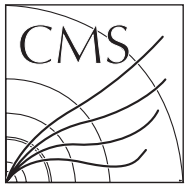
Available on CMS information server

CMS CR -2016/114



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

31 May 2016 (v4, 07 June 2016)

Emulation of a prototype FPGA track finder for the CMS Phase-2 upgrade with the CIDAF emulation framework.

C. Amstutz, F. A. Ball, M. N. Balzer, J. Brooke, L. Calligaris, D. Cieri, E. J. Clement, G. Hall, T. R. Harbaum, K. Harder, P. R. Hobson, G. M. Iles, T. James, K. Manolopoulos, T. Matsushita, A. D. Morton, D. Newbold, S.Paramesvaran, M. Pesaresi, I. D. Reid, A. W. Rose, O. Sander, T. Schuh, C. Shepherd-Themistocleous, A.Shtipliyski, S. P. Summers, A. Tapper, I. Tomalin, K. Uchida, P. Vichoudis, M. Weber for the CMS Collaboration

Abstract

The CMS collaboration is preparing a major upgrade of its detector, so it can operate during the high luminosity run of the LHC from 2026. The upgraded tracker electronics will reconstruct the trajectories of charged particles within a latency of a few microseconds, so that they can be used by the level-1 trigger. An emulation framework, CIDAF, has been developed to provide a reference for a proposed FPGA-based implementation of this track finder, which employs a Time-Multiplexed (TM) technique for data processing.

Presented at *IEEE-RT2016 IEEE-NPSS Real Time Conference (RT)*

Emulation of a Prototype FPGA Track Finder for the CMS Phase-2 Upgrade with the CIDAF Emulation Framework

C. Amstutz[‖], F. A. Ball[‡], M. N. Balzer[‖], J. Brooke[¶], L. Calligaris[*†], D. Cieri[*†‡], E. J. Clement[‖], G. Hall[¶], T. R. Harbaum[‖], K. Harder[†], P. R. Hobson[§], G. M. Iles[¶], T. James[¶], K. Manolopoulos[†], T. Matsushita[††], A. D. Morton[§], D. Newbold[†‡], S. Paramesvaran[‡], M. Pesaresi[¶], I. D. Reid[§], A. W. Rose[¶], O. Sander[‖], T. Schuh[‖], C. Shepherd-Themistocleous[†], A. Shtipliyski[¶], S. P. Summers[¶], A. Tapper[¶], I. Tomalin[†], K. Uchida[¶], P. Vichoudis[**], M. Weber[‖]

for the CMS collaborators

CIDAF email contact: luigi.calligaris@stfc.ac.uk

***Abstract*—The CMS collaboration is preparing a major upgrade of its detector, so it can operate during the high luminosity run of the LHC from 2026. The upgraded tracker electronics will reconstruct the trajectories of charged particles within a latency of a few microseconds, so that they can be used by the level-1 trigger. An emulation framework, CIDAF, has been developed to provide a reference for a proposed FPGA-based implementation of this track finder, which employs a Time-Multiplexed (TM) technique for data processing.**

***Index Terms*—LHC, HL-LHC, CMS, Phase-2 upgrade, track trigger, Hough transform, emulation, FPGA, VHDL, CIDAF**

I. Introduction

THE High-Luminosity LHC (HL-LHC) is the name given to a planned upgrade of the Large Hadron Collider at CERN, Geneva, currently expected to be completed in year 2026.

After this upgrade the instantaneous luminosity of pp collisions will increase at the high-luminosity interaction points with respect to current nominal values, reaching a value of $5 \cdot 10^{34}$ cm^{-2} s^{-1} (see Fig. 1), with the possibility of an additional increase later on if the machine performance proves satisfactory. The CMS experiment [1] is located at one of these high-luminosity interaction points.

As the time interval at which proton bunches cross head-on in the LHC machine to produce collisions (25 ns) will not be changed in the upgrade, the increase in luminosity will have to follow from an increase in the number of pp collisions taking place in each bunch crossing. In the rare cases where one of these collisions produces an interesting physics process (*primary collision*), the other collisions (*pile up collisions*) which occurred in the same bunch crossing act as a source of background, complicating the reconstruction and identification of the primary event.

The new conditions at the HL-LHC will entail a change from the pre-upgrade values of around $20-25$ in the number of pile-up (PU) collisions to design values of the order of 140 and, in case the machine allows operation beyond the design performance, these values could reach up to 200 pile-up collisions.

The described conditions are significantly more difficult than the ones the detectors have been initially designed for. As an example, the increased number of charged tracks translates to a larger number of hits that will need to be read out from the tracking detectors, increasing the bandwidth needed between detector modules and off-detector front end cards. To adapt the experiments to the new conditions, upgrades are therefore necessary. In the case of the CMS experiment, the upgrade is referred to as Phase-2 Upgrade [2].

