Dorm Room High-Frequency Trading

University of Florida

Concept Exploratory

Wayne Purtell

August 27, 2025

Contents

Executive Summary

High-Frequency Trading systems (HFTs) are complex, require expensive hardware, and need special data to feed them.

These issues can be remedied with the following:

1. Complexity issues by cutting the system into pieces
2. Resource issues by using lab scraps and the campus supercomputer
3. Data issues with some trusty Brownian motion to fill in the gaps

Motivations

Hard things pay *good* salaries.

But why would a bank want an engineer? One would be surprised.

Quant traders are a subset of traders that use advanced math and computing algorithms to power their decision making. But with what tools and data do they make these trades? Their trusty computer models and algorithms.
This of course opens the realm of low-latency systems in finance - most often engineered in C++. These systems are known to be insanely fast.

But do you know what's faster than software?
Hardware.

The stage is set: Designing ultra-fast systems that deal with high levels of precision and insane throughput rates. Seconds cost millions.

The Mission:

- Process data from the exchange into a local format as fast as possible
- Make decisions as fast as possible
- Send the decisions (buy/sell orders) back to the exchange (AFAP)
- Optional: Try not to mess up too much: Mistakes cost billions

Insane difficulty. Niche of the niche. Razorthin margins.
So how does one go about learning how these systems work?

We build one.

Problem Statement

Given the dominance of special firms that make use of this framework to improve their edge through specialized hardware, optimized networks, and low-latency algorithms, we must now find a way to mimic this infastructure.

While this setup may sound easy on paper, there is a significant barrier to entry for smaller entities including small firms, independent developers, and academic researchers like ourselves who lack expensive resources.

This presents three (3) key challenges:

1. **Technical Complexity** - High-Frequency Trading (HFT) platforms require highly specialized engineering skills across networking, FPGA design, and distrbuted systems. On their own these are already niche enough subjects, but when combined result in nothing short of a steep learning curve before even basic trading strategies can be protoyped.

2. **Resource Inequality** - Colocated servers, FPGA accelerators/boards, and expensive networking equipment and expensive networking equipment and expensive networking equipment and expensive enterprise-grade networking equipment limit participation in these markets to well-funded institutions - often excluding smaller players.

3. **Data Scarcity** - Getting ahold of generic stock-market data is easy enough with frameworks such as PolygonAPI and YahooFinance, yet sufficiently granular tick-level data is prohibitively expensive or outright unavailable as many providers provide delayed or aggregated feeds leaving sub-millisecond strategies nearly impossible.

As a result, innovation is slowed; and promising ideas from individuals or startups are often left unexplored. But do we really care?
No. We just want to prove we have the skills to run with the big leagues.
So it's ***hard***, it's ***expensive***, and premium gasoline is ***scarce***. What to do...

Approach

So how does one solve complexity? By cutting things up.

(4) Essential Components

1. An Exchange

We don't have a gas station. So let's make one.

Or at least something that behaves like one. We don't need real-time data; just enough data over a given timeframe that demonstrates our system can handle a measurable level of throughput. The key is that we need an absolute insane volume of data. We're essentially taking a very high-quality snapshot of the market - and μs tickers really start to add up.

We can just store this on a generic server PC - the only thing that matters here is the transmission speed at which it can export data. Getting the low-quality data can be done with anything, and filling in the μs gaps can be done with a sprinkle of Brownian motion can be done on our beloved supercomputer: HiPerGator.

2. A Network Card

We need some way to take an exchange protocol (like NASDAQ's ITCH) and parise it into a familiar data stream. This isn't as hard as it sounds.

For moving things between modules we will use the standard AXI-Stream dynamic. Their simple *valid && ready* scheme is effective. But what about between devices? We aren't about ease of use here; and with speed as a preference, ethernet (and potentially SFP) can easily achieve the speeds to fanout to our distributed system.

