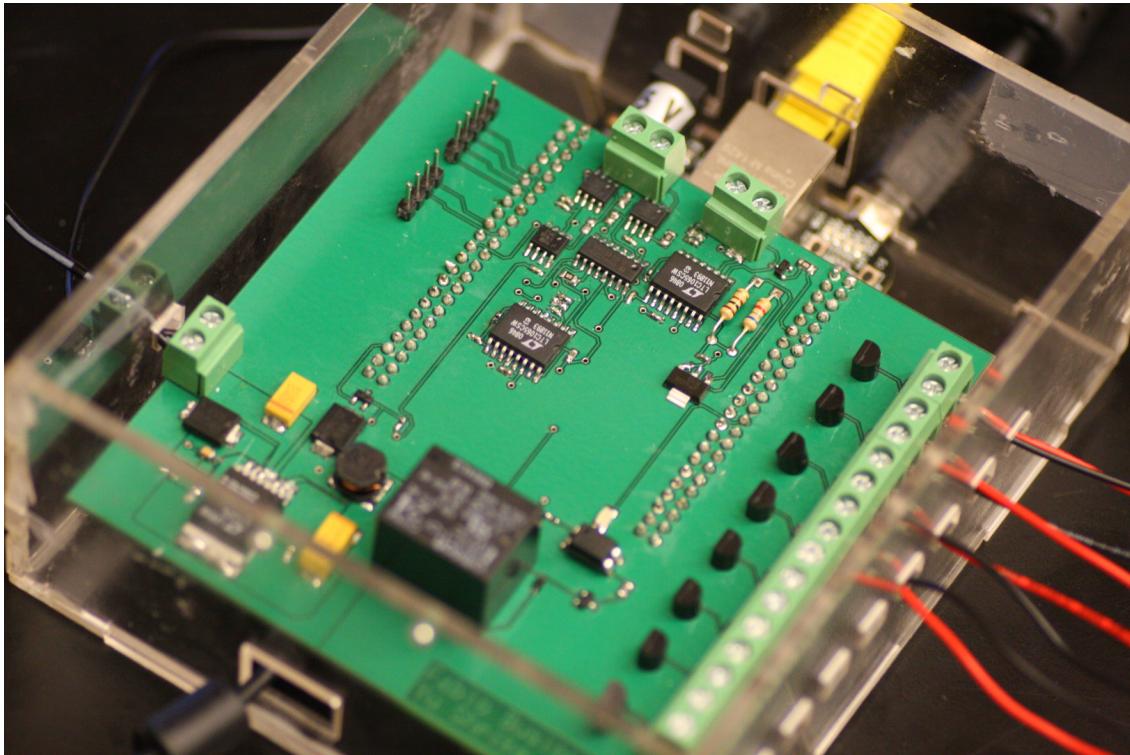


Figure 28: Final system mounted on a PCB

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.

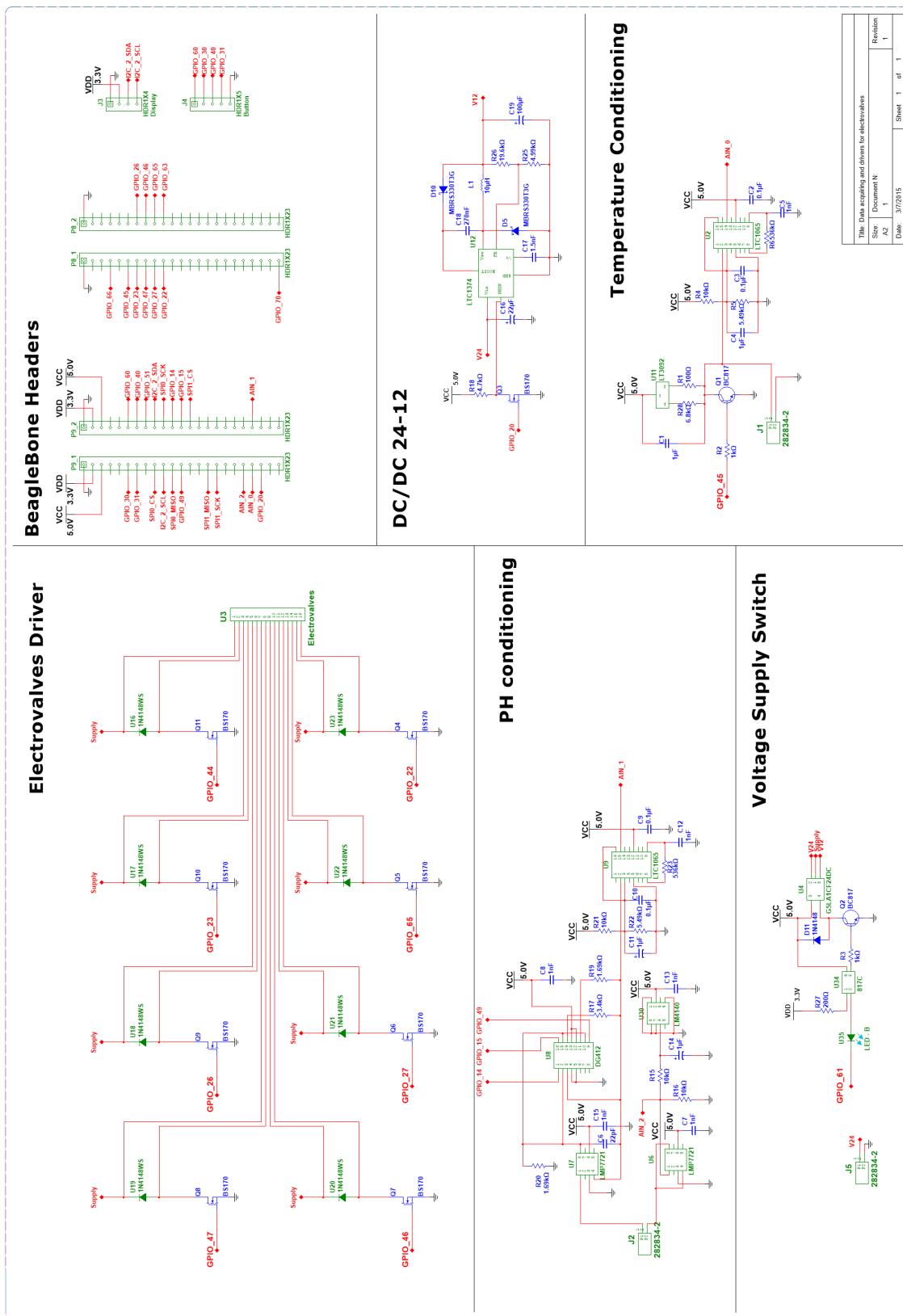


Figure 29: Schematic of Complete Circuit

Title: Data acquisition and driver for electrovalves	
A2	Document N
1	Sheet 1 of 1
Date: 3/7/2015	Revision
	1

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**.

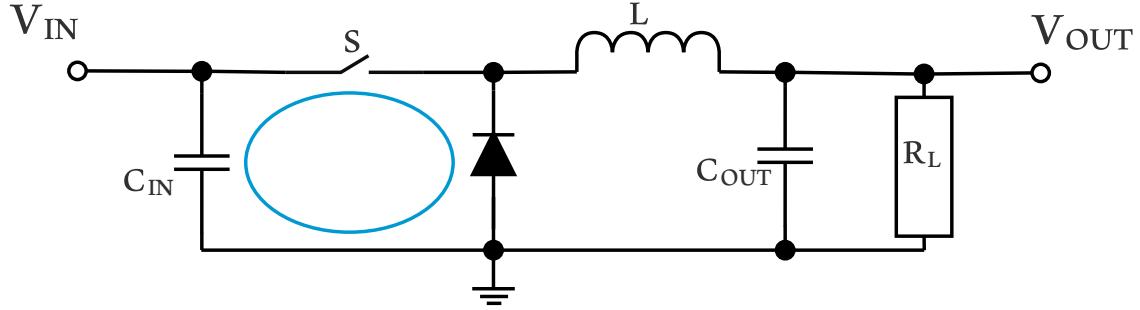


Figure 30: DC-DC High Speed Switching Path

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).

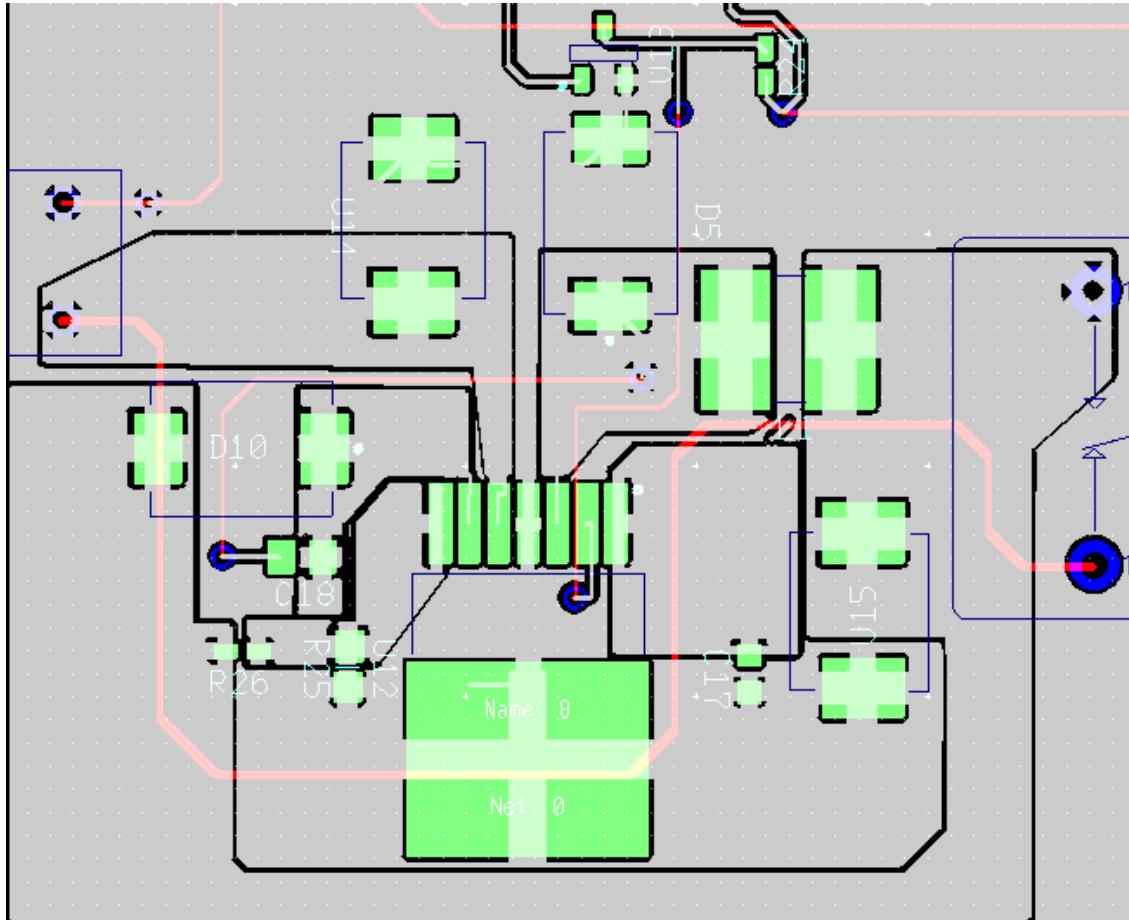


Figure 31: PCB Layout of DC-DC

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.