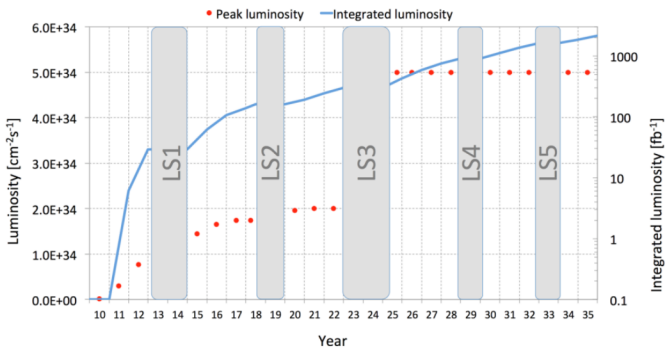


Fig. 1. Projections for instantaneous and integrated luminosity at the LHC. The HL-LHC phase begins after the Long Shutdown 3 (LS3) [2].

† STFC Rutherford Appleton Laboratory, Didcot, UK
‡ University of Bristol, Bristol, UK
§ Brunel University London, Uxbridge, UK
¶ Imperial College London, London, UK
‖ Karlsruhe Institute of Technology, Karlsruhe, Germany
** CERN, Geneva, Switzerland
†† Österreichische Akademie der Wissenschaften, Vienna, Austria

* Supported by EU-FP7-PEOPLE-2012-ITN project nr. 317446, INFIERI "Intelligent Fast Interconnected and Efficient Devices for Frontier Exploration in Research and Industry"

978-1-5090-2014-0/16/$31.00

II. Tracks at Trigger Level-1

The CMS trigger system is responsible for reducing the rate of data saved to permanent storage from the rate at which proton bunch collisions take place down to rates which are acceptable for the computing infrastructure. This is achieved by cascading two distinct trigger systems with different architectures: a fast *Level 1* (L1) trigger, implemented using a combination of custom, radiation-hard, on-detector ASICs and off-detector FPGAs, and a *High Level Trigger* implemented in software, running on a farm of commercial-off-the-shelf computers, which benefits from the flexibility and increased sophistication offered by the software implementation. In this paper we focus on the former of the two trigger systems.

While the L1 input event rate corresponds to the collision rate of 40 MHz and is fixed by the LHC machine parameters, the maximum event output rate of the L1 trigger is set by the input capabilities of the HLT, and for the Phase-2 L1 trigger the requirement is 500 kHz for 140 pile-up collision conditions and 750 kHz for 200 pile-up collision conditions [2]. A trigger decision on whether an event should be accepted into the HLT has to happen within 12.5 μs [2].

Using track information at L1 can add a significant improvement to the performance of the trigger, making it more selective and reducing the triggering rate. For this reason, CMS intends to reconstruct tracks at L1 and make them available to the trigger [2]. An example of this improvement can be seen from the simulations shown in Fig. 2, where rates for muon triggers not employing track information decrease slowly as a function of the muon p_T thresholds, making rates difficult to control, whereas rate reductions are effectively achieved in the case of triggers employing L1 track matching to the muons. This improvement is particularly significant in the forward region, corresponding to regions of higher pseudorapidity $|\eta|$ in the CMS coordinate system [1].

Track reconstruction at L1 [3] will use data from the upgraded outer tracker [2], where the latter will be composed of double-layer silicon detectors. The layers are separated by a small distance of the order of a few mm, in a configuration known as *"p_T-module"* [4]. In this setup, the two layers are read out by the same front-end ASIC making possible to correlate pairs of hits, where one hit is measured in one layer and the other is measured in the opposite one.

The p_T-module configuration allows to get a rough estimate of the p_T of the candidate track associated to a pair of hits, from the separation in ϕ of the two hits. This is done by exploiting, as a result of the Lorentz force acting on the charged particle, the relationship between the track transverse momentum, the angle of impact on the detector, the radial distance of the latter from the origin of tracks and the intensity of the magnetic field in the tracking volume, which is parallel to the z axis in the CMS reference frame.

