Burn-In Test Software Development

Taras Fedorchuk, Taras Shevchenko National University of Kyiv, Ukraine

September 8, 2018

Abstract

A new software package developed for the burn-in test of the CMS Phase 2 Outer Tracker is described. The framework is implemented in C++ using Qt and VISA libraries.

Contents

1 Introduction

For the upcoming CMS Phase 2 Outer Tracker upgrade several thousands of modules will be produced. These modules will be tested in several steps. One of them is the burn-in test, when small batches of modules will be assembled and tested together at different temperatures.This test needs a special software to control the whole setup. A new package has been developed in C++ using Qt and VISA libraries. Information about the software and hardware parts of the burn-in test is shown below.

2 LHC Upgrade

2.1 LHC Schedule

The High Luminosity Large Hadron Collider (HL-LHC) is a project which is aimed on increasing the luminosity of the current Large Hadron Collider(LHC) and according to the LHC/HL-LHC Plan will start in 2025. In 2018 there will be a LS2 (Long Shutdown) which will extend until 2020. The main feature of this shutdown is the upgrading of LHC Injector[1]. Luminosity in this period will increase by factor 2 of compared to the current operation. Run 3 is forseen for the years 2020 to 2022. After that the most interesting part of upgrade is coming. During LS3 the Phase 2 upgrade will be done. In this shutdown, the detectors of all major LHC experiments will have to be upgraded.

2.2 CMS Detector Structure

The CMS experiment (Compact Muon Solenoid) is one of the largest detectors in LHC. It is 21.6 metres long, 15 m in diameter, and weighs about 14,000 tonnes. The CMS magnet is a "solenoid" which is made of coils of wire and produces a very strong uniform magnetic field with strength up to 4T. Tracker and calorimeter detectors are located inside the coil. The general structure of the detector is shown in figure 1.
The luminosity will be substantially increased after upgrade. Under these conditions the number of collisions per bunch-crossing, or pile-up, is expected to reach or exceed 140 and the total integrated radiation dose is expected to increase by a factor of ten with respect to the initial LHC design value. Henceforth, the entire tracking system has to be improved. The main requirements

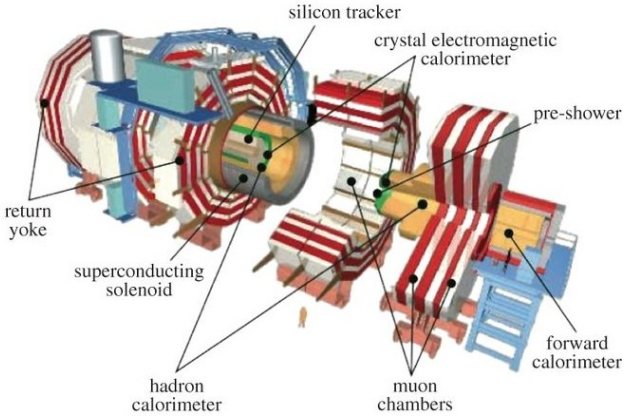

Figure 1: CMS detector structure

to tracker upgrade can be summarized as follows: an increased radiation tolerance, higher granularity to maintain the channel occupancy and finally, the upgraded tracker has to contribute to the level 1 trigger of CMS. The upgraded tracker detector will be composed of two parts: inner tracker (IT) and outer tracker (OT).

2.3 Future module concept

The modules for the outer tracker are composed of two closely-spaced silicon sensors, read out by front-end electronics placed at the edges of each module. The main purpose of these modules is to provide information about the track. For each hit on the bottom sensor an acceptance window on the top sensor is defined. A pair of hits passing the acceptance window is called a "stub" (first track in figure 2). These "stubs" indicate high p_T events and are transmitted to L1 trigger afterwards.
Two types of modules are planned: two-strip("2S") and pixel-strip("PS"). 2S modules are composed of two strip sensors. Each sensor will be approximately 10x10 cm^2 with 5 cm long strips and 90 μm pitch. This type of module will be placed in the outer part of the outer tracker. PS modules are composed of strip and pixel sensors. A strip sensor size will be 10x5 cm^2 with 2.35 cm strips with 10 μm pitch[2]. Each pixel on the pixel sensor will be 100 μm x 1400μm. PS modules will used in the inner part of the outer tracker to provide a more precise measurements of the z coordinate for tracking. The layout of the outer

