FPGA BASED STOCK MARKET DATA FEED HANDLER

By

Sara Sehgal
Saksham Jairath
Neel Ghoshal

Design Document for ECE 445, Senior Design, Spring 2026
TA: Gerasimos Gerogiannis

4 March 2026
Project No. 76

1. Introduction

Modern electronic trading systems require deterministic, low-latency processing of high-rate market data streams, which general-purpose software systems cannot reliably guarantee under heavy load. This proposal outlines the design of a hardware-based market data feed handler using a custom Ethernet front-end PCB and FPGA logic to achieve predictable packet processing and ingress timestamping. The report will present the planned system architecture, PCB and FPGA design approach, and the methodology that will be used to evaluate functionality, throughput, and latency performance.

1.1 Problem

Electronic financial markets generate extremely high-rate streams of market data that must be processed with very low and predictable latency. Exchanges continuously broadcast order book updates, trades, and price changes to participants, and trading firms rely on this information to make time-sensitive decisions. As markets have become increasingly electronic and competitive, even microsecond-scale delays can materially affect execution quality and risk exposure. Software-based feed handlers running on general-purpose processors often experience nondeterministic delays due to caching, interrupts, and operating system scheduling, making it difficult to guarantee bounded latency under heavy load. This has motivated the use of hardware-accelerated and FPGA-based systems in industry to achieve deterministic, low-latency performance.

The importance of reliable and timely market data processing extends beyond individual trading firms. Modern financial markets support capital allocation, price discovery, and liquidity provision across global economies. If market data is delayed, corrupted, or inconsistently processed, it can contribute to pricing inefficiencies, increased volatility, or systemic instability. Events such as flash crashes have highlighted how tightly coupled, automated systems can amplify errors or latency imbalances, raising concerns about market fairness and resilience. Ensuring predictable and verifiable processing of market data is therefore linked to broader issues of financial stability, economic welfare, and trust in electronic markets.

From a societal and global perspective, financial markets underpin retirement systems, corporate financing, government debt markets, and international trade. The infrastructure that processes market data must therefore meet high standards of reliability and determinism to protect public welfare and economic stability. Hardware-based, deterministic processing architectures reduce the risk of data loss and unpredictable behavior during peak trading activity, contributing to safer and more robust financial systems. As global markets operate continuously across regions and time zones, improving the reliability and transparency of electronic trading infrastructure has economic and societal implications that extend well beyond individual firms.

1.2 Solution

The proposed solution is a fully hardware-based market data feed handler implemented on an FPGA, coupled with a custom-designed Ethernet front-end PCB. The system ingests live market data over a physical Ethernet connection, processes packetized trading updates in real time, maintains per-symbol top-of-book state, and generates low-latency trigger events when predefined trading conditions are met. By implementing the complete data path in hardware—from physical network ingress to trigger generation—the design eliminates operating system and software-induced jitter, enabling deterministic and bounded processing latency suitable for latency-sensitive trading applications.

At a high level, the custom PCB provides the physical-layer network interface, including an RJ45 connector with magnetics, Ethernet PHY, clocking, power conditioning, and protection circuitry. The PHY communicates with the FPGA over an RMII interface through the Pmod+ expansion connector. Inside the FPGA, a hardware Ethernet MAC receives frames and immediately captures an ingress timestamp at the

physical boundary. The packet payload is then passed through FIFO buffers into a finite-state machine that parses market data messages and extracts relevant fields. A trading state manager implemented using on-chip memory updates best bid and best ask values per symbol, and a trigger engine evaluates predefined conditions to generate output signals and diagnostic metrics. This architecture creates a deterministic, end-to-end hardware pipeline that closely mirrors real-world FPGA-based trading systems while remaining suitable for controlled academic implementation and evaluation.

