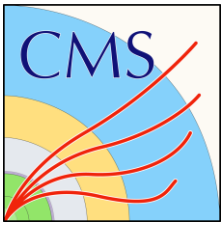



# Firmware Design for Particle Timing Measurements in the CMS ETL

**CERN Summer Student Program Report, 2023**
**At CMS Department**

**Hamza Elsayed**
**CERN Supervisor: Özgür Sahin**

# Acknowledgements

While working on the Data Acquisition Team at MTD, I had the most rewarding experience. When I first started, I had only worked on simple FPGA projects in my university courses. However, working here has allowed me to develop advanced VHDL skills and apply what I am learning in real-world applications. I have had the opportunity to learn and try new things every single day, and my supervisor and the DAQ team have been incredibly supportive. I am grateful for their guidance and mentorship, which has had a wonderful impact on my skill set and career.

I would like to extend my sincere appreciation to the esteemed leadership of the University of Bahrain, including Dr. Fuad Al-Ansari, the President of the University of Bahrain, Dr. Haifa Ebrahim Al-Khalifa, the Dean of the Engineering College, and Prof. Zuhair Khalifa Bahri, the Chairperson of the Electrical and Electronics Engineering Department. I am also grateful to all my knowledgeable and supportive instructors, particularly Dr. Bader Al-Mannai, Dr. Sarah Al-Shareeda, and Dr. Ebrahim Abdul-Rahman, for their invaluable guidance and care.

# Abstract

The MTD (MIP Timing Detector) is a new cutting-edge timing detector with 30 ps resolution, designed to measure particle timing with sophisticated precision that will be used to disentangle 200 nearly-simultaneous collisions in the CMS Phase-2. CMS is developing completely new back-end electronics to handle higher bandwidth interfaces to the front-end electronics within CMS and to process the resulting increased data volume of the HL-LHC era. To place the scale of the upgrade in perspective it is useful to consider the tracker readout, which will require an increase in bandwidth from 15 Tbit/s to 184 Tbit/s. This report summarizes my understanding of the MTD project that I joined its Data Acquisition Team during my Summer Student Program of 2023. Concluded with my contribution in developing ETL (Endcap Timing Layer) firmware readout logic.

# Table of Contents

# 1 Introduction

The Large Hadron Collider (LHC) at CERN is home to the Compact Muon Solenoid (CMS), a detector with a wide range of capabilities. With the LHC undergoing a luminosity upgrade, the CMS and other experiments on the ring will have an even broader range of data detection and analysis capabilities. To account for the increased luminosity, the CMS is currently in the process of developing the MIP Timing Detector (MTD) in **Fig.1**, an advanced timing detector. The MTD is specifically designed to allow the CMS to measure the production time of minimum ionizing particles (MIP) with extreme precision. This will enable scientists to disentangle the approximately 200 nearly-simultaneous collisions that occur in each bunch crossing of the LHC. The MTD is expected to be a game-changer for the CMS, providing scientists with the ability to analyze data in ways that were previously impossible.

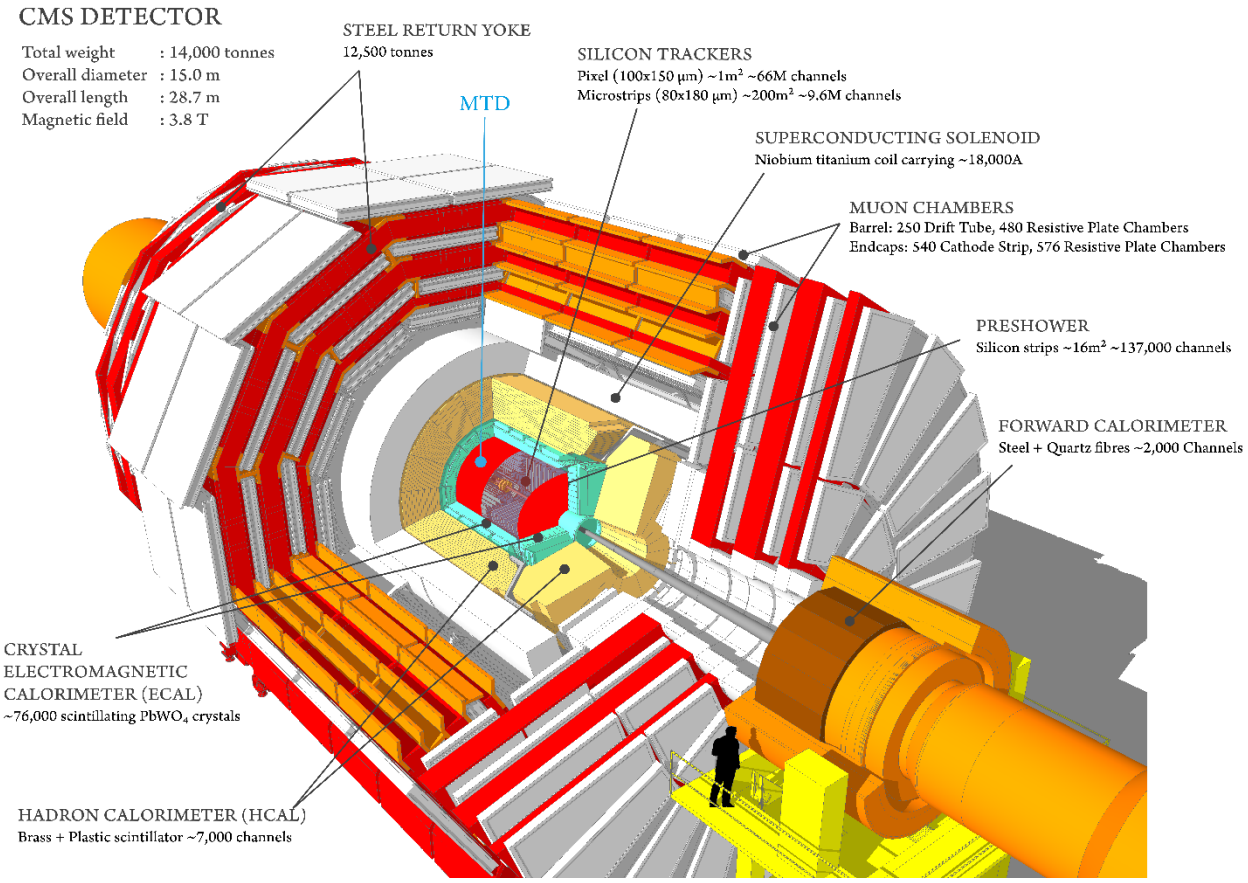



**Figure 1: The MIP Timing Detector's (MTD) location in The Compact Muon Solenoid (CMS).**

## 2 Background: Disentangling 200 Nearly Simultaneous Collisions

The purpose of the detector is to separate the tracks coming from different nearly-simultaneous collisions. This is done by measuring the time of the particles hitting the pixel silicon sensors with very high resolution. In **Fig. 2**, three tracks have the same time $t_1$, and one track hits the silicon sensors at different time $t_2$. Through this we conclude that the particle hitting the timing layer at $t_2$ is coming from a different collision than the ones hitting it at $t_2$.

