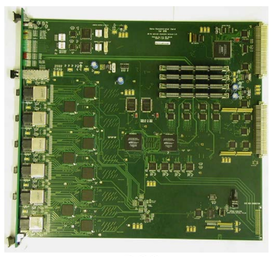
Hardware evaluation of Xilinx High Level Synthesis for building data readout systems – a CMS ECAL Data Concentrator Card case

Michal HUSEJKO (CERN, Switzerland), **John EVANS** (CERN, Switzerland),
Jose Carlos RASTEIRO da SILVA (LIP LISBON*, Portugal)

LIP Laboratorio de Instrumentacao e Fisica Experimental de Particulas, Portugal

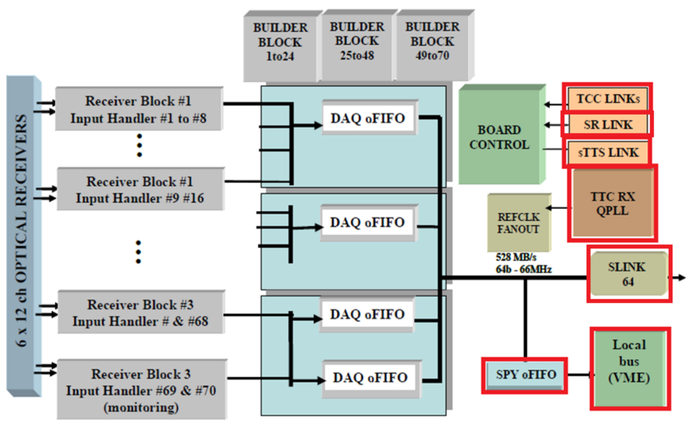
The CMS ECAL Data Concentrator Card (DCC) – a brief description of the current system

- 9U VME board with 12x FPGA devices: 9x VirtexII Pro, 2x Stratix, 1x Acex
- Three main building blocks implemented in HDL: Input Handler (**IH**), Event Merger (**EM**), Event Builder (**EB**)
- **IH**: Receives data from the on-detector Front-End electronics, applies Zero Suppression (**ZS**) algorithm (6-tap FIR), packages the data to be send to Data Acquisition System (**DAQ**). The **ZS** decision is based on Selective Readout (**SR**) packets received from a Selective Readout Processor (**SRP**) board
- **EM**: Concentrates data from 68 IHs
- **EB**: Pulls out data from IH and transfers it into **EM**, then appends other packets and event synchronization data, and sends Event to DAQ over SLink64 interface

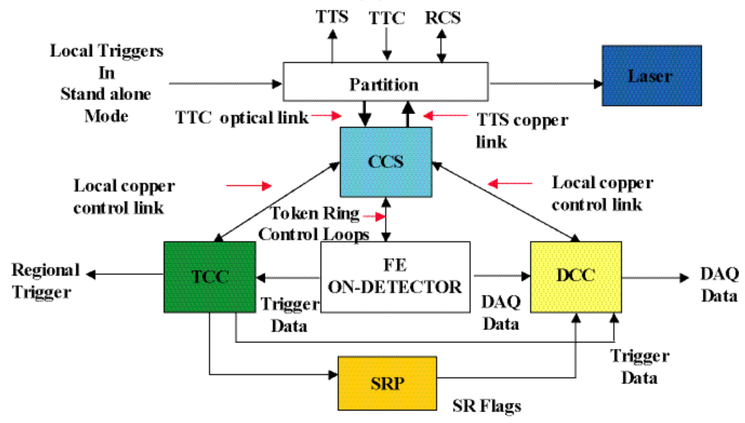


DCC

DCC Tester



CMS ECAL Off-detector trigger and Readout Architecture



DCC firmware – Current system

- Firmware in: 9x VirtexII Pro, 2x Stratix, and 1x Acex FPGAs
- Production design described in mixture of SystemVerilog, VHDL and Quartus Schematics
- DCC design SV/VHDL ~ 17'500 lines of code
- DCC testbench in SV ~ 3000 lines of code

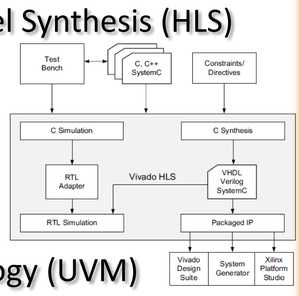
Broad project goal: Studying FPGA design methodologies for next generation FPGA based data acquisition systems:

FPGA firmware designer productivity improvement:

- Rising level of abstraction available to HDL designers - application of High Level Synthesis (HLS) compilers for C/C++ to HDL compilation
- Programming FPGAs by non-HDL designers (FPGA OpenCL compilers)

Verification methodologies:

- C-level simulation (fastest simulation time)
- C-generated RTL Co-simulation (RTL simulation environment)
- System Verilog (SV) verification language and Universal Verification Methodology (UVM)



Software quality management technologies applied to FPGA firmware verification

- Automated build systems
- Continuous Integration systems

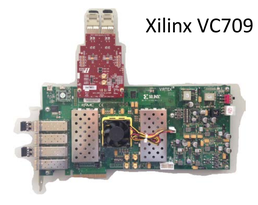
This study concentrates on studying application of HLS to DAQ systems' building

DCC firmware – HLS implementations

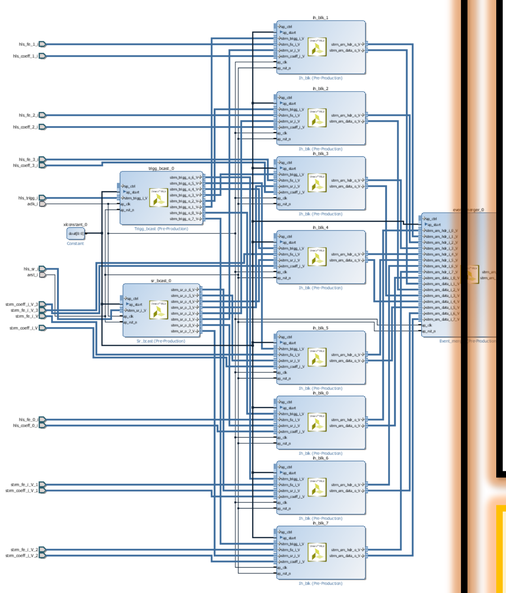
- Targeted for Zynq and Virtex-7 FPGA devices
- Written in C and C++ languages, and compiled to Verilog, then instantiated inside FPGA as a single component and connected to PCIe interfaces through AXIS and memory (BRAM like) interfaces.
- Do not include some other functionality of a production DCC (TCC, TTS. VME)
- DCCv1 HLS design:
  - Contains around ~ 1000 lines of code + 30 pragmas
  - Code was not modified after initial coding, only additional compiler pragmas were added (inside external pragma file) for design space exploration
- DCCv2 HLS design – complete code rewrite of DCCv1
  - Uses data streaming interfaces instead of arrays (DCCv1)
  - Contains around ~1000 lines of code and 20 pragmas
  - Coding style was tailored towards processing of data streams

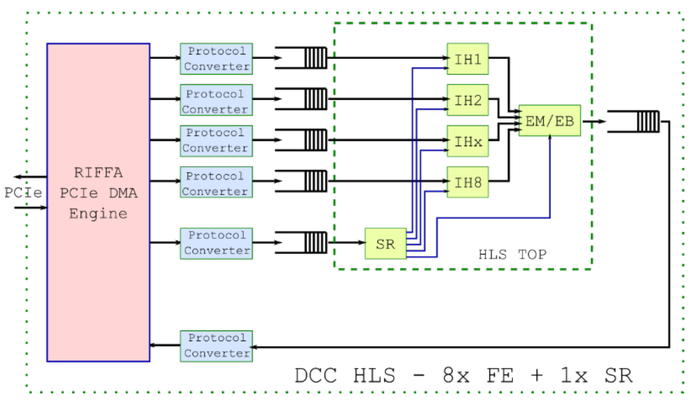
This study: DCC functionality implementation in C++ with High Level Synthesis tool and its hardware testing on PCIe FPGA kit

- DCC IH, SR, EM functionalities described in C and C++, compiled into single DCC TOP IP.
  - implements 8x FE + 1x SR + 1x "pseudo" DAQ channel
- All external links (FE, SR, DAQ) abstracted on top of PCIe. Transfers performed with DMA.
  - RIFFA PCIe DMA engine [2] used to provide PCIe interface inside FPGA
  - Protocol converter between RIFFA and HLS kernels.
  - Integrated on the Xilinx VC709 and ZC706 development kits
- Used Xilinx Vivado HLS 2015.2 [4] – C/C++ to HDL compiler
- Two versions of HLS design created and investigated
  - DCCv1 HLS – arrays as interfaces
  - DCCv2 HLS – hls::stream based interfaces
- Design space exploration performed with DCCv1 design – results not "good enough"
- Decision to change coding style and utilize data streams instead of memory arrays (DCCv1)
- Obtained results with DCCv2: II=1, Latency <50 cycles Design meet timing at 160 MHz.
- DCCv2 HLS Resources usage: LL/FF=4k DSP=8 BRAM=0.
- RIFFA + protocol converters: LL/FF= 17k



