POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

Tesi di Laurea Magistrale

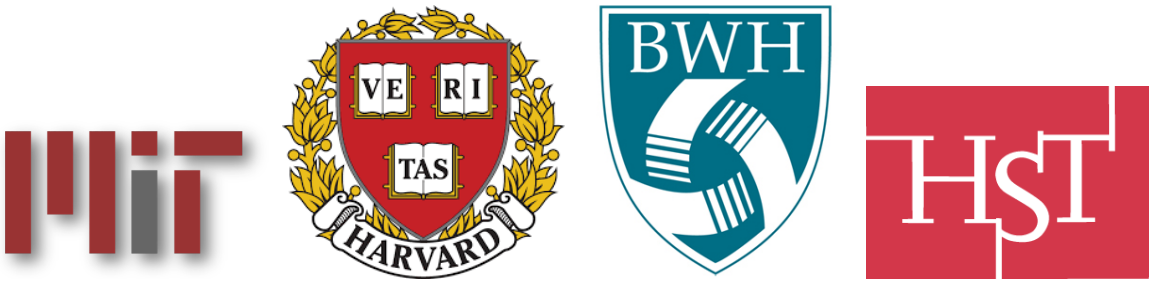
Google Glass Data Visualization and Monitoring for Organs-on-a-Chip and Biomedical Applications

Relatore
Prof. Danilo Demarchi

Candidato
Fabio Busignani s197883

Marzo 2015

ABSTRACT



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

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

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

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

[1] Google Glass Application

CONTENTS

INTRODUCTION

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

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

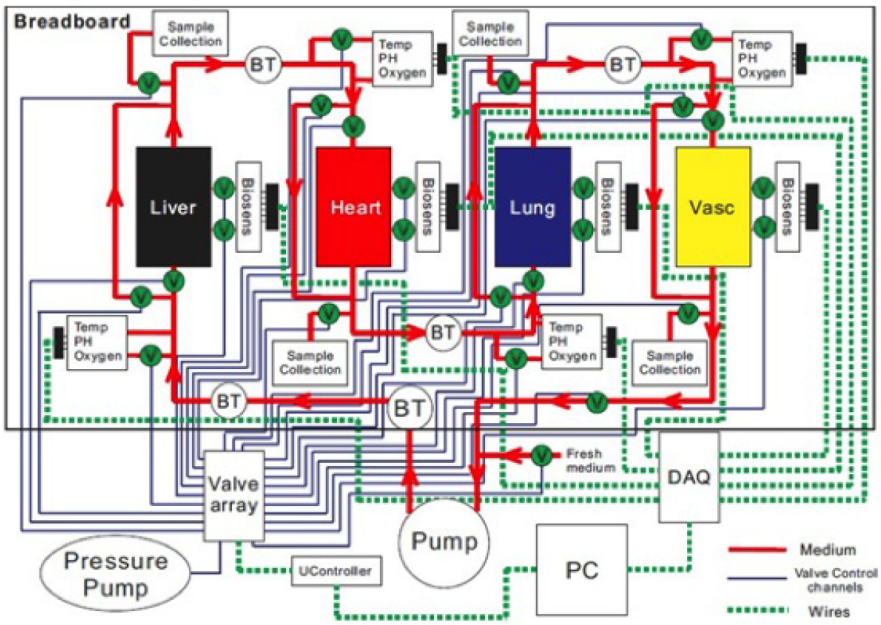


Figure 1: *XCEL* project (*Body-on-a-chip*)

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

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

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

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

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

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

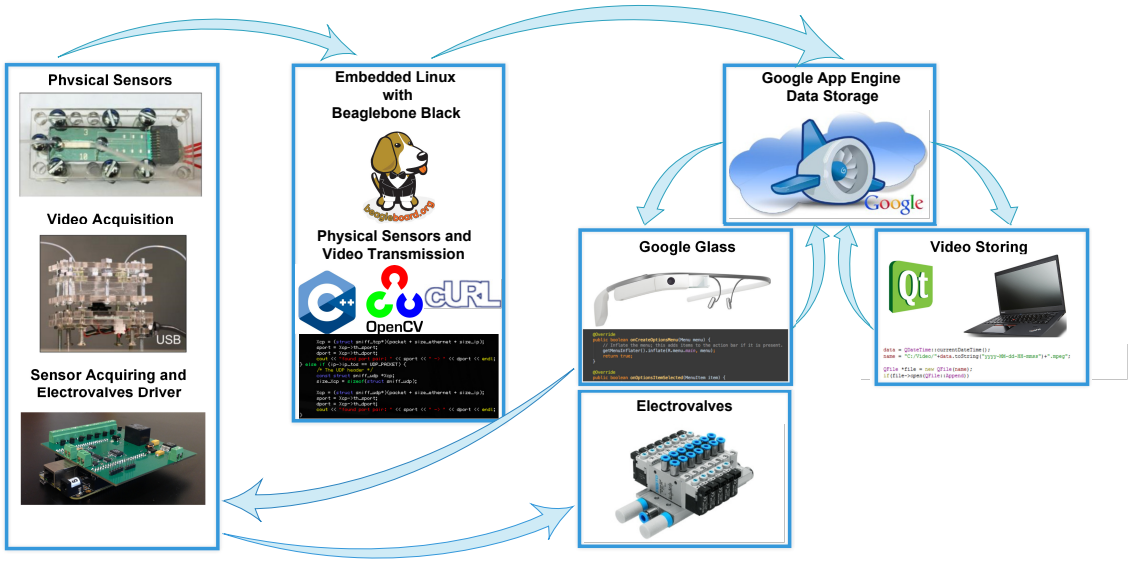


Figure 2: Block diagram of the system

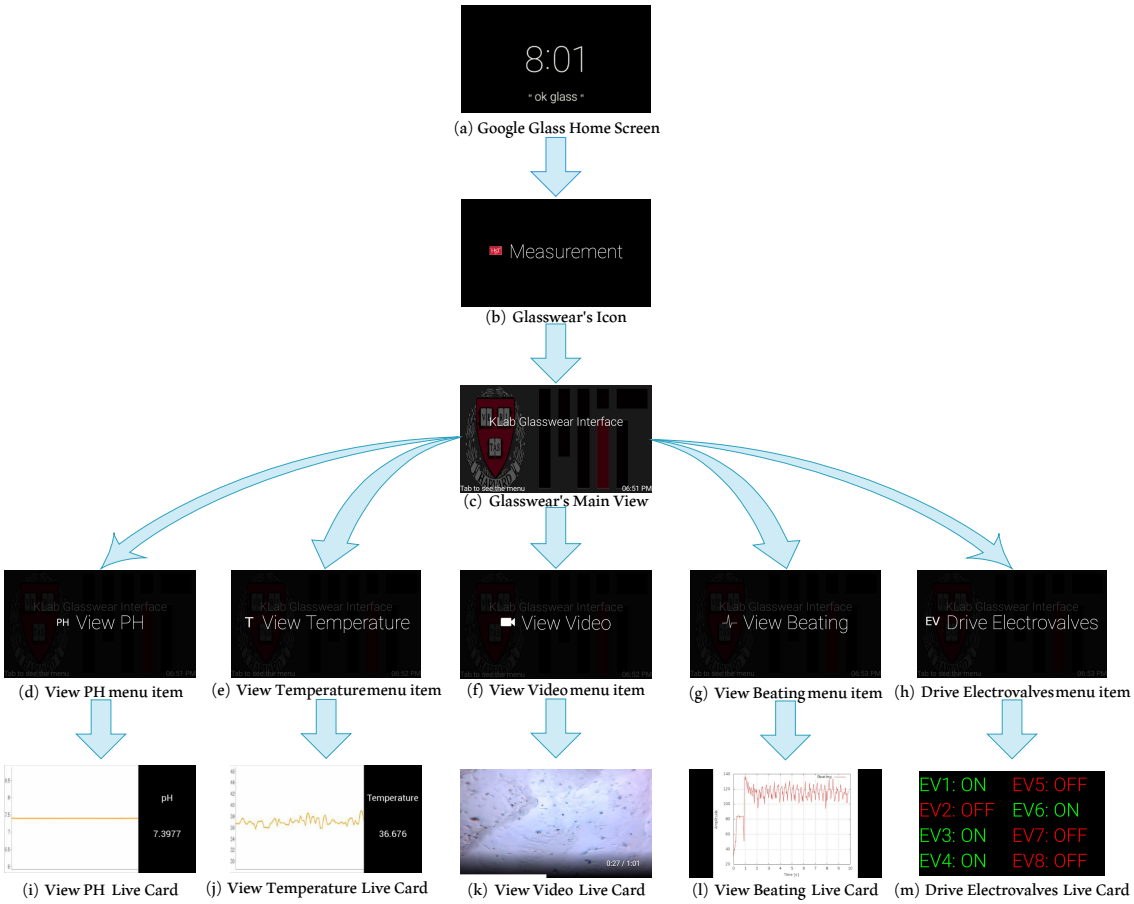
The (Fig.2) shows the principal steps of data transmission from physical and video sensors to the Google Glass via an *Embedded Linux System* performed using the **Beaglebone Black**.
The Beaglebone Black runs processes that are in charged to:

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

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

THE GLASSWEAR

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



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

Figure 3: Glasswear's Block Diagram

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

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

Figure 4: Drive Electrovalves Steps

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

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

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

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

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

Figure 5: *Back* menu item

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

Figure 6: *Exit* menu item

THE BOARD

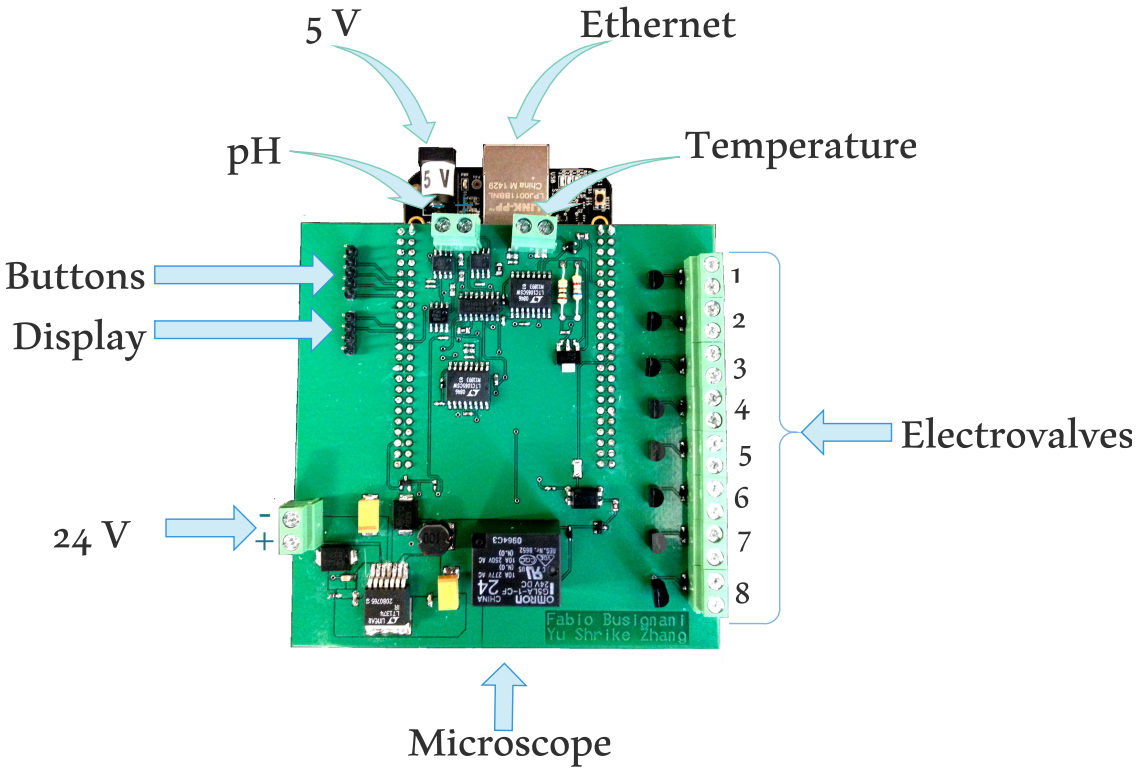


Figure 7: The Board

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

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

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

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

VIDEO STORING

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

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

Figure 8: Storing Microscope Video Program's Icon

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

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

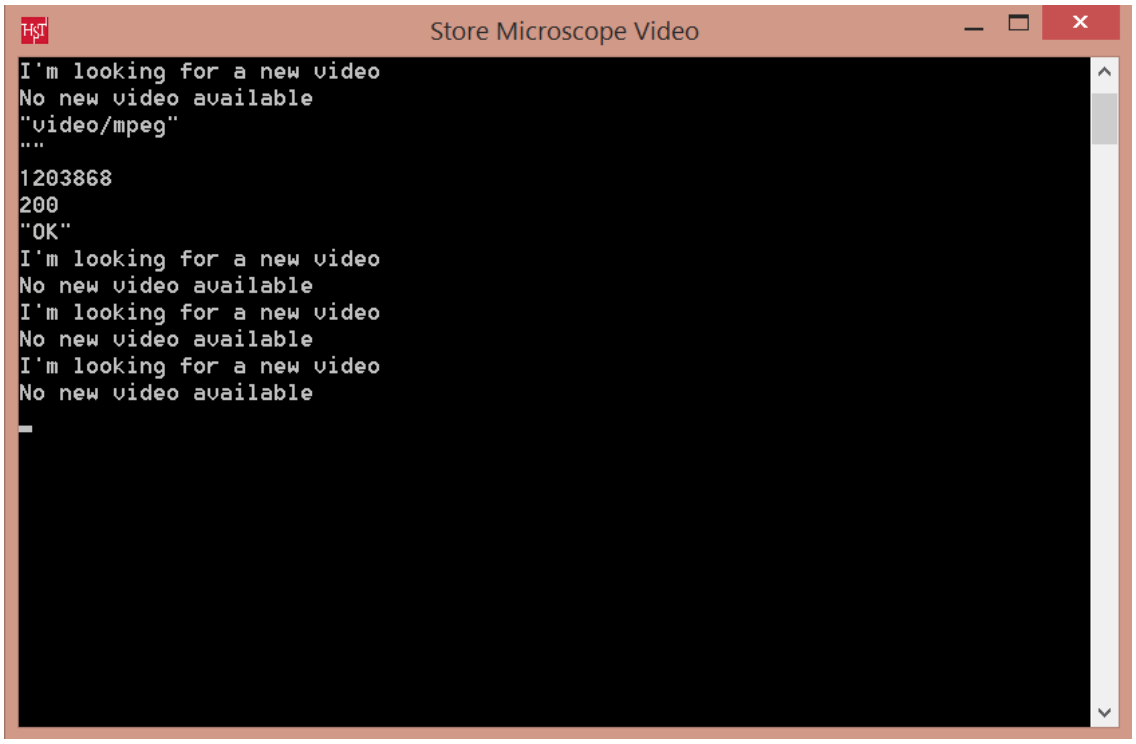


Figure 9: Storing Microscope Video Console

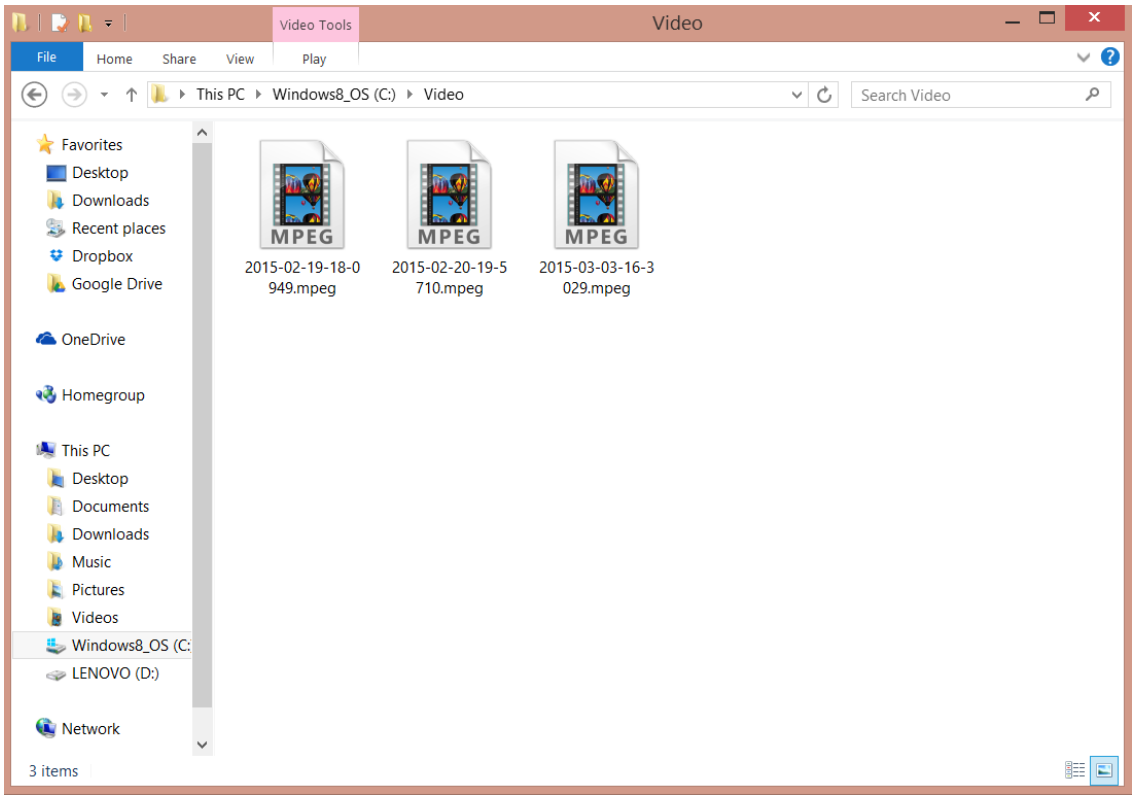


Figure 10: Video Stored in the Folder

Part I

HARDWARE

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

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

1 CONDITIONING CIRCUIT AND ELECTROVALVES DRIVERS

1.1 THE SENSORS

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

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

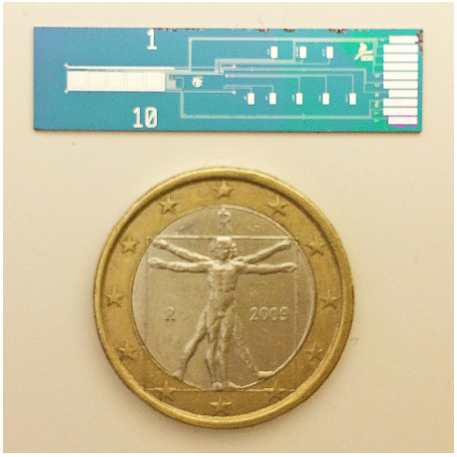


Figure 11: Multisensor - Size Comparison

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

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

1.1.1 PH Sensor

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

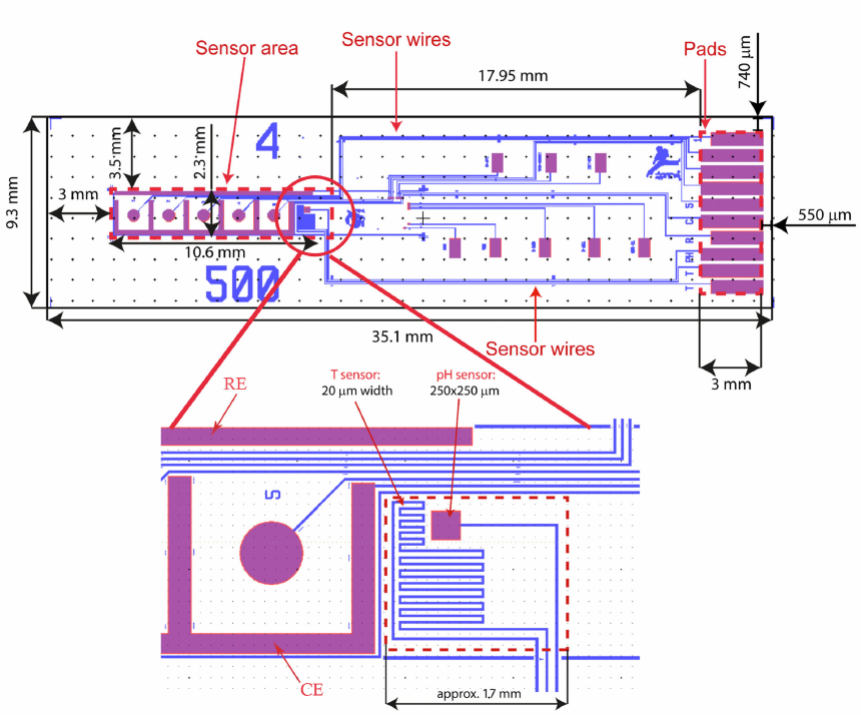


Figure 12: Multisensor - Scheme

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

1.1.2 Temperature Sensor

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

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

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

Where:

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

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

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

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

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

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

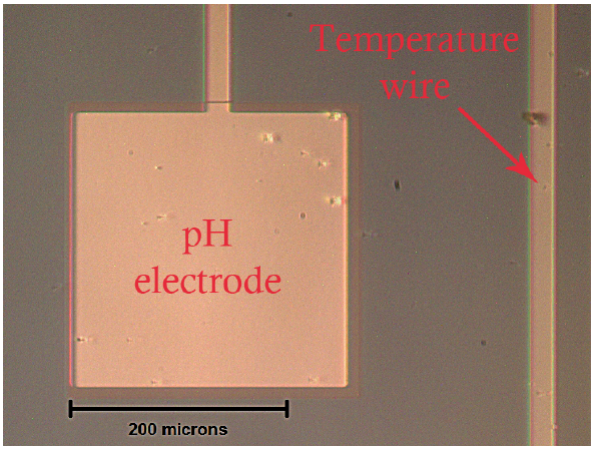


Figure 13: PH and Temperature Sensors from an Optical Image

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

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

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

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

1.2 PH CONDITIONING

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

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

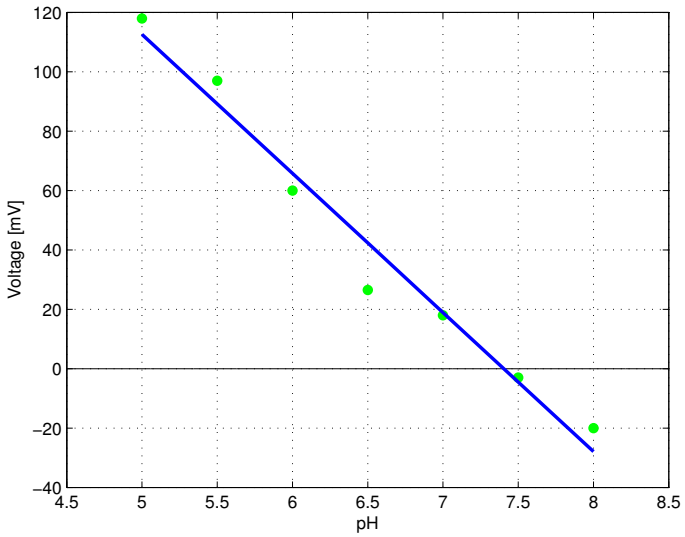


Figure 14: Typical pH-sensor transfer function

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

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

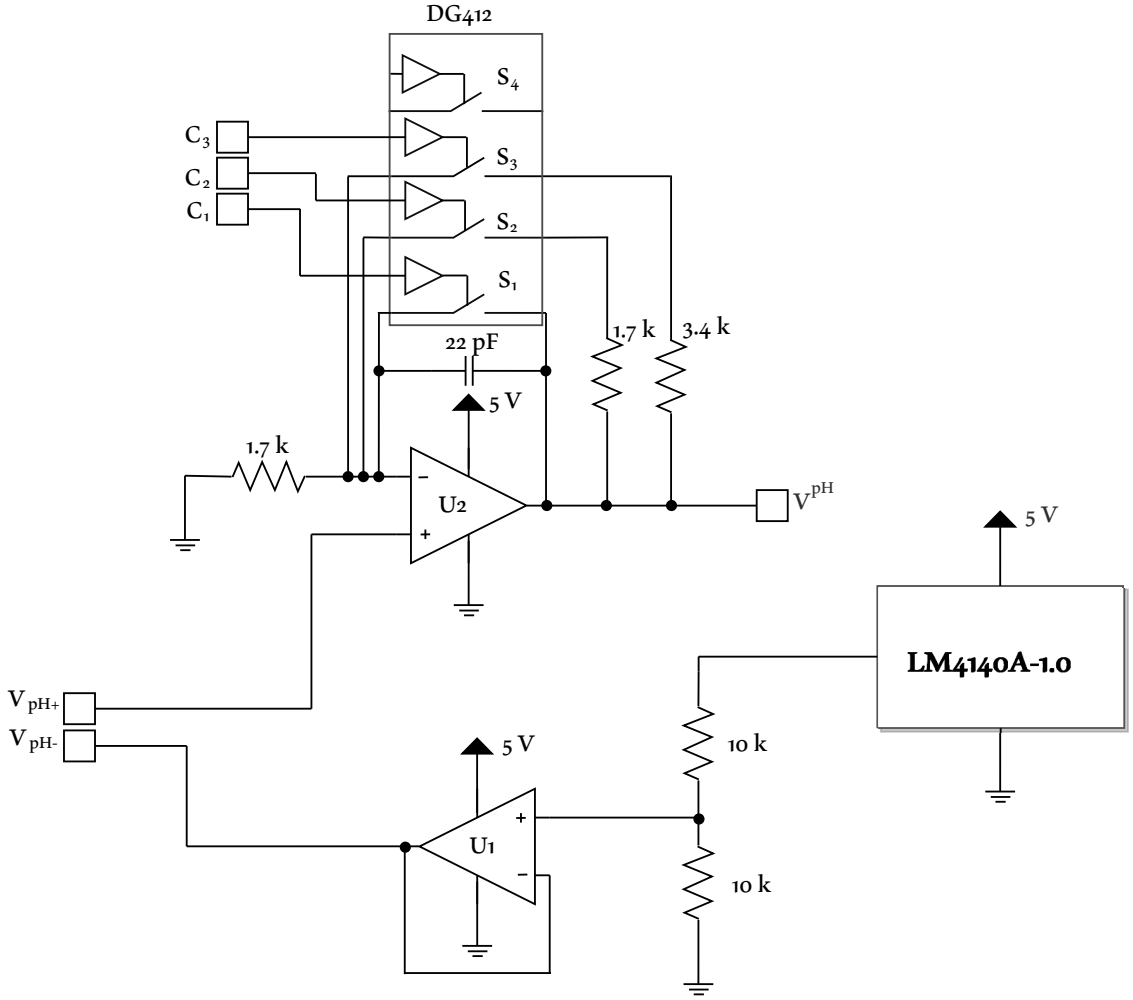


Figure 15: Conditioning circuit for pH sensor

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

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

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

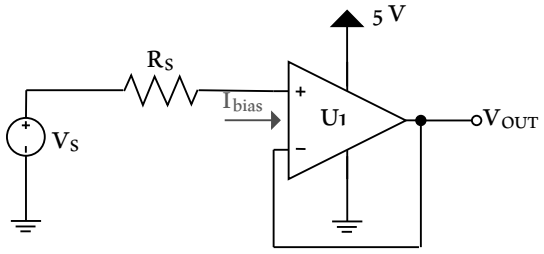


Figure 16: Error caused by Amplifier's Input Bias Current

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

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

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

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

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

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

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

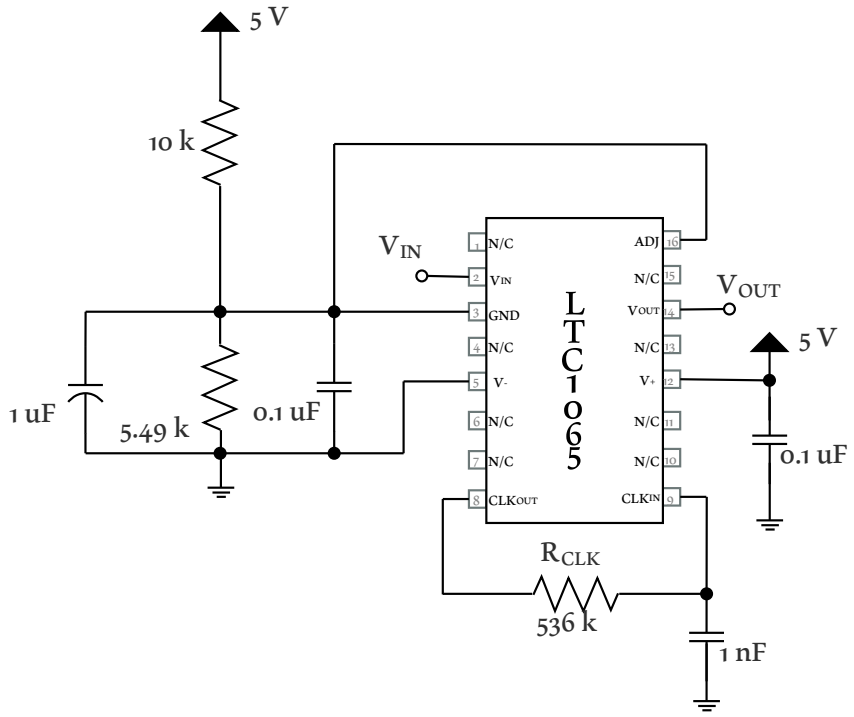


Figure 17: Low-Pass Filter Schematic

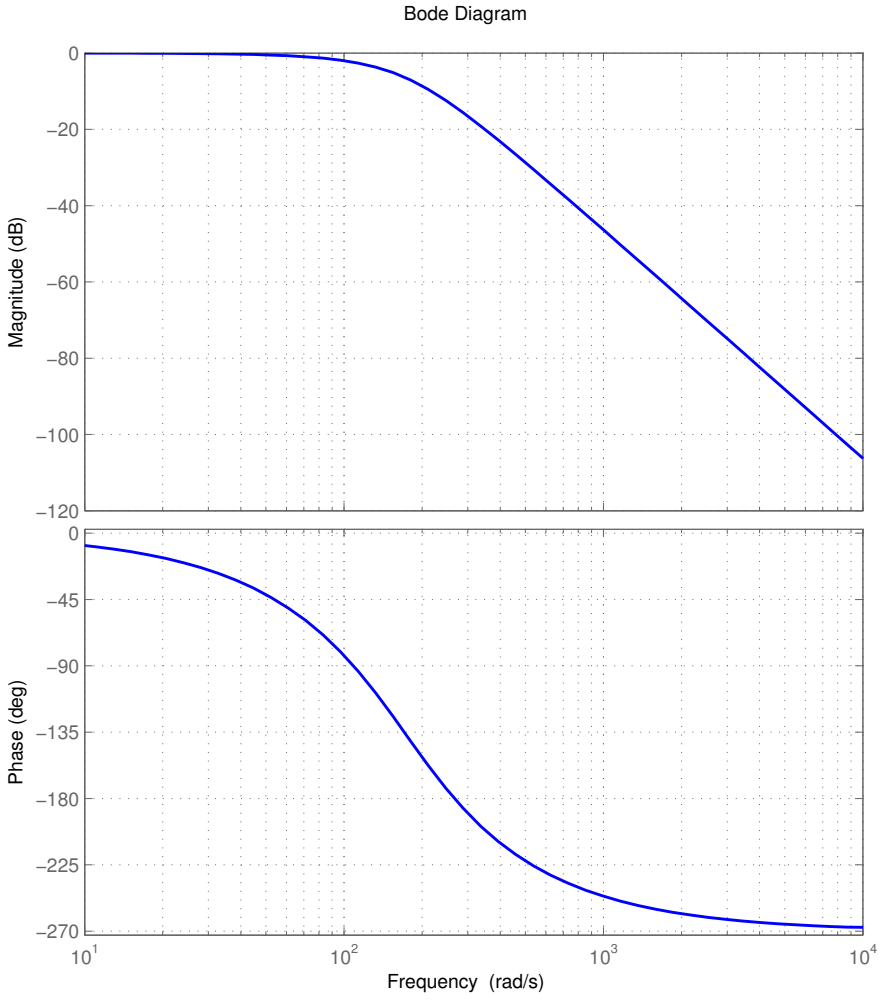


Figure 18: Low-Pass Filter Frequency Response

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

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

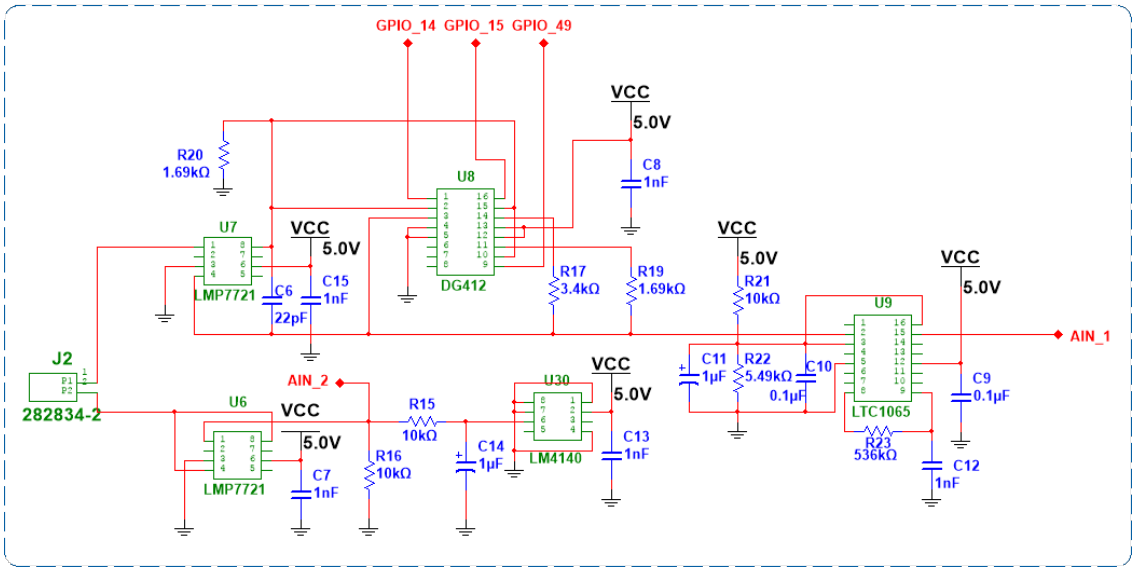


Figure 19: Acquisition Path for pH Sensor

1.3 TEMPERATURE CONDITIONING

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

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

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

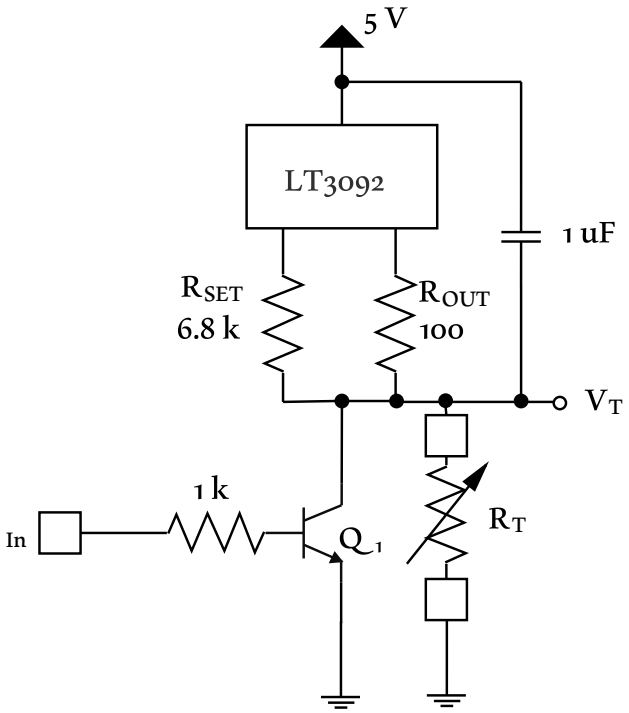


Figure 20: Conditioning Circuit for Temperature Sensor

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

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

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

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

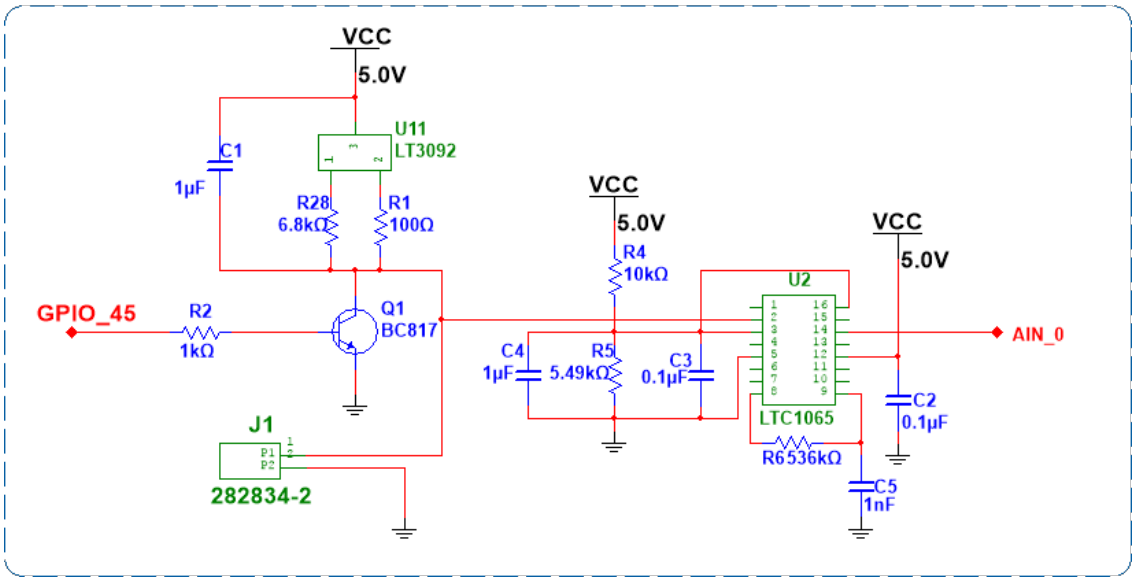


Figure 21: Acquisition Path for pH Sensor

1.4 ELECTROVALVES DRIVER

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

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

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

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

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

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

Figure 22: Electrovalves

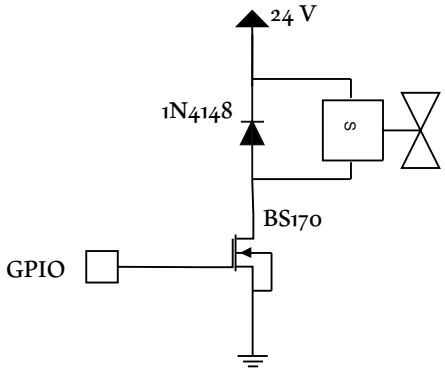


Figure 23: Driver for electrovalves

1.5 DC-DC CONVERTER

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

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

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

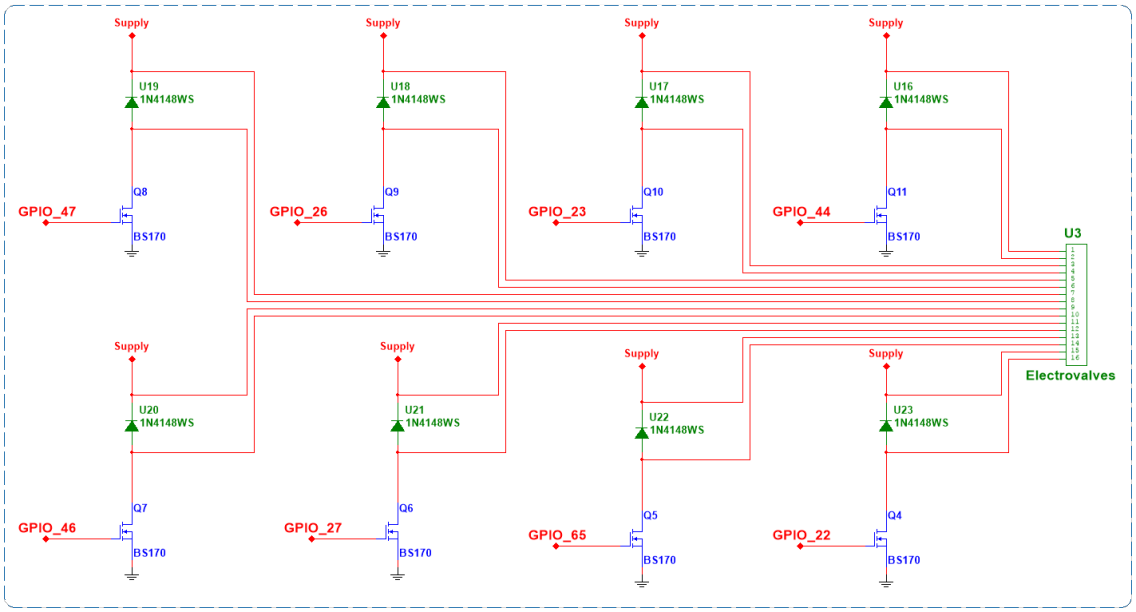


Figure 24: Electrovalves Driver Circuit

1.5.1 Components Choice

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

FEEDBACK RESISTORS

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

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

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

INDUCTOR

The choice of inductor is a trade-off among:

