Documentation DESY Summer Project 2016. Slow Control with Raspberry PI for Test Beam Telescope.

Manuel Morgado.[1,2]
[1]*University Simon Bolivar* (**USB**) *Caracas, Venezuela.* (manuelmorgadov@gmail.com)
[2]*Deutsch Elektronen Synchrotron* (**DESY**) *Hamburg, Germany.* (manuel.morgado.vargas@desy.de)

***Abstract*— Test Beam Telescopes are an invaluable tool for the improvement and development of the silicon detectors for experiments in High Energy Physics as ATLAS at LHC. In order to bring more information about the conditions of each test, it will be implemented an slow controller hardware which is responsible to measure different conditions in the test beam. This is achieved by implementing two sensors (BMP180 and SHT21), connected to a Raspberry Pi 2 (+B) which is also connected to the data acquisition system of the telescope (*AKA* EUDAQ).**

I. General

Basic Commands and Initial Set Up

To start using Raspberry PI is important to know some basic commands in the LINUX terminal and also get a basic idea of manage a repository.

In order to access to the Raspberry Pi (**RPi**) which should be connected to the DESY network, it can be used with `SSH` connection:

```
→ ssh pi@raspberry-name
```

(by example, `ssh pi@tb-raspberry-01`)
and then introduce the password of the RPi. In case that the connection is made from the DESY-Guest network, is necessary access to a computer in the DESY network, using your credential with the service Bastion. By example:

```
→ ssh username@bastion.desy.de
```

and the repeat the step before. Once in the RPi, is possible to access to any directory or folder inside also execute Python scripts and compile code. The Raspberry Pi is more like an computer than the PCB where you can use any of the commons commands of the terminal (`clear`, `cd ..`, `ls`, `scp`, `nano`, `wget`, `sudo`, `apt-get` etc.)

Now in order to get some repository (either *EUDAQ* or *RPi_HTP* repo), we can made:

```
→ git clone https:\\url.of.the\repo
```

and then we'll see the folder *repo* in our directory (In case that the RPi does not have the git core installed, then you have to install with: `sudo apt-get install git-core`). We now are allowed to read, use and modify this files but the changes can be only saved until we push it to the git. This could be done as:

```
→ git add .
→ git commit -m "Some comment."
```

and finally:

```
→ git push
```

For this last step it's necessarily to introduce our username and password of the account of the git. In other hand, is necessary install some C^{++} and Python libraries to control and check the data from the different ports of the RPi.

The Python library is *SMBus* and can be downloaded using:

```
→ sudo apt-get install smbus
```

But the tricky one to use for our purpose is the C^{++} library, *WiringPi*. And in order to use the GPIO with I^{2}C and SPI protocols is necessary to complete some steps before to be allowed to use the commands of this library. Because, the 2015 update of *Raspbian* includes the new *kernel* (version 3.18) of the RPi and also includes a configuration change to enable *Device Tree* (DT) support by default so is necessary to able the protocols as follows:

```
→ sudo raspi-config
```

and then enable SPI, I2C and Serial using the option number 9 and then selecting `<Yes>` when it asks you to enable each options. And in the end select the `<Finish>` option. Finally reboot with the RPi:

```
→ sudo reboot now
```

Installing the *WiringPi*, we first it all have to upgrade the git-core version:

```
→ sudo apt-get update now
→ sudo apt-get upgrade
```

and then clone the git of *WiringPi*:

```
→ git clone git://git.drogon.net/wiringPi
```

so, after this we have to build the code as below:

```
→ cd wiringPi
→ git pull origin
→ cd wiringPi
→ ./build
```

II. Hardware

The main goal is to obtain data from the sensor using the Raspberry Pi and then integrate it with the EUDAQ framework in the sense of all data can be merge in a single file.

In figure 1 it is shown how is the general set up of the slow control. In the next sections, each component will be explained.

A. Basics of Raspberry Pi

The Raspberry Pi is a device which is capable to plug to different peripherals as mouse, screen and keyword, in a few words is a tiny computer (credit card size) [1].