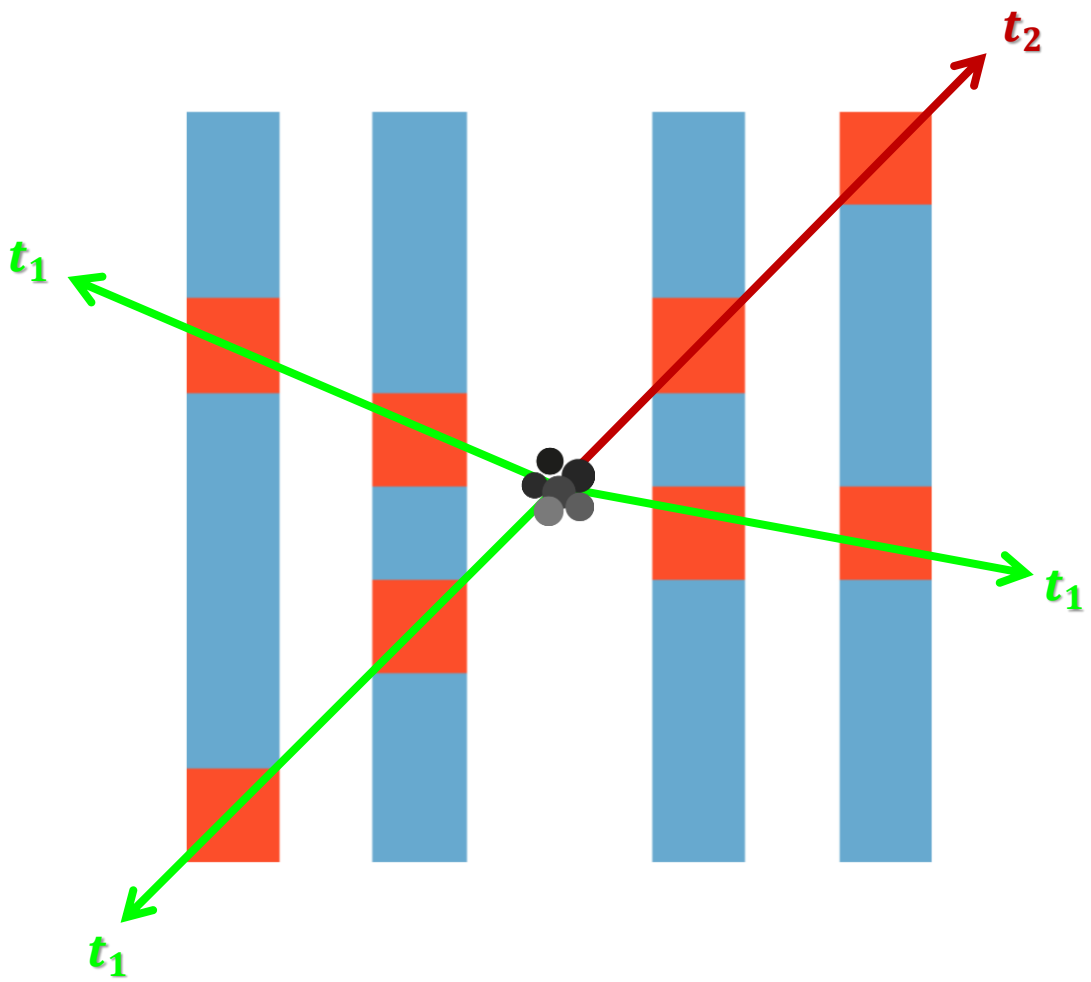



**Figure 2: Four particles hitting detector's silicon sensors.**

# 3 Methodology

## 3.1 The BTL, The ETL, And ETROC

The MTD consist mainly of two parts:

1. Barrel Timing Layer (BTL): The BTL is the central part of the MTD and is designed to measure the timing of charged particles. It includes active elements such as the LYSO:Ce (Lutetium Yttrium Orthosilicate doped with Cerium) crystals to detect and measure the energy of charged particles passing through them by emitting scintillation light when a charged particle hit it and deposit energy. Along with Silicon Photomultipliers (SiPMs) which are light-sensitive detectors used for detecting and converting the scintillation light emitted by the LYSO:Ce crystals into electrical signals. The BTL also has readout electronics, a structural design called Tracker Support Tube (Tracker-BTL TST mechanics), cooling and environmental control systems, and various services required for its operation.

2. Endcap Timing Layer: The ETL is another component of the MTD. It operates in the endcap region of the CMS detector and follows a similar principle of measuring the timing of charged particles. However, the radiation dose for the ETL is much greater than for the BTL and highly non-uniform. The SiPM technology selected for the BTL does not have sufficient radiation tolerance to work over most of the range of the ETL. Therefore, the ETL will use planar silicon devices with internal gain. The Low Gain Avalanche Detectors (LGAD), which are also under consideration for a fast-timing layer in the very forward region of the ATLAS experiment. The LGAD sensors have intrinsic gain of 10–30 provided by a special implant, which helps to overcome capacitance and other noise sources and achieve a low-jitter fast-rising pulse edge that enables precision timing reconstruction for MIPs. Sensors with a 50 μm active region, within a normal 300 μm thick silicon wafer, and a thin implanted gain layer are expected to provide the desired performance.

The ETL Readout Chip (ETROC) is the Application-Specific Integrated Circuit (ASIC) designed for precision timing of in CMS ETL. ETROC2 is the first full-size full-functionality prototype of ETROC, which aims to achieve 50 ps per hit time resolution together with LGAD. An ETROC2 includes a 16 × 16 pixel matrix and a chip peripheral. **Fig. 3** shows a simplified block diagram of ETROC2. The ETL exist on the front-end of the MTD, while the logic (i.e. Unpacking and processing) of the received data from it will be accomplished by the MTD back-end Serenity boards positioned in node slots; hence these boards will be the main drivers of the MTD DAQ system. [1]

In each pixel, the signal current from LGAD is conditioned by a preamplifier to generate an analog pulse on top of the baseline voltage of the preamplifier. The analog pulse is then converted into a digital pulse by a discriminator, which is followed by a low power Time-to-Digital Converter (TDC) specifically designed for ETROC [2]. A charge injector is included in each pixel to generate an emulated LGAD signal to facilitate the testing of the full signal process chain. A 10-bit Digital-to-Analog Converter (DAC) is used to generate the threshold voltage of the discriminator. The input of the DAC could be obtained with an optional threshold calibration integrated in each pixel [3]. Users can choose to bypass the calibration and write the DAC directly with the threshold obtained using their own calibration method.