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

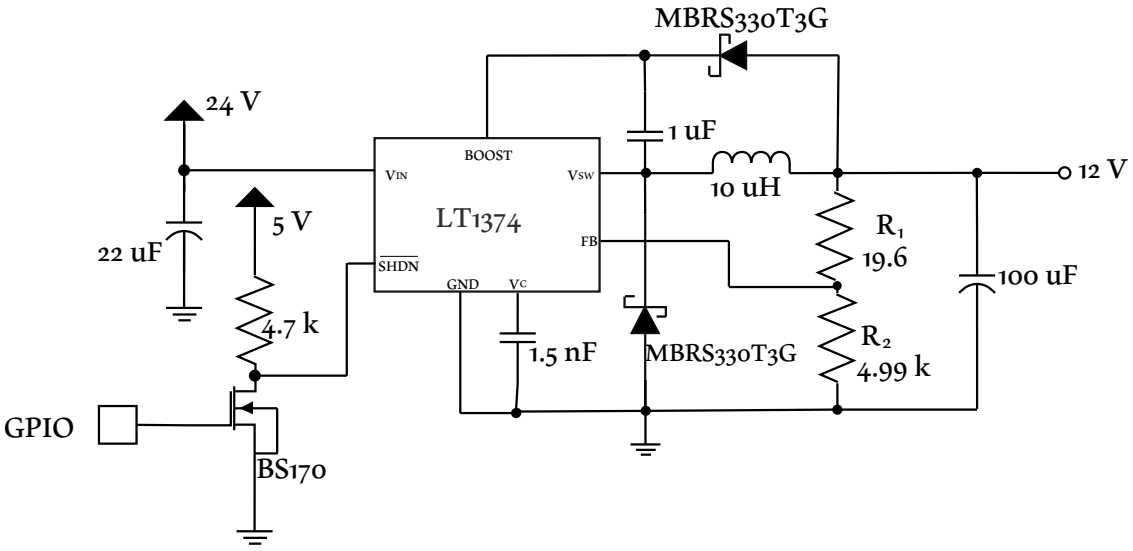


Figure 25: DC-DC circuit

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

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

OUTPUT CAPACITOR

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

SCHOTTKY DIODE

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

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

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

BOOST CAPACITOR

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

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

1.5.2 Relay

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

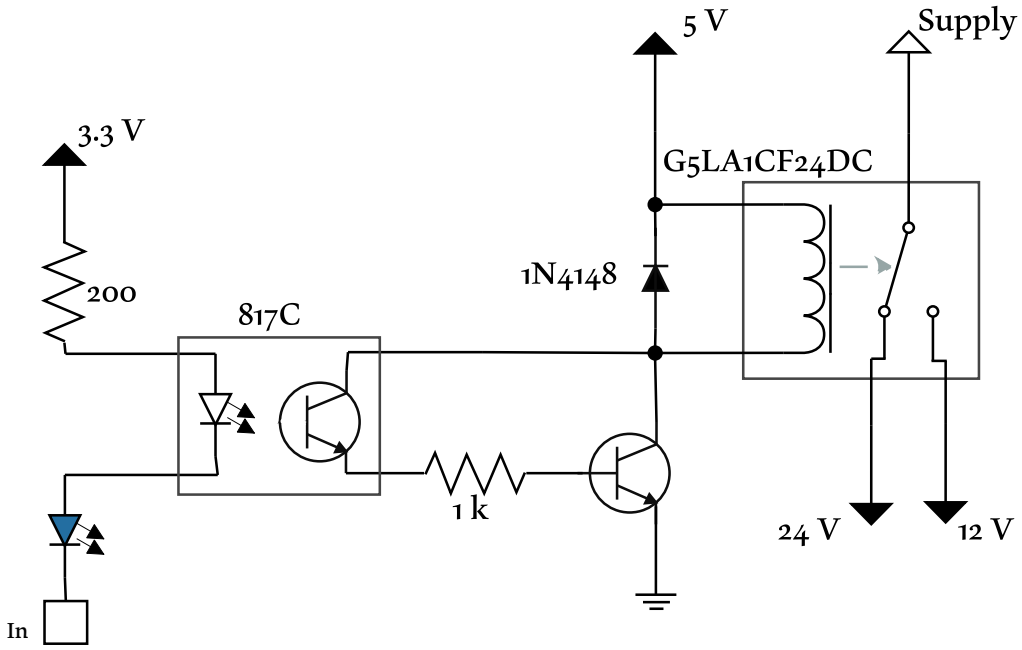


Figure 26: Relay circuit

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

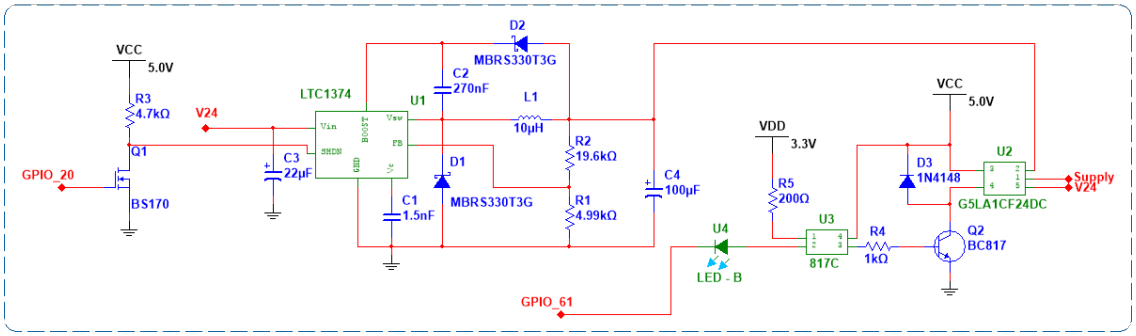


Figure 27: Electrovalves Supply Circuit

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

2 PCB

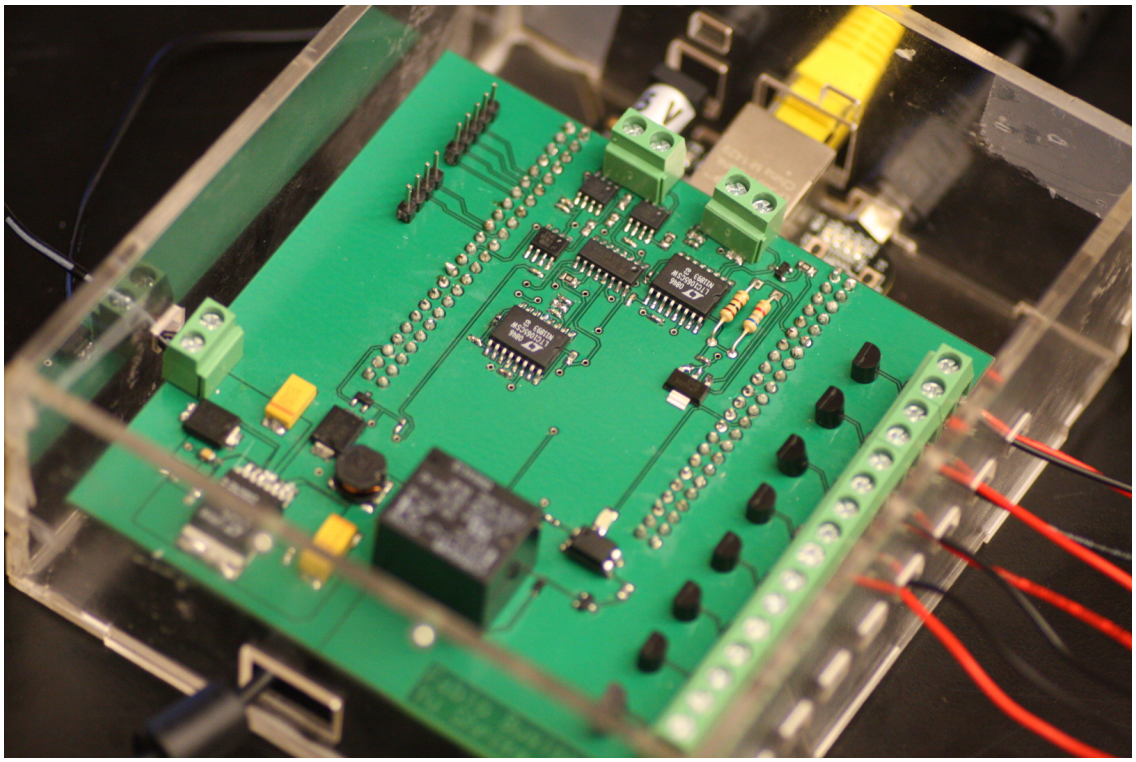


Figure 28: Final system mounted on a PCB

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

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

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

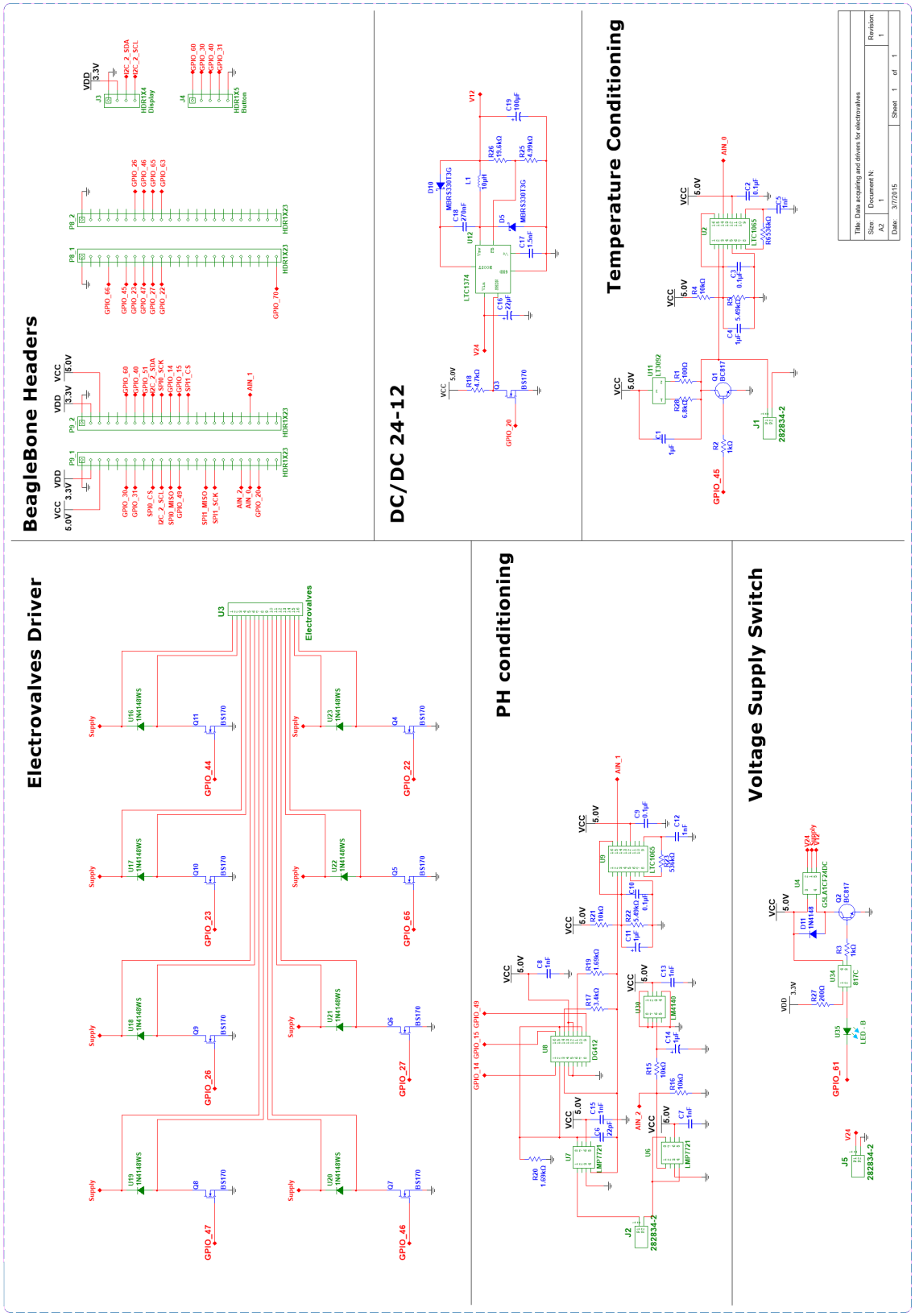


Figure 29: Schematic of Complete Circuit

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

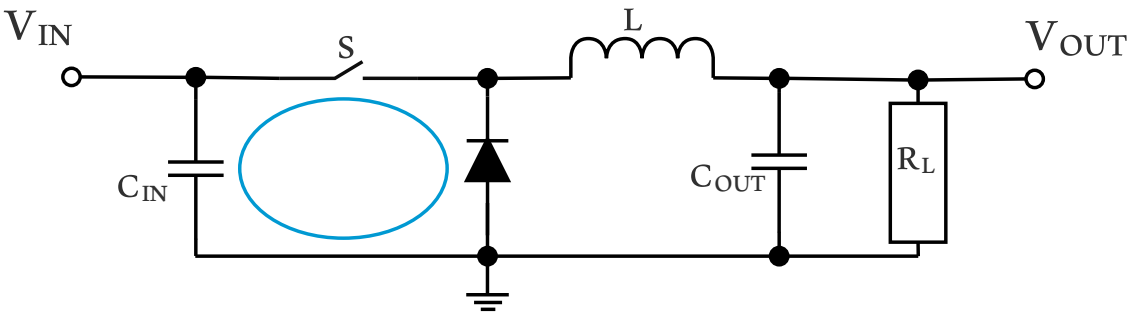


Figure 30: *DC-DC* High Speed Switching Path

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

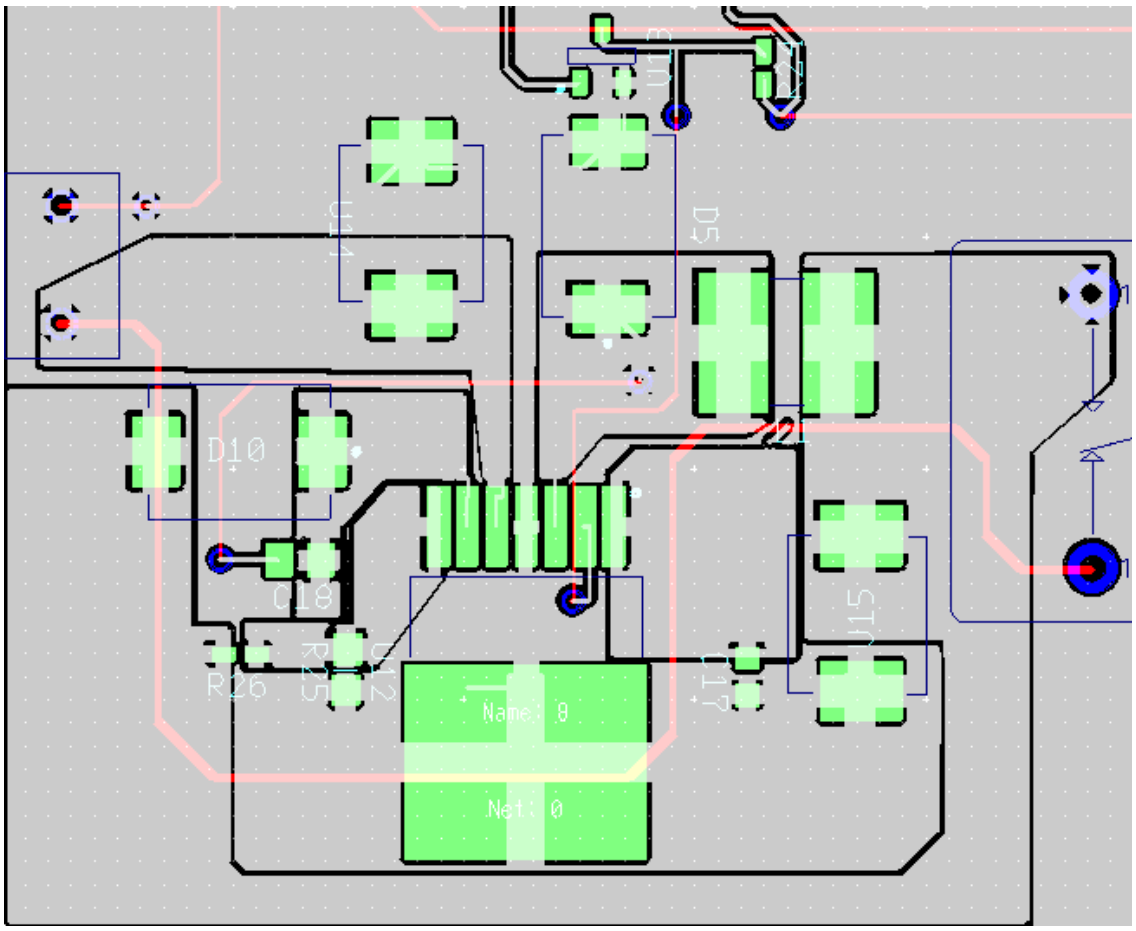


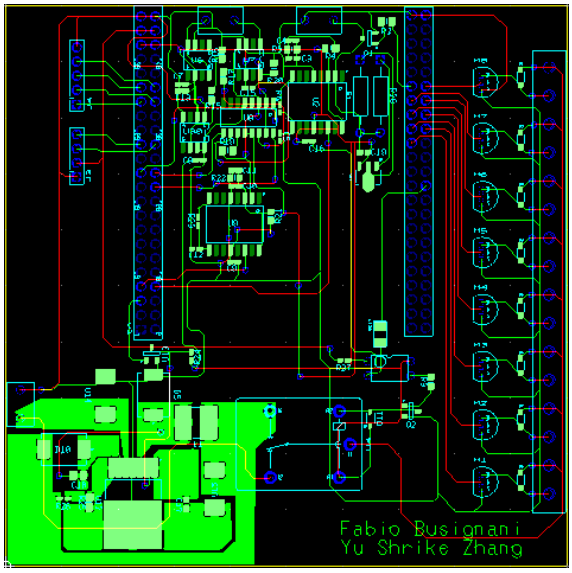
Figure 31: *PCB* Layout of *DC-DC*

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

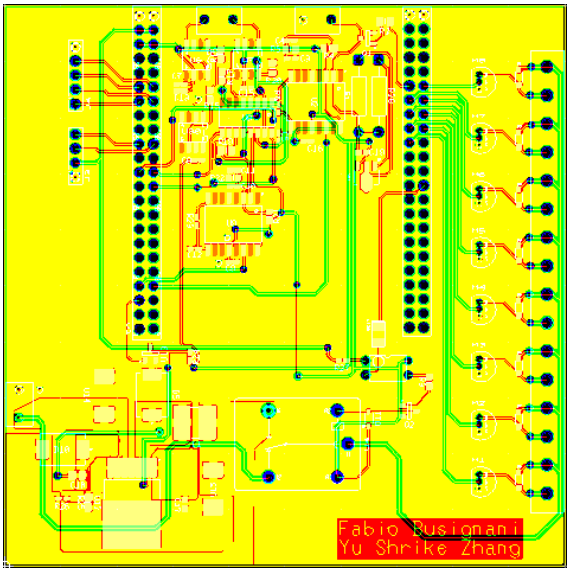
- the magnetic radiation is minimized by keeping by keeping catch diode and the input capacitor leads as short as possible;
- the electric radiation is minimized by reducing the area and length of all traces connected to the switch and boost pins.

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

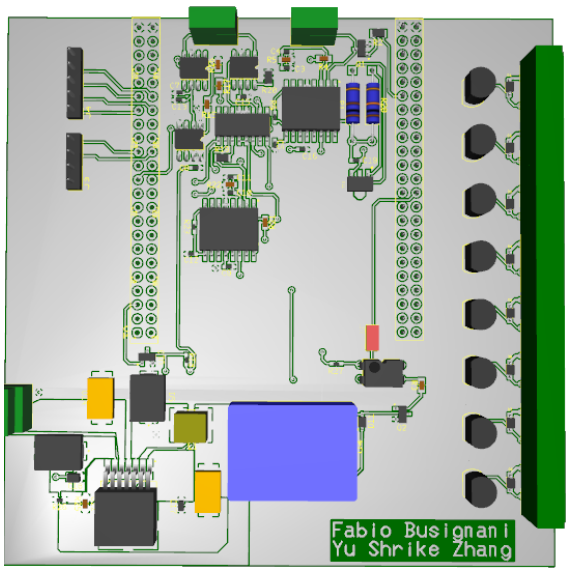
The (Fig.32) shows the *PCB* layout without ground plane (Fig.32a), then with both top and bottom ground plane (Fig.32b) which help with heat dissipation and *EMI* reduction. In (Fig.32c) and (Fig.32d) the 3-D model generated using *Ultiboard* and the final real result are compared.



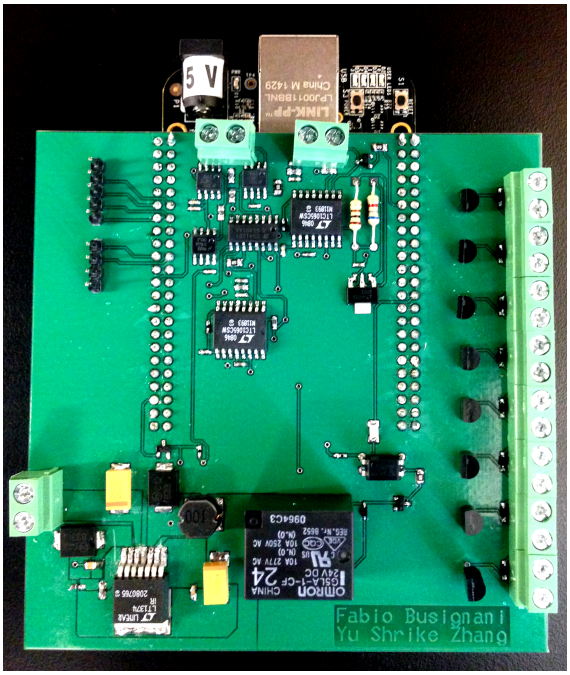
(a) *PCB* Layout Withot Ground Plane



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



(c) *PCB* 3-D Model View



(d) *PCB* Real Result

Figure 32: PCB of the System

Part II

FIRMWARE

3 INTRODUCTION

4 EMBEDDED LINUX

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.33).

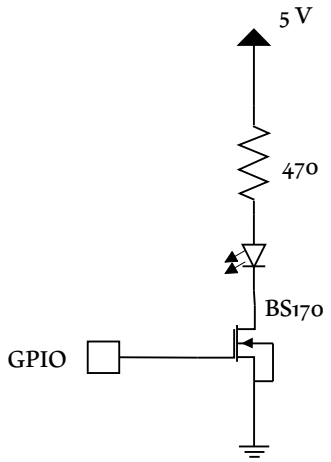


Figure 33: Driver for LED

The (Fig.34) 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.

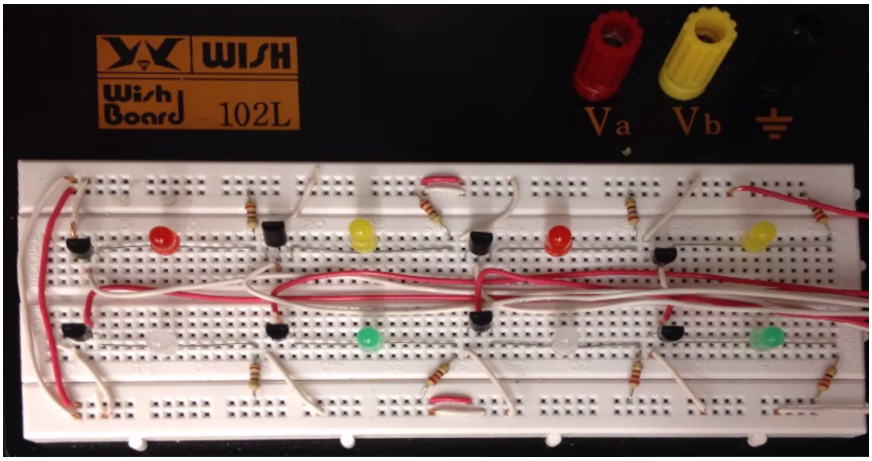


Figure 34: LEDs experiments board

8.2 ELECTROVALVES EXPERIMENTS

8.2.1 Breadboard Phase

The (Fig.35) 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.33), indeed the only difference is given by freewheeling diode, mandatory because of inductive behavior of electrovalve's solenoid.

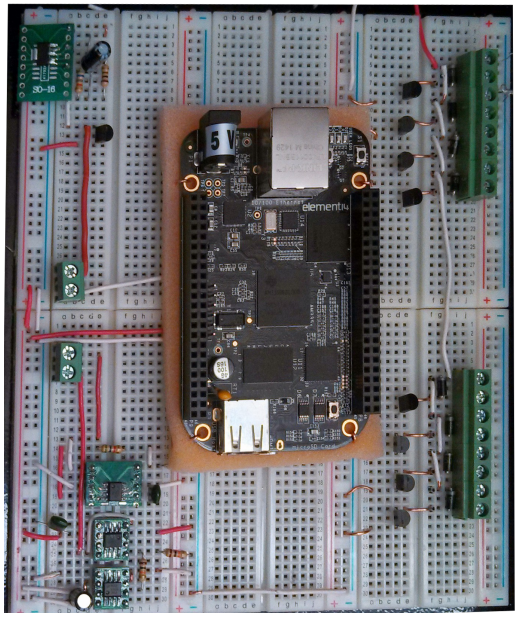


Figure 35: Circuit mounted on breadboard top view

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

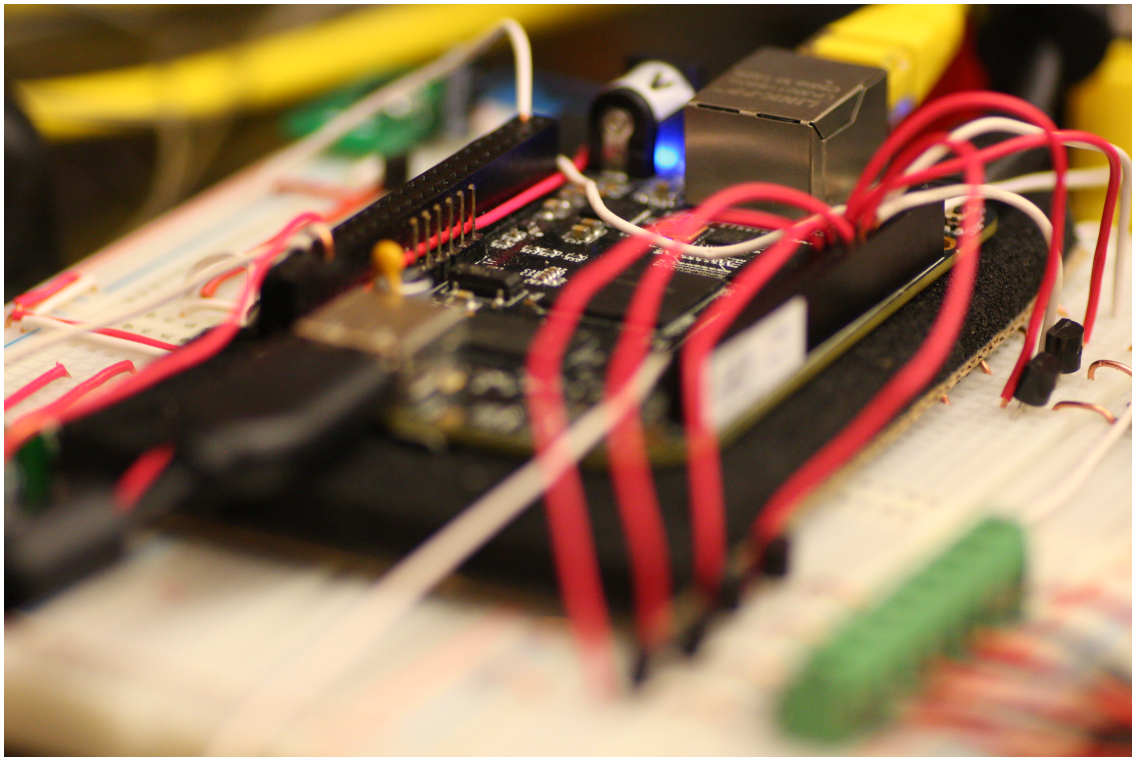


Figure 36: Circuit mounted on breadboard side view

8.2.2 PCB Phase

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

A CODE

A.1 FIRMWARE

Listing A.1: ElectrovalvesDriver.cpp

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

#define NUMBER_ELECTROVALVES 8

using namespace exploringBB;
using namespace std;

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

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


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

  CURL *curl;
  CURLcode res;

  GPIO_VALUE output_vector[NUMBER_ELECTROVALVES];

  //Set output pins
  EV1.setDirection(OUTPUT);
  EV2.setDirection(OUTPUT);
```

```
  EV3.setDirection(OUTPUT);
39 EV4.setDirection(OUTPUT);
  EV5.setDirection(OUTPUT);
41 EV6.setDirection(OUTPUT);
  EV7.setDirection(OUTPUT);
43 EV8.setDirection(OUTPUT);

45 RELAY.setDirection(OUTPUT);

47 RELAY.setValue(HIGH);

49 while(1)
  {
51   //acquire the values of electrovalves status and store them inside output_vector
    int i;
53   for(i=0;i<NUMBER_ELECTROVALVES; i++)
    {
55     curl = curl_easy_init();
      if(curl)
57     {
        std::string readBuffer;
59       std::ostringstream ss;
        ss << i+1;
61       std::string url = ELECTROVALVES_LINK + "EV" + ss.str();
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
63       /* example.com is redirected, so we tell libcurl to follow redirection */
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
65       curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
        /* Perform the request, res will get the return code */
67       res = curl_easy_perform(curl);
        /* always cleanup */
69       curl_easy_cleanup(curl);

71       if(readBuffer.compare(TRUE_VALUE) == 0)
