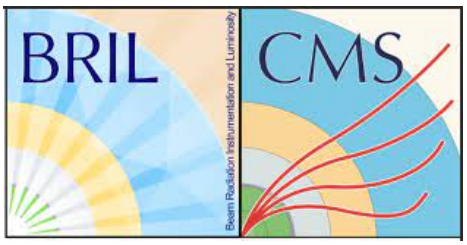
BRIL FBCM test bench development and BCM1F luminosity reprocessing - Summer Student Report

Marnix Barendregt
Supervisors: Grzegorz Wegrzyn, Joanna Wanczyk,
Olena Karacheban, Anne Dabrowski

September 7, 2023



Introduction

At CMS, the Beam Radiation, Instrumentation, and Luminosity (BRIL) project is responsible for measuring the luminosity. To this end, the group uses not only CMS detectors already used for physics measurements, but also develops and operates its own dedicated luminometers. The Fast Beam Condition Monitor (BCM1F), a silicon pixel detector, is one of these luminometers, providing per bunch crossing luminosity measurements completely independent of other CMS infrastructure in Run 3. To prepare for the High Luminosity LHC (HL-LHC), the BCM1F will be replaced by the new FBCM before Phase II, which is currently under development. The FBCM will be a silicon pixel detector similar to the BCM1F. The FBCM will have more channels, better radiation tolerance, the ability to measure time of arrival and time over threshold, and uses a digital ASIC [1]. My project consisted of three parts.

- Developing a test bench to quickly test FBCM ASICs.
- Developing a dashboard to store test results and other FBCM data.
- Analysing and reprocessing 2023 data from the BCM1F detector.

Background

Luminosity calibration

Luminosity is the conversion factor between rates of particle hits R and the cross section of a process σ:

$$R = \mathcal{L}\sigma \tag{1}$$

The cross section σ of a process is a physical property independent of the type of detector or accelerator used. To measure this property accurately, we must know the luminosity of our detectors as accurately as possible.

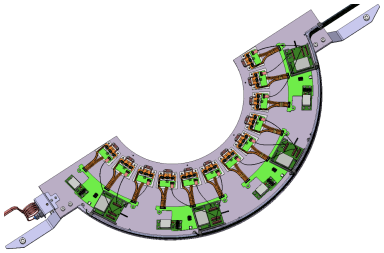
A luminometer is a particle detector which is designed to measure luminosity. The main requirement is that its reported number of hits is as linear as possible with the actual number of particle hits. This ensures a small uncertainty in the final luminosity measurement. Luminometers are calibrated during Van der Meer (VdM) scans, where the two beams are moved transversely to each other. VdM scans are performed once per year for absolute calibration. Emittance scans are shorter VdM-like scans, which are performed at the start of every LHC fill for LHC diagnostics calibration cross checks. From the evolution of the rates during the procedure, a converson factor called the visible cross section σ_{vis} can be extracted for each detector channel. The (instantaneous) luminosity can then be calculated using

$$\mathcal{L} = \frac{R}{\sigma_{vis}} \tag{2}$$

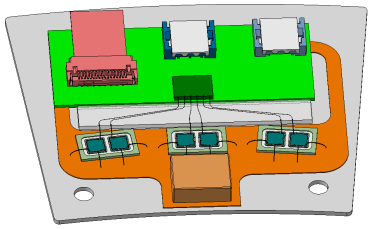
where R and σ_{vis} are the rate and visible cross section of a specific detector channel [2].

FBCM ASIC

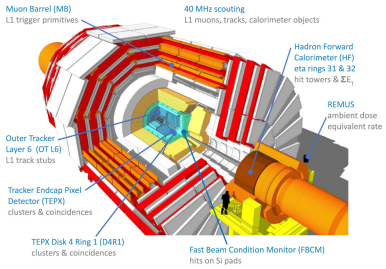
The FBCM is designed to be as linear as possible. It consists of four half-disks, located on both ends of the interaction point (Fig.1c). Each half-disk (Fig.1a) contains 12 hybrids (Fig.1b) which hold an Application-Specific Integrated Circuit (ASIC) and six silicon pads. When a particle hits a silicon pad, it creates a current pulse. The ASIC is an integrated circuit that was designed to convert this current pulse into a digital voltage pulse in three steps. First, the current pulse is converted into a voltage pulse and shaped by a transimpedance amplifier. Next, the signal is amplified by the booster. Lastly, the signal is converted into a digital SLVS [3] signal by a discriminator. In the final design, this digital signal is transported via optical fibres and used for luminosity calculations.



