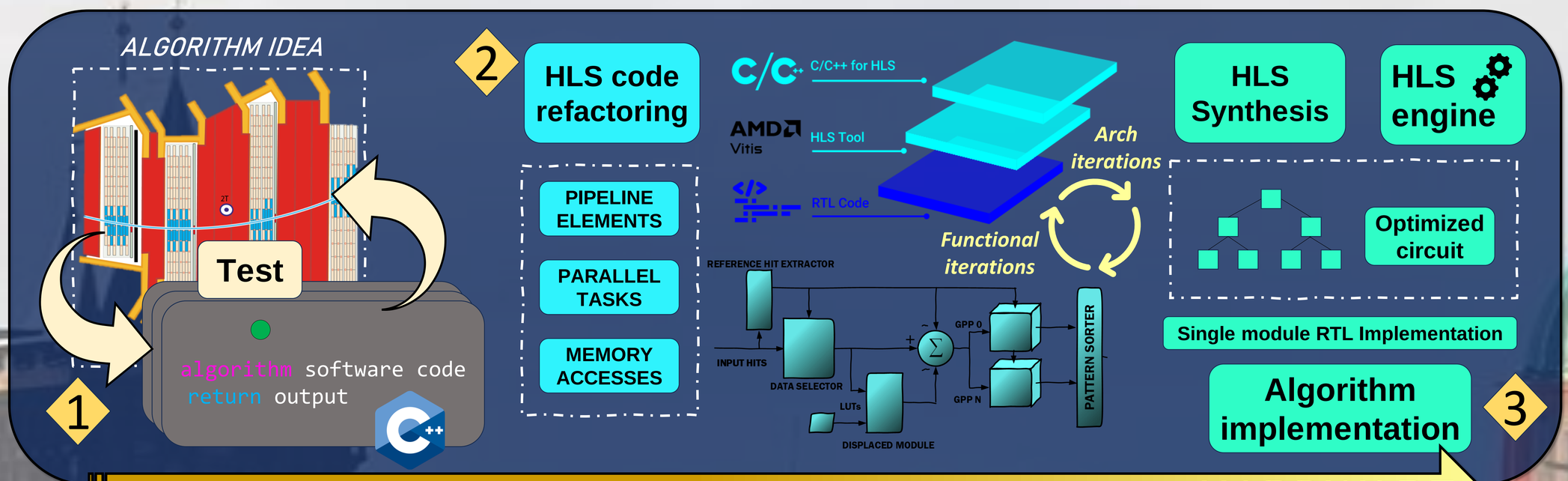
From Software to Hardware: An easy guide to accelerate algorithms for the HL-LHC upgrades of CMS Level-1 Trigger System

Funded by the European Union
European Research Council
Established by the European Commission
GOBIERNO DE ESPAÑA · MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES · AGENCIA ESTATAL DE INVESTIGACIÓN

The 42nd International Conference on High Energy Physics (ICHEP2024) – Prague, July 2024

Pelayo Leguina López – University of Oviedo, on behalf of the CMS Collaboration

Universidá d'Uviéu



MOTIVATION

Writing our algorithms in High Level Synthesis (HLS) language enables a faster developing iteration time for FPGA design.

- Users **can apply directives to the C code** to create the Register Transfer Level (RTL) specific to a desired implementation.
- C simulation can be used to validate the design **and allows faster iterations than a traditional RTL-based simulation**.
- Increased level of abstraction accelerates development.
- Verification time is reduced.
- **New possibilities for triggering on new physics signatures**.

FIRST STEPS

We will use the case study of a track-finding algorithm for muon reconstruction with the CMS experiment to show some of the optimizations with HLS [1].

1 *Design our algorithm and implement it in C++ in a modular approach:*

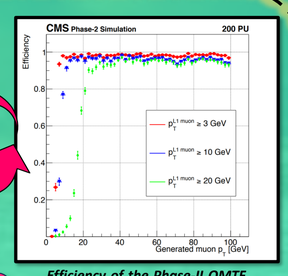
Detect muons in the **overlap region** of CMS detector

Algorithm **receives primitives from muon subdetectors**

- **Algorithm infers the muon's transverse momentum** based on the primitive's positions on each detector layer, using a **naïve Bayes classifier and a NN**

Module 1 .cpp

- Gather ideas
- Design your software modules
- Test to achieve expected functionality
- Check algorithm efficiency



Efficiency of the Phase-II OMTF algorithm for a sample with an average of 200 pile-up events as a function of muon P_T [1].

2 *Refactor our code into synthesizable hardware:*

MAIN PARADIGMS

Producer-Consumer
- Master thread initializes, **forks child threads for parallel tasks**, main thread collates results.
- Convert sequential program, **extract parallel functionality** to improve performance.

Streaming Data
- Streams represent unbounded, **continuously updating data sets**.
- FIFO/PIPO buffers facilitate data flow between producer and consumer processes, **improving throughput and preventing bottlenecks**.