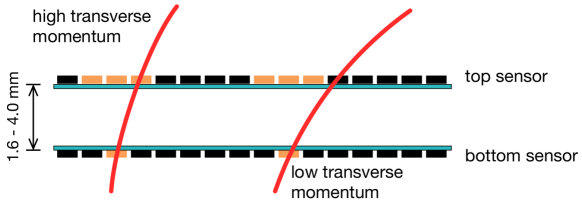

Figure 2: On-board pT discrimination

tracker can be seen in the figure 3. Blue lines correspond to PS modules and red ones to 2S modules.

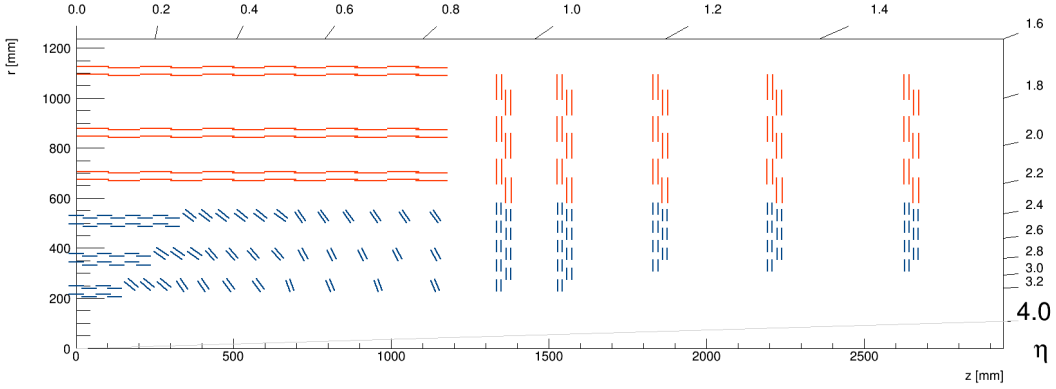

Figure 3: Layout of the future outer tracker

3 Burn-in test

A couple of thousands of modules will be produced and need to be tested for the upcoming detector upgrade. DESY takes part in production of these modules and end-cap is planned to be assembled on site. A sequence of tests will be performed as a part of production process:

- Reception test - a quick test of modules that just arrived.

- Burn-it test - several modules (up to 50) will be tested together at different temperatures.
- Integration test - modules that passed through two previous tests will be mounted on the end-cap. Sectors will be tested separately.

4 Hardware structure

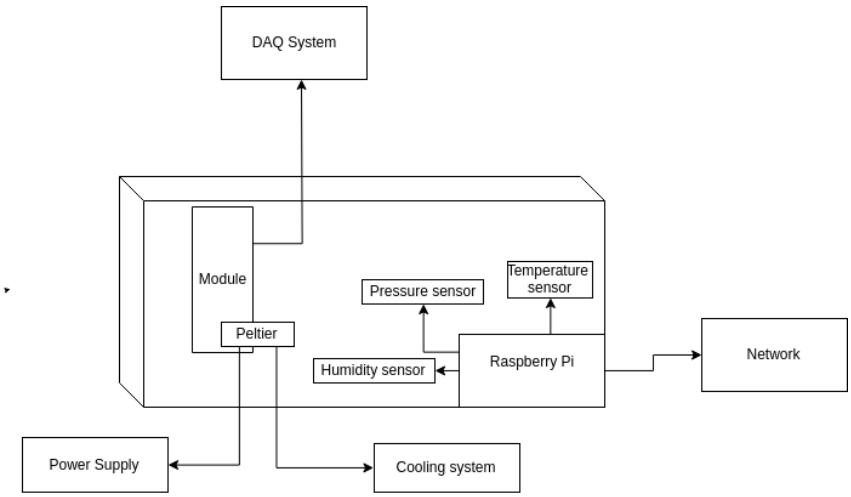

Figure 4: Hardware structure

In figure 4 one can see a schematic view of the demo version of the burn-in test setup. A module under test will be placed inside this insulated box. A Peltier element and liquid cooling system are responsible for the temperature control of the module. Raspberry Pi will perform the monitoring of the temperature, humidity,etc inside the box. Several power supplies will be used to apply the bias voltage, low voltage for the electronics and voltage for the Peltier cooling.

