July 2021

Master thesis

Development of integration tools to test the new ATLAS pixel module prototypes

Graduate School of Science, Osaka University
Yamanaka Taku Laboratory, Department of Physics

Mario Gonzalez

Contents

Abstract

A major upgrade of the Large Hadron Collider (LHC) at CERN is scheduled to take place before 2027. After the upgrade, the rate of proton-proton collisions at the core of the ATLAS detector (one of the four detectors built at the LHC) will increase by a factor of 5.

The ATLAS Pixel Detector, which is located closest to the collision point, will be completely replaced during the upgrade. Currently, a prototype of the new pixel modules is in a testing stage. Before 2027, 2000 of the total of 10000 modules are planned to be built in Japan. We are currently preparing for this module mass production, which has to be done as efficiently as possible.

The module production consists of two stages, assembling and testing. In the assembling stage the modules are built from their components. In the testing stage they are powered up, configured and read-out while being exposed to x-ray radiation. The goal of the testing stage is to ensure that all the pixels in the pixel module are working and can be used in the detector.

This thesis presents the work that has been done towards the automation of the testing stage. With this work, the testing stage has become more consistent, precise, faster, and safer. In addition, this automation prevents any biases in the results that could be originated from operating the modules by hand. Moreover, this automation allows us to estimate the module production time more precisely, helping to organize the whole production chain to avoid bottlenecks.

This thesis also describes a monitoring system which has been developed to monitor environmental variables such as temperature, humidity, and power supply data. The data are uploaded to a database by the monitor system. With this work, other programs such as the controller of the cooling unit can use the monitor data as feedback to keep the modules in optimal conditions.

Chapter 1

Introduction

1.1 The Large Hadron Collider and the ATLAS experiment

The Large Hadron Collider (LHC), located at CERN (Switzerland), is the most energetic particle accelerator ever built. It consists of a 27-km tube forming a ring, inside which two pipes surrounded by superconducting magnets are built to circulate two proton beams in opposite directions.

The two beams are crossed in four points of the ring. The ATLAS detector, which is located in one of them, is the detector in which this research focuses on.

One of the ATLAS experiment's main achievements so far has been the experimental discovery of the Higgs bosson. The current purpose of the ATLAS experiment is to explore new physics beyond the Standard Model, as well to continue improving the current Standard Model experimental results.

The ATLAS detector

The ATLAS detector, shown in Fig. 1.1, is composed of different subdetectors, each of them specialized in measuring a certain type of particle. A detailed description of the ATLAS detector can be found in Ref. [1]. This chapter provides a quick overview of the main components of the detector.

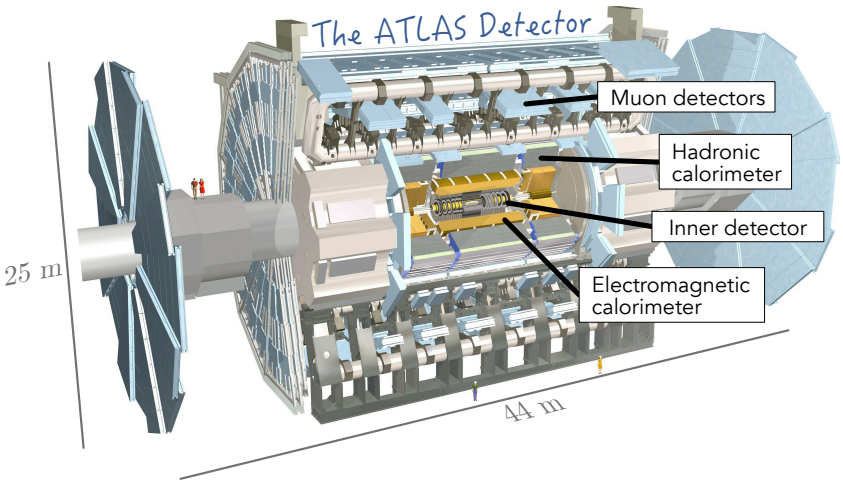

Figure 1.1: Schematic view of the ATLAS detector. The ATLAS subdetectoras have cylindrical symmetry around the beam line. Proton-proton collisions take place at the center of the inner detector.

From the innermost layer, the ATLAS detector consists of the following subdetectors:

- The **Inner Detector**, which is shown in Fig. 1.2, is placed into a magnetic field and consists of a high resolution pixel detector in the innermost layer, and other charged particle detectors in the outer side. The main purpose of the inner detector is to track the trajectory of charged particles, allowing for precise measurements of their momentum.

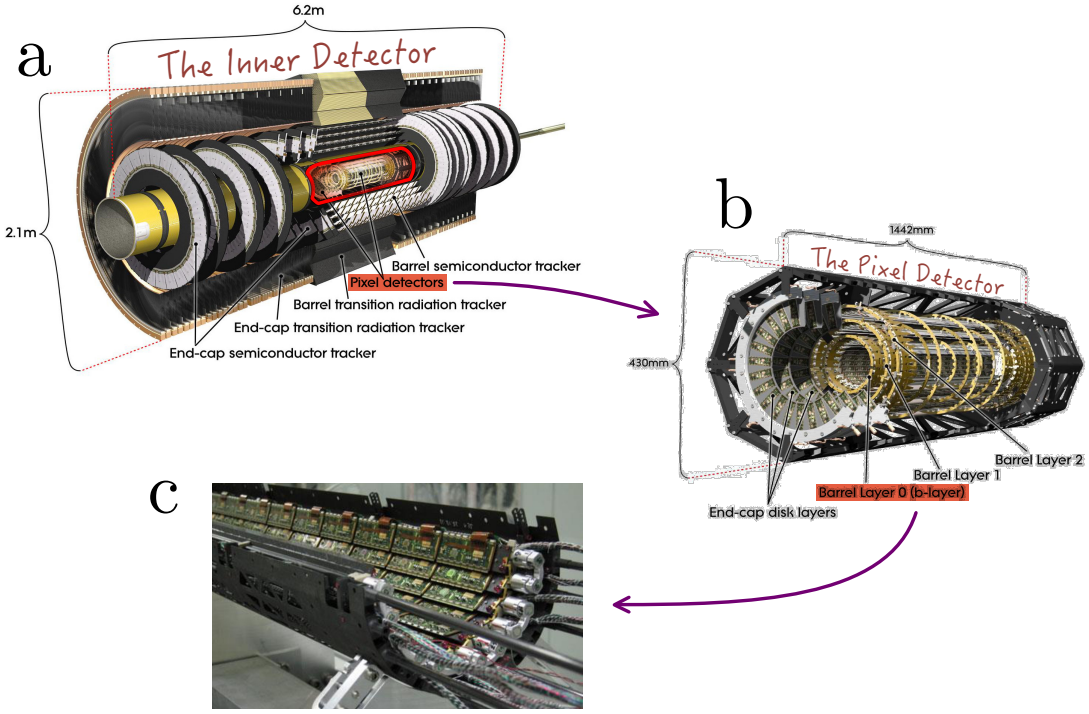

Figure 1.2: **(a)** Schematic view of the ATLAS's Inner Detector. **(b)** Schematic view of the ATLAS's Pixel Detector. **(c)** Picture of the pixel modules of the innermost layer of the Pixel Detector. The three figures show the current status of the detector.

- The **Electromagnetic Calorimeter**, whose main purpose is to measure the energy of electrons, positrons or photons.

- The **Hadronic Calorimeter**, that does the same with charged and neutral hadrons.
- The **Muon Detectors**, which detect muons, that penetrate through the electromagnetic and the hadronic calorimeters.

1.2 The High Luminosity LHC

The LHC is planned to be upgraded to the High-Luminosity LHC (HL-LHC). Starting from 2027, the HL-LHC will run at a center-of-mass energy of 14 TeV, and it will collide protons at a rate five times larger than the LHC[2].

Protons are currently colliding at 13 TeV at their center of mass. The energy of the protons is therefore 6.5 TeV, but that energy is actually spread across all the components of the proton, following certain probability distribution functions. The 13 TeV proton-proton collisions at the LHC are actually quark-quark or gluon-gluon collisions that take place at a much lower energy.

High energy particles (up to the order of TeV) are produced with a very low probability, and therefore all the studies around them are heavily restricted by the limited data. This particularly affects searches for Super Symmetry particles, that are predicted to have masses on the order of TeV. Making this kind of searches possible is the main motivation of the HL-LHC upgrade.

In preparation for the HL-LHC upgrade, the ATLAS detector also has to be upgraded. In particular, the pixel detector is planned to be fully renovated by a new one that is currently under production.

1.3 The basics of the ATLAS Pixel Detector for the High luminosity LHC

This section starts by introducing the building blocks of the pixel detector, and giving an overview of their assembly process. The process of signal readout is then explained, particularly focusing in the threshold selection, that is set to distinguish real particle hits from background noise.

1.3.1 Pixels, sensors and pixel modules

The pixel modules are the main building blocks of the pixel detector. A pixel module is composed of a silicon semiconductor sensor, and readout electronics that process the sensor's signal and send out the processed data to computers.

The sensors of the module are composed of *pixels* and powered by high voltage. The silicon pixels output a current when a charged particle passes through them. The size of the pixels in the sensor is 50×50 µm[1].

[1] $25\,\mu m \times 100\,\mu m$ pixels are also present in particular regions of the sensor.

The output signal of the sensor is processed by custom ASICs (Application Specific Integrated Circuits), that are composed of *readout pixels*. A single readout pixel in the ASICs processes the data coming from a single silicon pixel in the sensor. Hereafter, for simplicity, the term **pixel** will refer to the readout pixels of the ASIC unless explicitly specified.

A PCB (Printed Circuit Board), is also included in the module as an interface between the ASICs and the readout computers, and also between the power supplies and the ASICs and sensor. Sensor, ASICs, and PCB are the three building units of the **pixel modules**.

Different variations of the pixel modules have been built to be installed on different parts of the pixel detector. The ones this thesis focuses on have four ASICs, and they are usually referred to as **quad modules**. A real quad module placed in a metallic carrier is shown in Fig. 1.3, together with the dimensions of one of its ASICs.

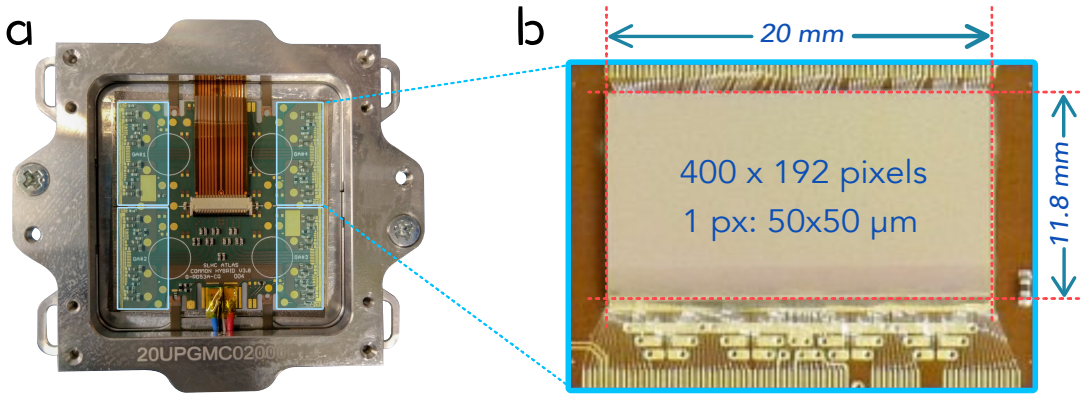

Figure 1.3: **(a)** Picture of a Quad Module, highlighting the position of the ASICs, located under the PCB. **(b)** Dimensions of one of the four ASICs of the quad module.

1.3.2 The pixel module assembly

The PCB, the sensor, and the ASICs are separately produced by different companies. The sensor is also connected to the ASICs before being shipped to the testing institutes.

The ATLAS collaborators in the testing sites receive the PCB and the sensor separately. The module **assembly** process consists of gluing the sensor to the PCB, and wire-bonding the PCB and the ASICs, as shown in Fig. 1.4.

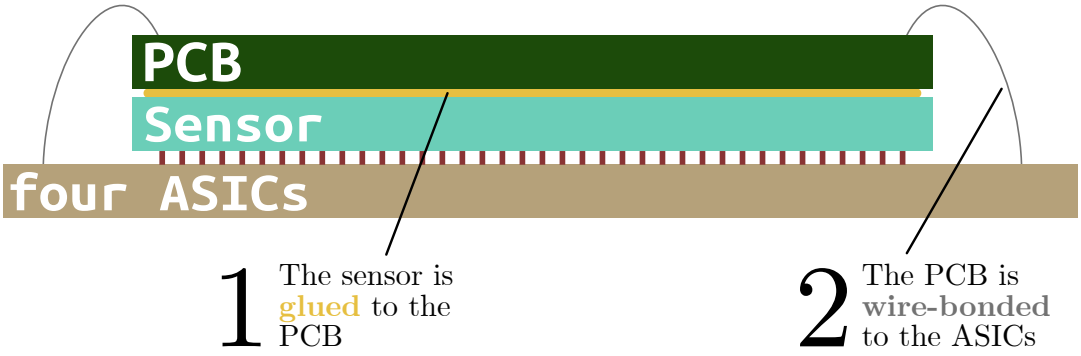

Figure 1.4: Scheme of the assembly process. The sensor is connected to the ASICs in an earlier stage of the production.

