Investigation of High-Level Synthesis tools' applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example

Investigation of High-Level Synthesis tools' applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example

Michal HUSEJKO and John EVANS
CERN, European Laboratory for Nuclear Research, Switzerland
E-mail: `michal.husejko@cern.ch`

Jose Carlos RASTEIRO DA SILVA
LIP, Laboratorio de Instrumentacao e Fisica Experimental de Particulas, Portugal

Abstract. High-Level Synthesis (HLS) for Field-Programmable Logic Array (FPGA) programming is becoming a practical alternative to well-established VHDL and Verilog languages. This paper describes a case study in the use of HLS tools to design FPGA-based data acquisition systems (DAQ). We will present the implementation of the CERN CMS detector ECAL Data Concentrator Card (DCC) functionality in HLS and lessons learned from using HLS design flow.

The DCC functionality and a definition of the initial system-level performance requirements (latency, bandwidth, and throughput) will be presented. We will describe how its packet processing control centric algorithm was implemented with VHDL and Verilog languages. We will then show how the HLS flow could speed up design-space exploration by providing loose coupling between functions interface design and functions algorithm implementation.

We conclude with results of real-life hardware tests performed with the HLS flow-generated design with a DCC Tester system.

1. Introduction

The DCC is a part of the Off-Detector Electronics sub-system of the CMS Electromagnetic Calorimeter (ECAL). It is responsible for collecting of data from the front-end system and the collection of trigger data from the ECAL trigger system. Data is transmitted from the DCC to the CMS DAQ system. The architecture of the CMS ECAL Off-detector trigger and readout architecture is presented on figure 1.

This study describes the implementation of DCC functionality in C and C++ languages by utilizing the Xilinx High Level Synthesis (HLS) compiler. Based on C/C++ input, the compiler is able to produce a VHDL/Verilog IP core which then can be instantiated inside a user design.

The main goal of this study is to perform feasibility investigation if the HLS tools are capable and mature enough to be applied to building data acquisition systems.

Due to some technical limitations, we are studding only FE On-detector data readout path (DAQ Data on the figure 1) and Selective Readout (SR Flags). Other DCC sub-systems are not included in this study (TCC, TTC, SpyMem, VME interface, etc.).

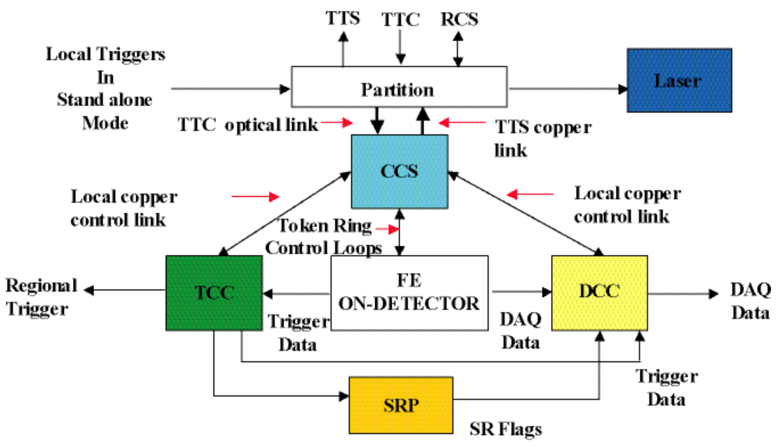


Figure 1. CMS ECAL Off-detector trigger and readout architecture.

2. The CERN CMS ECAL Data Concentrator Card (DCC)

Current production version of the DCC has been designed as a 9U VME board. Figure 2 presents the board's physical overview. The DAQ data processing path is distributed among 11 FPGA devices: nine Xilinx Virtex II Pro devices and two Altera Stratix II devices. The FPGA firmware is implemented in a mixture of SystemVerilog, VHDL and Altera Quartus schematics.