5 Software

5.1 Git repository

The most recent version of the code can be accessed using the next link *https://github.com/fedorchuk1/Burn-InSetupControlSoftware*. An example of the hardware description file is located in the folder *settings* and in the Appendix B of the current report.

5.2 Compiling the software

A quick start guide is shown below:

- VISA drivers are avaliable using the link `http://www.ni.com/download/ni-linux-device-drivers-2018/7664/en/`. Then one should choose the appropiate package and install it later with command "*sudo yum install ni-visa*" (for CentOs7).
- Use "`git clone https://github.com/fedorchuk1/Burn-InSetupControlSoftware`" to fetch the source code.
- Find location of VISA and add it to *LIBS* and *INCLUDEPATH* in *Burn-InSetupControlSoftwate.pro* file. Example is shown there.
- Change directory to *external* and use command `git clone https://github.com/DESY-FH-ELab/cmstkmodlab.git` to get source code for the JulaboFP50 cooling system(if needed).

5.3 Source code structure

In figure 5 a general software structure is shown. Main features of each class are listed below:

- **SystemController** - main control class, which supervises all the other classes. All test procedures are defined here.
- **PowerControl** - controls the voltages on modules (low and high). This class is a generic class where main methods of power supplies are defined.

- **EnvironmentalControl** - interfaces all environmental periphery, such as chiller, sensors, Peltier elements.
- **DatabaseInterface** - gets module information from the database and publishes test results there (will be implemented later).
- **DAQControl** - interfaces the DAQ system.
- **ConnectionInterface** - gets information from Raspberry Pi sensors.
- **Graphical User Inteface** - independent from all other classes, forms a wrapper around the SystemController Class.