1.3 Visual Aid

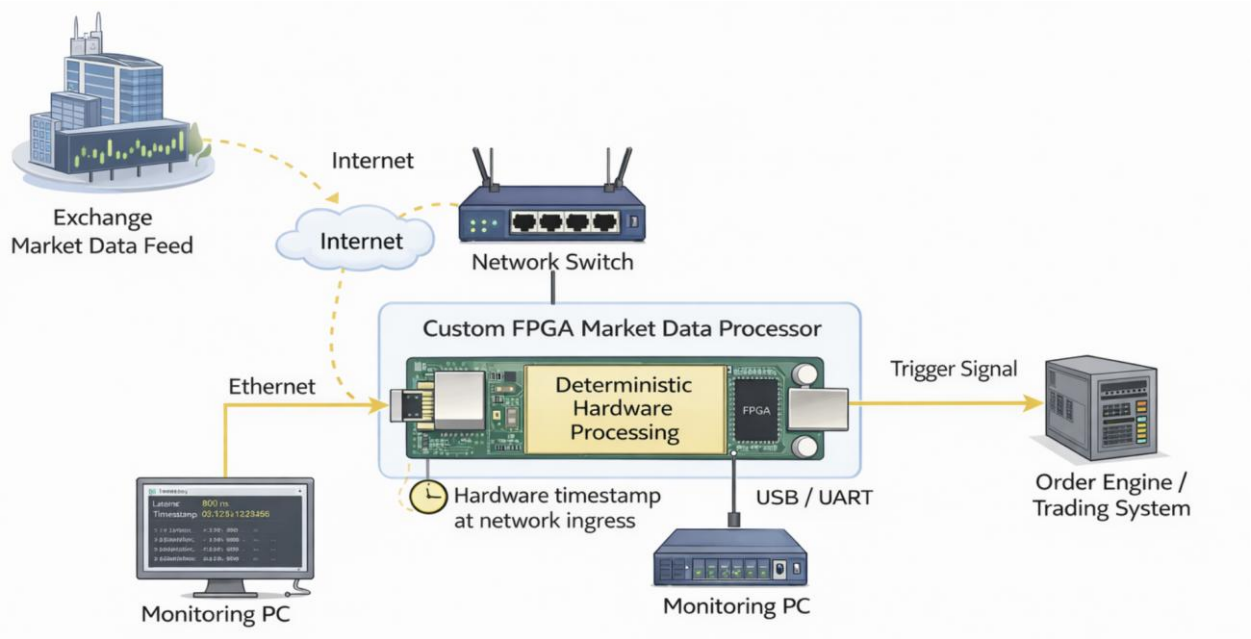

Figure 1: Contextual diagram showing the FPGA-based market data processor deployed between exchange market data infrastructure and a trading engine, with hardware-level deterministic processing and timestamped trigger generation.

High-Level Requirements

The following requirements define the essential characteristics that the system must achieve in order to successfully solve the problem described above.

- The system must receive and process Ethernet market data streams in real time using an FPGA-based hardware processing pipeline. Incoming packets must be accepted through the Ethernet interface and converted into a digital data stream that can be parsed by the FPGA logic.

- The system must correctly parse market data messages and extract key fields such as symbol identifier, price, and order size from the incoming data stream. These values must be captured and made available to the internal processing logic.

- The system must maintain an internal representation of market state for each processed symbol and update this state whenever new market data messages are received. This state information must be used to evaluate trigger conditions.

- The system must generate a hardware trigger signal when predefined conditions are satisfied by the processed market data. The trigger output must occur within a bounded and deterministic processing

latency after the relevant data is received.

Failure to meet any of these requirements would prevent the system from performing its intended function as a low-latency hardware-based market data processing device.