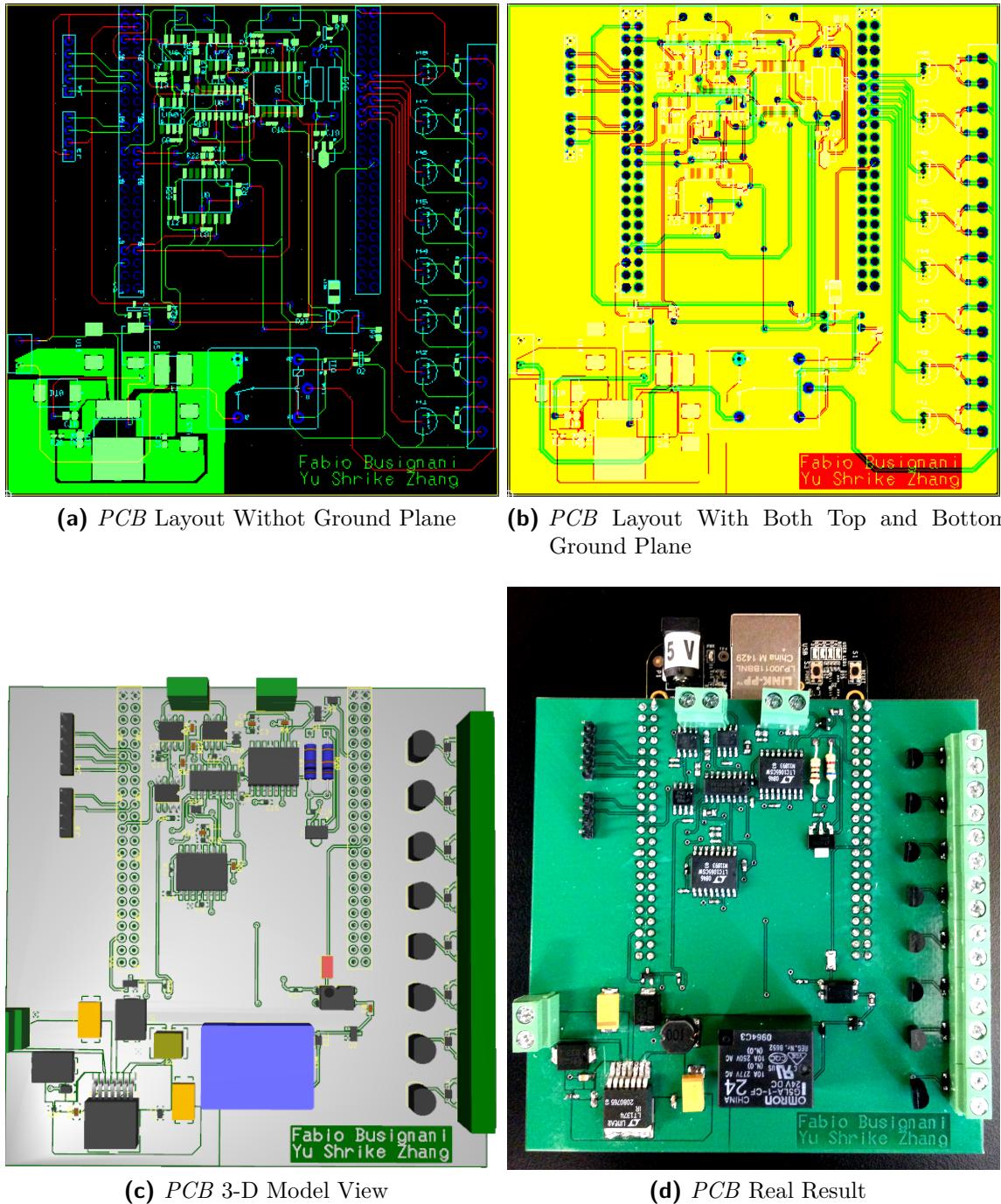


Figure 32: PCB of the System

Part II

FIRMWARE

3

INTRODUCTION

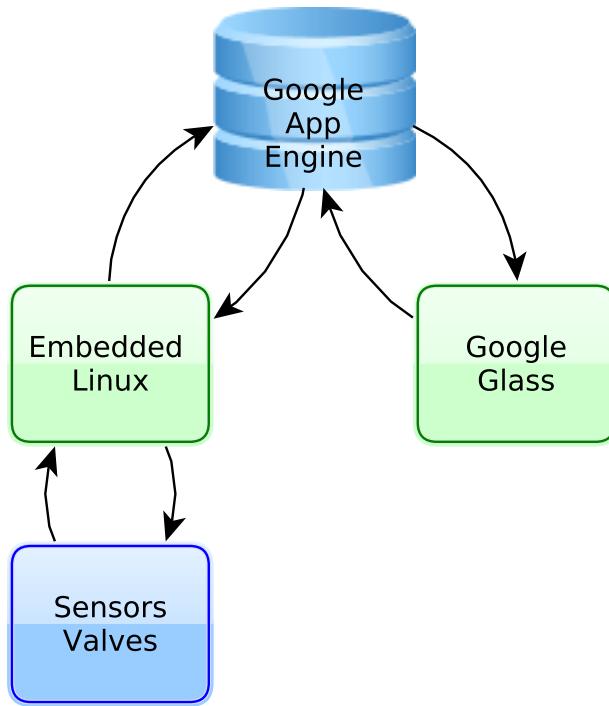


Figure 33: High Level System's Block Diagram

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..

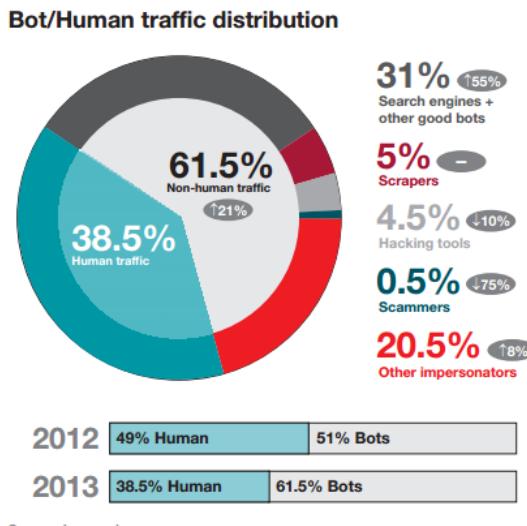


Figure 34: Distribution of Internet of Things

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].

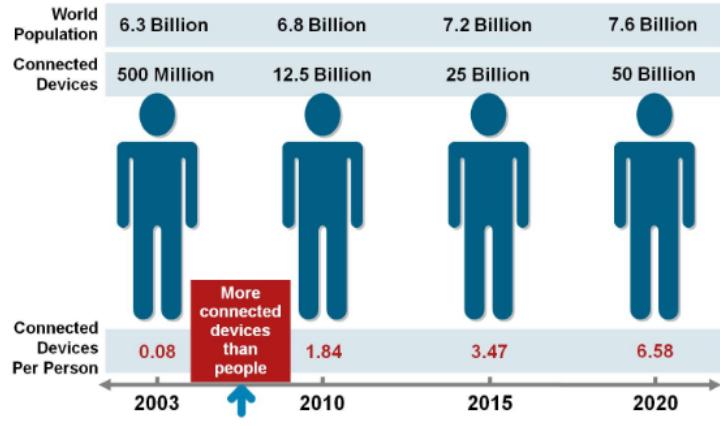


Figure 35: The *IoT* Evolution Prediction

3.2 THE BEAGLEBONE BLACK

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



Figure 36: The *Beaglebone Black*

4 | EMBEDDED LINUX INSIDE THE PROJECT

Part III

SOFTWARE

5 | GOOGLE APP ENGINE

6 | MICROSCOPE VIDEO STORING

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.37).

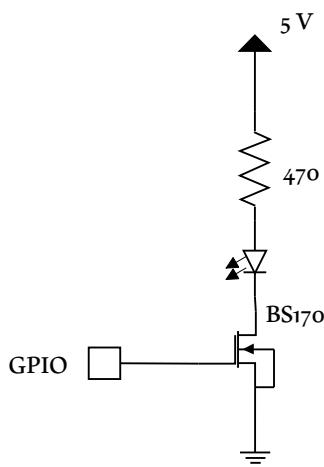


Figure 37: Driver for LED

The (Fig.38) 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](#).

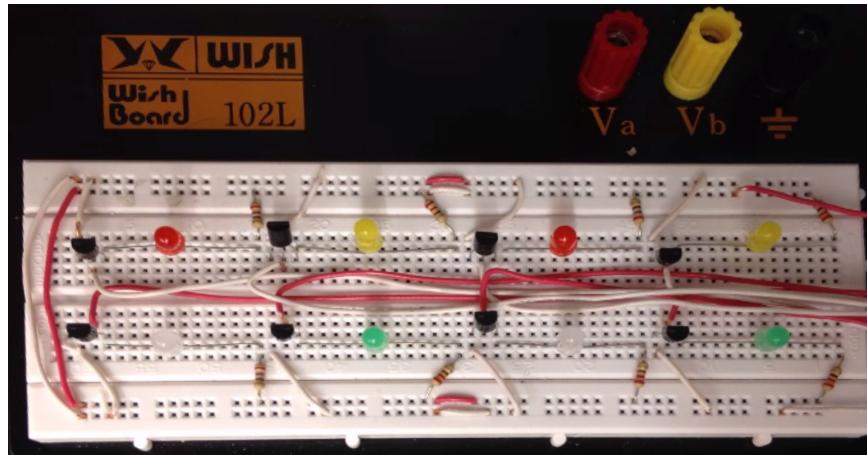


Figure 38: LEDs experiments board

8.2 ELECTROVALVES EXPERIMENTS

8.2.1 Breadboard Phase

The (Fig.39) 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.37), indeed the only difference is given by freewheeling diode, mandatory because of inductive behavior of electrovalve's solenoid.

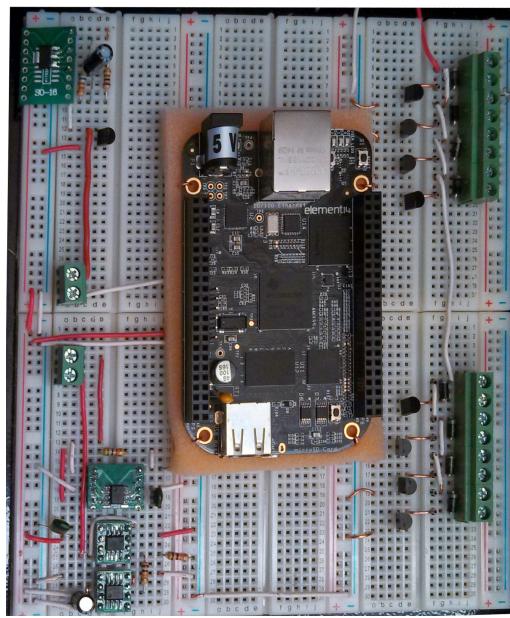


Figure 39: Circuit mounted on breadboard top view

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

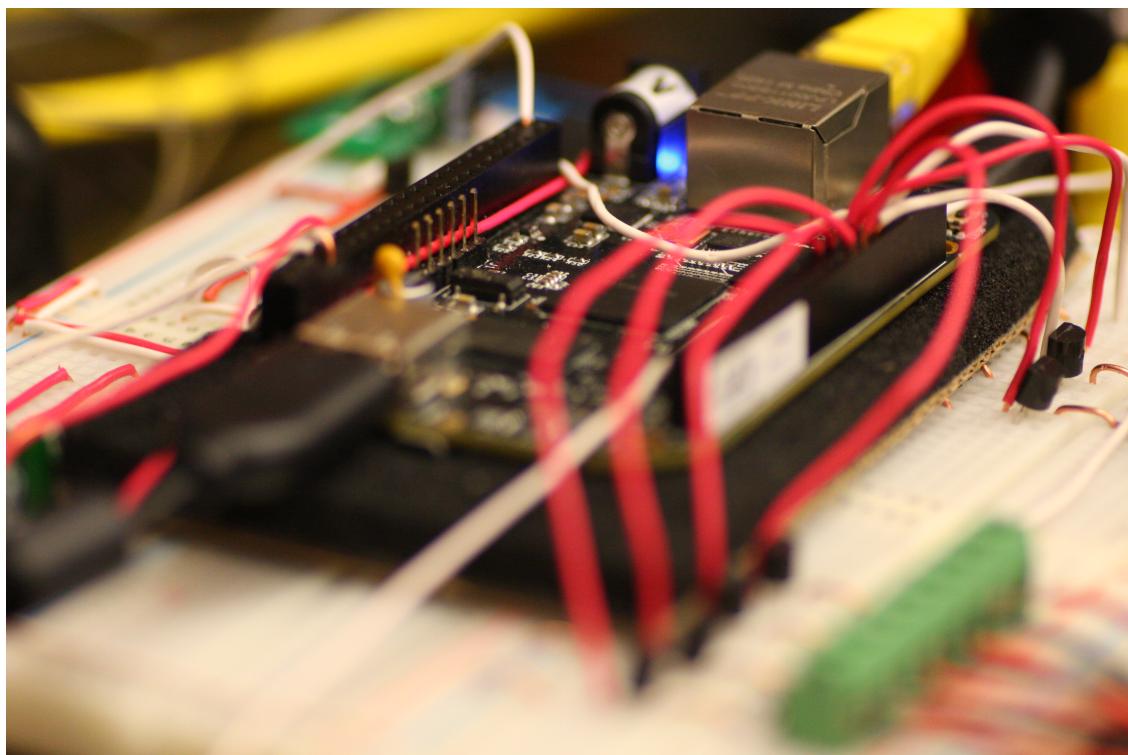


Figure 40: Circuit mounted on breadboard side view

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: 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>
7
8 #define NUMBER_ELECTROVALVES 8
9
10 using namespace exploringBB;
11 using namespace std;
12
13 std::string const ELECTROVALVES_LINK = "http://planar-contact-601.appspot.com/show/";
14 std::string const TRUE_VALUE = " True ";
15 std::string const FALSE_VALUE = " False ";
16
17 static size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
18 {
19     ((std::string*)userp)->append((char*)contents, size * nmemb);
20     return size * nmemb;
21 }
22
23
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
30     CURL *curl;
31     CURLcode res;
32
33     GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];
34
35     //Set output pins
36     EV1.setDirection(OUTPUT);
37     EV2.setDirection(OUTPUT);
```

```

EV3.setDirection(OUTPUT);
EV4.setDirection(OUTPUT);
EV5.setDirection(OUTPUT);
EV6.setDirection(OUTPUT);
EV7.setDirection(OUTPUT);
EV8.setDirection(OUTPUT);

RELAY.setDirection(OUTPUT);

RELAY.setValue(HIGH);

while(1)
{
    //acquire the values of electrovalves status and store them inside output_vector
    int i;
    for(i=0;i<NUMBER_ELECTROVALVES; i++)
    {
        curl = curl_easy_init();
        if(curl)
        {
            std::string readBuffer;
            std::ostringstream ss;
            ss << i+1;
            std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
            curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
            /* example.com is redirected, so we tell libcurl to follow redirection */
            curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
            curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
            /* Perform the request, res will get the return code */
            res = curl_easy_perform(curl);
            /* always cleanup */
            curl_easy_cleanup(curl);

