

VIVADO High Level Synthesis in CLAS12 Trigger System Design

21st IEEE Real Time Conference

Sergey Boyarinov, Benjamin Raydo

Thomas Jefferson National Accelerator Facility, Newport News, VA
bboyarinov@jlab.org, braydo@jlab.org



Introduction

CLAS12 Trigger System makes decisions based on information from various detector types such as electromagnetic calorimeters, time-of-flight counters, Cherenkov counters and drift chamber trackers. It is organized as 3-stage FPGA-based processing with total latency under 7 μ s. In the first trigger stage detector pulses are digitized using 250MHz Flash ADCs and streamed to the VTP (VXS Trigger Processor) designed to reconstruct trigger objects: hits and clusters.

Vivado HLS was primarily used to implement the first stage trigger logic in the VTP, which contains a Virtex 7 FPGA. The Trigger System was successfully complete and used during the first physics run at Jefferson Lab in CLAS12. Our experience with HLS is described.

CLAS12 Trigger Logic Diagram

Stage 1 (shown in green and brown boxes below):

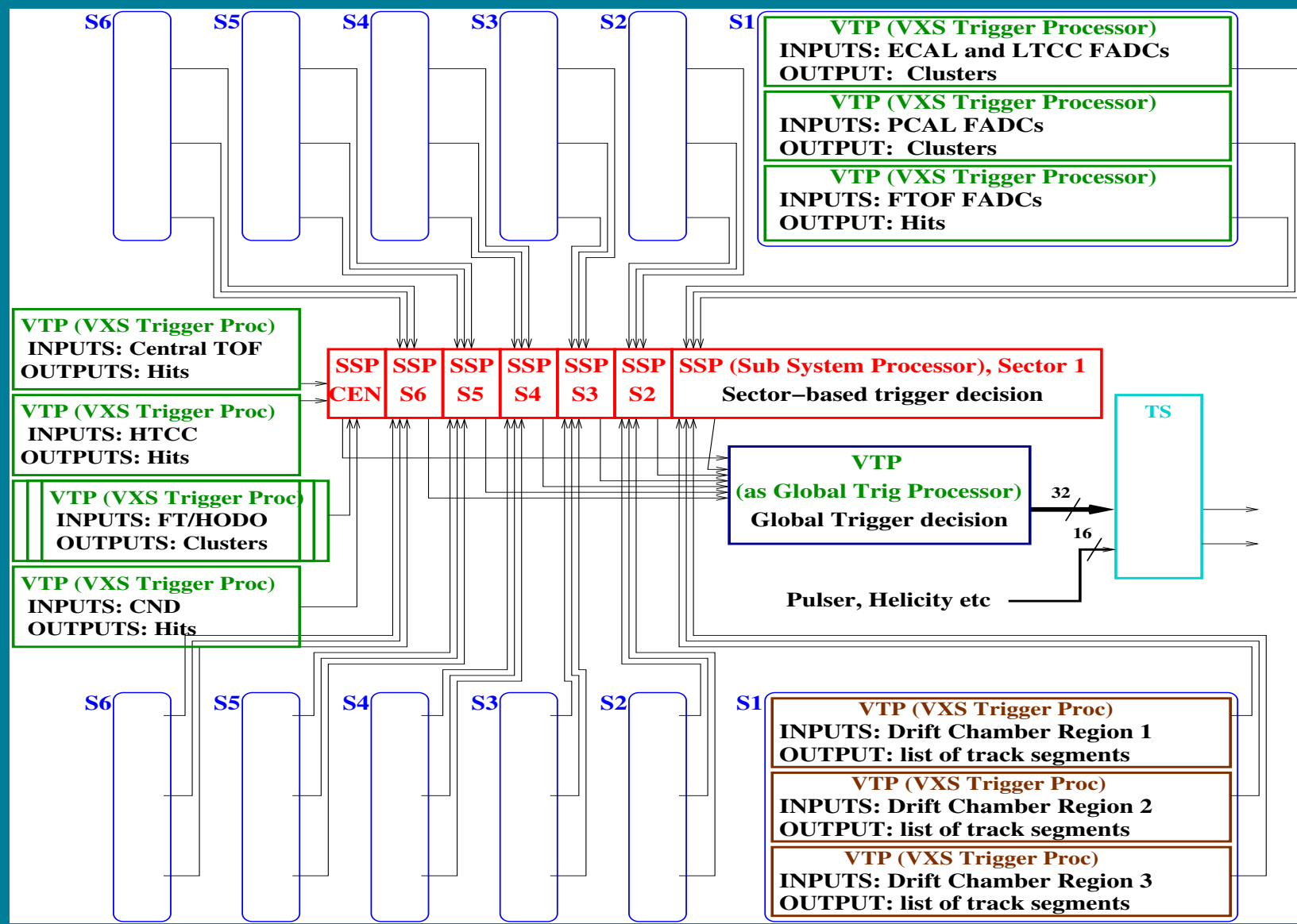
- VTP in each crate streams trigger objects to stage 2
- Sector based detectors: 6 VXS crates per sector (6 sectors)
- Central detector: 5 VXS crates