[1]More information about Raspberry Pi:`https://www.raspberrypi.org/help/faqs/#introWhatIs`

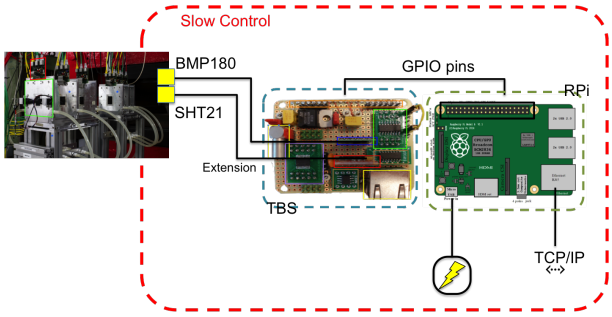

Fig. 1. *General overview of the hardware in the testbeam.*

RPi 2 model B was the one used for this project and it has 4 USB, 1 micro-usb, 1 Ethernet, 1 HDMI, 1 combined 3.5mm audio jack and composite video ports also 29 *General Purpose Input Output* (**GPIO**) pins, *Micro SD* card slot, *Camera interface* (**CSI**) and *Display interface* (**DSI**). The processor consist in a 900MHz quad-core ARM Cortex-A7 (BCM 2836 32bit) with 1GB of RAM.

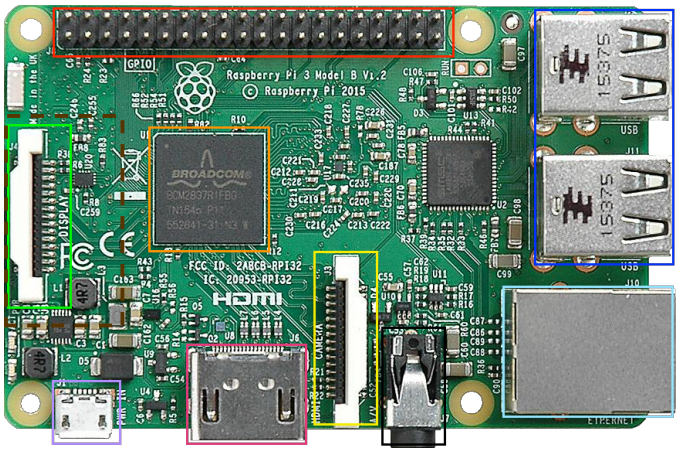

Fig. 2. Raspberry 3 Model B photo. Each color correspond to the same components of the RPi that we used for the slow control.

There are to many *OS* for the raspberry board, however we decide to install *Raspbian* [2]. So, in order to start to work with the raspberry without any OS already installed it is recommended to follow the next tutorial. In case there the raspberry has already a *OS* that should not appear any issue to use the procedures presented in the previous section.

B. Torsten setup

As we will in the next section, in order to integrated the slow control with the EUDAQ framework it is necessary to get the data from the slow sensors in the RPi considering the trigger signal of the testbeam. That is the reason of why there is an additional board in the raspberry, build it by Torsten

[2]More information about Raspberry Pi *OS*:`https://www.raspberrypi.org/downloads/`

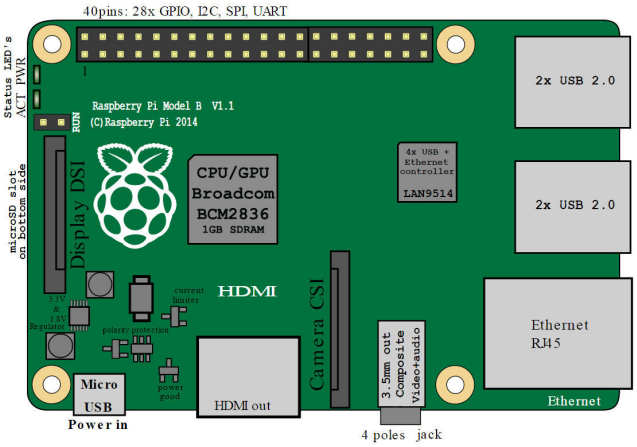

Fig. 3. Raspberry 2 Model B schematic.

Külper[3]. This board connect the GPIO ports to the sensors (SHT21 & BMP180), a Counter which also has an input for analog trigger and additional TCP/IP input.