        {
73         std::cout << "EV" + ss.str()+": ON" << std::endl;
          output_vector[i] = HIGH;
75       }
        else if(readBuffer.compare(FALSE_VALUE) == 0)
77       {
          std::cout << "EV" + ss.str()+": OFF" << std::endl;
79         output_vector[i] = LOW;
        }
81

      }
83

    }
85

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

```
87      EV1.setValue(output_vector[0]);
        EV2.setValue(output_vector[1]);
89      EV3.setValue(output_vector[2]);
        EV4.setValue(output_vector[3]);
91      EV5.setValue(output_vector[4]);
        EV6.setValue(output_vector[5]);
93      EV7.setValue(output_vector[6]);
        EV8.setValue(output_vector[7]);
95    }


97

    return 0;
99  }
```

Listing A.2: sensor_aquiring.cpp

```
1   #include<iostream>
    #include<fstream>
3   #include<unistd.h>
    #include<string>
5   #include<sstream>
    #include<stdlib.h>
7   #include<curl/curl.h>
    #include"../GPIO.h"
9   #include"../http.h"
    using namespace std;
11  #define LDR_PATH "/sys/bus/iio/devices/iio:device0/in_voltage"
    #define GPIO_PATH "/sys/class/gpio/"
13  #define OUTPUT false
    #define INPUT true
15  #define HIGH true
    #define LOW false
17

    float const CURRENT = 0.00068;
19  float const SLOPE_TEMPERATURE = 2;
    float const OFFSET_TEMPERATURE = 1870;
21  float const SLOPE_PH = -0.070;
    float const OFFSET_PH = 0.6;
23



25

    //void post_data_sensor(int type, float value){
27  //  CURL *curl;
    //  CURLcode res;
29  //  /* In windows, this will init the winsock stuff */
    //  curl_global_init(CURL_GLOBAL_ALL);
31  //
    //  /* get a curl handle */
33  //  curl = curl_easy_init();
```

```
//  if(curl) {
//    /* First set the URL that is about to receive our POST. This URL can
//       just as well be a https:// URL if that is what should receive the
//       data. */
//    curl_easy_setopt(curl, CURLOPT_URL, URL_POST.c_str());
//    /* Now specify the POST data */
//    std::ostringstream ss;
//    ss << value;
//    std::string s;
//    if ( type == 0)
//      s= POST_DATA_TEMPERATURE +(ss.str());
//    else
//      s= POST_DATA_PH +(ss.str());
//    curl_easy_setopt(curl, CURLOPT_POSTFIELDS, s.c_str());
//     /* Perform the request, res will get the return code */
//    res = curl_easy_perform(curl);
//    /* Check for errors */
//    if(res != CURLE_OK)
//      fprintf(stderr, "curl_easy_perform() failed: %s\n",
//              curl_easy_strerror(res));
//    /* always cleanup */
//    curl_easy_cleanup(curl);
//  }
//  curl_global_cleanup();
//}


class Sensor
{
  int number;


public:
  float m;
  float q;
  Sensor(int x): number(x){}
  void set_slope(float slope){
    m = slope;
  }
  void set_offset(float offset){
    q = offset;
  }
  float get_voltage_value(){
    stringstream ss;
    ss << LDR_PATH << number << "_raw";
    fstream fs;
    int adc_value;
    fs.open(ss.str().c_str(), fstream::in);
```

```
    fs >> adc_value;
    fs.close();
    float cur_voltage = adc_value * (1.80f/4096.0f);
    float actual_value;
    return cur_voltage;

  }
  float get_sensor_value(){
    float voltage_value = this->get_voltage_value();
    float sensor_value = (voltage_value - q)/m;
    return sensor_value;
  }
};

class Temperature_Sensor : public Sensor{
private:

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

class PH_Sensor : public Sensor{

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

  void post_ph_data(float data){
    post_data_sensor(1, data);
  }
};

int main(int argc, char* argv[])
{
```

```
  GPIO temperature_disable(45);
133 float slope_temperature;
  float slope_ph;
135 float offset_temperature;
  float offset_ph;
137
  if (argc == 5)
139 {
    slope_temperature = atoi(argv[1]);
141   offset_temperature = atoi(argv[2]);
    slope_ph = atoi(argv[3]);
143   offset_ph = atoi(argv[4]);
  }
145 else
  {
147   slope_temperature = SLOPE_TEMPERATURE;
    offset_temperature = OFFSET_TEMPERATURE;
149   slope_ph = SLOPE_PH;
    offset_ph = OFFSET_PH;
151 }

153 //set the pin GPIO_45 as output
  temperature_disable.setDirection(OUTPUT);
155 //Temporary disable the temperature
  temperature_disable.setValue(HIGH);
157
  Temperature_Sensor temperature_sensor(2);
159 PH_Sensor ph_sensor(1);
  Sensor offset_sensor(2);
161
  temperature_sensor.set_slope(slope_temperature);
163 temperature_sensor.set_offset(offset_temperature);
  ph_sensor.set_slope(slope_ph);
165 ph_sensor.set_offset(offset_ph);

167 int i= 0;
  for(i=0; i<100; i++)
169 {
    temperature_disable.setValue(LOW);
171   float ph = ph_sensor.get_sensor_value(offset_sensor);
    float temperature = temperature_sensor.get_sensor_value();
173   temperature_disable.setValue(HIGH);
    cout << "The voltage value is: " << temperature << " V." << endl;
175   temperature_sensor.post_temperature_data(temperature);
    ph_sensor.post_ph_data(ph);
177   usleep(100000);
  }
179 float temperature = temperature_sensor.get_voltage_value();
  cout << "The voltage value is: " << temperature << " V." << endl;
```

```
  return 0;
}
```