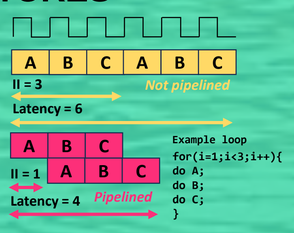
Pipelining
- Parallel tasks at instruction level (ILP), like a car factory production line.
- ILP boosts pipeline efficiency by **ensuring high data production/consumption rates**.

~~Dynamic~~ -> Static
- To be able to synthesize a hardware implementation the design must be fully self-contained, specifying all required resources.

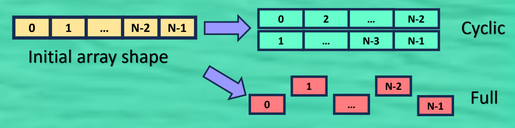
Data types
- Using standard C/C++ data types based on 8-bit boundaries can lead to inefficient hardware implementations, whereas arbitrary precision (AP) data types allow for smaller bit-widths, resulting in more efficient hardware operations, faster execution, and better utilization of FPGA resources.

MAIN STRUCTURES

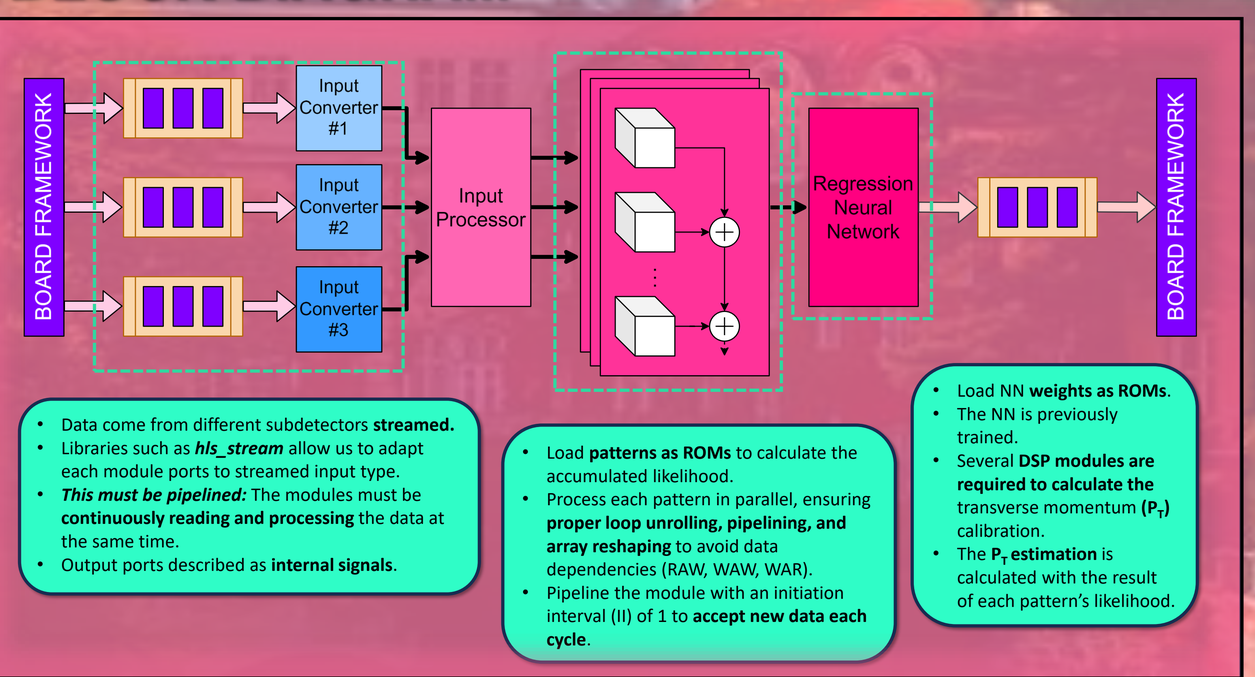
LOOPS
- Pipelining loops enables overlapping iterations, improving execution efficiency by starting new iterations before the previous one's finish.
- The Initiation Interval (II) defines the cycle delay between iterations; II = 1 is ideal, starting each iteration in the next cycle.



ARRAYS
- Arrays mapped to hardware can be registers or various types of memory (RAM, ROM), with different synthesis outcomes based on array size and optimization settings, like Array Partition and Array Reshape.
- Top-level arrays access external memory via RTL ports, while internal arrays are synthesized as registers or block RAM depending on their size.



BLOCK DIAGRAM

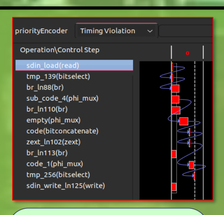


- Data come from different subdetectors **streamed.**
- Libraries such as ***hls_stream*** allow us to adapt each module ports to streamed input type.
- ***This must be pipelined:*** The modules must be **continuously reading and processing** the data at the same time.
- Output ports described as **internal signals**.

- Load **patterns as ROMs** to calculate the accumulated likelihood.
- Process each pattern in parallel, ensuring **proper loop unrolling, pipelining, and array reshaping** to avoid data dependencies (RAW, WAW, WAR).
- Pipeline the module with an initiation interval (II) of 1 to **accept new data each cycle**.