2 Design

2.1 Block Diagram

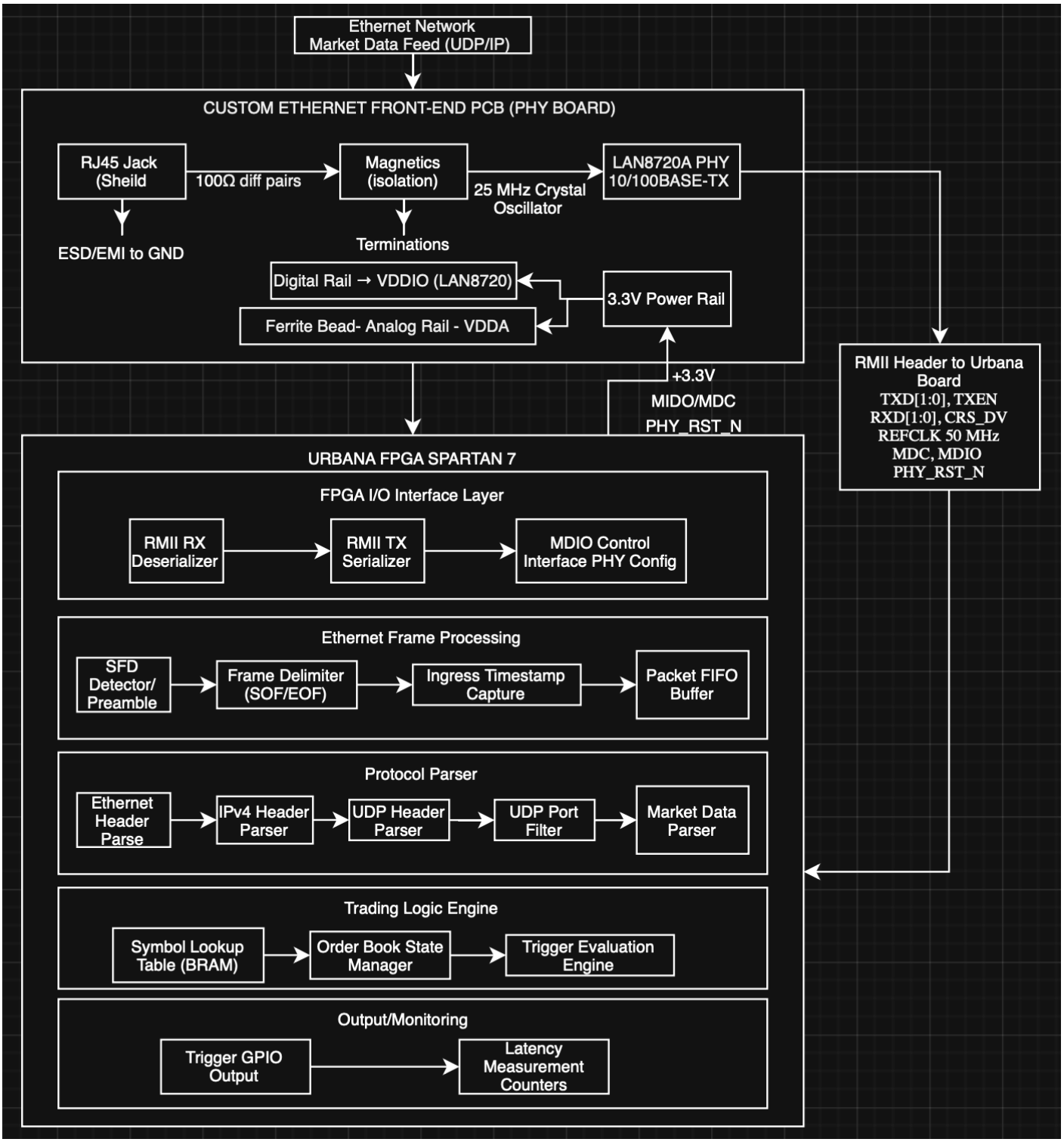

Figure 2: System level block diagram with detailed subsystems and requirements

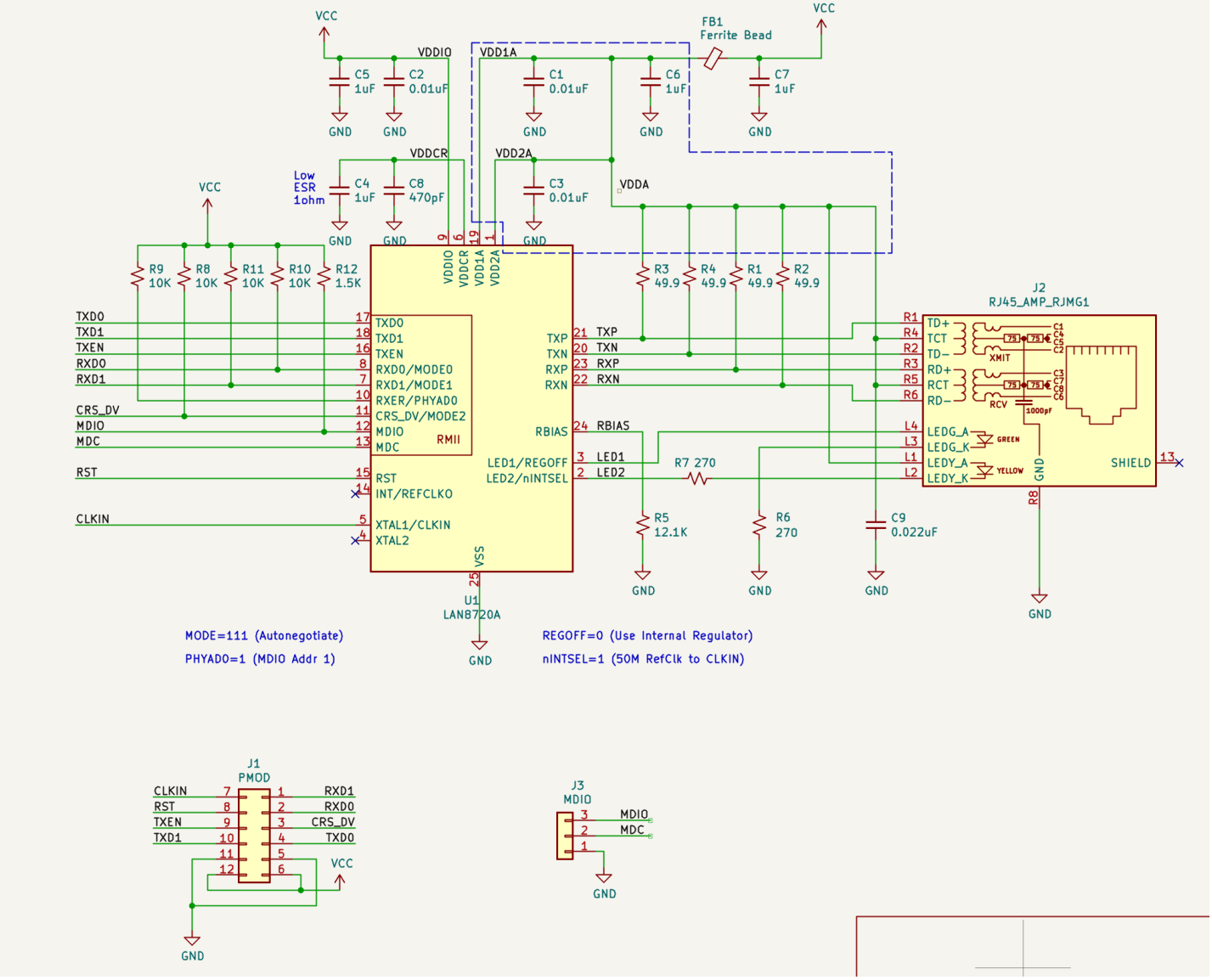

Figure 3: Electrical Schematic of the Custom PCB

2.2 Subsystem Overview

2.2.1 PCB Ethernet Front-End Subsystem

This subsystem provides the physical network entry point for market data. It accepts a standard Ethernet cable through an RJ45 connector (with magnetics), protects the interface against electrostatic discharge (ESD), and uses a 10/100 Ethernet PHY to convert the analog cable signaling into digital RMII signals. It connects to the FPGA subsystem through the Pmod+ connector using RMII data/control signals plus MDIO/MDC for configuration and a 50 MHz reference clock for synchronous operation.

2.2.2 PCB Power, Clock, and Debug Subsystem

This subsystem makes the PCB independently testable and ensures stable operation of the PHY. It distributes 3.3 V power (from the Pmod+ 3.3 V rail or an equivalent bench supply), filters and decouples the supply, provides a 50 MHz clock source for the PHY, and exposes link/activity LEDs and test points. It connects to the Ethernet PHY (power, clock, reset, LEDs) and indirectly supports the FPGA subsystem by ensuring the PHY is stable and observable during bring-up.

2.2.3 FPGA Ethernet MAC + RMII Interface Subsystem

This subsystem is the FPGA's hardware network interface to the PCB PHY. It receives RMII signals from the PHY, reconstructs Ethernet frames, and optionally transmits frames back through RMII. It provides a clean byte/packet stream to downstream logic and exposes MDIO/MDC transactions for PHY configuration and status monitoring. It connects upstream to the PCB via RMII/MDIO and downstream to the buffering subsystem via a streaming interface.

2.2.4 FPGA Ingress Timestamp Subsystem

This subsystem timestamps market-data arrival at the ingress boundary to measure deterministic latency. A free-running counter clocked by a stable FPGA clock latches a timestamp on the start-of-frame event (e.g., rising edge of `CRS_DV` or equivalent frame-start indication) before buffering and parsing. It connects to the MAC/RMII subsystem for the frame-start signal and to the FIFO/buffer subsystem by attaching a timestamp metadata word to each received frame.