Only pairs of hits in opposite layers, whose relative positions are compatible with a charged particle, originating from the center of the detector and having a p_T larger than a preset threshold (in this case $2-3$ GeV/c), are streamed off-detector to the L1 track reconstruction electronics. Pairs

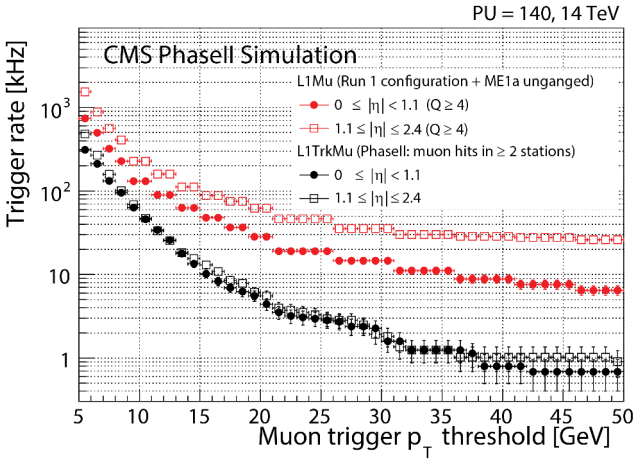


Fig. 2. Comparison of expected trigger rates at 140 pile-up collisions for single muons, as a function of the muon p_T trigger threshold, for pseudorapidity $|\eta| < 1.1$ (full dots) and $1.1 < |\eta| < 2.4$. The rates for a trigger using only muon chamber information are shown in red, while the rates for muons matched to L1 tracks are shown in black. Data from simulation [2].

passing the threshold are denominated *"stubs"*. Hits not passing the mentioned selection, instead, are streamed off-detector only upon receiving a L1 trigger signal. The p_T-module arrangement suppresses low-p_T tracks – which make up the bulk of tracks produced in a collision – bringing down to an acceptable value the requirements in terms of output bandwidth to the L1 track reconstruction electronics and of corresponding processing power.

III. Hough Transform Track Finder

We focus on a proposed implementation [5][6] of a track reconstruction system using a Hough Transform [7] for finding the tracks. A hardware demonstrator [8] is currently being built to prove the feasibility of the concept and to study its behavior with simulated events. This hardware demonstrator is built from MP7 cards [9][10], which are generic stream processor MicroTCA cards based on a Xilinx Virtex-7 XC7VX690T FPGA, coupled to 72 input and 72 output 10.3125 Gb/s optical links. A single Virtex-7 cannot contain the whole track finding processor logic, therefore the demonstrator uses a number of MP7 cards, each implementing a part of the algorithm, daisy chained via the optical links. It is expected that future FPGA technology will allow us to fit one entire track finding processor in one single chip. The Hough Transform method is briefly described below.

In the region of CMS occupied by the silicon tracker the magnetic field is uniform and normal to the $r\phi$ plane, so charged particles describe circular trajectories in this plane. For tracks having a transverse momentum in the range of interest ($p_T > 3$ GeV) the arc described by the trajectory in the tracker volume has a small deviation from a straight line, so the following approximation is valid:

$$\phi_T \simeq \phi_{\text{stub}} - C \cdot (q/p_T) * (r_{\text{stub}} - T) \qquad (1)$$

where q/p_T is the charge-to-transverse momentum ratio of the track, C is a constant proportional to the magnetic field

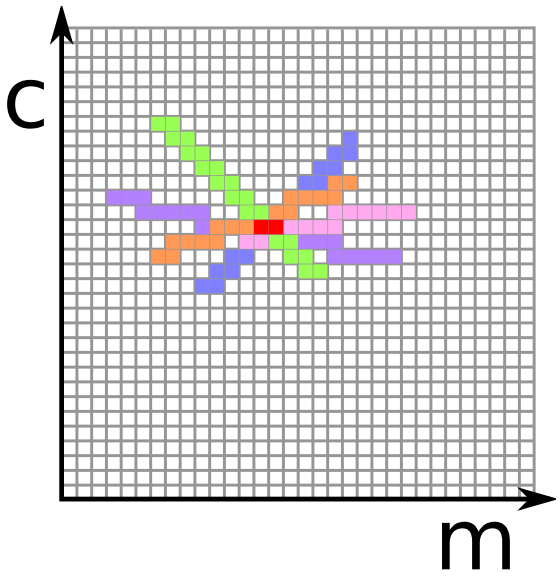


Fig. 3. An illustration of the Hough transform histogram, showing lines corresponding to 5 stubs crossing in the bins highlighted in red, where candidate tracks are found.

$\vec{B}$, ϕ_{stub} and r_{stub} are the azimuthal angle and radius from the origin of the stub (i.e. the proton bunch collision point in the $r\phi$ plane), T is a reference radius (in our case $T = 58$ cm) and ϕ_T is the azimuthal angle at which the track would cross the reference circle at radius T. This relationship can be rewritten by choosing $r_T = r_{\text{stub}} - T$ to get Eqn. 2:

$$c = \phi_{\text{stub}} - m * r_T \tag{2}$$

where m is proportional to q/p_T and c is equal to ϕ_T. It is possible then to represent each stub (*Hough Transform*) as a line described by this linear relation in the (m, c) plane. A collection of stubs consistent with a single track yields lines that cross at a single point in the (m, c) plane.