Listing A.3: recordVideo

```
#!/bin/bash

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



echo "Recording one minute of Video"
./capture -d /dev/video0 -F -o -c 300 > output.raw
avconv -f mjpeg -i output.raw -vcodec copy output.mp4
echo "Compressing Video"
ffmpeg -i output.mp4 -acodec mp2 Video.mp4
echo "Uploading the Video to the server"
curl -include --form Video.mp4 =@Video.mp4 -A "National Instruments LabVIEW" http://planar-contact-60
echo "Updating the flag for video"
curl  --data "name=video&status=on" http://planar-contact-601.appspot.com/add/electrovalve
curl  --data "name=glass&status=on" http://planar-contact-601.appspot.com/add/electrovalve
echo "Ploting the beating rate"
./compute_beating
echo "Uploading the beating plot"
curl -include --form Image.jpg=@Image.jpg -A "National Instruments LabVIEW" http://planar-contact-601
echo "finish"
```

Listing A.4: compute_beating.cpp

```


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

int main()
{
  CURL *curl;
  CURLcode res;
  struct stat file_info;
  FILE *file_to_send;
  file_to_send = fopen("Video.mp4", "rb");
```

```
20    double speed_upload, total_time;

22    if(!file_to_send)
      {
24      return 1;
      }
26    //to get the file size
      if(fstat(fileno(file_to_send), &file_info) != 0)
28    {
        return 1;
30    }

32    curl = curl_easy_init();
      if(curl)
34    {
        curl_easy_setopt(curl, CURLOPT_URL,
36          "http://planar-contact-601.appspot.com/video/submit");
        curl_easy_setopt(curl, CURLOPT_UPLOAD, 1L);
38      curl_easy_setopt(curl,CURLOPT_READDATA, file_to_send);
        curl_easy_setopt(curl,CURLOPT_INFILESIZE_LARGE,
40                  (curl_off_t)file_info.st_size);
        curl_easy_setopt(curl, CURLOPT_VERBOSE, 1L);
42      res = curl_easy_perform(curl);
        /* Check for errors */
44          if(res != CURLE_OK) {
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
46                  curl_easy_strerror(res));

48          }
            else {
50            /* now extract transfer info */
              curl_easy_getinfo(curl, CURLINFO_SPEED_UPLOAD, &speed_upload);
52            curl_easy_getinfo(curl, CURLINFO_TOTAL_TIME, &total_time);

54            fprintf(stderr, "Speed: %.3f bytes/sec during %.3f seconds\n",
                    speed_upload, total_time);
56

          }
58  //        /* always cleanup */
          curl_easy_cleanup(curl);
60    }

62      ofstream point_file;
        point_file.open("video_data.dat");
64      VideoCapture capture;//(0);   // capturing from file
        capture.open("output.mp4");
66



68
```

```
70
    // set any  properties in the VideoCapture object
72  capture.set(CV_CAP_PROP_FRAME_WIDTH,320);   // width pixels
    capture.set(CV_CAP_PROP_FRAME_HEIGHT,240);   // height pixels
74
    if(!capture.isOpened())
76  {
        cout << "Capture not open." << endl;
78  }

80  Mat frame, firstFrame, diffFrame;

82  capture >> frame; //save the first frame
    firstFrame = frame;
84  Scalar color_information = mean(firstFrame);
    float initial_point = (color_information[0] +color_information[1]+color_information[2])/3;
86
    int i;
88  for(i=1; i<capture.get(CV_CAP_PROP_FRAME_COUNT);i++)
    {
90      capture >> frame;          // capture the image to the frame
        if(frame.empty())
92      {
            cout << "Failed to capture an image" << endl;
94          return -1;
        }
96      absdiff(frame, firstFrame, diffFrame);
        color_information = mean(frame);
98      float red = color_information[0];
        float green = color_information[1];
100     float blue = color_information[2];

102     float mean_color = (red+green+blue)/3;

104      //qui poi chiamo la funzione che mi posta il punto nel db
         float temp = (i-1);
106     temp *= 0.033;

108  point_file << temp << '\t';
        point_file << mean_color << '\n';
110
     }
112


114  point_file.close();
    return 0;
116 }
```

Listing A.5: Pins Setting

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

/dts-v1/;
/plugin/;

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

       fragment@0 {
             target = <&am33xx_pinmux>;

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

      >;
      };
             };
       };

       fragment@1 {
    target = <&ocp>;
    __overlay__ {
      gpio_helper {
        compatible = "gpio-of-helper";
```

```
        status = "okay";
        pinctrl-names = "default";
        pinctrl-0 = <&eklab>;
      };
    };
  };
};
```

Listing A.6: GPIO.h

```
/*
 * @file GPIO.h
 * @author Derek Molloy
 * @version 0.1
 *
 * Created on: 29 Apr 2014
 * Copyright (c) 2014 Derek Molloy (www.derekmolloy.ie)
 * Made available for the book "Exploring BeagleBone"
 * See: www.exploringbeaglebone.com
 * Licensed under the EUPL V.1.1
 *
 * This Software is provided to You under the terms of the European
 * Union Public License (the "EUPL") version 1.1 as published by the
 * European Union. Any use of this Software, other than as authorized
 * under this License is strictly prohibited (to the extent such use
 * is covered by a right of the copyright holder of this Software).
 *
 * This Software is provided under the License on an "AS IS" basis and
 * without warranties of any kind concerning the Software, including
 * without limitation merchantability, fitness for a particular purpose,
 * absence of defects or errors, accuracy, and non-infringement of
 * intellectual property rights other than copyright. This disclaimer
 * of warranty is an essential part of the License and a condition for
 * the grant of any rights to this Software.
 *
 * For more details, see http://www.derekmolloy.ie/
 */

#ifndef GPIO_H_
#define GPIO_H_
#include<string>
#include<fstream>
using std::string;
using std::ofstream;

#define GPIO_PATH "/sys/class/gpio/"

namespace exploringBB {

```

```
typedef int (*CallbackType)(int);

/**
 * @class GPIO
 * @brief GPIO class for input and output functionality on a single GPIO pin
 */
class GPIO {
public:
  enum DIRECTION{ INPUT, OUTPUT };
  enum VALUE{ LOW=0, HIGH=1 };
  enum EDGE{ NONE, RISING, FALLING, BOTH };

private:
  int number;     /**< The GPIO number of the object */
  int debounceTime;   /**< The debounce time in milliseconds */
  string name;    /**< The name of the GPIO e.g. gpio50 */
  string path;      /**< The full path to the GPIO e.g. /sys/class/gpio/gpio50/ */

public:
  GPIO(int number);
  virtual int getNumber() { return number; } /**< Returns the GPIO number as an int. */

  // General Input and Output Settings
  virtual int setDirection(GPIO::DIRECTION);
  virtual GPIO::DIRECTION getDirection();
  virtual int setValue(GPIO::VALUE);
  virtual int toggleOutput();
  virtual GPIO::VALUE getValue();
  virtual int setActiveLow(bool isLow=true);  //low=1, high=0
  virtual int setActiveHigh(); //default
  //software debounce input (ms) - default 0
  virtual void setDebounceTime(int time) { this->debounceTime = time; }

  // Advanced OUTPUT: Faster write by keeping the stream alive (~20X)
  virtual int streamOpen();
  virtual int streamWrite(GPIO::VALUE);
  virtual int streamClose();

  virtual int toggleOutput(int time); //threaded invert output every X ms.
  virtual int toggleOutput(int numberOfTimes, int time);
  virtual void changeToggleTime(int time) { this->togglePeriod = time; }
  virtual void toggleCancel() { this->threadRunning = false; }

  // Advanced INPUT: Detect input edges; threaded and non-threaded
  virtual int setEdgeType(GPIO::EDGE);
  virtual GPIO::EDGE getEdgeType();
  virtual int waitForEdge(); // waits until button is pressed
  virtual int waitForEdge(CallbackType callback); // threaded with callback
  virtual void waitForEdgeCancel() { this->threadRunning = false; }
```

```
89
   virtual ~GPIO();  //destructor will unexport the pin
91
private:
93   //int write(string path, string filename, string value);
   //int write(string path, string filename, int value);
95   //string read(string path, string filename);
   int exportGPIO();
97   int unexportGPIO();
   ofstream stream;
99   pthread_t thread;
   CallbackType callbackFunction;
101   bool threadRunning;
   int togglePeriod;  //default 100ms
103   int toggleNumber;  //default -1 (infinite)
   friend void* threadedPoll(void *value);
105   friend void* threadedToggle(void *value);
};
107
void* threadedPoll(void *value);
109 void* threadedToggle(void *value);

111 } /* namespace exploringBB */

113 #endif /* GPIO_H_ */
```

Listing A.7: GPIO.cpp

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

```
 * of warranty is an essential part of the License and a condition for
 * the grant of any rights to this Software.
 *
 * For more details, see http://www.derekmolloy.ie/
 */

#include "GPIO.h"
#include "util.h"
#include<iostream>
#include<fstream>
#include<string>
#include<sstream>
#include<cstdlib>
#include<cstdio>
#include<fcntl.h>
#include<unistd.h>
#include<sys/epoll.h>
#include<pthread.h>
using namespace std;

namespace exploringBB {
/**
 * The constructor will set up the states and export the pin.
 * @param number The GPIO number to be exported
 */
GPIO::GPIO(int number) {
  this->number = number;
  this->debounceTime = 0;
  this->togglePeriod=100;
  this->toggleNumber=-1; //infinite number
  this->callbackFunction = NULL;
  this->thread = NULL;
  this->threadRunning = false;

  ostringstream s;
  s << "gpio" << number;
  this->name = string(s.str());
  this->path = GPIO_PATH + this->name + "/";
  this->exportGPIO();
  // need to give Linux time to set up the sysfs structure
  usleep(250000); // 250ms delay
}
/**
 * Private write method that writes a single string value to a file in the path provided
 * @param Path The sysfs path of the file to be modified
 * @param Filename The file to be written to in that path
 * @param String value The value to be written to the file
 * @return
 */
```

```
71
   /* int GPIO::write(string path, string filename, string value){
73     ofstream fs;
       fs.open((path + filename).c_str());
75     if (!fs.is_open()){
         perror("GPIO: write failed to open file ");
77       return -1;
       }
79     fs << value;
       fs.close();
81     return 0;
   }*/
83
   /*string GPIO::read(string path, string filename){
85     ifstream fs;
       fs.open((path + filename).c_str());
87     if (!fs.is_open()){
         perror("GPIO: read failed to open file ");
89      }
       string input;
91     getline(fs,input);
       fs.close();
93     return input;
   }*/
95
   /**
97  * Private write method that writes a single int value to a file in the path provided
    * @param Path The sysfs path of the file to be modified
99  * @param Filename The file to be written to in that path
    * @param int value The int value to be written to the file
101 * @return
    */
103 /*int GPIO::write(string path, string filename, int value){
       stringstream s;
105    s << value;
       return write(path,filename,s.str());
107 }*/

109 /**
     * Private method to export the GPIO
111  * @return int that describes if the operation fails
     */
113 int GPIO::exportGPIO(){
       return write(GPIO_PATH, "export", this->number);
115 }

117 int GPIO::unexportGPIO(){
       return write(GPIO_PATH, "unexport", this->number);
119 }
```

```
121 int GPIO::setDirection(GPIO::DIRECTION dir){
    switch(dir){
123 case INPUT: return write(this->path, "direction", "in");
      break;
125 case OUTPUT:return write(this->path, "direction", "out");
      break;
127 }
    return -1;
129 }

131 int GPIO::setValue(GPIO::VALUE value){
    switch(value){
133 case HIGH: return write(this->path, "value", "1");
      break;
135 case LOW: return write(this->path, "value", "0");
      break;
137 }
    return -1;
139 }

141 int GPIO::setEdgeType(GPIO::EDGE value){
    switch(value){
143 case NONE: return write(this->path, "edge", "none");
      break;
145 case RISING: return write(this->path, "edge", "rising");
      break;
147 case FALLING: return write(this->path, "edge", "falling");
      break;
149 case BOTH: return write(this->path, "edge", "both");
      break;
151 }
    return -1;
153 }

155 int GPIO::setActiveLow(bool isLow){
    if(isLow) return write(this->path, "active_low", "1");
157 else return write(this->path, "active_low", "0");
}
159

int GPIO::setActiveHigh(){
161 return this->setActiveLow(false);
}
163

GPIO::VALUE GPIO::getValue(){
165   string input = read(this->path, "value");
  if (input == "0") return LOW;
167   else return HIGH;
}
```

```
169
GPIO::DIRECTION GPIO::getDirection(){
171   string input = read(this->path, "direction");
   if (input == "in") return INPUT;
173   else return OUTPUT;
}
175
GPIO::EDGE GPIO::getEdgeType(){
177   string input = read(this->path, "edge");
   if (input == "rising") return RISING;
179   else if (input == "falling") return FALLING;
   else if (input == "both") return BOTH;
181   else return NONE;
}
183
int GPIO::streamOpen(){
185   stream.open((path + "value").c_str());
   return 0;
187 }
int GPIO::streamWrite(GPIO::VALUE value){
189   stream << value << std::flush;
   return 0;
191 }
int GPIO::streamClose(){
193   stream.close();
   return 0;
195 }

197 int GPIO::toggleOutput(){
   this->setDirection(OUTPUT);
199   if ((bool) this->getValue()) this->setValue(LOW);
   else this->setValue(HIGH);
201     return 0;
}
203
int GPIO::toggleOutput(int time){ return this->toggleOutput(-1, time); }
205 int GPIO::toggleOutput(int numberOfTimes, int time){
   this->setDirection(OUTPUT);
207   this->toggleNumber = numberOfTimes;
   this->togglePeriod = time;
209   this->threadRunning = true;
     if(pthread_create(&this->thread, NULL, &threadedToggle, static_cast<void*>(this))){
211     perror("GPIO: Failed to create the toggle thread");
       this->threadRunning = false;
213     return -1;
     }
215   return 0;
}
217
```

```
// This thread function is a friend function of the class
void* threadedToggle(void *value){
  GPIO *gpio = static_cast<GPIO*>(value);
  bool isHigh = (bool) gpio->getValue(); //find current value
  while(gpio->threadRunning){
    if (isHigh) gpio->setValue(GPIO::HIGH);
    else gpio->setValue(GPIO::LOW);
    usleep(gpio->togglePeriod * 500);
    isHigh=!isHigh;
    if(gpio->toggleNumber>0) gpio->toggleNumber--;
    if(gpio->toggleNumber==0) gpio->threadRunning=false;
  }
  return 0;
}

// Blocking Poll - based on the epoll socket code in the epoll man page
int GPIO::waitForEdge(){
  this->setDirection(INPUT); // must be an input pin to poll its value
  int fd, i, epollfd, count=0;
  struct epoll_event ev;
  epollfd = epoll_create(1);
    if (epollfd == -1) {
     perror("GPIO: Failed to create epollfd");
     return -1;
    }
    if ((fd = open((this->path + "value").c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
       perror("GPIO: Failed to open file");
       return -1;
    }

    //ev.events = read operation | edge triggered | urgent data
    ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
    ev.data.fd = fd;  // attach the file file descriptor

    //Register the file descriptor on the epoll instance, see: man epoll_ctl
    if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
       perror("GPIO: Failed to add control interface");
       return -1;
    }
  while(count<=1){  // ignore the first trigger
    i = epoll_wait(epollfd, &ev, 1, -1);
    if (i==-1){
      perror("GPIO: Poll Wait fail");
      count=5; // terminate loop
    }
    else {
      count++; // count the triggers up
    }
  }
```

```
267     close(fd);
        if (count==5) return -1;
269   return 0;
}
271
// This thread function is a friend function of the class
273 void* threadedPoll(void *value){
  GPIO *gpio = static_cast<GPIO*>(value);
275   while(gpio->threadRunning){
      gpio->callbackFunction(gpio->waitForEdge());
277     usleep(gpio->debounceTime * 1000);
    }
279   return 0;
}
281
int GPIO::waitForEdge(CallbackType callback){
283   this->threadRunning = true;
    this->callbackFunction = callback;
285     // create the thread, pass the reference, address of the function and data
      if(pthread_create(&this->thread, NULL, &threadedPoll, static_cast<void*>(this))){
287       perror("GPIO: Failed to create the poll thread");
        this->threadRunning = false;
289       return -1;
      }
291     return 0;
}
293
GPIO::~GPIO() {
295   this->unexportGPIO();
}
297
} /* namespace exploringBB */
```

Listing A.8: HTTP.h

