Milestone Report

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

Brandon Reponte, Leeza Gutierrez-Ramirez, Rudy Osuna

May 17, 2025

1 Feedback

- What is "competitive latency?" Try to be as precise as possible. Competitive latency is in the magnitude of microseconds. The system overall takes only a couple of tens of microseconds, and the transmission to the exchange can take a variable amount of time. We are primarily focusing on minimizing the tick-to-trade latency, which ignores the transmission time to the exchange.

- "I would mention the crypto aspect in the project overview.": While this is not as directly relevant to us now as we pivoted to U.S equities after consulting with Dr. Francesco, We will still go more into detail about the core strategy in our final overview. Our HFT algorithmic changes are listed below under Rudy's section.

- Work quickly to get a basic system and then optimize. It can take a lot of time to even get something basic working on FPGA. We have taken this into consideration and have focused our attention on other areas than Ethernet. Additionally, we have focused on getting a system to run on simulation at minimum, and deployment if necessary. High level logic has been implemented for the trading logic, and is undergoing translation into SystemVerilog code.

- Consider using the PYNQ software stack to make your prototype development move faster. Although we considered using PYNQ utilities such as the Jupyter notebook support and Vitis HLS for development, ultimately the goal is for speed and thus we focus more on the SystemVerilog as that is our final deliverable. We have already made considerable progress, so utilizing the PYNQ software stack would seem more of a step back at the current state of our project.

- MVP could be clearer. What is meant by functionally correct high frequency trading pipeline? Is this a complete system that gets data in real-time? Or is it a demonstration of the different components? Be as precise as possible. Our MVP is to implement a profitable high frequency trading pipeline. This is more specifically achieved by obtaining low latency through FPGA programming and correctness by proper implementation of trading strategies. For the sake of the MVP, we will be utilizing synthesized data and not real-time data for the sake of safer iterations in development.

- Group management could be more detailed. What happens if one person disagrees? What happens if one person is not doing their work? Etc. In the case of a disagreement, discussion will take place to undestand the varying perspectives of the problem. Additionally, we plan to consult mentors and experts to introduce clarity and additional information to arrive at a better judgement. In the case of someone lacking in their work, we plan to check up on progress regularly. If they end up not doing their work, we will meet with them and understand their situation and offer any support given a reasonable excuse. If there is no reasonable excuse

or if the situation worsens, we will consult with our mentor and instructors for advice on how to approach the situation.

- Testing plan that allows individual parts to progress without the others is needed. E.g., a testbench for the SV optimization pipeline before having the ethernet working. We have already planned to create several testbenches that cover the most important interactions. We plan to test the Ethernet reception and transmission, the UDP processing for both incoming and outgoing packets, and trading logic using the order book. More course-grained tests would include the entire pipeline with and without the Ethernet module.

- I did not see any milestone priorities. After understanding the project more and the necessities of the project, we have decided that our priority is to the high frequency trading logic with order book management first. Afterwards, processing the UDP packets takes priority over the Ethernet because information must be extracted from incoming packets and packaged correctly in outgoing packets. Ethernet interface development on FPGA has been strenuous because of the lack of documentation and information, but Ethernet on the system on chip is implemented by default and therefore the FPGA is just an optimization and not core to functionality as compared to the trading logic and the UDP processing.

2 MVP Progress

Our current MVP progress is an untested design of an Ethernet interface through the FPGA in SystemVerilog, a UDP packet module that is able to input and output the packets in SystemVerilog, and a trading logic module implemented in C#. These modules are isolated and have not yet been integrated with one another.

