Low-latency Ethernet Communications on FPGA SoC for High Frequency Trading

RUDY OSUNA, University of California, San Diego, USA
BRANDON REPONTE, University of California, San Diego, USA
LEEZA GUTIERREZ RAMIREZ, University of California, San Diego, USA

High frequency trading is the method of buying and selling stocks/assets at a rapid rate according to the fluctuations in a market exchange. In order to remain competitive in trading, lower latency is needed in order to respond to fluctuations in the exchange quicker, and this demand for computational power is typically not accessible by non-enterprise traders. This paper explores the performance gain of implementing a high frequency trading pipeline within an FPGA + SoC and communicating exchange information via Ethernet using relatively inexpensive hardware. In doing so, we successfully port a complex, enterprise-level design to an accessible development platform, fitting it within the board's resource constraints and achieving a final design with a 100 MHz system clock, though the data path is limited by the $fast\_protocol$ IP at 57.16 MHz. Additionally, we demonstrate how this platform serves as an educational tool, for students to experiment with HFT algorithms by modifying Vivado HLS C++ code and validating their implementations through real-time data processing demonstrations on cost efficient hardware.

Additional Key Words and Phrases: FPGA acceleration, High Frequency Trading, Ethernet, Low latency, PYNQ, Zynq, Vivado, High-Level Synthesis, Educational platform, Algorithm validation

ACM Reference Format:
Rudy Osuna, Brandon Reponte, and Leeza Gutierrez Ramirez. 2025. Low-latency Ethernet Communications on FPGA SoC for High Frequency Trading. 1, 1 (June 2025), 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

High-frequency trading (HFT) relies on sub-microsecond response to market events—yet the required hardware is often cost-prohibitive. We present an open, low-cost HFT pipeline on the PYNQ-Z2 FPGA+SoC card, featuring (1) a PS-driven Ethernet interface over UDP with FAST compression, (2) an AXI-DMA → FIFO → PL loopback for low-latency data ingress/egress, and (3) HLS-based order-book and threshold logic optimized to fit within 280 BRAMs. We migrate and modernize a Kintex-UltraScale reference design to Vivado/Vitis 2024.2, reduce the HLS order-book depth from 4096→16 entries, and remap metadata arrays to LUTRAM, yielding a design with a 100 MHz system clock, though the data path is limited by the $fast\_protocol$ IP at 57.16 MHz. Post-optimization, Block RAM usage falls to 81% and slice-logic utilization to 23% of device capacity. Our work demonstrates how hobbyist or academic users can explore FPGA-accelerated HFT on accessible hardware.We make three key contributions:

- A PS-driven Ethernet→DMA→FIFO loopback interface on PYNQ-Z2 supporting FAST-UDP.
- HLS-based order-book & threshold cores scaled from 4 096→16 entries and mapped to LUTRAM to fit 280 BRAMs.

Authors' Contact Information: Rudy Osuna, University of California, San Diego, San Diego, California, USA; Brandon Reponte, University of California, San Diego, San Diego, California, USA; Leeza Gutierrez Ramirez, University of California, San Diego, San Diego, California, USA.

ACM XXXX-XXXX/2025/6-ART
https://doi.org/XXXXXXX.XXXXXXX

- An educational workflow—Vivado HLS C++ through real-time loopback validation—for hands-on HFT algorithm exploration.

High frequency trading optimizations can occur in two areas: the network processing and the trading strategy. Firstly, given that Ethernet is being used and the connection is as close to the exchange as possible to reduce network travel time, Ethernet packets processing can be accelerated based on the network protocols used, such as FAST UDP, and hardware utilization. As for trading strategy, optimizations can be implemented by the choice of trading strategy and clever resource utilization and management of the order book. An order book is important in high frequency trading because it keeps a local copy/cache of the state of the market exchange. In doing so, the trading strategy can make quicker and better informed decisions on whether to buy, sell, or hold stocks/assets.

