CERN Summer Student Report

30 August 2024

Comparison of HLS and RTL Implementation of a Switch Matrix

Filip Trajkovski
Gabriel Alberto Barrios de Leon
Supervisor: Marcos Vinicius Silva Oliveira

Keywords: `ATLAS, FPGA, VHDL, Switch Matrix, HLS, RTL, Logic Utilization, Fmax`

Summary

This report presents a detailed comparison between High-Level Synthesis (HLS) and Register-Transfer Level (RTL) implementations of a sparse Switch Matrix, a critical component in FPGA designs used within the Liquid Argon Calorimeter (LAr) system of the ATLAS experiment at CERN. The study focuses on optimizing design abstractions to improve key metrics such as Logic Utilization and Timing Performance. Through various HLS strategies, including combinational, sequential, and inline implementations, the report explores the trade-offs in resource efficiency and timing between HLS and traditional RTL designs. Additionally, it highlights how optimizations at the synthesis stage can influence FPGA performance and offers valuable insights for future work with HLS.

Contents

1 Introduction

The efficient design of digital systems, particularly of Field-Programmable Gate Arrays (FPGAs), is critical to optimizing performance, resource usage, and power consumption. A key component in these systems is the Switch Matrix, a complex element that routes multiple inputs to multiple outputs based on selection signals. This report presents an in-depth comparison between the High-Level Synthesis (HLS) and Register-Transfer Level (RTL) implementations of a sparse Switch Matrix, emphasizing the optimization of design abstractions to improve metrics such as Fmax (maximum operating frequency) and Logic Utilization.

2 LATOME

2.1 The Large Hadron Collider

The Large Hadron Collider (LHC) is the most powerful accelerator ever constructed: it boosts particles in a loop of 27 kilometers in circumference and generates collisions at 13 TeV. The beams inside the LHC are made to collide in locations corresponding to big particle detectors: ATLAS, CMS, ALICE, and LHCb [1].

2.2 The ATLAS Experiment

ATLAS is a general-purpose particle physics experiment at the LHC at CERN, and it is the largest detector ever constructed for a particle collider. It records the high-energy particle collisions of the LHC, which take place at a rate of over a billion interactions per second [2].

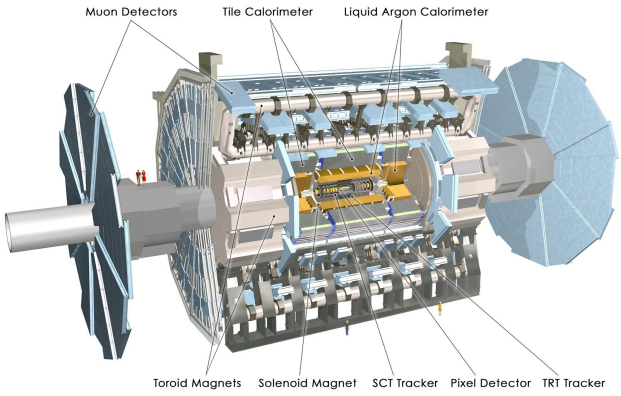

Figure 1: Overview of ATLAS.

2.2.1 Calorimeters

In a particle collider, a calorimeter is a machine that absorbs most of the particles coming from a collision by measuring their energy. Then, the deposits of energy become electric

signals that are reconstructed and analyzed to get a better understanding of the process. The ATLAS Experiment has two calorimeters: the Liquid Argon Calorimeter (LAr) and the Tile Hadronic Calorimeter.

While the LAr Calorimeter surrounds the ATLAS inner detector and measures the energy of electrons, photons and hadrons, the Tile Hadronic Calorimeter surrounds the LAr and measures the energy of hadronic particles that don't deposit all of their energy in the LAr [3].

