Commitment / Basic:

Team:

Each individual component works in isolation. Demonstrates functional correctness in simulation, in comparison to C++ or Python test script.

Individual components:

Ethernet/Parser:

- Capable of receiving structured message complying to Nasdaq ITCH protocol over Ethernet and grab relevant information Correctly.
- Supports outputs pertaining to add and cancel order messages.
- Correctness is verified by cross checking output over Ethernet.

Book building:

- Correctly supports adding, canceling orders, and market trades in simulation.
- State of the order book (price ladder per stock, how many quantities per order) are forwarded from simulation to a CSV file.
- Output from simulation (CSV file) will be compared against another script running on my laptop that is also fed the same data as the simulation, and make sure the order book structure matches with the simulation.

Trading logic:

- Has logic to update the covariance matrix and returns vector to update linear system to solve.
- Correctly solves a N by N linear system, subject to certain constraints (no fixed point overflow during the QR and upper triangular solve, bad condition number arising from fixed point error etc.) in simulation. Compare solutions to Python test bench.
- Correctly generates orders to submit over the Ethernet stack.

Goal:

Team:

System integration. Be able to feed live data from laptop to FPGA and get an order back. The order makes sense and is comparable to C++ system. Latency and performance measurements of the aggregate system are done by computing latency measurements from and to the FPGA on the laptop.

Individual components:

Parser:

- Capable of receiving structured message complying to Nasdaq ITCH protocol over Ethernet and grab relevant information Correctly.
- Supports outputs pertaining to add and cancel order messages.
- Capable of sending structured message complying to Nasdaq OTCH protocol over Ethernet by utilizing outputs from the Trading logic.
- Correctness is verified by cross checking output in over Ethernet.

Book building:

- Correct functionality when synthesized on FPGA.
- Interfaces with Parser module to get orders / cancels / trades from the market.

- Interfaces with trading module to expose best bid and ask prices for all the stocks.
- Periodically forwards state of the order book (price ladder per stock, how many quantities per order) to laptop through Ethernet so that it can be visually displayed on laptop.
- Optimize resource usage to fit with other modules (Parser, and Trading Logic).

Trading logic:

- Correct functionality when synthesized on FPGA.
- Correctly interfaces with the order book module, to take snapshots of the market state.
- Correctly interfaces with the Ethernet stack to actually send those orders over TCP.
- Optimizes resource usage so that it can fit alongside with the other modules.

Stretch goal:

Team:

System performance. Add latency measurements across the entire system (latency from getting an order to parsing it, from parsing to building the book, then from book to trade updates). Do resource measurements of our current system in terms of logic units used, how much memory used. Analyze what kind of throughout and latency our system can handle. Finally, Explore area/latency tradeoffs subject to resource constraints on lab hardware. Compare with state-of-the-art commercial solutions when normalized to resource usage.

Individual components:

Ethernet/Parser:

- Supports back-pressure from Book without dropping packets. This requires implementing and sizing a buffer space when there is bursts in the market, i.e, density relevant information momentarily increases due to sharp increase in market activity.
- Bypass trading logic operating on select information directly from the parsing module.
- Replace The Microblaze softcore IP used for abstracting TCP/IP protocol with light-weight ethernet stack. This will enable a tight pipeline from packet to parser
- Performance and latency measurement of the parser for a more precise latency measurement.

Book building:

The latency and memory usage of the order book is dependant on the number of stocks, how many distinct prices we support, and how many bits we allow for order ids. Analyzing the theoretical and empirical memory usage, and latency.
Explore an implementation of the order book where the orders for each price ladder is stored as linked list instead of arrays. This brings the cost of canceling an order in the data structure to O(1) from O(length of orders in price ladder). Will require implementing memory management (like malloc, free) on the FPGA, and implementing a double linked list on the FPGA. This will increase the memory usage on the FPGA, so there might be some tradeoffs to make in terms of resource usage. The higher memory usage might also increase the latency on BRAM.

Trading logic:

- Explore different designs of linear solver module (folded vs pipeline) and assess the area/latency tradeoff.
- Merge the rotator and arctan cordic IP modules in a custom cordic module. (Probably quite difficult, but described in literature.)

Time Line:

	Week 1 (Nov 4)	**Week 2 (Nov 11)**	**Week 3 (Nov 18)**	**Week 4 (Nov 25)**	**Week 5 (Dec 2)**
Ethernet/Parser	Microblaze IP creation and detailing its interaction with the parser.	Implement parser module with test bench.	debug parser module and start performance optimization.	Look into stretch goals. Explicit Integration.	Finish up implementing stretch goals, and performance optimizations.
Book building	implement order book software implementation, complete high level microarchitecture design with resource estimates.	implement first-pass verilog module with test bench.	debug order book verilog implementation, start looking at performance optimization.	Look into stretch goals. Explicit Integration.	Finish up implementing stretch goals, and performance optimizations.
Trading logic	Implement matrix inversion module and test bench in verilog to get resource estimate.	functionally debug matrix inversion module and get updated resource estimates.	implement the rest of the risk computations, trade updates, etc.	Look into stretch goals.Explicit Integration.	Finish up implementing stretch goals, and performance optimizations.

(Implicit Integration all throughout the project. This is done by having a main repo that has latest working bits of each of our projects integrated as a whole unit.)