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

//now I'm ready to write the output pins:

```

```

87     EV1.setValue(output_vector[0]);
88     EV2.setValue(output_vector[1]);
89     EV3.setValue(output_vector[2]);
90     EV4.setValue(output_vector[3]);
91     EV5.setValue(output_vector[4]);
92     EV6.setValue(output_vector[5]);
93     EV7.setValue(output_vector[6]);
94     EV8.setValue(output_vector[7]);
95 }
96
97
98     return 0;
99 }
```

Listing A.2: 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 #define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
12 #define GPIO_PATH "/sys/class/gpio/"
13 #define OUTPUT false
14 #define INPUT true
15 #define HIGH true
16 #define LOW false
17
18     float const CURRENT = 0.00068;
19     float const SLOPE_TEMPERATURE = 2;
20     float const OFFSET_TEMPERATURE = 1870;
21     float const SLOPE_PH = -0.070;
22     float const OFFSET_PH = 0.6;
23
24
25
26
27 //void post_data_sensor(int type, float value){
28 //    CURL *curl;
29 //    CURLcode res;
30 //    /* In windows, this will init the winsock stuff */
31 //    curl_global_init(CURL_GLOBAL_ALL);
32 //    /*
33 //    /* get a curl handle */
34 //    curl = curl_easy_init();
```

```

//  if(curl) {
35 //    /* First set the URL that is about to receive our POST. This URL can
36 //       just as well be a https:// URL if that is what should receive the
37 //       data. */
38 //    curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
39 //    /* Now specify the POST data */
40 //    std::ostringstream ss;
41 //    ss << value;
42 //    std::string s;
43 //    if ( type == 0)
44 //      s= POST_DATA_TEMPERATURE +(ss.str());
45 //    else
46 //      s= POST_DATA_PH +(ss.str());
47 //    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
48 //    /* Perform the request, res will get the return code */
49 //    res = curl_easy_perform(curl);
50 //    /* Check for errors */
51 //    if(res != CURLE_OK)
52 //      fprintf(stderr, "curl_easy_perform() failed: %s\n",
53 //              curl_easy_strerror(res));
54 //    /* always cleanup */
55 //    curl_easy_cleanup(curl);
56 //  }
57 //  curl_global_cleanup();
58 //}

59

60

61 class Sensor
62 {
63     int number;

64

65

66 public:
67     float m;
68     float q;
69     Sensor(int x): number(x){}
70     void set_slope(float slope){
71         m = slope;
72     }
73     void set_offset(float offset){
74         q = offset;
75     }
76     float get_voltage_value(){
77         stringstream ss;
78         ss << LDR_PATH << number << "_raw";
79         fstream fs;
80         int adc_value;
81         fs.open(ss.str().c_str(), fstream::in);

```

```

83     fs >> adc_value;
84     fs.close();
85     float cur_voltage = adc_value * (1.80f/4096.0f);
86     float actual_value;
87     return cur_voltage;
88 }
89 float get_sensor_value(){
90     float voltage_value = this->get_voltage_value();
91     float sensor_value = (voltage_value - q)/m;
92     return sensor_value;
93 }
94 };
95
96 class Temperature_Sensor : public Sensor{
97 private:
98
99 public:
100     Temperature_Sensor(int x) : Sensor(x) {}
101     float get_sensor_value(){
102         float voltage_value = this->get_voltage_value();
103         float resistor_value = voltage_value / CURRENT;
104         float sensor_value = (resistor_value - q)/m;
105         return sensor_value;
106     }
107     void post_temperature_data(float data){
108         post_data_sensor(0, data);
109     }
110 };
111
112 class PH_Sensor : public Sensor{
113
114 public:
115     PH_Sensor(int x) : Sensor(x) {}
116     float get_sensor_value(Sensor offset_sensor){
117         //Sensor offset_sensor(2);
118         float voltage_value = this->get_voltage_value();
119         q += offset_sensor.get_voltage_value();
120         float sensor_value = (voltage_value - q)/m;
121         return sensor_value;
122     }
123
124     void post_ph_data(float data){
125         post_data_sensor(1, data);
126     }
127 };
128
129 int main(int argc, char* argv[])
130 {

```

```
133     GPIO temperature_disable(45);
134
135     float slope_temperature;
136
137     float slope_ph;
138
139     float offset_temperature;
140
141     float offset_ph;
142
143
144     if (argc == 5)
145     {
146
147         slope_temperature = atoi(argv[1]);
148
149         offset_temperature = atoi(argv[2]);
150
151         slope_ph = atoi(argv[3]);
152
153         offset_ph = atoi(argv[4]);
154     }
155
156     else
157     {
158
159         slope_temperature = SLOPE_TEMPERATURE;
160
161         offset_temperature = OFFSET_TEMPERATURE;
162
163         slope_ph = SLOPE_PH;
164
165         offset_ph = OFFSET_PH;
166     }
167
168
169     //set the pin GPIO_45 as output
170     temperature_disable.setDirection(OUTPUT);
171
172     //Temporary disable the temperature
173     temperature_disable.setValue(HIGH);
174
175
176     Temperature_Sensor temperature_sensor(2);
177
178     PH_Sensor ph_sensor(1);
179
180     Sensor offset_sensor(2);
181
182
183     temperature_sensor.set_slope(slope_temperature);
184
185     temperature_sensor.set_offset(offset_temperature);
186
187     ph_sensor.set_slope(slope_ph);
188
189     ph_sensor.set_offset(offset_ph);
190
191
192     int i= 0;
193
194     for(i=0; i<100; i++)
195     {
196
197         temperature_disable.setValue(LOW);
198
199         float ph = ph_sensor.get_sensor_value(offset_sensor);
200
201         float temperature = temperature_sensor.get_sensor_value();
202
203         temperature_disable.setValue(HIGH);
204
205         cout << "The voltage value is: " << temperature << " V." << endl;
206
207         temperature_sensor.post_temperature_data(temperature);
208
209         ph_sensor.post_ph_data(ph);
210
211         usleep(100000);
212     }
213
214     float temperature = temperature_sensor.get_voltage_value();
215
216     cout << "The voltage value is: " << temperature << " V." << endl;
```

```

181    return 0;
}

```

Listing A.3: recordVideo

```

#!/bin/bash
2
echo "Setting the video format as MJPG"
4 v4l2-ctl --set-fmt-video=width=320,height=240,pixelformat=1 -d 0
v4l2-ctl --set-parm=30
6
8 echo "Recording one minute of Video"
./capture -d /dev/video0 -F -o -c 300 > output.raw
10 avconv -f mjpeg -i output.raw -vcodec copy output.mp4
echo "Compressing Video"
12 ffmpeg -i output.mp4 -acodec mp2 Video.mp4
echo "Uploading the Video to the server"
14 curl -include --form Video.mp4 =@Video.mp4 -A "National Instruments LabVIEW" http://planar-contact-601.appspot.com/add/electrovalve
echo "Updating the flag for video"
16 curl --data "name=video&status=on" http://planar-contact-601.appspot.com/add/electrovalve
curl --data "name=glass&status=on" http://planar-contact-601.appspot.com/add/electrovalve
18 echo "Plotting the beating rate"
./compute_beating
20 echo "Uploading the beating plot"
curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW" http://planar-contact-601.appspot.com/add/electrovalve
22 echo "finish"

```

Listing A.4: compute_beating.cpp

```

2 #include<iostream>
3 #include<fstream>
4 #include<string>
5 #include <curl/curl.h>
6 #include <sys/stat.h>
7 #include <fcntl.h>
8 #include<sstream>
9 #include<opencv2/opencv.hpp> // C++ OpenCV include file
10 using namespace std;
11 using namespace cv; // using the cv namespace too
12
13 int main()
14 {
15     CURL *curl;
16     CURLcode res;
17     struct stat file_info;
18     FILE *file_to_send;
19     file_to_send = fopen("Video.mp4", "rb");

```

```
20     double speed_upload, total_time;
21
22     if(!file_to_send)
23     {
24         return 1;
25     }
26
27     //to get the file size
28     if(fstat(fileno(file_to_send), &file_info) != 0)
29     {
30         return 1;
31     }
32
33     curl = curl_easy_init();
34     if(curl)
35     {
36         curl_easy_setopt(curl, CURLOPT_URL,
37                         "http://planar-contact-601.appspot.com/video/submit");
38         curl_easy_setopt(curl, CURLOPT_UPLOAD, 1L);
39         curl_easy_setopt(curl, CURLOPT_READDATA, file_to_send);
40         curl_easy_setopt(curl,CURLOPT_INFILESIZE_LARGE,
41                         (curl_off_t)file_info.st_size);
42         curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
43         res = curl_easy_perform(curl);
44
45         /* Check for errors */
46         if(res != CURLE_OK) {
47             fprintf(stderr, "curl_easy_perform() failed: %s\n",
48                     curl_easy_strerror(res));
49
50         }
51         else {
52             /* now extract transfer info */
53             curl_easy_getinfo(curl, CURLINFO_SPEED_UPLOAD, &speed_upload);
54             curl_easy_getinfo(curl, CURLINFO_TOTAL_TIME, &total_time);
55
56             fprintf(stderr, "Speed: %.3f bytes/sec during %.3f seconds\n",
57                     speed_upload, total_time);
58         }
59     }
60
61
62     ofstream point_file;
63     point_file.open("video_data.dat");
64     VideoCapture capture;//(0);    // capturing from file
65     capture.open("output.mp4");
66
67
68 }
```

```

70
71     // set any properties in the VideoCapture object
72     capture.set(CV_CAP_PROP_FRAME_WIDTH,320);      // width pixels
73     capture.set(CV_CAP_PROP_FRAME_HEIGHT,240);      // height pixels
74
75     if(!capture.isOpened())
76     {
77         cout << "Capture not open." << endl;
78     }
79
80     Mat frame, firstFrame, diffFrame;
81
82     capture >> frame; //save the first frame
83     firstFrame = frame;
84     Scalar color_information = mean(firstFrame);
85     float initial_point = (color_information[0] +color_information[1]+color_information[2])/3;
86
87     int i;
88     for(i=1; i<capture.get(CV_CAP_PROP_FRAME_COUNT);i++)
89     {
90         capture >> frame;           // capture the image to the frame
91         if(frame.empty())
92         {
93             cout << "Failed to capture an image" << endl;
94             return -1;
95         }
96         absdiff(frame, firstFrame, diffFrame);
97         color_information = mean(frame);
98         float red = color_information[0];
99         float green = color_information[1];
100        float blue = color_information[2];
101
102        float mean_color = (red+green+blue)/3;
103
104        //qui poi chiamo la funzione che mi posta il punto nel db
105        float temp = (i-1);
106        temp *= 0.033;
107
108        point_file << temp << '\t';
109        point_file << mean_color << '\n';
110    }
111
112
113    point_file.close();
114    return 0;
115}

```

Listing A.5: Pins Setting

```
/*
2 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
*
4 * This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License Version 2 as
6 * published by the Free Software Foundation
*
8 * Original from: github.com/jadonk/validation-scripts/blob/master/test-capemgr/
*
10 * Modified by Fabio Busignani fbusigna@mit.edu
*
12 */
13
14 /dts-v1/;
15 /plugin/;
16
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 - Temperture controller - 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                         0xa0 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             };
50         };
51     };
52 }
```

```

    status = "okay";
50     pinctrl-names = "default";
      pinctrl-0 = <&eklab>;
52   };
  };
54 };
}

```

Listing A.6: GPIO.h

```

1 /*
2  * @file GPIO.h
3  * @author Derek Molloy
4  * @version 0.1
5  *
6  * Created on: 29 Apr 2014
7  * Copyright (c) 2014 Derek Molloy (www.derekmolloy.ie)
8  * Made available for the book "Exploring BeagleBone"
9  * See: www.exploringbeaglebone.com
10 * Licensed under the EUPL V.1.1
11 *
12 * This Software is provided to You under the terms of the European
13 * Union Public License (the "EUPL") version 1.1 as published by the
14 * European Union. Any use of this Software, other than as authorized
15 * under this License is strictly prohibited (to the extent such use
16 * is covered by a right of the copyright holder of this Software).
17 *
18 * This Software is provided under the License on an "AS IS" basis and
19 * without warranties of any kind concerning the Software, including
20 * without limitation merchantability, fitness for a particular purpose,
21 * absence of defects or errors, accuracy, and non-infringement of
22 * intellectual property rights other than copyright. This disclaimer
23 * of warranty is an essential part of the License and a condition for
24 * the grant of any rights to this Software.
25 *
26 * For more details, see http://www.derekmolloy.ie/
27 */
28
29 #ifndef GPIO_H_
30 #define GPIO_H_
31 #include<string>
32 #include<fstream>
33 using std::string;
34 using std::ofstream;
35
36 #define GPIO_PATH "/sys/class/gpio/"
37
38 namespace exploringBB {
39