```
#ifndef HTTP
2 #define HTTP
#include<iostream>
4 #include<fstream>
#include<unistd.h>
6 #include<string>
#include<sstream>
8 #include<stdlib.h>
#include<curl/curl.h>
10
std::string const URL_POST = "http://planar-contact-601.appspot.com/sensor_values";
12 std::string const POST_DATA_TEMPERATURE =  "sensor=temperature&value=";
std::string const POST_DATA_PH =  "sensor=pH&value=";
14 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());
     /* Perform the request, res will get the return code */
28        res = curl_easy_perform(curl);
          /* Check for errors */
30        if(res != CURLE_OK)
            fprintf(stderr, "curl_easy_perform() failed: %s\n",
32                  curl_easy_strerror(res));
    /* always cleanup */
34        curl_easy_cleanup(curl);
        }
36      curl_global_cleanup();
}
```

A.2 VIDEO STORING SOFTWARE

Listing A.10: main.cpp

```
//#include <QCoreApplication>
//#include <QTimer>
//#include "downloader.h"
//#include <QDebug>

//int main(int argc, char *argv[])
//{
//    QCoreApplication a(argc, argv);

//    QDateTime data;
//    data = QDateTime::currentDateTime();
//    QString name = "./"+data.toString("yyyy-MM-dd-HH:mm:ss")+".mp4";
//    Downloader d(0,name);
//    d.doDownload();
////    for(;;)
////    {
////        qDebug() << "Ciao";
////        if(d.isTime())
////        {
////            d.sendFlag();
////            d.doDownload();
////        }
////    }
//    return a.exec();
//}
#include <QCoreApplication>
#include "downloader.h"
#include "mytimer.h"
#include <QDebug>
#include <windows.h> // for Sleep


int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);


    MyTimer t;

    //bool dummy = d.isTime();
//    while(1)
//    {
//        d.checkFlag();
```

```
45
//         if( d.getStatus())
47 //         {
//             d.resetFlag();
49 //             d.doDownload();
//         }
51 //         Sleep(20000);
//     }
53



55


57     return a.exec();
}
```

Listing A.11: VideoStoring.pro

```
#-------------------------------------------------
2 #
# Project created by QtCreator 2015-02-17T21:56:37
4 #
#-------------------------------------------------
6

QT       += core
8 QT       += network

10 QT       -= gui

12 #RC_FILE = myapp.rc

14 TARGET = VideoStoring
CONFIG   += console
16 CONFIG   -= app_bundle

18 TEMPLATE = app


20

SOURCES += main.cpp \
22     downloader.cpp \
    mytimer.cpp
24

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

Listing A.12: mytimer.h

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

```
3
#include <QObject>
5 #include <QtCore>
#include <downloader.h>
7
class MyTimer : public QObject
9 {
    Q_OBJECT
11 public:
    MyTimer();
13     ~MyTimer();
    QTimer *timer;
15
17 public slots:
    void timerSlot();
19 private:
    Downloader d;
21 };

23 #endif // MYTIMER_H
```

Listing A.13: mytimer.cpp

```
1 #include "mytimer.h"
#include <QtCore>
3 #include <QDebug>
MyTimer::MyTimer()
5 {
    timer = new QTimer(this);
7     connect(timer,SIGNAL(timeout()), this, SLOT(timerSlot()));

9     timer->start(10000);

11 }

13 MyTimer::~MyTimer()
{
15
}
17



19


21 void MyTimer::timerSlot(){
    d.checkFlag();
23     if( d.getStatus())
    {
25         d.resetFlag();
```

```
        d.doDownload();
27    }

29 }
```

Listing A.14: downloader.h

```
#ifndef DOWNLOADER_H
2 #define DOWNLOADER_H

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

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

20     void doDownload();
       void checkFlag();
22     bool getStatus();
       void resetFlag();
24
signals:
26
public slots:
28     void replyFinished (QNetworkReply *reply);
       void checkStatus(QNetworkReply *replay);
30
private:
32    QNetworkAccessManager *manager;
      QNetworkAccessManager *manager2;
34    QDateTime data;
      QString name;
36    bool flag_ready;
      bool status;
38


40 };

42 #endif // DOWNLOADER_H
```

Listing A.15: downloader.cpp

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

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

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

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

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

void Downloader::replyFinished (QNetworkReply *reply)
{
    if(reply->error())
    {
        qDebug() << "ERROR!";
        qDebug() << reply->errorString();
    }
    else
    {
        qDebug() << reply->header(QNetworkRequest::ContentTypeHeader).toString();
        qDebug() << reply->header(QNetworkRequest::LastModifiedHeader).toDateTime().toString();
        qDebug() << reply->header(QNetworkRequest::ContentLengthHeader).toULongLong();
        qDebug() << reply->attribute(QNetworkRequest::HttpStatusCodeAttribute).toInt();
        qDebug() << reply->attribute(QNetworkRequest::HttpReasonPhraseAttribute).toString();

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

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

    reply->deleteLater();
}
```

```
50
void Downloader::checkStatus(QNetworkReply *replay)
52 {
    QString result(replay->readAll());
54     int compare = QString::compare(result, FLAG_ON);
    if(compare == 0)
56     {
        status = true;
58     }
    else
60     {
        status = false;
62     }
    qDebug() << status;
64     flag_ready = true;

66 }
void Downloader::checkFlag()
68 {
    manager = new QNetworkAccessManager(this);
70     QNetworkReply *reply = manager->get(QNetworkRequest(QUrl("http://planar-contact-601.appspot.com/
    QEventLoop loop;
72     connect(reply, SIGNAL(finished()), &loop, SLOT(quit()));

74     loop.exec();
    qDebug() << "I'm looking for a new video";
76     QString result(reply->readAll());
    int compare = QString::compare(result, FLAG_ON);
78     if(compare == 0)
    {
80         status = true;
        qDebug() << "Found it";
82     }
    else
84     {
        status = false;
86         qDebug() << "No new video available";
    }
88

}
90

bool Downloader::getStatus()
92 {
    return status;
94 }

96 void Downloader::resetFlag()
{
```

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

A.3 GOOGLE APP ENGINE

Listing A.16: Main Script of Server

```
1  # 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
   from flask import make_response
9
   from models import MESSAGES
11 from google.appengine.ext import ndb
   from google.appengine.api import memcache
13 import logging
   import datetime
15 import cloudstorage
   import json
17
   key_list = []
19
   NUMBER_OF_ELECTROVALVES = 8
21 SECONDS_BETWEEN_UPDATES = 60
   SENSOR_TIMEOUT_IN_SECONDS = 300
23 ELECTROVALVES_TIME_OUT = 1000
   BUCKET_NAME = "/planar-contact-601.appspot.com/"
25 ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'mp4'}
   UPDATE_FLAG = 'database_updated_recently'
27


29 class DataPoint(ndb.Model):
       value = ndb.FloatProperty()
31     date = ndb.DateTimeProperty(auto_now_add=True)

33     @classmethod
```

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

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

41
class Sensor(ndb.Model):
43  @classmethod
    def sensor_list(cls):
45      return cls.query()

47
def update():
49  # Check to see if we have updated recently
    update_flag = memcache.get(UPDATE_FLAG)
51  if update_flag is not None:
        return
53  memcache.add(UPDATE_FLAG, 'YES', SECONDS_BETWEEN_UPDATES)
    # Delete old data points
55  now = datetime.datetime.now()
    count = 0
57  for point in DataPoint.oldest_points().iter():
        delta = (now - point.date).total_seconds()
59      if delta > 3600:
            point.key.delete()
61          count += 1
        else:
63          break
    logging.info("Deleted {} old data points".format(count))
65  # Loop through all of the sensors
    sensor_names = []
67  for sensor in Sensor.sensor_list().iter():
        # Store current values of each sensor
69      sensor_value = memcache.get(sensor.key.id() + "_value")
        if sensor_value is not None:
71          point = DataPoint(parent=sensor.key, value=sensor_value)
            point.put()
73      # Delete sensors that don't have any data
        curr_points = DataPoint.points_for_sensor(sensor.key)
75      if curr_points.count() == 0 and sensor_value is None:
            logging.info("Deleting sensor {}".format(sensor.key.id()))
77          sensor.key.delete()
        else:
79          sensor_names.append(sensor.key.id())
    # Store a list of sensor names in memcache
81  memcache.set("sensor_names", ",".join(sensor_names))
```

```
83
@app.route('/', methods=['POST'])
85 @app.route('/sensor_values', methods=['GET', 'POST'])
def process_values():
87     if request.method == 'GET':
           sensor_names = memcache.get('sensor_names')
89         if sensor_names is None:
               sensor_names = ",".join([sensor.key.id() for sensor in Sensor.sensor_list()])
91             memcache.set('sensor_names', sensor_names)
           sensor_names = sensor_names.split(",")
93         sensor_values = {}
           for sensor_name in sensor_names:
95             sensor_value = memcache.get(sensor_name + '_value')
               if sensor_value is not None:
97                 sensor_values[sensor_name] = sensor_value
           return json.dumps(sensor_values)
99     elif request.method == 'POST':
           update()
101        logging.info(request.values)
           sensor_name = request.form.get('sensor')
103        sensor_value = float(request.form.get('value'))
           sensor_names = memcache.get('sensor_names')
105        if sensor_names is not None and sensor_name not in sensor_names:
               sensor = Sensor()
107            sensor.key = ndb.Key('Sensor', sensor_name)
               sensor.put()
109            sensor_names = sensor_name if sensor_names == "" else sensor_names + "," + sensor_name
               memcache.set('sensor_names', sensor_names)
111        memcache.set(sensor_name + '_value', sensor_value, SENSOR_TIMEOUT_IN_SECONDS)
           return "Success\n"
113


115 @app.route('/sensor_names', methods=['GET'])
def print_names():
117     return json.dumps([sensor.key.id() for sensor in Sensor.sensor_list()])


119
@app.route('/graphing_data', methods=['GET'])
121 def graphing_data():
    last_timestamp = request.args.get('last_timestamp')
123     sensor_name = request.args.get('sensor')
    if last_timestamp is None:
125         last_timestamp = 0
    else:
127         last_timestamp = float(last_timestamp)
    points = DataPoint.points_for_sensor(ndb.Key('Sensor', sensor_name))
129     epoch = datetime.datetime(1970, 1, 1)
    res = []
131     for point in points.iter():
```

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


@app.route('/clear', methods=['GET'])
def clear_data():
    points = DataPoint.query()
    for point in points:
        point.key.delete()
    for sensor in Sensor.sensor_list():
        sensor.key.delete()
    return "Success"


@app.route('/picture/submit', methods=['GET', 'POST'])
def set_picture():
    if request.method == 'POST':

        _file = request.files['Image.jpg']

        if _file:
            cloud_file = cloudstorage.open(BUCKET_NAME + "microscope_image." + _file.filename.rsplit
                                           mode='w', content_type="image/jpeg")
            _file.save(cloud_file)
            cloud_file.close()
            return "Success"
    else:
        return '''
        <!doctype html>
        <title>Upload microscope file V.1</title>
        <h1>Upload microscope file</h1>
        <form method="POST"
        action=""
        role="form"
        enctype="multipart/form-data">

        <p><input type=file name=Image.jpg>
                <input type=submit value=Upload>
        </form>
        '''


@app.route('/picture/view', methods=['GET'])
```

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


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


219


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

```
            cloud_file.close()
231             response.mimetype = _file.content_type
                return response
233     return "No file found"


235
@app.errorhandler(404)
237 def page_not_found(e):
    """Return a custom 404 error."""
239     return 'Sorry, nothing at this URL.', 404


241
class Electrovalve(ndb.Model):
243     # name = ndb.StringProperty()
    # status = ndb.BooleanProperty()  # if the status is true, the Electrovalve is on
245     # date = ndb.DateTimeProperty(auto_now_add=True)
    #
247     # def __init__(self, name, status, **kwds):
    #     super(Electrovalve, self).__init__(**kwds)
249     #     self.name = name
    #     self.status = status
251     #
    # @classmethod
253     # def get_status(cls):
    #     return cls.status
255     #
    # @classmethod
257     # def get_ev_by_name(cls, ev_name):
    #     return cls.query(ancestor=ev_name).fetch
259
    @classmethod
261     def ev_list(cls):
        return cls.query()
263


265 # @app.route('/add/<key>/<message>')
# def update_message(key, message):
267 #     if key and message:
#
269 #         if message == 'on' or message == 'On' or message == 'ON':
#             MESSAGES[key] = True
271 #
#         else:
273 #             MESSAGES[key] = False
#
275 #             # temp = Electrovalve(name=key, status=message)
#             # temp.put()
277 #     return "%s Updated" % key
```

```
279
@app.route('/electrovalves/<name>', methods=['GET'])
281 def electrovalve(name):
    ev = memcache.get(name);
283     if ev is None:
        return ' %r ' % MESSAGES[name] or "%s not found!" % name
285     else:
        return ' %r ' % memcache.get(name)
287     #     ev_values = {}
    #    # return "boia"
289     # #     ev_names = ",".join([ev.key.id() for ev in Electrovalve.ev_list()])
    # #     memcache.set('ev_names', ev_names)
291     # # ev_names = ev_names.split(",")
    #     ev_values = {}
293     #     for ev_name in ev_names:
    #         ev_value = memcache.get(ev_name + '_value')
295     #
    #         if ev_value is not None:
297     #             ev_values[ev_name] = ev_value
    #
299     #     return ' %r ' % ev_values[name]

301     #
    # # ev_name =
303     # ev_key = ndb.Key("EV", name or "*notitle")
    # ev = Electrovalve.query(name)
305    # return '%r' % ev.st

307
@app.route('/add/electrovalve', methods=['POST'])
309 def add_electrovalve():
    key = request.form.get('name')
311     message = request.form.get('status')
    if key and message:
313         if message == 'on' or message == 'On' or message == 'ON':
            MESSAGES[key] = True
315             memcache.set(key, True)

317         else:
            MESSAGES[key] = False
319             memcache.set(key, False)

321        # ev.put()
    return '%r' % memcache.get(key)
323

325 @app.route('/glass')
def glass():
327     return "Buso e‘ qui"
```

```
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;

3  import com.google.android.glass.timeline.LiveCard;
   import com.google.android.glass.timeline.LiveCard.PublishMode;
5
   import android.app.PendingIntent;
7  import android.app.Service;
   import android.content.Context;
9  import android.content.Intent;
   import android.graphics.Bitmap;
11 import android.graphics.BitmapFactory;
   import android.net.ConnectivityManager;
13 import android.net.NetworkInfo;
   import android.os.AsyncTask;
15 import android.os.Environment;
   import android.os.Handler;
17 import android.os.HandlerThread;
   import android.os.IBinder;
19 import android.util.Log;
   import android.view.animation.Animation;
21
   import com.google.android.glass.timeline.LiveCard;
23 import com.googlecode.charts4j.AxisLabelsFactory;
   import com.googlecode.charts4j.Data;
25 import com.googlecode.charts4j.GCharts;
   import com.googlecode.charts4j.LineChart;
27 import com.googlecode.charts4j.Plot;
   import com.googlecode.charts4j.Plots;
29 import com.jjoe64.graphview.GraphView;
   import com.jjoe64.graphview.GraphViewSeries;
31
   import org.json.JSONArray;
```

```
33 import org.json.JSONException;
   import org.json.JSONObject;
35 import org.json.JSONTokener;

37 import com.jjoe64.graphview.GraphView;
   import com.jjoe64.graphview.LineGraphView;
39


41 import java.io.BufferedInputStream;
   import java.io.File;
43 import java.io.FileOutputStream;
   import java.io.IOException;
45 import java.io.InputStream;
   import java.net.HttpURLConnection;
47 import java.net.URL;
   import java.net.URLConnection;
49 import java.nio.channels.FileLock;
   import java.util.ArrayList;
51 import java.util.Collections;
   import java.util.HashMap;
53 import java.util.Map;


55
   /**
57  * Service owning the LiveCard living in the timeline.
    */
59 public class MainService extends Service {
       /** TAG associated to the LiveCard */
61     private static final String LIVE_CARD_TAG = "BWH interface";
       /** TAG associated to the Menu view   */
63     private static final String MENU_TAG = "Menu";
       /** TAG associated to the Temperature view  */
65     private static final String TEMPERATURE_TAG = "Temperature";
       /** TAG associated to the PH view  */
67     private static final String PH_TAG = "pH";
       /** TAG associated to the Video view   */
69     private static final String VIDEO_TAG = "Video";
       /** TAG associated to the Beating view   */
71     private static final String BEATING_TAG = "Beating";

73     private Bitmap bmp;

75     /** URL where the image is stored   */
       private static final String URL_IMAGE = "http://planar-contact-601.appspot.com/picture/view";
77



79
       /** Runnable which describes the task to download the Beating's graph */
81     private final ImageDownloader mImageDownloader = new ImageDownloader(URL_IMAGE, this);
```

```
    /** Action is an enumerate used to implement switch-case for the extra text appended to the inte
83  private static enum Action
    {
85      Menu, Temperature, pH, Video, Beating
    }
87

89  private AppDrawer mCallback;

91  private LiveCard mLiveCard;

93  /** HandlerThread used to launch a background thread that manages the Data updating   */
    private HandlerThread mHandlerThread;
95  /** Handler used to launch a background thread that manages the Data updating   */
    private Handler mHandler;
97

    /** INT associated to the Menu view request  */
99  private static final int MENU = 0;
    /** INT associated to the PH view request  */
101 private static final int PH = 1;
    /** INT associated to the Menu view request  */
103 private static final int TEMPERATURE = 2;
    /** INT associated to the Video view request  */
105 private static final int VIDEO = 3;
    /** INT associated to the Beating view request  */
107 private static final int BEATING = 4;

109 /** updating data period (in ms)  */
    private static final long DATA_UPDATE_DELAY_MILLIS = 500;
111 /** updating graph period (in ms)  */
    private static final long GRAPH_UPDATE_DELAY_MILLIS = 500;
113 /** updating video period (in ms)  */
    private static final long FRAME_TIME_MILLIS = 100;
115 private static final long VIDEO_UPDATE_DELAY_MILLIS = 60*1000; //time for video

117 /** Runnable which describes the task to update the pH and Temperature sensors values   */
    private final UpdateSensorValuesRunnable mUpdateSensorValuesRunnable = new UpdateSensorValuesRun
119 /** Runnable which describes the task to compute the pH and Temperature graph (starting from the
    private final UpdateSensorGraphsRunnable mUpdateSensorGraphsRunnable = new UpdateSensorGraphsRun
121
    private final UpdateMicroscopeVideoRunnable mUpdateMicroscopeVideoRunnable = new UpdateMicroscop
123
    private static final String VIDEO_FILE_NAME = Environment.getExternalStorageDirectory()+"/micros
125 private static final String TEMP_VIDEO_FILE_NAME = Environment.getExternalStorageDirectory()+"/t