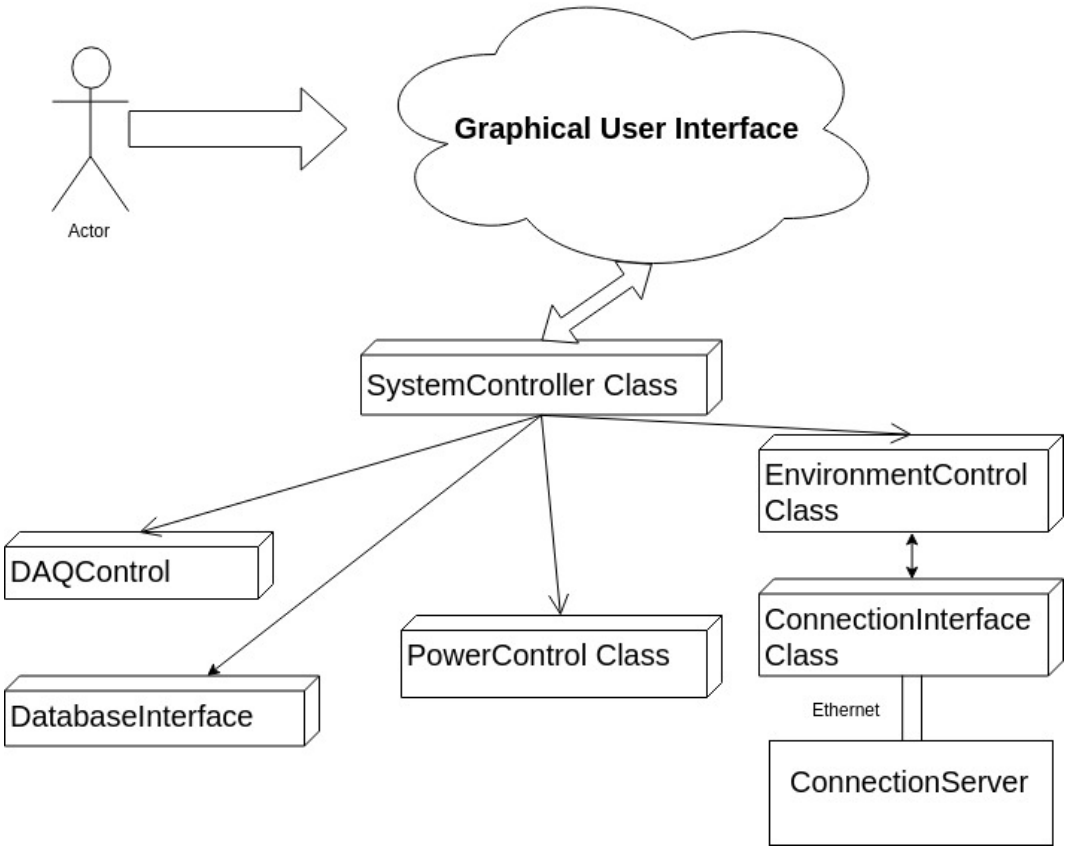

Figure 5: Software structure

5.4 Methods description

In this section description of the main methods can be found. GUI consists of three tabs: *Main Test*, *Voltage Control* and *Environmental Control*. Screenshots of the GUI are attached in the Appendix A.
To start one must read a config file. Once a configuration file was parsed the information about all devices is uploaded. By pressing the *Initialize Hardware* button connection to the devices is checked. A double-click on needed command will add command from the *List of commands* to the *Added commands*. In the widget *Added commands* one can delete command by double-clicking and move up or down command with appropriate push button. In the *Voltage Control* tab voltage and current control is placed.The last tab, *Environmental Control* is responsible for monitoring Raspberry Pi‘s sensors and controling cooling system.

Table 1: SystemControllerClass main features

struct fParameters
This struct is used for storing commands that must be done for test. Has two members. First member *string cName* displays the name of the command and second,*double cValue* - value that must set.
void ReadXmlFile(std::string pFileName)
Takes a name of the file as argument and opens it. Extension must be .xml. Parsing of this file implemented in the HWDescriptionParser class.
void ParseChiller(), void ParseVSources(), void ParseRaspberry()
This functions read config file and make an objects of corresponding classes. Then put created objects to general map that called *fGenericInstrumentMap*.
bool Initialize()
Checks the connection to all devices listed in configuration file.
string getConnectionString(string pConnection , string pAddress, string pPort)
Returns a string for connecting to the devices using VISA library.
void startDoingList()
This function is running the sequence of commands that user set in GUI.

Table 2: Description of main members of MainWindow class

struct output_pointer_t, struct output_Raspberry, struct output_Chiller
These structs are defined for displaying environmental features in their widgets. First one is for voltage sources, second for Raspberry Pi and the last one is for cooling system. Each struct is a template for its instrument. So one can easily add new instruments to the GUI.
void on_listOfCommands_doubleClicked(const QModelIndex &pIndex)
Takes a QModelIndex as argument and puts this item to list of test commands.
void on_Start_pushButton_clicked()
Starts running a sequence of commands. One can add command to this list by reading a *.txt file or from the user interface.
void updateRaspWidget(QString pStr)
Takes a string with information from sensors, connected to a Raspberry Pi and updates widgets in the GUI.
void updateTTiIWidget(PowerControlClass::fVACvalues *pObject); **void updateKeithleyWidget(PowerControlClass::fVACvalues *pObject)**
Current/voltage monitoring.
void on_AddedComands_tabelView_doubleClicked(const QModelIndex &pIndex)
Deletes choosen row from sequence of control commands.
void on_AddedComands_tabelView_clicked(const QModelIndex &pIndex)
When one clicked on the command it will be highlighted and one can move this highlighted command up or down with buttons placed below this widget.
bool readXmlFile()
Takes an information from *fGenericInstrumentMap* and creates widgets for devices. Templates are: *output_pointer_t*, *output_Raspberry* and *output_Chiller*.
void doListOfCommands()
Creates a list of commands, formed from the *.xml file..
output_pointer_t SetSourceOutputLayout(std::string pType), **output_Raspberry setRaspberryLayout(string pName)** **output_Chiller setChilerLayout(string pType)**
These methods define a template of a widget for the voltage sources, Raspberry Pi sensors and cooling system respectively.
output_pointer_t *SetVoltageSource(QLayout *pMainLayout, std::string pName, **std::string pType, int pNoutputs),** **output_Raspberry *SetRaspberryOutput(QLayout *pMainLayout, vector <string>pNames,** **string pNameGroupBox),** **output_Chiller* SetChillerOutput(QLayout *pMainLayout, std::string pName,** **std::string pType)**
Create a widget based on template that shown above. To add widget to GUI just call one of these methos with needed arguments, first must be a general layout of your group box, second is a name of instrument and the last one is a type of it.