Stage 2 (shown in red boxes):

- Perform timing, geometry, energy cuts & coincidences between detectors within a sector
- VTP streams sector/central triggers to stage 3

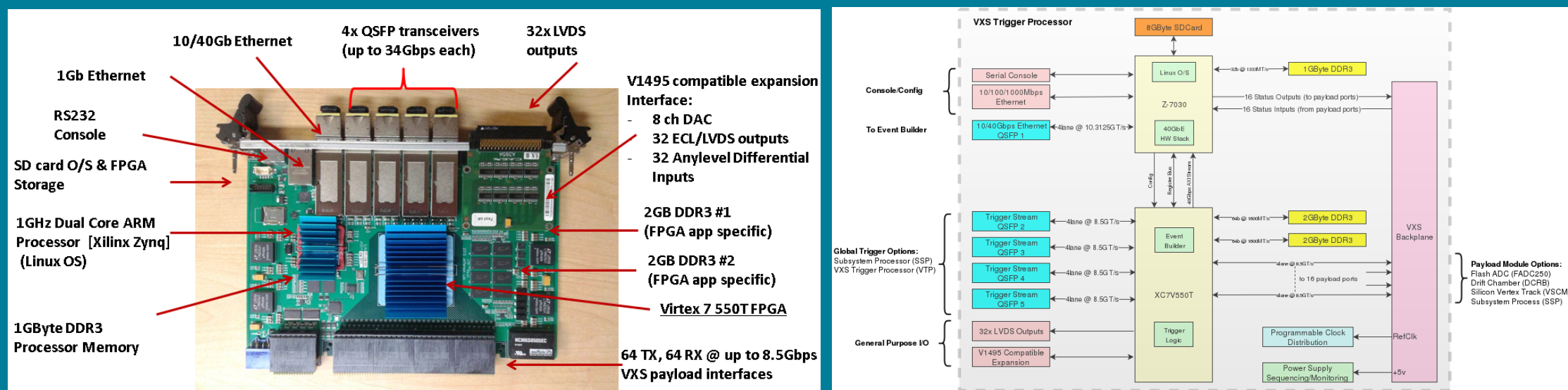
Stage 3 (shown in purple box):

- Defines physics triggers (up to 32) from stage 2 decisions



VTP (VXS Trigger Processor) Board

VTP board was specifically designed to be used on stage 1 of the Trigger System. The VTP PCB is a 20-layer impedance controlled FR-408HR stackup. Configuration & diagnostic event readout is performed by Xilinx Zynq processor running Arch Linux, while all trigger logic was implemented in Xilinx Virtex 7 chip. Bus interface, serial I/O, signal/clock distribution, and much of the event builder is implemented in VHDL running in Virtex 7, while trigger processing logic was written in C/C++ synthesized using HLS.



