Contribution ID: **32** Type: **Oral**

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

Thursday, 23 September 2021 14:20 (16 minutes)

A first version of the firmware blocks of the trigger primitive generator for the CMS endcap calorimeter upgrade (HGCAL) are 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 is used to automate the production of multiple firmware variants based on VHDL and HLS C/C++ templates. In addition, the design steps are integrated into Continuous Integration tools to automatically test and validate every change, and as much as possible avoid repetitive human tasks and the associated errors.

Summary (500 words)

The Level 1 (L1) trigger primitive generator (TPG) of the future Phase-2 upgrade High Granularity Calorimeter (HGCAL) of CMS is composed of two off-detector processing stages. 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. 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. A sorting and potential truncation of these TC data are then applied in each individual bin. Stage 1 processing will run in six identical copies of 24 FPGAs each, with one copy for each 120 degree endcap sector. Each of these 24 FPGAs sees a different portion of the detector and the TC data routing and sorting is different in each of them. As an example, each FPGA contains 84 different sorting networks with numbers of inputs varying from 2 to more than 200 in different combinations for each FPGA. Having a single firmware design for all 24 FPGAs is not possible as it would require too many resources and excessive latency to handle all the different cases. It is therefore necessary to design the firmware individually for each of the 24 FPGAs. Doing this manually would be extremely difficult to develop and maintain. 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. Therefore an automated design workflow is developed, featuring automated firmware generation and continuous testing and integration. The key inputs to this workflow are code templates (VHDL and HLS C/C++) and configuration data, containing among other things information on the detector geometry and the frontend-backend connection map. A template engine, Jinja2, produces firmware source files from these inputs which can be automatically assembled into Vivado HLS and Vivado projects. The full workflow pipeline, composed of source code generation, high level synthesis, RTL synthesis, simulation, and place-and-route, is described as a Directed Acyclic Graph (DAG) and handled by a workflow manager. In addition the full process is integrated with Gitlab Continuous Integration tools such that every step can be tested and validated automatically for each update of the code templates and configuration data. The firmware blocks performing the TC data processing in the HGCAL TPG Stage 1, developed in C/C++ for some of them and in VHDL for others, will be presented as well as the methods and tools used to automatize the design process and to update firmware in a continuous fashion, as changes occur (for instance in the detector design).

Primary authors: SAUVAN, Jean-Baptiste (Centre National de la Recherche Scientifique (FR)); ROMANTEAU, Thierry (Centre National de la Recherche Scientifique (FR)); BEAUJEAN, Florence Danielle (Centre National de la Recherche Scientifique (FR))

Presenter: SAUVAN, Jean-Baptiste (Centre National de la Recherche Scientifique (FR))

Session Classification: Programmable Logic, Design Tools and Methods

Track Classification: Programmable Logic, Design Tools and Methods