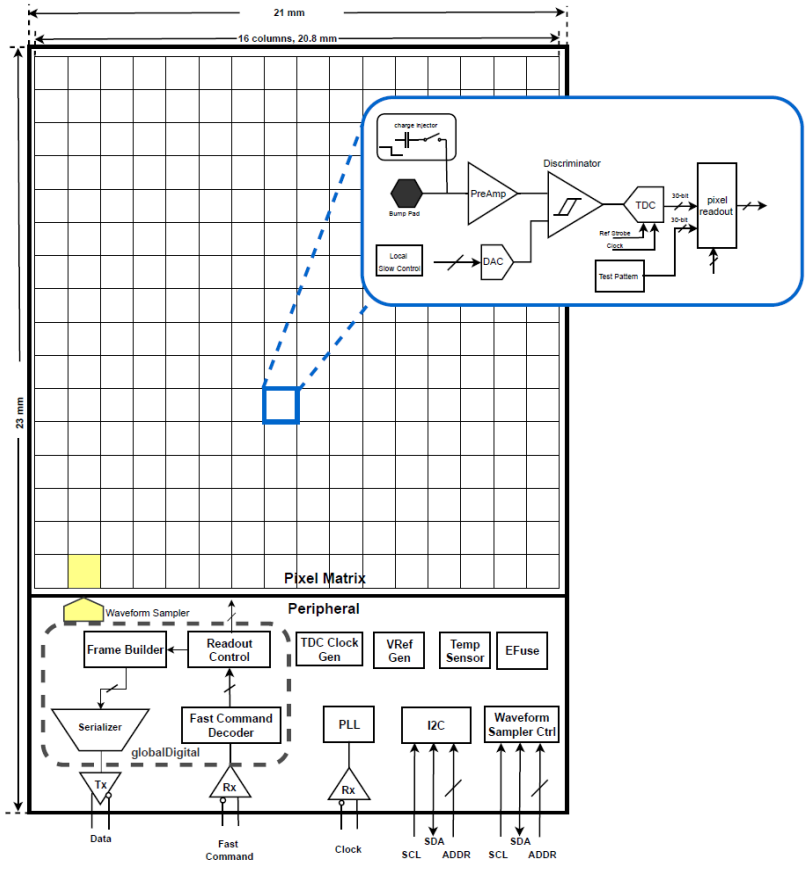



**Figure 3. ETROC2 diagram.**

## 3.2 Data Trail: From The Front-End To The Back-End

The data read out from the detector is transmitted to the back-end via Versatile Links+ (VL+). The Versatile Links consist of:

- lpGBT (Low Power GigaBit Transceiver): a radiation tolerant multi-gigabit communication ASICs.
- (VTRx+): radiation tolerant optical transceivers which are capable of handling the data rate.

### lpGBT

A single lpGBT will handle communication of 12 readout ASICs in BTL and 24 or 12 readout ASICs depending on the rapidity in ETL. There will be in total 864 lpGBTs and bidirectional links in BTL and 1600 in ETL. In order to cope with the required data throughput, the lpGBTs will be operated at 10.24 Gb/s and 5.12 Gb/s operation modes depending on the occupancy of the particular detector region. [4]

### VTRx+

The data received from the E-links of a lpGBT are serialized and transmitted to VTRx+, which transmits them through 70 m multi-mode optical fibers to the back-end electronics. All of these components are specified to safely tolerate the radiation expected from the full HL-LHC run period. [4]

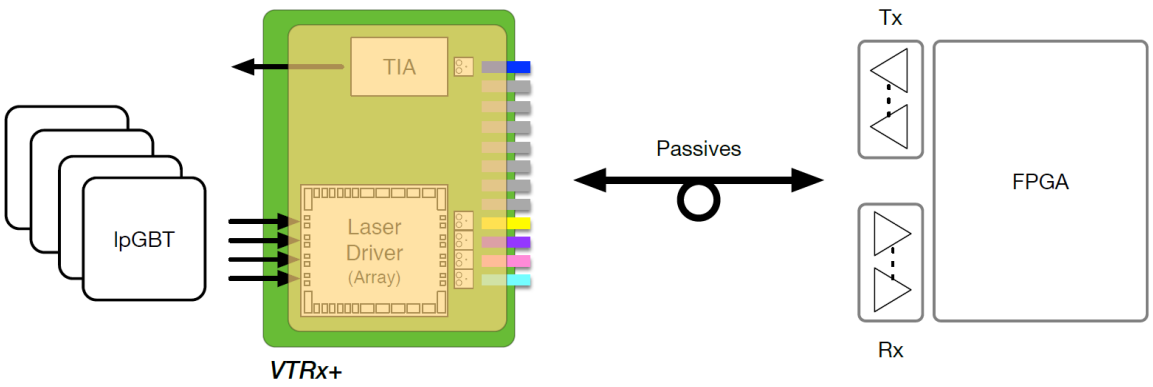



**Figure 4**: Overview of the Versatile Link + system showing the components of the VTRx+. [6]

## MTD DAQ boards

Data are then received in USC-55 (USC stands for Underground Service Cavern) by Firefly transceivers on the MTD DAQ boards through their front-panels, which process and package them for transmission to the DTH400 (DTH stands for Data Trigger Hub) and DAQ800 boards via 25 Gb/s (or 16 Gb/s depending on the FPGA architecture) front-panel optical connections. [4]

The number of DAQ boards needed to handle the total DAQ throughput depends on the number of bidirectional data links available on the boards. For the Serenity boards with 144 bidirectional links per board, the back-end system will comprise eight DAQ boards for the BTL readout and seven DAQ boards for the readout of each endcap of the ETL. The total DAQ throughput can be handled by one DTH400 and three DAQ800 for BTL and by one DTH400 and two DAQ800 for each endcap of the ETL detector. In total, the DAQ system will occupy three crates in two racks. The ATCA (Advanced Telecommunications Computing Architecture) layout of the BTL and ETL crates are shown in **Fig. 5**. [4]

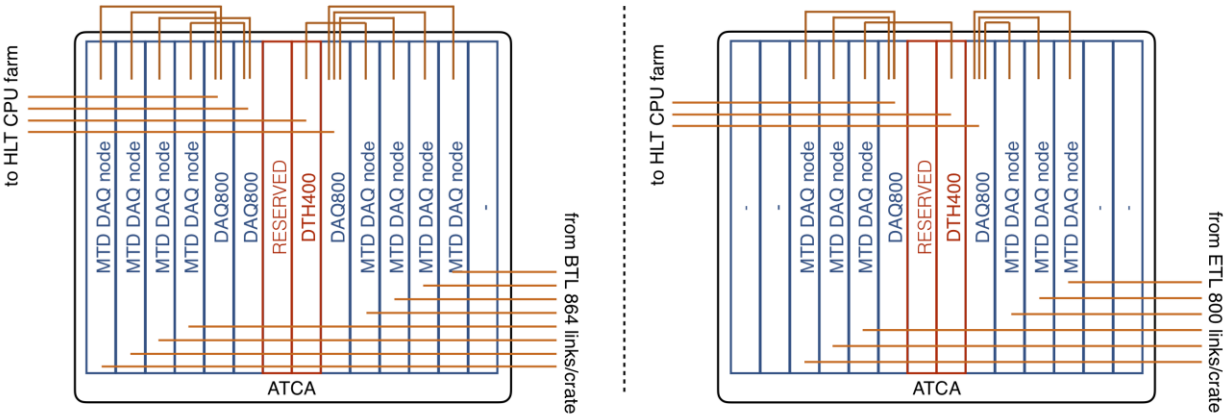



**Figure 5**: The MTD DAQ ATCA crate layout with 8 (left) and 6 (right) Serenity boards (carrying 2 Xilinx Kintex Ultrascale Plus KU15P FPGA daughter boards) for the BTL detector and each endcap of the ETL detector, respectively.

## SERENITY

Serenity is a general-purpose board compatible with most sub-detectors being developed for the CMS phase-2 upgrade. The Serenity Carrier Card has been developed to handle the huge data rates required for high-luminosity operation of CMS at the LHC as part of the Phase-2 upgrade program. CMS is developing completely new back-end electronics to handle higher bandwidth interfaces to the front-end electronics within CMS and to process the resulting increased data volume (i.e. for delivery for DAQ or to extract a trigger signal). To place the scale of the upgrade in perspective it is useful to consider the tracker readout, which will require an increase in bandwidth from 15 Tbit/s to 184 Tbit/s.