2.2.5 FPGA FIFO / Buffer Subsystem

This subsystem absorbs bursty arrivals and prevents packet loss by buffering incoming frame bytes and metadata in BRAM-based FIFOs. It also serves as a clock-domain crossing boundary if the RMII side and processing side use different clocks. It connects upstream to the MAC/timestamp subsystems and downstream to the packet parser subsystem, ensuring continuous throughput and protecting the design's lossless requirement.

2.2.6 FPGA Packet Parser Subsystem

This subsystem converts raw packet payload bytes into structured market-data messages. Implemented as a finite state machine, it detects frame boundaries, validates basic formatting, and extracts fields (e.g., symbol ID, side, price, size, sequence number). It connects upstream to the FIFO/buffer subsystem and downstream to the trading state and trigger subsystems by outputting parsed message fields with a valid/ready handshake.

2.2.7 FPGA Trading State Manager Subsystem

This subsystem maintains per-symbol top-of-book state (best bid and best ask). It updates BRAM-resident state using parsed messages and provides the current top-of-book to the trigger engine. It connects upstream to the packet parser and downstream to the trigger engine, forming the core "market state" needed for trading-condition evaluation.

2.2.8 FPGA Trigger + Telemetry Output Subsystem

This subsystem evaluates predefined conditions on updated market state and produces low-latency trigger outputs. It also exports telemetry (timestamps, latency measurements, error counters) over UART and optionally asserts a GPIO trigger pin. It connects upstream to the trading state manager (state values) and timestamp subsystem (ingress timestamps) and provides outputs to external measurement/monitoring equipment.

2.3 Subsystem Requirements

Subsystem	Requirement	Verification Method
PCB Ethernet Front-End	The subsystem shall support **10/100BASE-TX Ethernet operation at 100 Mbps**.	Connect the board to a network switch and verify link establishment at **100 Mbps** using PHY status registers via MDIO and link LEDs.
PCB Ethernet Front-End	The subsystem shall provide **RMII interface signals (RXD[1:0], CRS_DV, TXD[1:0], TX_EN, REF_CLK = 50 MHz)** to the FPGA.	Probe RMII signals with an oscilloscope or logic analyzer and