In the Torsten setup there are other components very easy to see, the Analog digital Converter (**ADC**) also other couple of chips REF196 (is a low dropout voltage device) and DS90LV032A (is a differential line receiver CMOS). [4]

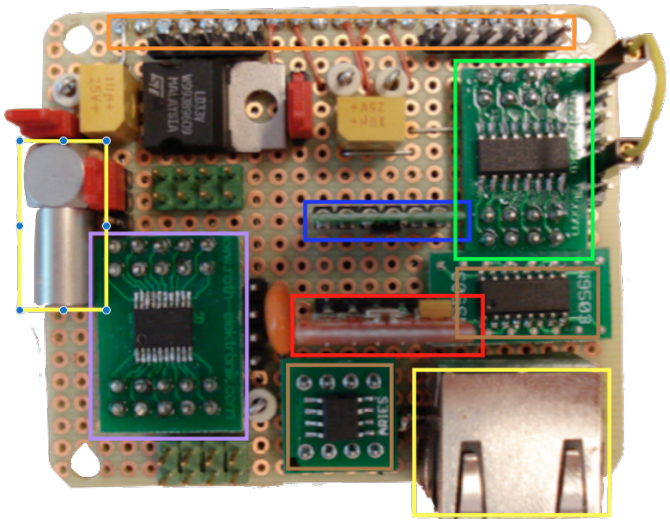

Fig. 4. Torsten setup photo. More details about the board can be find it in the documentation "Raspi Piggy-Back"

C. Sensors

The two sensors (SHT21 and BMP180) are connected to the GPIO ports 3 and 2 (9 and 8 respectively in WiringPi

[3]Raspi Piggy-Back documentation can be find here.

[4]In the Torsten setup there are jumpers which are very useful at the moment of select the kind of trigger or connect other component using the ADC.

library) and they can be used through I^2C protocol connection (Inter Integrated Circuit or I^2C is a serial computer bus developed in order to communicate with devices and peripherals of lower speed) trough some libraries in C^{++} (wiringPi) and python (SMBus).

SHT21 sensor can measure the parameters of humidity and temperature in a ranges between 10-60^{0}C with a precision of 0.4^{0}C and 20-80%RH with a precision of 2%RH as we can see in the next graphs.

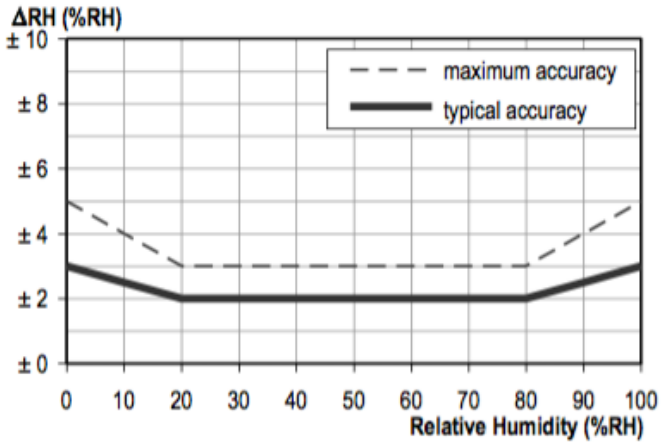

Fig. 5. Humidity performance of the SHT21 sensor.

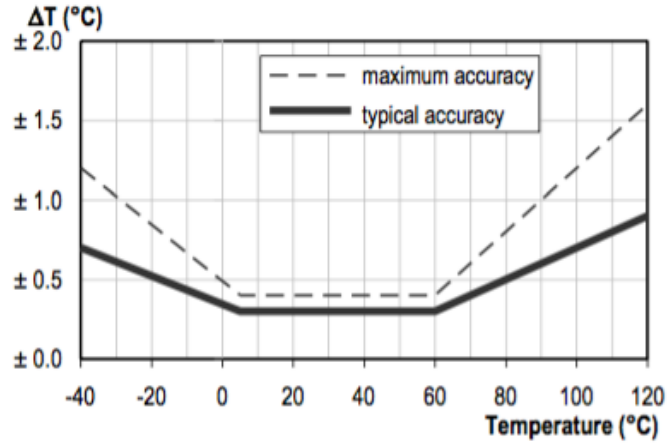

Fig. 6. Temperature performance of the SHT21 sensor.

Also, in the figure 7 is shown the correlations between the accuracy of the temperature and the humidity.

