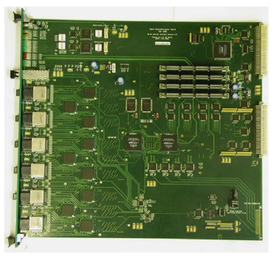
Investigation of High-Level Synthesis tools' applicability to data acquisition systems design based on the CMS ECAL Data Concentrator Card example

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

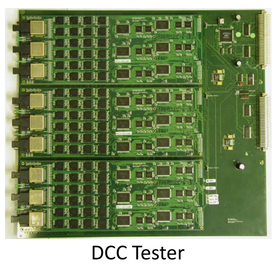
LIP Laboratorio de Instrumentacao e Fisica Experimental de Particulas, Portugal

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

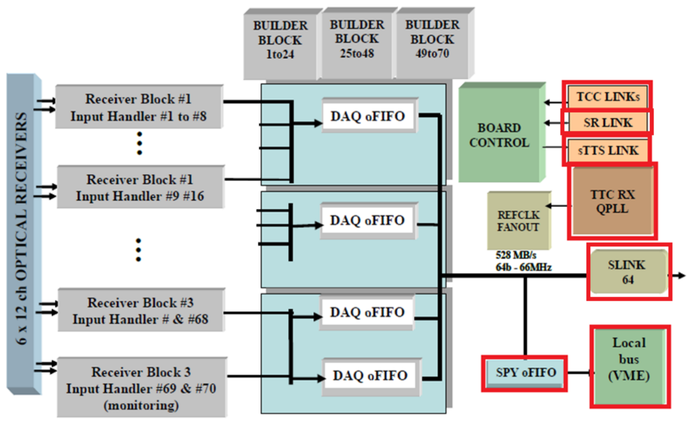
- 9U VME board with 12x FPGA devices: 9x VirtexII Pro, 2x Stratix, 1x Acex
- Three main building blocks implemented in HDL: Input Handler (**IH**), Event Merger (**EM**), Event Builder (**EB**)
- **IH**: Receives data from the on-detector Front-End electronics, applies Zero Suppression (**ZS**) algorithm (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



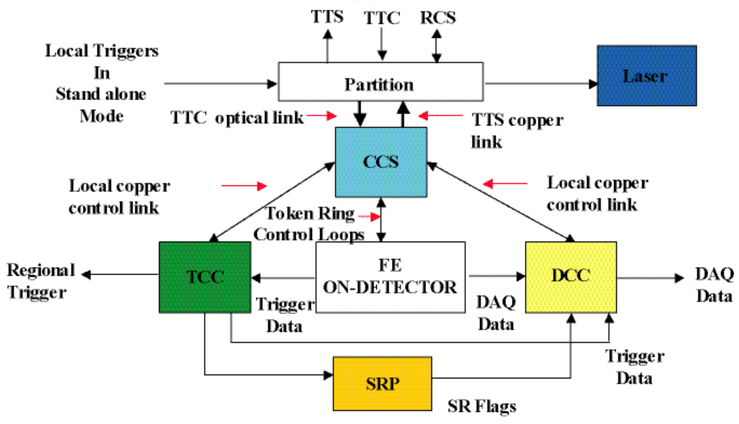
DCC



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

DCC firmware – HLS implementation

- Single Virtex-7 FPGA device
- Written in C language, and compiled to Verilog, then instantiated inside FPGA as a single component and connected to interfaces' IP cores by either AXIS or memory (BRAM like) bus
- Design contains around ~ 860 lines of code + 30 pragmas
- Does not include some other functionality of a production DCC
- DCC HLS code was not modified after initial coding, only additional compiler pragmas were added (inside external pragma file) for design space exploration

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)

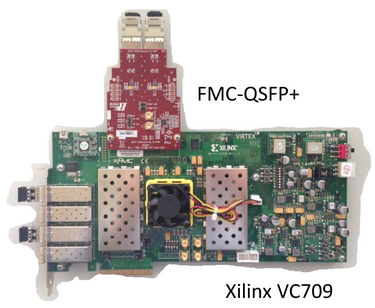
Software quality management technologies applied to FPGA firmware verification

- Automated build systems
- Continuous Integration systems

<u>This study concentrates on studying application of HLS to DAQ systems' building</u>

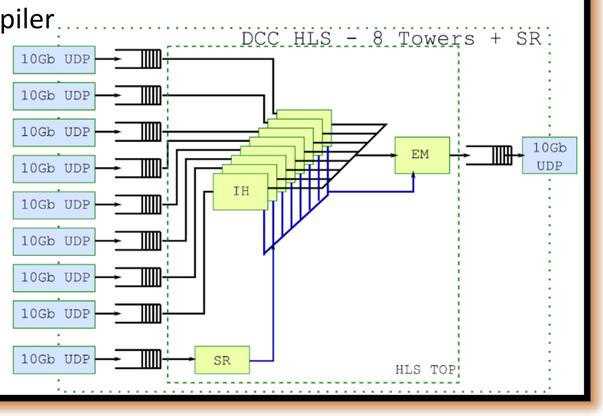
This study: <u>DCC functionality implementation in C with High Level Synthesis tool</u>