In the track finder, a histogram in the (m, c) plane counts the number of stubs in each bin consistent with the p_T estimate from the dual layer of the tracker p_T module. The latter consistency check is referred to as "*bend filtering*" and reduces the number of fake tracks found by the histogram. A candidate track is found in a (m, c) bin when it contains 5 or more stubs from different layers of the tracker, as shown in Fig. 3. The cut at 5 layers is a compromise between the number of tracker layers producing stubs when traversed by a track (there are 6 layers in the barrel region and 5 endcap disks on each side) and the rate of false positives.

Three firmware architectures have been proposed to implement the Hough Transform histogram:

- A systolic array, where each cell of the array represents a cell in the histogram and performs the calculation independently from other cells.
- A pipelined design, where the calculation of all the possible (m, c) pairs for each stub is performed in advance, and then used as an address to fill the cells of the histogram.
- An architecture (the "*daisy-chained*" design) sharing commonalities with the other two, where columns of the histogram perform the calculation from the result of the previous one and pass stubs to the next column, such that the columns are arranged in a pipelined daisy chain.

All three firmwares are built from the composition of multiple instances of a limited number of basic blocks. Since an emulator is needed to study the expected behavior of each firmware, an emulation framework that enables the composition of several emulated firmware blocks was developed. This framework, written in C++ language, has been named CIDAF, which stands for *CIrcuit DAta Flow* and is being presented for the first time in this manuscript.

In the context of the track reconstructor system demonstrator, emulators developed using CIDAF are now complementing the previously existing emulators of the Hough Transform track finder, which were implemented in CMSSW (the latter is the analysis software used by the CMS experiment), by adding the capability of studying the firmware state at each clock cycle and replicating the exact firmware behavior specified in the firmware VHDL source, such as in the case of buffer overflows.

IV. The CIDAF Emulation Framework

CIDAF is based on a C++ library that provides facilities to help the composition of individual instances of emulated blocks, taking care of the clocking of the blocks to evolve their internal state and the transfer of data between blocks. It is currently built using CMake [11], as this build system is flexible, available on a variety of platforms and supported by many IDEs. Development and testing is currently being performed under Linux x86_64 (gcc 4.7 and later, clang 3.2 and later) and Mac OSX (clang 3.2 and later).

CIDAF is independent from other packages apart from CMake and BOOST – the latter of the two being kept as a dependence to give the possibility to use it in future revisions. The reduced dependence from other packages allows it to be easily included in other applications such as detector and trigger simulations. Preliminary tests showed that CMSSW can be linked against CIDAF and can run an instance of a dummy emulator in its main process.

A. Component Clocking

The *Clockable* abstract base class describes the interface that a class emulating a component needs to define in order to be clocked by the CIDAF clocking system, this consists of one method named *Tick()*. The latter specifies what a component should do to evolve its internal state upon receiving a clock tick. Clockable also has facilities to register and unregister to clocks, which are described below.

The *Clock* class contains a set of pointers to Clockable objects that have been registered into it, and calls their *Tick()* method when its own *Tick()* method is called. There can be an arbitrary number of clocks in CIDAF, giving full flexibility if a particular clocking order of the components is required by the emulated design.

CIDAF's clock system allows a clock-accurate emulation of the behavior of a firmware, a feature that has been demonstrated while emulating the Hough Transform track finder (see below).

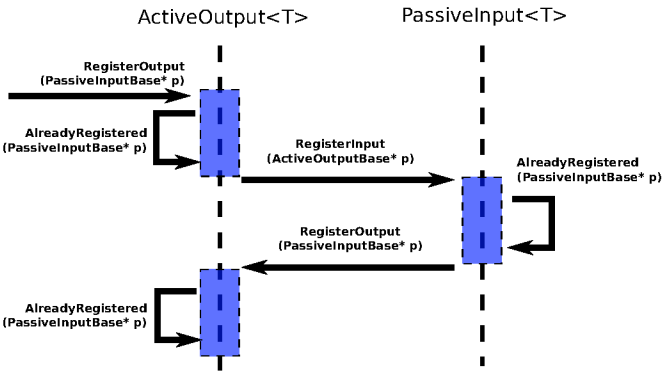


Fig. 4. Sequence diagram showing connection establishment between an ActiveOutput and a PassiveInput object, both having their template argument, representing the type of data being transferred, set to the type T.