The quality of the modules has to be carefully controlled during the assembly process. The module contains many delicate parts, such as the wire bonds connecting the PCB and the sensor. Any anomaly found on them at any stage of its production has to be identified and fixed before continuing to the next stage.

1.3.3 The pixel module functioning

Once assembled, the modules are ready for usage. The module functioning is shown in Fig. 1.5. A charged particle exciting a pixel in the sensor would produce an analog pulse, that would be digitized and identified as a hit by the ASICs. The ASICs' output would then be sent to a computer through the PCB.

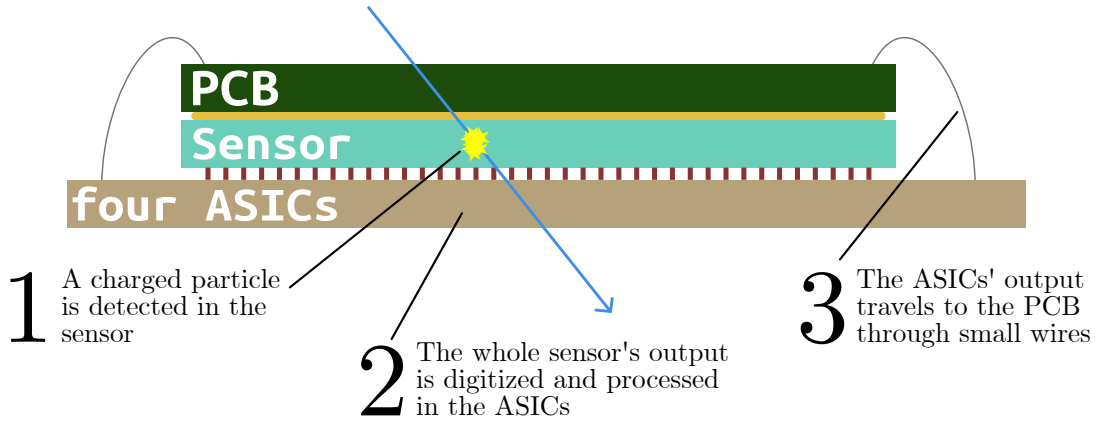

Figure 1.5: Scheme showing the data readout process in the pixel modules.

The hit decision in the ASICs is based on two properties of the signal from the sensor: Its charge and its Time over Threshold (ToT). The charge measures the strength of the signal, and a **threshold** is set to define the minimum charge that an analog pulse has to carry to be a hit candidate. The **ToT** is the time that a pulse spends over that threshold, as represented in Fig. 1.6. A minimum requirement on the ToT is also set for a pulse to be considered a hit.

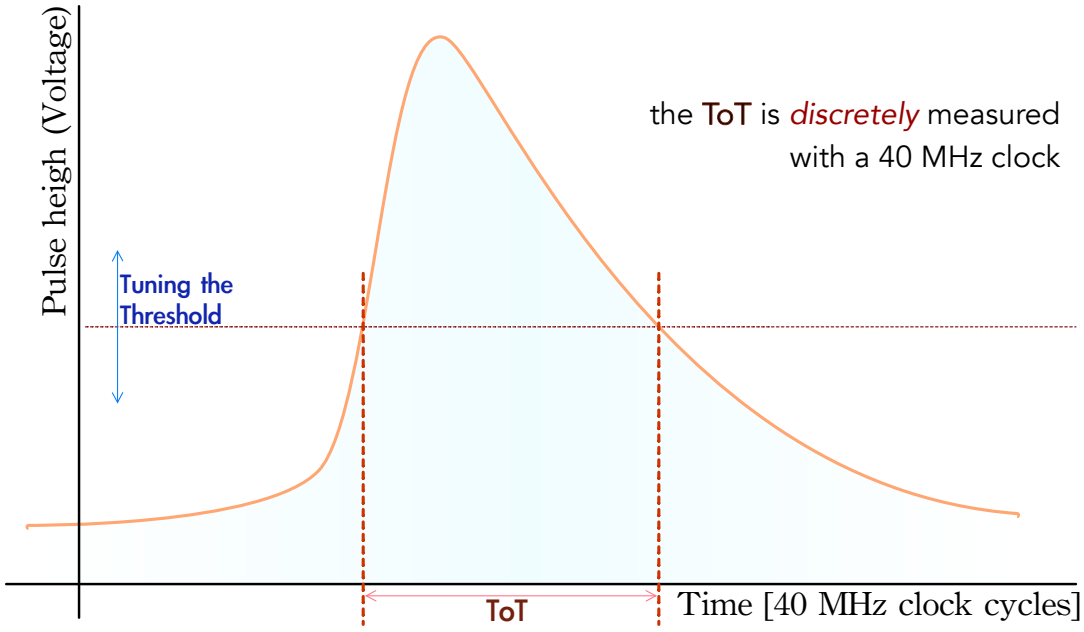

Figure 1.6: A sample pule corresponding to a hit in the sensor is represented. The threshold is measured in charge (electrons) and the ToT is measured in 40 MHz clock cycles.

The threshold tuning

The process to set the threshold for all the pixels in an ASIC to match the target threshold is called **tuning**. The threshold is set pixel by pixel by modifying a register that exists for every pixel, but the actual threshold, in units of voltage, is unknown until it is measured.

The tuning consists of the following steps:

1. 100 analog pulses whose charge matches the target threshold are sent to each pixel. If the threshold was tuned to its target threshold, and there was no noise at all, a pixel would identify those 100 pulses as hits. However, due to the effect of noise, the readout charge of a hit can be slightly below or above the threshold. If a pixel is tuned properly, it should record half of the input pulses, 50 in this case, as hits.

2. If the pixel reads less than 50 hits, its threshold is lowered by modifying a voltage register in the ASIC. If it reads more, its threshold is increased.

3. This process is repeated until the pixel records 50 of the 100 received pulses as hits.

The threshold measurement

The threshold tuning may not work for all the pixels in the matrix. Pixels could be very noisy, and record a different number of hits each time they are tested. Others could be damaged and output a constant number of hits regardless the number of analog input pulses.

Therefore, checking the quality of the tuning by measuring each pixel's threshold is needed. The following steps are taken to measure the threshold.

1. A pack of 50 pulses of an initial charge is generated and sent to each pixel. The number of detected hits is recorded.

2. The process is repeated by increasing the charge. The number of hits recorded per pack, is then plotted as a function of the charge of the pack, to make an S-curve as shown in Fig. 1.7.

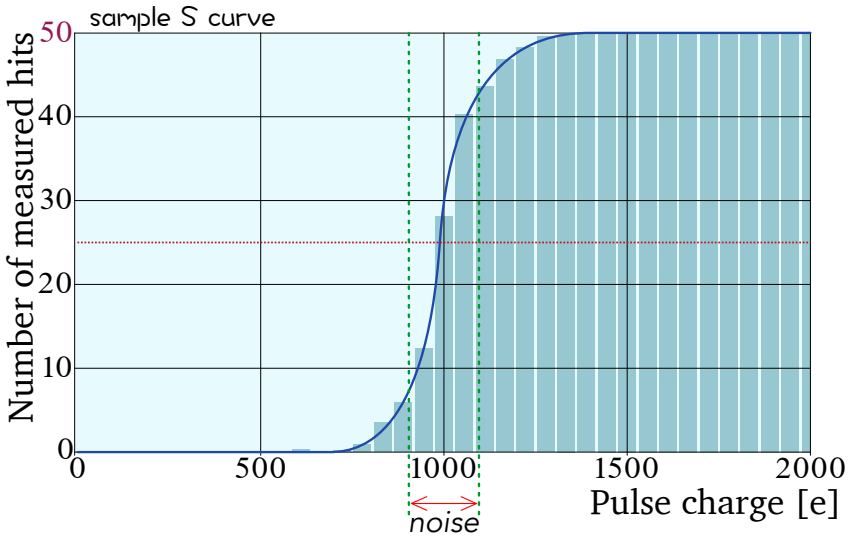

Figure 1.7: Sample S-curve for a pixel tuned at 1000 electrons.

The histogram is fit to an analytical function and the threshold and the noise are extracted from the fit to evaluate the quality of the tuning. This process is repeated for all the pixels in the matrix.

1.4 The RD53A ASIC

A prototype ASIC called **RD53A**[3] was initially designed for testing, and it is currently being used in preparation for the future production. A final version, RD53B[2], has been more recently designed and produced based on the experiences from its predecessor, and it has already been successfully tested in some institutes, including the Osaka University.

The RD53A ASIC, shown in Fig. 1.8 contains a pixel matrix of 400 columns and 192 rows in 20.0 mm by 11.8 mm, making a total of 76800 pixels. For testing purposes, the pixel matrix is divided into three sections. Each section implements a different front-end (FE) design, namely the **synchronous**, **linear** and **differential** according to how they process the signal coming from the sensor. A detailed explanation on the design of these three front-ends can be found in the RD53A manual, in Ref. [3]. Based on past tests on RD53A, the differential front-end was chosen for RD53B.

[2]ATLAS refers to RD53B as ItkPix, since RD53B was developed by the RD53 collaboration and it is not specific to the ATLAS experiment. In this thesis, however, we refer to both RD53B and ItkPix just as RD53B.

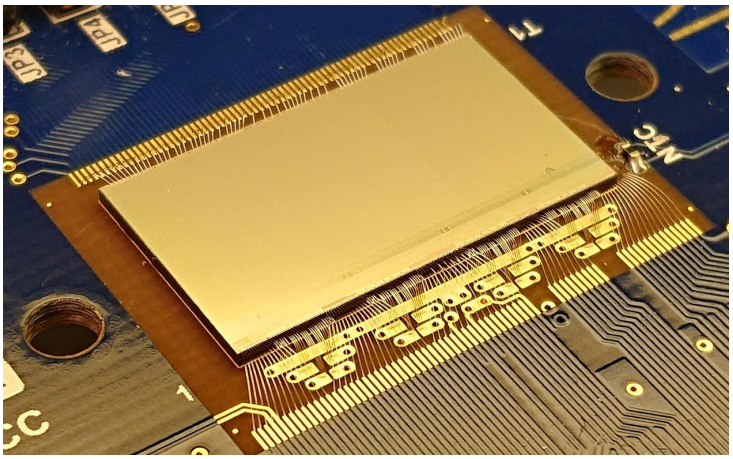

Figure 1.8: Picture of the RD53A ASIC, mounted on a card that permits its readout. Picture taken at the University of Bergen (Norway).

In RD53A, pixels are grouped in small 8×8 matrices called cores. Pixels within the same core share the same digital readout circuit and memory buffers. The output of each core is sent to the edge of the ASIC and then through the wire bonds to the PCB.

The RD53A ASIC has already been exhaustively tested. By the time this thesis is written, pixel modules equipped with RD53A ASICs (hereafter *RD53A modules*) are being assembled and tested in preparation for the real RD53B production that will take place in the future.

1.5 Issues and requirements on the module production

This section starts by describing the main safety conditions under which the modules have to be produced. It will later describe other requirements to make the production consistent among institutes and as fast as possible. It will finish by introducing the measures that should be applied in case of exception, in order to preserve the integrity of the modules.

1.5.1 Module handling

The ATLAS pixel modules have to be operated under strict conditions.

The module temperature has to be kept under 40°C during testing. Otherwise, the ASICs as well as the glue layer between the PCB and the sensor shown in Fig. 1.4 might suffer damage. The working current of the ASICS of the ATLAS pixel modules is 4.4 A, which quickly increases the temperature of the modules if they are not actively cooled down.

Also, we always have to ensure that the dew point is low enough to make the power-off operation safe. When switching off the power while actively cooling the module, the module temperature quickly drops by around 10 degrees. If it reaches the dew point at the module surface. water will start condensing on the surface, possibly causing shorts that could lead to serious damage

Finally, the whole module assembly and testing also have to be carried on inside a clean room under strict dust concentration requirements. The distance between the wires connecting the PCB to the ASICs is shorter than the average size of a dust particle (on the order of microns), so any dust falling onto the module surface might connect two wires leading to module damage or readout failures.

We need a common tool that monitors the supply voltages, currents, dust concentration, temperature, and dew point of the module. The tool must be also capable of acting in case of a problem occurs.

1.5.2 Reproducibility among modules and institutes

Around 10,000 modules have to be eventually produced and installed in the detector. Any module with defective PCB, ASICs, and sensor should be identified. Also, the module production will be split between institutes all around the world. Even though all the institutes will follow the same testing procedure, different testing tools used by different people could also introduce variations in the testing results.

The testing software should therefore be common. It should also be configurable enough to adapt itself to different testing environments, such as different power supplies, monitor systems, and monitored quantities.

1.5.3 Speed and simplicity

Many stages during assembly, such as sensor attachment to the PCB, cannot be automated and they are performed manually. However, most of the testing phase can be fully automated. This automation is necessary not only to make them faster, but also to have a precise estimation on how much time the testing chain would take. The module production chain has to be done in parallel for many modules. This allows us to organize the production to avoid any bottleneck, maximizing the efficiency of the operator's work.

The module testing software should be easy to use. It should also be autonomous enough to require human intervention only in case of exception.