```

```

41     typedef int (*CallbackType)(int);
42
43 /**
44  * @class GPIO
45  * @brief GPIO class for input and output functionality on a single GPIO pin
46 */
47
48 class GPIO {
49
50     public:
51         enum DIRECTION{ INPUT, OUTPUT };
52         enum VALUE{ LOW=0, HIGH=1 };
53         enum EDGE{ NONE, RISING, FALLING, BOTH };
54
55     private:
56         int number;      /**< The GPIO number of the object */
57         int debounceTime; /*< The debounce time in milliseconds */
58         string name;       /**< The name of the GPIO e.g. gpio50 */
59         string path;       /**< The full path to the GPIO e.g. /sys/class/gpio/gpio50/ */
60
61     public:
62         GPIO(int number);
63         virtual int getNumber() { return number; } /*< Returns the GPIO number as an int. */
64
65         // General Input and Output Settings
66         virtual int setDirection(GPIO::DIRECTION);
67         virtual GPIO::DIRECTION getDirection();
68         virtual int setValue(GPIO::VALUE);
69         virtual int toggleOutput();
70         virtual GPIO::VALUE getValue();
71         virtual int setActiveLow(bool isLow=true); //low=1, high=0
72         virtual int setActiveHigh(); //default
73         //software debounce input (ms) - default 0
74         virtual void setDebounceTime(int time) { this->debounceTime = time; }
75
76
77         // Advanced OUTPUT: Faster write by keeping the stream alive (~20X)
78         virtual int streamOpen();
79         virtual int streamWrite(GPIO::VALUE);
80         virtual int streamClose();
81
82
83         virtual int toggleOutput(int time); //threaded invert output every X ms.
84         virtual int toggleOutput(int numberoftimes, int time);
85         virtual void changeToggleTime(int time) { this->togglePeriod = time; }
86         virtual void toggleCancel() { this->threadRunning = false; }
87
88
89         // Advanced INPUT: Detect input edges; threaded and non-threaded
90         virtual int setEdgeType(GPIO::EDGE);
91         virtual GPIO::EDGE getEdgeType();
92         virtual int waitForEdge(); // waits until button is pressed
93         virtual int waitForEdge(CallbackType callback); // threaded with callback
94         virtual void waitForEdgeCancel() { this->threadRunning = false; }

```

```

89     virtual ~GPIO(); //destructor will unexport the pin
91
92 private:
93     //int write(string path, string filename, string value);
94     //int write(string path, string filename, int value);
95     //string read(string path, string filename);
96     int exportGPIO();
97     int unexportGPIO();
98     ofstream stream;
99     pthread_t thread;
100    CallbackType callbackFunction;
101    bool threadRunning;
102    int togglePeriod; //default 100ms
103    int toggleNumber; //default -1 (infinite)
104    friend void* threadedPoll(void *value);
105    friend void* threadedToggle(void *value);
106 };
107
108 void* threadedPoll(void *value);
109 void* threadedToggle(void *value);
110
111 } /* namespace exploringBB */
112
113 #endif /* GPIO_H_ */

```

Listing A.7: GPIO.cpp

```

1 /*
2  * GPIO.cpp  Created on: 29 Apr 2014
3  * Copyright (c) 2014 Derek Molloy (www.derekmolloy.ie)
4  * Made available for the book "Exploring BeagleBone"
5  * If you use this code in your work please cite:
6  *   Derek Molloy, "Exploring BeagleBone: Tools and Techniques for Building
7  *   with Embedded Linux", Wiley, 2014, ISBN:9781118935125.
8  * See: www.exploringbeaglebone.com
9  * Licensed under the EUPL V.1.1
10 *
11 * This Software is provided to You under the terms of the European
12 * Union Public License (the "EUPL") version 1.1 as published by the
13 * European Union. Any use of this Software, other than as authorized
14 * under this License is strictly prohibited (to the extent such use
15 * is covered by a right of the copyright holder of this Software).
16 *
17 * This Software is provided under the License on an "AS IS" basis and
18 * without warranties of any kind concerning the Software, including
19 * without limitation merchantability, fitness for a particular purpose,
20 * absence of defects or errors, accuracy, and non-infringement of
21 * intellectual property rights other than copyright. This disclaimer

```

```
23 * of warranty is an essential part of the License and a condition for
24 * the grant of any rights to this Software.
25 *
26 * For more details, see http://www.derekmolloy.ie/
27 */
28
29 #include "GPIO.h"
30 #include "util.h"
31 #include<iostream>
32 #include<fstream>
33 #include<string>
34 #include<sstream>
35 #include<cstdlib>
36 #include<cstdio>
37 #include<fcntl.h>
38 #include<unistd.h>
39 #include<sys/epoll.h>
40 #include<pthread.h>
41 using namespace std;
42
43 namespace exploringBB {
44 /**
45 * The constructor will set up the states and export the pin.
46 * @param number The GPIO number to be exported
47 */
48 GPIO::GPIO(int number) {
49     this->number = number;
50     this->debounceTime = 0;
51     this->togglePeriod=100;
52     this->toggleNumber=-1; //infinite number
53     this->callbackFunction = NULL;
54     this->thread = NULL;
55     this->threadRunning = false;
56
57     ostringstream s;
58     s << "gpio" << number;
59     this->name = string(s.str());
60     this->path = GPIO_PATH + this->name + "/";
61     this->exportGPIO();
62     // need to give Linux time to set up the sysfs structure
63     usleep(250000); // 250ms delay
64 }
65 /**
66 * Private write method that writes a single string value to a file in the path provided
67 * @param Path The sysfs path of the file to be modified
68 * @param Filename The file to be written to in that path
69 * @param String value The value to be written to the file
70 * @return
71 */
```

```
71  /* int GPIO::write(string path, string filename, string value){  
73    ofstream fs;  
74    fs.open((path + filename).c_str());  
75    if (!fs.is_open()){  
76      perror("GPIO: write failed to open file ");  
77      return -1;  
78    }  
79    fs << value;  
80    fs.close();  
81    return 0;  
82  }*/  
83  
84  /*string GPIO::read(string path, string filename){  
85    ifstream fs;  
86    fs.open((path + filename).c_str());  
87    if (!fs.is_open()){  
88      perror("GPIO: read failed to open file ");  
89    }  
90    string input;  
91    getline(fs, input);  
92    fs.close();  
93    return input;  
94  }*/  
95  
96  /**  
97   * Private write method that writes a single int value to a file in the path provided  
98   * @param Path The sysfs path of the file to be modified  
99   * @param Filename The file to be written to in that path  
100  * @param int value The int value to be written to the file  
101  * @return  
102  */  
103 /*int GPIO::write(string path, string filename, int value){  
104    stringstream s;  
105    s << value;  
106    return write(path,filename,s.str());  
107 }*/  
108  
109 /**  
110  * Private method to export the GPIO  
111  * @return int that describes if the operation fails  
112  */  
113 int GPIO::exportGPIO(){  
114    return write(GPIO_PATH, "export", this->number);  
115 }  
116  
117 int GPIO::unexportGPIO(){  
118    return write(GPIO_PATH, "unexport", this->number);  
119 }
```

```
121 int GPIO::setDirection(GPIO::DIRECTION dir){
122     switch(dir){
123         case INPUT: return write(this->path, "direction", "in");
124             break;
125         case OUTPUT: return write(this->path, "direction", "out");
126             break;
127     }
128     return -1;
129 }
130
131 int GPIO::setValue(GPIO::VALUE value){
132     switch(value){
133         case HIGH: return write(this->path, "value", "1");
134             break;
135         case LOW: return write(this->path, "value", "0");
136             break;
137     }
138     return -1;
139 }
140
141 int GPIO::setEdgeType(GPIO::EDGE value){
142     switch(value){
143         case NONE: return write(this->path, "edge", "none");
144             break;
145         case RISING: return write(this->path, "edge", "rising");
146             break;
147         case FALLING: return write(this->path, "edge", "falling");
148             break;
149         case BOTH: return write(this->path, "edge", "both");
150             break;
151     }
152     return -1;
153 }
154
155 int GPIO::setActiveLow(bool isLow){
156     if(isLow) return write(this->path, "active_low", "1");
157     else return write(this->path, "active_low", "0");
158 }
159
160 int GPIO::setActiveHigh(){
161     return this->setActiveLow(false);
162 }
163
164 GPIO::VALUE GPIO::getValue(){
165     string input = read(this->path, "value");
166     if (input == "0") return LOW;
167     else return HIGH;
168 }
```

```

169     GPIO::DIRECTION GPIO::getDirection(){
171         string input = read(this->path, "direction");
172         if (input == "in") return INPUT;
173         else return OUTPUT;
174     }
175
176     GPIO::EDGE GPIO::getEdgeType(){
177         string input = read(this->path, "edge");
178         if (input == "rising") return RISING;
179         else if (input == "falling") return FALLING;
180         else if (input == "both") return BOTH;
181         else return NONE;
182     }
183
184     int GPIO::streamOpen(){
185         stream.open((path + "value").c_str());
186         return 0;
187     }
188
189     int GPIO::streamWrite(GPIO::VALUE value){
190         stream << value << std::flush;
191         return 0;
192     }
193
194     int GPIO::streamClose(){
195         stream.close();
196         return 0;
197     }
198
199     int GPIO::toggleOutput(){
200         this->setDirection(OUTPUT);
201         if ((bool) this->getValue()) this->setValue(LOW);
202         else this->setValue(HIGH);
203         return 0;
204     }
205
206     int GPIO::toggleOutput(int time){ return this->toggleOutput(-1, time); }
207
208     int GPIO::toggleOutput(int numberOfTimes, int time){
209         this->setDirection(OUTPUT);
210         this->toggleNumber = numberOfTimes;
211         this->togglePeriod = time;
212         this->threadRunning = true;
213         if(pthread_create(&this->thread, NULL, &threadedToggle, static_cast<void*>(this))){
214             perror("GPIO: Failed to create the toggle thread");
215             this->threadRunning = false;
216             return -1;
217         }
218         return 0;
219     }

```

```

// This thread function is a friend function of the class
219 void* threadedToggle(void *value){
220     GPIO *gpio = static_cast<GPIO*>(value);
221     bool isHigh = (bool) gpio->getValue(); //find current value
222     while(gpio->threadRunning){
223         if (isHigh) gpio->setValue(GPIO::HIGH);
224         else gpio->setValue(GPIO::LOW);
225         usleep(gpio->togglePeriod * 500);
226         isHigh=!isHigh;
227         if(gpio->toggleNumber>0) gpio->toggleNumber--;
228         if(gpio->toggleNumber==0) gpio->threadRunning=false;
229     }
230     return 0;
231 }

233 // Blocking Poll - based on the epoll socket code in the epoll man page
234 int GPIO::waitForEdge(){
235     this->setDirection(INPUT); // must be an input pin to poll its value
236     int fd, i, epollfd, count=0;
237     struct epoll_event ev;
238     epollfd = epoll_create(1);
239     if (epollfd == -1) {
240         perror("GPIO: Failed to create epollfd");
241         return -1;
242     }
243     if ((fd = open((this->path + "value").c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
244         perror("GPIO: Failed to open file");
245         return -1;
246     }
247     //ev.events = read operation | edge triggered | urgent data
248     ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
249     ev.data.fd = fd; // attach the file file descriptor
250
251     //Register the file descriptor on the epoll instance, see: man epoll_ctl
252     if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
253         perror("GPIO: Failed to add control interface");
254         return -1;
255     }
256     while(count<=1){ // ignore the first trigger
257         i = epoll_wait(epollfd, &ev, 1, -1);
258         if (i==-1){
259             perror("GPIO: Poll Wait fail");
260             count=5; // terminate loop
261         }
262         else {
263             count++; // count the triggers up
264         }
265     }

```

```

267     close(fd);
268     if (count==5) return -1;
269   return 0;
270 }
271
// This thread function is a friend function of the class
273 void* threadedPoll(void *value){
274   GPIO *gpio = static_cast<GPIO*>(value);
275   while(gpio->threadRunning){
276     gpio->callbackFunction(gpio->waitForEdge());
277     usleep(gpio->debounceTime * 1000);
278   }
279   return 0;
280 }
281
int GPIO::waitForEdge(CallbackType callback){
282   this->threadRunning = true;
283   this->callbackFunction = callback;
284   // create the thread, pass the reference, address of the function and data
285   if(pthread_create(&this->thread, NULL, &threadedPoll, static_cast<void*>(this))){
286     perror("GPIO: Failed to create the poll thread");
287     this->threadRunning = false;
288     return -1;
289   }
290   return 0;
291 }
292
GPIO::~GPIO() {
293   this->unexportGPIO();
294 }
295
} /* namespace exploringBB */

```

Listing A.8: HTTP.h

```

#ifndef HTTP
#define HTTP
#include<iostream>
#include<fstream>
#include<unistd.h>
#include<string>
#include<sstream>
#include<stdlib.h>
#include<curl/curl.h>
std::string const URL_POST = "http://planar-contact-601.appspot.com/sensor_values";
std::string const POST_DATA_TEMPERATURE = "sensor=temperature&value=";
std::string const POST_DATA_PH = "sensor=pH&value=";
std::string const POST_DATA_BEATING = "sensor=beating&value=";

```

```

16
void post_data_sensor(int type, float value);
18
#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
    /* get a curl handle */
10
    curl = curl_easy_init();
    if(curl) {
12
        /* 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
14
           data. */
        curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
16
        /* Now specify the POST data */
        std::ostringstream ss;
18
        ss << value;
        std::string s;
20
        if (type == 0)
            s = POST_DATA_TEMPERATURE +(ss.str());
22
        else if (type == 1)
            s = POST_DATA_PH +(ss.str());
24
        else
            s = POST_DATA_BEATING + (ss.str());
26
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
28
        /* Perform the request, res will get the return code */
        res = curl_easy_perform(curl);
29
        /* Check for errors */
30
        if(res != CURLE_OK)
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
31
                    curl_easy_strerror(res));
32
        /* always cleanup */
33
        curl_easy_cleanup(curl);
34
    }
35
    curl_global_cleanup();
}