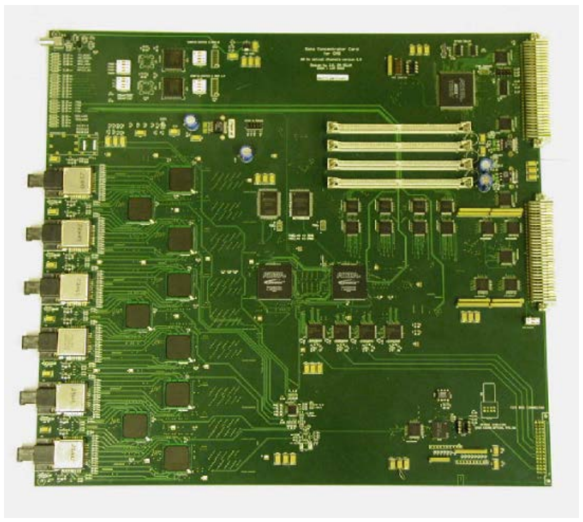


Figure 2. CERN CMS ECAL Data Concentrator Card (DCC).

There are three main building blocks on the DCC which jointly implement the data processing path: Input Handler (IH), Event Merger (EM) and Event Builder (EB). The Input Handler is replicated nine times (9x VIIPro FPGAs). The Event Merger has its own FPGA, as does the Event Builder.

The main functionality of the DCC is to receive FE data from 70 optical links. After data quality and synchronization checks are performed the data reduction algorithm is applied (6-tap Finite Impulse Response filter). More details about DCC data processing algorithm can be found in [2]. The 64 links are terminated in eight Input Handlers; the remaining 6 links are terminated in the 9th IH. The 9th IH also receives data from the Selective Readout (SR) Processor, which then are then distributed to other Input Handlers.

The Event Merger merges data from all Input Handlers, under the control of the Event Builder. When all the data is ready, the DCC sends data into DAQ system over the SLINK64 interface. The block diagram of this functionality is presented on figure 3.

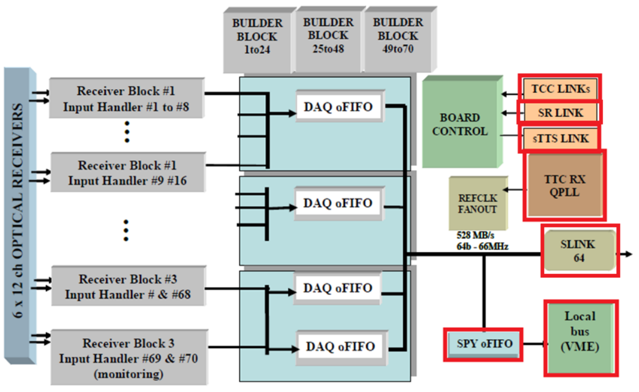

BUILDER BLOCK 1to24
BUILDER BLOCK 25to48
BUILDER BLOCK 49to70
6 x 12 ch OPTICAL RECEIVERS
Receiver Block #1 Input Handler #1 to #8
Receiver Block #1 Input Handler #9 #16
Receiver Block #3 Input Handler # & #68
Receiver Block 3 Input Handler #69 & #70 (monitoring)
DAQ oFIFO
BOARD CONTROL
TCC LINKs
SR LINK
sTTS LINK
TTC RX QPLL
REFCLK FANOUT
528 MB/s 64b - 66MHz
SLINK 64
SPY oFIFO
Local bus (VME)

Figure 3. DCC block diagram.

The FPGA firmware verification environment was built with SystemVerilog and Advanced Verification Methodology (AVM).

3. High Level Synthesis with Xilinx Vivado HLS

The Xilinx Vivado HLS application transforms a C/C++ design into a HDL implementation (either VHDL or Verilog). The HDL implementation can be then synthetized into Xilinx FPGA device. Together with the C/C++ design, the user should provide constraints and directives to achieve good quality of result. The C/C++ design shall be accompanied with a Testbench (in C/C++), which can also be automatically converted into a HDL testbench [3]. The block diagram of the HLS design flow process is presented on figure 4.

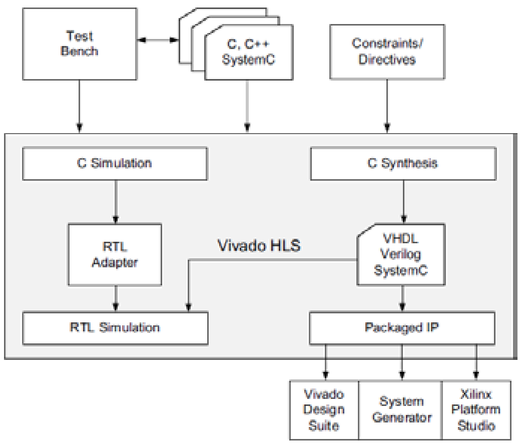


Figure 4. Vivado HLS Design Flow as represented in [3].