1.5.4 Interlock system

Finally, we shall not overlook any failure in the module or in the software, that could lead to a readout exception, or environmental variables going outside their safety range. A hardware-based interlock is implemented in each site to stop the testing sequence if considered necessary, but a software interlock should be also integrated in the testing tool.

The testing software should then be capable of identifying any issue and acting in consequence. If the issue is solvable without human intervention, the software should do it by itself.

If the issue needs external operation, the software should safely interrupt the testing chain and wait until it is restarted manually.

1.6 Situation before my intervention

The following items were already part of the testing framework when I started working on it, and have served as a baseline for most of my contributions.

- Three databases. The first one has been explicitly built to store module information and testing results in each local institute, and it is called **localDB**[4]. The second one is **InfluxDB**[5], a widely used database to store time-ordered data (monitored quantities such as temperatures, voltages, etc). The third one is called **production database** and has been built explicitly for the module production. It is common to all the institutes and accessible by all the collaborators.
- A dedicated software used to communicate with the modules and read their output, called **YARR** (Yet Another Rapid Readout)[6], that also uploads the testing results to localDB.
- Many different scripts to retrieve data from InfluxDB, upload data to localDB and register modules into localDB.
- An independent framework to control power supplies, called **labRemote**[7].

All of these tools have their independent documentation, their independent configuration files, and their independent usage method. Setting them up and making them work together is complex, takes a long time and can lead to numerous sources of errors. Therefore, a common integration tool to simplify the process becomes a must for all the institutes.

1.7 Purpose of this research

The main goal of this research has been to build an integration tool that addresses all the issues presented in Section 1.5, making a module testing framework consistent, safe, fast, and simple to use.

1.8 Structure of this thesis

Chapter 2 describes the tools that have been developed to fulfill the requirements of the module testing. In Chapter 3 the module testing procedure is detailed, highlighting the significance of the tools we made for this purpose. The results of this development are summarized in the latter part of Chapter 3. Chapters 4 and 5 are devoted to a short discussion and conclusion, respectively.

Chapter 2

Development of Integration Tools for the module testing

This chapter starts by giving an overview of the testing setup, followed by a description of the tools that have been developed to fulfill the requirements of the module testing. All the tools described in this chapter can be found in the same GitLab repository, accessible from Ref. [8].

2.1 Overview: Hardware and Software setup

Most of the testing in Japan is currently being carried on at KEK. The subject for testing is usually a quad module. During testing, the module is placed in a cooling box where temperature and humidity are monitored and controlled by an external cooling system. We are using a commercial chiller for cooling. The cooling box, and its monitor system have been designed and produced at the Tokyo Institute of Technology. Figure 2.1 shows the cooling box, as well as part of its monitor system.

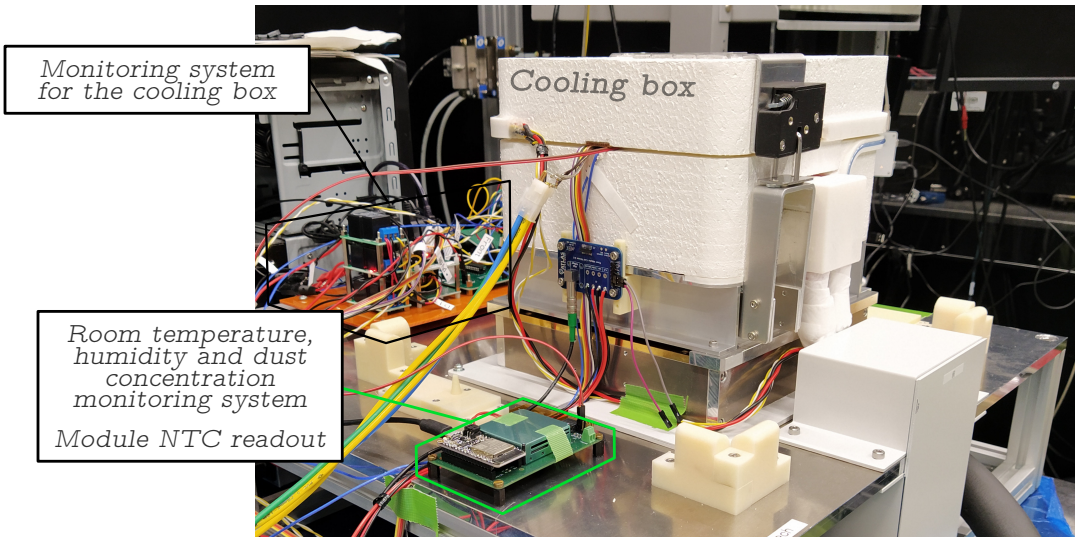

Figure 2.1: Module testing setup at KEK. Not shown in this picture are the chiller and the power supplies.

A schematic view of the module testing setup is shown in Fig. 2.2. The module gets Low Voltage and High Voltage power from two independent power supplies. The two power supplies are controlled and constantly monitored by the PC through the tool we have prepared. A monitoring board prepared in Italy, highlighted in green in Fig. 2.1, reads the output of an NTC[1] placed on the module's surface to measure its temperature. It also measures the temperature, humidity, and dust concentration of the room, and sends the data to a computer through a USB cable.

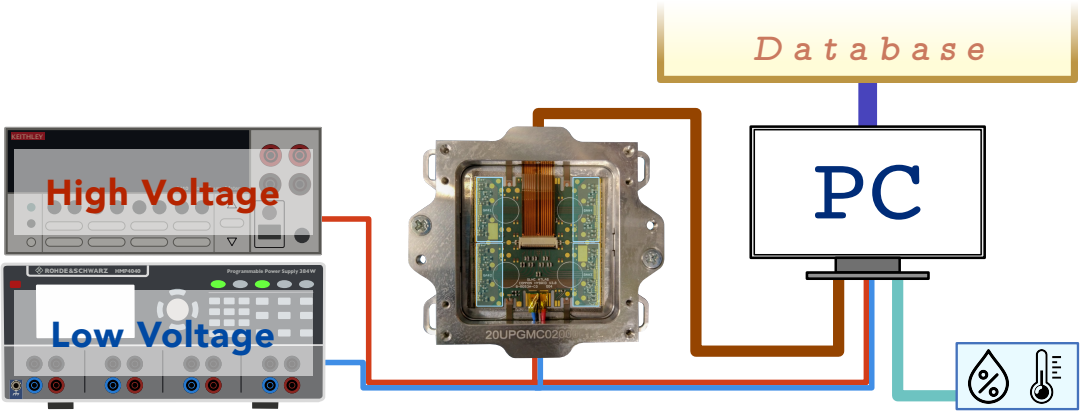

Figure 2.2: Schematic view of the setup when testing modules at KEK.

The readout data coming from the module is read though a thin flat cable, shown in Fig. 2.3, that goes to an adapter board where the data for each ASIC is sent to the PC through four independent Display Port cables. On the PC side, the data is received and processed in a FPGA with dedicated firmware, and finally analyzed by software. Ultimately, the results of

[1]NTCs are resistors whose resistance decreases as their temperature increases, normally used as small temperature sensors.

each analysis are uploaded to localDB.

The relevant power supply and environmental data are also synchronized with the analysis results in localDB. Power supply or environmental data will be referred hereafter to as **DCS** (Detector Control System) data.

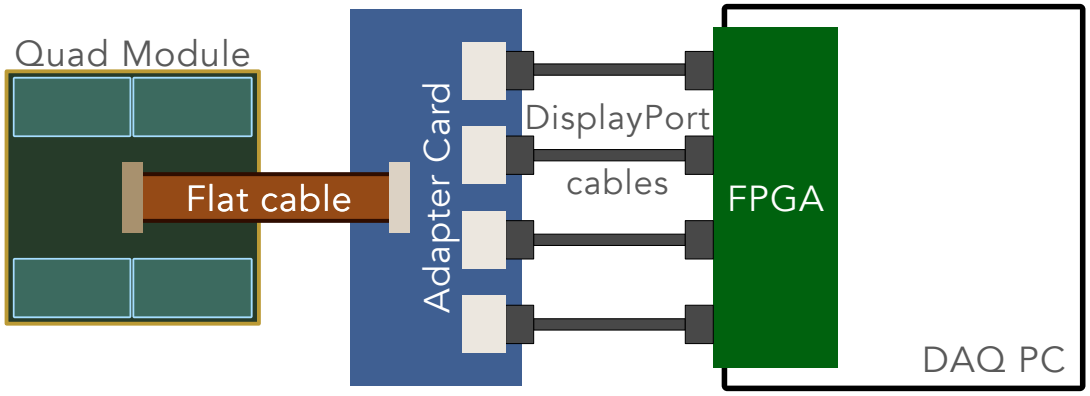

Figure 2.3: Schematic view of the setup when testing modules at KEK.

2.2 Software tools for the module testing

This section describes the software tools that have been built to facilitate the module testing. First, it describes the ones that monitor the environment and the power supply, that are constantly running during the module testing. Secondly, it describes the ones we use to look for any electrical failures in the modules before reading them out. It later describes the main tool we have worked on, which automates the module readout and handles all the organization of the readout results into the databases. Finally, the features of a tool to analyze the readout data and share the testing results are also described. The contents of each of the following subsections are shown in Fig. 2.4.

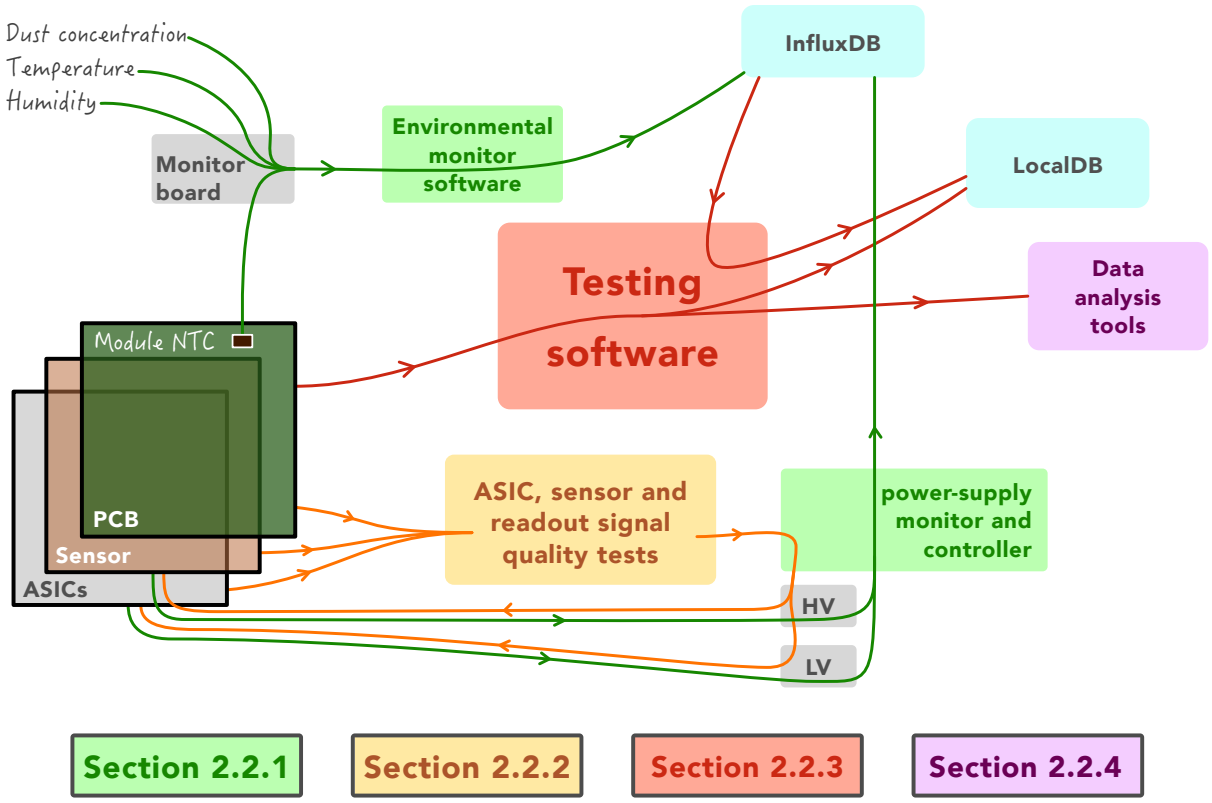

Figure 2.4: Overview of the contents of this section, in the context of the module testing.

None of the tools we have developed has a graphical interface, and all of them are used directly from the command line of the Data Acquisition computer, that runs on Linux. InfluxDB and localDB are hosted in remote computers.

2.2.1 Contributions to the environmental monitor system

Some of the most relevant contributions presented in this thesis are on the DCS monitoring.

Temperature, Humidity and dust concentration monitoring

On the PC side, I have written a software that takes the serial output of the board reading the module NTC, room temperature, humidity, and dust concentration. The software interprets the data and uploads it to InfluxDB, located in a remote server. Grafana (an Open Source real-time db-monitoring software)[9] is used to display on a screen all these monitored values, together with the output of more sensors located in the cooling box.

The monitoring software has been designed to constantly run in the background, without representing a significant CPU load for the host computer. The monitor data comes from the readout board every 5 seconds, and the software reads the USB output with the same frequency, sleeping the rest of the time.

