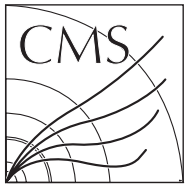
Available on CMS information server

CMS CR -2021/208



The Compact Muon Solenoid Experiment

Conference Report

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

17 October 2021 (v2, 22 October 2021)

Automated firmware generation and continuous testing for the CMS HGCAL trigger primitive generator

Florence Beaujean, Thierry Romanteau, Jean-Baptiste Sauvan for the CMS Collaboration

Abstract

The prototype version of the trigger primitive generator firmware for the Phase-2 CMS endcap calorimeter upgrade is being implemented in order to assess the FPGA resource requirements and dimension the system. For the development of some of these blocks, a data-driven design flow based on VHDL and HLS C/C++ templates is used to automate the production of multiple firmware variants. In addition, the design steps are integrated into Gitlab Continuous Integration tools to automatically test and validate every change, and as much as possible avoid repetitive manual tasks and the associated errors.

Presented at *TWEPP2021 TWEPP 2021 Topical Workshop on Electronics for Particle Physics*

Prepared for submission to JINST

TWEPP 2021 Topical Workshop on Electronics for Particle Physics
20-24 Sep. 2021

Automated firmware generation and continuous testing for the CMS HGCAL trigger primitive generator

F. Beaujean[a] **T. Romanteau**[a] **J.-B. Sauvan**[a,1] **on behalf of the CMS Collaboration**

[a] *Laboratoire Leprince-Ringuet, CNRS, École polytechnique, Institut Polytechnique de Paris*

E-mail: jean-baptiste.sauvan@cern.ch

Abstract: The prototype version of the trigger primitive generator firmware for the Phase-2 CMS endcap calorimeter upgrade is being implemented in order to assess the FPGA resource requirements and dimension the system. For the development of some of these blocks, a data-driven design flow based on VHDL and HLS C/C++ templates is used to automate the production of multiple firmware variants. In addition, the design steps are integrated into Gitlab Continuous Integration tools to automatically test and validate every change, and as much as possible avoid repetitive manual tasks and the associated errors.

Keywords: Calorimeter methods, Trigger concepts and systems

[1]Corresponding author

Contents

1 Introduction

The Level 1 (L1) trigger primitive generator (TPG) of the future Phase-2 High Granularity Calorimeter (HGCAL) upgrade of CMS [1, 2] is composed of two off-detector processing stages based on Serenity ATCA boards [3]. The first stage (Stage 1) mainly performs a synchronization, reorganization and truncation of the incoming data and time multiplexes its output data to the second stage (Stage 2). The latter then builds the actual trigger primitives and sends them to the central L1 trigger. One essential task of the Stage 1 firmware is to group trigger cell (TC) data coming from multiple detector modules into bins corresponding to projective regions of the detector. Each of the Stage 1 FPGA sees a different portion of the detector and therefore requires different configurations that depend on the full set of modules seen by the FPGA. In addition, since the geometry of the HGCAL is still evolving and the connection map between frontend detector modules and backend FPGAs is not yet fixed, the content of each FPGA will need to be updated several times in the future. In order to limit as much as possible manual tasks and reduce the probability of configuration errors, an automated design workflow has been developed featuring firmware generation using a template engine widely used in web development, Jinja [4], and continuous integration with Gitlab [5].

Details on the TC processing performed in the HGCAL TPG Stage 1 are presented in Section 2. The firmware generation workflow is then detailed in Section 3 and its automation with Gitlab continuous integration tools is finally presented in Section 4.

2 The Stage 1 of the HGCAL trigger primitive generator

One of the main role of the HGCAL TPG Stage 1 is to reorganize data coming from on-detector modules into a format that can be directly used by the reconstruction algorithms running in the Stage 2. In particular, TCs coming from detector modules need to be packed into groups covering projective detector regions called *bins* and sent bin by bin to the Stage 2 in a specific order. In addition, given the limited bandwidth available between the Stage 1 and the Stage 2, the number of TCs sent in each bin needs to be reduced. This TC pre-processing is schematized in Figure 1.