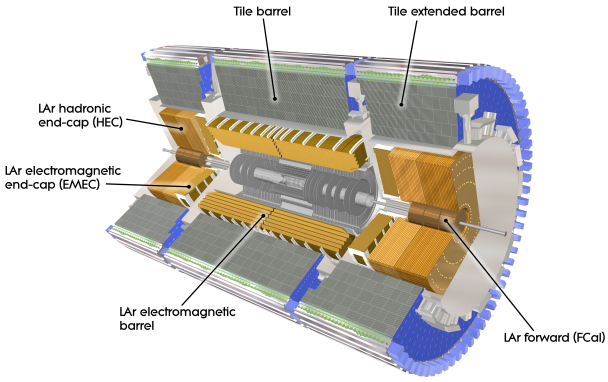

Figure 2: ATLAS Calorimeter System.

2.2.2 The LAr Calorimeter and LATOME

The incoming particles ionize the liquid Argon between the layers of the detector, producing an electrical signal that can be measured. By combining all of the detected currents, it is possible to determine the energy of the original particle that hit the detector [3].

The LATOME is a mezzanine card used in the ATLAS experiment at CERN, specifically within the Liquid Argon Calorimeter (LAr) system. Its primary function is to handle the real-time processing of data from the calorimeter, converting the raw electrical signals generated by particle collisions into meaningful digital information that can be analyzed by scientists [7].

The firmware is crucial for efficiently managing the large volumes of data produced during collisions, ensuring that only the most relevant information is retained for further analysis. It does so by performing tasks such as data compression, signal filtering, and triggering, all within the stringent timing constraints required by the high-speed environment of the LHC.

3 Understanding the Switch Matrix

A switch matrix in digital systems is essential for routing signals in FPGA designs. It can be categorized as:

- Dense (Full-Mesh) Switch Matrix where every input can connect to every output, providing maximum flexibility at the cost of increased complexity and resource usage.
- Sparse (Partial-Mesh) Switch Matrix which allows only certain predefined connections, reducing complexity and resource usage, making it more suitable for specific applications.

The ISM is a critical component within the LATOME firmware used in the Liquid Argon Calorimeter system. It functions as a switch matrix that manages the dynamic routing of input signal from the front-end electronics to the appropriate processing modules within the firmware [6].

The project's objective is to compare the HLS implementation of a sparse Switch Matrix (ISM) with its traditional RTL implementation using VHDL. By analyzing metrics such as Fmax and Logic Utilization, the study seeks to optimize these designs, offering insights into potential improvements in FPGA implementations.

4 Metrics for Evaluation

When comparing the HLS and RTL implementations of a sparse Switch Matrix, it is essential to evaluate the designs based on specific, quantifiable metrics. These metrics help in understanding the efficiency, performance, and overall feasibility of each implementation in real-world FPGA applications. The two critical metrics used in this comparison are: Logic Utilization (in ALMs) and Timing Performance.

4.1 Logic Utilization

The Logic Utilization (in Adaptive Logic Modules - ALMs) metric refers to the amount of hardware resources used by the synthesized design on an FPGA. It is a key indicator of how efficiently the design uses the available hardware resources. Higher ALM utilization means more FPGA resources are being consumed by the design, which could impact the ability to fit the design on the FPGA or leave room for additional logic. Efficient use of ALMs is important for optimizing both the performance and cost of the design, especially in resource-constrained environments [5].

4.2 Timing Performance

Timing Performance is a critical factor that determines the operational ability and speed of the FPGA design. It is primarily expresses through two related metrics: Slack and Fmax.

4.2.1 Slack

Slack is a crucial timing analysis parameter that measures the difference between the required time and the actual time taken by a signal to propagate through a path in a design [5].

Positive slack occurs when the actual time is less than the required time. The design meets the timing requirements, and there is some "margin" available before timing constraints are violated. Positive slack indicates that the design is operating within its timing budget and can handle variations in process, voltage, and temperature (PVT) without timing failures.

Negative slack occurs when the actual time exceeds the required time. The design fails to meet its timing constraints, leading to potential timing errors and unreliable operation. It is critical and requires designers to optimize the path, either by speeding up the logic, reducing the clock frequency, or reworking the design to meet timing requirements.