4. Initial set of requirements

For the purpose of evaluating the Vivado HLS for building data acquisition system we decided to implement in HLS all three main building blocks of the DCC i.e. IH, EM and EB. We have also included the SR receiver.

Due to software license we were constrained to use series-7 and later FPGA devices inside Vivado HLS. For that reason, we have selected Virtex-7 device and corresponding Xilinx development kit (Xilinx VC709) as a hardware test platform. Due to the limited number of high speed I/O available from that hardware we decided to scale down the number of IH links from 70 down to 8. The figure 5 presents the block diagram of the HLS DCC design. Figure 6 represents hardware realization.

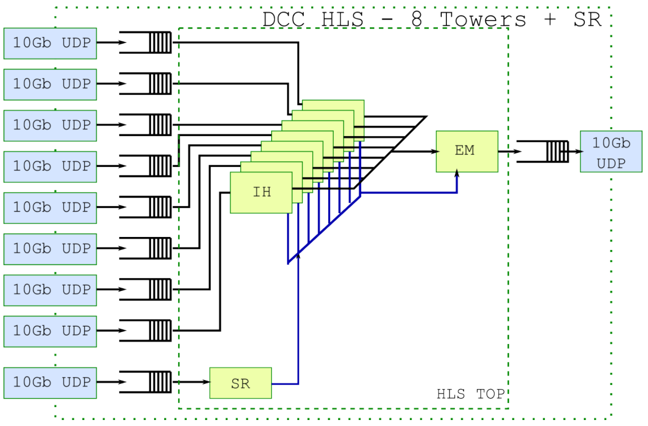


Figure 5. Block diagram of the HLS DCC test platform.

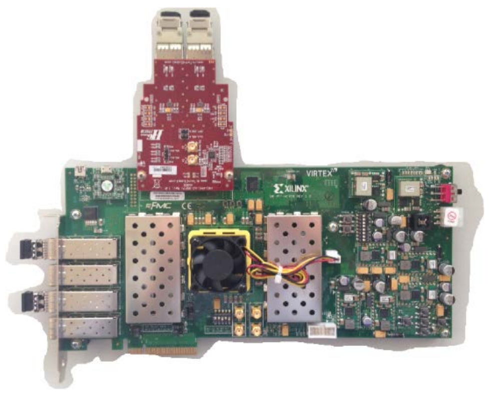


Figure 6. Hardware realization of HLS DCC.

The total number of links attached to the HLS DCC is 10. It allows emulating eight Front End (FE) data links (towers), one SR link, and one DAQ link. All the links are attached to a tester through 10Gb Ethernet UDP cores. In this evaluation, the DAQ link is configured as a standard 10 Gb UDP stream as opposed to Slink64 used in the production DCC system.

Each FE link attached to IH caries packets of 284 words. In each packet 250 words (250x16bits) are ADC samples (25 crystals x 10 ADC samples). If the packet can not be suppressed (due to ZS decision) is has to be completely forwarded to EM as a 75 long long packet (75 x 64 bits). The SR link caries packets of 22 16-bit words. Both the FE and SR data formats are defined in [4].

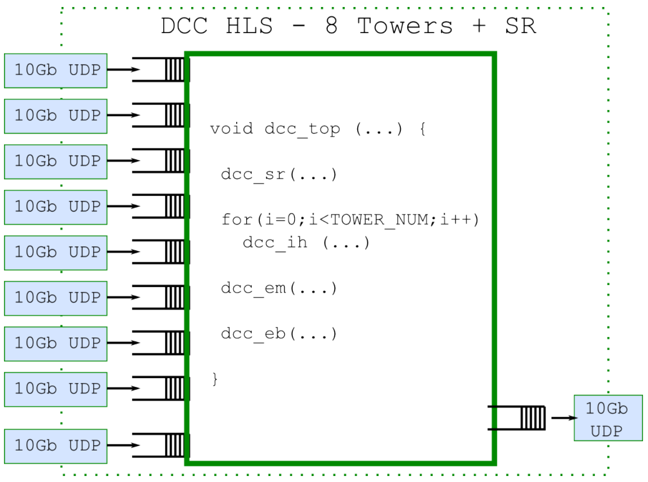
In ideal conditions where data for each IH is cross-aligned and there is no wait time between IH and SR packets, the latency of the HLS DCC system should not be more than 22+284+Nx75 cycles, which for eight channel test system (N=8) gives 906 cycles. Our requirement is to achieve latency no worst that 50% of the 906 clock cycles.