This sensor consists of 6 pin, but the most important are the SDA (*data*) and SCL (*clock*) in order to understand how the scripts can manipulate the data using the I^2C with this pins, it is why it will be explained briefly. The communication with the sensor should have a *start* and *stop* process, the first one consist in a header of 7-bits follows of one bit which tell the direction of the SDA (write '0' or read '1') and then ACK bit (bit of receipt of response). And

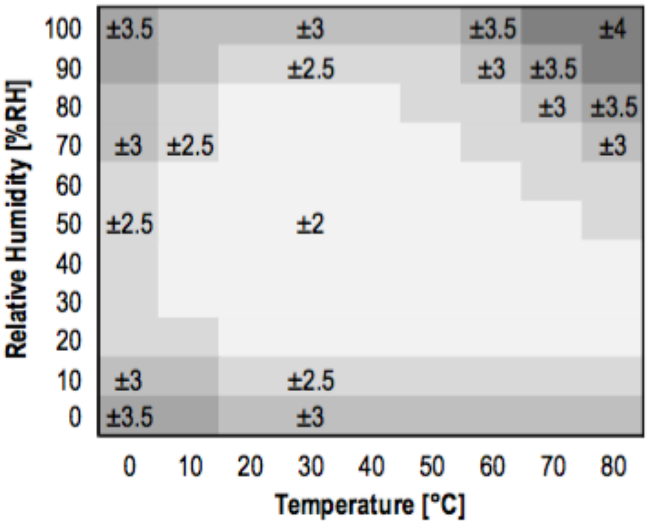

Fig. 7. Correlation of temperature and humidity for SHT21 sensor.

is used the vice verse way for the stop process. After this initial process, ones can send a command for measure the temperature or the humidity in the different modes (Hold Master or No Hold Master, the second one allows to use other I^2C communication), by example the NHM measure of the humidity is shown in the figure 8 [5]:

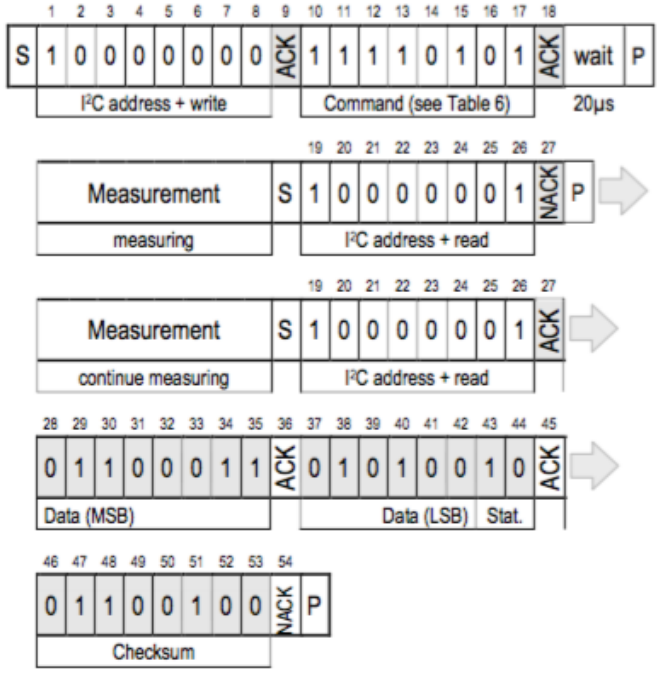

Fig. 8. No Hold Master communication example for SHT21.

The final data come from two 8-bits set (MSB Most

[5]For more details about the communication with SHT21 see the data sheet.

Significant Bit and LSB Last Significant Bit) in the case of our example is `01100011` and `01010010` in left aligned is `0110001101010010` = 25426 = S_{RH} and then we can use the expression for the humidity provided by the manufacturer:

$$RH = 125 \cdot \left(\frac{S_{RH}}{2^{16}}\right) - 6 \tag{1}$$

which results in 42.49%RH. Also, can be compute the relative humidity respect with the water:

$$RH_i = RH_w \cdot \left(\frac{exp\left\{\frac{\beta_w t}{\lambda_w + t}\right\}}{exp\left\{\frac{\beta_i t}{\lambda_i + t}\right\}}\right) \tag{2}$$

where the values of the constants λ and β could be find it in the data sheet of the sensor. And in the same way it can be used for the measure of the temperature, using this expression:

$$T = 175.72 \cdot \frac{S_T}{2^{16}} - 46.85 \tag{3}$$

III. Hardware Management

In order to create some scripts which control the sensor through the RPi must be used some libraries which contains the functions to set, read, write and close the pins using the I^2C protocol.

A. Libraries for RPi (Python & C^{++})

There are a few libraries which can be used to control the pins for the sensors using directly some functions from the shell or well using it in the scripts.