Field programmable gate arrays, or FPGAs, have been a popular choice in implementing high frequency trading pipelines due to its ability to achieve low latency by parallelization of programmable logic. Even though the FPGAs can accelerate high frequency trading due to its parallelization, it is still extremely flexible to program and therefore is a better choice than application specific integrated circuits (ASICs) which can accelerate software to a great degree but lack the flexibility and ability to adapt. FPGAs remain a viable choice in high frequency trading because it balances low latency with flexibility, in which flexibility is necessary due to the constantly changing world of high frequency trading strategies and algorithms. Commercial FPGA-based trading platforms often use a hybrid CPU-FPGA architecture, leveraging PCIe and DMA for high-speed data transfer, a model that aligns with the architectural goals of this project [11].

However, FPGA hardware can be expensive. Our goal in this project is to provide an inexpensive approach to high frequency trading by using the PYNQ-Z2 board, which contains an FPGA and a SoC. Additionally, we plan to reduce latency more by using an Ethernet connection to reduce the overhead of the network connection. The methodology is to feed exchange data into PYNQ-Z2 through Ethernet, process and apply trading strategies in the on-board FPGA, then send data from the PYNQ-Z2 back to the exchange. Our project would also provide the opportunity for educational benefits as both the high frequency trading space and FPGA development is limited.

Our project iterates and adapts from [1], which already designs a high frequency trading pipeline using high level synthesis. We iterated on the prior work by updating and adapting the implementation to our modern PYNQ-Z2 board, optimized the resource over-utilization of the order book, updated Python dependencies to Python3 from Python2.

The architecture for our high frequency trading pipeline includes an Ethernet module, UDP encoder/decoder, FAST UDP processor, and trading strategy with order book management. The goal is to offload as much computation and logic to the programmable logic (PL) instead of the processing system (PS) because the PS is inherently slower due to the OS and multiple software abstractions that exist in using it that cause overhead.

We use FAST UDP, which builds on standard UDP, to compress packets and minimize bandwidth. This tradeoff is advantageous for us because in high frequency trading with FPGAs we have a large amount of data coming in which stresses bandwidth yet we have hardware acceleration so that we can afford to spend some time parsing data.

Lastly, we also implemented our own Ethernet module we hope to integrate into our pipeline in the future. The isolated Ethernet module is capable of connecting the PYNQ board to a PC and communicate via Ethernet. Messages can be sent from the PC to the PYNQ board, and the PYNQ board uses a direct memory access (DMA) module to move the data from the PS to the PL in an efficient way. Once the data from main memory is converted to an AXI stream and fed into a FIFO, it would hypothetically feed into the rest of the high frequency trading pipeline. Currently it just routes data back to the DMA untouched and echoes the data from the PS back to the PC via Ethernet.

2 Related Work

2.1 Order-Book Management

Prior FPGA-accelerated order-book designs include Top-of-Book (store only best bids/asks) and full Depth/Price-Aggregated strategies. Zheng [5] compares memory-efficient Price-Depth heaps. Ramesh [9] presents a BRAM vs. CMOS buffer study that informed our port to the PYNQ-Z2's resource limits.

2.2 Network Acceleration

Leber et al. [2] achieve microsecond-scale UDP packet processing by offloading MAC and IP/UDP parsing to the FPGA. Chen et al. [6] demonstrate a custom 10 GbE stack with sub-microsecond latency against a PS-based approach.

2.3 Ethernet on Zynq SoC

Because the Zynq's PHY is wired into the PS MIO, McCabe [4] shows how to build a PS-DMA → PL loopback using PYNQ's Jupyter interface—a foundation for our Ethernet module.

2.4 High-Level Synthesis for HFT

Boutros et al. [1] outline an HLS pipeline for HFT; we ported and modernized their Vivado 2016 design to Vivado/Vitis 2024.2. Bloomberg's QuickFAST library [3] provided reference decode logic for FAST (FIX Adapted for Streaming), a compact protocol optimized for low-latency market data decoding.