(a) Half-disk with 12 hybrids



(b) Hybrid with one asic and six Si pads



(c) Position in CMS detector

Figure 1: The FBCM detector design

The ASIC has multiple voltage references, which are supplied by digital-to-analog convertors (DACs), and various switches. These DAC inputs and switches are set by values stored in memory registers on the ASIC. The values in these registers can be read or set via an I2C interface [4].

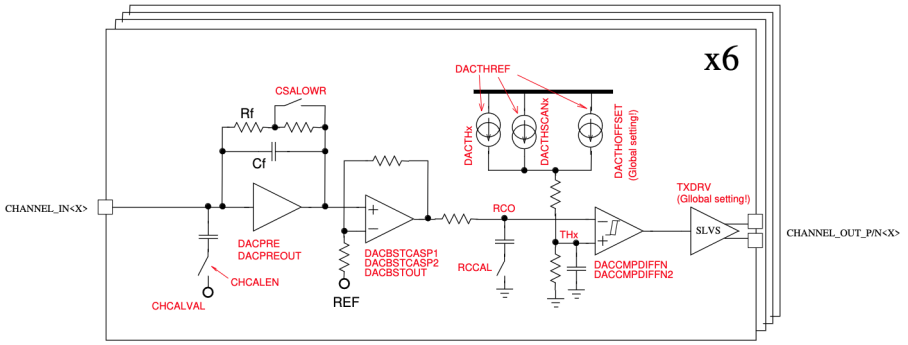


Figure 2: Channel diagram with configuration DACs and bits highlighted. [5]

The ASIC has a built-in calibration circuit, which sends current pulses that immitate particle hits in the silicon pads with tunable intensity. These calibration pulses are also configurable via I2C, and are triggered via the `CAL_STROBE` input pin.

There are various test points on the ASIC circuit, which are routed to an analog multiplexer (AMUX). By setting a register, the desired test point can be selected and outputted to the AMUX output. This allows internal voltages to be easily measured. For the purpose of testing the ASIC functionality, noise and performance during irradiation, the ASICs are mounted on test PCBs (Figure 3) which include an ADC to measure the voltage on the VDDD and VDDA line, and to measure the AMUX output. The ADC is separate from the ASIC, but is also controlled via an I2C interface.

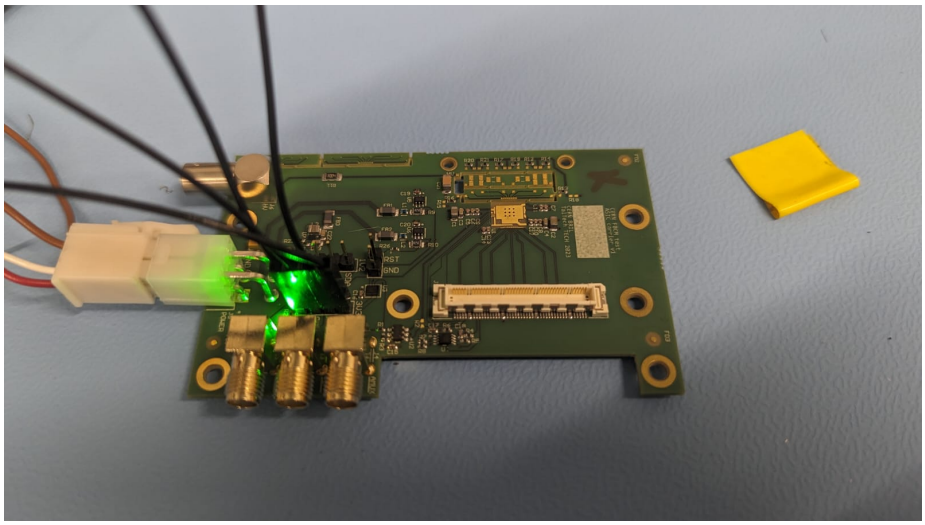
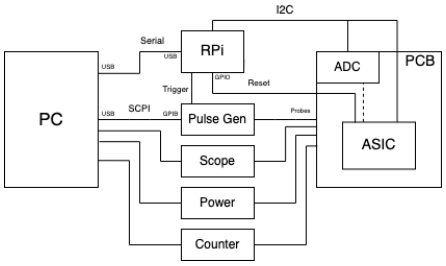