B. Component Input and Output

CIDAF provides a standard facility to connect components together in order for them to transfer data, this is the *ActiveOutput / PassiveInput* class pair. These classes are templated on the type of data being transferred. The connections are established at run-time, components can be disconnected at a later time and connections are terminated cleanly upon object destruction. This allows for a future addition of a graphical user interface in the future.

When developing a class emulating a firmware component, the developer wishing to add output capabilities to the class adds as a data member of the class an ActiveOutput< T >, with T the type of data being transferred. Similarly, to add an input to a class, a PassiveInput< T > is added as a data member of the class.

An important distinction between ActiveOutput and PassiveInput is that the former is clocked to initiate a transfer, while the latter acts as a passive slot into which ActiveOutputs can write. The possibility of independently driving the transfer between components from the evolution of the internal state of these components protects against the need of dynamically finding which component should be clocked first, and the order of transition of the ones that should follow (a problem that may prove very difficult to solve or outright not have a solution in complex designs with feedback loops).

CIDAF employs a double-handshake connection establishment between ActiveOutputs and PassiveInputs. By being able to access the other partner in the connection, each object is capable of terminating it cleanly in case it is destroyed or required to disconnect, and re-handshaking it in case it is moved to another memory address. This solution prevents undefined behavior at run-time. A diagram showing the connection establishment procedure is shown in Fig. 4. A similar double-handshake system is in place for connections established between Clockable classes and Clocks.

C. Treatment of Integers with Fixed Size

Signed and unsigned integral types, with widths from 1 to 64 bits, are available in CIDAF to match the behavior of signed and unsigned integers available in the hardware description language. A series of helper functions is also available to standardize operations, such as reversal of the representation of a number (from most significant bit first to least significant bit first), generate bit-masks and truncate a signed integer number to a smaller width integer number.

V. Emulation of the Daisychained HT with CIDAF

The daisy-chained firmware implementation of the Hough Transform is highly modular. An illustration of the structure of a single segment in this firmware is shown in Fig. 5.

Stubs are routed from the input optical link pair assigned to the considered segment into a *Book Keeper* component. The Book Keeper unpacks the data from the link, stores the entire stub data into a RAM for further retrieval if the stub is found to be on a track, reduces the stub data to the subset needed for the histogram computation and sends this latter data down a daisy chain of "*Bins*". Each of these "Bins" represents one column of cells, bounded by two values of q/p_T, within the HT array shown in Fig. 3. The ordering of the numbering of the bins is not sequential and is designed to give priority in output to higher p_T tracks.

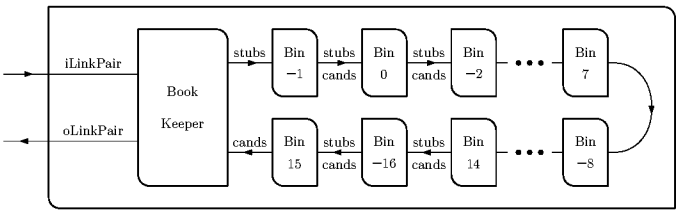


Fig. 5. An illustration of the daisy-chained firmware implementing the Hough Transform track finder.

The main task in developing an emulation for the firmware was the development of an emulation for the single segment, as the segments operate independently from each other. Five C++ classes are used to model the whole firmware: the Book Keeper, the Bin, the Segment and the whole Hough Transform processing card. Each of these classes is equipped with ActiveOutputs and PassiveInputs transferring data between the components. Two clocks are used, one evolves the internal state of the components and one triggers the transfer of data between the components. They are sequentially triggered to perform a full emulation cycle.

Adopting a consistent programming style while developing an emulation of the firmware in C++ helped reducing the frequency of bugs and allowed to speed up development. This style is now described briefly.

Experience showed that decomposition of the C++ firmware emulation into smaller and smaller classes produced code which was fragile from the standpoint of maintenance. Indeed, implementing changes to the emulation following changes to the firmware is simpler when the part subjected to modifications is contained inside one single C++ class. It was found that the optimal decomposition of the firmware into independent emulation C++ classes corresponds to one where the interfaces between components are expected to be subjected to few changes during the development of the firmware and the emulation.

A C++ class modeling the behavior of a individual firmware component should have at least three methods. One method emulates the asynchronous processes of the component, another method emulates the synchronous processes and a third one manages the evolution of signals whose state changes in the synchronous process.

Each signal declared in the HDL has two corresponding variables in the C++ code. One of these variables represents the current state of the signal and the other one represents the state in which the signal will be after the end of the synchronous process. Decoupling the current and the "next" state removes the need of finding the correct order of state changes of the variables arising from their inter-dependence. After the execution of the class method modeling the synchronous process, a dedicated method takes care of copying the state of the "next" variables into the "current" ones.