```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```

1 // #include <QCoreApplication>
2 // #include < QTimer >
3 // #include "downloader.h"
4 // #include <QDebug>
5
6 // int main(int argc, char *argv[])
7 //{
8 //     QCoreApplication a(argc, argv);
9
10 //     QDateTime data;
11 //     data = QDateTime::currentDateTime();
12 //     QString name = "./"+data.toString("yyyy-MM-dd-HH:mm:ss")+".mp4";
13 //     Downloader d(0, name);
14 //     d.doDownload();
15 //     for(;;)
16 //     {
17 //         qDebug() << "Ciao";
18 //         if(d.isTime())
19 //         {
20 //             d.sendFlag();
21 //             d.doDownload();
22 //         }
23 //     }
24 //     return a.exec();
25 //}
26
27 #include <QCoreApplication>
28 #include "downloader.h"
29 #include "mytimer.h"
30 #include <QDebug>
31 #include <windows.h> // for Sleep
32
33 int main(int argc, char *argv[])
34 {
35     QCoreApplication a(argc, argv);
36
37     MyTimer t;
38
39     //bool dummy = d.isTime();
40 //     while(1)
41 //     {
42 //         d.checkFlag();
43 //

```

```
45
//           if( d.getStatus() )
47 //           {
48 //               d.resetFlag();
49 //               d.doDownload();
50 //           }
51 //           Sleep(20000);
52 //       }
53
54
55
56
57     return a.exec();
58 }
```

Listing A.11: VideoStoring.pro

Listing A.12: mytimer.h

```
1 #ifndef MYTIMER_H  
#define MYTIMER_H
```

```

3 #include <QObject>
5 #include <QtCore>
7 #include <downloader.h>
9
11 class MyTimer : public QObject
12 {
13     Q_OBJECT
14 public:
15     MyTimer();
16     ~MyTimer();
17     QTimer *timer;
18
19 public slots:
20     void timerSlot();
21 private:
22     Downloader d;
23 };
24 #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);
10 }
11
12 MyTimer::~MyTimer()
13 {
14 }
15
16
17
18
19
20 void MyTimer::timerSlot(){
21     d.checkFlag();
22     if( d.getStatus())
23     {
24         d.resetFlag();
25     }
26 }

```

```

        d.doDownload();
27    }
29}

```

Listing A.14: downloader.h

```

#ifndef DOWNLOADER_H
#define DOWNLOADER_H

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

14 class Downloader : public QObject
{
15     Q_OBJECT
16 public:
17     explicit Downloader(QObject *parent = 0);

20     void doDownload();
21     void checkFlag();
22     bool getStatus();
23     void resetFlag();

24 signals:
25
26 public slots:
27     void replyFinished (QNetworkReply *reply);
28     void checkStatus(QNetworkReply *replay);

30 private:
31     QNetworkAccessManager *manager;
32     QNetworkAccessManager *manager2;
33     QDateTime data;
34     QString name;
35     bool flag_ready;
36     bool status;

38
39
40};

#endif // DOWNLOADER_H

```

Listing A.15: downloader.cpp

```

1 #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(QUrl("http://planar-contact-601.appspot.com/video/view")));
17 }
18
19 void Downloader::replyFinished (QNetworkReply *reply)
20 {
21     if(reply->error())
22     {
23         qDebug() << "ERROR!";
24         qDebug() << reply->errorString();
25     }
26     else
27     {
28         qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
29         qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).toDateTime().toString();
30         qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).toULongLong();
31         qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt();
32         qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).toString();
33
34         data = QDateTime::currentDateTime();
35         name = "C:/Video/" + data.toString("yyyy-MM-dd-HH-mmss") + ".mpeg";
36
37         QFile *file = new QFile(name);
38         if(file->open(QFile::Append))
39         {
40             file->write(reply->readAll());
41             file->flush();
42             file->close();
43         }
44         delete file;
45     }
46
47     reply->deleteLater();
48 }
```

```
50
51     void Downloader::checkStatus(QNetworkReply *replay)
52 {
53     QString result(replay->readAll());
54     int compare = QString::compare(result, FLAG_ON);
55     if(compare == 0)
56     {
57         status = true;
58     }
59     else
60     {
61         status = false;
62     }
63     qDebug() << status;
64     flag_ready = true;
65 }
66
67 void Downloader::checkFlag()
68 {
69     manager = new QNetworkAccessManager(this);
70     QNetworkReply *reply = manager->get(QNetworkRequest(QUrl("http://planar-contact-601.appspot.com/
71     QEEventLoop loop;
72     connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));
73
74     loop.exec();
75     qDebug() << "I'm looking for a new video";
76     QString result(reply->readAll());
77     int compare = QString::compare(result, FLAG_ON);
78     if(compare == 0)
79     {
80         status = true;
81         qDebug() << "Found it";
82     }
83     else
84     {
85         status = false;
86         qDebug() << "No new video available";
87     }
88 }
89
90 bool Downloader::getStatus()
91 {
92     return status;
93 }
94
95 void Downloader::resetFlag()
96 {
```

```

98     status = false;
99     manager2 = new QNetworkAccessManager(this);
100
101     QUrl flag_url = QUrl("http://planar-contact-601.appspot.com/add/electrovalve");
102     QByteArray postData;
103     postData.append("name=video&status=off");
104     manager2->post(QNetworkRequest(flag_url), postData);
105 }
```

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```

1 # the application id inside app.yaml has to match with the ID in google app engine

3 from flask import Flask

5 app = Flask(__name__)

7 from flask import request
8 from flask import make_response
9
10 from models import MESSAGES
11 from google.appengine.ext import ndb
12 from google.appengine.api import memcache
13 import logging
14 import datetime
15 import cloudstorage
16 import json
17
18 key_list = []
19
20 NUMBER_OF_ELECTROVALVES = 8
21 SECONDS_BETWEEN_UPDATES = 60
22 SENSOR_TIMEOUT_IN_SECONDS = 300
23 ELECTROVALVES_TIME_OUT = 1000
24 BUCKET_NAME = "/planar-contact-601.appspot.com/"
25 ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
26 UPDATE_FLAG = 'database_updated_recently'
27
28
29 class DataPoint(ndb.Model):
30     value = ndb.FloatProperty()
31     date = ndb.DateTimeProperty(auto_now_add=True)
32
33     @classmethod
```

```
def points_for_sensor(cls, sensor_key):
    return cls.query(ancestor=sensor_key).order(-cls.date)

@classmethod
def oldest_points(cls):
    return cls.query().order(+cls.date)

class Sensor(ndb.Model):
    @classmethod
    def sensor_list(cls):
        return cls.query()

def update():
    # Check to see if we have updated recently
    update_flag = memcache.get(UPDATE_FLAG)
    if update_flag is not None:
        return

    memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
    # Delete old data points
    now = datetime.datetime.now()
    count = 0
    for point in DataPoint.oldest_points().iter():
        delta = (now - point.date).total_seconds()
        if delta > 3600:
            point.key.delete()
            count += 1
        else:
            break
    logging.info("Deleted {} old data points".format(count))
    # Loop through all of the sensors
    sensor_names = []
    for sensor in Sensor.sensor_list().iter():
        # Store current values of each sensor
        sensor_value = memcache.get(sensor.key.id() + "_value")
        if sensor_value is not None:
            point = DataPoint(parent=sensor.key, value=sensor_value)
            point.put()
        # Delete sensors that don't have any data
        curr_points = DataPoint.points_for_sensor(sensor.key)
        if curr_points.count() == 0 and sensor_value is None:
            logging.info("Deleting sensor {}".format(sensor.key.id()))
            sensor.key.delete()
        else:
            sensor_names.append(sensor.key.id())
    # Store a list of sensor names in memcache
    memcache.set("sensor_names", ",".join(sensor_names))
```

```

83     @app.route('/', methods=['POST'])
85     @app.route('/sensor_values', methods=['GET', 'POST'])
86     def process_values():
87         if request.method == 'GET':
88             sensor_names = memcache.get('sensor_names')
89             if sensor_names is None:
90                 sensor_names = ",".join([sensor.key.id() for sensor in Sensor.sensor_list()])
91                 memcache.set('sensor_names', sensor_names)
92             sensor_names = sensor_names.split(",")
93             sensor_values = {}
94             for sensor_name in sensor_names:
95                 sensor_value = memcache.get(sensor_name + '_value')
96                 if sensor_value is not None:
97                     sensor_values[sensor_name] = sensor_value
98             return json.dumps(sensor_values)
99         elif request.method == 'POST':
100             update()
101             logging.info(request.values)
102             sensor_name = request.form.get('sensor')
103             sensor_value = float(request.form.get('value'))
104             sensor_names = memcache.get('sensor_names')
105             if sensor_names is not None and sensor_name not in sensor_names:
106                 sensor = Sensor()
107                 sensor.key = ndb.Key('Sensor', sensor_name)
108                 sensor.put()
109                 sensor_names = sensor_name if sensor_names == "" else sensor_names + "," + sensor_name
110                 memcache.set('sensor_names', sensor_names)
111             memcache.set(sensor_name + '_value', sensor_value, SENSOR_TIMEOUT_IN_SECONDS)
112             return "Success\n"
113
114
115     @app.route('/sensor_names', methods=['GET'])
116     def print_names():
117         return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])
118
119
120     @app.route('/graphing_data', methods=['GET'])
121     def graphing_data():
122         last_timestamp = request.args.get('last_timestamp')
123         sensor_name = request.args.get('sensor')
124         if last_timestamp is None:
125             last_timestamp = 0
126         else:
127             last_timestamp = float(last_timestamp)
128         points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
129         epoch = datetime.datetime(1970, 1, 1)
130         res = []
131         for point in points.iter():

```

```
    timestamp = (point.date - epoch).total_seconds()
133    if timestamp > last_timestamp:
134        _res = {"timestamp": timestamp, "value": point.value}
135    res.append(_res)
136    else:
137        break
138    res.reverse()
139    return json.dumps(res)

141
142 @app.route('/clear', methods=['GET'])
143 def clear_data():
144     points = DataPoint.query()
145     for point in points:
146         point.key.delete()
147     for sensor in Sensor.sensor_list():
148         sensor.key.delete()
149     return "Success"

151
152 @app.route('/picture/submit', methods=['GET', 'POST'])
153 def set_picture():
154     if request.method == 'POST':
155
156         _file = request.files['Image.jpg']
157
158         if _file:
159             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image." + _file.filename.rsplit(
160                                         '.', 1)[0],
161                                         mode='w', content_type="image/jpeg")
162             _file.save(cloud_file)
163             cloud_file.close()
164             return "Success"
165
166         else:
167             return ''
168
169         <!doctype html>
170         <title>Upload microscope file V.1</title>
171         <hi>Upload microscope file</hi>
172         <form method="POST">
173             action=""
174             role="form"
175             enctype="multipart/form-data">
176
177             <p><input type=file name=Image.jpg>
178                 <input type=submit value=Upload>
179             </form>
180             ''
181
182
183 @app.route('/picture/view', methods=['GET'])
```

```

181 def view_picture():
182     for _file in cloudstorage.listbucket(BUCKET_NAME):
183         if "microscope_image" in _file.filename:
184             _file = cloudstorage.stat(_file.filename)
185             logging.info(_file.filename)
186             logging.info(_file.content_type)
187             cloud_file = cloudstorage.open(_file.filename, mode='r')
188             response = make_response(cloud_file.read())
189             cloud_file.close()
190             response.mimetype = _file.content_type
191             return response
192
193     return "No file found"

194
195 @app.route('/video/submit', methods=['GET', 'POST', 'PUT'])
196 def set_video():
197     if request.method == 'GET':
198         return ''
199         <!doctype html>
200         <title>Upload microscope video</title>
201         <h1>Upload microscope video</h1>
202         <form method="POST"
203 action=""
204 role="form"
205 enctype="multipart/form-data">
206             <p><input type=file name=Video.mp4>
207                 <input type=submit value=Upload>
208             </form>
209             ''
210
211     else:
212         _file = request.files['Video.mp4']
213         if _file:
214             cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_video." + _file.filename.rsplit(
215                                         mode='w', content_type="video/mpeg"))
216             _file.save(cloud_file)
217             cloud_file.close()
218             return "Success"

219
220
221 @app.route('/video/view', methods=['GET'])
222 def view_video():
223     for _file in cloudstorage.listbucket(BUCKET_NAME):
224         if "microscope_video" in _file.filename:
225             _file = cloudstorage.stat(_file.filename)
226             logging.info(_file.filename)
227             logging.info(_file.content_type)
228             cloud_file = cloudstorage.open(_file.filename, mode='r')
229             response = make_response(cloud_file.read())