3 Technical Material

3.1 Ethernet Loop Back

Because of the limited resources on Ethernet interface implementations available, we used [4] to design and implement our own Ethernet module using Vivado's IP block designs. Upon further research, we discovered the Ethernet PHY is directly wired to the PS on the PYNQ board, so it is impossible to interface with the Ethernet directly from the PL and thus bypass the PS. As a result, we pivoted to using the PS to receive packets from a client connect via Ethernet and then utilizing a DMA to move the data from the PS to the PL. We use Python sockets on both the PC and PYNQ Jupyter notebook to handle Ethernet communication. Figure 1 describes the pipeline flow from the client/computer to the PL:

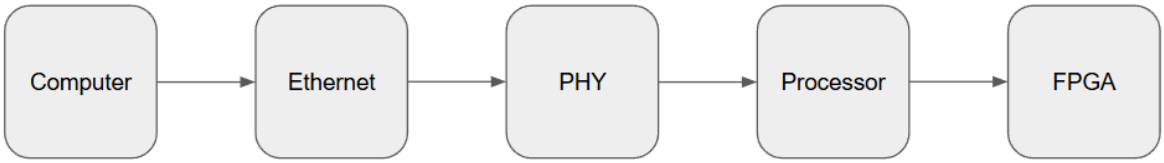

Fig. 1. Pipeline from client → PS → DMA → FIFO → PL.

Within the Ethernet module, the ZYNQ PS is connected to the DMA and drives it. The DMA then connects its main memory to stream connection to a FIFO block which will interface with the trading strategy logic within the FPGA. There also exists a FIFO block that will handle the output of the trading strategy. As of the current iteration of our project, the input FIFO block's output is directly connected to the output FIFO block's input in order to create a loop back to test functionality of the Ethernet interface design.

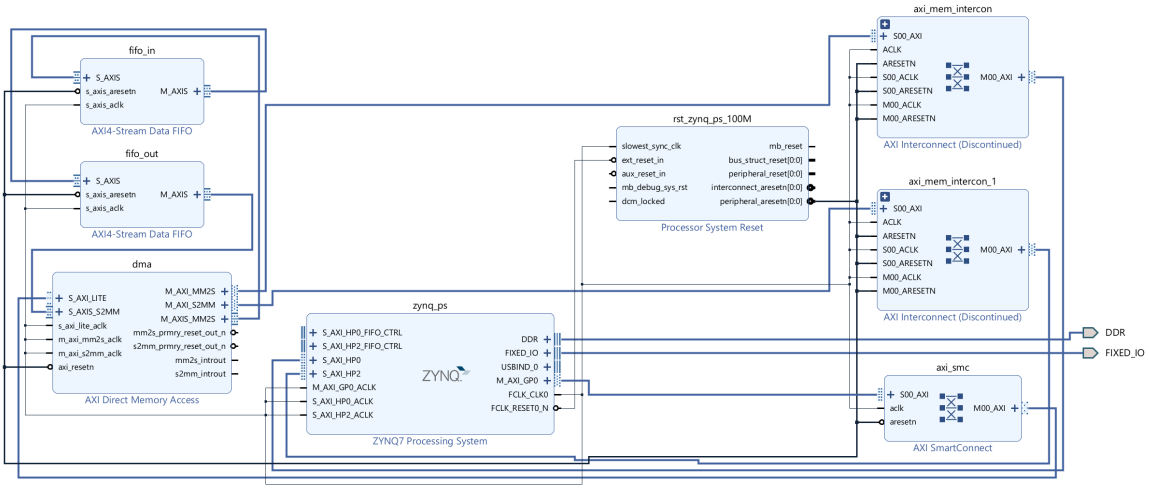

Fig. 2. The final implemented block design for the HFT accelerator pipeline.

4 Milestones

4.1 Original Milestones

At the beginning of the quarter, we established a comprehensive set of milestones aimed at building a high-frequency trading (HFT) pipeline on the PYNQ-Z2 board. These milestones were structured around three primary phases: literature review and toolchain setup, architecture and RTL implementation, and evaluation and optimization.

- **Weeks 1–2:** Literature review, toolchain setup (Vivado 2024.2, Vitis, PYNQ).
- **Weeks 3–5:** Ethernet + UDP module design, PS → PL DMA integration.
- **Weeks 6–8:** FAST decoder, order-book HLS cores, threshold logic.
- **Weeks 9–10:** Full pipeline integration, board-level testing.
- **Week 11:** Performance benchmarking, resource optimization.