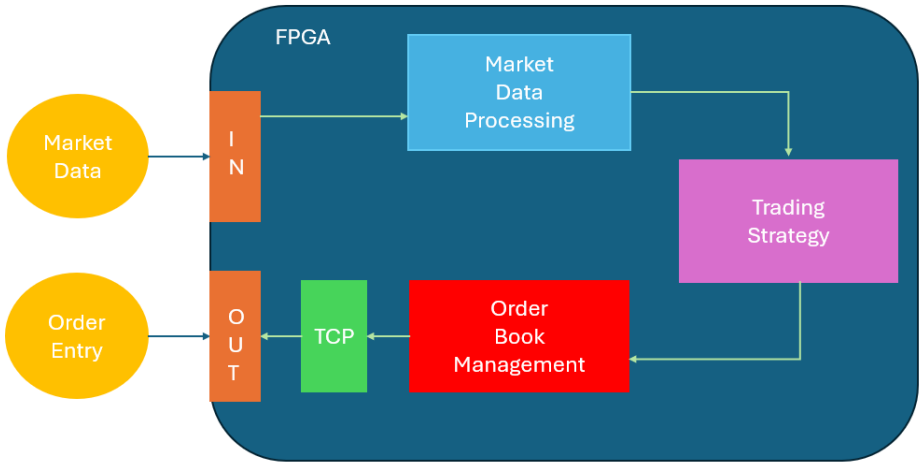
3 Milestone Completion

3.1 Literature Review

We have reviewed all the resources that Francesco provided and have planned an architecture according to existing literature. The architecture generally agreed upon in the literature is a market data processor module (which handles the UDP packet processing and extraction of information), the order book data structure and management, the trading strategy logic that utilizes the incoming packet data and the order book, UDP post processing in order to prepare the order decision from the trading strategy for transmission to the exchange, and the Ethernet interface module to handle both reception and transmission to and from the market exchange.

3.2 Architecture and RTL implementation

We have established a plan for the HFT pipeline:



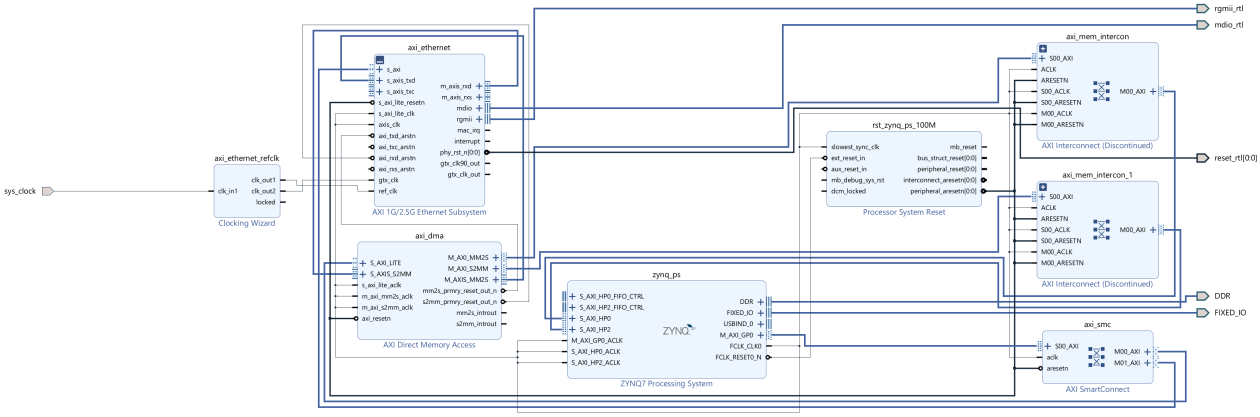
We have not created a structure for a testbench that measures tick-to-trade latency because we still do not know what to expect as input and output to the testbench since our Ethernet module has not yet been tested and therefore the connections to the testbench have not been determined.

Because we spent so much time on the Ethernet interface module, we were unable to make progress on the UDP packet data processing and the order book data structure.