Our motivation to use HLS

- Make it easier to incorporate well-established data processing algorithms, typically written in C++ or other high-level language, into FPGA-based projects
- Involve programmers who developed algorithms for offline data processing but with limited FPGA programming experience
- Make it possible to validate code inside offline processing framework

Trigger components implemented with HLS

High Threshold Cherenkov Counter (cluster energy reconstruction)
Various Time-of-Flight Counters (hit pattern reporting)

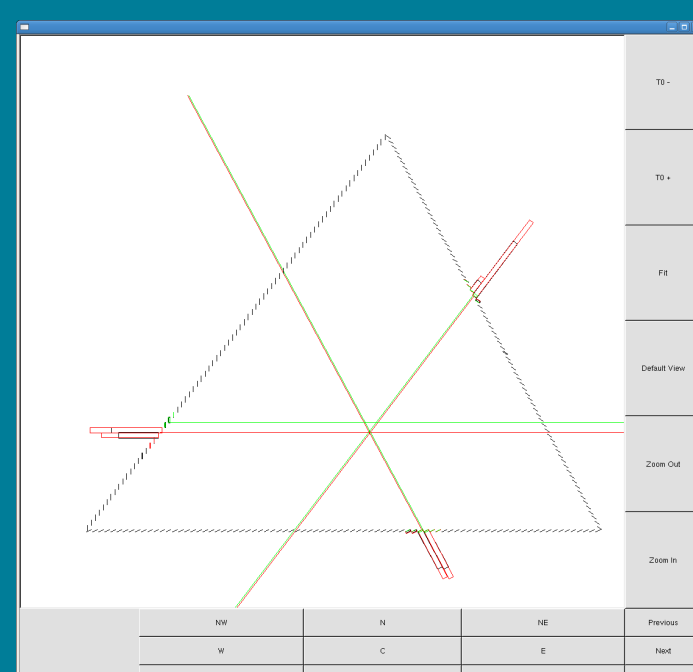
- Time-of-Flight and Cherenkov counters implementation was rather trivial, it would typically takes less than 10-15% of the Virtex 7 chip and timing requirements were easily met.

Electromagnetic and Preshower Calorimeters (cluster energy and position reconstruction)

- Calorimeter trigger implementations required much more effort because of its complex nature which requires significant FPGA resources. Details are further explained in the next sections using Electromagnetic Calorimeters as the example.

CLAS12 Electromagnetic Calorimeters

Among all trigger system elements, the most challenging for FPGA implementation was trigger component serving two CLAS12 electromagnetic calorimeters. Due to its structure, those calorimeters do not provide cluster coordinates or energies without significant event reconstruction. Both calorimeters consists of three sets of scintillation strip layers with PMT readout on one side of strip. To reconstruct clusters it is needed to find peaks in all three layers, find crosses of those peaks, correct peaks energies, correct clusters energies, and finally report clusters coordinates and energies to the following stage of the trigger system. Cluster reconstruction procedure was developed before as part of offline analysis and was well established, so we decided to use it as starting point for trigger component design.



Event with single cluster is shown in CLAS12 PCAL (preshower calorimeter); corrected energies are shown for individual strips

C++ and HLS C++

FPGA implementation of the electromagnetic calorimeter trigger was done in a 125MHz domain, a balance between speed and resource utilization. Trigger system components, in general, require a fixed latency, it set certain constraints on design. Reconstruction algorithm borrowed from offline analysis framework was adopted for Vivado HLS by rewriting it to C++, using HLS streams, HLS pragmas, unrolling for-loops, pipelining and making all other needed changes. Resulting implementation was tested on simulated data and shown correct results. After that we started to run it through Vivado HLS and Vivado tools and address various issues related to generating an FPGA image that met timing and fit within the resource allotment.

HLS and HDL

When HLS is used, compiling the design consists of the following main steps:

- VIVADO HLS – convert C/C++ to HDL
- VIVADO synthesis – HDL to FPGA primitives
- VIVADO implementation – map FPGA primitives to chip and route connections

For large designs VIVADO HLS will very often report extremely optimistic results suggesting a viable solution, but during VIVADO implementation will fail to meet timing. To address this the failing paths must be traced back to the HLS component where it can be changed to try to improve the design. It often took many iterations to either find the workable HLS settings, code structure, or clock period adjustments.

HLS clock domain

For different modules, we were using different clock domains between 250MHz and 31.25MHz. In 250MHz domain, modules typically more than 10% of a XC7V550 Xilinx FPGA failed to meet timing. In 125MHz and lower frequencies domains FPGA utilization was close to 100%. For Calorimeter project with chip utilization about 70%, 125MHz clock was used.