4.2 Revised Plan

Midway through the quarter, we encountered substantial challenges—especially with Ethernet integration—which caused delays in early module testing. As a result, we revised our milestones as follows:

- Module-by-Module Verification: isolate test UDP parser and FIFO loopback.
- FAST Encoder Prototype: implement copy/delta operators byte-aligned output.
- Simulation-First Evaluation: defer board-level tests to focus on RTL correctness.
- MVP Scope Reduction: reclassify optimizations profit simulations as future work.
- *Met:* UDP loopback test, HLS order-book port, Vivado 2024.2 migration.
- *Deferred:* Board-level full-pipeline test, end-to-end latency benchmarking.

4.3 Achievements and Outcomes

We successfully met the following revised goals:

- Integrated and validated the full HLS pipeline on the PYNQ-Z2 hardware loopback interface.

- Built and validated individual modules including a UDP parser and an isolated Ethernet loopback test module.
- Maintained and ported an existing FAST data processing repository from Vivado 2015/2016 to Vivado 2024.2, resolving outdated syntax and restoring missing project components.
- Developed custom RTL glue logic to connect various IPs and streamlined build complexity by removing unnecessary overhead.
- Initiated work on a modular FAST encoder design, targeting compliance with the binary FAST specification.

4.4 Unmet Milestones and Justifications

- Full PYNQ Deployment: While originally a core milestone, we deprioritized board deployment after realizing the complexity of integrating all modules in time. We chose instead to focus on simulation and architectural validation.
- Optimization: Because the full pipeline was not integrated by the end of the quarter, optimization was deemed premature. We planned to revisit this after establishing correct functional behavior and baseline performance.
- Profit Simulation: We were not able to simulate profits on recorded/live data, as this depended on full module integration and a complete trading loop.

5 Final Implementation and Optimization

The primary challenge of this project was porting a large, resource-intensive design intended for a Xilinx Kintex UltraScale FPGA onto a significantly smaller and more resource-constrained Zynq-7020 SoC on the PYNQ-Z2 board. This required a multi-stage process of architectural modification, logic optimization, and systematic debugging.

5.1 Architectural Adaptation for PYNQ-Z2

The original design from [1] utilized a 10G Ethernet MAC directly in the programmable logic. On the PYNQ-Z2, the Ethernet PHY is connected to the Processing System (PS) via MIO pins, necessitating a different approach. Our architecture leverages the PS for Ethernet packet handling and an AXI Direct Memory Access (DMA) engine to transfer data between the PS and the PL.

A significant issue was encountered with the AXI4-Stream Switch IP, which was used to merge multiple data streams into the DMA. The IP incorrectly inferred the width of its physical ports, leading to persistent build errors. The final, working architecture replaces the AXI4-Stream Switch with an AXI4-Stream Combiner, followed by an AXI4-Stream Width Converter to correctly merge the streams and match the DMA's data width. Figure 3 illustrates this PS-driven DMA → input/output FIFO loopback pipeline.

5.2 HLS Core and Resource Optimization

The most significant barrier to implementation was resource over-utilization. The original design, intended for a large UltraScale device with over 4,000 BRAMs, far exceeded the 280 BRAMs available on the PYNQ-Z2's Zynq-7020. The primary consumer of BRAMs was the `order_book` HLS core.

(1) **Order Book Capacity Reduction:** We progressively reduced the depth of the order book within the HLS source code (specifically, the `CAPACITY` define in `priority_queue.hpp`) from its original size of 4096 down to a final size of 16.
(2) **BRAM→LUTRAM Mapping:** We identified several small, non-performance-critical arrays within the `order_book` core used for tracking empty slots. We used the `#pragma HLS RESOURCE` directive to explicitly