5. Implementation of DCC in C/C++

The HLS DCC design was split into four C functions: DCC_SR, DCC_IH, DCC_EM and DCC_EB, each respectively implementing (SR, IH, EM and EB functionalities).

The HLS DCC top level C/C++ function (DCC_TOP) is presented on the figure 7. It contains sequential calls to C sub-functions. Each sub-function implements one of the four functional elements of DCC (IH, SR, EM, and EB). The interface between HSL top function (DCC_TOP) and 10 Gb UDP cores is AXI-S and is automatically synthetizes from DCC_TOP parameter list. At C level the interface list is defined as a one dimension and multi-dimensional arrays.

The first call is to DCC_SR, which receives SR flags and distributes them to DCC_IH. The DCC_IH are called sequentially in the loop, and by default the calling loop is rolled. The results provided by DCC_IH functions are merged by DCC_EM. Finally the DCC_EB performs the final event building and sends data to the 10 Gb UDP link.



```
void dcc_top (...) {

 dcc_sr(...)

 for(i=0;i<TOWER_NUM;i++)
   dcc_ih (...)

 dcc_em(...)

 dcc_eb(...)

}
```

Figure 7. HSL DCC high level C function. TOWER_NUM=8

Each function was implemented using standard C data structures and all memory was modelled with multi-dimensional arrays. We have avoided multi-access pointers as they can cause unexpected results.

5.1. Default HLS compiler constraints - serial processing
We started our evaluation by utilizing default constraints and directives of Vivado HLS. By default the synthetized design is serial. The C functions are synthetized into HDL hierarchical blocks. We do not specify initiation interval (II) so the Vivado HSL tries to minimize latency and then area. Loops are left rolled. All the tasks are executed in sequence. This creates designs with very high latency, but the final size of the design is small as many building blocks can be re-used to implement similar functionality. Figure 8 presents the schedule of functions' calls.

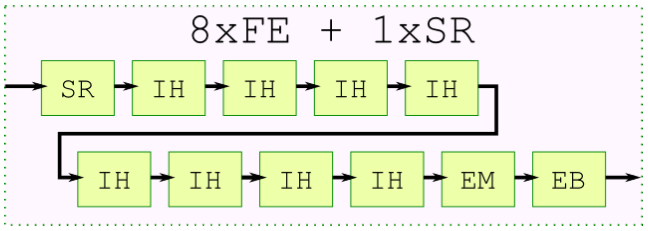


Figure 8. Default HLS constraints - serial implementation.

5.2. Tasks Parallelisation
The next step is to parallelize the tasks taht are independent. This is the case for all DCC Input Handlers which can execute in parallel.

To achieve parallelization of tasks we unroll the FOR loop which executes the calls to DCC_IH. This creates multiple independent operations rather than a single collection of operations. We use the HLS compiler UNROLL directive.

5.3. Pipeline functions
The next step was to pipeline the functions. The design contains many loops. By applying function-pipelining directives we allow for concurrent execution of operations, flattening nested

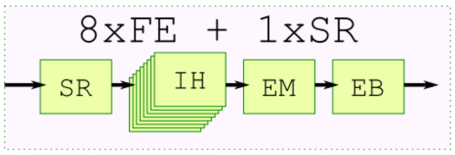


Figure 9. Parallel execution of DCC_IH tasks.

loops, and rewinding consecutive appearances to fill the execution gaps if the loop is running continuously.

5.4. Partition arrays into parallel memory blocks

FPGA has thousands of dual port BRAM memories. We can utilize them to improve throughput (more RAM ports, which can enable vectorised operations. This also decreases latency of the design as the data can be read and processed faster. For that purpose we use ARRAY_PARTITION directive to partition a single C array into multiple parallel FPGA block RAMs (times N), virtually creating N port BRAM.

5.5. Pipeline tasks

The last optimization is to enable DCC_TOP task pipelining, by utilizing DATAFLOW directive. This decouples tasks' execution in the top level function and allows partial overlap of computations. This further reduces the latency of the design.

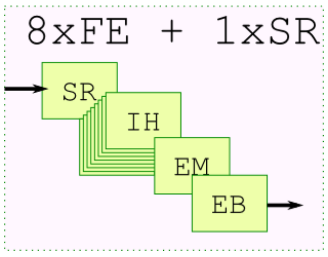


Figure 10. Enabling partially overlapping computations.