Figure 3: Test PCB with ADC. ASIC to be bonded later

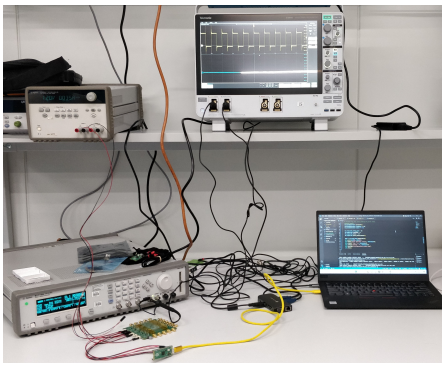
1 ASIC Test bench

The test bench consists of various instruments (Fig **??**).

- Agilent E3646A power supply to power the PCB and ASIC
- Hewlett-Packard HP81110A pulse generator to trigger the calibration circuit
- Tektronix MSO64B oscilloscope to measure the ASIC output
- Agilent 53132A counter to count the number of ASIC digital output pulses



(a) Block diagram of lab setup



(b) Lab setup

These instruments can be controlled using SCPI commands sent over USB (to GPIB). SCPI is a standard for controlling instruments where commands and responses are simple ASCII strings. Examples can be found in the instruments programming manuals. These instruments are controllable via a Python program that contains separate classes for each instrument, and getters and setters or other functions for each command. This layer of abstraction allows for easy control of the instruments. As an example, to set the voltage of the power supply, the following code can be used:

```
tb.powerSupply.CH1.voltage = 1.2
```

1.1 PC - I2C interface

The ASIC registers and the ADC on the PCB are controlled via I2C. To communicate between the PC and these components, a Serial-I2C interface has been developed. It was written in MicroPython for the Raspberry Pi Pico microcontroller (RPi). The RPi takes in text commands via USB and sends the corresponding I2C command to the ASIC or ADC. Error handling has been implemented. For get query commands, the RPi will return the value of the register or ADC if successful. If unsuccessful, it will return an error message. Similarly, for set commands, the RPi will return a success message if successful, and an error message if unsuccessful. Because at the time of development, the FBCM ASICs were still in the production phase, the I2C interface was successfully tested on a different ASIC. Additionally, the ADCs on the test PCBs were successfully tested using the interface. The RPi also has a reset pin connected to the ASIC and supplies a trigger to the pulse generator for calibration pulses. Once again, to speed up ASIC testing, the text commands sent to the I2C interface are abstracted away in a Python class which is part of the main test bench program. This allows for easy control of the ASIC and ADC. For example, to set the ASIC calibration pulse amplitude, the following code can be used:

```
tb.asic.chcalval = 1.0
```

A JSON file is loaded when running the program and contains all the settings, e.g. the RPi pin used for the reset signal, or the I2C address of the ASIC.

1.2 ASIC tests

Abstracting away the lower level communication with the instruments and I2C devices was necessary for automating the ASIC tests. While the ASICs are still being manufactured, the automated test scenarios have all been implemented [6]. I will give a short overview of the tests here.

- **Power consumption** - Measure the power consumption on the VDDA and VDDD lines while changing the SLVS output current.
- **I2C** - Check that all registers on ASIC are equal to their specified default values. Next, write random values to them and read them back.
- **DACs characteristics** - The output voltage of digital-to-analog converts on the ASIC may not be linear or monotonic with the input value. For all the DACs, the output voltage is measured while sweeping the input value. To measure the voltage, the AMUX is set to output the correct test point and the AMUX output is measured by the ADC on the PCB via I2C.
- **DACs calibration** - Use the DACs characteristics from the previous test to reach the desired standard output voltage.
- **Noise occupancy** - This test quantifies the internal noise of the ASIC. For each channel, sweep over the threshold value for the discriminator, while having no input signal. The counter measures the number of false output pulses. The number of pulses against voltage is an s-curve, from which the noise intensity can be extracted.
- **Threshold scan** - The same as above, but calibration pulses are inputted to the ASIC. The resulting count against threshold voltage curve can be used to estimate the gain of each channel.
- **SEU test** - Single event upsets (SEUs) are changes in the memory of the ASIC caused by high energy particles during irradiation. The ASIC contains a register which counts the number of SEUs. While irradiating the sample, we write random values to all registers and periodically read them back, and check the SEU counter.