6 Conclusion

The software for the burn-in test has been developed using C++. Qt and VISA libraries were used for the graphical user interface and for the power supplies control respectively. New voltage sources or cooling system can be easily implemented. Still some work is planned to be done: a DAQ Interface, the connection to the module database, etc.

7 Appendix A

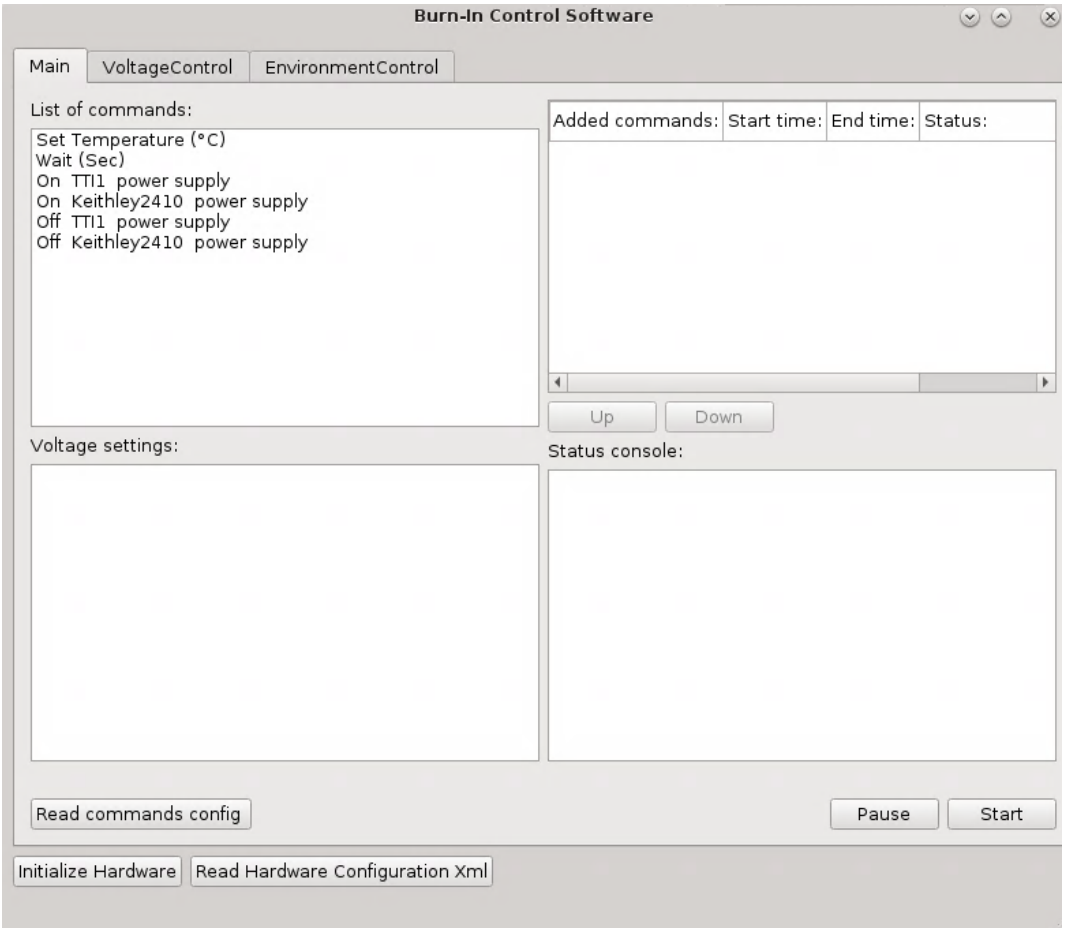

Figure 6: Main test tab

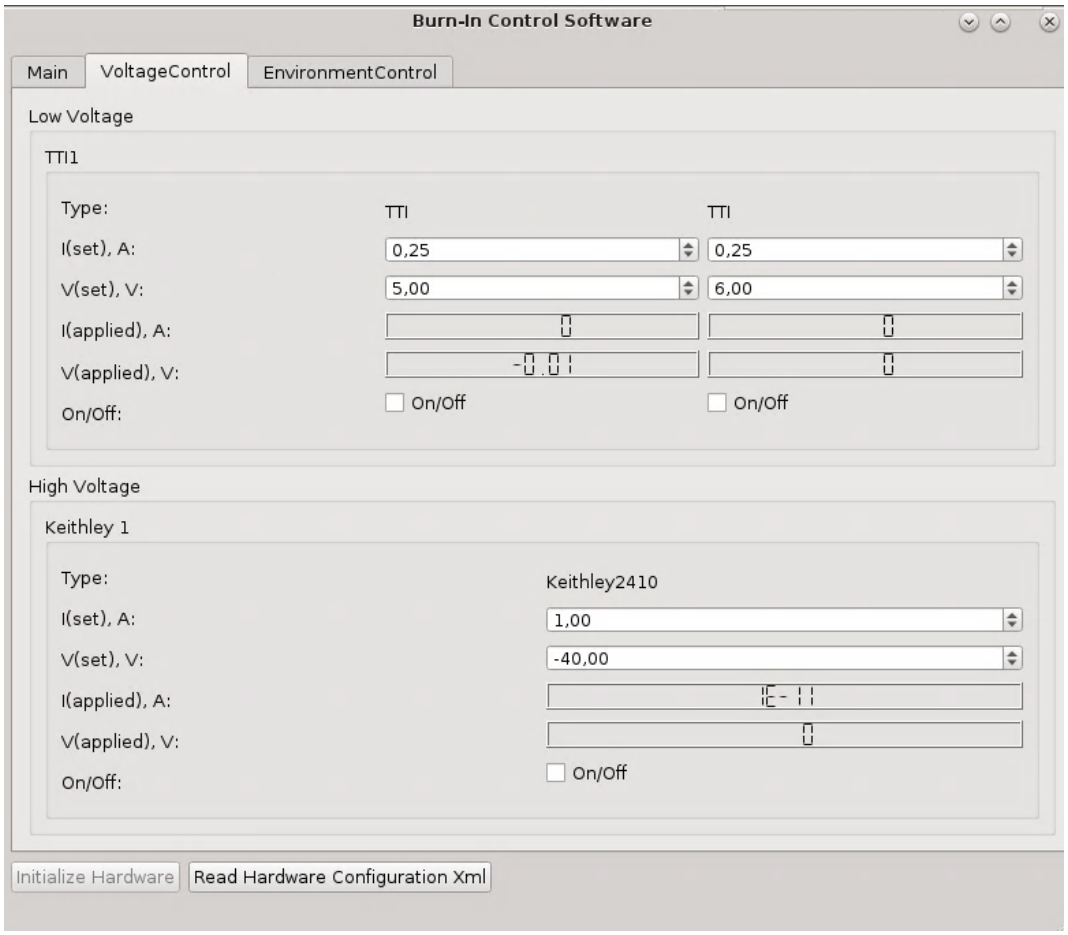

Figure 7: Voltage control tab

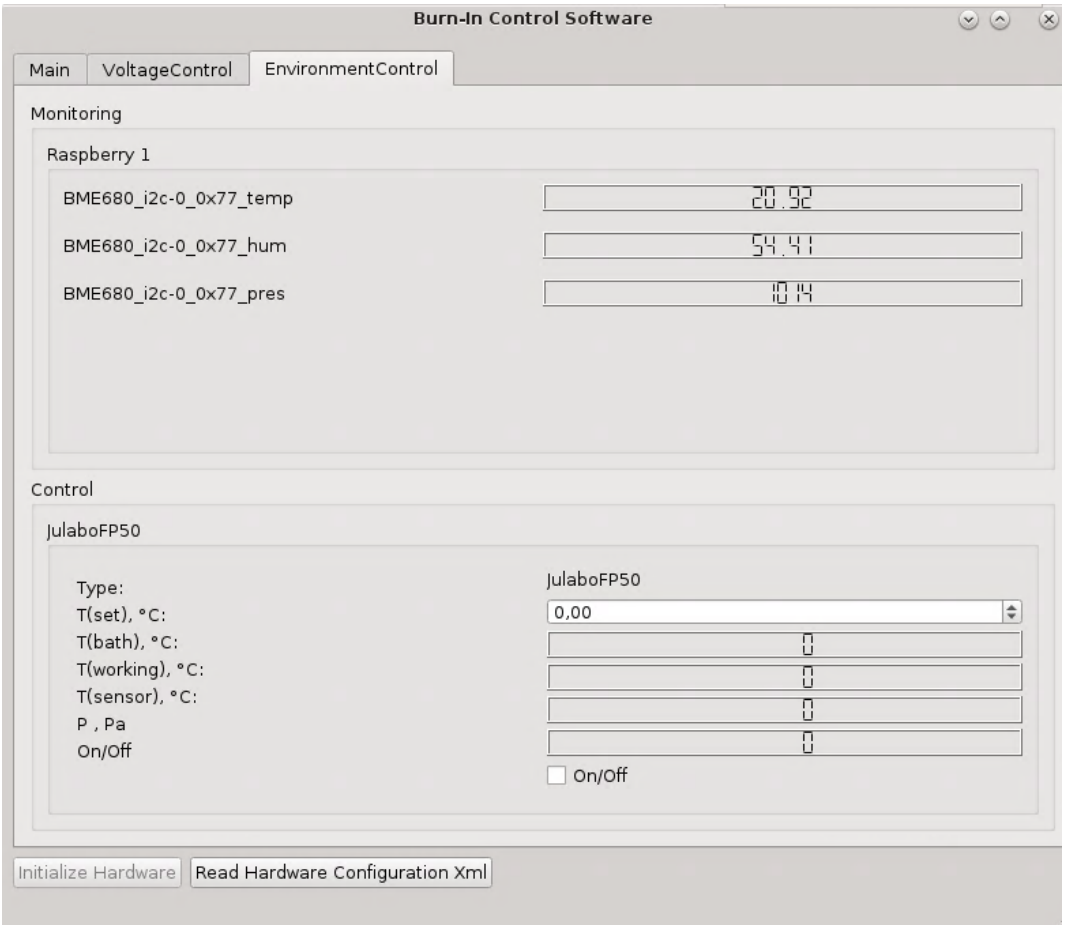

Figure 8: Environmental control tab

8 Appendix B

```
<?xml version="1.0" encoding="utf-8"?>
<HardwareDescription>
    <!-- section with power control -->
    <Power>

        <!-- Low Voltage Section -->
        <LowVoltage>
            <VoltageSource name="TTI1" class="LowVoltageSource" type="TTI" noutputs="2" connection="ethernet"
            port="9221" address="192.168.1.180" description="used for something">