		confirm correct toggling synchronized to the **50 MHz clock**.
PCB Ethernet Front-End	The PHY shall operate with **3.3 V logic compatible with FPGA I/O standards**.	Measure supply voltage using a multimeter and confirm voltage remains within **3.3 V ±5%** under operation.
PCB Ethernet Front-End	The subsystem shall include **Ethernet magnetics for signal isolation and impedance matching**.	Visual inspection of PCB components and continuity testing to confirm magnetics placement between RJ45 and PHY.
PCB Ethernet Front-End	The subsystem shall include **ESD protection on Ethernet lines**.	Verify presence of ESD protection components in schematic and PCB layout review.
PCB Ethernet Front-End	PHY configuration straps shall set **RMII mode and valid PHY address**.	Read PHY registers through MDIO after boot and confirm correct configuration values.
PCB Power / Clock / Debug	The subsystem shall supply **3.3 V ±5%** to the PHY.	Measure power rail with multimeter during operation.
PCB Power / Clock / Debug	The subsystem shall support **≥200 mA current capacity**.	Verify regulator specifications from datasheet and confirm no voltage droop under load.
PCB Power / Clock / Debug	The subsystem shall provide a **stable reference clock (50 MHz RMII)**.	Measure REF_CLK with oscilloscope and confirm **50 MHz frequency** and stable duty cycle.
PCB Power / Clock / Debug	The subsystem shall provide **deterministic reset behavior for the PHY**.	Observe reset line during power-up with oscilloscope and confirm delayed release after power stabilizes.
PCB Power / Clock / Debug	The subsystem shall include **link status indicators or MDIO readable status**.	Verify LED indication or read PHY status registers via MDIO.
FPGA Ethernet MAC / RMII Interface	The subsystem shall receive **Ethernet frames at 100 Mbps without packet loss**.	Send continuous packets from a test PC and verify all packets are received by the FPGA logic.
FPGA Ethernet MAC / RMII Interface	The subsystem shall detect **start-of-frame and end-of-frame boundaries**.	Use simulation and logic analyzer outputs to confirm frame boundary detection signals.
FPGA Ethernet MAC / RMII Interface	The subsystem shall support **MDIO communication with the PHY**.	Read PHY ID register and link status registers via MDIO interface.
FPGA Ethernet MAC / RMII Interface	The subsystem shall meet **FPGA timing closure at 50 MHz RMII interface**.	Check FPGA timing report after synthesis to confirm setup/hold timing constraints are met.
FPGA Ingress Timestamp Subsystem	The subsystem shall maintain a **free-running counter ≥100 MHz**.	Verify clock frequency in FPGA design and confirm counter increments every clock cycle.
FPGA Ingress Timestamp Subsystem	The subsystem shall timestamp **every received frame ingress event**.	Send test packets and verify each packet is assigned a timestamp in captured output logs.
FPGA Ingress Timestamp Subsystem	Timestamp width shall be **≥32 bits**.	Confirm counter bit-width in HDL implementation and simulation.
FPGA FIFO / Buffer Subsystem	FIFO shall buffer data to support **100 Mbps burst traffic without overflow**.	Stress test using continuous packet streams and monitor overflow flags.
FPGA FIFO / Buffer Subsystem	FIFO shall preserve **packet ordering and frame boundaries**.	Verify packet order and frame markers during simulation and hardware testing.
FPGA FIFO / Buffer Subsystem	FIFO shall expose **overflow/underflow debug indicators**.	Monitor debug signals during burst traffic tests.

FPGA Packet Parser	The subsystem shall parse packets with **100% correctness under test vectors**.	Use simulation with known packet test vectors and verify parsed fields match expected results.
FPGA Packet Parser	The subsystem shall detect **malformed packets and raise error flags**.	Inject corrupted packets in simulation and verify error flag assertion.
FPGA Packet Parser	Parser throughput shall support **100 Mbps input rate**.	Simulate maximum packet rate and verify no backpressure or packet drops.
FPGA Trading State Manager	The subsystem shall maintain **top-of-book state for at least N symbols (e.g., 256)**.	Verify BRAM storage and simulate updates for multiple symbols.
FPGA Trading State Manager	State updates shall occur within **bounded deterministic cycles**.	Measure update latency using simulation waveform timing.
FPGA Trading State Manager	The subsystem shall provide **deterministic state updates based on message sequence**.	Verify update logic through simulation with ordered and out-of-order inputs.
FPGA Trigger and Telemetry Output	The subsystem shall generate a **trigger signal within the defined latency bound (≤1 µs)**.	Measure time difference between ingress timestamp and trigger output using FPGA counters.
FPGA Trigger and Telemetry Output	The subsystem shall output **telemetry via UART**.	Connect UART to PC and confirm transmission of timestamp and latency values.
FPGA Trigger and Telemetry Output	Telemetry output shall **not stall the real-time processing pipeline**.	Stress test system while sending telemetry and confirm no packet drops occur.
FPGA Trigger and Telemetry Output	The subsystem shall provide an **external observable trigger signal**.	Measure trigger signal using oscilloscope or logic analyzer.