Power supply monitor and remote controller

The power supplies (PSs) also have to be carefully monitored. For safety, their operation has to be done remotely as much as possible, instead of on the PS's buttons.

I have written a power supply monitor tool that can also be used as a PS controller. When used as a monitor tool, it runs in the background, and different instances of it can be launched in parallel to monitor multiple power-supplies at the same time. It also uploads the output of the power supply to InfluxDB. When used as a controller, it can turn on/off, and set/read current or voltage for any channel of the power supply. The control actions are given by the user in the command line. The actual communication with the power supply is done following the SCPI command language set, that is common for most test and measurement instruments. An introduction to the SCPI command set can be found in Appendix A.

Minimizing the complexity of the framework has been one of the priorities of the development, and even other software tools can use this tool autonomously to get the current [I, V] status or modify any of these two quantities. This large flexibility is the main merit of this tool.

Software Interlock and exception handling

Monitoring the DCS is not enough to keep the module safe. A software interlock should take immediate action when an event that is potentially dangerous for the module occurs.

Particularly dangerous is the situation where the operator turns off the power of the power supply even though the temperature of the module is not high enough above its dew point. To avoid this situation, we decided that the power supply should be fully controlled remotely, and that it should be done by the tool I wrote for that. When attempting a power off, the tool will retrieve the last pair of temperature and humidity from the database, calculate the dew point and check if the temperature is at least 15 degrees above it before proceeding. If that is not the case, a message like the one shown in Fig. 2.5 is printed to the terminal.

```
[ info ][ps] T = -13.50; dp = -24.87
[ warn ][ps] (T - DP) = 11.37, which is smaller than 15.
             Cannot power off the LV in these conditions
```

Figure 2.5: Sample error message shown when trying to power off the Low Voltage in a dangerous situation.

There might be cases where we have to power off the module even though the software thinks we should not. The most common example is an accidental disconnection of the NTC measuring the module temperature. Such disconnection would be interpreted as if the resistance of the NTC was very high, which corresponds to a much lower temperature than the actual one, and would make the software think that we should not power off the module. These particular scenarios are relatively easy to identify, since the readout temperature on the "NTC disconnected state" stays always constant around -90°C, much lower than the minimum reachable by the cooling system. The issue is addressed by offering an option to forcibly turn off the power.

An even more important issue derived from a bad NTC readout comes from the fact that the cooling system actively heats up or cools down the module to bring it to its setting temperature, based on its current temperature which is retrieved from the database. Once in the past, an accidental NTC disconnection made the cooling system recognize that the module was actually at -90°C, so it tried to heat it up to its setting value, actually bringing it to more than 80°C. To avoid such an incident in the future, a trigger is fired when the readout module temperature is lower than -50°C, which is unreachable by the cooling system. The trigger raises a flag in the database. The cooling system reads the flag and turns off the feedback mode and keeps the cooling power constant until the issue is manually solved.

Finally, the temperature and dew point are also read out at the beginning of every stage of the testing, and the testing software does not proceed unless the two are within safety ranges. Pausing the testing in this scenario does not have any benefit for the module, but allows for a smoother restart in case the power is externally cut for safety.

2.2.2 Development of tools to check the Module's built quality

Whenever we receive or produce a new module, there are some preliminary tests we have to run on it to check its quality, before proceeding to the actual module testing.

The IV and VI curves

Making IV and VI curves, being V voltage and I current, is a quick way to check for electrical failures of any electronic component. A "VI" curve is taken by varying the current from 0 to the component's working point, while measuring the voltage. The "IV" curve is taken by ramping voltage while measuring the current instead. An illustration of these definitions is represented in Fig. 2.6.

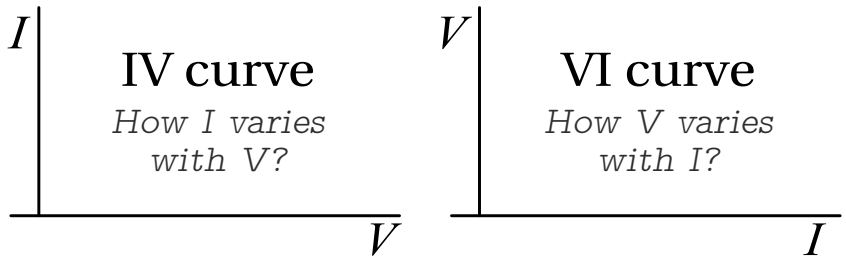

Figure 2.6: Graphical explanation of VI and IV curves

During the module testing, the VI curve is taken for the ASICs, varying the current from 0 A to the working point of 4.4 A, usually corresponding to around 1.5 V. For the sensor, the IV curve is taken by ramping its high voltage to its working point, usually -100 V. Once taken, the IV and the VI curves are uploaded to localDB.

The tool to build VI curves does not necessary take measurements of V right after changing I, and the equivalent is true also when building IV curves. Instead, the tool waits for a certain

amount of time for the system to stabilize, and then it takes some consecutive measurements to compute the average and the standard deviation. By taking repetitive measurements, we minimize the effects of small noise fluctuations. How much the software should sleep, and how many measurements we need to take to palliate the effect of noise, depends on the amount of noise coming from what is being tested and from the environment. Therefore, each institute should decide on their own configuration after studying the stability of their readout setup. An example of the configuration to take an IV curve is shown in Fig. 2.7.

```
{
    "name": "HV",
    "initial_voltage": 0,
    "final_voltage": -150,
    "step_size": -5,
    "measurements_per_step": 5,
    "sleeping_time_per_step_s": 3,
    "current_protection": 0.0001
}
```

Figure 2.7: Configuration options to build an IV curve. "HV" is a label for the high voltage power supply, defined in another configuration file. The plots shown in figures 2.8 and 2.9 have been taken with this configuration.

Figure 2.8 shows an example of the IV curve taken from the sensor, and the actual monitored current read from it. Time is represented in the x axis, and each step corresponds to a different voltage. Significant current drifts can be observed in most of the steps.

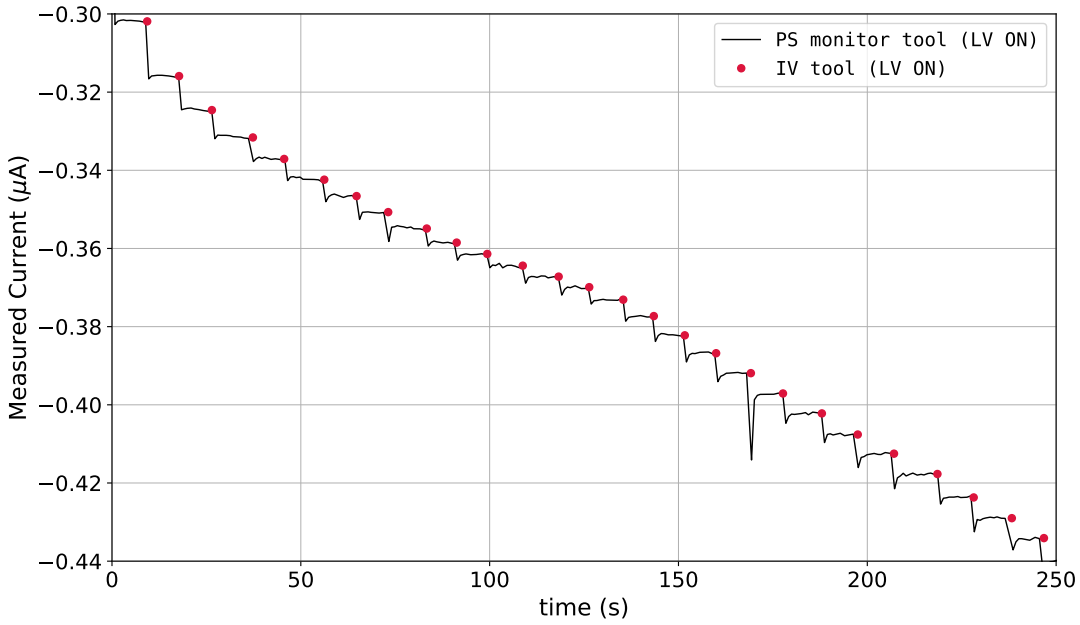

Figure 2.8: Results from an IV scan, where the horizontal axis is set to *time* so the results can be compared to the monitored values. The red dots represent the current measurements by the IV tool. The black curve represents the monitored current over time, taken independently with our PS monitor tool.

For the first three seconds of every step, our tool waits for the data to stabilize, and after

that it takes 5 repeated measurements, separated by 0.2 s plus the PS's latency, and computes their average, shown with red dots in Fig. 2.8, and their standard deviation.

When powering the module at its working point (4.4 A), its temperature easily rises by 10 to 20 degrees. The electrons in the sensor's material then gain energy, which makes easier for them to jump to the conduction band and produce current, as they would do when excited by a charged particle. The IV curve changes accordingly, as shown in Fig. 2.9. These variations are safe unless they make the sensor's break down voltage[2] move below its working voltage. Figure 2.9 shows that the break down voltage of the sensor is larger than -150 V, far away enough from its working point of -100 V. A failure in the cooling system causing the temperature of the sensor to rise too much while being powered could damage the sensor, and has to be avoided.

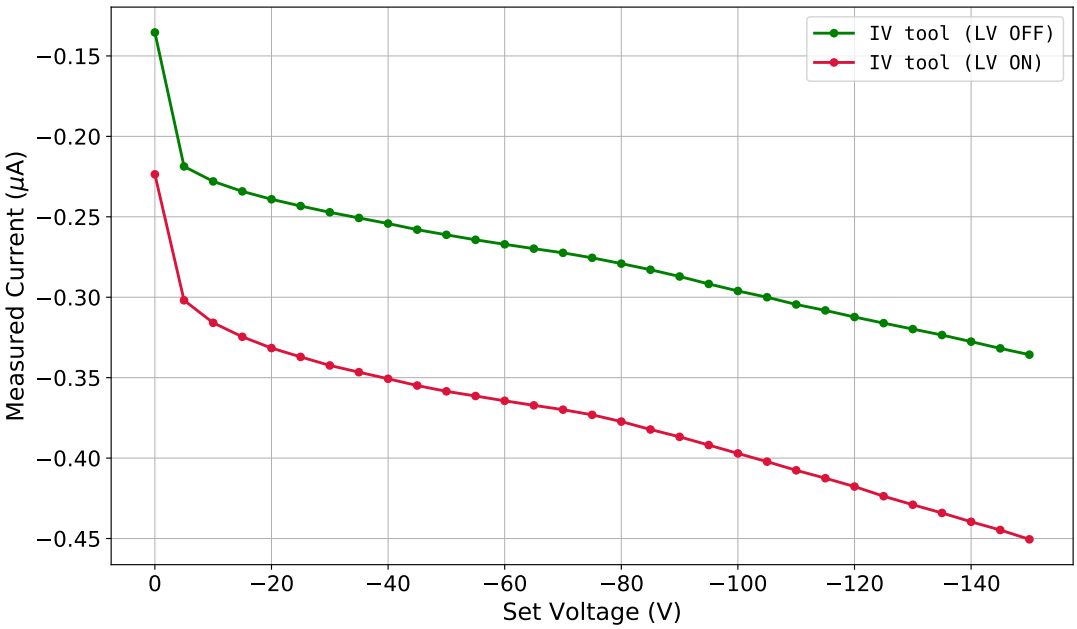

Figure 2.9: IV curves, taken while powering up the ASICs at 4.4A (red) and while keeping it powered off (green).

Other than showing the results tabulated in the terminal, this tool also writes the them to a file that can be uploaded to the module database, and optionally to InfluxDB.

Module parameter optimization

The signal propagating from the module to the FPGA can be attenuated or get some noise due to nearby electronics, or due to a bad shielding of the data cables, which could lead to corrupted data. Two parameters are usually set to the ASICs to compensate this effect. One parameter emphasizes the signal's amplitude and the other emphasizes the high frequency component of the signal, as shown in Fig. 2.10. The optimal value of these two parameters highly

[2]Increasing too much the sensor's voltage will eventually lead to a drastic increase of its current. At that point, we say we have reached the *break out voltage*. This phenomenon takes place at the edge of the sensor, due to manufacturing imperfections that lead to very high electric fields. Different sensor units might have different breakout voltages.

depends on each testing site's setup, and has to be determined before testing the modules.