4.2.2 Fmax

The Fmax summary metric refers to the maximum clock frequency at which a design can reliably operate on an FPGA without violating timing constraints. It is determined by the critical path in the design, which is the longest delay path between sequential elements when considering both logic delays and routing delays. Fmax is calculated as the inverse of the critical path delay, and is a direct measure of the performance of the FPGA design. A higher Fmax indicates that the design can run at a faster clock speed, leading to higher throughput or better performance [5].

5 HLS Strategies

High-Level Synthesis (HLS) offers an abstract approach to hardware design, where high-level programming languages like C/C++ are used to describe the functionality of the hardware, which is then automatically synthesized into RTL code. This section elaborates on the HLS strategies employed in the project to implement the sparse Switch Matrix, focusing on different approaches to optimize performance and resource utilization.

5.1 Block C-CORE Combinational Strategy

In the C-CORE combinational implementation, multiple operations within the block are combined into a single cycle. This means that the operations inside the block are scheduled to execute concurrently within a single clock cycle, as long as the hardware resources and dependencies allow it. It is synthesized as a purely combinational logic block, meaning it has no internal state or storage elements. The outputs of this block depend only on the current inputs, with no memory of past inputs [8].

5.2 Block C-CORE Sequential Strategy

In the C-CORE sequential implementation, operations within the block are executed sequentially over multiple clock cycles. Each operation is scheduled to execute in a different clock cycle, adhering to a sequential order. It is synthesized with internal state elements, such as flip-flops, meaning it can retain information across clock cycles [8].

5.3 Inline Strategy

The inline implementation refers to the practice of integrating the block's operations directly into the calling function or module, without creating a separate hardware block or module. Essentially, the block's functionality is expanded inline with the calling context. Inlining is often used to reduce function call overhead and can potentially lead to optimizations where the compiler or synthesis tool can better optimize the code. The code for the block is "inlined" into the higher-level module, meaning its logic is merged with the logic of the parent block [8].

6 RTL Switch Matrix Implementation

The initial phase of the project involved developing a multiplexer in VHDL, which is a foundational component for constructing the Switch Matrix. The first iteration was designed with specific input and output configurations - four input ports and two selection ports controlling a 1-bit output. This basic multiplexer was thoroughly tested using a testbench in QuestaSim, ensuring correct functionality.

Subsequently, the design was enhanced to support an arbitrary number of inputs, resulting in a new entity called *mux*. Achieving an arbitrary number of inputs necessitated the definition of a new VHDL package for handling 2D arrays, which required compatibility with VHDL 2008 due to its advanced features for array handling. The architecture of *mux* was designed to operate under a clock signal (*clk_i*), ensuring that the selection inputs change with each rising clock edge. Consequently, the outputs also adjust according to the clock period. The bit-width for each input was parameterized, with specific values set: the data width was 13 bits, the data length was 18 bits, and the selection width was 5. To verify the functionality of this generic multiplexer, another testbench was developed and successfully tested in QuestaSim.

Finally, the project culminated in the definition of a more complex entity, *switch_matrix* (SM), which processes a 2D matrix of inputs. Building on the constants defined for *mux*, additional parameters were established for the SM: 384 inputs, 320 outputs, and a *bcid* width of 12 bits. Several ports were introduced to accommodate the expanded functionality: the selection inputs are now 2D array with dimension 320 x 5, while the data inputs are configured as 384 x 13 array. The *bcid* is represented as a 12-bit *std_logic_vector*. The *switch_matrix* relies on the previously developed *mux* entity and the associated packages, passing the inputs to the generic multiplexers for further processing.