127 /** String array for the sensors (PH and Temperature) */
    private String[] mSensors = new String[]{PH_TAG, TEMPERATURE_TAG};
129

    /*                          Hash table used for creating the graphs
```

```
131     /** Hash table in which the sensors values are stored  */
        private Map<String,Double> mCurrentSensorValues;
133     /** Hash table in which the graphs points are stored  */
        private Map<String, ArrayList<DataPoint>> mSensorGraphData;
135     /** Hash table in which the sensors graphs are stored */
        private Map<String,Bitmap> mCurrentSensorGraphs;
137
        private Map<String,Double> mSensorAverage;
139
    //  private ConnectivityManager cm = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SE
141


143     @Override
        public IBinder onBind(Intent intent) {
145         return null;
        }
147
        @Override
149     public int onStartCommand(Intent intent, int flags, int startId) {
            if (mLiveCard == null) {
151             AppManager.getInstance().setState(MENU);
                mLiveCard = new LiveCard(this, LIVE_CARD_TAG);
153
                // Keep track of the callback to remove it before unpublishing.
155             mCallback = new AppDrawer(this);
                mLiveCard.setDirectRenderingEnabled(true).getSurfaceHolder().addCallback(mCallback);
157
                Intent menuIntent = new Intent(this, MenuActivity.class);
159             menuIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
                mLiveCard.setAction(PendingIntent.getActivity(this, 0, menuIntent, 0));
161             mLiveCard.attach(this);
                mLiveCard.publish(PublishMode.REVEAL);
163


165              /*                  Launch the task to update the data in another thread
                mHandlerThread = new HandlerThread("myHandlerThread");
167             mHandlerThread.start();
                mHandler = new Handler(mHandlerThread.getLooper());
169             DataTask task = new DataTask();
                task.execute(this);
171         } else {
                if(intent.getStringExtra(Intent.EXTRA_TEXT)!= null) {
173                 Action action = Action.valueOf(intent.getStringExtra(Intent.EXTRA_TEXT));
                    switch (action) {
175                     case Menu:
                            Log.i(LIVE_CARD_TAG, "State = Menu");
177                         AppManager.getInstance().setState(MENU);
                            break;
179                     case Beating:
```

```
                        Log.i(LIVE_CARD_TAG, "State = Beating");
                        AppManager.getInstance().setState(BEATING);
                        break;
                    case pH:
                        Log.i(LIVE_CARD_TAG, "State = pH");
                        AppManager.getInstance().setState(PH);
                        break;
                    case Temperature:
                        Log.i(LIVE_CARD_TAG, "State = Temperature");
                        AppManager.getInstance().setState(TEMPERATURE);
                        break;
                    case Video:
                        Log.i(LIVE_CARD_TAG, "State = Video");
                        AppManager.getInstance().setState(VIDEO);
                        break;
                    default:
                        mLiveCard.navigate();
                        break;
                }
            }
            else
            {
                mLiveCard.navigate();
            }
        }

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

    @Override
    public void onDestroy() {
        // Stop the Task which update the beating image
        if(!mImageDownloader.isStopped())
        {
            mImageDownloader.setIsStopped(true);
            Log.i(BEATING_TAG,"Removed Task");
        }
        // Stop the Task which update the sensors Graphs
        if(!mUpdateSensorGraphsRunnable.isStopped())
        {
            mUpdateSensorGraphsRunnable.setStop(true);
            Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Graphs");
        }
        // Stop the Task which update the sensors value
        if(!mUpdateSensorValuesRunnable.isStopped())
        {
```

```
229             mUpdateSensorValuesRunnable.setStop(true);
                Log.i(PH_TAG + " " + TEMPERATURE_TAG,"Removed Task of Values");
231         }
            if (!mUpdateMicroscopeVideoRunnable.isStopped())
233         {
                mUpdateMicroscopeVideoRunnable.setStop(true);
235             Log.i(VIDEO_TAG, "Removed Task of Uploading values");
            }
237         if (mLiveCard != null && mLiveCard.isPublished()) {
                mLiveCard.unpublish();
239             mLiveCard = null;
            }
241         super.onDestroy();
    }
243
    /**
245  * Asynchronous Task for downloading the graphs and video in background
     */
247 private class DataTask extends AsyncTask<MainService,Void,Void>
    {
249
        @Override
251     protected Void doInBackground(MainService... params) {

253         Log.i(LIVE_CARD_TAG, "Loading initial data");

255         // create the three hash tables
            mCurrentSensorValues = new HashMap<String, Double>();
257         mSensorGraphData = new HashMap<String, ArrayList<DataPoint>>();
            mCurrentSensorGraphs = new HashMap<String, Bitmap>();
259         mSensorAverage = new HashMap<String, Double>();

261         // initializes each hash map with dummy values
            for (String mSensor : mSensors)
263         {
                mCurrentSensorValues.put(mSensor, 0.0);
265             mSensorAverage.put(mSensor, 0.0);
                mSensorGraphData.put(mSensor, new ArrayList<DataPoint>());
267             mCurrentSensorGraphs.put(mSensor, null);
            }
269
            Log.i(LIVE_CARD_TAG, "Download the data");
271
            //uploads the value captured by PH and Temperature sensors
273         //NetworkInfo ni = cm.getActiveNetworkInfo();
            //if(ni!= null) {
275
               // if (ni.isConnected()) {
277                 mUpdateSensorValuesRunnable.run();
```

```
                    // give the hash table to the Callback
279                 // mCallback.setSensorValues(mCurrentSensorValues);
                    // compute the graphs with previous values and give them to the AppDrawer
281                 mUpdateSensorGraphsRunnable.run();

283                 // give the hash table to the Callback
                    // mCallback.setSensorGraphs(mCurrentSensorGraphs);
285                 // download the beating image and give this to the AppDrawer
                    mImageDownloader.run();
287
            mUpdateMicroscopeVideoRunnable.run();
289         //    }
    //        }
291


293


295         return null;
        }
297 }

299 /**
     * This runnable updates the sensors values taking them from the google engine
301  */
    private class UpdateSensorValuesRunnable implements Runnable
303 {
        private boolean mIsStopped = false;
305
        /** It implements the task of runnable
307      *
         * @see UpdateSensorValuesRunnable
309      */
        @Override
311     public void run()
        {
313         if (!isStopped())
            {
315             /** JavaScript object in which the sensor values are temporary stored */
                JSONObject values = getSensorValues();
317             if (values != null)
                {
319                 for (String mSensor : mSensors)
                    {
321                     try
                        {
323                         // update the hash table of sensor values
                            mCurrentSensorValues.put(mSensor, values.getDouble(mSensor));
325
                        }
```

```
327                         catch (JSONException ignored)
                            {}
329                     }
                }
331
                // restart the Runnable after a given amount of time
333             mHandler.postDelayed(mUpdateSensorValuesRunnable, DATA_UPDATE_DELAY_MILLIS);
            }
335     }


337


339     /** It is the getter function that shows the status of runnable
         *
341      * @return mIsStopped, true if the runnable is stopped, false otherwise
         */
343     public boolean isStopped()
        {
345         return mIsStopped;
        }
347
        /** It is the setter that allows to stop the runnable
349      *
         * @param isStopped, true if the user wishes to stop the runnable, false otherwise
351      */
        public void setStop(boolean isStopped)
353     {
            this.mIsStopped = isStopped;
355     }
    }
357
    /** It gets the values of all of the sensors
359  *
     * @return JSONOObject which contains all the values of sensors
361  */
    private JSONObject getSensorValues()
363 {
        try
365     {
            String values = getURL("http://planar-contact-601.appspot.com/sensor_values");
367         // return a JSONObject constructed by JSONTokener, which takes a source string and extrac
            return new JSONObject(new JSONTokener(values));
369     }
        catch (Exception e)
371     {
            Log.e(LIVE_CARD_TAG,"Failed to get sensor values",e);
373         return null;
        }
375 }
```

```
377
    // Get the new data points that we will need to graph
379
    /** This function computes the points that have to be plotted
381  *
     * @param sensor , the name of sensor (pH or Temperature)
383  * @param last_timestamp , the previous value of timestamp
     * @return JSONArray, return a list of values that corresponds to the points that have to be plo
385  */
    private JSONArray getDataPoints(String sensor, double last_timestamp)
387 {
        try
389     {
            String url = "http://planar-contact-601.appspot.com/graphing_data?sensor=" + sensor + "&
391         String values = getURL(url);
            return new JSONArray(new JSONTokener(values));
393     }
        catch (Exception e)
395     {
            Log.e(LIVE_CARD_TAG,"Failed to get data points",e);
397         return null;
        }
399 }

401 /** This method gets data from the given url website
     *
403  * @param _url, url in which the data are contained
     * @return String, contained data from the given url
405  * @throws IOException if an IO exception occurred during the download
     */
407 private String getURL(String _url) throws IOException
    {
409     URL url = new URL(_url);
        InputStream is = url.openStream();
411     int ptr;
        StringBuffer buffer = new StringBuffer();
413     while ((ptr = is.read()) != -1)
        {
415         buffer.append((char)ptr);
        }
417     return buffer.toString();
    }
419
    /** This runnable updates the graphs of pH and Temperature    */
421 private class UpdateSensorGraphsRunnable implements Runnable {
        private boolean mIsStopped = false;
423     public void run()
        {
```

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

```
                }
475             // Store the timestamps and values in separate arrays for graphing
                ArrayList<Double> timestamps = new ArrayList<Double>();
477             ArrayList<Double> values = new ArrayList<Double>();
                for (DataPoint curr_point : curr_data) {
479                 timestamps.add(curr_point.getTimestamp());
                    values.add(curr_point.getValue());
481                 //System.out.println("Value = " + values.get(j));
                }
483
                mSensorAverage.put(curr_sensor,computeAverage(values));
485             // Scale the timestamp data
                double maxTimestamp = Collections.max(timestamps);
487             double minTimestamp = Collections.min(timestamps);
                maxTimestamp *= 1.2;
489             minTimestamp *= 0.8;
                double intervalSize = maxTimestamp - minTimestamp;
491             for (int i1 = 0; i1 < timestamps.size(); i1++) {
                    double currTimestamp = timestamps.get(i1);
493                 currTimestamp -= minTimestamp;
                    currTimestamp /= intervalSize;
495                 currTimestamp *= 100;
                    timestamps.set(i1, currTimestamp);
497             }
                // Scale the value data
499             double maxVal = Collections.max(values);
                double minVal = Collections.min(values);
501             maxVal *= 1.2;
                minVal *= 0.8;
503             intervalSize = maxVal - minVal;
                for (int i1 = 0; i1 < values.size(); i1++) {
505                 double currVal = values.get(i1);
                    currVal -= minVal;
507                 currVal /= intervalSize;
                    currVal *= 100;
509                 values.set(i1, currVal);

511             }

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


517             Plot plot = Plots.newPlot(yData);
                LineChart lineChart = GCharts.newLineChart(plot);
519             lineChart.setSize(400, 200);
                lineChart.addYAxisLabels(AxisLabelsFactory.newNumericRangeAxisLabels(minVal, max
521
```

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

525                 }

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

531                 // restart it after a given amount of time
                    mHandler.postDelayed(mUpdateSensorGraphsRunnable,GRAPH_UPDATE_DELAY_MILLIS);
533             }
        }
535
        private double computeAverage(ArrayList<Double> values) {
537         double sum = 0.0;
            if(!values.isEmpty()){
539             for(Double value : values){
                    sum += value;
541             }
                return sum/values.size();
543         }
            return sum;
545     }

547     /** It is the getter function that shows the status of runnable
         *
549      * @return mIsStopped, true if the runnable is stopped, false otherwise
         */
551     public boolean isStopped()
        {
553         return mIsStopped;
        }
555
        /** It is the setter that allows to stop the runnable
557      *
         * @param isStopped, true if the user wishes to stop the runnable, false otherwise
559      */
        public void setStop(boolean isStopped)
561     {
            this.mIsStopped = isStopped;
563     }
    }
565 //
    //
567 //    class DataPoint {
    //        private final double mTimestamp;
569 //        private final double mValue;
    //        DataPoint(double timestamp, double value) {
571 //            mTimestamp = timestamp;
```

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


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

        /** Class constructor of ImageDownloader runnable
         *
         * @param url, url link of image
         * @param c, is the context in which the request of downloading image has been sent
         * @see
         */
        public ImageDownloader(String url,Context c )
        {
            this.url = url;
            this.c = c;
        }

        /** It implements the task of ImageDownloader runnable
         *
         * @see
         */
        @Override
        public void run()
        {
            if(!isStopped())
            {
                bmp = getBitmapFromURL(url);

                mCallback.setBMP(bmp);
                Log.i(BEATING_TAG,"bmp settato");
            }
        }

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

```
621         public boolean isStopped()
            {
623             return mIsStopped;
            }
625
            /** It is the setter that allows to stop the runnable
627          *
             * @param isStopped, true if the user wishes to stop the runnable, false otherwise
629          */
            public void setIsStopped(boolean isStopped)
631         {
                this.mIsStopped = isStopped;
633         }
        }
635     /** This method pulls an image from the given url
         *
637      * @param urlLink where the image is stored
         * @return Bitmap which contains the image of the graph
639      * @throws java.io.IOException if an IO exception occurred during the download
         */
641     public static Bitmap getBitmapFromURL(String urlLink){
            try{
643             Log.i(BEATING_TAG, "start downloading image ");
                long startTime = System.currentTimeMillis();
645             URL url = new URL(urlLink);
                HttpURLConnection connection = (HttpURLConnection) url.openConnection();
647             connection.setDoInput(true);
                connection.connect();
649             InputStream inputStream = connection.getInputStream();
                Log.i(BEATING_TAG,"download completed in "
651                     + ((System.currentTimeMillis() - startTime) / 1000)
                        + " sec");
653             return BitmapFactory.decodeStream(inputStream);
            } catch (IOException e) {
655             e.printStackTrace();
                Log.e(BEATING_TAG, e.getMessage());
657             return null;
            }
659     }

661     //================================================================================================ /

663     // ================================================================================================ //

665     // Runnable that updates the microscope video
        private class UpdateMicroscopeVideoRunnable implements Runnable {
667         private boolean mIsStopped = false;
            public void run() {
669             if (!isStopped()) {
```

```
                getMicroscopeVideo();
671             mHandler.postDelayed(mUpdateMicroscopeVideoRunnable, VIDEO_UPDATE_DELAY_MILLIS);
            }
673         }
            public boolean isStopped() {
675             return mIsStopped;
            }
677         public void setStop(boolean isStopped) {
                this.mIsStopped = isStopped;
679         }
    }
681
    // Pull the microscope video from a URL
683 private void getMicroscopeVideo() {
        try {
685             URL url = new URL("http://planar-contact-601.appspot.com/video/view");
                long startTime = System.currentTimeMillis();
687             Log.i(VIDEO_TAG, "video download beginning: "+url);
                URLConnection ucon = url.openConnection();
689             ucon.setReadTimeout(0);
                ucon.setConnectTimeout(0);
691             // Define InputStreams to read from the URLConnection.
                InputStream is = ucon.getInputStream();
693             BufferedInputStream inStream = new BufferedInputStream(is, 1024*5);
                File file = new File(TEMP_VIDEO_FILE_NAME);
695
                FileOutputStream outStream = new FileOutputStream(file);
697
                FileLock lock = outStream.getChannel().lock();
699             byte[] buff = new byte[1024*5];
                // Read bytes (and store them) until there is nothing more to read(-1)
701             int len;
                while ((len = inStream.read(buff)) != -1) {
703                 outStream.write(buff,0,len);
                }
705             // Clean up

707             outStream.flush();
                lock.release();
709             outStream.close();
                inStream.close();
711             Log.i(VIDEO_TAG, "download completed in "
                        + ((System.currentTimeMillis() - startTime) / 1000)
713                     + " sec");
        }
715     catch (IOException e) {
            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;
2
3  import com.google.android.glass.timeline.DirectRenderingCallback;
   import com.jjoe64.graphview.GraphView;
5
   import android.content.Context;
7  import android.content.Intent;
   import android.graphics.Bitmap;
9  import android.graphics.Canvas;
   import android.os.Environment;
11 import android.os.Handler;
   import android.os.SystemClock;
13 import android.util.Log;
   import android.view.SurfaceHolder;
15 import android.view.View;
16
17 import java.util.Map;
18
19 /**
    * {@link DirectRenderingCallback} used to draw the chronometer on the timeline {@link com.google.and
21  * Rendering requires that:
    * <ol>
23  * <li>a {@link SurfaceHolder} has been created through monitoring the
    *     {@link SurfaceHolder.Callback#(SurfaceHolder)} and
25  *     {@link SurfaceHolder.Callback#(SurfaceHolder)} callbacks.
    * <li>rendering has not been paused (defaults to rendering) through monitoring the
27  *     {@link com.google.android.glass.timeline.DirectRenderingCallback#renderingPaused(SurfaceHolder
    * </ol>
29  * As this class uses an inflated {@link View} to draw on the {@link SurfaceHolder}'s
    * {@link Canvas}, monitoring the
31  * {@link SurfaceHolder.Callback#(SurfaceHolder, int, int, int)} callback is also
    * required to properly measure and layout the {@link View}'s dimension.
33  */
   public class AppDrawer. implements DirectRenderingCallback {
35      /** INT associated to the Menu view request  */
        private static final int MENU = 0;
37      /** INT associated to the PH view request  */
        private static final int PH = 1;
39      /** INT associated to the Menu view request  */
        private static final int TEMPERATURE = 2;
41      /** INT associated to the Video view request  */
        private static final int VIDEO = 3;
43      /** INT associated to the Beating view request  */
```

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


47      private VideoThread mRenderThread;

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

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


55
        private final MainView mMainView;
57      private final BeatingView mBeatingView;
        /** View object of the GlassWear pH window */
59      private final PHViewer mPhViewer;
        private final TemperatureView mTemperatureView;
61
        private SurfaceHolder mHolder;
63      private boolean mRenderingPaused;

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


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


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

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

```
93                  //state = PH;
                    updateRenderingState();
95              }


97
        };
99
        private final BeatingView.Listener mBeatingListener = new BeatingView.Listener() {
101
            @Override
103         public void onChange() {
                updateRenderingState();
105         }
        };
107
        private final TemperatureView.Listener mTemperatureListener = new TemperatureView.Listener() {
109
            @Override
111         public void onChange() {
                updateRenderingState();
113         }
        };
115
        /** Defines the ManageBitmap of Beating viewer, it is used to communicate with
117      * that viewer in order to update the bitmap to be displayed*/
        private final BeatingView.ManageBitmap mBeatingBitmap = new BeatingView.ManageBitmap(){
119         @Override
            /* Getter that returns the bitmap of the beating image to be displayed
121          *
             * @return Bitmap, sensor values with the values of pH and Temperature
123          */
            public Bitmap getBitmap() {
125             return bmp;
            }
127
            public boolean isReady(){
129             return mReady;
            }
131     };

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

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

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