5.6. Summary

All the above design space exploration has been done without changing a single line of code of our DCC C/C++ implementation. Utilizing Vivado HLS pragmas, allowed us to control both the scheduling and binding of our high level code and optimise performance.

Design space exploration	Latency	Init Inter	BRAM	DSP48E	FF	LUT
Step1	26590	26590	12	4	988	1951
Step2	7960	7960	20	7	3555	6089
Step3	5192	5192	20	350	22203	27385
Step4	1734	1734	52	351	25966	25091
Step5	1443	603	104	401	29999	28366
FPGA Device utilization (%)			3	11	3	6

Table 1. Summary of design space exploration performed on HLS DCC design. Latency is defined as 6 ns clock cycles.

The results presented in the table 1 show that the final HLS design which is represented by Step 5 in the above table, has a latency which is 59 % higher than the theoretical latency of 906 cycles. This is very close to our goal of 50 %. Step 5 also achieved more than 600 clock cycles of initiation interval which means that there are 600 clock cycles of dead time.

6. Hardware tests

The hardware tests were carried out to verify that the code can be synthetized and a correct bitstream generated.

We have built a HLS DCC Tester system based on a standard PC workstation for testing. The workstation was fitted with 10Gb low latency cards. Five dual SFP+ rail cards were installed to provide 10 links, each operating at 10Gb speed. The HSL DCC Tester system is presented in figure 11.

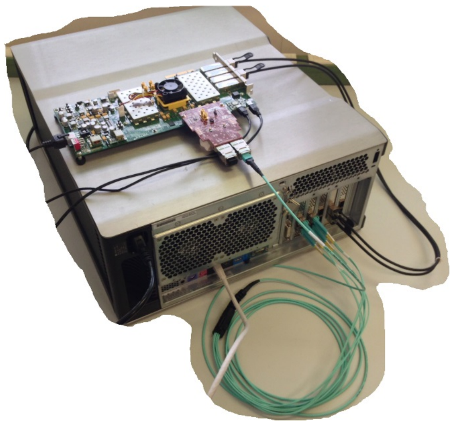


Figure 11. HLS DCC Tester.

A set of test applications has been written in C to communicate over sockets with the HLS DCC. These applications were used to stimulate design with eight parallel FE-like streams and SR flags and check the results arriving over the DAQ link.

Only functional tests were carried out without performing performance tests. This was due to high initiation interval represented by the final design (600 cycles for Step 5 in table 1) which would create meaningless results.

7. Lessons learned

Our evaluation has show that notably good results can be achieved even with C code which is not optimized for FPGA architecture. This indicates that much better results can be obtained if the code is more optimized towards FPGA devices e.g. using streaming data and algorithms to operate on it.

Based on our experience gained during this investigation we can claim that an engineer who would use HLS tool could possibly achieve higher productivity than a non-HLS enabled engineer. A junior engineer can gain high productivity gain by using HLS when supervised by experienced engineer.

8. Future plans

We plan to investigate in the near future different C coding styles which might better be suited for HLS implementation. We also plan to migrate the HLS DCC design to streamed architecture which would allow us to achieve single clock initiation interval - the missing design feature which could prove that HLS could possibly replace HDLs for DAQ system building. This would also allow us to perform performance analysis of a new design.

For the next step we plan to extend our study to Altera OpenCL SDK and Xilinx SDAccel compilers.

9. Conclusion

We have shown that Vivado HLS application can be used in building DAQ systems. It is capable enough to take a DSP behavioural C code (FIR filtration) wrapped with conditional packet processing control logic and translate it into working HDL. The generated HDL can be synthetized and implemented on FPGA in the same way as an ordinary HDL IP core. Even though C is sequential language in its nature, the HLS compiler directives provide a way to perform design tuning.

References

[1] Data concentrator card and test system for the CMS ECAL readout, Almeida N. 9th Workshop on Electronics for LHC Experiments, Amsterdam, The Netherlands, 29 Sep - 3 Oct 2003, pp.115-118
[2] Data filtering in the readout of the CMS Electromagnetic Calorimeter, Almeida N. et al. JINST 2008
[3] High Level Synthesis, Vivado Design Suite User Guide, Document version 2015.1
[4] ECAL Data Concentrator Card Specifications, LIP internal document