To ensure the robust verification of the *switch_matrix*, the project employed Cocotb (Coroutine-based cosimulation testbench), an open-source framework that facilitates the simulation and verification of VHDL designs from Python scripts [4]. Cocotb automates the process of uploading files and creating testbenches in QuestaSim, significantly streamlining the verification workflow. For this project, an existing Cocotb script, originally developed by the LATOME group, was modified to generate random signals for the selection inputs of the *switch_matrix*. This modification allowed for comprehensive testing under various

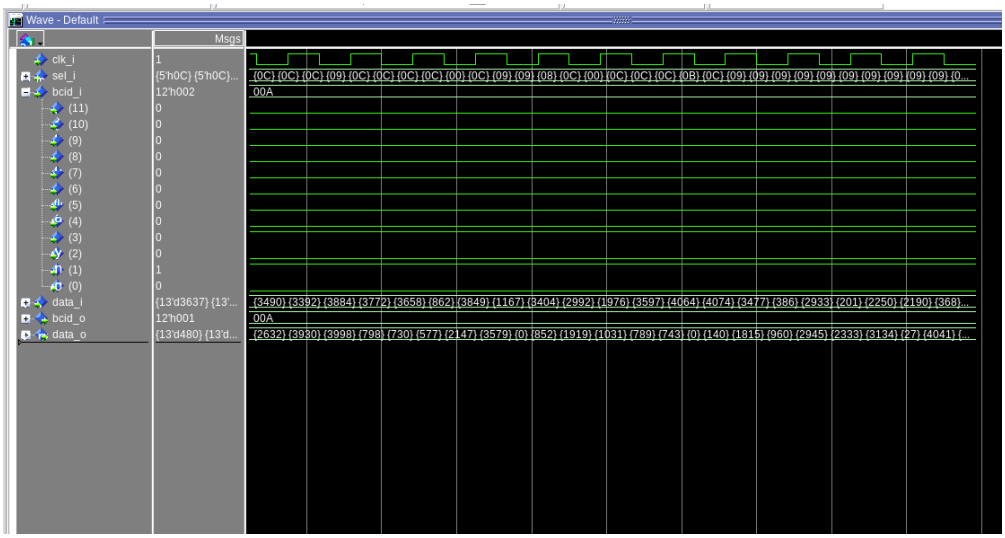

Figure 3: SM wave in QuestaSim generated using Cocotb.

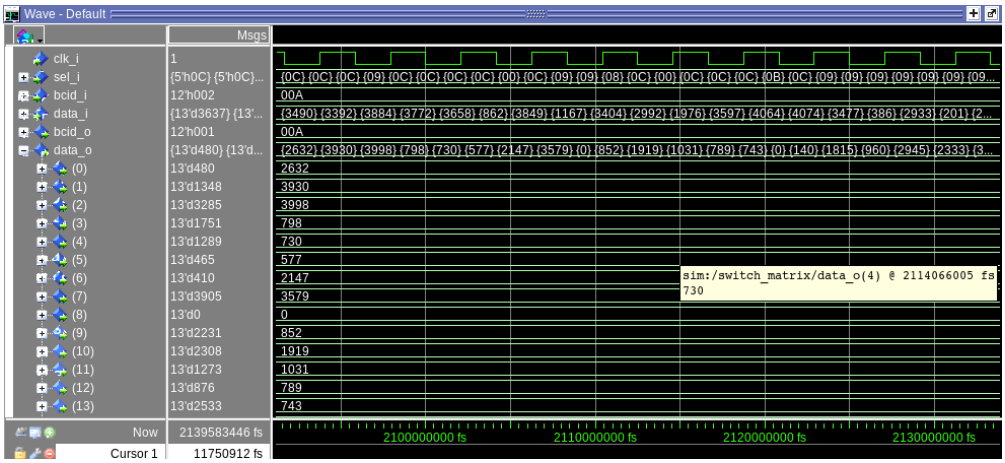

Figure 4: SM outputs in QuestaSim generated using Cocotb.

conditions, ensuring that the *switch_matrix* would operate reliably in diverse scenarios. To execute the Cocotb tests, a dedicated Conda environment was set up. A Makefile was created to automate the setup process, allowing easy access to the test environment. This Makefile facilitated the opening of QuestaSim with all necessary files preloaded and the desired ports configured in the wave interface for simulation and analysis.

