Project Specification

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

Brandon Reponte, Leeza Gutierrez-Ramirez, Rudy Osuna

May 31, 2025

1 Project Charter

1.1 Project Overview

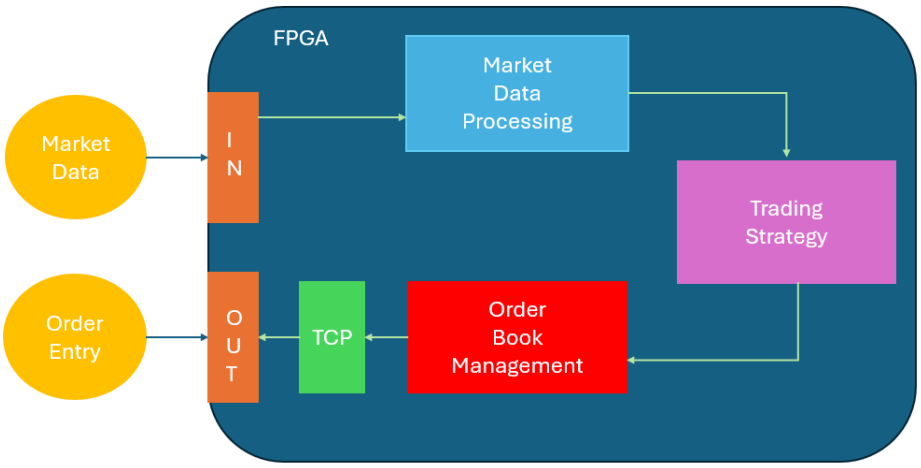
High frequency trading is the act of trading a large quantity of assets at a finer granularity using computational power. Typically, market competition uses technology that is unreachable to retail/non-enterprise traders, which allows them to achieve high computational power. However, this project proposes a relatively low cost solution for achieving competitive computational power and trading speed latency by using field programmable gate array (FPGA) hardware. This hardware allows for low-latency computation through parallelization of financial indicators and replacement of the NIC for the FPGA. The latter reduces latency by bypassing the kernel since we're moving the trading decisions as close as possible to the network.

This project will strive to utilize this low-latency feature of FPGAs for all parts of the high frequency trading pipeline, such as Ethernet reception of exchange data, data processing and interpretation, order book keeping of the state of the exchange, and trading strategy evaluation to make decisions on how to respond to changes in the exchange.

1.2 Project Approach

The goal is to implement a high frequency trading pipeline on an FPGA SoC connected by Ethernet that has competitive latency according to the market and the existing literature. In order to accomplish this goal, it can be split up according to a rough pipeline: Ethernet interface, market data processing, order book management, and trading strategy. We plan to first outline a pipeline diagram to flush out the interactions between the separate modules, as well as implement a skeleton of the testbench that will define the evaluation metrics. We intend to evaluate based on the tick-to-trade latency, which is the difference in time between when the high frequency trading pipeline first receives the Ethernet packet and the time it outputs its order back to the exchange. We will program our FPGA using SystemVerilog. We will connect the FPGA with our laptop over Ethernet, and have a script that sends market data to the FPGA through an Ethernet cable.

Diagram of the module dependencies which we dive into after:



For the Ethernet interface, we have discovered GitHub repositories that implement this feature on FPGAs, so the next step is to adjust the code to fit our specific needs or use it as a reference in our own implementation.

Concurrently, the market data processing module can be implemented based on the expectations of the data input outlined from our planning for the pipeline and module interactions. FOR US EQUITIES: Additionally, much literature suggests that the frame contents pertaining in the market is formatted according to the FAST UDP protocol, which attempts to reduce transmission latency by compressing certain areas of the frame. Our data processing module will process this data according to the protocol, and send the processed information to the trading strategy so that our pipeline can respond with an order.

Additionally, we will implement the order book as a hash map data structure, with the key being the asset and the value being the list of orders for that asset. For the sake of memory, we will use a price depth scheme for storing orders, which stores every order that has been done and the corresponding ask/bid volume of the order. For speed optimization, we will implement the order data structure using a 2D array, with one dimension representing buckets of ranges.

Lastly, we will explain the **core algorithmic trading strategy** that capitalizes on our computational advantages:

1.2.1 Trading Strategy

Lastly, we will implement the core trading strategy for this project. Assuming our reliable source of Quote data (we have performed some research and found a free live + historical data source), we will use a lightly modified version of the classic Order Book Imbalance strategy:

$$\text{OBI}(t) \;=\; \frac{B(t) - A(t)}{B(t) + A(t)}$$

Where:

$$\text{OBI}(t) > 0.7 \quad \Longrightarrow \quad \text{Long Signal}$$

$$\text{OBI}(t) < -0.7 \quad \Longrightarrow \quad \text{Short Signal}$$

Similarly, our sell conditions are:

$$\text{OBI}(t) > 0.7 \quad \Longrightarrow \quad \text{If Short At} < \text{-0.7}$$