In general slower clock speeds (31.25MHz) for smaller projects where resources were plentiful. When using a slow clock the HLS code was able to be written as a single module and have no problem meeting timing during implementation.

Larger projects, like for the calorimeters, require more efficient use of the FPGA resources and have latency requirements that require a faster clock, but couldn't be too fast or HLS modules couldn't reliably meet timing – 125MHz was found to be the optimal middle ground for the -1 speed grade Virtex 7 for use in CLAS12.

HLS project size and organization

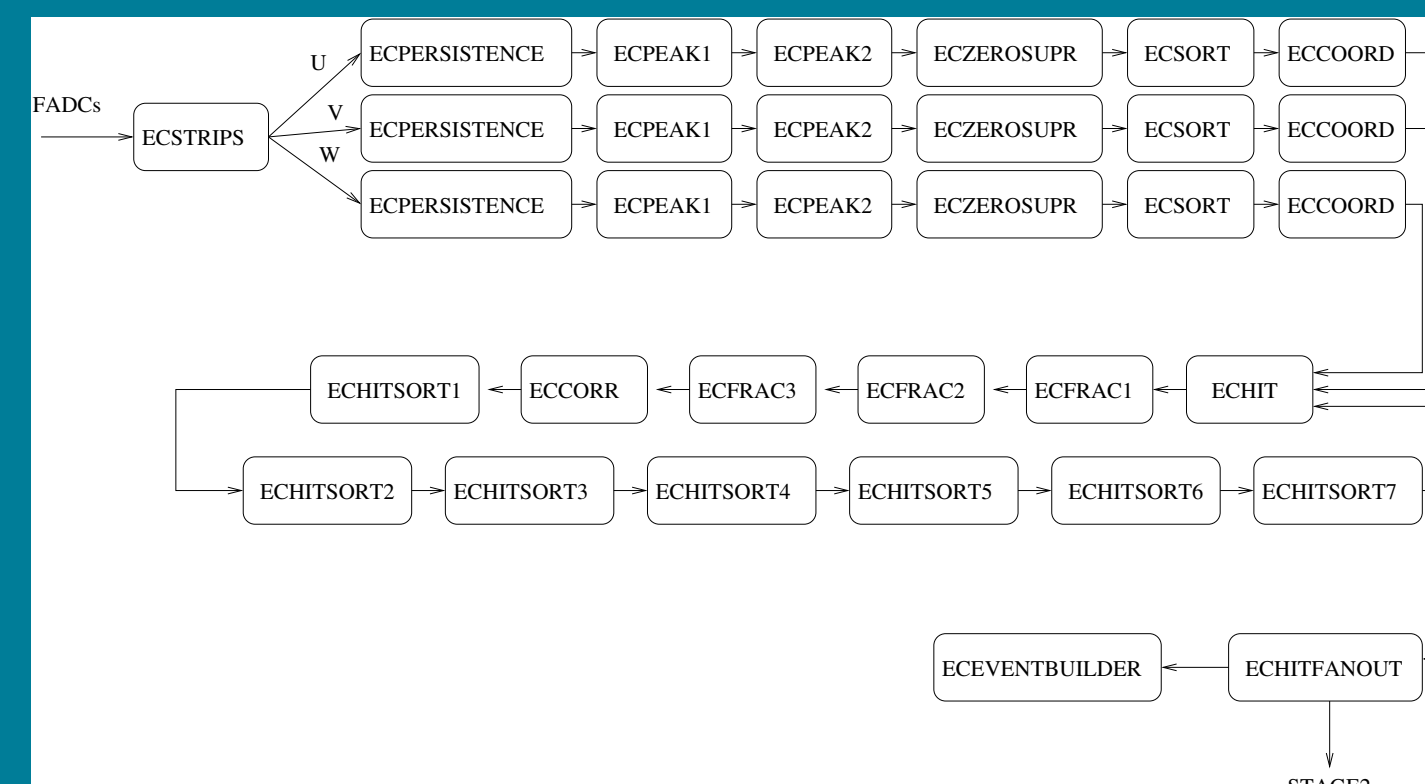
The typical HLS project for CLAS12 Trigger System contains few routines, and used HLS streams in the function parameter list to communicate easily with HDL surrounding. This scheme works well for small projects.