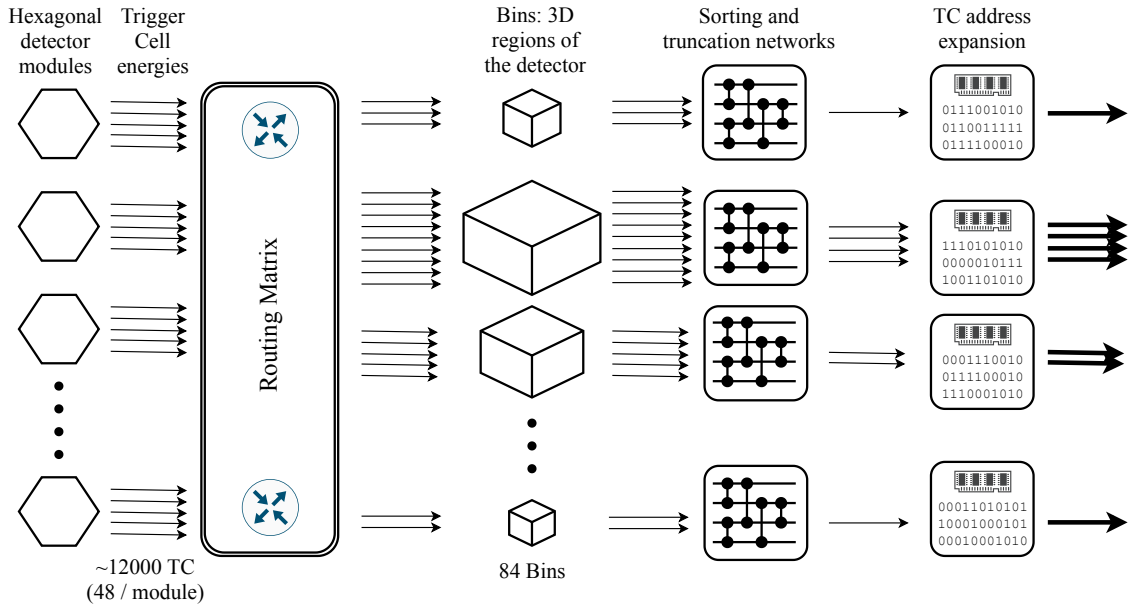


Figure 1. Sketch of the TC pre-processing performed in the Stage 1. It is composed of a routing of TC data into bins corresponding to projective detector regions, followed in each bin by a sorting and selection of TCs based on their energies, and an expansion of the selected TC addresses.

Each of the Stage 1 FPGA can receive TC data from up to about 250 detector modules. The 48 TCs in each module are fully unpacked in order to provide a parallel stream of up to about 12000 TCs every 25 ns. These TCs are re-ordered and routed into their corresponding bin, among a total of 84 bins. Bins can contain from 1 to about 400 TCs. This routing is fixed for a given FPGA and is extracted from the geometry of the detector, which defines the positions of all existing TCs, as well as from the link connections between detector modules and Stage 1 boards. A sorting network based on the Batcher odd-even mergesort algorithm [6] is associated to each bin. The network sorts TCs by their energy and also truncates progressively the least energetic TCs as they are sorted. Out of all the input TCs, 1 to 30 of the highest energetic TCs are kept in each bin by the networks. Local TC identifiers are propagated through the networks and are expanded to more global identifiers, unique within a FPGA. The selected TC energies and their associated identifier (or address) are finally packed and sent in a time-multiplexed fashion to the Stage 2 boards.

Sorting networks are written in C/C++, while the rest of the design is written in VHDL using VHDL-2008 specific syntax in some parts. Vivado HLS and Vivado, the high level synthesis and VHDL backend tools from Xilinx, are used to build the firmware from these sources.

3 Firmware generation workflow

One of the main challenges of the Stage 1 TC pre-processing described in Section 2 is the fact that each FPGA covers a different region of the detector. Each FPGA therefore sees a different number of detector modules, has a different routing matrix of TCs into bins and different numbers of TCs to be sorted in each bin. In order to limit the usage of FPGA logic resources and in particular of

LUT resources, the strategy has been followed to generate different firmware versions for each of the Stage 1 FPGA of the system, instead of having a single configurable firmware able to handle all the possible cases. In addition, the exact detector geometry and connection mapping between detector modules and Stage 1 FPGAs is still being optimized and will evolve in the future. This multiplicity of present and future firmware versions required the development of a highly flexible design workflow based on generic code configurable with data.