The wave interface of the SM can be appreciated in Figures 3 and 4. All the simulations passed.

7 Collaboration with Siemens

The first step after creating the VHDL Switch Matrix was to compare it with the HLS one utilized in ISM. To obtain the results for the key metrics for the different implementations of the Switch Matrix we used the compilation reports from Catapult-Quartus integration and

standalone Quartus compilations. For the purposes of this research we used Quartus version 22.1. The clock is set to 280 MHz in both cases, and the latency is one clock cycle.

In Table 1 are presented the initial results we obtained where we can notice that the RTL solution has lower Logic Utilization compared to the HLS implementation. We decided to provide the results to Siemens support asking them about the reasons behind the difference of the area of the two comparable implementations.

Siemens identified that the HLS code does not optimize out different mux instances in

Table 1: Comparison of resource utilization by HLS and RTL implementation of a Switch Matrix

Method	Abstraction	Logic Utilization (in ALMs)
Quartus	RTL	20806/427200 (5%)
Catapult-Quartus Integration	HLS	27579/427200 (6.5%)
Quartus	HLS	32073/427200 (8%)

the generated RTL when using the same C-CORE for every mux instance. In the RTL Switch Matrix, Quartus exploits from the beginning the fact that most of the multiplexers are choosing between fewer than the full 18 inputs, and thus is able to optimize the Logic Utilization. However, Catapult generated RTL code instantiates 320 copies of the same full 18:1 multiplexer functions.

Furthermore, they suggested modifying the HLS approach to make the constant values visible and enable optimizations that could lead to a significant reduction in the area to approximately 62% of the original. The key is to make sure the same optimizations done in Quartus from the beginning are also implemented in the HLS step, resulting in moving this optimization from the downstream RTL synthesis step to the HLS abstraction step. This can be achieved by creating a custom unique C-CORE module for each of the 320 instances by making the constant values from the ISM configuration (mapping) array visible to the function. The group is currently evaluating these suggestions and expect them to contribute to additional optimization.

8 Results and Analysis

While waiting for the feedback from Siemens, we continued our exploration with trying out different strategies for the HLS solution, in particulat the block C-CORE combinational, block C-CORE sequential, and inline option for the mux instance which is part of the Switch Matrix. In Table 2 are shown the obtained results where it can be clearly observed that the inline HLS solution reduced the area even further than the RTL (VHDL) implementation going down to 19026 ALMs.

After this, we focused on experimenting with different compiler settings for the inline HLS solution in order to see whether this can have an impact on the Logic Utilization, as well as

Table 2: HLS with different implementations for the mux instance

Method	Block type for mux	Wrapper	Logic Utilization (in ALMs)	Fmax	Slack from Catapult	Slack from Catapult-Quartus Integration	Slack from Quartus
Catapult-Quartus Integration	block C-CORE combinational	No	27579	/	1.59	-0.94	/
Quartus	block C-CORE combinational	No	32088	314.56 MHz	/	/	0.392
Quartus	block C-CORE combinational	Yes	31930	200.92 MHz	/	/	-1.406
Catapult-Quartus Integration	block C-CORE sequential	No	27681	/	1.17	-1.01	/
Quartus	block C-CORE sequential	No	32069	304.79 MHz	/	/	0.29
Quartus	block C-CORE sequential	Yes	32076	198.53 MHz	/	/	-1.466
Catapult-Quartus Integration	inline	No	19026	/	1.57	-1	/
Quartus	inline	No	22893	370.64 MHz	/	/	0.873
Quartus	inline	Yes	25234	263.78 MHz	/	/	-0.22

the timing metrics such as Fmax and Slack. For this purpose, we tried out three different compiler settings: balanced (normal flow), high and superior performance with maximum placement effort. The results are given in Table 3. It can be observed that the high and superior performance with maximum placement effort settings result in higher Fmax (and Slack) and comparable area to the default balanced (normal flow) setting which can be seen as a slight improvement from the default balanced setting.