The described arrangement does not apply to signals whose state change is defined in the asynchronous parts of the HDL. In that case, the state changes of variables representing the signals should follow the correct order of dependence and assignments to "next" variables should be followed by immediate commits of its value to the "current" variables. Keeping the different treatment of variables in two separate methods (the synchronous and the asynchronous one) makes the source code more tidy and less error-prone.

VI. Results

The output from the emulator and the output from the Hough Transform daisy chain firmware, running on an MP7 card, have been compared for different samples of simulated Monte Carlo events. The agreement between firmware and emulation in terms of average number of tracks found per event is shown in Table I. The small discrepancy – probably the result of a different rounding behavior between the emulator and the firmware in a calculation – is being investigated and is expected to be fixed soon.

Sample (1000 evts each)	avg. # tracks / event	
	fw	emu
Muons + PU0	5.442	5.442
Muons + PU140	215.350	215.339
Top Pair + PU140	321.999	321.537
Top Pair + PU200	874.799	873.926

TABLE I
AVERAGE NUMBER OF TRACKS PER EVENT FOUND BY THE HOUGH TRANSFORM "DAISYCHAIN" FIRMWARE RUNNING ON AN MP7 CARD AND BY ITS EMULATION DEVELOPED USING CIDAF. THE AVERAGES ARE SHOWN FOR MUON AND TOP PAIR SIMULATED EVENT SAMPLES IN DIFFERENT PILE-UP COLLISION (PU) REGIMES.

In Fig. 6 the number of tracks found in different η regions is compared in the case of the emulator and of the firmware. Similarly, in Fig. 7 a comparison is shown for the number of tracks found per unit of (q/p_T), and in Fig. 8 for the number of stubs per candidate track. The plots show an almost perfect agreement between the behavior of the firmware and that of the emulator.

VII. Summary

The importance of track reconstruction at trigger level 1 for the Phase 2 upgrade of CMS has been introduced. One of the proposed demonstrator implementations uses MP7 processing cards, based on Xilinx Virtex 7 FPGAs, and employs a Hough Transform to find candidate tracks. A hardware demonstrator for the system is being prepared and emulators for it are needed, in order to compare the expected behavior with the one observed from running the firmware.

The need for a C++ emulation of the firmware behavior prompted the development of a framework – CIDAF – simplifying the development of firmware emulators. The main features of CIDAF have been presented and the results from an emulation of the latest, optimized, firmware for the Hough Transform track finder have been shown. The agreement between tracks found by the emulator and those found by the hardware is remarkable, and proves that the emulation represents correctly the behavior of the firmware.

Concerning future plans, CIDAF is currently being employed to write emulators for the other components of the demonstrator chain, and it is expected that a full emulation chain will be available by the time all the demonstrator elements will be commissioned and ready to be tested. Additionally, documentation for CIDAF is being prepared at the present time, in view of a future release of its source code in the public domain.

Acknowledgment

The research leading to these results has received funding from the People Programme (Marie Curie Actions) of the European Unions Seventh Framework Programme FP7/2007-2013/ under REA grant agreement n [317446] INFIERI "INtelligent Fast Interconnected and Efficient Devices for Frontier Exploitation in Research and Industry". This work was supported in part by the the UK Science and Technology Facilities Council, we gratefully acknowledge their support.

References

[1] S. Chatrchyan et al., *The CMS experiment at the CERN LHC. The Compact Muon Solenoid experiment*, 2008 JINST 3 S08004, http://cds.cern.ch/record/1129810.

[2] J. Butler, D. Contardo, M. Klute, J. Mans and L. Silvestris, *Technical Proposal for the Phase-II Upgrade of the CMS Detector*, CERN-LHCC-2015-010 and LHCC-P-008, http://cds.cern.ch/record/2020886.

[3] F. Palla, M. Pesaresi, A. Ryd, *Track Finding in CMS for the Level-1 Trigger at the HL-LHC*, 2016 JINST 11 C03011, *doi:10.1088/1748-0221/11/03/C03011*.

[4] C. Foudas, A. Rose, J. Jones and G. Hall, *A study for a tracking trigger at first level for CMS at SLHC*, LECC, Heidelberg, 2005.

[5] G. Hall, D. Newbold, M. Pesaresi and A. Rose, *A time-multiplexed track-trigger architecture for CMS*, 2014 JINST 9 C10034, doi:10.1088/1748-0221/9/10/C10034.

[6] D. Cieri et al., *L1 track finding for a time multiplexed trigger*, Nucl. Instrum. Meth. A824 (2016) 268-269 *doi:10.1016/j.nima.2015.09.117*.

[7] P. V. C. Hough, *Method and means for recognizing complex patterns*, US Patent US3069654 A.