For Calorimeters with some versions being close to 100% of FPGA utilization situation was quite a different. The biggest problem we faced was inability to meet timing during implementation (even when HLS reports timing is good). HLS uses state machines to schedule the operations it synthesizes. For large HLS components the generated state machines can have massive control signal fanouts. As the clock period shrinks, so must the maximum signal fanout of general control signals for a design to reliably meet timing. For a clock period of 8ns using a -1 speed grade Virtex 7 each HLS module was kept smaller than ~30k LUTs (<10% of the LUT resources) to achieve a design that consistently meet timing.

The original calorimeter project consisted of about 20 C/C++ procedures that occupied most of the FPGA resources – with HLS generating big fanouts on this scale it was impossible to meet 8ns timing on implementation stage. The work around was to split entire project into smaller procedures, glued together in HDL by using well defined, simple interfaces between these separate procedures. Still, some procedures were too big, especially for the sorting algorithms. We were able to split some procedures feather and finally entire project met timing and resulting FPGA image was loaded into the hardware.

After every significant change we retested the code on simulated data, making sure it still produce correct results. Chart below shows how many HLS projects were created in the end.

Another reason for subdividing project is the lack of multi-clock domain support. Since event builder in VTP board works in 250MHz domain and most projects using slower clock every project was subdivided and separate pieces communicated over HDL-written interface. The necessity of subdivide HLS projects and use HDL to assemble the together is probably the most annoying feature in HLS usage.