Brandon researched tutorials and implemented a simple Ethernet interface module design using the IP catalog's ZYNQ PS, DMA block, and AXI Ethernet Subsystem block. The thought process is for the AXI Ethernet Subsystem to interface with the on-board PHY using its MAC. The PHY connects to the Ethernet port, and therefore there is a stream of data between the Ethernet port and the AXI Ethernet Subsystem block. This Ethernet block will process the data into an AXI stream to be read by the other blocks within the design. The DMA will handle the movement of the data received from the AXI Ethernet Subsystem block, and the ZYNQ PS will handle all the higher level control of the entire system.

Leeza had attempted integration and adaptation of the taxi Ethernet repository that Francesco had recommended. The taxi repository promised working Ethernet interfacing, but the Ethernet module was not designed for our specific board and adaptation was difficult because of the complexity of implemention and amount of dependencies for taxi's Ethernet module. The taxi repository was also missing the logic to process the UDP packets on FPGA. Therefore, Rudy attempted to use code from the implementation from the "blue-udp" repository and merged it with the taxi repository. The blue udp "provides modules for generating and parsing UDP/IP/Ethernet packets" and also offers co-simulation with cmac using vivado -(*https://github.com/datenlord/blue-udp*). This helps us bridge the AXI-Stream interface of the generated BlueUDP core (using the BlueSpec Compiler in WSL) to a generic AXI-Stream interface.

The AXI-Stream interface simulates how it would connect to an Ethernet MAC (from the taxi project).

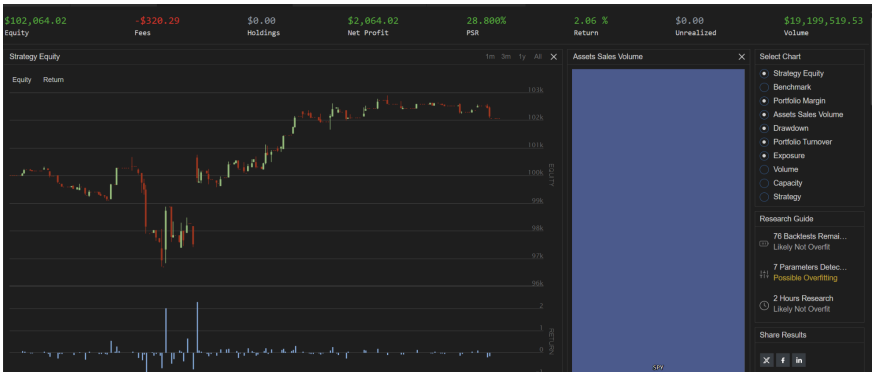


Rudy implemented the strategy originally in Python using the QuantConnect framework (see below). The strategy focuses on keeping the original order book imbalance (OBI) approach using Quote Bars (one bar for the bid and one bar for the ask in the order book). This strategy, although performing as intended, was not placing as many orders (taking as many trades) solely since we are limited to the data. With this in mind, our next iteration was written in a jupyter notebook using LOBSTER data provider's $SPY order book data - (*https://lobsterdata.com/info/DataSamples.php*). Since LOBSTER provides 50 depths of order book, this gave us a more accurate performance metrics of how this algorithm would perform on the board using Ethernet. This week, we will rewrite this logic in system verilog and attempt to get it working using this imported data from LOBSTER. This way, we can abstract away the work of the Ethernet connection for now while we test out the license and get the other components working.

(Image of core logic to enter and liquidate positions)

```python
# Order Book Imbalance: OBI(t) = (B(t) - A(t)) / (B(t) + A(t))
obi = (bid_vol - ask_vol) / (bid_vol + ask_vol)
self.Debug(f"Time: {self.Time}, OBI: {obi}")

# Long signal - OBI > 0.7
if obi > 0.9 and not self.is_long:
    if self.is_short:
        # Close short position first
        self.Liquidate()
        self.is_short = False

    # Enter long position
    self.SetHoldings(self.symbol, 0.95)  # Using 95% of portfolio
    self.is_long = True
    self.Debug(f"LONG signal at OBI: {obi}")

# Short signal - OBI < -0.7
elif obi < -0.9 and not self.is_short:
    if self.is_long:
        # Close long position first
        self.Liquidate()
        self.is_long = False

    # Enter short position
    self.SetHoldings(self.symbol, -0.95)
    self.is_short = True
    self.Debug(f"SHORT signal at OBI: {obi}")
```