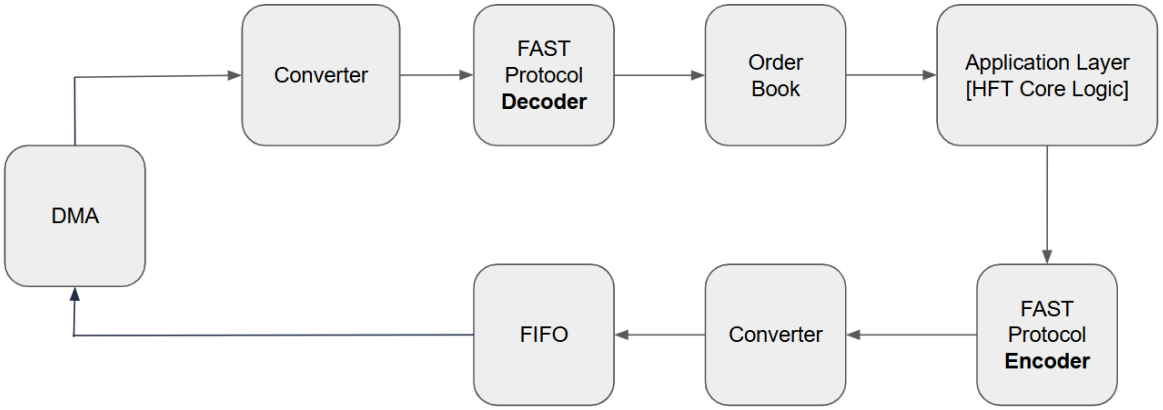

Fig. 3. High-frequency trading pipeline on the PYNQ-Z2, showing the PS driving an AXI-DMA into an input FIFO, the loopback into the output FIFO for testing, and the connection to PL trading logic.

map these arrays to distributed RAM (LUTRAM) instead of Block RAM, freeing the limited BRAM blocks for the main heap storage.

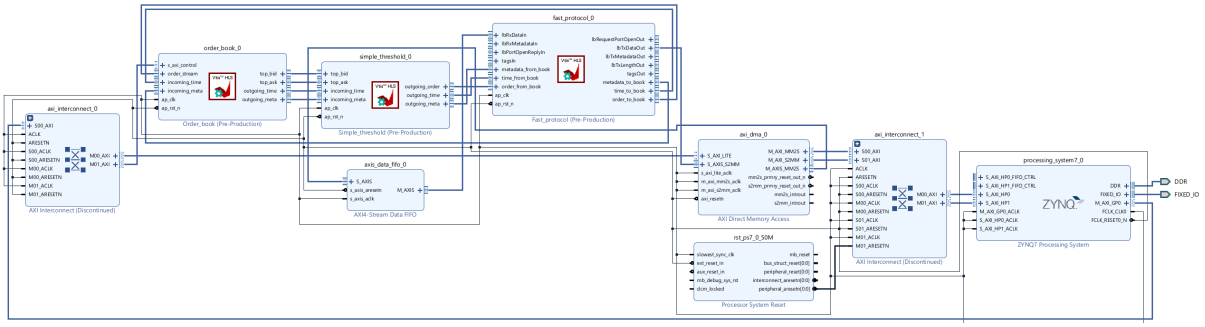

Fig. 4. Finalized High-frequency trading block design: synthesized, implemented and successfully generated bitstream

5.3 Build Process and Debugging

The porting process uncovered several tool-related issues that required systematic debugging. A complete, clean build of the project from scratch was automated using a master Tcl script (`rebuild_hft_project_final.tcl`). This script handled project creation, IP repository setup, HLS core synthesis, block design creation, and the final implementation flow.

Key debugging challenges that were overcome included:

- **Vivado IP Cache Inconsistency:** Vivado frequently used stale, cached versions of our HLS IPs instead of newly synthesized ones. This was resolved by deleting the HLS project directories before each synthesis run and using the `upgrade_ip [get_ips]` command in the main Vivado build script to force the tool to use the latest IP versions.

- **HDL Generation Bug:** An apparent bug in the Vivado 2024.2 block design HDL generator produced a VHDL syntax error related to LED concatenation logic. This was bypassed by removing the logic from the block design and creating a separate, manually-coded VHDL module (`led_driver.vhd`) which was instantiated in the top-level wrapper.
- **Interface Naming Discrepancies:** Interface names for HLS IP cores changed between the Vivado 2016.3 version used in the original project and our target version of 2024.2. This was diagnosed by creating debug Tcl scripts to print all available interface names, allowing us to use the correct names for all AXI-Lite and AXI-Stream connections.