SMBus (*Python library*)
This library basically recreate an I^2C using the GPIO ports, was principally used in the BMP180 python script, in which one can create a bus using GPIO connection and then simulate the I^2C connection with the sensor. More information and explanation can be found it directly in the codes.

RPi.GPIO (*Python library*)
Using the RPi.GPIO library for the SHT21 sensor (`sudo python RPi_HTP/SHT21/Python/sht21C.py`) we can see the measurements (see figure 9).

The most common commands used are the the `setup`, `output` and `sleep`, could be check it the properly use in the python script `rpi_i2c.py` which is used in the `sht21C.py` class.

WiringPi (*C^{++} library*)
Using the WiringPi library there is many commands that can be used in order to use with the pins of the RPi.

- `wiringPiSetup(void);` Initialize WiringPi using numbers between 0-16.

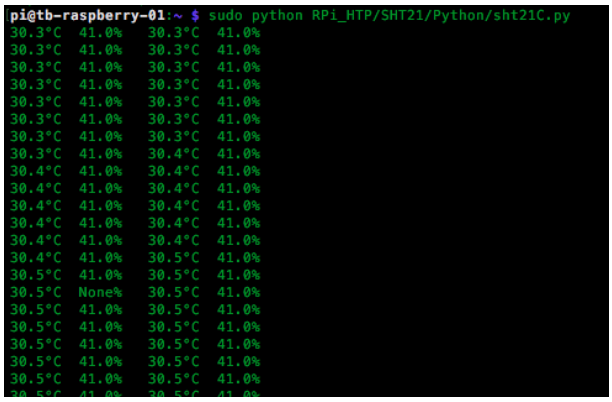

Fig. 9. Measurement of the Temperature and Humidity using SHT21 sensor and Python script.

- `wiringPiSetupGpio(void);` Initialize WiringPi without remaping.
- `wiringPiSetupsPhys(void);` Physical numbers pin of bus 1.
- `wiringPiSetupSys(void);` Use `/sys/class/gpiointerface`
- `void pinMode(int pin, int Mode);` set the mode of the pin INPUT, OUTPUT, PWM_OUTPUT, GPIO_CLOCK.
- `void pullUpOnControl(int pin, int pud);` Pull up or down the internal resistance.
- `void digitalWrite(int pin, int value);` Set a digital value of the pin (0 or 1).
- `void pwmWrite(int pin, int value);` Write the value of the register of the pin (value between 0-1024).
- `void digitalRead(int pin);` Read the pin value (HIGH or LOW).
- `void analogRead(int pin);` read the ping in analog way.
- `void analogWrite(int pin, int value);` write the pin in analog way.

B. Basic lines of LINUX terminal for RPi

In Unix terminal is possible to access to some features of the RPi. Some examples will be shown below:

IV. Software

A. SHT21 & BMP180 Code

For this project was developed scripts for the both sensors using Python and C^{++}. The code will be explained very roughly in this section, all codes are in this github repository `https://github.com/manuelmorgado/RPi_HTP.git`, the comments on the code will be very useful in the case that a more detailed explanation be needed. The pseudocode for each sensor will be shown below (Algorithm 1).

This algorithm build the SHT21 class by example that is represented in the scheme of the figure 12,

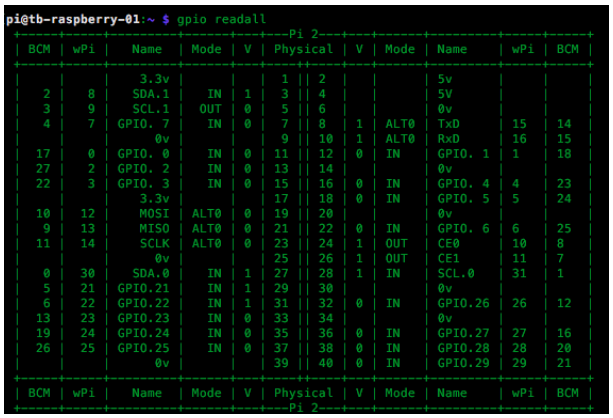

Fig. 10. Readall command, shows a map of the GPIO ports in the RPi.