                <Output output_id="0" Voltage="5" CurrentLimit="0.250"/>
                <Output output_id="1" Voltage="6" CurrentLimit="0.250"/>
            </VoltageSource>
        </LowVoltage>

        < !-- High Voltage Section -->
        <HighVoltage>
            <VoltageSource name="Keithley2410" class="Keithley2410" type="HighVoltageSource" noutputs="1"
            connection="rs232" address="/dev/ttyS5" description="used for bias">
                <Output id="0" Voltage="-40" CurrentLimit="1.0"/>
            </VoltageSource>
        </HighVoltage>

    </Power>

        <!-- environment section -->
            <Environment>

                    <!-- Chiller Section -->
                    <ChillerControl name="JulaboFP50" class="" type="Chiller" connection="rs232"
                    address="/dev/ttyUSB1" description="used to cool done the cooling block" />

                    <!-- Peltier Section -->
                    <!-- to be defined -->


                    <!--Raspberry  Section -->
                    <RaspberryControl name="fhlthermorasp4" class="" type="Raspberry" connection="ethernet"
                    address="fhlthermorasp4.desy.de" port="50007" description="used for display temperature
                    humidity pressure" >
                        <Sensor sensor="BME680_i2c-0_0x77_temp"/>
                        <Sensor sensor="BME680_i2c-0_0x77_hum"/>
                        <Sensor sensor="BME680_i2c-0_0x77_pres"/>
                    </RaspberryControl>

            </Environment>
</HardwareDescription>
```

References

[1] The upgrade programme of the major experiments at the Large Hadron Collider 2014 J. Phys.: Conf. Ser. 515 012012 *P. La Rocca and F.Riggi*

[2] Phase-2 Upgrade of the CMS Tracker, Nuclear and Particle Physics Proceedings 273275 (2016) *Stefano Mersi*