In order to reach the degree of generalization needed to describe all possible scenarios, VHDL and HLS C/C++ templates are used. The code templates contain fragments of generic VHDL and C/C++ code as well as rules describing how to instantiate and combine these fragments. These rules are written with the Jinja template language, which provides high-level instructions and functions using a syntax similar to Python. The set of rules and code fragments are developed by digital electronics engineers and describe the hardware architecture of the design.

The different steps and commands that are run to produce and test the firmware are described in a `yaml` file following the Gitlab CI/CD pipeline syntax. A schematic view of these steps is shown in Figure 2.

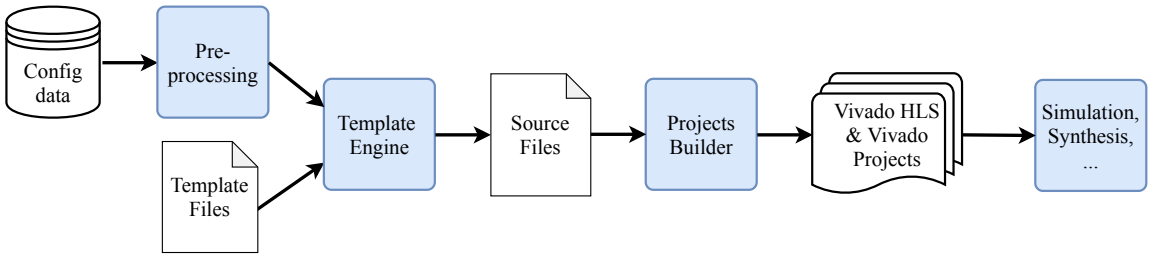


Figure 2. Diagram of the data-driven workflow used to generate and test firmware using templates and configuration data as inputs. The Jinja template engine is used to generate source code from templates. Vivado HLS and Vivado are used to build firmware from these generated source files.

The inputs of the workflow are template files (VHDL and HLS C/C++ templates based on the Jinja template language) as well as configuration data used to configure the templates. Configuration data come from various sources, in particular from the CMS geometry and simulation software. They are stored in several files using different formats, including binary formats (such as `ROOT`) and text formats (such as `JSON`). These raw configuration data are first pre-processed into Python dictionaries and stored in a single `Pickle` file. The Jinja template engine then generates source files and test bench files from the templates and pre-processed configuration data. The generated files are used to build Vivado HLS and Vivado projects and finally simulate, synthesize and test the design.

4 Automation with Gitlab Continuous Integration

The workflow described in Section 3 is automated with Gitlab Continuous Integration (CI) tools (Gitlab CI/CD). The two main items of this automation, depicted in Figure 3, are:

- **Two interlinked Git repositories**. The first (*Main*) repository stores and controls revisions of the input configuration data and templates, while the second (*Source*) repository stores and

controls revisions of the products of the workflow: the source code, test benches and project files.

- **A Gitlab CI pipeline**. It describes the steps of the workflow in a `yaml` format and runs them when specific events happen.