(Image of perfomance)


$102,064.02
Equity
-$320.29
Fees
$0.00
Holdings
$2,064.02
Net Profit
28.800%
PSR
2.06 %
Return
$0.00
Unrealized
$19,199,519.53
Volume
Strategy Equity
1m 3m 1y All
Equity Return
103k
102k
101k
100k
99k
98k
97k
96k
EQUITY
2
1
0
-1
RETURN
Assets Sales Volume
Select Chart
Strategy Equity
Benchmark
Portfolio Margin
Assets Sales Volume
Drawdown
Portfolio Turnover
Exposure
Volume
Capacity
Strategy
Research Guide
76 Backtests Remai...
Likely Not Overfit
7 Parameters Detec...
Possible Overfitting
2 Hours Research
Likely Not Overfit
Share Results

4 Future Milestones

4.1 Architecture and RTL implementation

Because the difficulties with Ethernet set us far behind, our future milestones still include the implementation of all the modules of our architecture, and testing individual testing of them.

4.2 Design and Development of FAST Order Encoder

We are developing a modular FAST encoder to support both internal simulation and future network transmission of new order messages. The encoder is being designed to apply FAST operators (copy, delta, none), build presence maps dynamically, and write output to byte-aligned buffers. For simulation, messages will encoded into in-memory buffers and verified through manual decoding and debugging tools. In preparation for network transmission, we're designing a packet format with a custom header (including sequence number, timestamp, and message count), enabling batching, and integrating a low-latency UDP sender. Ongoing work includes validating encoding correctness and ensuring full compliance with the FAST binary specification.

4.3 Evaluate prototype design on simulation

According to our initial estimation, we planned to evaluate the design on simulation next week (5/18-5/24). With the current state of our progress, we will adjust this milestone by moving it to the following week (5/25-5/31) since we are still in the process of initially implementing all the modules, and have not yet started integration of the modules and individual testing of them.

4.4 Deployment on PYNQ board

Similarly, our initial estimation was to deploy the design on the PYNQ board, but progress has been slower than we assumed. As for our current goal for an MVP, deployment is not as necessary as we believed, so deployment will be a "good to have" but no longer a project deliverable. Specifically, we are adjusting this milestone to be no longer needed, but our next priority given that we complete all the previous milestones.

4.5 Optimization

Optimization has less priority than deployment, and thus is categorized likewise as a "good to have". Our initial assumption was to use SystemVerilog as our "optimization" and further optimize it by programming the SystemVerilog to best use the FPGA's parallelization. Our revised goal is to have a functional MVP that at least makes profit,

and optimization can come after deployment. Specifically, we are adjusting this milestone to be no longer needed, but only considered once deployment has been completed.

5 Rest of Quarter Timeline

	Deliverable:	5/18-5/24	5/25-5/31	6/1-6/7	6/8-6/14
Architecture and RTL implementation	CSV file logging tick-to-trade latency results.	Brandon	Brandon		
Development of FAST Order Encoder	Github Pull request with proof of testing	Leeza	Leeza		
Evaluate prototype design on simulation	Graph comparing different latencies and profits		Rudy		
Tragic Logic Signal generation	Demo showcasing the correctly computing buy and sell signals	Rudy	Rudy	Rudy	
System Verilog Logic	Translating Jupyter Notebook logic into AMD Vivado using System Verilog	Rudy	Rudy		
Simulating on AMD Vivado	Example simulation showing encoding of fabricated orders			Brandon & Leeza	Brandon & Leeza
Optimization	GitHub commits documenting refactored code				All