Xilinx VC709





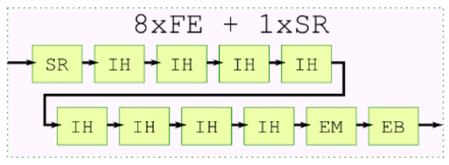
DCCv1 HLS – Design space exploration:

- Serial implementation (default constraints)
- Task level optimization (LOOP UNROLL, DATAFLOW)
- Loop level optimization (PIPELINE, FLATTEN)
- Vectorization (ALLOCATE, ARRAY_PARTITION)
- Automatic interface insertion/synthesis (AXIS, BRAM)

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

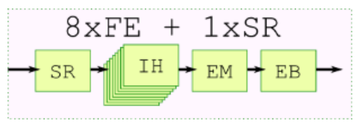
Step 1 – Default HLS compiler constraints

- Serial implementation
- C functions synthetized into HDL hierarchical blocks
- No initiation interval specified, minimize latency then minimize area.
- Loops are "rolled" – serial execution
- Arrays synthetized into BRAMs
- Serial execution of tasks – very high latency, but small device utilization



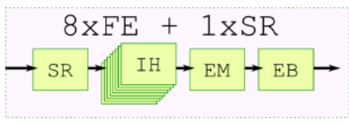
Step 2 – Parallelize tasks

- Execute tasks in parallel - Loop unrolling to create multiple independent operations, rather than single collection of operations



Step 3 – Pipeline functions

- Loop pipelining - concurrently execute the operations
- Loop flattening - flatten nested loops
- Loop rewinding - if the loop runs "continuously", rewind consecutive appearances to fill the gaps

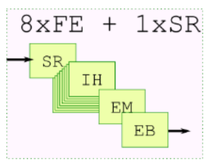


Step 4 –Partition arrays into parallel BRAMs

- FPGA has thousands of dual port BRAM memories – utilize them to improve throughput (more RAM ports, vectorized operations) and lower latency
- Step 3 + Apply array partitioning on internal arrays, splitting single array onto Nx BRAMs, virtually creating N port BRAM

Step 5 – Pipeline tasks

- Pipeline tasks' execution
- Partially overlapping computations



Conclusions

- **We have shown that Vivado HLS tool is capable enough to be used for building DAQ systems. The systems can include both DSP and packet processing algorithms and their mixture.**
- **Directives of HLS compiler are powerful way to perform design space exploration, yet the coding style is critical to obtain matching with HDL in terms of throughput, latency and resources' usage.**
- **Vivado HLS is actively developed and improved by Xilinx. It has multiple constraints on what C/C++ code can be processed with it. It is imperative that a potential user consults user guides [4] and training materials [5][6] and Xilinx Community Forums [3] before assuming that "any" C/C++ code can be compiled with it into WORKING Verilog/VHDL design.**

Future work

- Integrate production DCC testbench (SystemVerilog + AVM/UVM) with HLS RTL Co-simulation
- Investigate different C coding styles which works the best with Vivado HLS application.
- Design CERN technical training module for HLS. Tailor it specifically for DAQ applications.
- Document and share DCCv1/v2 source code with community.

References

[1] Jose Carlos Da Silva, Michal Husejko and Joao Varela "*Design of a Data Concentrator Card for the Readout of the Compact Muon Solenoid Electromagnetic Calorimeter*". 4th IEEE International Symposium on Electronic Design, Test & Applications (2008)
[2] Jacobsen, M., Richmond, D., Hogains, M., and Kastner, R. *"RIFFA 2.1: A reusable integration framework for FPGA accelerators." ACM Transactions on Reconfigurable Technology and Systems (TRETS), September 2015.* http://riffa.ucsd.edu/
[3] Xilinx Community Forums. https://forums.xilinx.com/
[4] Vivado Design Suite User Guide. *High-Level Synthesis*. Xilinx 2015. UG902. Version 2015.2 April 1, 2015
[5] Xilinx University Program. *High-Level Synthesis Flow for Zynq* . http://www.xilinx.com/support/university/workshops.html
[6] Vivado Design Suite Puzzlebook. *HLS*. Xilinx 2015. UG1170. Version 2015.2 July 17, 2015