Serenity is an ATCA prototyping platform consisting of three elements: An ATCA Carrier-Card which provides the common board services, that is, power, clocking, optical interfaces, electrical interconnections between FPGAs, the Intelligent Platform Management Controller (IPMC) functionality and an on-board CPU to control and manage the board; Daughter-Cards which host the data-processing elements themselves (in our case FPGAs) and a framework of generic, flexible firmware and software. [5]

The board is capable of carrying two high-speed, high-capacity FPGAs and driving 144 bidirectional links to the front-end and to the DTH400 or the DAQ800 via Multi-fiber Termination Push-on (MTP) fibers. The reference clock necessary for driving the DAQ links from the FPGAs is distributed by low-jitter Si5345 phase locked loops (PLLs). [4]

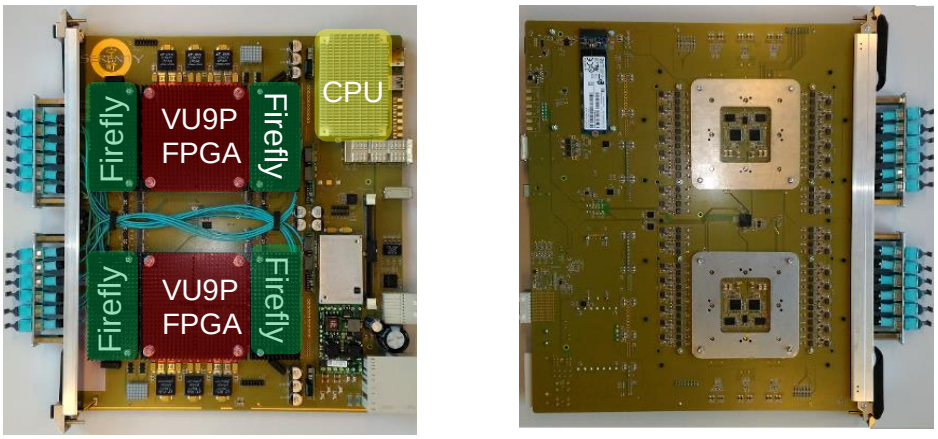


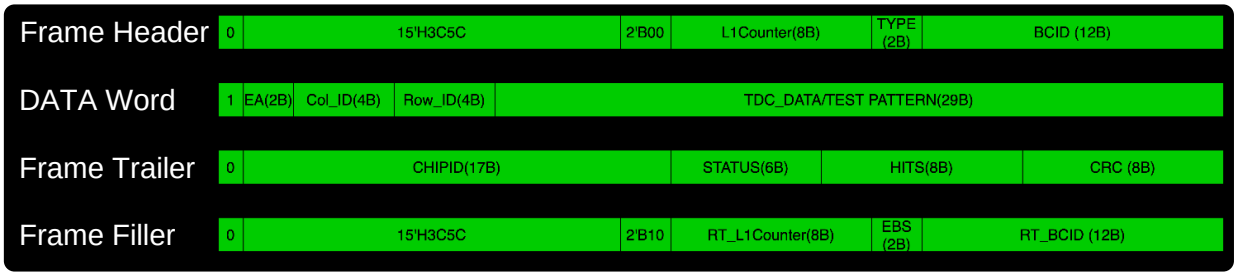

**Figure 6. The SERENITY Carrier Card**

## Event Builder

Data collected from all MTD DAQ boards are sent to the event builder using the 100 Gb/s data-to-surface (D2S) protocol. Where the hardware and software moves data from the Underground Service Cavern at Point 5 to the Surface. [4]

## 3.3 Raw Data Received From The Front End

There are four types of raw data that the back-end is receiving:




The header and filler always started with 16 bits of code 0x3C5C. If the 16 bits of code "0x3C5C" is followed by '00', it is a header. If it is followed by '10', it is a filler. The header has a two-bit data type to describe the type of the data frame. The detailed definition is in **Table 1**. The header includes a 12-bit Bunch Crossing Identification (BCID) to identify the event BCID. The L1 counter is an 8-bit counter to count the L1A signals when the event is accepted. [1]

| Type | Meaning |
|---|---|
| 00 | Regular data format |
| 01 | Random test pattern |
| 10 | Counter test pattern |
| 11 | Reserved |

**Table 1. The type of data frame definition.**

Each data word has a 4-bit column id and 4-bit row id to determine its source pixel. Beside the 29 bits of TDC data and 8 bits of pixel id, there are two bits of EA for potential Single Event Effect (SEE) or memory error detection. When TDC data (or test pattern) is written in circular buffer with Hamming code and is decoded in the frame builder. The two bits of EA are the error status of Hamming code decoder in the frame builder. If EA is '00', it means there is no error detected. If EA is '01', it means there is one bit error that has been detected. and corrected. If EA is '10', it means there are 2 or more bits error, and the Hamming code cannot correct it. [1]

A full data frame is always terminated with a trailer. The trailer always started with ’0’ and 17-bit chip id. We will avoid using chip id 01111001011100XX. By avoiding using these 4 chip id numbers, the trailer does not start with 16 bits of code 0x3C5C, different from the filler and header. The 6-bit of status in the trailer describe the status when this event is accepted. The status definition is shown in **Table 2**. It is not expected to receive a broken data frame without a trailer even if the bandwidth of the data is beyond the data taking bandwidth. When the occupancy is too high and the L1 event buffer is almost full, the readout switches to quick read mode in which the data words are not included, and this data frame is tagged as an overflow event in the status field. The hits count presents an 8-bit count of the data words in this data frame. The 8-bit CRC code is to verify the data integrity of the whole data frame from header to the trailer. [1]

| Status Bit | Define | Meaning |
|---|---|---|
| 0 | L1 Event buffer overflow | L1 event buffer is full, event may lose |
| 1 | L1 Event buffer almost full | L1 Events >= 96 (can be user defined) |
| 2 | L1 Event buffer half full | L1 Events >= 64 |
| 3 | Data error occur but corrected | error occurred, corrected |
| 4 | Data error occur but not corrected | 2 or more bits error occurred, not corrected |
| 5 | Reserve | –– |

**Table 2. The definition of status in a trailer.**

It is a common practice to inject filler words in the data output stream when there is no trigger selected data. These fillers are used to maintain the data transmission rate and to provide a continuous data flow to the downstream systems. The filler starts with a 16-bit code of 0x3C5C and is followed by ’10’ to indicate that it is a filler word. The filler dumps the status of readout when it is sent out for diagnostic purpose. A 2-bit Event Buffer Status (EBS) is the 2 MSB of number of hits left in event buffer. The definitions of the EBS bits are shown in **Table 3**. [1]

| ESB | Meaning |
|---|---|
| 00 | Hits in buffer is less than 32. |
| 01 | Hits in buffer is between 32 to 63. |
| 10 | Hits in buffer is between 64 to 95. |
| 11 | Hits in buffer is more than 96. |

**Table 3. The definition of EBS.**

# 4 Hardware Description Language Code

My primary responsibility was to write the read-out logic for the front-end ETL on the Xilinx Kintex Ultrascale+ KU15P FPGA daughter boards placed on the Serenity board at the back-end. To achieve this, I wrote a Very High-Speed Integrated Circuit Hardware Description Language (VHDL) code to receive input bitstream and categorize it into different types (Header, Filler, Trailer, Data Word) based on specific patterns. Then it stores and processes the data accordingly, while also updating counters and output signals.