Exiting position at OBI $< +0.7$; Reversal signal

$$\text{OBI}(t) < -0.7 \quad \Longrightarrow \quad \text{If Long At} > +0.7$$

Exit position at OBI < 0.7; Reversal signal

source of this idea (source uses $\pm\frac{1}{3}$ as the threshold but we will round to ± 0.7 for both simplicity and latency from our distance to exchange): https://osquant.com/papers/key-insights-limit-order-book/

This is a very simple strategy that capitalizes on order book arbitrage, essentially constantly computing imbalance in order book volume at best bid vs. ask. If there is a significant imbalance of ± 0.7 we will execute a MARKET order (immediate fill at best ask (for buy signals) or best bid (for sell signals)). An OBI of (< -0.7) shows dominant sell-side liquidity while An OBI of ($> +0.7$) shows dominant buy-side liquidity

This strategy offers opportunity for parallelism. The most intuitive approach to parallelize we will take on this MVP is Pipelining Parallelism. By Pipelining the Order Book Imbalance Strategy:

Parse UDP packet $\Longrightarrow$

Extract bid, ask (from depth5) $\Longrightarrow$

Compute B - A, B + A $\Longrightarrow$

Compute division $\Longrightarrow$

Compare against ± 0.7 threshold $\Longrightarrow$

Generate signal

We can process a new quote nearly every clock cycle

Overall, this algorithmic trading strategy MVP is simplistic enough to capitalize on the computational speedup of the PYNQ board FPGA while still allowing for parallelism. This gives us wiggle room to explore additional filtering and refinements while not over-promising in our MVP.

1.3 Minimum Viable Product

Our minimum viable product would be a functionally correct high frequency trading pipeline without any optimization strategies. This would more specifically be a high-level implementation in Vitis HLS that handles all components of the high frequency trading pipeline. Once we achieve a minimum viable product and gain feedback, we will begin to optimize by first incrementally transitioning the codebase into SystemVerilog. Afterwards, our future iterations would be optimizations on the several modules of the pipeline, such as translating from a hash map order book to the price depth order book mentioned and utilizing parallelization of the FPGA more in wiring.

Our objectives for this quarter is to create a high frequency trading pipeline that is competitive in terms of tick-to-trade latency with similar implementations in existing literature. Additionally, we would want to organize the code to be modular and able to increment with further optimizations in the future. Long-term goals of the project, beyond the current spring academic quarter, is to optimize the trading pipeline to be competitive with current market benchmarks.

1.4 Constraints, Risk, and Feasibility

A working high frequency trading pipeline that achieves near competitive tick-to-trade latency compared to existing literature would be realistically feasible, given potential stumbling blocks of adjusting to an unfamiliar toolchain in Vivado, unpredicted complexity of certain modules such as Ethernet interfacing, complexity of integration, and other unforeseen personal issues.

1.4.1 Risks and Measures

- Vivado familiarization - Work through several tutorials and videos on simple hardware in order to understand project workflow and capability of Vivado
- Ethernet interface - Utilize preexisting code that interfaces with the Ethernet on an FPGA
- Integration - Get a high-level implementation working and dedicate approximately a week to integration

2 Group Management

2.1 Major Roles

Due to our small group size of three, we plan to work as a democracy and will not have an established leadership.

2.2 Decision Making

Decisions are made by consensus. Since we are a team of only three members this is the most convenient method for decision making.

2.3 Communication

We will mainly communicate through the CSE 145 Discord chat channel for our group. Additionally, we will see each other during class. We also planned to meet on Tuesdays with Dr. Francesco Restuccia for guidance and to have weekly team meetings after o discuss our progress.

2.4 Scheduling

We will know when we are off schedule when we fall behind based on our Gantt chart and milestone deadlines. We plan to deal with schedule slips by prioritizing functionality first and then optimizations.

3 Project Development

3.1 Development Roles

Brandon and Leeza will be in charge of general FPGA and network programming, such as creating tests and implementing the several components of the pipeline. Rudy will primarily handle the trading strategy logic, but will also help in general programming when needed.

3.2 Technical Stack

3.2.1 Hardware

- PYNQ-Z2 FPGA board and PYNQ-Z1 FPGA board with systems on chip
- Two Ethernet cables

3.2.2 Software

- AMD Vivado
- Vitis HLS
- SystemVerilog
- GitHub

3.3 Testing

We will use a testbench to measure the tick-to-trade latency of our pipeline. Additionally, we will create testbenches that incrementally test each module. For example, for the Market Data Processing block we'll want to test that it is translating the bytes of the payload correctly, for the Trading Strategy block we'll test that it makes the decisions we expect based on fabricated data. Finally, we'll test that the Order Book Management Block encodes the order entries in the correct format as required by the Binance's data exchange.

3.4 Documentation

Push and pull from our code editor (most likely VSCode) to the Github repository. We also made sure to set up pull requests that require the approval of the other two members. Furthermore, the standard for the pull requests are specified in the repository.