2 FBCM Dashboard

The data from ASIC tests is stored in JSON and CSV format. To make it easier to analyse and compare the data, a dashboard has been developed. Other data related to the FBCM development, such as Si pad sensor test results and information of other components is also stored. The dashboard is written in Python using the Django framework. All the data is stored in a PostgreSQL database.

The source code is available on GitLab [7]. Django was used because it offers the flexibility of storing and handling various types of data while streamlining much of the server back-end development.

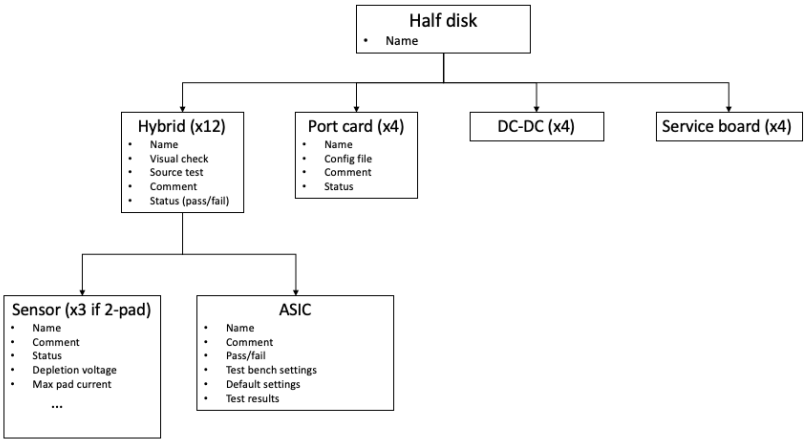


Figure 4: The preliminary structure of data stored in the database. Each child contains a connection to its parent using the Django Foreignkey field.

Figure 4 shows the structure used for the FBCM data. Information about other FBCM components will be added later. Each component has its own upload form with fields corresponding to the data stored in the database. The ASIC test data is stored outside the database as raw CSV and JSON files to allow for easy analysis. The Si sensor test data contains various voltage, current and capacitance curves, which are automatically plotted. When uploading this data, various quantities such as the depletion voltage and maximum capacitance are automatically extracted and stored in the database. There is a compare functionality, where data from multiple sensors is plotted on the same figure (see Figure 5).

Figure 5: Si sensor data comparison functionality.

The UI was realised in HTML with UIKit for styling. The HTML templates contain sections which are filled in with data or plots by the Django code.

3 BCM1F Luminosity reprocessing

While the FBCM is still under development, its predecessor the BCM1F is still providing online luminosity measurements. In the online 2022 luminosity data from the BCM1F it was noticed that the luminosity would suddenly drop (Figure 6).

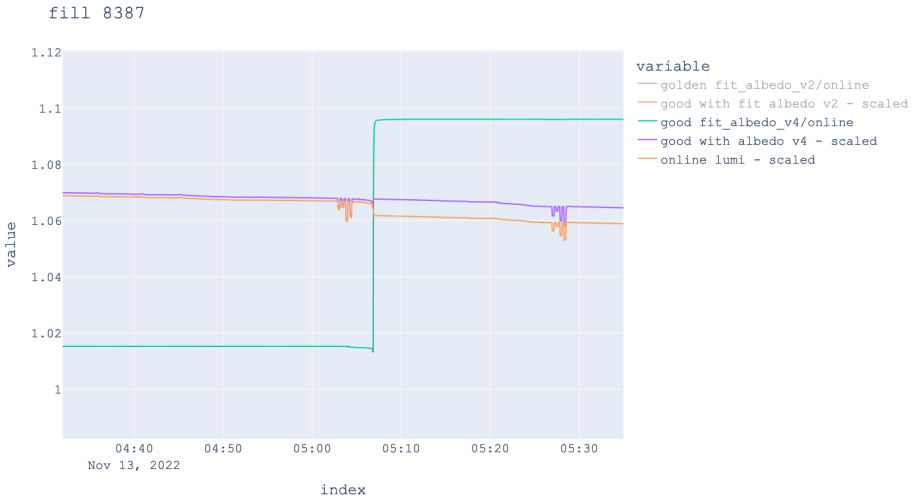


Figure 6: Online and offline (reprocessed) luminosity data from the BCM1F in 2022. Notice the sudden drop in the online data after 05:00. The purple line is the reprocessed luminosity