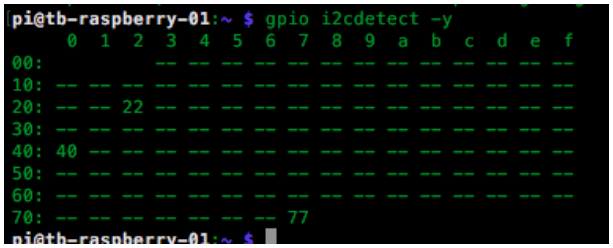

Fig. 11. i2Cdetect shows the available ports for I^{2}C protocol connection. The I^{2}C ports 0x40 and 0x77 are the BMP180 and SHT21 sensors respectively and 0x22 is the ADC.

others commands that can be used in the terminal that they are very intuitive:

→ `gpio mode PIN# in/out`
→ `gpio read PIN#`
→ `gpio write PIN# value`

more information can be found it using `man gpio`

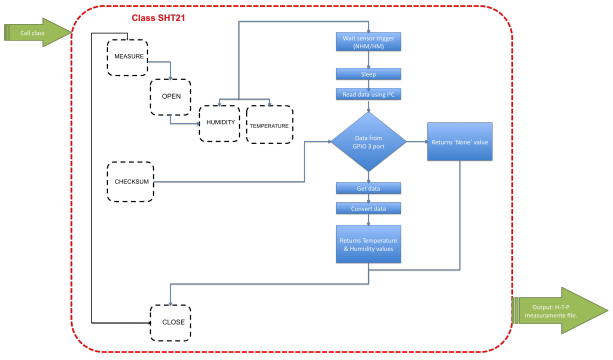

Fig. 12. *Pseudocode* for sensor SHT21.

In order to show the sensor is getting data, we measure the temperature in two sites. One was with the sensor attached directly at the board and the other one was using a cable extension developed by my self. The difference can be noticed very easily in figure 13 and the extension is in the

Output: Temperature and Humidity values.

begin

-Call the class I^{2}C in order to emulate I^{2}C bus using GPIO.

`// Define functions of the class.`

function MEASURE(`self, dev, scl,sda`)
`//Self: is the device.dev: is the bus.`
`//scl: is the clock.sda: is the data.`
-Open device.
-Get temperature (from temperature function).
-Get humidity (from humidity function).
-Close device.
Return (temperature, humidity)
end function

function OPEN(`self, dev, scl, sda`)
-Open I^{2}C object.
-Write softreset.
-Sleep 50 ms.
end function

function TEMPERATURE(`self`)
-Writer trigger (NHM).
-Sleep (66 ms).
-Write data.
if *there is data from gpio pin 2* **then**
-Return data converted as a number
else
-Save 'None'.
end
end
end function

function HUMIDITY(`self`)
-Writer trigger (NHM).
-Sleep (25 ms).
-Read data from gpio pin 3..
if *there is data from gpio pin 2* **then**
-Return data converted as a number
else
-Return 'None'.
end
end
end function

function CLOSE(`self`)
-Close I^{2}C object.
end function
`// Standard function.`

-Use CHECKSUM function

end

Algorithm 1: *Pseudocode* of the SHT21 sensor. (Python)

figure 14

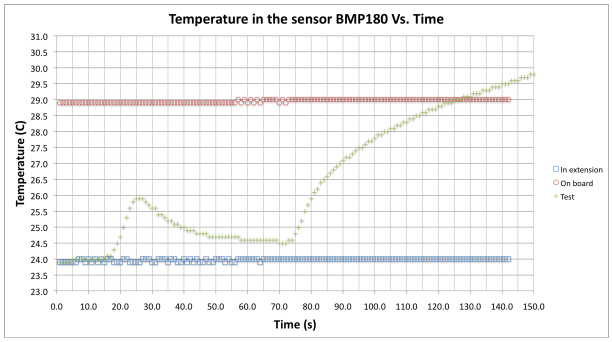

Fig. 13. Temperature measurament using the BMP180 sensor. Attached directly in the TBS (red), with a cable extension (blue) and testing data. Some extra data come from a sensor test.

Fig. 14. Extension connections for the Test beam shield. For 6 and 5 pins.

B. Counter

In the test beam shield are two ways of get trigger, TCP/IP connection and the analog connection. The last one could be connected via Jumper 5 in to the counter which is the responsible to tell to the device when is necessary to send data, in the worst case that the device does not have any new data, the last value of data should be send it.

C. Shared Memory

IPC (Inter Process Communication) is a method which creates a map of the address space (virtual space of the programs and process) of the region of the shared memory without involve the Kernel in the transfer process of the data through the method.