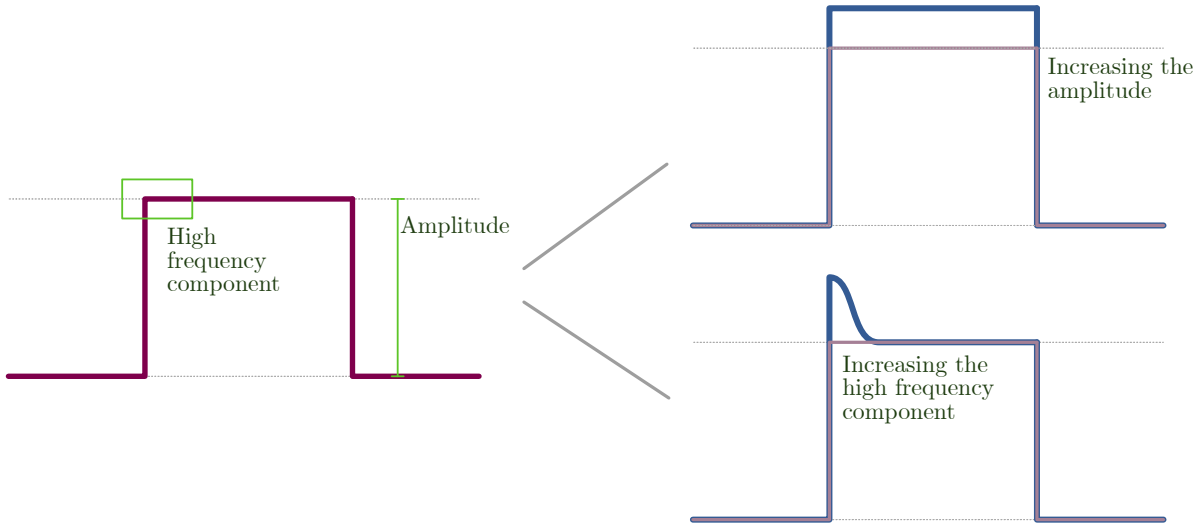

Figure 2.10: Schematic illustration of the meaning of increasing the amplitude or the high frequency component of a digital pulse.

Finding the best settings manually would require a long time, and it would not be as precise as doing it automatically. Under this motivation, we have written a tool that automates the whole process and requires no intervention from the operator.

Assume that we want to find the best working point for the parameters that emphasize the high frequency component p_{hfreq} and the amplitude of the pulse p_{amp} in the space $p_{hfreq} \in [0, 400]$ and $p_{amp} \in [200, 800]$. The quality of the point under test is measured by a function that returns the number of unresponsive pixels[3] from the total of 76800 the ASIC consists of. If the software fails to even communicate with the module, the output would be "-1". A sample result of this script would look like the one shown in Fig. 2.11. The zone filled with zeros will be more or less wide depending on the amount of noise of the testing setup. If the region is wide enough, the testing parameters are chosen to be the one closest to the defaults.

[3]Unresponsive pixels are not necessary broken. The readout failure usually comes instead from high noise or signal attenuation between the ASIC and the readout computer.

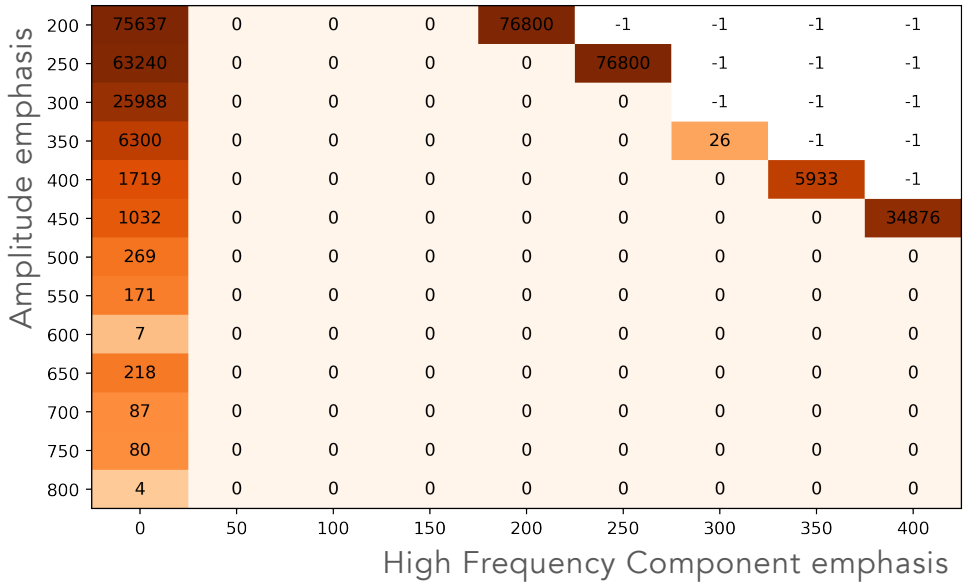

Figure 2.11: Sample result of the script described in section 2.2.2. The number of unresponsive pixels is shown for all the parameter combinations.

To build the matrix in Fig. 2.11, our tool will start by preparing an empty matrix of parameter combinations, which is shown in the terminal. Among all the possible pairs, it will then choose three random points and compute the result at them. In the example shown in Fig. 2.12, the three of them gave a perfect result. The result of these three points is used as a reference to guess where the best working zone could be.

```
haxis: p1; vaxis: p0

 #   0   50  100  150  200  250  300  350  400
200
250
300
350
400                        0.0
450
500
550
600
650     0.0
700
750
800                        0.0
```

Figure 2.12: Output to the terminal after the three first results, got by randomly choosing p_{hfreq} (p1 in the figure) and p_{amp} (p0 in the figure).

From that stage, the script will take the last three measurements and find the one which gave the best result. If all of them gave a communication failure ("-1"), a new random point is found. In other case, the next testing point is chosen randomly around the minimum of the three previous results, within a radius of 1 position. If all the points 1 position away form the best one have already been tested, the radius is incremented by one unit and a new search is performed.

```
haxis: p1; vaxis: p0

 #      0      50  100  150    200      250    300   350   400
200  75637.0  0.0  0.0  0.0  76800.0    -1.0   -1.0
250  63240.0  0.0  0.0  0.0    0.0    76800.0  -1.0
300  25988.0  0.0  0.0  0.0    0.0      0.0    -1.0
350   6300.0  0.0  0.0  0.0    0.0      0.0    26.0  -1.0
400   1719.0       0.0         0.0      0.0
450
500
550
600
650           0.0
700
750
800                                     0.0
```

Figure 2.13: The matrix is getting filled around the valley of 0's.

Thus this algorithm will first find the best working region in the matrix and then expand from there, showing a new matrix in the terminal after every step. Therefore, if a good working point is already clear in an intermediate state there is no need to wait until the whole matrix gets filled and the scan can be manually terminated to save time.

Finally, to diminish random contributions from noise, each pair of values can be tested multiple times and the number of unresponsive pixels can be computed as an average. The real-time results of this script are shown in Fig. 2.14. If needed, they can be passed to a plotter to present them in a cleaner way, as shown in Fig. 2.11.

```
haxis: p1; vaxis: p0

 #      0      50  100  150    200      250    300    350      400
200  75637.0  0.0  0.0  0.0  76800.0    -1.0   -1.0   -1.0     -1.0
250  63240.0  0.0  0.0  0.0    0.0    76800.0  -1.0   -1.0     -1.0
300  25988.0  0.0  0.0  0.0    0.0      0.0    -1.0   -1.0     -1.0
350   6300.0  0.0  0.0  0.0    0.0      0.0    26.0   -1.0     -1.0
400   1719.0  0.0  0.0  0.0    0.0      0.0    0.0   5933.0    -1.0
450   1032.0  0.0  0.0  0.0    0.0      0.0    0.0    0.0    34876.0
500   269.0   0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
550   171.0   0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
600    7.0    0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
650   218.0   0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
700    87.0   0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
750    80.0   0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
800    4.0    0.0  0.0  0.0    0.0      0.0    0.0    0.0      0.0
```

Figure 2.14: Results after all the pairs have been tested, shown in the terminal once the script finishes.

2.2.3 Development of a tool to automate the module testing

The `scan-operator` acts as a wrapper around YARR, labRemote, the monitor software and the databases. It automates all the testing workflow, organizes the configurations and handles the

data storage into the databases.

Testing workflow automation

The module testing has many stages (module registration into localDB, module readout, DCS and module data synchronization in localDB, etc.), and requires in principle the usage of different tools that are independent to each other, as described in Section 1.6. The `scan-operator` automates most of the work, according to the procedure explained in this Section.

After assembling the modules, and even before starting the testing, they have to be registered in the production DB which is accessible worldwide. The module information should then be retrieved from the production database to localDB. This process has to be done by hand. The flow of information through databases is shown in Fig. 2.15.

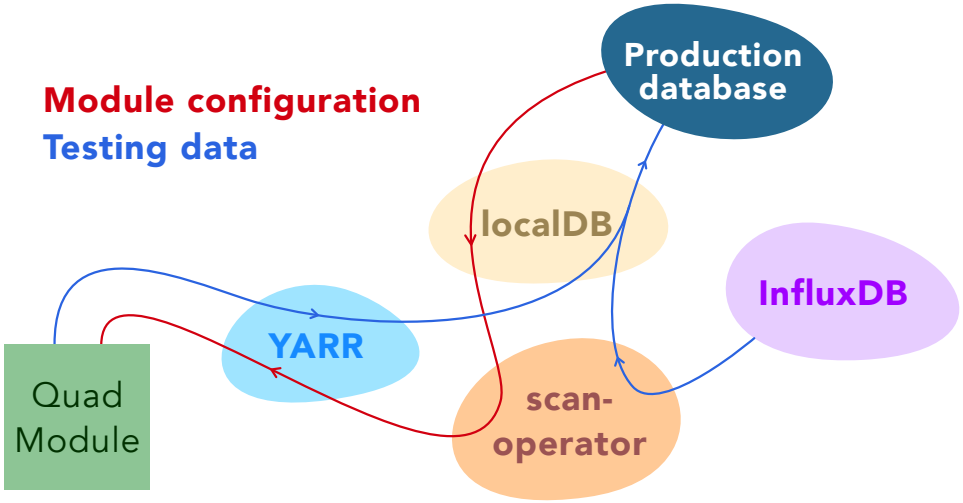

Figure 2.15: Data flow through databases. The module configuration is first retrieved from production database. Once all the results are merged and ready in localDB, they are uploaded to the production database.

During the testing, the configuration for the module under test is retrieved from the local database, which is done automatically by the `scan-operator`. The module is then configured according to it. Each ASIC has more than 150 configurable parameters, and each module contains four ASICs. Most of them are always left in the defaults, but sometimes a parameter has to be modified. The `scan-operator` provides an interface to do it in a much simpler way than opening and editing manually the large ASIC configuration files. The design of this configuration goes as follows:

1. The configuration for a module is specified in a single file, following the structure presented in Fig. 2.16.

```
"modules":
{
    "QUAD_SERIAL_NUMBER":{
        "chipType": "RD53A",
        "chips":[
            {
                "chip1_parameter_1": 123,
                "chip1_parameter_2": 34,
                ...
                "chip1_parameter_n": 583
            },
            {
                "chip2_parameter_1": 42355,
                ...
            },
            {chip3},
            {chip4}
        ]
    }
}
```

Figure 2.16: Scheme of the module configuration file.

2. In the chip (chip meaning ASIC) configuration section, the user specifies only the parameters that need to be modified from the defaults.

3. The `scan-operator` will look for those parameters in the actual ASIC config files, which are hidden to the user.

4. If the parameters are found, they are modified according to the `scan-operator`'s configuration. Otherwise a warning is printed and the parameter is ignored.

Just by looking at a single file, the user perfectly knows how the module is configured, without having to open four big files and remembering which parameters were changed and which were not. Sharing the configurations also becomes much easier when they are summarized into a single file. This is one of the main appealing points of the `scan-operator`.

Once the configuration is ready, and before start operating the module, the `scan-operator` starts monitoring the power supply. There are two ways of doing it. The first one is by having the script I wrote for that constantly doing the job in the background (described in Section 2.2.1), as it was the case in Figs. 2.8 and 2.9. The second one, in case we only want to store PS data while the tests are ongoing, the `scan-operator` can autonomously run the same monitoring script just before starting the tests, and kill it once the module readout is finished.

Once the DCS monitoring is working, we are ready to start the tuning procedure, consisting on a sequence of steps written in one of the `scan-operator`'s configuration files. The `scan-operator` is the responsible of calling Yarr, that actually communicates with the ASICs. Once finished, the relevant results are uploaded to the local database.

In the case of the actual production, once the tuning sequence is finished, the tuning results are stored in localDB. Once the operator checks that all of them are correct, they are uploaded to the production database together with the module configuration, making them public for

everyone.

Structure and organization

The `scan-operator` is meant to simplify the module testing process, so it needs to be simple to use, and the documentation has to be clear and concise.

The whole framework is stored in the same GitLab repository[8], and all the documentation is written in a single `README.md` file. However, numerous error scenarios like the one shown in Fig. 2.18 have been implemented in all the scripts, so referring to the documentation becomes useful only for completeness. The intended user experience behind the design of this software is summarized by the following three points.

1. The user first sees what the software can do by looking at its command line options, as shown in Fig. 2.17.

```
./ScanOperator.sh
Help:
  -m [str]   : Serial number of the module
  -W         : upload Yarr's results to localDB (scanConsole's -W)
  -o         : turn on / off the output of the PS before and after all the scans
  -e         : Check if T and H are within a setting range before running each scan
  -d         : Monitor the PS and upload the DCS data to influxDB.
  -t         : Move DCS data from InfluxDB to LocalDB after each scan.
  -c         : Create / override Yarr's config files for your chip.
  -s         : Don't look for updates in the config (saves time)
  -j         : Just update the configs, and exit before running any scan
  -q         : "quiet" mode (cleaner output).
```

Figure 2.17: List of options available in the `scan-operator`.

2. After looking at the available options, the user can use the options he/she considers convenient, without having to know what configuration is required or where it has to be specified.