This was explained by the failure of the online automasking algorithm. When a Si sensor receives too much radiation it can start driving large currents, and it is turned off automatically. The online algorithm detects when the luminosity from each channel is significantly lower than the average of all channels. In this case it is excluded from the total luminosity measurement. This automasking algorithm is supposed to exclude switched-off channels. However, in this case the algorithm did not detect a channel switching off, thus lowering the average luminosity.
Only the final online 2023 luminosity data is available, as the logs produced by the online algorithm are lost. The algorithm takes histograms of the number of hits for each channel as input. The histogram data has a time resolution of 16384 integrated orbits, corresponding to 1.6s. My task was to reprocess the 2023 data: recalculate the luminosity from the histogram data, and see if similar automasking failures occured. The online and offline (reprocessing) algorithms are identical and should give the same result. The offline algorithm can then be used to reproduce the log files and find failures or test improvements to the luminosity calculation algorithm. I reused a Python script from the 2022 reprocessing, and adapted it to the updated 2023 online algorithm. The script works as follows:

1. Initially exclude some channels, e.g. because they are known to be noisy.
2. Load the raw histogram data from HDF5 files.
3. Separate the data by channel.

4. Calculate the total luminosity from all initially included channels: the reference luminosity.

5. Calculate the luminosity for each channel, and mask the channel out if its luminosity is less than 80% of the reference luminosity.

6. Recalculate the total luminosity from the remaining channels.

This result is then stored in another HDF5 file. To calculate the luminosity from the hit histogram data, we use the zero-counting method. The number of hits in a time period follows a Poisson distribution. We can extract the mean rate from the fraction of zero-counts in the histogram.

$$p(n) = \frac{\lambda^n e^{-\lambda}}{n!} \implies \lambda = -\ln(p(0))$$

In practise, $p(n)$ is the histogram data per 4NB mentioned above.

A separate script was made to plot the reprocessed luminosity, the online luminosity, and their ratio for comparison, similarly to figure 6. These scripts were based on the 2022 reprocessing scripts. Next, to investigate differences between the online and reprocessed data, another script was made. This script calculates and stores the luminosity for each channel. This in turn has a plotting script for visualising the per channel luminosity.
Additionally, a script was made to plot the raw channel rates. This data is available at a per bunch crossing time resolution.

I used these scripts to look for discrepancies between online and reprocessed offline data in all the 2023 fills. The calculation and plotting was all done automatically and in parallel. I looked at the total luminosities and online/offline ratios for each fill. Nearly all fills had a constant ratio, incdicating that online and offline calculations were consistent. Two fills showed a sudden drop in the online luminosity, similar to the 2022 data. This happened at exactly the same time as a Si pad being switched off, judging from the per channel luminosity dropping to zero. Another two fills showed a sudden drop in the online luminosity, but without any sign in the per channel luminosity. In this case, both the online and offline luminosity showed steps.
The latter, and other spikes in the luminosity ratio were found to coincide with a new fill starting. Unfortunately, due to time constraints I was unable to further investigate these issues. Since the offline Python and the online C++ algorithms are identical, there is likely an undiscovered error in one of them. I suggested that one could wrap the online algorithm in a 'simulator' where we pass the stored histogram data directly to the online algorithm. This would allow for debugging the online algorithm, and keeping track of when certain channels are masked out. Additionally, this reveal potential but unlikely differences between the online histogram datastream and the stored histogram data.

References

[1] "The phase-2 upgrade of the cms beam radiation, instrumentation and luminosity detectors - technical design report," Tech. Rep. CMS-TDR-023, CERN, July 2021.

[2] O. Karacheban and P. Tsrunchev, "Emittance scans for cms luminosity calibration," *EPJ Web of Conferences*, vol. 201, 2019.

[3] I. S. Bulbakov, E. V. Atkin, and A. G. Voronin, "High speed slvs transmitter and receiver," *Journal of Physics: Conference Series*, vol. 675, p. 042035, January 2016.

[4] J. Valdez and J. Becker, "Understanding the I2C bus," Tech. Rep. SLVA704, Texas Instruments, June 2015.

[5] J. Kaplon and G. Wegrzyn, "FBCM 2023 datasheet," tech. rep., BRIL CMS CERN, 2023.

[6] M. Barendregt, "FBCM23 ASIC testbench GitLab repository." `https://gitlab.cern.ch/cms-bril-fbcm/fbcm23_asic_testbench`.

[7] M. Barendregt and L. Liu, "FBCM dashboard code." `https://gitlab.cern.ch/cms-bril-fbcm/fbcm-django-system`.

[8] "Final CMX presentation indico page." `https://indico.cern.ch/event/1313710/`.