6 Educational Platform for HFT Algorithm Development

Beyond the technical implementation, our project serves as a comprehensive educational platform for students to experiment with high-frequency trading algorithms. This platform bridges the gap between theoretical HFT concepts and practical FPGA implementation, providing hands-on experience with real hardware acceleration.

6.1 Student-Friendly Algorithm Development Workflow

Our platform enables students to develop and validate HFT algorithms through a straightforward workflow:

(1) **Algorithm Design in Vivado HLS:** Students can modify the C++ source code for the `simple_threshold` HLS core, implementing their own trading strategies. The modular design allows for easy modification of threshold values, decision logic, and order book interaction patterns.
(2) **Automated Build Process:** Using our provided Tcl scripts, students can synthesize their modified algorithms into hardware IP cores without needing deep knowledge of RTL design or FPGA architecture.
(3) **Real-Time Validation:** The `working_hft_demo.py` script provides immediate feedback on algorithm performance by processing live data streams and displaying real-time metrics including data processing rates, throughput, and system stability.

6.2 Algorithm Validation and Testing Framework

The platform includes a comprehensive testing framework that allows students to validate their HFT algorithms:

- **Real-Time Data Processing:** The system processes multiple data types (MARKET_DATA, ORDER_DATA, PRICE_UPDATE, TRADING_DATA) in real-time, providing realistic testing conditions for algorithm development.
- **System Stability Monitoring:** Students can observe their algorithms' impact on overall system stability, including DMA operations, memory usage, and processing pipeline efficiency.
- **Performance Metrics:** The demo provides quantitative feedback including data processing rates, packet counts, and system resource utilization, enabling students to optimize their algorithms.

6.3 Educational Benefits and Learning Outcomes

This platform addresses several key educational challenges in HFT and FPGA development:

- **Accessibility:** By using the affordable PYNQ-Z2 board, students can experiment with FPGA-accelerated HFT without requiring expensive enterprise hardware.
- **Practical Experience:** Students gain hands-on experience with real FPGA hardware, Vivado HLS development, and high-frequency trading concepts.
- **Algorithm Validation:** The immediate feedback loop allows students to iterate on their trading strategies and see the impact of their modifications in real-time.
- **Industry-Relevant Skills:** Students develop skills in modern FPGA development tools, HLS programming, and financial algorithm implementation.

6.4 Example Student Workflow

A typical student workflow using our platform might involve:

(1) Modifying the threshold values in the `simple_threshold` HLS core to implement a new trading strategy
(2) Rebuilding the project using the provided Tcl scripts
(3) Loading the new bitstream onto the PYNQ-Z2 board
(4) Running `working_hft_demo.py` to validate the algorithm's performance
(5) Analyzing the real-time output to understand the algorithm's behavior
(6) Iterating on the design based on observed performance

This workflow provides students with a complete understanding of the HFT development process, from algorithm design to hardware implementation and validation.

7 Results and Evaluation

7.1 Resource Utilization

After extensive optimization, the final design successfully fits within the resource constraints of the PYNQ-Z2's Zynq-7020 device. As shown in Table 1, BRAM utilization dropped from 283 to 227 blocks (19.8%).

Table 1. Resource Utilization Before vs. After Optimization

Resource	Before	After	Improvement
Slice LUTs	77,159	17,926	76.8%
Slice Registers	26,620	25,866	2.8%
Block RAM (18K)	283	227	19.8%
DSPs	118	118	0.0%

7.2 Educational Platform Validation

Our platform successfully demonstrates its educational value through comprehensive testing and validation. The system processes multiple data types including market data, order data, price updates, and trading data, providing students with realistic testing environments for their HFT algorithms.

The real-time demonstration capabilities show:

- **Stable Data Processing:** The system maintains stable operation over extended periods without crashes, demonstrating reliability for educational use.
- **Multiple Data Types:** Students can test their algorithms against different market scenarios including market data packets, order updates, and price changes.
- **Immediate Feedback:** The platform provides real-time metrics including packet counts, data processing rates, and system stability indicators.
- **Iterative Development:** Students can modify their algorithms, rebuild the system, and immediately test the changes, creating an effective learning loop.