[8] C. Amstutz et al., *An FPGA-based track finder at Level-1 for the CMS experiment at the High Luminosity LHC*, these proceedings and CMS Conference Report CR-2016/112.

[9] K. Compton, *The MP7 and CTP-6: Multi-hundred Gbps processing boards for calorimeter trigger upgrades at CMS*, 2012 JINST 7 C12024, doi:10.1088/1748-0221/7/12/C12024.

[10] The Imperial Master Processor, Virtex-7 (MP7), http://www.hep.ph.ic.ac.uk/mp7/.
[11] CMake Cross-Platform Build System by Kitware, https://cmake.org/.

Luigi Calligaris is a post-doctoral researcher in the Compact Muon Solenoid (CMS) group at the STFC Rutherford Appleton Laboratory, UK. Luigi is currently enrolled as an Experienced Researcher in the INFIERI Network, a Marie Curie Action of the European Unions Seventh Framework Programme (FP7). He completed a PhD in physics at the University of Hamburg, Germany in 2015, working in the CMS group at DESY Hamburg.

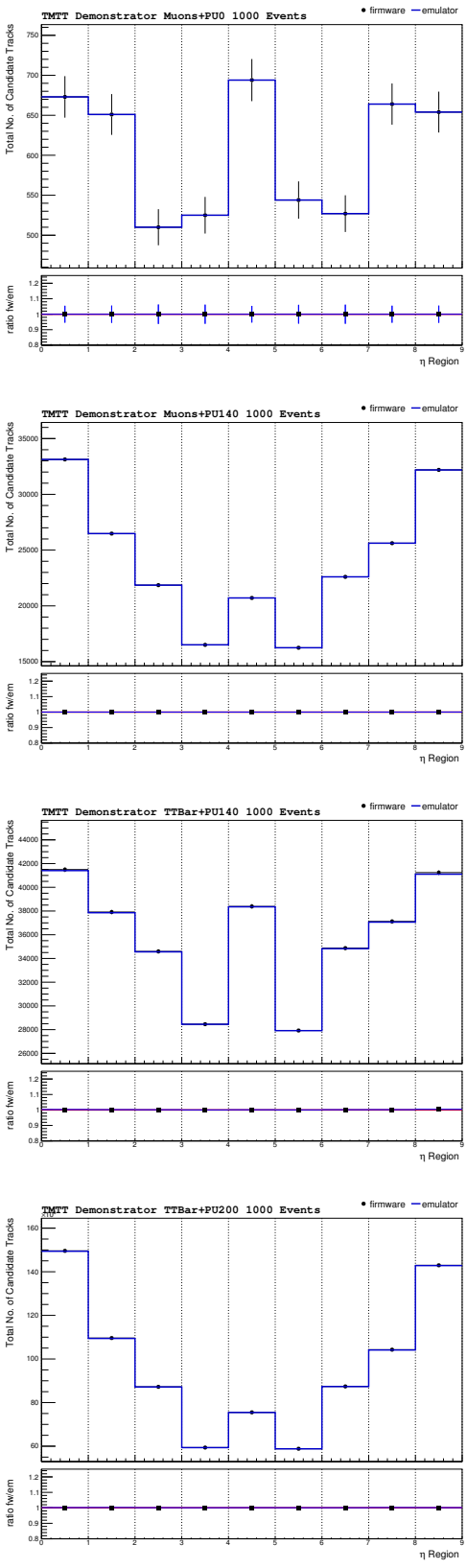


Fig. 6. Comparison between firmware running on FPGA (dots) and emulation (blue line) of the number of tracks found per pseudorapidity region for 1000 simulated events in the following samples: Single muon with 0 pile-up collisions (top), single muon with 140 pile-up collisions (upper center), $t\bar{t}$ events with 140 pile-up collisions (lower center), $t\bar{t}$ events with 200 pile-up collisions (bottom).