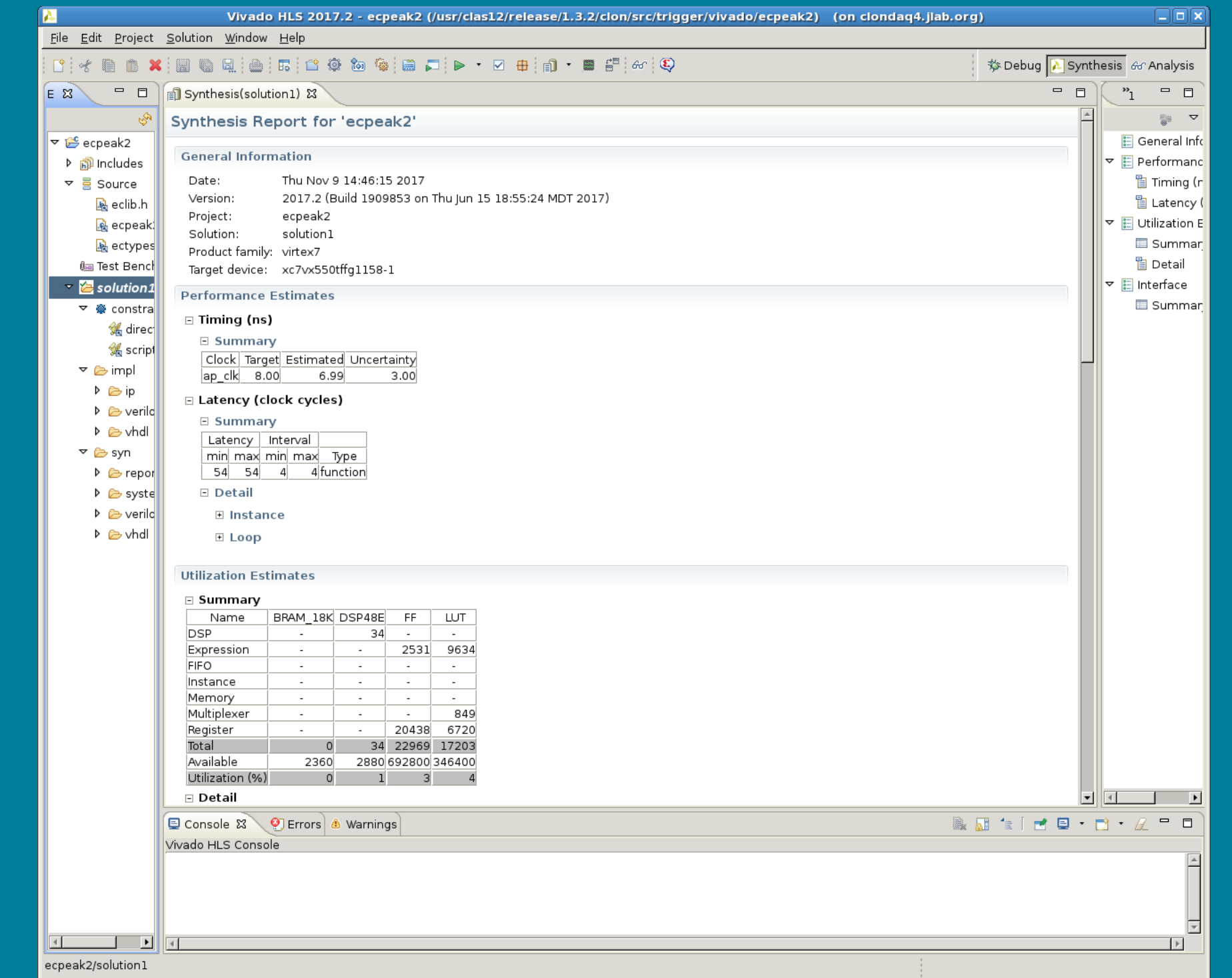


HLS versions and cross-project dependences

As it mentioned before splitting project on smaller pieces allowed us to meet timing. It worked in particular because we were able to eliminate combinatorial paths between HLS projects connected by streams. Such dependence can be clearly seen looking into schematic for failed timing chains, and usually related to the large state machine control signals going between modules. Initially we were using HLS version 2015, and despite of all our efforts we could not eliminate these long combinatorial paths across modules. This was resolved after switching to HLS version 2017 where streams could be fully registered (with pragma 'axis register both port='). This meant that if registered HLS streams were used between separate HLS projects then the state machines paths were also registered between modules. With that, it was only a matter of splitting projects on smaller pieces to improve/meet timing.