```

```
        cloud_file.close()
231    response.mimetype = _file.content_type
232    return response
233
234
235
236    @app.errorhandler(404)
237    def page_not_found(e):
238        """Return a custom 404 error."""
239        return 'Sorry, nothing at this URL.', 404
240
241
242    class Electrovalve(ndb.Model):
243        # name = ndb.StringProperty()
244        # status = ndb.BooleanProperty() # if the status is true, the Electrovalve is on
245        # date = ndb.DateTimeProperty(auto_now_add=True)
246        #
247        # def __init__(self, name, status, **kwds):
248        #     super(Electrovalve, self).__init__(**kwds)
249        #     self.name = name
250        #     self.status = status
251        #
252        # @classmethod
253        # def get_status(cls):
254        #     return cls.status
255        #
256        # @classmethod
257        # def get_ev_by_name(cls, ev_name):
258        #     return cls.query(ancestor=ev_name).fetch
259
260
261    @classmethod
262    def ev_list(cls):
263        return cls.query()
264
265
266    # @app.route('/add/<key>/<message>')
267    # def update_message(key, message):
268    #     if key and message:
269    #         if message == 'on' or message == 'On' or message == 'ON':
270    #             MESSAGES[key] = True
271    #
272    #     else:
273    #         MESSAGES[key] = False
274    #
275    #         temp = Electrovalve(name=key, status=message)
276    #         temp.put()
277    #
278    return "%s Updated" % key
```

```

279 @app.route('/electrovalves/<name>', methods=['GET'])
281 def electrovalve(name):
282     ev = memcache.get(name);
283     if ev is None:
284         return '%r' % MESSAGES[name] or "%s not found!" % name
285     else:
286         return '%r' % memcache.get(name)
287     #     ev_values = {}
288     #     # return "boia"
289     #     ev_names = ",".join([ev.key.id() for ev in Electrovalve.ev_list()])
290     #     memcache.set('ev_names', ev_names)
291     #     ev_names = ev_names.split(",")
292     #     ev_values = {}
293     #     for ev_name in ev_names:
294         #         ev_value = memcache.get(ev_name + '_value')
295         #
296         #             if ev_value is not None:
297         #                 ev_values[ev_name] = ev_value
298         #
299     #         return '%r' % ev_values[name]
300
301     #
302     # # ev_name =
303     # ev_key = ndb.Key("EV", name or "*notitle")
304     # ev = Electrovalve.query(name)
305     # return '%r' % ev.st
306
307 @app.route('/add/electrovalve', methods=['POST'])
308 def add_electrovalve():
309     key = request.form.get('name')
310     message = request.form.get('status')
311     if key and message:
312         if message == 'on' or message == 'On' or message == 'ON':
313             MESSAGES[key] = True
314             memcache.set(key, True)
315
316         else:
317             MESSAGES[key] = False
318             memcache.set(key, False)
319
320         # ev.put()
321     return '%r' % memcache.get(key)
322
323
324 @app.route('/glass')
325 def glass():
326     return "Buso e' qui"
327

```

```
329  
  @app.route("/show/<key>")  
331  def get_message(key):  
    return '%r' % MESSAGES[key] or "%s not found!" % key  
333  
335 if __name__ == "__main__":  
    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
41 import java.io.BufferedInputStream;
42 import java.io.File;
43 import java.io.FileOutputStream;
44 import java.io.IOException;
45 import java.io.InputStream;
46 import java.net.HttpURLConnection;
47 import java.net.URL;
48 import java.net.URLConnection;
49 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 /**
57  * Service owning the LiveCard living in the timeline.
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);
```

```

83     /** Action is an enumerate used to implement switch-case for the extra text appended to the inte
84
85     private static enum Action
86     {
87         Menu, Temperature, pH, Video, Beating
88     }
89
90
91     private AppDrawer mCallback;
92
93     /** HandlerThread used to launch a background thread that manages the Data updating */
94     private HandlerThread mHandlerThread;
95     /** Handler used to launch a background thread that manages the Data updating */
96     private Handler mHandler;
97
98     /** INT associated to the Menu view request */
99     private static final int MENU = 0;
100    /** INT associated to the PH view request */
101    private static final int PH = 1;
102    /** INT associated to the Menu view request */
103    private static final int TEMPERATURE = 2;
104    /** INT associated to the Video view request */
105    private static final int VIDEO = 3;
106    /** INT associated to the Beating view request */
107    private static final int BEATING = 4;
108
109    /** updating data period (in ms) */
110    private static final long DATA_UPDATE_DELAY_MILLIS = 500;
111    /** updating graph period (in ms) */
112    private static final long GRAPH_UPDATE_DELAY_MILLIS = 500;
113    /** updating video period (in ms) */
114    private static final long FRAME_TIME_MILLIS = 100;
115    private static final long VIDEO_UPDATE_DELAY_MILLIS = 60*1000; //time for video
116
117    /** Runnable which describes the task to update the pH and Temperature sensors values */
118    private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable = new UpdateSensorValuesRunn
119    /** Runnable which describes the task to compute the pH and Temperature graph (starting from the
120    private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable = new UpdateSensorGraphsRunn
121
122    private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable = new UpdateMicroscop
123
124    private static final String VIDEO_FILE_NAME = Environment.getExternalStorageDirectory() + "/micro
125    private static final String TEMP_VIDEO_FILE_NAME = Environment.getExternalStorageDirectory() + "/t
126
127    /** String array for the sensors (PH and Temperature) */
128    private String[] mSensors = new String[]{PH_TAG, TEMPERATURE_TAG};
129
130    /*
131             Hash table used for creating the graphs

```



```
181             Log.i(LIVE_CARD_TAG, "State = Beating");
182             AppManager.getInstance().setState(BEATING);
183             break;
184
185         case pH:
186             Log.i(LIVE_CARD_TAG, "State = pH");
187             AppManager.getInstance().setState(PH);
188             break;
189
190         case Temperature:
191             Log.i(LIVE_CARD_TAG, "State = Temperature");
192             AppManager.getInstance().setState(TEMPERATURE);
193             break;
194
195         case Video:
196             Log.i(LIVE_CARD_TAG, "State = Video");
197             AppManager.getInstance().setState(VIDEO);
198             break;
199
200         default:
201             mLliveCard.navigate();
202             break;
203         }
204     }
205
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
210     return START_NOT_STICKY;
211 }
212
213 @Override
214 public void onDestroy() {
215     // Stop the Task which update the beating image
216     if(!mImageDownloader.isStopped())
217     {
218         mImageDownloader.setIsStopped(true);
219         Log.i(BEATING_TAG, "Removed Task");
220     }
221
222     // Stop the Task which update the sensors Graphs
223     if(!mUpdateSensorGraphsRunnable.isStopped())
224     {
225         mUpdateSensorGraphsRunnable.setStop(true);
226         Log.i(PH_TAG + " " + TEMPERATURE_TAG, "Removed Task of Graphs");
227     }
228
229     // Stop the Task which update the sensors value
230     if(!mUpdateSensorValuesRunnable.isStopped())
231     {
```

```

229         mUpdateSensorValuesRunnable.setStop(true);
230         Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Values");
231     }
232
233     if (!mUpdateMicroscopeVideoRunnable.isStopped())
234     {
235         mUpdateMicroscopeVideoRunnable.setStop(true);
236         Log.i(VIDEO_TAG , "Removed Task of Uploading values");
237     }
238
239     if (mLiveCard != null && mLiveCard.isPublished()) {
240         mLiveCard.unpublish();
241         mLiveCard = null;
242     }
243
244     super.onDestroy();
245 }
246
247 /**
248 * Asynchronous Task for downloading the graphs and video in background
249 */
250 private class DataTask extends AsyncTask<MainService,Void(Void>
251 {
252
253     @Override
254     protected Void doInBackground(MainService... params) {
255
256         Log.i(LIVE_CARD_TAG , "Loading initial data");
257
258         // create the three hash tables
259         mCurrentSensorValues = new HashMap<String, Double>();
260         mSensorGraphData = new HashMap<String, ArrayList<DataPoint>>();
261         mCurrentSensorGraphs = new HashMap<String, Bitmap>();
262         mSensorAverage = new HashMap<String, Double>();
263
264         // initializes each hash map with dummy values
265         for (String mSensor : mSensors)
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             // if (ni.isConnected()) {
280                 mUpdateSensorValuesRunnable.run();
281             }
282         }
283     }
284 }
```

```
279         // give the hash table to the Callback
280         // mCallback.setSensorValues(mCurrentSensorValues);
281         // compute the graphs with previous values and give them to the AppDrawer
282         mUpdateSensorGraphsRunnable.run();
283
284         // give the hash table to the Callback
285         // mCallback.setSensorGraphs(mCurrentSensorGraphs);
286         // download the beating image and give this to the AppDrawer
287         mImageDownloader.run();
288
289         mUpdateMicroscopeVideoRunnable.run();
290     }
291
292
293
294
295     return null;
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
304     private boolean mIsStopped = false;
305
306     /** It implements the task of runnable
307      *
308      * @see UpdateSensorValuesRunnable
309      */
310
311     @Override
312     public void run()
313     {
314
315         if (!isStopped())
316         {
317
318             /** JavaScript object in which the sensor values are temporary stored */
319             JSONObject values = getSensorValues();
320             if (values != null)
321             {
322
323                 for (String mSensor : mSensors)
324                 {
325
326                     try
327                     {
328
329                         // update the hash table of sensor values
330                         mCurrentSensorValues.put(mSensor, values.getDouble(mSensor));
331
332                     }
333
334                 }
335
336             }
337
338         }
339
340     }
341 }
```

```

327             catch (JSONException ignored)
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  */
343 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 extracts
369         // the values
370         return new JSONObject(new JSONTokener(values));
371     }
372     catch (Exception e)
373     {
374         Log.e(LIVE_CARD_TAG, "Failed to get sensor values", e);
375         return null;
376     }
377 }

```

```
377 // Get the new data points that we will need to graph
379
380 /**
381 * @param sensor , the name of sensor (pH or Temperature)
382 * @param last_timestamp , the previous value of timestamp
383 * @return JSONArray, return a list of values that corresponds to the points that have to be plotted
384 */
385
386 private JSONArray getDataPoints(String sensor, double last_timestamp)
387 {
388     try
389     {
390         String url = "http://planar-contact-601.appspot.com/graphing_data?sensor=" + sensor + "&last_timestamp=" + last_timestamp;
391         String values = getURL(url);
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
409 private String getURL(String _url) throws IOException
410 {
411     URL url = new URL(_url);
412     InputStream is = url.openStream();
413     int ptr;
414     StringBuffer buffer = new StringBuffer();
415     while ((ptr = is.read()) != -1)
416     {
417         buffer.append((char)ptr);
418     }
419     return buffer.toString();
420 }
421
422 /**
423 * This runnable updates the graphs of pH and Temperature */
424
425 private class UpdateSensorGraphsRunnable implements Runnable {
426     private boolean mIsStopped = false;
427     public void run()
428     {
```

```

425     if (!isStopped())
426     {
427         // GraphViewSeries exampleSeries = new GraphViewSeries(new GraphView.GraphViewData[] +
428         // { new GraphView.GraphViewData(1, 2.0d),
429         //   new GraphView.GraphViewData(2, 1.5d),
430         //   new GraphView.GraphViewData(3, 2.5d),
431         //   new GraphView.GraphViewData(4, 1.0d)
432         // });
433     }
434
435     // GraphView graphView = new LineGraphView( MainService.this, "GraphViewDemo");
436
437     // graphView.addSeries(exampleSeries);
438
439     // mCallback.setPHGraphViewer(graphView);
440
441
442
443     // Loop through each of the sensors
444     for (String curr_sensor : mSensors) {
445         ArrayList<DataPoint> curr_data = mSensorGraphData.get(curr_sensor);
446         double lastTimestamp = curr_data.size() > 0 ? curr_data.get(curr_data.size() - 1).getTimestamp();
447         // Get a list of the new data points for this sensor
448         JSONArray newPoints = getDataPoints(curr_sensor, lastTimestamp);
449         for (int j = 0; j < newPoints.length(); j++) {
450             // Save each data point
451             try {
452                 JSONObject point = newPoints.getJSONObject(j);
453                 double timestamp = (Double) point.get("timestamp");
454                 double value = (Double) point.get("value");
455                 curr_data.add(new DataPoint(timestamp, value));
456             } catch (JSONException e) {
457                 Log.e(LIVE_CARD_TAG, "JSON error", e);
458                 //continue;
459             }
460         }
461         // Clear out points that are over an hour old
462         while (true) {
463             if (curr_data.size() > 0 && curr_data.get(0).getTimestamp() < (curr_data.get(0).getTimestamp() - 3600000))
464                 Log.i(LIVE_CARD_TAG, "Deleting old data point");
465                 curr_data.remove(0);
466             } else {
467                 break;
468             }
469         }
470
471         // If there are no points don't show a graph
472         if (curr_data.size() == 0) {
473             continue;

```

```
        }

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;
489     double intervalSize = maxTimestamp - minTimestamp;
490     for (int i1 = 0; i1 < timestamps.size(); i1++) {
491         double currTimestamp = timestamps.get(i1);
492         currTimestamp -= minTimestamp;
493         currTimestamp /= intervalSize;
494         currTimestamp *= 100;
495         timestamps.set(i1, currTimestamp);
496     }
497     // Scale the value data
498     double maxVal = Collections.max(values);
499     double minVal = Collections.min(values);
500     maxVal *= 1.2;
501     minVal *= 0.8;
502     intervalSize = maxVal - minVal;
503     for (int i1 = 0; i1 < values.size(); i1++) {
504         double currVal = values.get(i1);
505         currVal -= minVal;
506         currVal /= intervalSize;
507         currVal *= 100;
508         values.set(i1, currVal);

511     }

513     // Data xData = Data.newData(timestamps);
514     Data yData = Data.newData(values);
515

517     Plot plot = Plots.newPlot(yData);
518     LineChart lineChart = GCharts.newLineChart(plot);
519     lineChart.setSize(400, 200);
520     lineChart.addYAxisLabels(AxisLabelsFactory.newNumericRangeAxisLabels(minVal, maxVal, 1000), 1000);
```

```

523             mCurrentSensorGraphs.put(curr_sensor, getBitmapFromURL(lineChart.toURLString()));

525         }

527         mCallback.SetSensorAvg(mSensorAverage);
528         mCallback.setSensorGraphs(mCurrentSensorGraphs);
529         Log.i("Main Service", "Graphs updated");

531         // restart it after a given amount of time
532         mHandler.postDelayed(mUpdateSensorGraphsRunnable, GRAPH_UPDATE_DELAY_MILLIS);
533     }
534 }

535     private double computeAverage(ArrayList<Double> values) {
536         double sum = 0.0;
537         if(!values.isEmpty()){
538             for(Double value : values){
539                 sum += value;
540             }
541             return sum/values.size();
542         }
543         return sum;
544     }

545

546     /**
547      * It is the getter function that shows the status of runnable
548      *
549      * @return mIsStopped, true if the runnable is stopped, false otherwise
550      */
551     public boolean isStopped()
552     {
553         return mIsStopped;
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  *     }
574  * }
575 */