## 4.1 The Required VHDL Code Description (Optimal Case)

The VHDL code receives a stream of data, and it locks 8 bits by 8 bits chunks of the received data stream and should be able to process three chunks of 8-bit at the same time to determine the data type of the incoming bitstream from four different 40 bits long data types:

1. **Header:** first bit is 0,
followed by 15 bits of a pre-defined register called "0x3C5C",
followed by two bits 00,
followed by an incrementation counter called L1Counter which is 8-bit long to count the number of Header data types received at the FPGA, This information should appear to the user.
followed by a two-bit data type to describe the type of the Data Word (an indicator to know if this is a Regular data format is 00, Random test pattern is 01, Counter test pattern is 10, Reserved is 11). This information should appear to the user,
followed by 12 bits that will be stored in a register called "BCID". This information should appear to the user.

2. **Filler:** first bit is 0,
followed by 15 bits of a pre-defined register called "0x3C5C",
followed by two bits 01,
followed by an incrementation counter called RL_L1Counter which is 8-bit long to count the number of Filler data types received at the FPGA, This information should appear to the user.
followed by a 2-bit Event Buffer Status (EBS) which is the 2 Most Significant Bits of the number of hits left in the event buffer (an indicator to know if the Hits in the buffer is less than 32 meaning '00', Hits in the buffer is between 32 to 63 meaning '01', Hits in the buffer is between 64 to 95 meaning '10', or Hits in the buffer is more than 96 meaning '11')

followed by 12 bits that will be stored in a register called "RT_BCID". This information should appear to the user.

3. **Trailer:** first bit is 0,
followed by 17-bit that matches the Chip_ID which is "10000011010001100" for an initial value that can be changed by the user later,
followed by 6-bit of status (an indicator that describes whether: L1 Event buffer overflow meaning '000000', L1 Event buffer almost full '000001', L1 Event buffer half full '000010', Data error occur but corrected meaning '000011', Data error occur but not corrected meaning '000100', or Reserve meaning '000101'). This information should appear to the user,
followed by an incrementation counter called HITS which is 8-bit long to count the number of Data Words received at the FPGA, This information should appear to the user,
followed by 8-bit will be stored in a register called "CRC_whole_header_DATA_trailler".

4. **Data Word:** first bit is 1, followed by
followed by 2-bit will be stored in a register called "EA" (An indicator that contains four cases: If EA is ’00’, it means there is no error detected. If EA is ’01’, it means there is one bit of error that has been detected and corrected. If EA is ’10’, it means there is 2 or more bits error and the Hamming code cannot correct it), This information should appear to the user,
followed by 4-bit will be stored in a register called "Col_ID", This information should appear to the user,
followed by 4-bit will be stored in a register called "Row_ID", This information should appear to the user,
followed by 29-bit will be stored in a register called "TDC_data", This information should appear to the user.

**Important notes:**

1. Most of the time the FPGA that we are programming will receive Filler data type.
2. The Data Word follows the Header and concluded by the Trailer.

Q: How do you know if it's a Header?

A:

a. It starts with a 0.

b. The 17th and the 18th bits are 00.

Q: How do you know if it's a Filler?

A:

a. It starts with a 0.

b. The 17th and the 18th bits are 01.

Q: How do you know if it's a Trailer?

A:

a. It starts with a 0.

b. The first 17 bits match the “Chip_ID”.

Q: How do you know if it's a Data Word?

A:

a. It comes right after the Header data type (after the 40th bit of the header).

b. It starts with a 1.

Q: Does the first bit of any of the data types have to be at the beginning of the 8-bit chunks?

A:

Not necessarily, the beginning of the 40 long data types could be anywhere in the 8 bits. Also, it could not be in this particular 8-bit chunk therefore we have to keep looking for the next one in the coming chunks.

Q: After we have identified the data type, what should we do with it?

A: If the data type is a Filler; ignore it. But if it is a Header, we have to store the 40-bit Header, the 40-bit Data Word, and the 40-bit Trailer that follows it.

## 4.2 The VHDL Code

The following is my attempt to achieve the optimal required VHDL code. It's important to note that the code may require further refinement before it is ready for implementation, as there are currently some synthesis errors that need to be addressed:

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity DataProcessor is
    port (
        clk : in std_logic;
        reset : in std_logic;
        data_in : in std_logic_vector(7 downto 0);
        L1Counter : out std_logic_vector(7 downto 0);
        RL_L1Counter : out std_logic_vector(7 downto 0);
        BCID : out std_logic_vector(11 downto 0);
        RT_BCID : out std_logic_vector(11 downto 0);
        HITS : out std_logic_vector(7 downto 0);
        CRC_whole_header_DATA_trailler : out std_logic_vector(7 downto 0);
        EA : out std_logic_vector(1 downto 0);
        Col_ID : out std_logic_vector(3 downto 0);
        Row_ID : out std_logic_vector(3 downto 0);
        TDC_data : out std_logic_vector(28 downto 0)
    );
end entity;

architecture rtl of DataProcessor is

    -- States for tracking data types:
    type DataType is (Unknown, Header, Filler, Trailer, DataWord);
    signal current_type : DataType := Unknown;
    signal next_type : DataType := Unknown;

    -- Registers for storing data:
    signal header_reg : std_logic_vector(39 downto 0);
    signal filler_reg : std_logic_vector(39 downto 0);
    signal trailer_reg : std_logic_vector(39 downto 0);
    signal data_word_reg : std_logic_vector(39 downto 0);

    -- Counters for L1Counter and RL_L1Counter:
    signal l1_counter_reg : unsigned(7 downto 0) := (others => '0');
    signal rl_l1_counter_reg : unsigned(7 downto 0) := (others => '0');

    -- Buffer for storing previous 8-bit chunks:
    signal prev_data_in : std_logic_vector(7 downto 0) := (others => '0');

begin
```

```
43 begin
44     process (clk)
45     begin
46         if rising_edge(clk) then
47             if reset = '1' then
48 
49                 -- Reset all signals and registers:
50                 current_type <= Unknown;
51                 next_type <= Unknown;
52                 header_reg <= (others => '0');
53                 filler_reg <= (others => '0');
54                 trailer_reg <= (others => '0');
55                 data_word_reg <= (others => '0');
56                 l1_counter_reg <= (others => '0');
57                 rl_l1_counter_reg <= (others => '0');
58                 prev_data_in <= (others => '0');
59             else
60 
61                 -- Determine the next data type based on the current and
   previous inputs:
62                 case current_type is
63                     when Unknown =>
64                         if data_in(0) = '0' and prev_data_in(0) = '0' and
   prev_data_in(1) = '0' then
65                             next_type <= Header;
66                         elsif data_in(0) = '0' and prev_data_in(0) = '0' and
   prev_data_in(1) = '1' then
67                             next_type <= Filler;
68                         elsif data_in(0) = '1' and prev_data_in(0) = '1' then
69                             next_type <= DataWord;
70                         else
71                             next_type <= Unknown;
72                         end if;
73 
74                     when Header =>
75                         if prev_data_in(0) = '0' then
76                             next_type <= Unknown;
77                         elsif prev_data_in(0) = '1' then
78                             next_type <= DataWord;
79                         end if;
80 
81                     when Filler =>
82                         if prev_data_in(0) = '0' then
83                             next_type <= Unknown;
84                         end if;
85 
86                     when Trailer =>
```