3. If any option, configuration, or command is incorrect or missing, the tool itself will tell the user what is wrong and how to fix it. The example shown in Fig. 2.18 is one among many of the same kind that have been implemented in the software.

```
$ ./ScanOperator.sh -m "QUAD08"
---------------------------------
[ error ][so] No module ID "QUAD08" found under the entry "modules" in /daq/so/configs/so_modules.json.
[ info  ][so] Only the following module IDs are currently defined in the config file:
-   example_scc
-   example_quad
[ info  ][so] Please add yours (or rename one of the above) and try again.
```

Figure 2.18: The user wants to operate a module "`QUAD08`", but does not know where to write its configuration. The user then just runs the `scan-operator`, which points him to the configuration file that has to be modified.

Efforts have been also put into well commenting the code, so understanding it becomes easier for future developers of the tool.

Also following this simplistic philosophy, the `scan-operator` uses many small configuration files, a single one for every purpose. The `scan-operator` has many options, and very rarely a user would want to use them all. This reduces the amount of code the user has to look at before using a particular option of the software, making the user's experience more pleasant.

Organization of the results

YARR produces one folder per readout attempt, but these are not organized into any particular way into YARR's output directory. The `scan-operator`, however, takes all the readout attempts corresponding to the same testing sequence and puts them into a common folder, named according to the date it was created. Inside that folder, a file indicating the time consumed by every stage of the module tuning/readout procedure is also created. This is useful to have a quick overview of how much time the sequence took to finish, that would have been difficult to compute in other case.

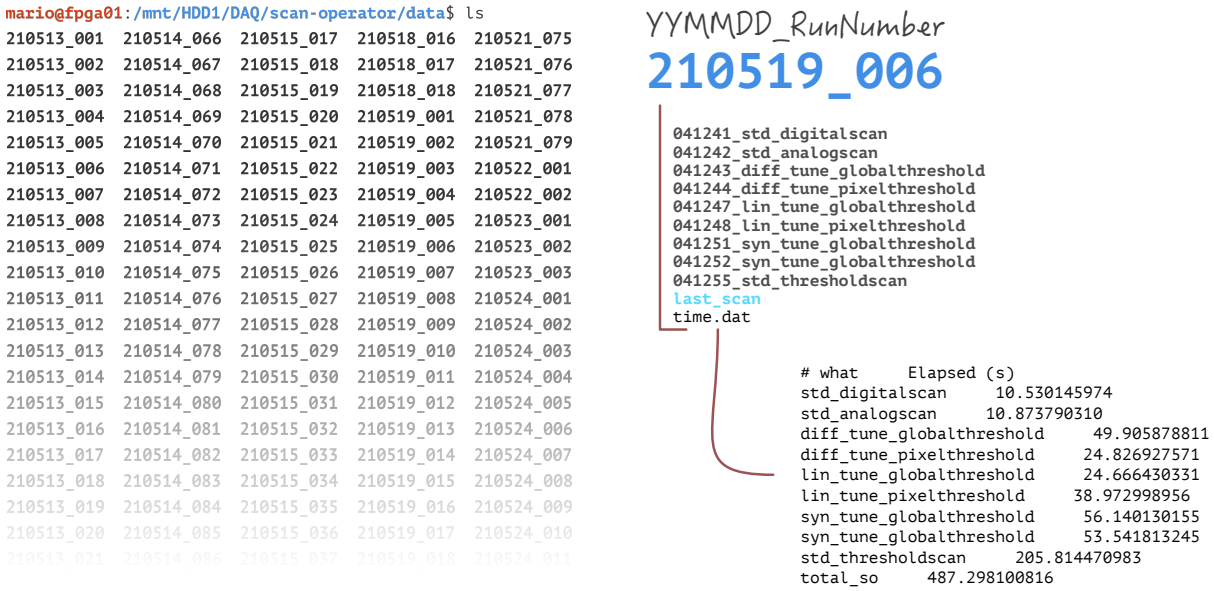

Figure 2.19: The testing results (`041241_std_digitalscan`, etc.) are stored inside one folder per scan sequence (`210519_006`, etc.). The elapsed time per scan is also calculated.

Integrated interlock system

There are a few features included in the `scan-operator` that simplify and make the module testing more secure. These are triggered by different events.

- Sometimes, a few very noisy pixels might overflow memory buffers and cause data corruption in the ASIC side. In this case YARR will either crash from a segmentation fault or report the error and exit. We can have a rough estimation on the frequency with which these corrupted data-related errors occur, which depends mainly on the amount of noise coming from the readout setup. However, we cannot predict when they are going to stop the sequence.

The status of the module is saved after every stage of the testing sequence, so if a sudden software crash happens in the middle of one stage, just restarting the process at the point it was interrupted is enough to get over the issue. Doing this by hand takes some time, to what we have to add the time from the point the software crashes until we actually notice it. The `scan-operator` re-runs the scans for us in this situation, up to a maximum number of attempts before interrupting the testing permanently, for safety.

- In the scenario of a critical error or a interruption signal from the user, the software will exit safely, restoring the terminal to its original state and closing all the child processes to leave the environment exactly as it was before the run.

2.2.4 Development of data-analysis tools

YARR generates some plots automatically as a quick overview of the test results, but the data are also generated to be used for a more extensive offline analysis.

Currently, localDB automatically produces a predefined set of plots from the readout data of every tested module in the database. These plots are produced dynamically when the user access the database from an internet browser. This approach saves memory, but increases considerably the amount of time the web page takes to load. LocalDB is also usually *local*, and the localDB in one institution is usually not accessible from outside.

LocalDB is not intended to be used as an analysis tool or to share plots between institutes. In order to facilitate these actions, I have built a small analysis framework that produces plots that can be uploaded to any server, where anyone can not only view them but also interact with them.

Offline data analysis framework

This analysis tool takes the results coming from YARR as input. It then builds a class where all the pixels in the matrix are registered together with their coordinates, measured threshold, measured noise, etc. Once all the information has been read, the histograms requested by the user are created. These histograms can be classified into three main categories:

- 2D histograms, of either the entire ASIC or all its cores stacked onto each other. The Z axis corresponds to one of the attributes of the pixels. All the pixel cores are clones of each other, so the behavior of a pixel usually depends more on its relative position inside its core, rather than on its location in the whole matrix.

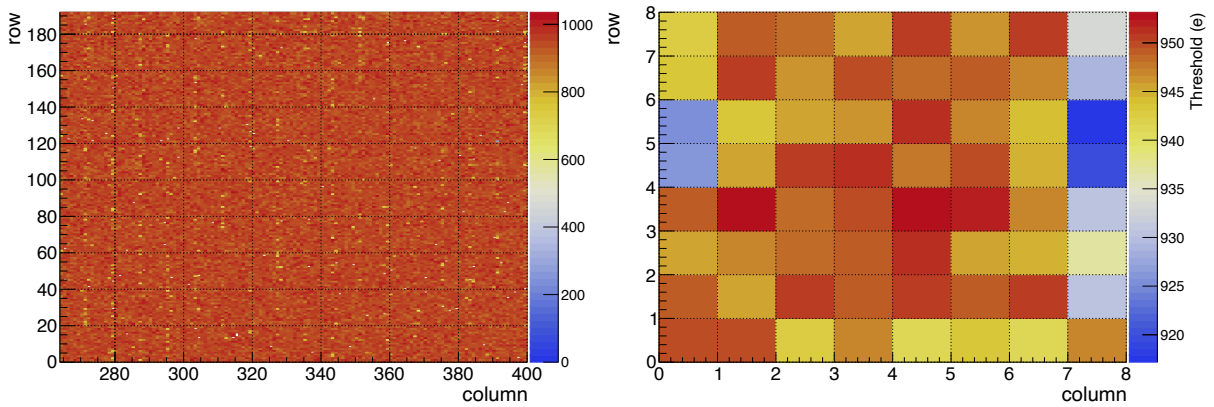

Figure 2.20: Threshold map in the whole differential FE (left). Threshold map of all the differential FE's cores stacked. The target threshold in both cases is 950 electrons.

The plots in Fig. 2.20 show the threshold distribution of the pixels in the differential FE, after tuning. When stacking all the pixel cores and plotting the average threshold, it is clear that some pixels in the right side of the cores have lower threshold than expected. The discrepancy between their threshold and the target one is visibly larger than the threshold dispersion shown by the rest of the pixels in the matrix. This kind of plots were useful to identify the cause of the issue in RD53A, which has already been corrected in the design of RD53B.

- 1D histograms: This allows us to build a histogram on a selected property of the pixels (noise in Fig. 2.21), separating in the tuned and untuned pixels in the distribution. A pixel is categorized as tuned if its measured threshold agrees with the target threshold within three sigmas[4]. An example of such a threshold distribution is shown in Fig. 3.2.

[4]Sigma refers to the standard deviation of a Gaussian. The value of sigma is taken from fitting to a Gaussian the threshold distribution in the tuned state.

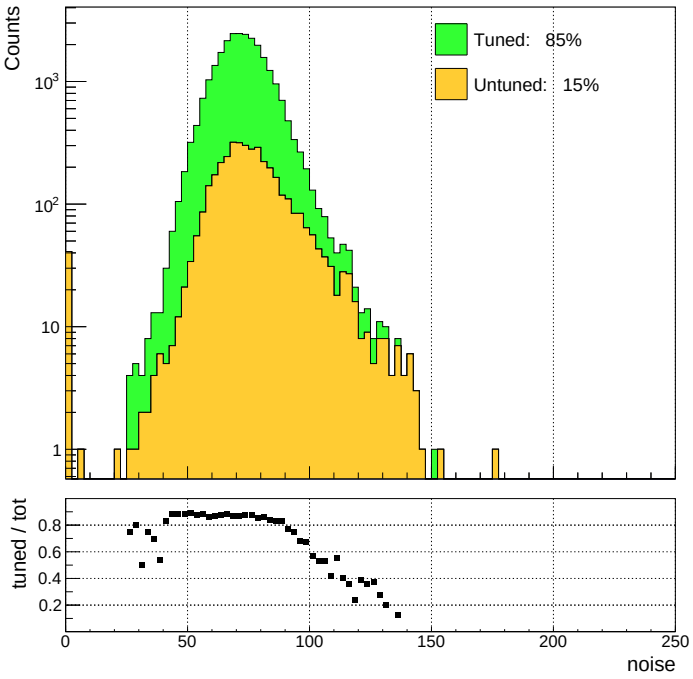

Figure 2.21: Sample noise distribution in a tuned state. Noise 0 is by convention associated to the pixels whose threshold measurement failed.

Pixels with higher noise are intuitively more difficult to tune, and therefore more likely to be untuned. However, the actual relation between the tuned state and the noise of a pixel is not clear until we build plots like the one in Fig. 2.21. More specifically, the lower part of it represents the ratio "tuned/total" as a function of the noise. This allows, for example, to set a quality requirement on the pixels based on their noise. We could define "normal" pixels the ones with the noise below 100, and define "noisy" pixels the ones above that level, that are more than 50% likely to be untuned. Many other quantities can be plotted instead of noise, to get similar conclusions.

- Correlation plots:

  Finally, we are able to select any two characteristics of the pixels and plot them against each other in a 2D histogram, as shown in Fig. 2.22.

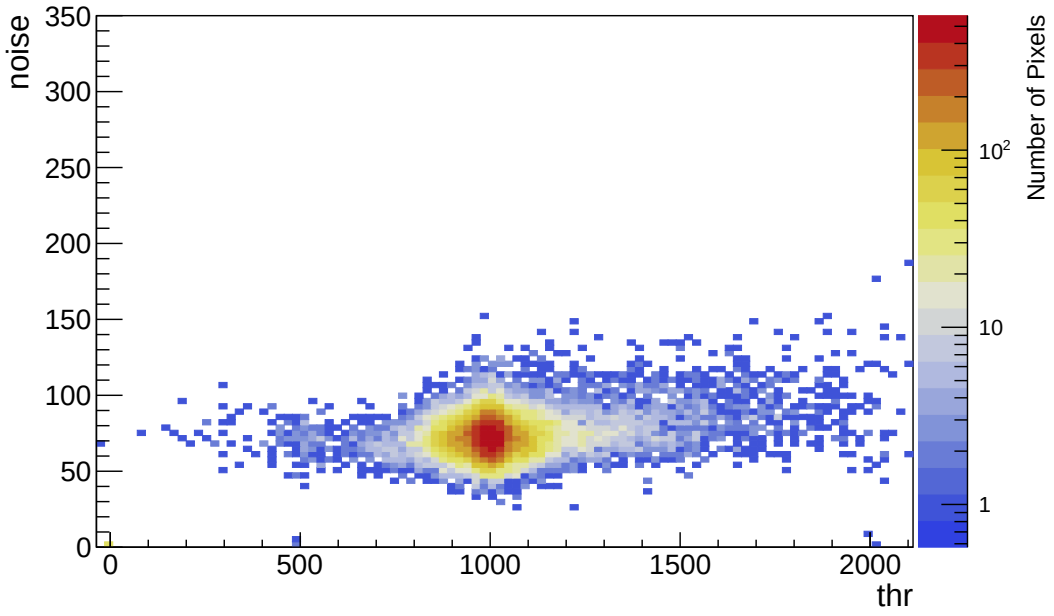

Figure 2.22: Noise and threshold plotted against each other.