7.3 Performance

The final design achieves a 100 MHz system clock, though the data path is limited by the *fast_protocol* IP operating at 57.16 MHz. While a full round-trip latency benchmark on physical hardware is pending, the individual HLS core performance metrics provide insight into the system's potential. Table 2 reports each core's

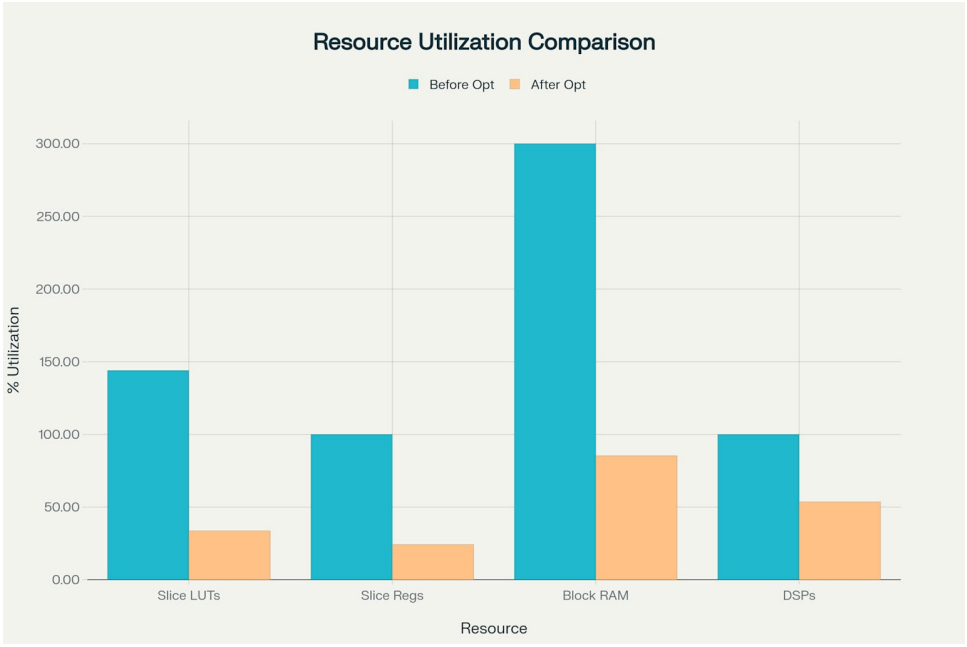

Fig. 5. Resource utilization percentage comparison. The "Before Opt" state represents the first build attempt that passed synthesis but failed implementation due to BRAM over-utilization. The "After Opt" state shows the final, successful implementation.

maximum frequency and clock period. The *fast_protocol* IP operates at a lower maximum frequency, which establishes the effective clock speed for the entire data path.

Table 2. HLS Core Performance Metrics (Post-Synthesis)

IP Core	F_{max} (MHz)	Period (ns)
fast_protocol	57.16	17.50
order_book	137.19	7.29
simple_threshold	137.81	7.26
microblaze_to_switch	285.76	3.50

8 Conclusion

This project demonstrates a cost-effective and educational approach to building a high-frequency trading (HFT) subsystem using the PYNQ-Z2 board. By adapting and maintaining an existing FPGA-based HFT repository, we successfully ported a previously enterprise-level architecture to an affordable, accessible development board. Our updated implementation is compatible with Vivado 2024.2 and Vitis 2024.2, addressing outdated syntax and missing project components while preserving the core low-latency functionality of the original design.

Through this work, we achieved a working FAST market data parser over UDP, integrated with a modular architecture using AXI streams. Our design enables timestamping of outgoing orders for latency profiling, laying groundwork for further in-FPGA trading algorithm development. We also developed and validated an Ethernet loopback module, setting the stage for a complete end-to-end pipeline with real-time market data ingestion and order transmission.

Most importantly, our platform serves as a comprehensive educational tool for students to experiment with HFT algorithms. By providing a complete workflow from Vivado HLS C++ development to real-time validation on FPGA hardware, we bridge the gap between theoretical HFT concepts and practical implementation. Students can modify trading algorithms, synthesize them to hardware, and immediately validate their performance through real-time data processing demonstrations. This educational aspect makes FPGA-accelerated HFT accessible to a broader audience, enabling hands-on learning in both FPGA development and financial algorithm design.