```

```
//           mValue = value;
573 //       }
//       public double getTimestamp() {
575 //           return mTimestamp;
//       }
577 //       public double getValue() {
//           return mValue;
579 //       }
//   }

581

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

588     /**
589      * Class constructor of ImageDownloader runnable
590      *
591      * @param url, url link of image
592      * @param c, is the context in which the request of downloading image has been sent
593      * @see
594      */
595     public ImageDownloader(String url, Context c) {
596         this.url = url;
597         this.c = c;
598     }

599

600     /**
601      * It implements the task of ImageDownloader runnable
602      *
603      * @see
604      */
605     @Override
606     public void run() {
607         if (!mIsStopped) {
608             Bitmap bmp = getBitmapFromURL(url);
609
610             mCallback.setBMP(bmp);
611             Log.i(BEATING_TAG, "bmp settato");
612         }
613     }
614 }

615

616     /**
617      * It is the getter function that shows the status of ImageDownloader runnable
618      *
619      * @return mIsStopped, true if the runnable is stopped, false otherwise
620      */
621 }
```

```

621     public boolean isStopped()
622     {
623         return mIsStopped;
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      *
637      * @param urlLink where the image is stored
638      * @return Bitmap which contains the image of the graph
639      * @throws java.io.IOException if an IO exception occurred during the download
640      */
641     public static Bitmap getBitmapFromURL(String urlLink){
642         try{
643             Log.i(BEATING_TAG, "start downloading image ");
644             long startTime = System.currentTimeMillis();
645             URL url = new URL(urlLink);
646             HttpURLConnection connection = (HttpURLConnection) url.openConnection();
647             connection.setDoInput(true);
648             connection.connect();
649             InputStream inputStream = connection.getInputStream();
650             Log.i(BEATING_TAG,"download completed in "
651                   + ((System.currentTimeMillis() - startTime) / 1000)
652                   + " sec");
653             return BitmapFactory.decodeStream(inputStream);
654         } catch (IOException e) {
655             e.printStackTrace();
656             Log.e(BEATING_TAG, e.getMessage());
657             return null;
658         }
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 (!isStopped()) {

```

```
        getMicroscopeVideo();
671    mHandler.postDelayed(mUpdateMicroscopeVideoRunnable, VIDEO_UPDATE_DELAY_MILLIS);
    }
673}
public boolean isStopped() {
675    return mIsStopped;
}
677public void setStop(boolean isStopped) {
678    this.mIsStopped = isStopped;
}
680}
681
// Pull the microscope video from a URL
683private 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    }
}
```

719
721 }

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;

17 import java.util.Map;

19 /**
20 * {@link DirectRenderingCallback} used to draw the chronometer on the timeline {@link com.google.an
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     /** INT associated to the PH view request */
39     private static final int PH = 1;
40     /** INT associated to the Menu view request */
41     private static final int TEMPERATURE = 2;
42     /** INT associated to the Video view request */
43     private static final int VIDEO = 3;
44     /** INT associated to the Beating view request */

```

```
    private static final int BEATING = 4;
45

47    private VideoThread mRenderThread;

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

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

55

56    private final MainView mMainView;
57    private final BeatingView mBeatingView;
58    /** View object of the GlassWear pH window */
59    private final PHViewer mPhViewer;
60    private final TemperatureView mTemperatureView;

61

62    private SurfaceHolder mHolder;
63    private boolean mRenderingPaused;

64

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

71

72

73    private final MainView.Listener mMainListener = new MainView.Listener() {
74

75

76        @Override
77        public void onChange() {
78            // mMainDone = true;
79            //mBeatingView.setBaseMillis(0);
80            updateRenderingState();
81        }
82    };

83

84

85    /** Defines the Listener of pH viewer, it is used to communicate with
86     * that viewer and allowing its viewing */
87    private final PHViewer.Listener mPhListener = new PHViewer.Listener(){
88
89        @Override
90        /* This function is used when the {@link com.example.alik.bwhglass.PHViewer} Class wants
91         * to change the view object.
92         */
93        public void onChange(){
```

```

93         //state = PH;
94         updateRenderingState();
95     }
96
97 }
98
99 private final BeatingView.Listener mBeatingListener = new BeatingView.Listener() {
100
101     @Override
102     public void onChange() {
103         updateRenderingState();
104     }
105 }
106
107 private final TemperatureView.Listener mTemperatureListener = new TemperatureView.Listener() {
108
109     @Override
110     public void onChange() {
111         updateRenderingState();
112     }
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 /**
135  * Defines the ManageDataGraph of PH viewer, it is used to communicate with
136  * that viewer in order to update the hash map which contains the graph to be displayed*/
137 private final PHViewer.ManageBitmap mPHManageDataGraph = new PHViewer.ManageBitmap(){
138
139     public Bitmap getBitmap() {
140         return mCurrentSensorGraphs.get("pH");
141     }
142
143     public double getAvg(){return mSensorAverage.get("pH");}
144 }
```

```
143     public boolean isReady(){
144         return mReady;
145     }
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");}
158
159     public boolean isReady(){
160         return mReady;
161     }
162 };
163
164
165     public AppDrawer.(Context context) {
166         this(new MainView(context), new BeatingView(context), new PHViewer(context), new TemperatureView());
167     }
168
169     public AppDrawer.(MainView countDownView, BeatingView chronometerView, PHViewer phViewer, TemperatureView temperatureView) {
170         mMainView = countDownView;
171         mMainView.setListener(mMainListener);
172
173         mBeatingView = chronometerView;
174         mBeatingView.setListener(mBeatingListener);
175         mBeatingView.setManageBPM(mBeatingBitmap);
176
177         mPhViewer = phViewer;
178         mPhViewer.setListener(mPhListener);
179         mPhViewer.setManageBPM(mPHManageDataGraph);
180
181         mTemperatureView = temperatureView;
182         mTemperatureView.setListener(mTemperatureListener);
183         mTemperatureView.setManageBPM(mTemperatureManageDataGraph);
184
185         mRenderThread = new VideoThread(mHolder);
186         mReady = false;
187     }
188
189 /**
190 *
```

```
191     * Uses the provided {@code width} and {@code height} to measure and layout the inflated
192     * {@link MainView} and {@link BeatingView}.
193     */
194
195     @Override
196     public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
197         // Measure and layout the view with the canvas dimensions.
198         int measuredWidth = View.MeasureSpec.makeMeasureSpec(width, View.MeasureSpec.EXACTLY);
199         int measuredHeight = View.MeasureSpec.makeMeasureSpec(height, View.MeasureSpec.EXACTLY);
200
201         mMainView.measure(measuredWidth, measuredHeight);
202         mMainView.layout(
203             0, 0, mMainView.getMeasuredWidth(), mMainView.getMeasuredHeight());
204
205         mBeatingView.measure(measuredWidth, measuredHeight);
206         mBeatingView.layout(
207             0, 0, mBeatingView.getMeasuredWidth(), mBeatingView.getMeasuredHeight());
208
209         mPhViewer.measure(measuredWidth, measuredHeight);
210         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
```

```
241     public void renderingPaused(SurfaceHolder holder, boolean paused) {
242         mRenderingPaused = paused;
243         updateRenderingState();
244     }
245
246     /**
247      * Starts or stops rendering according to the {@link com.google.android.glass.timeline.LiveCard}
248      */
249     private void updateRenderingState() {
250         if (mHolder != null && !mRenderingPaused) {
251             switch (AppManager.getInstance().getState())
252             {
253                 case MENU:
254                     mRenderThread.quit();
255                     // mMediaPlayer.setDisplay(null);
256                     draw(mMainView);
257                     mBeatingView.stop();
258                     mPhViewer.stop();
259                     mTemperatureView.stop();
260                     mMainView.start();
261                     break;
262                 case BEATING:
263                     // mMediaPlayer.setDisplay(null);
264                     draw(mBeatingView);
265                     mMainView.stop();
266                     mPhViewer.stop();
267                     mTemperatureView.stop();
268                     mBeatingView.start();
269                     break;
270                 case PH:
271                     // mMediaPlayer.setDisplay(null);
272                     draw(mPhViewer);
273                     mMainView.stop();
274                     mBeatingView.stop();
275                     mTemperatureView.stop();
276                     mPhViewer.start();
277                     break;
278                 case TEMPERATURE:
279                     // mMediaPlayer.setDisplay(null);
280                     draw(mTemperatureView);
281                     mMainView.stop();
282                     mBeatingView.stop();
283                     mPhViewer.stop();
284                     mTemperatureView.start();
285                     break;
286                 case VIDEO:
287                     // mMediaPlayer.setDisplay(mHolder);
288                     mRenderThread.setShouldRun(true);
289                     mRenderThread.start();
290             }
291         }
292     }
```

```
289         // mRenderThread.run();
290         mTemperatureView.stop();
291         mBeatingView.stop();
292         mMainView.stop();
293         mPhViewer.stop();

294         break;
295     default:
296         // mMediaPlayer.setDisplay(null);
297         draw(mMainView);
298         mPhViewer.stop();
299         mBeatingView.stop();
300         mTemperatureView.stop();
301         mMainView.start();
302         break;
303
304     }
305
306
307 } else {
308     mMainView.stop();
309     mBeatingView.stop();
310     mPhViewer.stop();
311     mTemperatureView.stop();
312 }
313
314 }
315
316 /**
317 * Draws the view in the SurfaceHolder's canvas.
318 */
319 private void draw(View view) {

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         view.draw(canvas);
330         mHolder.unlockCanvasAndPost(canvas);
331     }
332 }
333
334 /**
335 * Setter for the beating graph
336 *
337 * @param bmp , Bitmap which contains the beating graph
```

```

  */
339  public void setBMP(Bitmap bmp){

341      this.bmp = bmp;
342      this.mReady = true;
343      Log.i("DRAWER", "bmp settato");
344  }

345  public void setSensorGraphs( Map<String, Bitmap> sensorGraphs ){
346      this.mCurrentSensorGraphs = sensorGraphs;
347  }

348  public void setPHGraphViewer( GraphView graphView ){
349      this.mGraphView = graphView;
350  }

351

352

353

354  /**
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  public void setSensorValues( Map<String,Double> currentSensorValues ){
361      this.mCurrentSensorValues = currentSensorValues;
362  }

363

364

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

368

369

370 }

371 }

```

Listing A.19: MainView.java

```

/*
1  * Copyright (C) 2013 The Android Open Source Project
2  *
3  * Licensed under the Apache License, Version 2.0 (the "License");
4  * you may not use this file except in compliance with the License.
5  * You may obtain a copy of the License at
6  *
7  *     http://www.apache.org/licenses/LICENSE-2.0
8  *
9  * Unless required by applicable law or agreed to in writing, software
10 * distributed under the License is distributed on an "AS IS" BASIS,
11 * 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;
28
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     /**
50      * Notified when the countdown is finished.
51      */
52
53     public void onChange();
54
55     /** Time delimiter specifying when the second component is fully shown. */
56     private static final long DELAY_MILLIS = 40;
57
58
59     private final TextView timeText;
60
61 }
```

```
63     private final Handler mHandler = new Handler();
64
65     private final Runnable mUpdateViewRunnable = new Runnable() {
66
67         @Override
68         public void run() {
69             if (mRunning) {
70                 updateView();
71                 postDelayed(mUpdateViewRunnable, DELAY_MILLIS);
72             }
73         }
74     };
75
76
77     private Listener mListener;
78     private boolean mRunning = false;
79
80
81     public MainView(Context context) {
82         this(context, null, 0);
83     }
84
85     public MainView(Context context, AttributeSet attrs) {
86         this(context, attrs, 0);
87     }
88
89     public MainView(Context context, AttributeSet attrs, int style) {
90         super(context, attrs, style);
91         LayoutInflater.from(context).inflate(R.layout.live_card_layout, this);
92         timeText = (TextView) findViewById(R.id.timestamp);
93
94     }
95
96     /**
97      * Sets a {@link Listener}.
98      */
99
100    public void setListener(Listener listener) {
101        mListener = listener;
102    }
103
104    /**
105     * Returns the set {@link Listener}.
106     */
107    public Listener getListener() {
108        return mListener;
109    }
110
111    @Override
112    public boolean postDelayed(Runnable action, long delayMillis) {
113        return mHandler.postDelayed(action, delayMillis);
114    }
115
```

```

111
112     /**
113      * Starts the countdown animation if not yet started.
114      */
115     public void start() {
116         if (!mRunning) {
117             postDelayed(mUpdateViewRunnable, 0);
118         }
119         mRunning = true;
120     }
121
122     /**
123      * Stops the chronometer.
124      */
125     public void stop() {
126         if (mRunning) {
127             removeCallbacks(mUpdateViewRunnable);
128             // mStarted = false;
129         }
130         mRunning = false;
131     }
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     void updateView() {
141         //if (mRunning) {
142             timeText.setText( new SimpleDateFormat("hh:mm a").format( new Date() ) );
143             // updateView(millisLeft);
144             if (mListener != null) {
145                 mListener.onChange();
146             }
147         //}
148     }
149 }

```

Listing A.20: BeatingView.java