```
                    when Trailer =>
                        if prev_data_in(0) = '0' then
                            next_type <= Unknown;
                        elsif prev_data_in(0) = '1' then
                            if data_in(0) = '0' then
                                next_type <= Header;
                            else
                                next_type <= Trailer;
                            end if;
                        end if;

                    when DataWord =>
                        if prev_data_in(0) = '0' then
                            next_type <= Unknown;
                        end if;
                end case;

                -- Store the current input based on the data type:
                case current_type is
                    when Header =>
                        header_reg <= prev_data_in & data_in;
                        l1_counter_reg <= l1_counter_reg + 1;

                    when Filler =>
                        filler_reg <= prev_data_in & data_in;
                        rl_l1_counter_reg <= rl_l1_counter_reg + 1;

                    when Trailer =>
                        trailer_reg <= prev_data_in & data_in;
                        CRC_whole_header_DATA_trailler <= prev_data_in(7
downto 1) xor data_in(7 downto 1);

                    when DataWord =>
                        data_word_reg <= prev_data_in & data_in;
                        l1_counter_reg <= l1_counter_reg + 1;
                end case;



                -- Update current type with the next type:
                current_type <= next_type;

                -- Update output signals:
                L1Counter <= std_logic_vector(l1_counter_reg);
                RL_L1Counter <= std_logic_vector(rl_l1_counter_reg);
                BCID <= header_reg(19 downto 8);
                RT_BCID <= header_reg(39 downto 28);
                HITS <= data_word_reg(7 downto 0);
                EA <= data_word_reg(9 downto 8);
                Col_ID <= data_word_reg(15 downto 12);
                Row_ID <= data_word_reg(11 downto 8);
                TDC_data <= data_word_reg(28 downto 0);