The main point of this tool is that it allows us to easily build many types of histograms from any object than can be created inside a Python class.

The results on the Web

All of these plots are produced offline and organized in folders. A simple bash script is used to upload all of them to a web server, and all the `index.html` files are automatically generated accordingly, being linked to each other and reflecting all the contents of each folder as shown in Fig. 2.23.

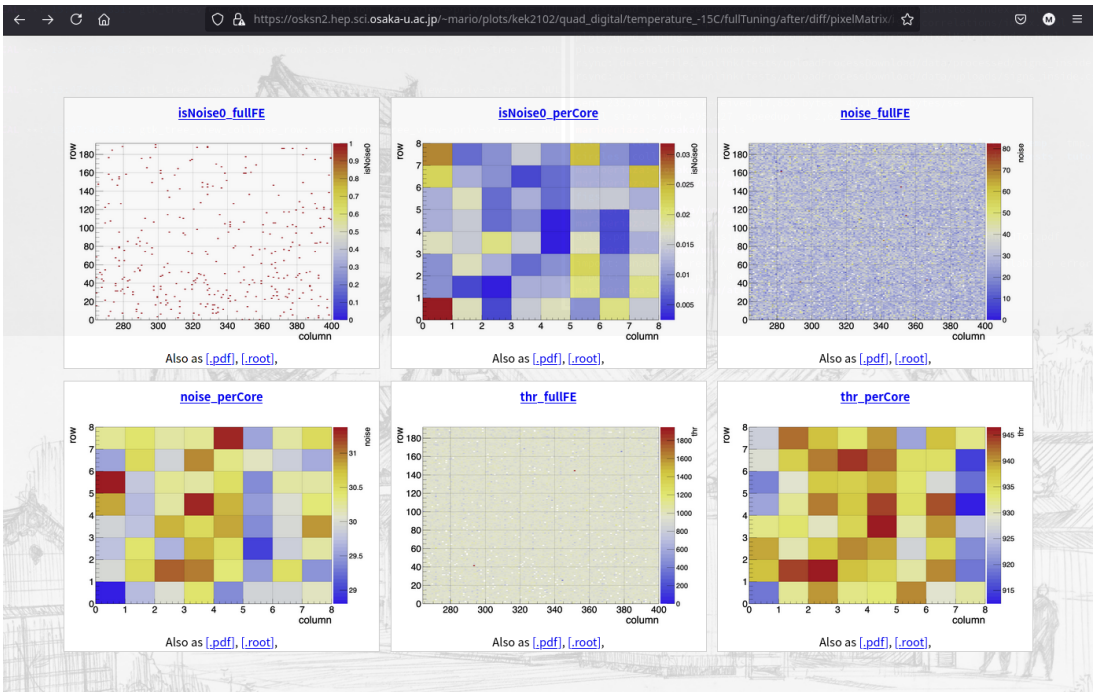

Figure 2.23: This web page has been automatically generated from a folder in my local computer.

The plots in Fig. 2.23 are shown in the browser in `.png` format, and the time this page takes to load is negligible as it requires almost no computing. The analysis tool producing the plots also allows us to export them in `.pdf`, `.root` and `.txt` (raw data). If the same plot is located in the same folder in multiple formats, those will be shown under the figures as it can be seen in Fig. 2.23. Clicking `.pdf` will show the pdf version of the above `.png`. Clicking `.root` will show an interactive version of the plot based on JSroot (JavaScript Root)[10], allowing to zoom it, view the contents of each bin, modify any visualization option, and re-export it in multiple other formats[5].

This tool also makes the results easy to share with anyone, and the interactive visualizations do not have any special software requirement on the user's machine.

Timing plots

Automating and accelerating the module testing loses its meaning if other production stages take longer and become a bottleneck in the module production chain. This is a situation we really want to avoid. For that, we need a time estimation on how much it takes to complete the testing stage.

The timing information that the `scan-operator` outputs can be used as the input of another script that uploads it to the web. A modified version of a tool which is currently used for a similar purpose at the CMS experiment, is used to produce interactive plots like the ones show in Fig. 2.24.

[5] An example interactive histogram based on JSRoot can be accessed from `https://root.cern/js/latest/?nobrowser&file=../files/hsimple.root&item=hpxpy;1&opt=COLZ`

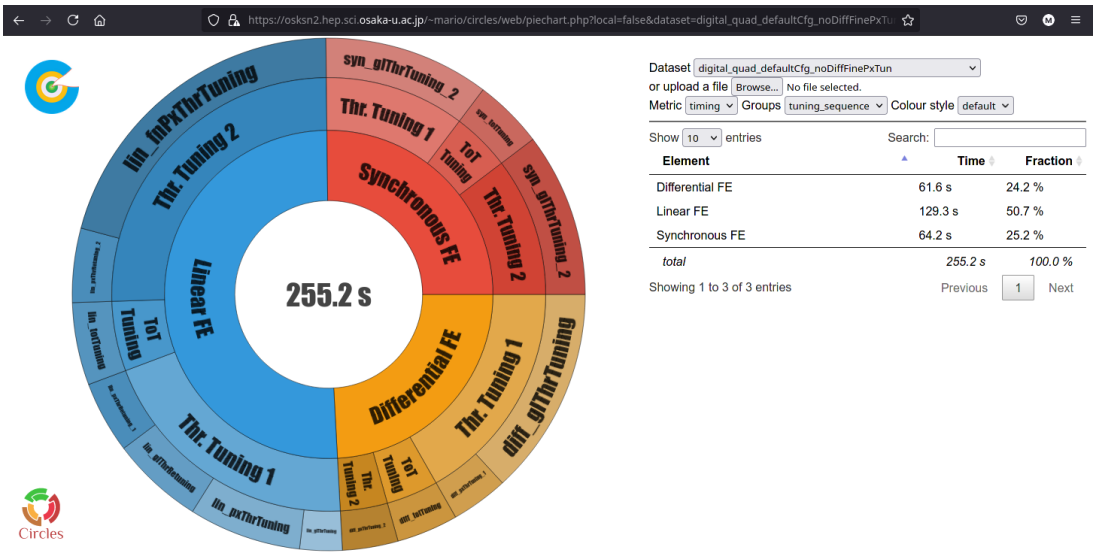

Figure 2.24: Time taken to completely tune the three front-ends of one ASIC.

The plot in Fig. 2.24 shows the time consumed by every stage in the tuning sequence, broken down into the three front ends the ASIC consists of. It is an interactive plot, where we can expand every section or hover over it to get the time it particularly took. These plots allow us to have a quick visual comparison of all the tuning stages, to quickly know which one we should work on if we want to accelerate the tuning sequence. In the above plot, the algorithm called "`lin_fnPxThrTuning`" was slowing down the entire tuning process. After comparing the tuning results with and without it, we decided not to include it in the tuning sequence, saving more than 50 seconds per module.

Chapter 3

Results: Significance of my contributions in the Module testing flow

This section summarizes the module testing, highlighting the importance of my contributions in the process.

3.1 First steps after assembly

IV and VI curves

The first thing we do once the module is ready to be tested is taking its VI curve. We do it using the tool I wrote for that, and then we compare the results to what we expect from the experience with other modules. We also take the sensor IV, ramping this time the voltage from 0 to -150 V. We should expect the current to stay low all the time.

Any result out of the ordinary when preforming an IV or VI scan would be a sign of a powering issue. The issue could be originated by damage in the sensor or in the ASICs, which is not reparable, but it could also come from broken wire-bonds or damaged components in the PCB, which are replaceable.

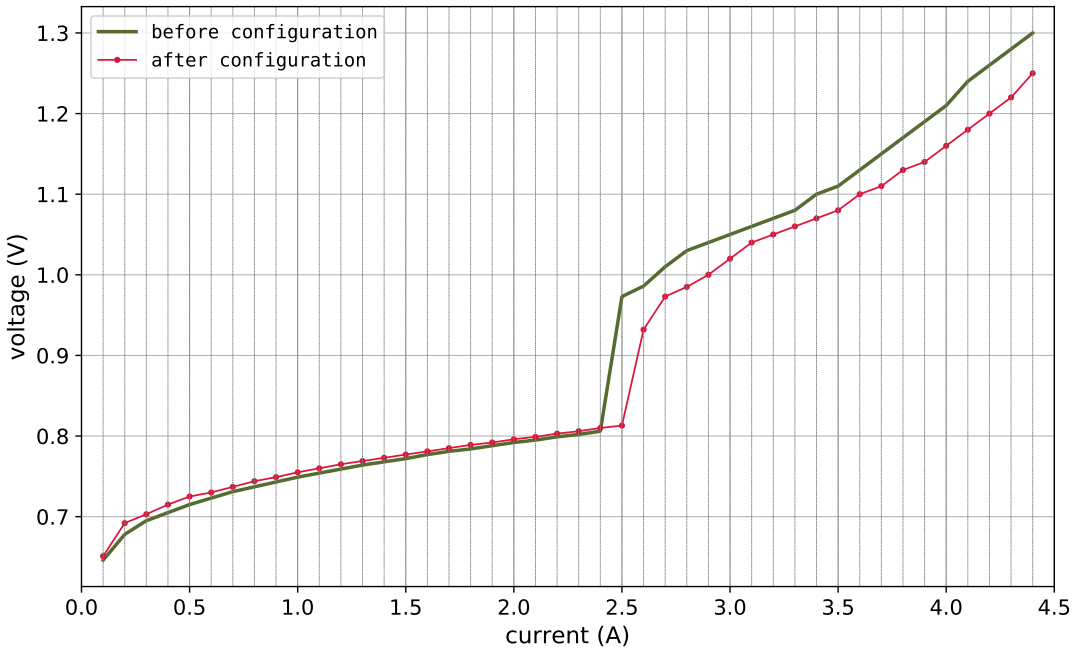

Figure 3.1: Before reading out modules, they are configured by YARR. This is done by modifying some registers in the ASICs. Here two VI curves are shown for a damaged module, before and after configuring it. An ASIC is taking more current than what it should, lowering the voltages to a level that makes the readout fail.

The module from where Fig. 3.1 was taken presents an abnormally high current consumption over one single ASIC. The problematic ASIC was identified using a thermal camera, since the increased current made its temperature rise above the other three ASICs. In a normal scenario, each ASIC would consume 1.1 A from the total of 4.4 that are supplied to the module, and no significant temperature difference would be observed among them.

The solution for this module was to completely disconnect the problematic ASIC, so the three other ASICs could be normally used at 3.3 A.

Output signal enhancement

We use the tool described in section 2.2.2 to find the most adequate shape of the output signal from the module. This configuration should work for all the modules tested in the same testing site, but this procedure has to be repeated in all the sites. After this stage, the module is ready for tuning.

3.2 Module tuning

The tuning is done by the `scan-operator`.

The target threshold is decided such that it is larger than the mean noise level, but not larger than most of the real hits. The quickest and most simple way to decide on a good threshold is to tune a module to various threshold candidates and perform multiple readout tests under

x-ray irradiation. A threshold too low would give a noisy result, whilst a threshold too high would give a low number of recorded hits. An example of the results after a good choice for the threshold is shown in Fig. 3.4.

The tuning starts by scanning the ASICs to find any damaged pixels, that would behave as being constantly hit. The large fake hit rate coming from them might overload memory buffers in the ASIC and cause memory corruptions. Identifying these pixels is not complicated, since their hit rate is typically four orders of magnitude larger than the one of a normal pixel. Once identified, these problematic pixels are tagged at this stage and ignored hereafter.

We then proceed to the actual tuning. We are particularly interested in tuning the differential front-end, because that is the only front-end used in the final RD53B. However, in order to properly test the sensor, we are tuning also the linear and the synchronous, enabling as many pixels as possible. The distribution from the threshold readout before and after tuning can be seen in Fig. 3.2. The value of each pixel's measured threshold is shown in Fig. 3.3. The threshold of the pixels in which the algorithm measuring the threshold fails to fit the S-curve is set to 0 by default.

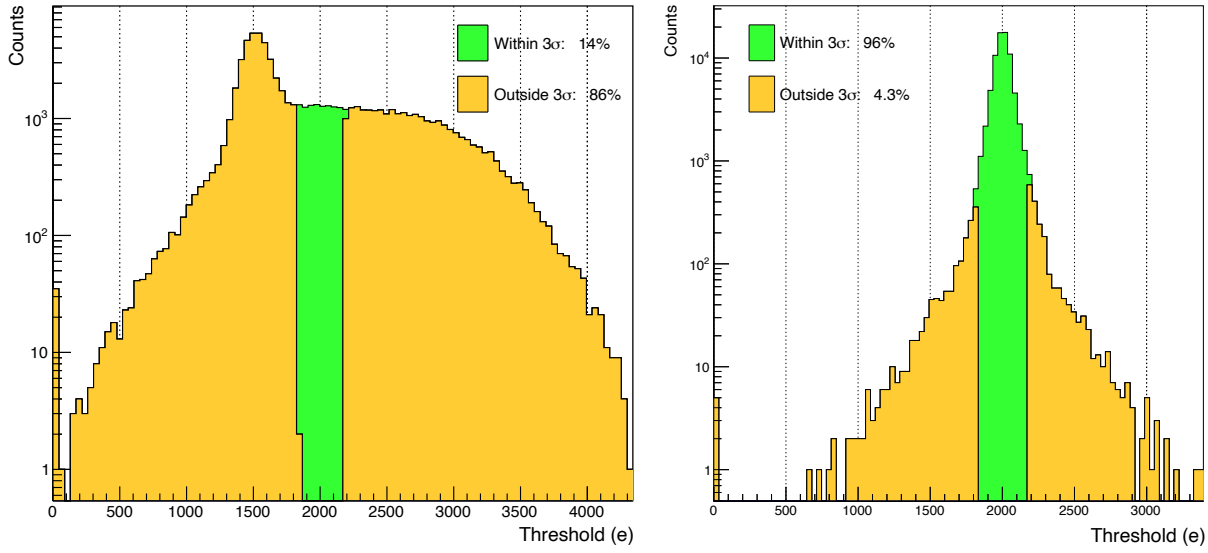

Figure 3.2: Threshold distribution before (left) and after (right) tuning, for the whole RD53A. The total number of entries is the total number of pixels in the pixel matrix.