- DCC IH, SR, EM functionalities described in C, compiled into single DCC TOP IP
- HLS DCC implements 8x FE + 1x SR + 1x "pseudo" DAQ channel
- All external links (FE, SR, DAQ) abstracted on top of 10Gb UDP
- Xilinx Vivado HLS 2014.4 – C/C++ to HDL compiler
- 10Gb UDP IP cores external to HLS



FMC-QSFP+

Xilinx VC709



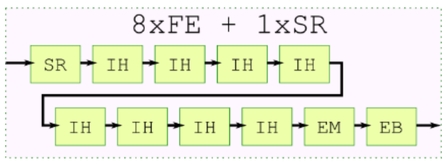
DCC HLS – Design space exploration:

- 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

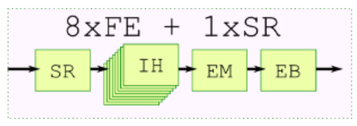
Step 1 – Default HLS compiler constraints

- 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



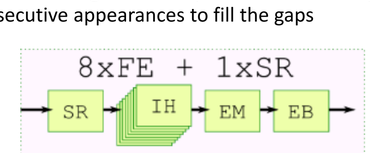
Step 2 – Parallelize tasks

- 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

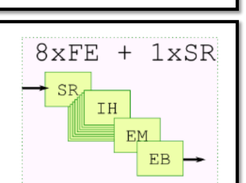


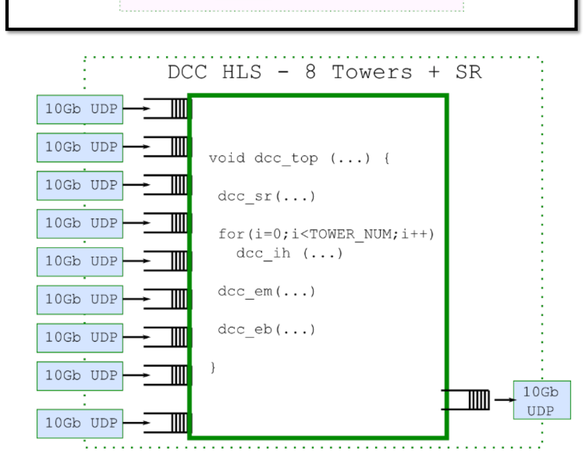
Step 4 –Partition arrays into parallel BRAMs

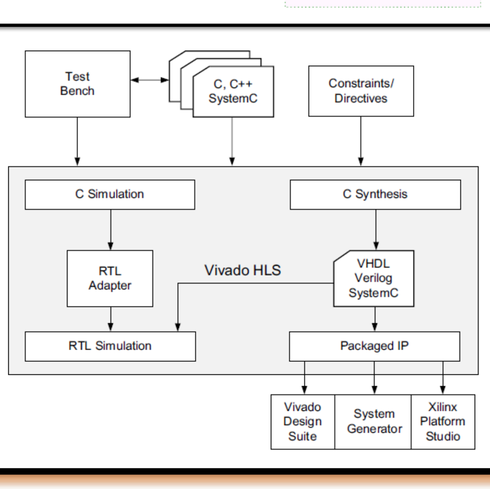
- FPGA has thousands of dual port BRAM memories – utilize them to improve throughput (more RAM ports, vectorized operations) and lower latency
- Step 3 + Apply array partitioning on internal arrays, splitting single array onto Nx BRAMs, virtually creating N port BRAM

Step 5 – Pipeline tasks

- Pipeline tasks' execution
- Partially overlapping computations

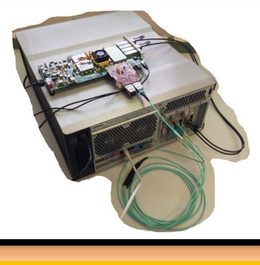






HLS DCC Hardware evaluation

- Xilinx VC709 development kit with Virtex-7 690 device
- FMC Dual QSFP+ = 8 Front End channels
- Two SFP+ on the VC709 = SR and DAQ channels
- Sandy Bridge dual CPU server
- 5x Dual 10 Gb NICs (Total of 10x 10 Gb UDP streams)
- Work in progress at the time of CHEP 2015



<u>Conclusions</u>

- **Engineer who would use HLS tool could possibly achieve higher productivity than non-HLS enabled engineer**
- **Junior engineer can gain high productivity gain by using HLS when supervised by experienced engineer.**
- **We have shown that Vivado HLS tool is capable enough to take a basic DSP like behavioural C code and translate it to HDL**
- **Directives of HLS compiler are powerful way to perform design space exploration.**

<u>Future work</u>

- Integrate production DCC testbench (SV+AVM) with HLS RTL Co-simulation
- Investigate different C coding styles for streaming data processing algorithms