```

package com.google.android.glass.sample.klabinterface;
2
import android.content.Context;
4 import android.graphics.Bitmap;
5 import android.graphics.drawable.Drawable;
6 import android.os.Handler;

```

```
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

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 {

23
24     boolean mState = false;
25     private Bitmap bmp;

26     /**
27      * Interface to listen for changes on the view layout.
28      */
29
30     public interface Listener {
31         /** Notified of a change in the view. */
32         public void onChange();
33     }

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         Bitmap getBitmap();
56     }
```

```
56     public boolean isReady();
58 }
59
60     private Listener mChangeListener;
61
62     private ManageBitmap mManage;
63
64     public BeatingView(Context context) {
65         this(context, null, 0);
66     }
67
68     public BeatingView(Context context, AttributeSet attrs) {
69         this(context, attrs, 0);
70     }
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/lay
79         beatingImage.setImageResource(id);
80
81
82     }
83
84
85 /**
86 * Sets a {@link Listener}.
87 */
88 public void setListener(Listener listener) {
89     mChangeListener = listener;
90 }
91
92 /**
93 * Returns the set {@link Listener}.
94 */
95 public Listener getListener() {
96     return mChangeListener;
97 }
98
99
100 /**
101 * Starts the chronometer.
102 */
103 public void start() {
104     if (!mRunning) {
```

```
    postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
106
107    }
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
121 public boolean postDelayed(Runnable action, long delayMillis) {
122     return mHandler.postDelayed(action, delayMillis);
123 }
124
125 @Override
126 public boolean removeCallbacks(Runnable action) {
127     mHandler.removeCallbacks(action);
128     return true;
129 }
130
131 /**
132 * Sets a {@link Listener}.
133 */
134 public void setManageBPM(ManageBitmap manager) {
135     mManage = manager;
136 }
137
138 /**
139 * Updates the value of the chronometer, visible for testing.
140 */
141 void updateText() {
142     if (mManage.isReady())
143     {
144         Log.i("viewer", "ce prova");
145         bmp = mManage.getBitmap();
146         beatingImage.setImageBitmap(bmp);
147     }
148     // else
149     //{
150         // int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawa
151         // beatingImage.setImageResource(id);
152     //}
153     if (mChangeListener != null) {
```

```
154         mChangeListener.onChange();
155     }
156 }
158
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;
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.LinearLayout;
14 import android.widget.TextView;
15
16 import com.jjoe64.graphview.GraphView;
17 import com.jjoe64.graphview.GraphViewSeries;
18 import com.jjoe64.graphview.LineGraphView;
19
20
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     }
```

```
39  /** About 24 FPS, visible for testing. */
40  static final long DELAY_MILLIS = 41;
41
42  private ImageView beatingImage;
43  private final TextView mTextTitle;
44  private TextView AvgView;
45
46  private final Handler mHandler = new Handler();
47  private final Runnable mUpdateTextRunnable = new Runnable() {
48
49      @Override
50      public void run() {
51          if (mRunning) {
52              updateText();
53              postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
54          }
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      LayoutInflator.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");
```

```
89         int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/layer1");
90         beatingImage.setImageResource(id);
91     //     int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/layer2");
92     //     beatingImage.setImageResource(id);
93
94 }
95
96
97 /**
98 * Sets a {@link Listener}.
99 */
100 public void setListener(Listener listener) {
101     mChangeListener = listener;
102 }
103
104 /**
105 * Returns the set {@link Listener}.
106 */
107 public Listener getListener() {
108     return mChangeListener;
109 }
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 /**
122 * Stops the chronometer.
123 */
124 public void stop() {
125     if (mRunning) {
126         removeCallbacks(mUpdateTextRunnable);
127     }
128     mRunning = false;
129 }
130
131 @Override
132 public boolean postDelayed(Runnable action, long delayMillis) {
133     return mHandler.postDelayed(action, delayMillis);
134 }
135
136 @Override
```

```

137     public boolean removeCallbacks(Runnable action) {
138         mHandler.removeCallbacks(action);
139         return true;
140     }
141
142     /**
143      * Sets a {@link Listener}.
144      */
145     public void setManageBPM(ManageBitmap manager) {
146         mManage = manager;
147     }
148
149     /**
150      * Updates the value of the chronometer, visible for testing.
151      */
152     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/beating");
164             // beatingImage.setImageResource(id);
165         //}
166         if (mChangeListener != null) {
167             mChangeListener.onChange();
168         }
169     }
170
171 }
172
173
174 }
```

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;
25     private TextView AvgView;
26
27 /**
28  * Interface to listen for changes on the view layout.
29 */
30
31     public interface Listener {
32         /** Notified of a change in the view. */
33         public void onChange();
34     }
35
36 /**
37  * About 24 FPS, visible for testing.
38 */
39     static final long DELAY_MILLIS = 41;
40
41     private ImageView beatingImage;
42     private final TextView mTitle;
43
44     private final Handler mHandler = new Handler();
45     private final Runnable mUpdateTextRunnable = new Runnable() {
46
47         @Override
48         public void run() {
49             if (mRunning) {
50                 updateText();
51                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
52             }
53         }
54     };
55
56     private boolean mRunning;
57
58     public interface ManageBitmap{
59         public Bitmap getBitmap();
60     }
```

```
    public boolean isReady();  
58  
    public double getAvg();  
60 }  
  
62  
private Listener mChangeListener;  
64  
private ManageBitmap mManage;  
66  
public TemperatureView(Context context) {  
    this(context, null, 0);  
}  
70  
public TemperatureView(Context context, AttributeSet attrs) {  
    this(context, attrs, 0);  
}  
74  
public TemperatureView(Context context, AttributeSet attrs, int style) {  
    super(context, attrs, style);  
    LayoutInflator.from(context).inflate(R.layout.buso_layout, this);  
    beatingImage = (ImageView) findViewById(R.id.image_left);  
    mTitle = (TextView) findViewById(R.id.message);  
    AvgView = (TextView) findViewById(R.id.avg);  
  
    mTitle.setText("Temperature");  
    int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/la  
    beatingImage.setImageResource(id);  
  
86  
}  
88  
  
90 /**  
 * Sets a {@link Listener}.  
 */  
92 public void setListener(Listener listener) {  
    mChangeListener = listener;  
}  
94  
96 /**  
 * Returns the set {@link Listener}.  
 */  
98 public Listener getListener() {  
    return mChangeListener;  
}  
102  
104 /**  
 * Starts the chronometer.  
 */
```

```

106     */
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     }
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      * Sets a {@link Listener}.
137      */
138     public void setManageBPM(ManageBitmap manager) {
139         mManage = manager;
140     }
141
142     /**
143      * Updates the value of the chronometer, visible for testing.
144      */
145     void updateText() {
146         if (mManage.isReady())
147         {
148             Log.i("Temperature", "ce prova");
149             bmp = mManage.getBitmap();
150             AvgView.setText(Double.toString(mManage.getAvg()));
151             beatingImage.setImageBitmap(bmp);
152         }
153         // else
154     }

```

```
156     //{
157     //  int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/
158     //  beatingImage.setImageResource(id);
159     //}
160     if (mChangeListener != null) {
161         mChangeListener.onChange();
162     }
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             filepat = extras.getString("filepath");
30         else {
31             filepat = copyAsset(VIDEO_FILE_NAME);
32         }
33
34         Intent i = new Intent();
```

```

    i.setAction("com.google.glass.action.VIDEOPLAYER");
35   i.putExtra("video_url", filepath);
    startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);
37 }
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
39   if (requestCode == VIDEO_PLAY_REQUEST_CODE)
        finish();
41 }
String copyAsset(String filename) {
43   final String PATH = Environment.getExternalStorageDirectory().toString() + "/myvideoapps/";
44   File dir = new File(PATH);
45   if (!dir.exists()) {
46     if (!dir.mkdirs()) {
47       Log.v(TAG, "ERROR: Creation of directory " + PATH + " on sdcard failed");
48       return null;
49     } else {
50       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 }
}

```

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     private final Handler mHandler = new Handler();
32     private int state = 0;
33
34     @Override
35     public void onAttachedToWindow() {
36         super.onAttachedToWindow();
37         openOptionsMenu();
38     }
39
40     @Override
41     public boolean onCreateOptionsMenu(Menu menu) {
42         MenuInflater inflater = getMenuInflater();
43         inflater.inflate(R.menu.stopwatch, menu);
44         return true;
45     }
46
47     @Override
48     public boolean onPreparePanel(int featureId, View view, Menu menu) {
49         AppManager appManager = AppManager.getInstance();
50
51         boolean initialView = appManager.getState() == MENU;
52         boolean imageView = appManager.getState() == PH ||
53             appManager.getState() == TEMPERATURE ||
54             appManager.getState() == BEATING ||
55             appManager.getState() == VIDEO;
56     }
57 }
```

```

    setOptionsMenuState(menu, R.id.action_back, imageView);
58    setOptionsMenuState(menu, R.id.action_view_ph, initialView);
59    setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
60    setOptionsMenuState(menu, R.id.action_view_video, initialView);
61    setOptionsMenuState(menu, R.id.action_view_beating, initialView);
62    setOptionsMenuState(menu, R.id.action_stop, true);

64    return true;
}

66
@Override
68    public boolean onOptionsItemSelected(MenuItem item) {
69        // Handle item selection.
70        switch (item.getItemId()) {
71            case R.id.action_stop:
72                // Stop the service at the end of the message queue for proper options menu
73                // animation. This is only needed when starting a new Activity or stopping a Service
74                // that published a LiveCard.
75                post(new Runnable() {
76
77                    @Override
78                    public void run() {
79                        stopService(new Intent(MenuActivity.this, StopwatchService.class));
80                    }
81                });
82                return true;
83            case R.id.action_view_beating:
84                handleViewBeating();
85                return true;
86            case R.id.action_back:
87                handleViewBack();
88                return true;
89            case R.id.action_view_temperature:
90                handleViewTemperature();
91                return true;
92            case R.id.action_view_ph:
93                handleViewPh();
94                return true;
95            case R.id.action_view_video:
96                handleViewVideo();
97                return true;
98            default:
99                return super.onOptionsItemSelected(item);
100        }
101    }

102
private void handleViewVideo() {
103    Toast.makeText(this, "Video", Toast.LENGTH_SHORT).show();
104    // launch a new thread for starting a new live card

```

```
106     mHandler.post(new Runnable() {
107         @Override
108         public void run() {
109             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
110             intent.putExtra(Intent.EXTRA_TEXT, "Video");
111             startService(intent);
112         }
113     });
114 }
115
116 private void handleViewTemperature() {
117     Toast.makeText(this, "Temperature", Toast.LENGTH_SHORT).show();
118     // launch a new thread for starting a new live card
119     mHandler.post(new Runnable() {
120         @Override
121         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
129 private void handleViewPh() {
130     Toast.makeText(this, "pH", Toast.LENGTH_SHORT).show();
131     // launch a new thread for starting a new live card
132     mHandler.post(new Runnable() {
133         @Override
134         public void run() {
135             Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
136             intent.putExtra(Intent.EXTRA_TEXT, "pH");
137             startService(intent);
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 }
```

```

156     @Override
157     public void onOptionsMenuClosed(Menu menu) {
158         // Nothing else to do, closing the Activity.
159         finish();
160     }
161
162     /**
163      * Posts a {@link Runnable} at the end of the message loop, overridable for testing.
164      */
165
166     protected void post(Runnable runnable) {
167         mHandler.post(runnable);
168     }
169
170     /**
171      * The function handle the request to view the beating plot
172      */
173
174     private void handleViewBeating() {
175         Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
176         // launch a new thread for starting a new live card
177         mHandler.post(new Runnable() {
178             @Override
179             public void run() {
180                 Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
181                 intent.putExtra(Intent.EXTRA_TEXT, "Beating");
182                 startService(intent);
183             }
184         });
185     }
186
187     private static void setOptionsMenuState(Menu menu, int menuItemId, boolean enabled){
188         MenuItem menuItem = menu.findItem(menuItemId);
189         menuItem.setVisible(enabled);
190         menuItem.setEnabled(enabled);
191     }
192 }
```

Listing A.25: AppManager.java

```

1 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();
```

```
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

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

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	Driver for LED	42
Figure 38	LEDs experiments board	43
Figure 39	Circuit mounted on breadboard top view	44

Figure 40 Circuit mounted on breadboard side view 45

LIST OF LISTINGS

A.1	ElectrovalvesDriver.cpp	46
A.2	sensor_aquiring.cpp	48
A.3	recordVideo	52
A.4	compute_beating.cpp	52
A.5	Pins Setting	54
A.6	GPIO.h	56
A.7	GPIO.cpp	58
A.8	HTTP.h	64
A.9	HTTP.cpp	65
A.10	main.cpp	66
A.11	VideoStoring.pro	67
A.12	mytimer.h	67
A.13	mytimer.cpp	68
A.14	downloader.h	69
A.15	downloader.cpp	70
A.16	Main Script of Server	72
A.17	MainService.java	79
A.18	AppDrawer.java	94
A.19	MainView.java	101
A.20	BeatingView.java	104
A.21	PHViewer.java	108
A.22	TemperatureView.java	111
A.23	VideoPlayerActivity.java	115
A.24	MenuActivity.java	116
A.25	AppManager.java	120
A.26	DataPoint.java	121

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] 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>.