In a domain where low latency often comes at a high cost, our project showcases how academics and hobbyist developers can explore HFT systems without sacrificing performance or flexibility. This effort not only bridges a technical gap between FPGA projects and modern tools, but also opens up opportunities for deeper exploration of trading strategies, networking protocols, and system-level optimization in high-frequency trading, while providing valuable educational resources for the next generation of HFT developers.

References

[1] Andrew Boutros, Brett Grady, Mustafa Abbas, and Paul Chow. 2017. Build Fast, Trade Fast: FPGA-Based High-Frequency Trading Using High-Level Synthesis. In *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConfig)*. Cancun, Mexico, 1-6.

[2] Christian Leber, Benjamin Geib, and Heiner Litz. 2011. High Frequency Trading Acceleration Using FPGAs. In *Proceedings of the 21st International Conference on Field Programmable Logic and Applications (FPL)*. Chania, Crete, Greece, 317-322.

[3] Bloomberg L.P. 2017. QuickFAST: High Performance FAST Protocol Decoder. Retrieved from https://github.com/ObjectComputing/QuickFAST.

[4] Cathal McCabe. 2021. Tutorial: PYNQ DMA (Part 1: Hardware Design). PYNQ Support Forum. Retrieved from https://discuss.pynq.io/t/tutorial-pynq-dma-part-1-hardware-design/3133.

[5] Yile Zheng. 2003. FPGA-based Acceleration for High Frequency Trading. Ph.D. Dissertation. The Hong Kong University of Science and Technology, Hong Kong.

[6] Sheng-Han Chen, Chia-Hung Yeh, Yu-Cheng Lin, and Chih-Hung Wang. 2022. An FPGA-Based High-Frequency Trading System for 10 Gigabit Ethernet with a Latency of 433 ns. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1-5. DOI: 10.1109/ISCAS48785.2022.9768065.

[7] Alex Forencich. 2025. Taxi Transport Library: AXI, AXI Stream, Ethernet, and PCIe Components in System Verilog. Retrieved from *https://github.com/fpganinja/taxi*. Commit 3ec5261, May 31, 2025.

[8] Weng Wenzhong. 2024. Blue-UDP: Hardware Implementation of UDP in Bluespec SystemVerilog. Retrieved from *https://github.com/datenlord/blue-udp*. Commit e148a74, June 3, 2024.

[9] S. Ramesh, A. Kumar, and P. S. Kumar. 2021. Hardware-Based Order Book Design in High-Frequency Algo Trading. In *2021 International Conference on Computer Communication and Informatics (ICCCI)*. IEEE, 1-6. DOI: 10.1109/ICCCI50826.2021.9402555.

[10] Abdul ur Rahman, Laiba Abbas, Javeria Khurshid, Rizwan Ahmed, and Hafiz Durad. 2020. Design and Implementation of Secure Video Communication Over IP Using PYNQ-Z1 Board. *VFAST Transactions on Software Engineering*, 8(1), 8–13. DOI: 10.21015/vtse.v8i1.572.

[11] Magmio. 2025. FPGA-Based System for Ultra-Low Latency Trading. Retrieved from https://www.magmio.com/product.

[12] Lei Wang, Zeke Wang, and Hayden Kwok-Hay So. 2017. Exploring the Potential of Reconfigurable Platforms for Order Book Update: A Hardware-Friendly Algorithm for High-Frequency Trading. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1-8. DOI: 10.1109/FPL.2017.8056843.

[13] Xilinx. 2025. Accelerating High-Frequency Trading with Zynq-7000 SoC: Design Patterns and Optimization Techniques. Retrieved from https://www.xilinx.com/applications/financial/hft.html.

[14] A. Gupta and S. Bhardwaj. 2021. Hybrid ARM-FPGA Architecture for Microsecond Latency Trading Systems. In *2021 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. IEEE, 276–283. DOI: 10.1109/SAMOS52014.2021.9466225.

[15] R. Kumar, S. Patel, and Y. Li. 2022. BRAM-Efficient Order Book Design for FPGA-Based Trading Systems. In *2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 143–152. DOI: 10.1145/3490422.3507364.

1

[1]Used genAI to polish this final report