2.4 Tolerance Analysis

Risky aspect: The highest risk to successful completion is reliable **100 Mbps Ethernet reception** over the **RMII (Reduced Media Independent Interface)** connection between the Ethernet PHY on the custom PCB and the Urbana FPGA through the Pmod+ header. RMII runs a **50 MHz synchronous interface** with multiple single-ended signals (RXD[1:0], CRS_DV, TXD[1:0], TX_EN plus REF_CLK). If the **clock-to-data skew** and propagation delays are too large (from PCB routing, connector, and I/O buffer variation), the FPGA can sample incorrect bits, causing frame corruption and packet loss.

2.4.1 Feasibility via timing budget

RMII uses a **50 MHz clock**, so the clock period is:

- $T = \frac{1}{50\ \text{MHz}} = 20\ \text{ns}$

For correct sampling at the FPGA, the received data must meet:

- **Setup constraint:** data must be stable at least t_{setup} before the sampling clock edge
- **Hold constraint:** data must remain stable at least t_{hold} after the edge

Setup Time

A conservative worst-case setup budget can be written as:

$$t_{co(PHY)} + t_{pd(data)} + t_{skew} + t_{setup(FPGA)} < T$$

Where:

- $t_{co(PHY)}$: PHY clock-to-out delay (data launched relative to REF_CLK)
- $t_{pd(data)}$: PCB/connector propagation delay on data
- t_{skew}: mismatch between clock and data arrival (routing mismatch + buffer skew + jitter)
- $t_{setup(FPGA)}$: FPGA input setup requirement (including internal input path)

Plugging in conservative numbers

At 10/100 speeds, typical PHY and FPGA I/O delays are on the order of a few ns. Use conservative assumptions:

- $t_{co(PHY)} \approx 3$ ns (safe upper estimate for RMII output timing)
- PCB trace propagation: FR-4 signal speed ≈ **6 in/ns ⇒ 0.167 ns/in**
  - If traces are short (≤ 3 inches from PHY to Pmod+), then $t_{pd(data)} \leq 0.5$ ns
- Routing mismatch: if you length-match RMII lines within **1 inch**, skew from routing is:
  - $\Delta t_{route} \leq 0.167$ ns
- Add buffer/jitter margin:
  - $t_{skew} \approx 1.0$ ns (includes routing mismatch, connector variation, and clock jitter margin)
- FPGA setup requirement (conservative):
  - $t_{setup(FPGA)} \approx 2$ ns

$$3+0.5+1.0+2=6.5\text{ns}<20\text{ns}$$

Setup slack (margin):

$$20 - 6.5 = 13.5 \text{ ns}$$

This is a large margin at 50 MHz, indicating the interface is feasible if routing is kept short and reasonably matched.

<u>Hold Time</u>

Hold violations occur if data changes too soon after the clock edge at the receiver. A conservative hold condition is:

$$t_{co(PHY,min)} + t_{pd(data,min)} - t_{skew} > t_{hold(FPGA)}$$

Using safe values:

- $t_{co(PHY,min)} \approx 1$ ns
- $t_{pd(data,min)} \approx 0.2$ ns
- $t_{skew} \approx 1.0$ ns
- $t_{hold(FPGA)} \approx 0.5$ ns

$$1 + 0.2 - 1.0 = 0.2 \text{ ns} >/0.5 \text{ ns}$$

This shows **hold is the tighter risk** under worst-case skew assumptions, which is typical for synchronous interfaces.

Mitigation (design choices that fix hold risk)

To make hold robust in practice, the design will include at least one of the following (standard hardware practice):

1. **Clock/data co-routing and length matching**
   - Match RMII data lines to REF_CLK within **≤ 0.5 inch** (≤ 0.084 ns mismatch)
2. **Add small series resistors (22–33 Ω) near the PHY** on RMII lines
   - Reduces ringing/edge-rate issues that worsen effective sampling uncertainty
3. **FPGA input registering and timing constraints**
   - Register RMII inputs immediately on the REF_CLK domain at the I/O boundary and constrain timing accordingly (ensures tools optimize the input path)
4. **Keep the PHY close to the Pmod+ connector**
   - Minimize trace lengths so t_{pd} is small and predictable

With reasonable routing (short traces and matched lengths), t_{skew} drops substantially (e.g., from 1.0 ns to ~0.3–0.5 ns), and the hold inequality becomes satisfied with margin.

The RMII interface is feasible at **50 MHz** with significant setup margin. The main tolerance risk is **hold sensitivity to clock/data skew**, which is mitigated through controlled PCB routing (short, matched traces),

optional series damping resistors, and I/O-boundary registering in the FPGA. This analysis supports that a Pmod+-connected RMII PHY front-end can reliably sustain **100 Mbps** operation, enabling the project's throughput and deterministic-latency requirements.

3. Cost and Schedule

3.1 Cost Analysis

The cost of the proposed system consists of both labor costs and material costs required to design and implement the final hardware prototype. The parts cost includes only the components necessary for the final integrated system. Hardware purchased purely for development purposes, such as temporary FPGA development boards used for testing during early implementation, is not included in the final product cost.