```
143         public boolean isReady(){
                return mReady;
145         }
    };
147
    /** Defines the ManageDataGraph of PH viewer, it is used to communicate with
149  * that viewer in order to update the hash map which contains the graph to be displayed*/
    private final TemperatureView.ManageBitmap mTemperatureManageDataGraph = new TemperatureView.Man
151
        public Bitmap getBitmap() {
153         return mCurrentSensorGraphs.get("pH");
        }
155
        public double getAvg(){return mSensorAverage.get("Temperature");}
157
        public boolean isReady(){
159         return mReady;
        }
161 };


163
    public AppDrawer.(Context context) {
165     this(new MainView(context), new BeatingView(context), new PHViewer(context), new Temperature
    }
167
    public AppDrawer.(MainView countDownView, BeatingView chronometerView, PHViewer phViewer, Temper
169     mMainView = countDownView;
        mMainView.setListener(mMainListener);
171
        mBeatingView = chronometerView;
173     mBeatingView.setListener(mBeatingListener);
        mBeatingView.setManageBPM(mBeatingBitmap);
175
        mPhViewer = phViewer;
177     mPhViewer.setListener(mPhListener);
        mPhViewer.setManageBPM(mPHManageDataGraph);
179
        mTemperatureView = temperatureView;
181     mTemperatureView.setListener(mTemperatureListener);
        mTemperatureView.setManageBPM(mTemperatureManageDataGraph);
183
        mRenderThread = new VideoThread(mHolder);
185     mReady = false;
    }
187



189
    /**
```

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

211          mTemperatureView.measure(measuredWidth, measuredHeight);
             mTemperatureView.layout(
213                  0, 0, mTemperatureView.getMeasuredWidth(), mTemperatureView.getMeasuredHeight());
         }
215
         /**
217       * Keeps the created {@link SurfaceHolder} and updates this class' rendering state.
          */
219      @Override
         public void surfaceCreated(SurfaceHolder holder) {
221          // The creation of a new Surface implicitly resumes the rendering.
             mRenderingPaused = false;
223          mHolder = holder;
             updateRenderingState();
225      }

227      /**
          * Removes the {@link SurfaceHolder} used for drawing and stops rendering.
229       */
         @Override
231      public void surfaceDestroyed(SurfaceHolder holder) {
             mHolder = null;
233          updateRenderingState();
         }
235
         /**
237       * Updates this class' rendering state according to the provided {@code paused} flag.
          */
239      @Override
```

```
    public void renderingPaused(SurfaceHolder holder, boolean paused) {
241         mRenderingPaused = paused;
            updateRenderingState();
243     }

245     /**
         * Starts or stops rendering according to the {@link com.google.android.glass.timeline.LiveCard}
247      */
        private void updateRenderingState() {
249         if (mHolder != null && !mRenderingPaused) {
                switch (AppManager.getInstance().getState())
251             {
                    case MENU:
253                     mRenderThread.quit();
                      //  mMediaPlayer.setDisplay(null);
255                     draw(mMainView);
                        mBeatingView.stop();
257                     mPhViewer.stop();
                        mTemperatureView.stop();
259                     mMainView.start();
                        break;
261                 case BEATING:
                      //  mMediaPlayer.setDisplay(null);
263                     draw(mBeatingView);
                        mMainView.stop();
265                     mPhViewer.stop();
                        mTemperatureView.stop();
267                     mBeatingView.start();
                        break;
269                 case PH:
                       // mMediaPlayer.setDisplay(null);
271                     draw(mPhViewer);
                        mMainView.stop();
273                     mBeatingView.stop();
                        mTemperatureView.stop();
275                     mPhViewer.start();
                        break;
277                 case TEMPERATURE:
                      //  mMediaPlayer.setDisplay(null);
279                     draw(mTemperatureView);
                        mMainView.stop();
281                     mBeatingView.stop();
                        mPhViewer.stop();
283                     mTemperatureView.start();
                        break;
285                 case VIDEO:
                      //  mMediaPlayer.setDisplay(mHolder);
287                     mRenderThread.setShouldRun(true);
                        mRenderThread.start();
```

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

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

305             }

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

321     Canvas canvas;

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

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

```
     */
339     public void setBMP(Bitmap bmp){

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


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


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


369


371 }
```

Listing A.19: MainView.java

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

```
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

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

import android.content.Context;
import android.media.AudioManager;
import android.media.SoundPool;
import android.os.Handler;
import android.os.SystemClock;
import android.util.AttributeSet;
import android.view.LayoutInflater;
import android.widget.FrameLayout;
import android.widget.TextView;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.TimeUnit;

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

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

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

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


    private final TextView timeText;


```

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

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


75  private Listener mListener;
    private boolean mRunning = false;
77
    public MainView(Context context) {
79      this(context, null, 0);
    }
81
    public MainView(Context context, AttributeSet attrs) {
83      this(context, attrs, 0);
    }
85
    public MainView(Context context, AttributeSet attrs, int style) {
87      super(context, attrs, style);
        LayoutInflater.from(context).inflate(R.layout.live_card_layout, this);
89      timeText = (TextView) findViewById(R.id.timestamp);

91  }

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

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

```
111
    /**
113  * Starts the countdown animation if not yet started.
     */
115 public void start() {
        if (!mRunning) {
117         postDelayed(mUpdateViewRunnable, 0);
        }
119     mRunning = true;
    }
121
    /**
123  * Stops the chronometer.
     */
125 public void stop() {
        if (mRunning) {
127         removeCallbacks(mUpdateViewRunnable);
           // mStarted = false;
129     }
        mRunning = false;
131 }


133


    /**
135  * Updates the view to reflect the current state of animation, visible for testing.
137  *
     * @return whether or not the count down is finished.
139  */
    void updateView() {
141     //if (mRunning) {
        timeText.setText( new SimpleDateFormat("hh:mm a").format(new Date()));
143         // updateView(millisLeft);
            if (mListener != null) {
145             mListener.onChange();
            }
147
        //}
149 }
}
```

Listing A.20: BeatingView.java

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

import android.content.Context;
4 import android.graphics.Bitmap;
import android.graphics.drawable.Drawable;
6 import android.os.Handler;
```

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


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

    boolean mState = false;
    private Bitmap bmp;

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

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

    private ImageView beatingImage;
    private final TextView mTitle;

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

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

    private boolean mRunning;

    public interface ManageBitmap{
        public Bitmap getBitmap();
```

```
56
        public boolean isReady();
58  }

60
    private Listener mChangeListener;
62
    private ManageBitmap mManage;
64
    public BeatingView(Context context) {
66      this(context, null, 0);
    }
68
    public BeatingView(Context context, AttributeSet attrs) {
70      this(context, attrs, 0);
    }
72
    public BeatingView(Context context, AttributeSet attrs, int style) {
74      super(context, attrs, style);
        LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
76      mTitle = (TextView) findViewById(R.id.message);
        beatingImage = (ImageView) findViewById(R.id.image_left);
78      mTitle.setText("Beating");
        int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/lay
80      beatingImage.setImageResource(id);

82
    }
84

86  /**
     * Sets a {@link Listener}.
88   */
    public void setListener(Listener listener) {
90      mChangeListener = listener;
    }
92
    /**
94   * Returns the set {@link Listener}.
     */
96  public Listener getListener() {
        return mChangeListener;
98  }

100 /**
     * Starts the chronometer.
102  */
    public void start() {
104     if (!mRunning) {
```

```
            postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
        }
        mRunning = true;
    }

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

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

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


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

```
154             mChangeListener.onChange();
        }
156     }

158


160
}
```

Listing A.21: PHViewer.java

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

3  import android.content.Context;
   import android.graphics.Bitmap;
5  import android.graphics.drawable.Drawable;
   import android.os.Handler;
7  import android.text.Layout;
   import android.util.AttributeSet;
9  import android.util.Log;
   import android.view.LayoutInflater;
11 import android.widget.FrameLayout;
   import android.widget.ImageView;
13 import android.widget.LinearLayout;
   import android.widget.TextView;
15
   import com.jjoe64.graphview.GraphView;
17 import com.jjoe64.graphview.GraphViewSeries;
   import com.jjoe64.graphview.LineGraphView;
19


21 /**
    * View used to display draw a running Chronometer.
23  *
    * This code is greatly inspired by the Android's Chronometer widget.
25  */
   public class PHViewer extends FrameLayout {
27
       private Context context;
29     private Bitmap bmp;

31     /**
        * Interface to listen for changes on the view layout.
33      */
       public interface Listener {
35         /** Notified of a change in the view. */
           public void onChange();
37     }
```

```
39      /** About 24 FPS, visible for testing. */
        static final long DELAY_MILLIS = 41;
41
        private ImageView beatingImage;
43      private final TextView mTextTitle;
        private TextView AvgView;
45
        private final Handler mHandler = new Handler();
47      private final Runnable mUpdateTextRunnable = new Runnable() {

49          @Override
            public void run() {
51              if (mRunning) {
                    updateText();
53                  postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
                }
55          }
        };
57
        private boolean mRunning;
59
        public interface ManageBitmap{
61          public Bitmap getBitmap();

63          public boolean isReady();

65          public double getAvg();
        }
67


69      private Listener mChangeListener;

71      private ManageBitmap mManage;

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

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

81      public PHViewer(Context context, AttributeSet attrs, int style) {
            super(context, attrs, style);
83          LayoutInflater.from(context).inflate(R.layout.buso_layout, this);
            beatingImage = (ImageView) findViewById(R.id.image_left);
85          mTextTitle = (TextView) findViewById(R.id.message);
            AvgView = (TextView) findViewById(R.id.avg);
87          mTextTitle.setText("PH");
```

```
        int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/lay
89      beatingImage.setImageResource(id);
//        int id = getResources().getIdentifier("com.google.android.glass.sample.stopwatch:drawable/l
91      //        beatingImage.setImageResource(id);

93

    }
95


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

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

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

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

131 @Override
    public boolean postDelayed(Runnable action, long delayMillis) {
133     return mHandler.postDelayed(action, delayMillis);
    }
135

    @Override
```

```
137     public boolean removeCallbacks(Runnable action) {
            mHandler.removeCallbacks(action);
139         return true;
        }
141     /**
         * Sets a {@link Listener}.
143      */
        public void setManageBPM(ManageBitmap manager) {
145         mManage = manager;
        }
147


149     /**
         * Updates the value of the chronometer, visible for testing.
151      */
        void updateText() {
153         if(mManage.isReady())
            {
155             Log.i("PH", "ce prova");
                bmp = mManage.getBitmap();
157             AvgView.setText(Double.toString(mManage.getAvg()));
                beatingImage.setImageBitmap(bmp);
159


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


173


175 }
```

Listing A.22: TemperatureView.java

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

        import android.content.Context;
4       import android.graphics.Bitmap;
        import android.graphics.drawable.Drawable;
6       import android.os.Handler;
        import android.text.Layout;
```

```
8         import android.util.AttributeSet;
          import android.util.Log;
10        import android.view.LayoutInflater;
          import android.widget.FrameLayout;
12        import android.widget.ImageView;
          import android.widget.TextView;
14


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

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

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

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

52      private boolean mRunning;

54      public interface ManageBitmap{
            public Bitmap getBitmap();
56
```

```
        public boolean isReady();

        public double getAvg();
    }


    private Listener mChangeListener;

    private ManageBitmap mManage;

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

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

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

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


    }


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

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

    /**
     * Starts the chronometer.
```

```
106      */
         public void start() {
108          if (!mRunning) {
                 postDelayed(mUpdateTextRunnable, DELAY_MILLIS);
110          }
             mRunning = true;
112      }

114      /**
          * Stops the chronometer.
116       */
         public void stop() {
118          if (mRunning) {
                 removeCallbacks(mUpdateTextRunnable);
120          }
             mRunning = false;
122      }

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


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

             }
154          //  else
```

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

164


166

}
```

Listing A.23: VideoPlayerActivity.java

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

3  import android.app.Activity;
   import android.content.Intent;
5  import android.content.res.AssetManager;
   import android.net.Uri;
7  import android.os.Bundle;
   import android.os.Environment;
9  import android.util.Log;
   import android.view.Menu;
11 import android.view.MenuItem;
   import android.widget.MediaController;
13 import android.widget.VideoView;

15 import java.io.File;
   import java.io.FileOutputStream;
17 import java.io.IOException;
   import java.io.InputStream;
19 import java.io.OutputStream;
   public class VideoPlayerActivity extends Activity {
21    private static final int VIDEO_PLAY_REQUEST_CODE = 200;
      private static final String TAG = "VIDEO_TAG";
23    public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
25        String filepath;
          Bundle extras = getIntent().getExtras();
27        if (extras != null)
              filepath = extras.getString("filepath");
29        else {
              filepath = copyAsset(VIDEO_FILE_NAME);
31        }

33        Intent i = new Intent();
```

```
        i.setAction("com.google.glass.action.VIDEOPLAYER");
        i.putExtra("video_url", filepath);
        startActivityForResult(i, VIDEO_PLAY_REQUEST_CODE);
    }
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == VIDEO_PLAY_REQUEST_CODE )
            finish();
    }
    String copyAsset(String filename) {
        final String PATH = Environment.getExternalStorageDirectory().toString() + "/myvideoapps/";
        File dir = new File(PATH);
        if (!dir.exists()) {
            if (!dir.mkdirs()) {
                Log.v(TAG, "ERROR: Creation of directory " + PATH + " on sdcard failed");
                return null;
            } else {
                Log.v(TAG, "Created directory " + PATH + " on sdcard");
            }
        }
        if (!(new File( PATH + filename).exists())) {
            try {
                AssetManager assetManager = getAssets();
                InputStream in = assetManager.open(filename);
                OutputStream out = new FileOutputStream(PATH + filename);
                byte[] buf = new byte[1024];
                int len;
                while ((len = in.read(buf)) > 0) {
                    out.write(buf, 0, len);
                }
                in.close();
                out.close();
            } catch (IOException e) {
                Log.e(TAG, "Was unable to copy " + filename + e.toString());
                return null;
            }
        }
        return PATH + filename;
    }
}
```

Listing A.24: MenuActivity.java

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

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.Menu;
```

```
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Toast;

import java.lang.Runnable;

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

    /** INT associated to the Menu view request  */
    private static final int MENU = 0;
    /** INT associated to the PH view request  */
    private static final int PH = 1;
    /** INT associated to the Menu view request  */
    private static final int TEMPERATURE = 2;
    /** INT associated to the Video view request  */
    private static final int VIDEO = 3;
    /** INT associated to the Beating view request  */
    private static final int BEATING = 4;

    private final Handler mHandler = new Handler();
    private int state = 0;

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.stopwatch, menu);
        return true;
    }

    @Override
    public boolean onPreparePanel(int featureId, View view, Menu menu) {
        AppManager appManager = AppManager.getInstance();

        boolean initialView = appManager.getState() == MENU;
        boolean imageView = appManager.getState() == PH ||
                appManager.getState() == TEMPERATURE ||
                appManager.getState() == BEATING ||
                appManager.getState() == VIDEO;

```

```
        setOptionsMenuState(menu, R.id.action_back, imageView);
        setOptionsMenuState(menu, R.id.action_view_ph, initialView);
        setOptionsMenuState(menu, R.id.action_view_temperature, initialView);
        setOptionsMenuState(menu, R.id.action_view_video, initialView);
        setOptionsMenuState(menu, R.id.action_view_beating, initialView);
        setOptionsMenuState(menu, R.id.action_stop, true);

        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection.
        switch (item.getItemId()) {
            case R.id.action_stop:
                // Stop the service at the end of the message queue for proper options menu
                // animation. This is only needed when starting a new Activity or stopping a Service
                // that published a LiveCard.
                post(new Runnable() {

                    @Override
                    public void run() {
                        stopService(new Intent(MenuActivity.this, StopwatchService.class));
                    }
                });
                return true;
            case R.id.action_view_beating:
                handleViewBeating();
                return true;
            case R.id.action_back:
                handleViewBack();
                return true;
            case R.id.action_view_temperature:
                handleViewTemperature();
                return true;
            case R.id.action_view_ph:
                handleViewPh();
                return true;
            case R.id.action_view_video:
                handleViewVideo();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

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

```
106         mHandler.post(new Runnable() {
                @Override
108             public void run() {
                    Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
110                 intent.putExtra(Intent.EXTRA_TEXT, "Video");
                    startService(intent);
112             }
            });
114     }

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

142     private void handleViewBack() {
            Toast.makeText(this, "Menu", Toast.LENGTH_SHORT).show();
144         // launch a new thread for starting a new live card
            mHandler.post(new Runnable() {
146             @Override
                public void run() {
148                 Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
                    intent.putExtra(Intent.EXTRA_TEXT, "Menu");
150                 startService(intent);
                }
152         });
        }
154
```

```
    @Override
156 public void onOptionsMenuClosed(Menu menu) {
        // Nothing else to do, closing the Activity.
158     finish();
    }
160
    /**
162  * Posts a {@link Runnable} at the end of the message loop, overridable for testing.
     */
164 protected void post(Runnable runnable) {
        mHandler.post(runnable);
166 }

168 /**
     * The function handle the request to view the beating plot
170  */
    private void handleViewBeating() {
172     Toast.makeText(this, "Beating", Toast.LENGTH_SHORT).show();
        // launch a new thread for starting a new live card
174     mHandler.post(new Runnable() {
            @Override
176         public void run() {
                Intent intent = new Intent(MenuActivity.this, StopwatchService.class);
178             intent.putExtra(Intent.EXTRA_TEXT, "Beating");
                startService(intent);
180         }
        });
182 }

184 private static void setOptionsMenuState(Menu menu, int menuItemId, boolean enabled){
        MenuItem menuItem = menu.findItem(menuItemId);
186     menuItem.setVisible(enabled);
        menuItem.setEnabled(enabled);
188 }

190 }
```

Listing A.25: AppManager.java

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

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

8     private int state;

10    private static  AppManager instance = new AppManager();
```

```
12      public static AppManager getInstance(){
            return instance;
14      }

16      public void setState(int value){
            state = value;
18      }

20      public int getState(){
            return state;
22      }

24  }
```

Listing A.26: DataPoint.java

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

[3]

LIST OF FIGURES

LIST OF LISTINGS

BIBLIOGRAPHY

[1] Sara Ghoreishizadeh Giovanni De Micheli Sandro Carrara Andrea Cavallini, Camilla Baj-Rossi. 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] 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`.