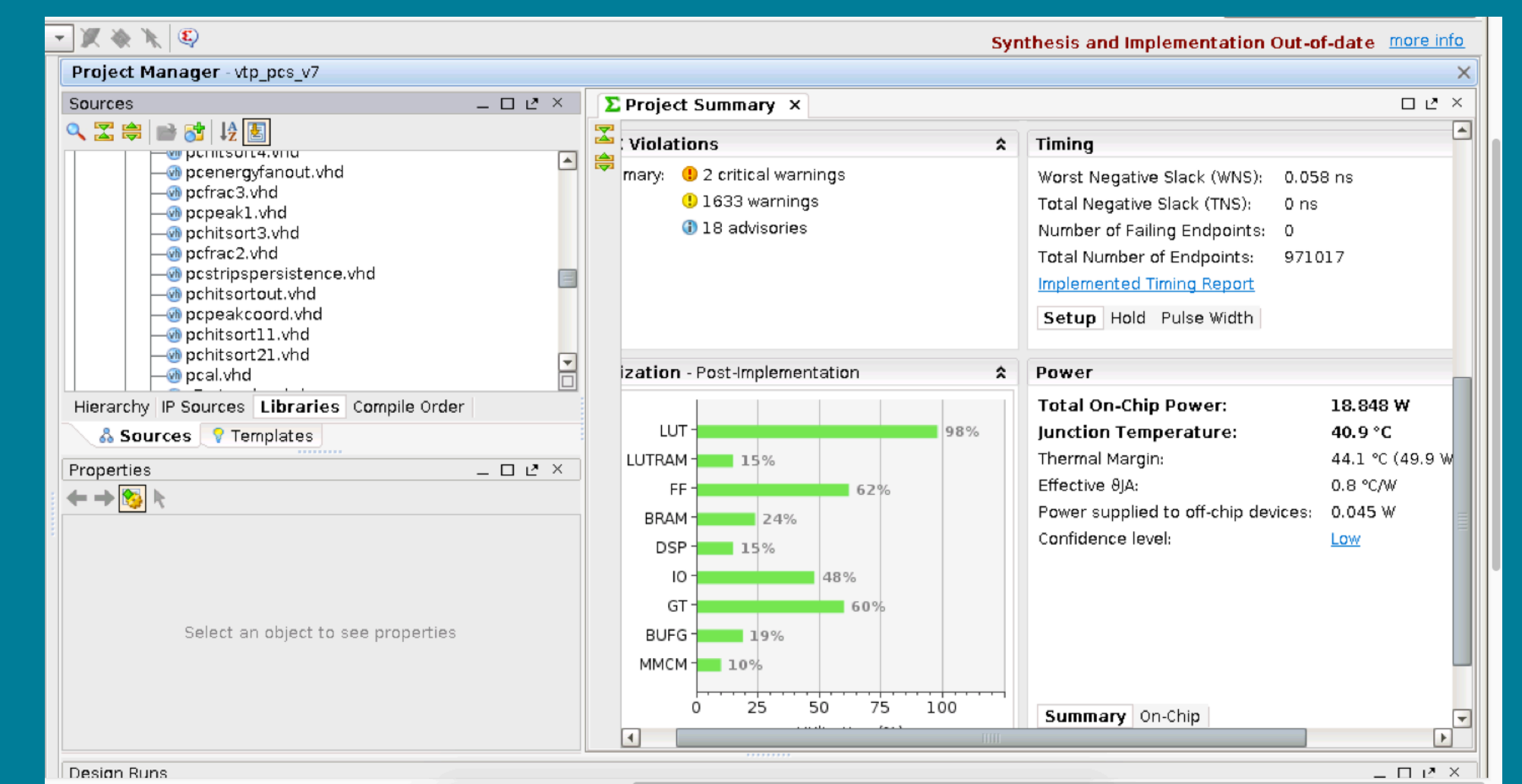
HLS Settings

Clock uncertainty is set as 30% of main clock, we found that it forces HLS to produce more realistic timing estimates. A single HLS project often cannot exceed several percent of FF and LUT budget, otherwise it may be a problem to meet timing requirement on VIVADO implementation step. Typical project from Calorimeter trigger is shown below.



VIVADO Settings

Common settings were used as shown on picture below. It usually takes 3+ hours to compile Calorimeter project on Dell R730 server under RHEL7. For some firmware versions, we were able to utilize 100% of LUTs and still meet timing – if clock domain was 125MHz or lower.



Firmware validation

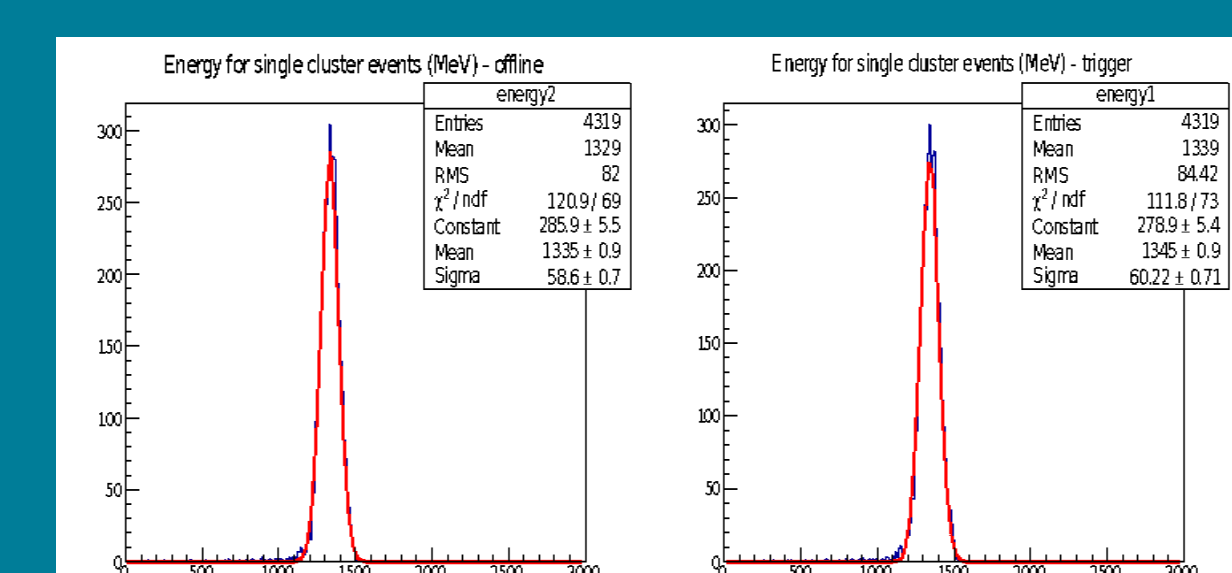
The ability to validate firmware using C++ implementation is the one of the biggest advantages of HLS. During the course of development and commissioning we ran HLS C++ code on simulated and real data from CLAS12 detectors, implementing required features and fixing bugs. During data taking we were able to find and fix observed problems or add new features in several hours, which was very important to save beam time.