A more visual explanation is shown in figure 15:

Further information and examples can be find in the Boost library documentation: `http://www.boost.org/doc/libs/1_55_0/doc/html/interprocess/sharedmemorybetweenprocesses.html`

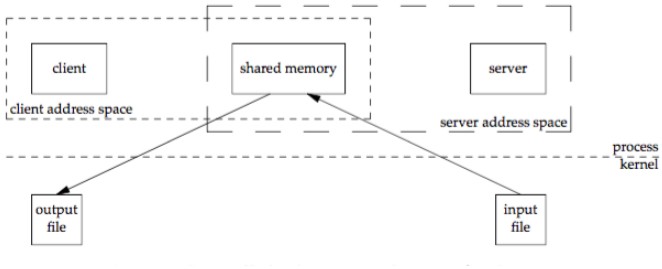

Fig. 15. Scheme of the Share Memory process.

D. Daemon

A daemon script is a script or code which allow to run a program as background process in order to execute other task simultaneously for the purpose of obtain two measurements from 2 different sensors at the same time. it is necessary to use this kind of scripts and then save the data in some place (in the shared memory) to later send it to the Data Collector using normal EUDAQ Producer. More information about libraries for multiprocessing in Python can be found here `https://docs.python.org/2/library/multiprocessing.html#`

V. Hardware + Software Implementation

A. EUDAQ Framework

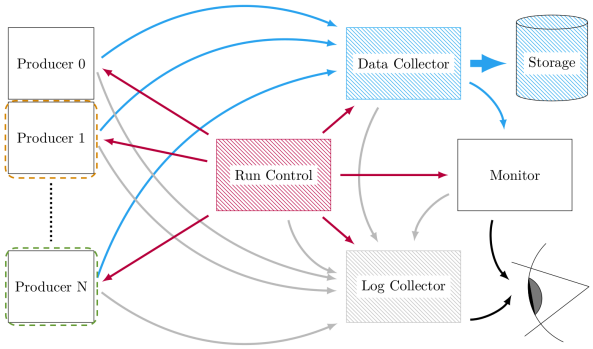

Fig. 16. *EUDAQ architecture.*

An overview of the EUDAQ framework is showed in the figure 16 this explain how the Producers, Data Collector and Run Control are communicated. In a few words:

1) *Run Control*(RC) ask the addresses to *Data Collector*(DC) and *Log Collector*(LC).

2) Report it to the producers and the monitor.

3) Producers send its data and log data to DC and LC.
4) DC handles and merges the data.
5) DC save it in a file and also shown on the Monitor.

B. Slow Control/Slow Producer + EUDAQ

There is two possibilities to integrate the peripherals to EUDAQ:

1) *Slow Control as normal producer* (Triggered device).
   - Daemon and shared memory features need it in order to send data for every trigger.
   - Data duplication (**i.e** Trigger rate is $\sim$KHz and read-out time of RPi is $\sim$1s)
2) *Slow Producer as new kind of producer* (Not triggered).
   - Could be identified by the DC in order to get data or not data.
   - DC can receive data and continue in otherwise DC can keep going.

VI. More

A. Future improvements

For future works there is a good idea to change two other sensors (e.g **BME280**) which one let use a higher temperature and humidity range in order that the user are available to used a lower temperatures also get the three parameters at once and then get a little bit faster the data. Other good idea is to add an IR camera in order to get a complete map of heat of the sensor also the complete bench of the testbeam. Finally other idea for the future is the made a benchmark between the new idea of *Slow Producer* and the Classical idea of the *Slow Control System* (further development of Shared Memory and Daemon necessary) in order to chose the faster and reliability way to do it.

VII. References

References

[1] WiringPi
[2] Raspberry Pi Documentation
[3] Torsten's Documentation
[4] E. Corrin. "EUDAQ Software Manual". Edition (2010).
[5] H. Jansen, S. Spannagel, J. Behr, A. Bulgheroni, G. Claus, E. Corrin, D. G. Cussans, J. Dreyling-Eschweiler, D. Eckstein, T. Eichhorn, M. Goffe, I. M. Gregor, D. Haas, C. Muhl, H. Perrey, R. Peschke, P. Roloff, I. Rubinskiy, M. Winter. "Performance of the EUDET-type beam telescopes. ". arXiv:1603.09669v2.(2016)