                -- Update previous input:
                prev_data_in <= data_in;
            end if;
        end if;
    end process;
end architecture;

```

# 5 Conclusion and Future Work

In conclusion, the firmware design for particle timing measurements in the CMS ETL is crucial for successfully including the MIP Timing Detector in the HL-LHC era upgrade plan. Its primary objective is to accurately assign charged tracks to the correct interaction vertices within bunch crossings containing up to 200 collisions. To achieve this, resolving any synthesis errors in the VHDL code before proceeding with simulations and implementation is essential.

# References


[1] ETROC Design Team, "ETROC2 Reference Manual," March 13, 2023. [Online]. Available: https://etl-rb.docs.cern.ch/.

[2] W. Zhang, H. Sun, C. Edwards, D. Gong, X. Huang, C. Liu, T. Liu, T. Liu, J. Olsen, Q. Sun, X. Sun, J. Wu, J. Ye, and L. Zhang, "A low-power time-to-digital converter for the cms endcap timing layer (etl) upgrade," *IEEE Transactions on Nuclear Science*, vol. 68, no. 8, p. 1984–1992, 2021.

[3] H. Sun, D. Gong, C. Edwards, G. Huang, X. Huang, C. Liu, T. Liu, T. Liu, J. Olsen, Q. Sun, J. Wu, J. Ye, L. Zhang, and W. Zhang, "In-pixel automatic threshold calibration for the cms endcap timing layer readout chip," SEP 2021. [Online]. Available: https://dx.doi.org/10.1088/1748-0221/16/09/T09006.

[4] CMS Collaboration, "A MIP Timing Detector for the CMS Phase-2 Upgrade," 2019. [Online]. Available: https://cds.cern.ch/record/2667167?ln=en.

[5] Rose, Andrew William and Parker, Duncan and Iles, Gregory and Sahin, Ozgur and Bausson, Pierre-Anne and Tsirou, Andromachi and Fedi, Giacomo and Verdini, Pierro-Giorgio and Ardilla, Luis and Balzer, Matthias and Schuh, Thomas and Wil, "Serenity: An ATCA prototyping platform for CMS Phase-2," *PoS*, vol. TWEPP2018, p. 115, 2019.