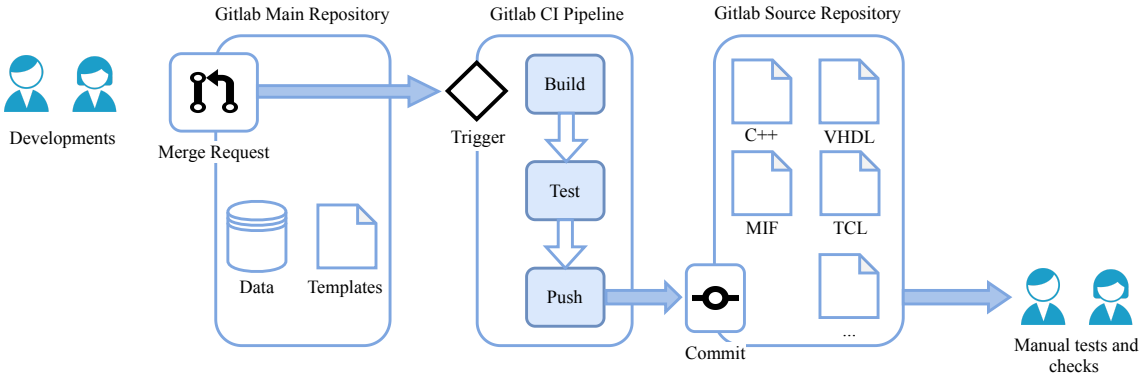


Figure 3. Main components of the workflow automation with Gitlab CI/CD. Two Git repositories store the inputs and the products of the workflow, respectively. A Gitlab CI pipeline defines and runs the different steps needed to create and test the products from the inputs.

Gitlab CI pipelines are attached to the Main repository and triggered in particular when a Merge Request is created or updated, and when a Merge Request is merged. Each time a pipeline is triggered it generates source files, builds projects and runs simulation and synthesis in order to test the generated design. When the updates from a Merge Request are merged, the generated files and projects are additionally pushed to the Source repository such that the design can be checked out and further tested manually if needed. Two sets of designs can be generated:

- Reduced designs, or *mini-designs*, based on a reduced number of input detector modules and bins, are generated for quick tests for every new or updated Merge Request.
- Full designs, based on the complete set of modules and bins, are generated only when a new release is created.

Since the details of the HGCAL geometry and of the TPG system architecture are not yet completely defined, several versions of the Stage 1 firmware will need to be evaluated in parallel. In order to handle these multiple versions, several parallel Git branches will be used. These multiple branches in the Main repository will all contain the same template files but different configuration data corresponding to the different architecture versions, and will serve to generate multiple designs in different branches of the Source repository.

A preliminary design targeting an implementation on a Xilinx KU15P FPGA with 72 input links has been generated and will be tested on the prototype Serenity boards currently available. It is nevertheless foreseen to use VU13P FPGAs in the HGCAL TPG system. Therefore, multiple versions targeting a VU13P FPGA with different numbers of input links will also be evaluated in the future.

5 Conclusion

The development of firmware for the HGCAL TPG Stage 1 has a strict dependency on rapidly evolving parameters such as the detector geometry and the mapping of connections between detector modules and the backend FPGAs. In addition, each FPGA in the Stage 1 requires different configurations as they cover different portions of the detector. In order to handle this variability and the future evolutions, an automated workflow based on generic VHDL and HLS C/C++ code templates has been implemented such that multiple firmware versions can automatically be generated with the provision of configuration data. The Jinja template engine is used to generate VHDL and HLS C/C++ source code and the process of generation and testing is automated within Gitlab Continuous Integration tools. A preliminary design targeting a Xilinx KU15P FPGA with 72 input links has been generated and multiple other designs targeting a VU13P FPGA with different numbers of links will also be evaluated in the future.

Acknowledgments

The authors of this paper would like to thank Andrea Sartirana for his precious help on the Gitlab server setup at LLR. This work has been partly funded by the French National Research Agency (ANR) via the project HiGranTS number ANR-18-CE31-0007, and by the P2IO LabEx (ANR-10-LABX-0038) in the framework "Investissements d'Avenir" (ANR-11-IDEX-0003-01) managed by the ANR.

References

[1] CMS Collaboration, *The Phase-2 Upgrade of the CMS Endcap Calorimeter*, Tech. Rep. CERN-LHCC-2017-023, CMS-TDR-019, CERN (2018).

[2] CMS Collaboration, *The Phase-2 Upgrade of the CMS Level-1 Trigger*, Tech. Rep. CERN-LHCC-2020-004, CMS-TDR-021, CERN (2020).

[3] A. Rose, D. Parker, G. Iles, O. Sahin, P.-A. Bausson, A. Tsirou et al., *Serenity: An ATCA prototyping platform for CMS Phase-2*, *PoS* **TWEPP2018** (2019) 115.

[4] Pallets Organization, "Jinja." https://github.com/pallets/jinja/.

[5] Gitlab, "Gitlab CI/CD." https://docs.gitlab.com/ee/ci/.

[6] D.E. Knuth, *The Art of Computer Programming, Volume 3: (2nd ed.) Sorting and Searching*, Addison Wesley (1998).