- Load NN **weights as ROMs**.
- The NN is previously trained.
- Several **DSP modules are required to calculate the** transverse momentum **(P_T)** calibration.
- The **P_T estimation** is calculated with the result of each pattern's likelihood.

IMPLEMENTATION



After designing our kernels and synthetizing, the tool can help us to trace resource limitations or timing closure problems.

If the module has timing problems...
- Simplify the logic
- Reduce logic levels
- Reduce data path

For an improved synthesis report, export RTL with param *-flow impl* active and in verilog.

Check the reports for detailed info !!

COMMON SYNTH PARAMS.
- Vitis HLS library.
- HLS global params.
  - FPGA part
  - Clk period
  - CSIM/COSIM
- Export path
- Flow

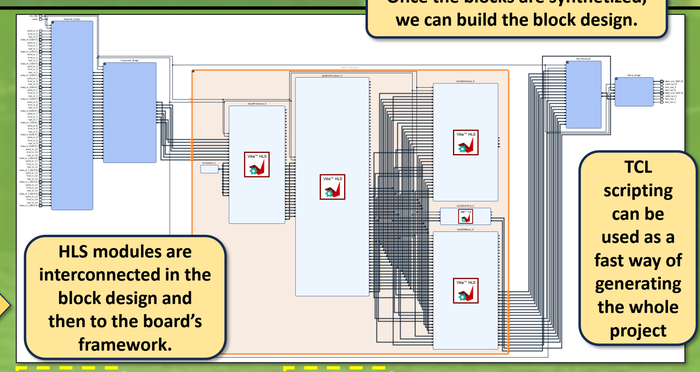
Build TCL script

Module Sources → Apply params → Make target → CoSIM + Flow → Export IP+Token → Timing check Unzip → To IP catalog

We can parallelize module synthesis

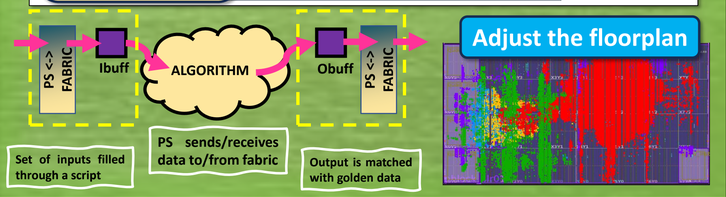
3

Once the blocks are synthetized, we can build the block design.



HLS modules are interconnected in the block design and then to the board's framework.

TCL scripting can be used as a fast way of generating the whole project



Set of inputs filled through a script

PS sends/receives data to/from fabric

Output is matched with golden data

INTEGRATION

Cmake custom pipeline

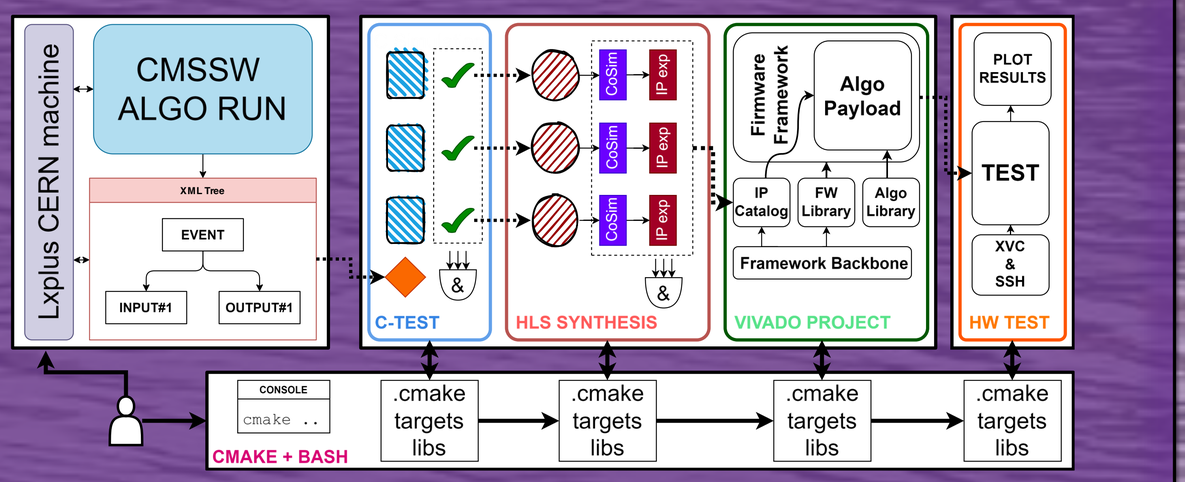
Allows the designer to automatize the implementation process

First, run the emulation SW and obtain the test events.

Second, each module's target is tested and the results logged.

Third, every module is synthetized and co-simulated to check HW results match SW.

Finally, the modules are combined and loaded into the firmware



The next step is to migrate to an online version control tool and utilize a Continuous Integration service

[1] The Phase-2 Upgrade of the CMS Level-1 Trigger, Tech. Rep. CMS-TDR-021, CERN, Geneva (2020) https://cds.cern.ch/record/2714892