[6] Jan Troska, Alexander Brandon-Bravo, Stéphane Detraz, Andrea Kraxner, LauriOlanterä, Carmelo Scarcella, Christophe Sigaud, Csaba Soòs, and François Vasey, "The VTRx+, an optical link module for data transmission at HL-LHC," 20 March 2018. [Online]. Available: https://doi.org/10.22323/1.313.0048.

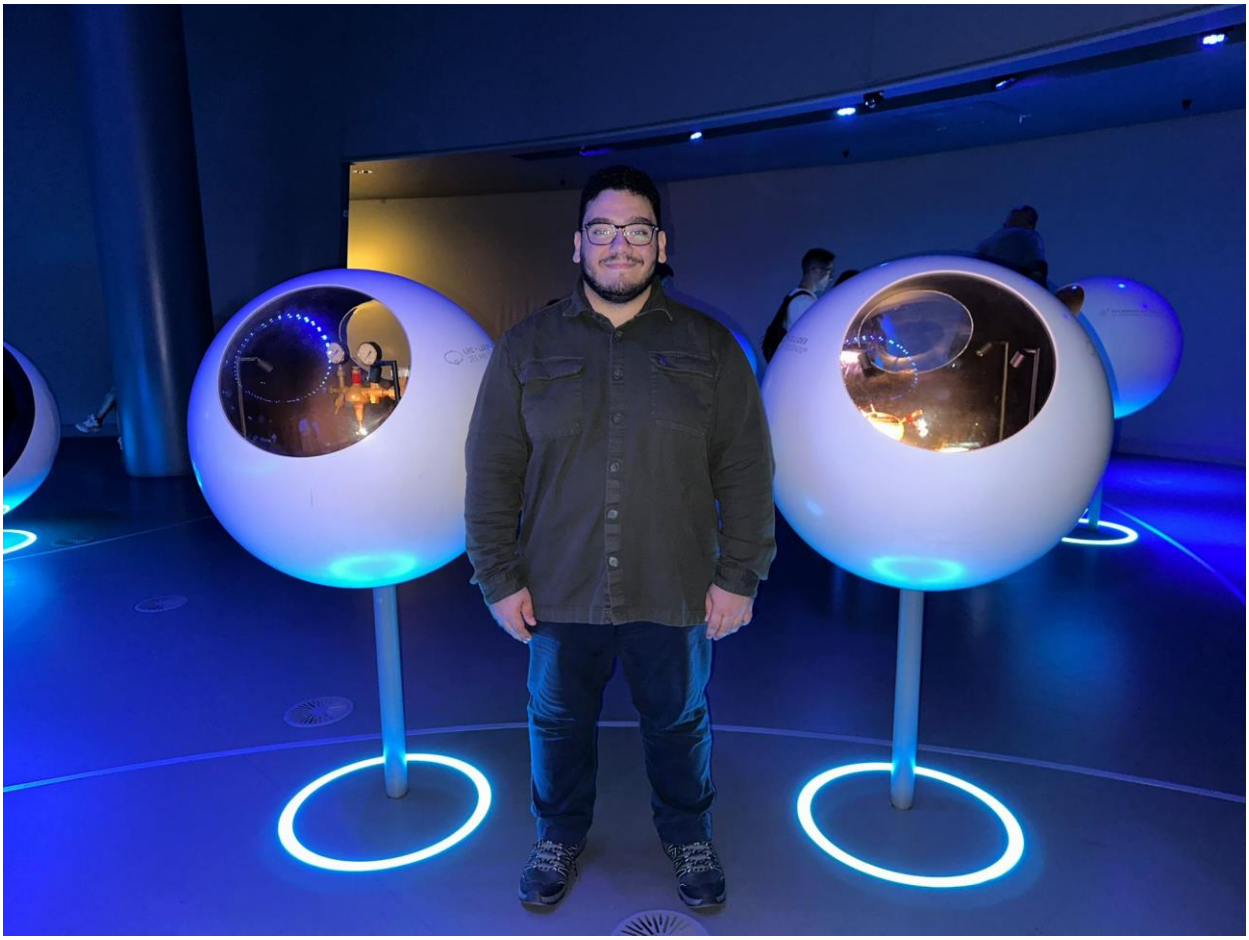
# Appendix A - Captured Moments from My Training Experience



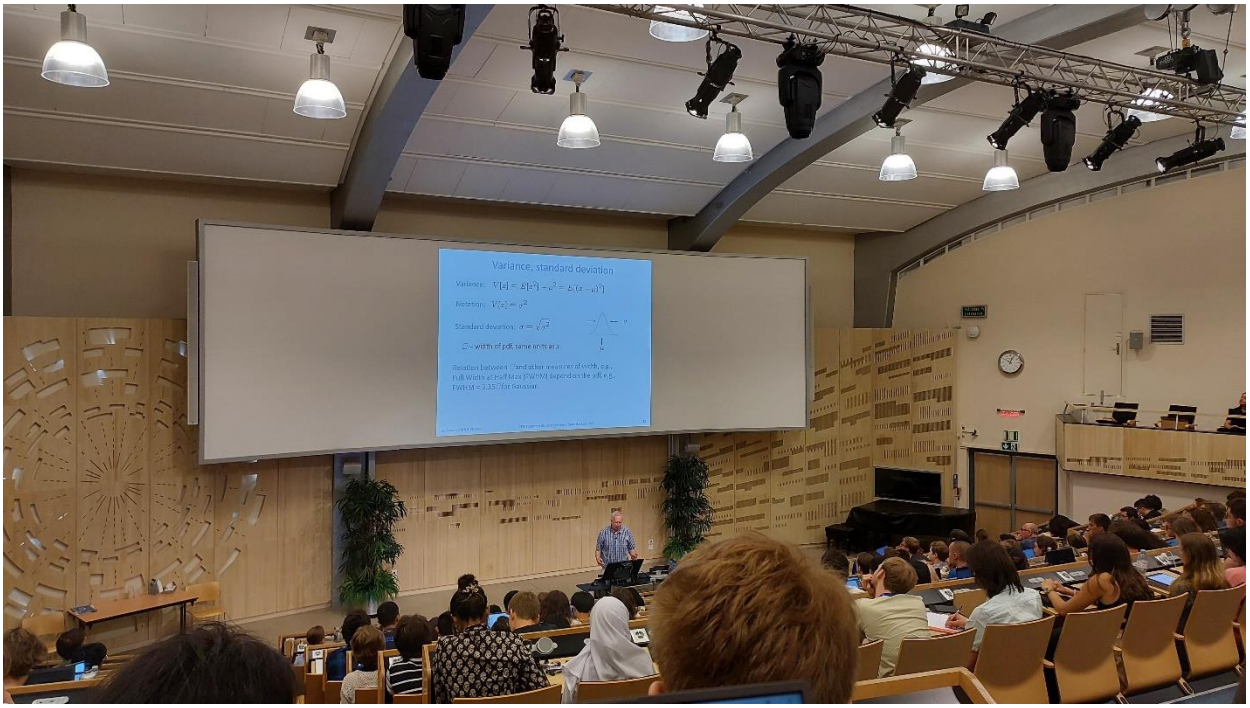
Where innovation is, Bahrain is.

Radiation emergency kit.

Globe of Science and Innovation.

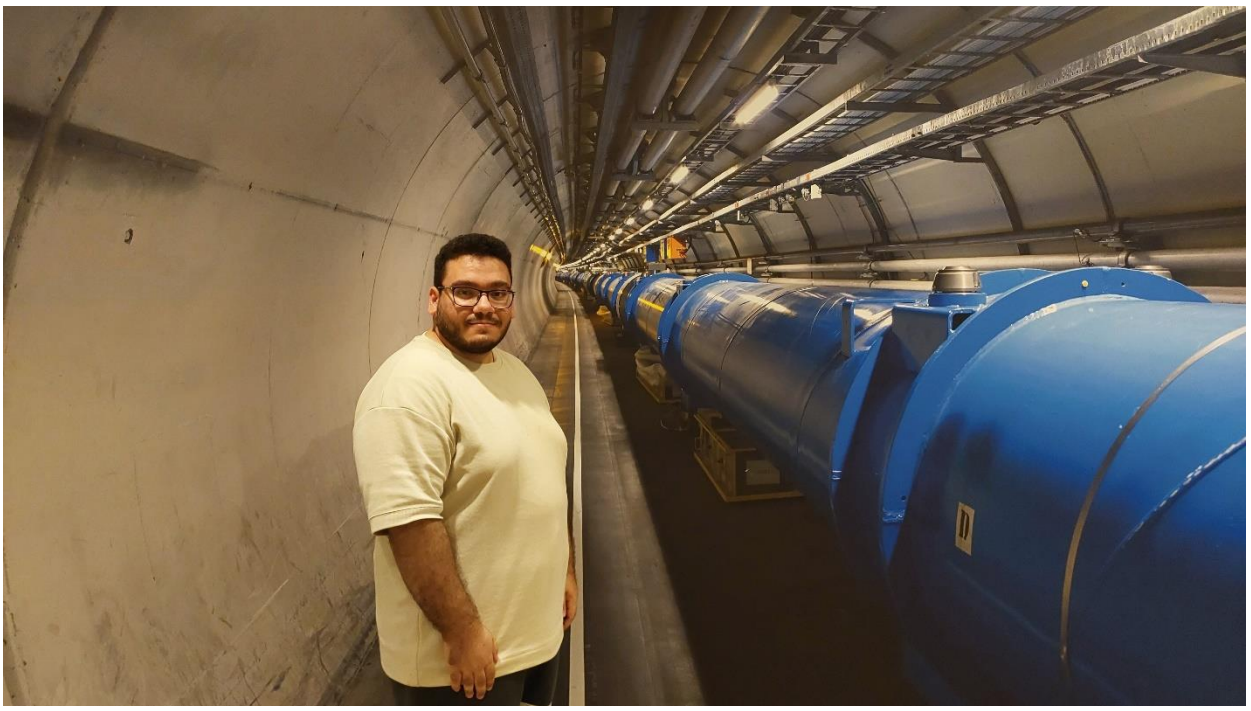


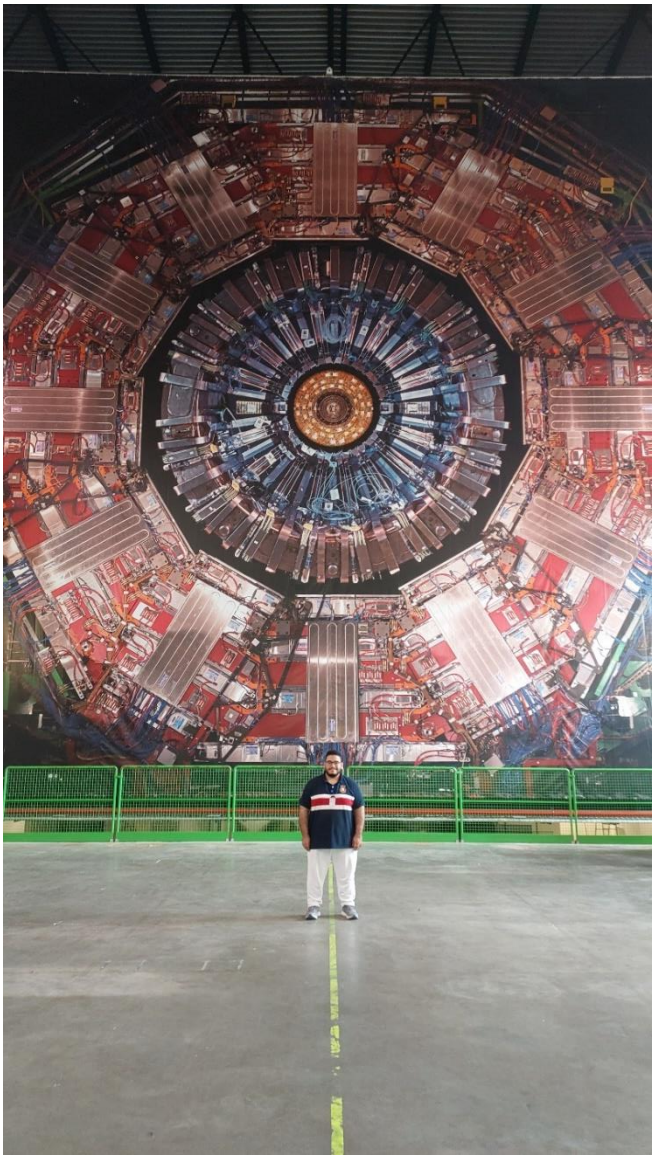
Inside The Globe.




The Main Auditorium, where scientists from around the world share their knowledge about a wide range of topics in the fields of theoretical and experimental particle physics and computing.
In this image, Glen Cowan, professor of Particle Physics at Royal Holloway, University of London is giving us, summer students, the Foundation of Statistics lecture.

# L H C

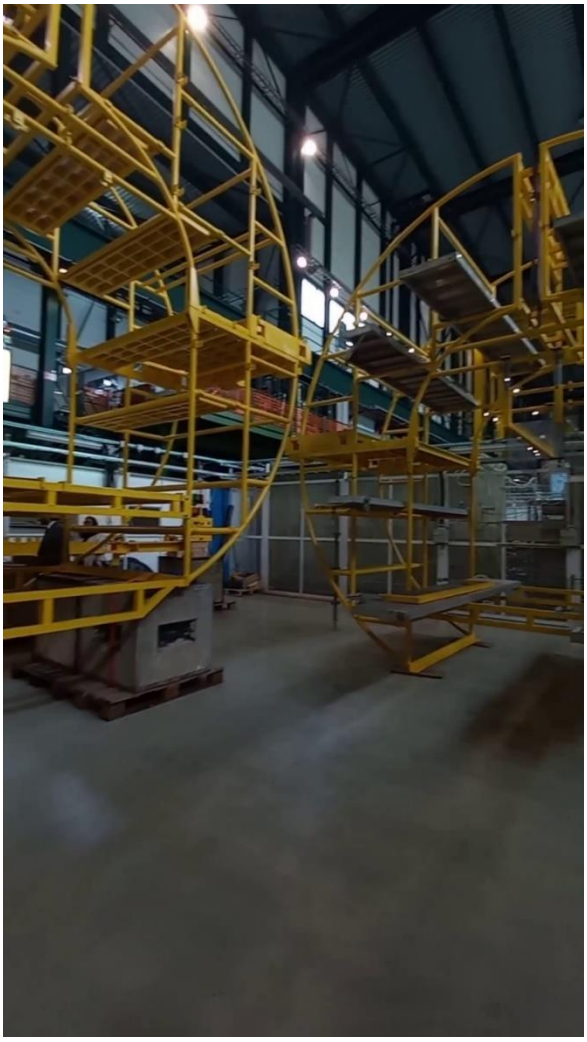


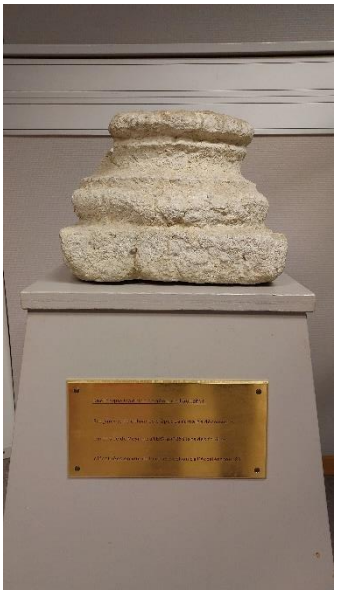

CHAMP MAGNETIQUE
DANGER
MAGNETIC FIELD
YCA01
UPX56
RADIATION
SUPERVISED AREA
ZONE SURVEILLÉE

# C M S

# Made in Bahrain

Automated Jig for work platform installation into the CMS detector.
By Eng. Abdel-Salam Suleiman and Prof. Zuhair Khalifa Bahri of University of Bahrain

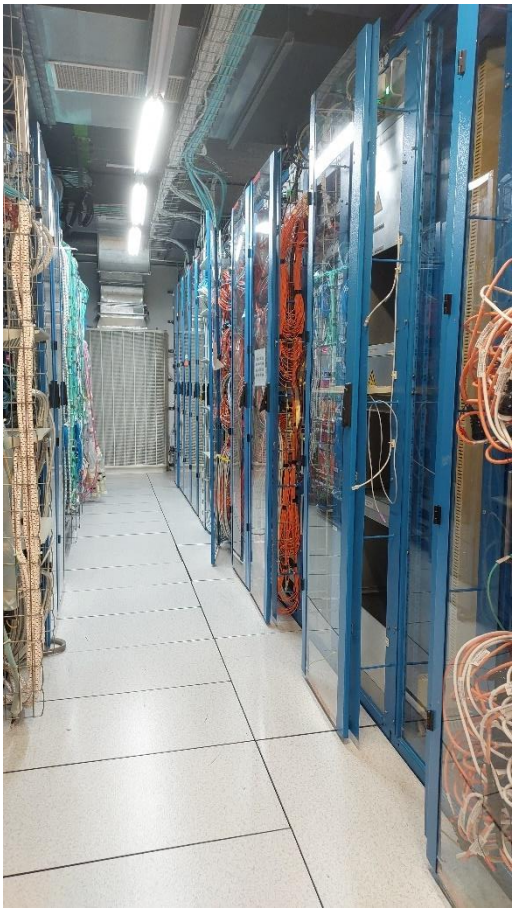


Roman pillar ruins discovered at CMS dig site.
While building a great civilization, they found the remains of a previous one.


With my supervisor, the MTD DAQ Team Admin Özgür Sahin, and two of my teammates and dear friends Akbar and Abdul-Rahman.

Wall of the brilliant minds behind the CMS.



CMS Data Acquisition center.


# Social Gatherings

Dr. Clara Nellist.

Niels Bohr's Statue.