4 Project Milestones and Schedule

4.1 Milestones

4.1.1 Literature Review (in Francesco's slides)

- Toolchain familiarization (AMD Vivado, simulation tools, open-source repo and tools)
- Establish version control (github repo, setup pull/push settings)

4.1.2 Architecture and RTL implmentation (Ethernet on FPGA)

- Implement testbench, measuring tick-to-trade latency
  - Simulated waveform screenshots showing testbench input/output
  - CSV file logging tick-to-trade latency results.
- Establish Architecture/HFT Pipeline
  - Block diagram of entire HFT pipeline
- Implement FPGA module for network processing
  - RTL code for network parser
  - Short video of module operating on FPGA
- Implement FPGA module for data processing
  - Testbench that feeds in mock data and verifies parsing correctness.
- Implement FPGA order book module
  - Example simulation showing encoding of fabricated orders
- Implement trading strategies
  - Example simulation showing encoding of fabricated orders

4.1.3 Evaluate the prototype design on simulation and on FPGA SoC

- Measure tick-to-trade latency and profit on recorded data
  - Graph comparing different latencies and profits
- Measure tick-to-trade latency and profit on live data
  - Graph comparing different latencies and profits

4.2 Optimization

- Parallize architecture to increase throughput and reduce latency
  - GitHub commits documenting refactored code

4.3 Milestone Progress Report

4.3.1 Brandon

4.3.2 Rudy

4.3.3 Leeza

Objective: To implement a FAST encoder capable of:

- Encoding new order messages into FAST format for internal simulation.
- Preparing encoded data for network transmission.

Steps:

- FAST Encoding Logic Develop a modular OrderEncoder class that:
  - Encodes order fields using FAST operators (copy, delta, none)
  - Builds presence maps dynamically
  - Writes encoded data to a byte buffer
- Internal Simulation Support
  - Encoded orders are written to in-memory buffers.
  - Verified encoding correctness with manual decoding, debugging output, comparison against expected field values.
- Network Transmission Preparation
  - Design a network-ready packet structure that:
    - Includes custom PacketHeader (sequence number, timestamp, message count).
    - Encodes multiple orders per packet (batching).

    * Prepared encoder to write into fixed-size aligned buffers.

    * Integrated UDP socket transmission using sendto() (low-latency path)

- Testing and Validation

  - Simulation testing: Encoded sample orders and validated field-level correctness.

  - Performance baseline: Initial encoding throughput benchmarks show [insert messages/sec if available].

  - Transmission dry run: Packets sent to a local UDP socket and verified via Wireshark.

- Current Challenges

  - Ensuring field encoding complies with FAST binary format spec.

Gantt Chart Tasks		4/13-4/19	4/20-4/26	4/27-5/3	5/4-5/10	5/11-5/17	5/18-5/24	5/25-5/31	6/1-6/7	6/8-6/14
Course Assignments:	Deliverable:									
Project Specification (4/18)	PDF document for Canvas	All								
Oral Project Update (5/6)	Written script of updates				All					
Milestone Report (5/14)	PDF document for Canvas					All				
Project Webpresence (6/3)	Website with our project overview and demos								All	
Final Oral Presentation (6/6)	PowerPoint with project results								All	
Final Project Video (6/9)	Video of us demoing our project									All
Final Report (6/13)	PDF document for Canvas									All
1. Literature Review (in Francesco's slides)										
- 1.1 Toolchain familiarization (AMD Vivado, simulation tools, open-source repo and tools)		All								
- 1.2 Establish version control (github repo, setup pull/push settings)		All								
2 Architecture and RTL implmentation (Ethernet on FPGA)										
- 2.1 Implement testbench, measuring tick-to-trade latency	Simulated waveform screenshots showing testbench input/output. CSV file logging tick-to-trade latency results.		Brandon							
- 2.2 Establish Architecture/HFT Pipeline	Block diagram of entire HFT pipeline		Brandon							
- 2.3 Implement FPGA module for network processing	RTL code for network parser Short video of module operating on FPGA			Leeza	Leeza					
- 2.4 Implement FPGA module for data processing	Testbench that feeds in mock data and verifies parsing correctness.			Brandon	Brandon					
- 2.5 Implement FPGA order book module	Example simulation showing encoding of fabricated orders			Leeza	Leeza					
- 2.6 Implement trading strategies	Video demo of module creating orders in real-time simulation.				Ryan	Ryan				
3. Evaluate the prototype design on simulation and on FPGA SoC										
- 3.1 Measure tick-to-trade latency and profit on recorded data	Graph comparing different latencies and profits						All			
- 3.2 Measure tick-to-trade latency and profit on live data	Graph comparing different latencies and profits						All			
4. Optimization										
- 4.1 Parallize architecture to increase throughput and reduce latency	GitHub commits documenting refactored code							All	All	