Table 3: HLS inline implementation with different compiler settings

Compiler settings	Wrapper	Logic Utilization (in ALMs)	Slack	Fmax
Balanced	No	22893	0.873	370.64 MHz
Balanced	Yes	25234	-0.22	263.79 MHz
High performance with maximum placement effort	No	22934	1.238	428.63 MHz
High performance with maximum placement effort	Yes	24344	0.038	283.05 MHz
Superior performance with maximum placement effort	No	22908	1.085	402.25 MHz
Superior performance with maximum placement effort	Yes	25380	0.189	295.68 MHz

Then, we decided to perform the same experiments with the compiler settings on the VHDL Switch Matrix and see whether this will also play a role. From the results in Table 4 can be seen that this is also the case here as the superior performance with maximum placement effort has higher Fmax which comes with a slightly higher Logic Utilization cost.

Table 4: VHDL Switch Matrix with wrapper and different compiler settings

Compiler Settings	Logic Utilization (in ALMs)	Slack	Fmax
Balanced	20218	-0.013	279.02 MHz
High performance with maximum placement effort	20131	0.313	306.94 MHz
Superior performance with maximum placement effort	21350	0.388	314.17 MHz

Since all the previous tests were done with isolating the Switch Matrix, in the last step we decided to replace the ISM in LATOME firmware with the one that can potentially have better results and test it when it is fully integrated into the firmware. Based on our previous experiments that was the ISM with inline HLS solution. In Table 5 is shown a comparison of the Logic Utilization between the current Remap block and ISM instance with the proposed ones. In both cases the area is lower for the proposed solution by around 3500 ALMs.

Table 5: Comparison of resource utilization between current ISM and the proposed one with inline high-level mux

Entity	Resource Utilization (in ALMs)
Current Remap	28016
Current ISM inst	26928
Proposed Remap	24545
Proposed ISM inst	23468

9 Conclusion

The project provided the team with crucial experience in HLS, particularly in colaboration with Siemens. The optimizations and strategies explored in this study offer potential for reducing the area used by switch matrices compared to current firmware implementations. Future work will focus on evaluating the custom C-CORE modules and further refining the HLS implementations for better performance and efficiency.

10 Acknowledgements

We would like to express our sincere gratitude to Marcos Vinicius Silva Oliveira, our supervisor, for his invaluable guidance, support, and mentorship throughout our summer

internship at CERN. His expertise and insights have significantly shaped our understanding of High-Level Synthesis and firmware design.

We also extend our gratitude to the entire LATOME HLS team for their collaboration and encouragement, which made this experience truly rewarding and enjoyable.

Lastly, we are grateful to CERN for providing an exceptional platform for learning, exploration and innovation, and for giving us the opportunity to work here as summer students.

References

[1] CERN. *Large Hadron Collider*. URL: https://home.cern/science/accelerators/large-hadron-collider. (accessed: 23.07.2024).

[2] CERN. *The ATLAS Experiment*. URL: https://atlas.cern/about. (accessed: 23.07.2024).

[3] ATLAS Collaboration. *Calorimeter*. URL: https://atlas.cern/Discover/Detector/Calorimeter. (accessed: 23.07.2024).

[4] FOSSi Foundation. *COCOTB*. URL: https://www.cocotb.org/. (accessed: 23.07.2024).

[5] Intel. *Intel® Quartus® Prime Standard Edition User Guide: Design Optimization*. https://www.intel.com/programmable/technical-pdfs/qps-ugs.pdf. 2024.

[6] Marcos Vinicius Silva Oliveira. *Firmware Insights*. 2022. URL: https://indico.cern.ch/event/1203458/#78-firmware-insights.

[7] Marcos Vinicius Silva Oliveira. *LATOME Firmware in the Long-Term*. 2022. URL: https://indico.cern.ch/event/1226981/#84-latome-firmware-in-the-long.

[8] SIEMENS EDA. *Catapult® Synthesis HLS Bluebook*. Software Version 10.6a. Version 10.6a. Apr. 2021.