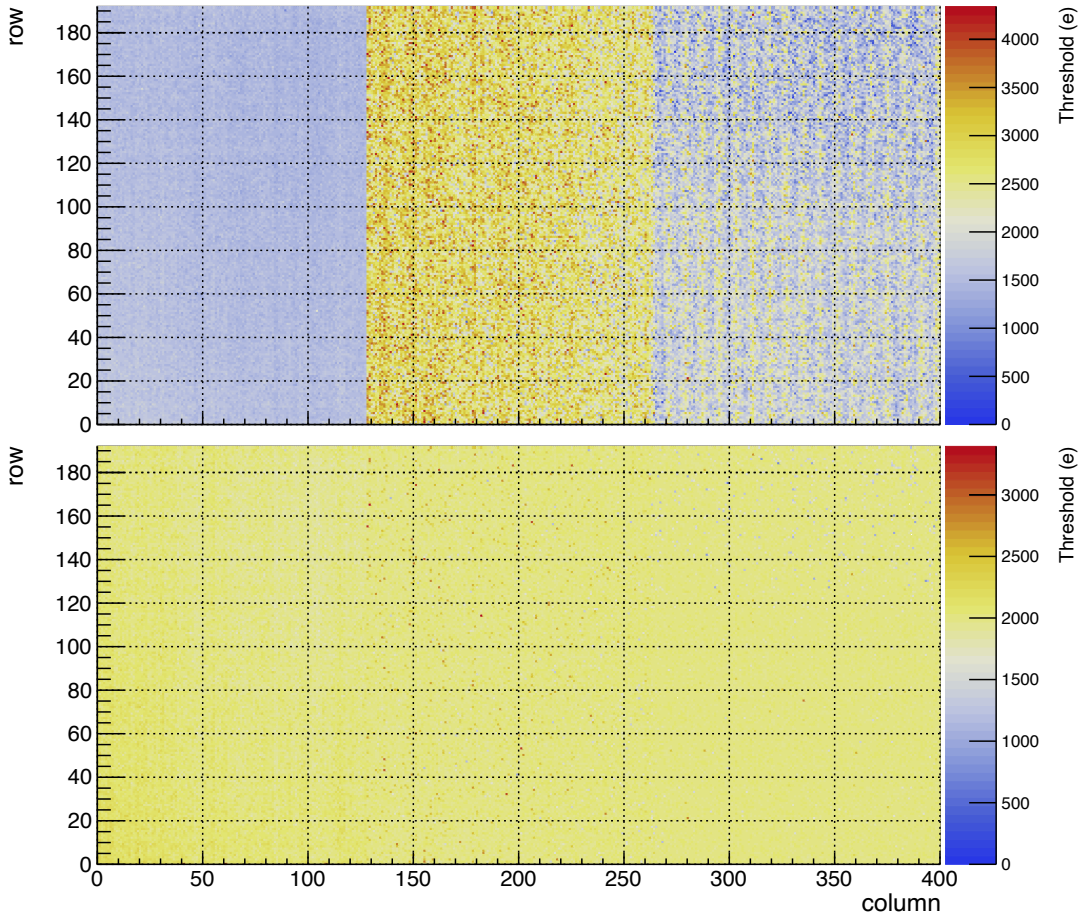

Figure 3.3: Threshold map before (up) and after (down) tuning. The three front-ends of RD53A (synchronous, linear and differential) can be well distinguished from their initial default states in the upper picture. The target threshold is 2000 electrons.

After each stage, the readout results are uploaded to localDB. The `scan-operator` then takes from influxDB the environmental data (typically Voltage, Current and Temperature) that was read out *during* the scan, and uploads it again to localDB, associated to the module information.

The `scan-operator` can tune a module and leave all the results ready in localDB to be uploaded to the production database. In a production stage, the only required external intervention would be changing from one module to another. Other than that, the operator could work on anything else while a module is being tested, minimizing the number of people required to be in the clean room, and contributing to the organization of an efficient testing chain.

3.3 The X-ray scan

Once the module has been tuned, we proceed to the x-ray test. This is the final stage of the module testing, that checks at the same time the correct behavior of all the pixels in the ASIC and in the sensor. The X-ray scan is run from the `scan-operator` as any other stage of the module testing, but the `scan-operator` does not play any special role here. This scan has been included in this Section of the thesis mainly for completeness.

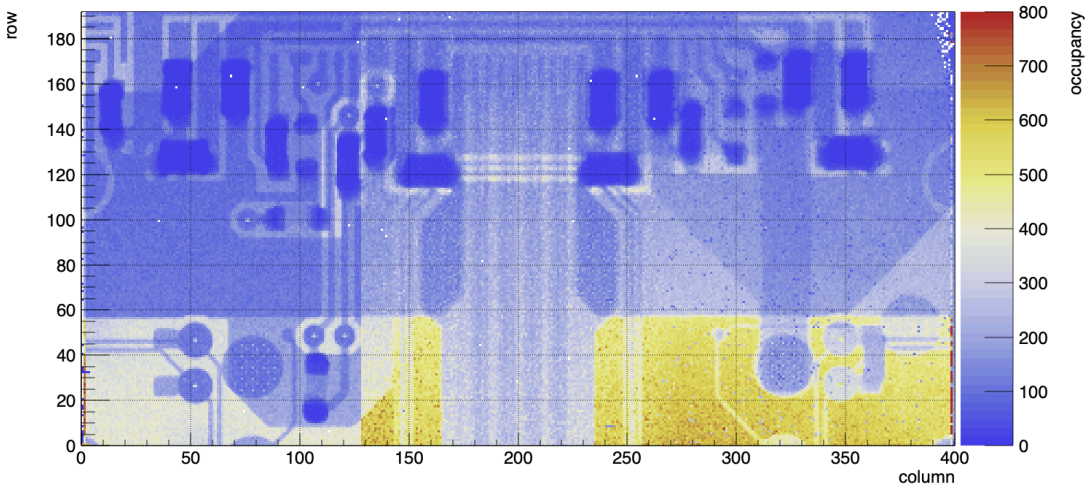

Figure 3.4: Results of an x-ray scan. The z axis represents the number of hits recorded in every pixel.

Figure 3.4 shows the output of an X-ray scan taken at KEK. Most of the X-says get trapped into the thick PCB components (capacitors, etc) before reaching the sensor, which is located right under the PCB. This is why we can see the PCB layout as the result of the scan.

3.4 Summary of the main improvements due to my contributions

There are three categories where my contributions have improved the module testing.

Safety improvements

The module testing has become safer for the module. Controlling the power supplies, and performing delicate procedures such as ramping up and down the high voltage from remote, and not directly operating the PS, makes the whole module operation safer. Monitoring temperature and dew point at the module surface, and using the data to trigger safety actions in case of need has also meant an improvement.

Precision improvements

This applies mainly to the IV and VI curves. Using a common tool makes the way the curves are taken identical for every institute. This allows for reliable comparison between modules, ensuring that any discrepancy would come from the modules themselves rather than from the way their data were taken.

Time consumption reduction

Time has been gained in many different stages by automating most of the module testing. Part of it comes also from the IV and VI scans. As an example, in the case of the IV scans,

ramping from 0 to -150 V in steps of 5 V, and taking 5 measurements per step, requires to take 150 measurements and to change the PS's voltage 30 times. Doing this automatically is clearly much faster than doing it manually, but more importantly this approach requires no manual intervention at all while the data is being taken.

Also, manual intervention is not needed during the parameter optimization and the tuning sequence. Apart from making the testing faster in overall, this considerably reduces the workload of the operator, making his/her time more valuable.

Chapter 4

Discussion

The `scan-operator`, as well as the power supply monitor tool, the script to take IV and VI data, and the one to find the best configuration of a module have been already used extensively both in Osaka and in KEK, by different people and in different situations, motivated by all the advantages that have been already described. However, its usage still has to be tested in the new RD53B's, and in other forms of RD53A modules, such as 3-ASICs' modules and 2-ASICs' ones).

The module production is still in an early stage. Once the RD53A production finishes, new RD53B modules have to go through a similar process, and finally around 10000 of them have to be produced and installed in the detector during the HL-LHC upgrade. Modules are now carefully assembled and tested to identify any potential issue and solve it before the actual production, but once the production starts the modules have to be made quickly in order to get all of them ready on time. It is at that stage where tools like the ones presented here will become essential.

However, work is still to be done on these tools. The software that these tools are using is constantly evolving, so their future maintainer has to ensure that compatibility is preserved within all of them. The hardware is also changing. Recently, a new LV power supply was decided to be used for the module testing in Japan. The power supply controller and monitoring tool has to be confirmed to work with this new device. If this was not the case, more development in the labRemote repository would also need to be carried out by someone with access to the power supply.

Finally, more features could be added to these tools if they are eventually needed, or some existing ones might be deprecated if they are not needed anymore.

Chapter 5

Conclusion

This thesis has presented the work done in optimizing the ATLAS pixel modules testing from the software side. The contributions to the testing presented in this document have made the procedure become faster, safer and less biased by human intervention. These tools will be specially relevant in the module mass-production stage, when speed, safety and consistency are three essential factors that need to be maximized.

Acknowledgments

On top of everything, I would like to deeply thank all the staff members of the Yamanaka Laboratory. Special thanks to Taku Yamanaka and Hajime Nanjo for all their advises and great patience while revising this thesis, helping me to improve as an academic writer.

Many thanks also to the students of the laboratory, specially to Taylor Nunes, Lakmin Wickremasinghe and Ryota Shiraishi for becoming good friends inside and outside the lab.

Finally, I am also deeply grateful to the members of the music club “Ginshokai” in the university, and to Gaizan Aeba and Kaoru Iwasa among others for so many good moments and experiences also outside the university.

References

[1] The ATLAS collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003. 437 p, 2008. Also published by CERN Geneva in 2010.

[2] CERN. High-luminosity lhc. https://home.cern/science/accelerators/high-luminosity-lhc. Accessed: 2021-08-16.

[3] Maurice Garcia-Sciveres. The RD53A Integrated Circuit. Technical report, CERN, Geneva, Oct 2017.

[4] The Atlas Collaboration. LocalDB. https://localdb-docs.readthedocs.io/en/top/.

[5] InfluxData Inc. InfluxDB. https://www.influxdata.com/.

[6] The Atlas Collaboration. YARR (Yet Another Rapid Readout). https://gitlab.cern.ch/YARR/YARR.

[7] The Berkeley laboratory. labRemote. https://gitlab.cern.ch/berkeleylab/labRemote/.

[8] The Atlas Collaboration. the Scan Operator. https://gitlab.cern.ch/YARR/utilities/scan-operator.

[9] Grafana Labs. Grafana. https://grafana.com/docs/.

[10] CERN. JavaScript Root. https://root.cern/js/.

[11] keysight. Introduction to the SCPI Language . https://rfmw.em.keysight.com/spdhelpfiles/33500/webhelp/us/content/__I_SCPI/00%20scpi_introduction.htm. Accessed: 2021-08-16.

Appendix A

The SCPI command set

The code we use to monitor and control power supplies can be used to communicate to many devices from different manufactures. Some of the commands, such as `power-on` or `power-off`, would work also for other lab equipment such as an oscilloscope.

The reason behind this fact is the SCPI (Standard Commands for Programmable Instruments) command set[11]. It is an ASCII-based set of commands that are both human readable and can be interpreted by a device. The following is an example showing the sequence of commands that are needed to turn on the output of a specific power supply channel, after setting the voltage and current to some values.

1. `INST:NSEL 3`, selects the channel 3 of the power supply.
2. `CURR 4.4`, fixes the current to 4.4 A.
3. `VOLT:PROT 1.8` sets the maximum voltage, also known as Over Voltage Protection to 1.8 V.
4. `OUTP ON` turns on the output of the power supply. The power supply will increase the output voltage until either the target current is reached or the voltage protection is reached.

Some commands are also available to measure the output current or voltage. `MEAS:CURR?` will return the output current, whilst `CURR?` will return its setting value.

These commands are serialized and sent to the device via a USB cable. Since the communication process can be complex to implement, common tools like LabRemote[7] are used instead to act as an interface between the user and the actual serial communication.