We have our faucet turned on and it's flowing fast. Time to connect the pipes to something meaningful.

3. Order Book Something has to hold the information on the trades from the exchange. By making use of an arbitrary tier system to tickers we arrive at (2) fundamental core variants.

1. A general dictionary-like core that can store buy/sell orders. By utilizing a hashing system we can store different tickers across multiple different instances of the core for a more scalable approach.
   Let's call this a **General Core**.

2. A specialized, high-speed core that is specific to a ticker. Because each core is it's own specific ticker, we don't have the latency propagated by the hash computations and can provide a more competitive edge on a subset of favored stocks.
   Let's call this a **Special Core**.

4. Software Interface

We have data from the exchange sitting in our cores. Now we need to do something with them. And to make decisions, we need ***rules***.

An example of a **rule** might be "buy AAPL if the price drops below 300$". Who makes up the rules? Not us. That's for the researchers. We're just here to give them somewhere to execute their strategies, and do it fast.

First of all we need to deploy these rules to our systems. The wonderful QSPI system can help us here. This will require some software-software interfacing to configure rules from software to bitsream but this shouldn't be too hard.

Another crucial use of our software interface will be logging and risk management. Logging things in pure hardware *hardly* makes sense, so keeping track of what orders we execute via software is the next best thing. On top of that, we need to manage our risk profiles. Hardware systems do exactly what you tell them to. It's just a matter of how fluent you are in the language of the universe. The FPGA doesn't care if we don't have the capital to wager

700million Norwegian Krone, but we can make it care.

Now the question is, how do we interface software and hardware this close as fast as possible? QSPI is useful for uploading our bitstreams, ethernet requirements PHY layers, and AXIS doesn't quite fit. PCIe is our solution. GPUs think it's good enough so it should be good enough for us.

The Staging Area Now comes the question of where we put everything. Dorm rooms come with a few neet things:

1. Fast *Wired* Internet
2. Uninterrupted Power Supplies
3. Water

Everyone has heard the story of the kid mining bitcoin in his dorm room. This is typically a result of the fact that dorms bundle together amenities and utilities into the yearly fee. They get caught because of the nosy RAs and the high sustained internet traffic Just kidding. It's because of the power. CPUs and GPUs (and some of the special mining ASICs) use alot of power - often in the realm of 100s of Ws per unit. FPGAs on the other hand use 1/10th of that power with common footprints using somewhere in the range of Ws to 10s of Ws per unit. The crucial fact is that the power cannot cut off. In my time in the dorms, I have yet to have my eternal PC ever powercycle or safe-mode boot.

In terms of internet, the actual *refining* of the data - inserting our extra data in between ticks - will occur on a server machine that is closer to it's hardware. Then, over the course of a short period, we will move that prepared data over to our "Exchange" machine in our room. The faster the internet the shorter (and if needed, more frequent) that transfer will take.

Another consideration for the machine that will be in the dorm room is **noise**. A light-sleeper of a roomate can make or break this. Thanks to my

rather heavy-sleeper roomate - let's call him Perkins - we won't have any issues with some rather tame, but constant fan noise. Thanks Perkins.

And last but not least is water. Our showers have water that we could *in theory* use for cooling. I sordidly expect cooling to be a bottle neck when it comes to our little makeshift server setup, but a clever mechanical engineer could be enlisted to construct a full fledged Roman aqueduct. *I have zero expectation for this to be utilized but it is a funny possibility to take this to the next level.*

Lab Area Because of the LAN quality of EduRoam, a lab on the same network would in theory have easy access to the dorm servers. Should hardware not be movable there for a remote execution environment one could easily be setup in (an available) lab space. A spare monitor and an active fan 1U server rack would do the trick.

List of Requirements

1. Spare Networking Hardware: Old NiCs and gigabit capable FPGAs
2. Lab Space: A small host PC system that can interface over PCIe for the software interface system
3. A Group to Title Under: a research group or lab by Which I can associate with