Labor Cost

Labor costs are estimated based on an assumed starting salary equivalent to $25 per hour for an electrical or computer engineering graduate. Following the recommended cost estimation guidelines for engineering projects, the hourly rate is multiplied by 2.5 to account for additional employment costs such as benefits, overhead, and administrative expenses.

Each team member is expected to contribute approximately 120 hours toward the project. These hours include time spent on system design, FPGA development, simulation, hardware design, debugging, testing, and documentation.

The labor cost per team member is calculated as:

$$25\ (\$/\text{hour}) \times 2.5 \times 120\ \text{hours} = 7{,}500\ \text{USD}$$

Since the project team consists of three members, the total labor cost is:

$$7{,}500 \times 3 = 22{,}500\ \text{USD}$$

Therefore, the total estimated labor cost for the project is $22,500.

Parts Cost

The following components are required to build the final hardware system. These components include the FPGA device, networking interface hardware, printed circuit board, and supporting electronic components required for normal operation.

The estimated parts cost includes:

Item	Qty	Description	Notes	Price (USD)
Urbana Spartan-7 FPGA Board	1	FPGA development board	Main processing platform (Spartan-7)	$120.00

USB Programming Cable	1	USB cable for programming/power	Depends on Urbana connector type	$5.00
PMOD / Expansion Header	1	Pin header for FPGA connection	Only if not preinstalled on board	$1.00
PMOD Interconnect / Ribbon	1	Connection between PHY PCB and FPGA	Alternative to direct header connection	$3.00
Ethernet Cable (Cat5e/Cat6)	1	Network connection cable	Used for testing and feed input	$5.00
LAN8720A Ethernet PHY	1	10/100 Ethernet PHY transceiver	RMII interface, 3.3 V operation	$4.80
RJ45 Connector with Integrated Magnetics	1	Ethernet MagJack connector	Includes isolation transformers	$5.50
ESD Protection Diode Array	1	TVS protection for Ethernet lines	Protects PHY from electrostatic discharge	$0.60
25 MHz Crystal	1	PHY reference clock source	Used by LAN8720A PLL to generate 50 MHz RMII clock	$0.35
Crystal Load Capacitors	2	Capacitors for crystal oscillator	Typically 18–22 pF	$0.10
RBIAS Resistor (12.1 kΩ)	1	PHY bias reference resistor	Required for LAN8720A operation	$0.08
MDIO Pull-up Resistor (4.7 kΩ)	1	Pull-up resistor for MDIO line	Ensures stable management interface	$0.05
Reset Pull-up Resistor (10 kΩ)	1	Ensures defined PHY reset state	Prevents floating reset pin	$0.05
PHY Strap Resistors (10 kΩ)	5	Configuration strap resistors	Define PHY boot mode/address	$0.25
Ethernet Termination Resistors (49.9 Ω)	4	Differential pair termination	Used in PHY analog interface	$0.32
Ferrite Bead	1	Power rail filtering component	Separates analog and digital 3.3 V rails	$0.15
Decoupling Capacitors (0.1 µF)	8	Local power decoupling	Placed near PHY supply pins	$0.24
Bulk Capacitors (1–10 µF)	3	Supply stabilization capacitors	Smooths power supply noise	$0.30
High-Frequency Capacitors (0.01 µF)	2	Additional noise filtering	Improves supply stability	$0.06
Status LEDs	2	Link and activity indicators	Optional debugging indicators	$0.24
LED Current Limiting Resistors	2	Current control resistors	Typically 270 Ω–1 kΩ	$0.10
RMII Header Connector	1	Connector between PHY PCB and FPGA	Carries RMII signals	$0.50
Test Points	6	Debug measurement pads	Used for probing signals during testing	$0.30
Mounting Hardware	1 set	Standoffs and screws	For PCB mounting	$1.50

Total Parts Cost: $149.49

It should be noted that the labor cost reflects the engineering effort required to design and implement the system. In a commercial manufacturing scenario, this cost would be distributed across many units, resulting in a much lower per-unit cost.

3.2 Schedule

The project schedule is organized into several stages that correspond to the major phases of system development: architectural design, subsystem implementation, hardware design, system integration, and final validation.

The planned development schedule is outlined below:

Week	Phase	Key Tasks	Deliverables
1	System Architecture	Finalize system architecture; define FPGA subsystems; determine Ethernet data flow from PHY to FPGA; assign responsibilities and establish milestones	System architecture diagram; subsystem specifications
2	PCB Hardware Design	Select major hardware components (LAN8720 PHY, RJ45 with magnetics, regulators, clock); begin schematic design for Ethernet interface board	Initial schematic draft; component list
3	PCB Schematic Completion	Complete schematic design including power distribution, PHY connections, FPGA I/O interface, and debugging test points	Finalized schematic; preliminary BOM
4	PCB Layout	Perform PCB layout; route power, ground planes, and signal traces; ensure design rule compliance	PCB layout design; DRC verification
5	Fabrication Preparation	Final review of PCB design; generate fabrication files; submit PCB for manufacturing	Gerber files; fabrication submission
6	FPGA Communication Pipeline Design	Design Ethernet packet processing pipeline; define packet structure and parsing logic	Packet format specification; pipeline design
7	Frame Parser Implementation	Implement frame parser module to process incoming data stream and detect packet boundaries	Frame parser RTL; simulation results
8	Timestamping Subsystem	Implement ingress timestamp module to measure packet processing latency	Timestamp module RTL; testbench verification
9	Trading Logic Implementation	Implement trading state manager and trigger generation logic for event detection	State machine module; trigger logic
10	FPGA Core Integration	Integrate parser, timestamp, and trading logic modules; perform full system simulation	Integrated FPGA pipeline; simulation validation
11	Hardware Assembly and Integration	Receive fabricated PCB; assemble hardware; connect FPGA board to Ethernet PHY board; perform initial hardware bring-up	Assembled hardware system; basic connectivity tests
12	System Testing and Optimization	Validate packet processing functionality; measure system latency; optimize performance; prepare final demonstration and documentation	Final system validation; project demonstration

Work will be distributed among the three team members to allow parallel progress on FPGA logic development, hardware design, and system testing. This parallel development approach helps reduce project risk and ensures sufficient time for debugging and performance validation before the final demonstration.

4. Discussion of Societal Impact, Engineering Standards, Ethics, and Safety Considerations

Societal Impact

The proposed system contributes to the development of high-performance computing infrastructure used in modern financial markets. Electronic trading systems rely on extremely low-latency data processing in order to react to rapidly changing market conditions. By implementing a deterministic hardware pipeline capable of processing market data streams in real time, the project demonstrates techniques used in high-performance network processing and hardware acceleration.

Beyond financial applications, the techniques used in this system—such as hardware-accelerated packet parsing, deterministic processing pipelines, and FPGA-based networking—are widely applicable in other fields including telecommunications infrastructure, network security appliances, and high-frequency data acquisition systems. These technologies improve the performance and efficiency of data processing systems that support modern digital infrastructure.

Engineering Standards

Several engineering standards are relevant to the design and implementation of this system.

First, the networking interface is designed to operate according to the **IEEE 802.3 Ethernet standard**, which defines the physical and data link layer protocols for Ethernet communication. Compliance with this standard ensures interoperability with existing network infrastructure and hardware devices.

Second, the electrical design of the hardware system must follow standard electronic design practices including appropriate signal integrity considerations, voltage regulation, and PCB layout techniques. These practices help ensure reliable operation of high-speed digital circuits and reduce the likelihood of electromagnetic interference.

Additionally, the design process follows recommended engineering documentation and development practices to ensure that the system architecture is modular, verifiable, and maintainable.

Ethical Considerations

Engineers have an ethical responsibility to design systems that operate reliably, transparently, and safely. In accordance with the IEEE and ACM codes of ethics, the development of this project emphasizes accuracy, reliability, and responsible engineering practices.

Although the system processes financial market data, the project itself does not execute trading decisions or interact directly with financial markets. Instead, it serves as a research and educational demonstration of hardware-accelerated networking techniques. Care is taken to ensure that the system processes data accurately and does not introduce unintended behavior that could lead to incorrect outputs.

The design process also emphasizes proper attribution of all external resources, including reference materials and technical documentation used during the development of the system.

Safety Considerations

The hardware system operates entirely within standard low-voltage digital electronics ranges, typically between **1.0 V and 3.3 V**, which significantly reduces electrical safety risks. Nonetheless, several safety considerations must be addressed during development and testing.

First, proper handling procedures must be followed to avoid electrostatic discharge (ESD), which can damage sensitive semiconductor components such as the FPGA and Ethernet PHY. Team members will use grounded workstations and appropriate handling techniques when assembling and testing hardware.

Second, the power supply circuits must be carefully designed to ensure that voltage regulators provide stable and correct voltage levels to all components. Incorrect voltage levels could damage integrated circuits or cause unreliable operation.

Finally, all testing and debugging procedures will follow the laboratory safety guidelines established for the course. This includes using appropriate lab equipment, verifying connections before applying power, and monitoring system behavior during testing to prevent component damage.

Through careful design practices and adherence to established safety guidelines, the project minimizes risks to both developers and equipment while ensuring safe operation of the hardware prototype.

References

[1] IEEE Standards Association, *IEEE Standard for Ethernet*, IEEE Std 802.3, 2018.

[2] A. Putnam et al., “A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services,” *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, 2014.

[3] J. W. Lockwood, N. Naufel, and J. Turner, “Reconfigurable Network Processing on Field Programmable Gate Arrays,” *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2001.

[4] AMD/Xilinx, *Artix-7 FPGA Data Sheet: DC and AC Switching Characteristics*, DS181, 2023.

[5] Microchip Technology Inc., *LAN8720A Ethernet Transceiver Datasheet*, 2022.

[6] OpenAI, “ChatGPT,” OpenAI, 2026. [Online].