Conclusion

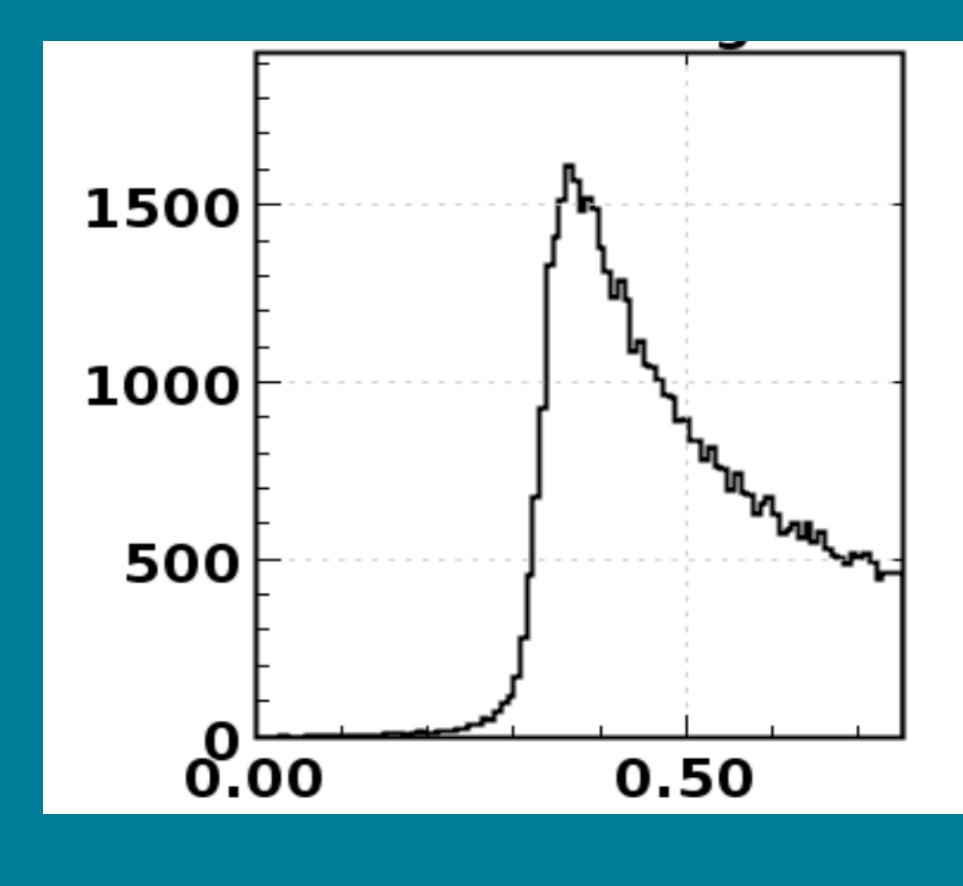
CLAS12 Calorimeters and other detectors were successfully implemented into trigger system using HLS to produce core part of the firmware. This trigger was used in the first physics run and worked as expected. We were able to select events based on individual cluster energy, something which was possible before only during offline data processing.

HLS in general appears to be a useful tool, especially to implement smaller trigger components like Cherenkov or time-of-flight counters. For components utilizing significant portion of the FPGA, it will be great to improve HLS in following directions:

- Support multi-clock domains
- Improve subroutine calls by allowing option to fully registered paths between modules.
- Improve state machine logic, for example support streams between routines inside the project and be able to generate separate state machines for separate routines. It will allow to avoid splitting project manually and use HDL as top interface as we are currently forced to do.



Electromagnetic calorimeter cluster energy reconstruction by offline analysis and by trigger system shows very good agreement



Offline-reconstructed calorimeter energy for the data taken with trigger threshold 300MeV demonstrates correct work of trigger system