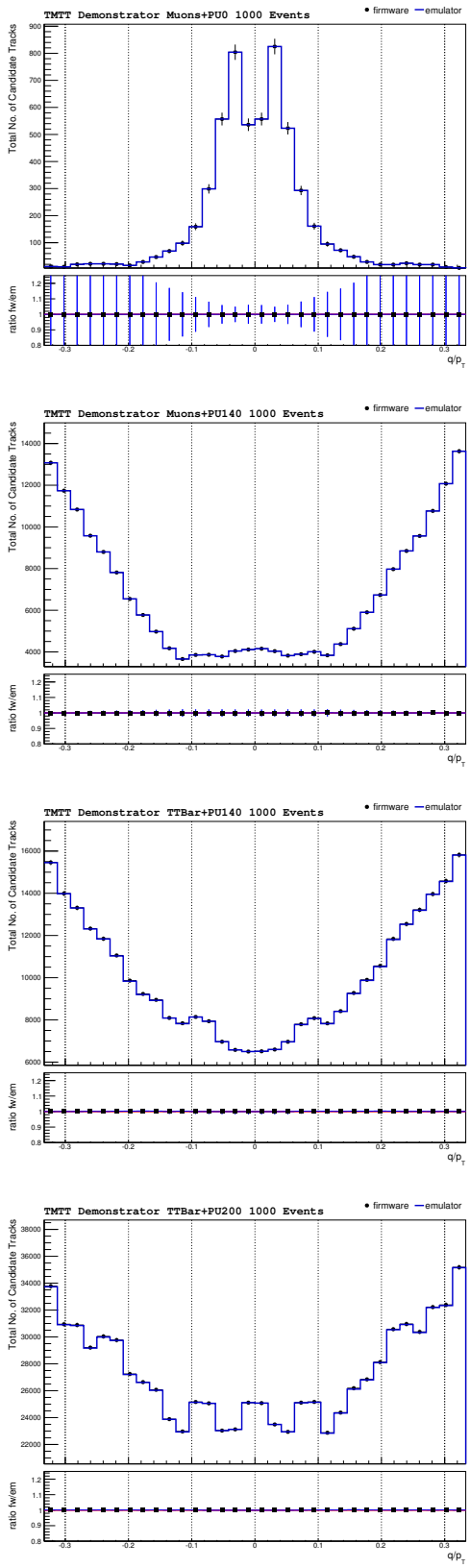


Fig. 7. Comparison between firmware running on FPGA (dots) and emulation (blue line) of the number of tracks found per unit of q/p_T for 1000 simulated events in the following samples: Single muon with 0 pile-up collisions (top), single muon with 140 pile-up collisions (upper center), $t\bar{t}$ events with 140 pile-up collisions (lower center), $t\bar{t}$ events with 200 pile-up collisions (bottom).

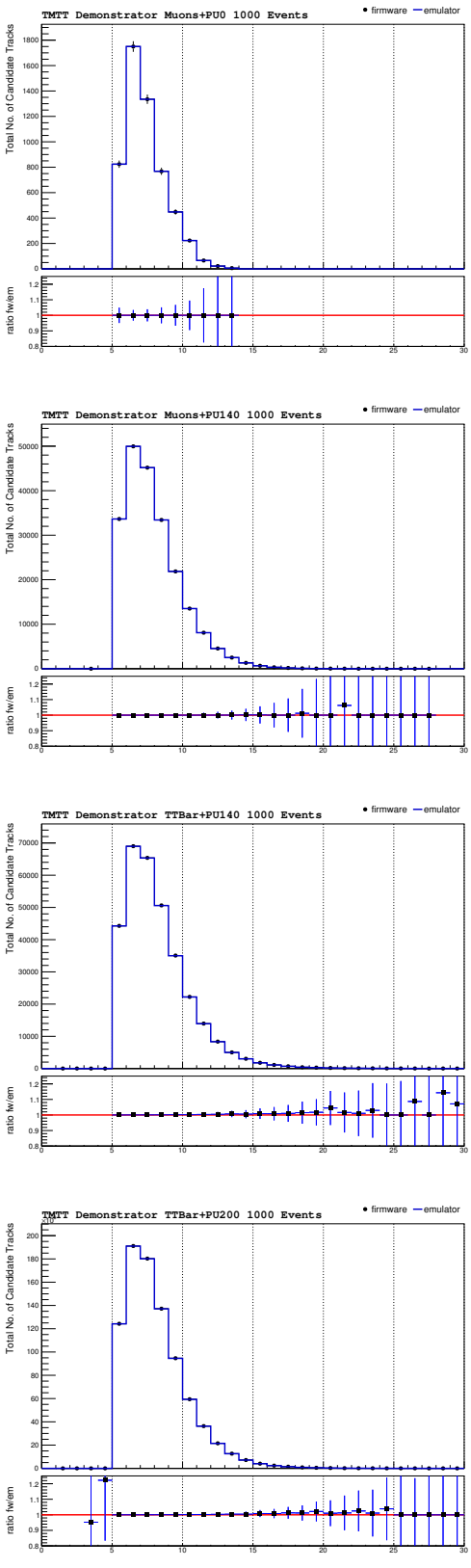


Fig. 8. Comparison between firmware running on FPGA (dots) and emulation (blue line) of the number of stubs per candidate track for 1000 simulated events in the following samples: Single muon with pile-up collisions (top), single muon with 140 pile-up collisions (upper center), $t\bar{t}$ events with 140 pile-up collisions (lower center), $t\bar{t}$ events with 200 pile-up collisions (bottom).