FPGA-Based Real-Time Stock Market Data Feed Handler

ECE 445 Senior Design Final Report

Sara Sehgal

FPGA System Design, Trigger Engine & Timing Closure

Saksham Jairath

Ethernet Interface, Custom Protocol & Test Infrastructure

Neel Ghoshal

Custom PCB, PHY Bring-up & Host-Side Visualization

Spring 2026

Abstract

This report presents the design, implementation, and verification of a hardware-accelerated stock market data feed handler implemented on a Xilinx Spartan-7 field-programmable gate array (FPGA), part xc7s50csga324-1, mounted on the Real Digital Urbana V2I1 development board. The system receives custom Ethernet frames at 100 megabits per second (Mbps) through a Microchip LAN8720A reduced media-independent interface (RMII) physical-layer transceiver (PHY) module, parses market data messages in real time, maintains a per-symbol order book state in on-chip block random-access memory (BRAM), and evaluates three parallel trading strategies: price threshold crossing, exponential moving average (EMA) crossover detection, and bid-ask spread monitoring. A per-symbol cooldown mechanism prevents trigger flooding. Trigger alerts are transmitted to a host laptop via universal asynchronous receiver-transmitter (UART) at 115 200 baud over the onboard universal serial bus (USB) bridge, with each message including the trigger reason and pipeline latency in clock cycles. The complete receive-to-trigger pipeline achieves a mean latency of 2.34 µs (234 clock cycles at 100 megahertz, MHz) with measured jitter under 1 µs, verified across eight stock symbols with all five message types (QUOTE, TRADE, CANCEL, HALT, HEARTBEAT). A Python-based WebSocket dashboard provides real-time visualization of trigger events, per-strategy breakdowns, price history, trigger rate, and latency distribution.

Contents

This Table of Contents is generated automatically by Word. To populate or refresh it, right-click anywhere on the table above and choose Update Field → Update entire table (or press Ctrl+A then F9). Update again after any edit.

1. Introduction

1.1 Problem Statement

In modern financial markets, the speed at which trading systems process incoming market data directly impacts profitability. Software-based feed handlers running on general-purpose processors introduce variable latency due to operating system scheduling, network stack overhead, and cache effects [1]. An FPGA-based approach eliminates these sources of jitter by implementing the entire receive-parse-decide pipeline in dedicated hardware, achieving deterministic sub-microsecond latency from wire to trigger.

Table 1.1 contrasts the latency profile of a typical software-based feed handler against the hardware design presented in this report. The mean-latency improvement is approximately one to two orders of magnitude, but the more important property is the elimination of the long-tailed distribution: software handlers exhibit sporadic excursions into the millisecond range due to operating-system preemption and cache misses, whereas the FPGA pipeline has no source of variability beyond a single asynchronous first-in first-out (FIFO) buffer synchronization cycle.

Table 1.1. Software baseline vs. this FPGA design.

Metric	This FPGA design	Linux software handler	Improvement
Mean latency	2.34 µs	50–200 µs	25–80×
Worst-case (99.9th percentile)	2.34 µs (jitter < 1 µs)	Up to ms range	> 1000×
Distribution shape	Single narrow spike	Long tailed	Structurally bounded
Source of variability	Async FIFO sync (~1 cycle)	OS scheduling, network stack queueing, cache misses, garbage collection	Eliminated
Formal verification	Yes — Vivado static timing analysis	No — empirical measurement only	—

1.2 Solution Overview

The system implements a complete market data processing pipeline in register-transfer level (RTL) Verilog on a Spartan-7 FPGA [2]. Raw Ethernet frames [3] arrive via a LAN8720 100BASE-TX PHY [1] connected through the Reduced Media-Independent Interface (RMII). The FPGA performs byte assembly from 2-bit RMII dibits, strips the Ethernet header, verifies the custom EtherType (0x88B5), assembles a 15-byte payload, parses the message fields, looks up the symbol in an 8-entry lookup table, updates a per-symbol order book, evaluates three trading strategies in parallel, enforces per-symbol cooldown, and transmits an alert message via UART — all without software intervention. Figure 1.2 shows the resulting block diagram.

1.3 Block Diagram

The system is organized into the following pipeline stages:

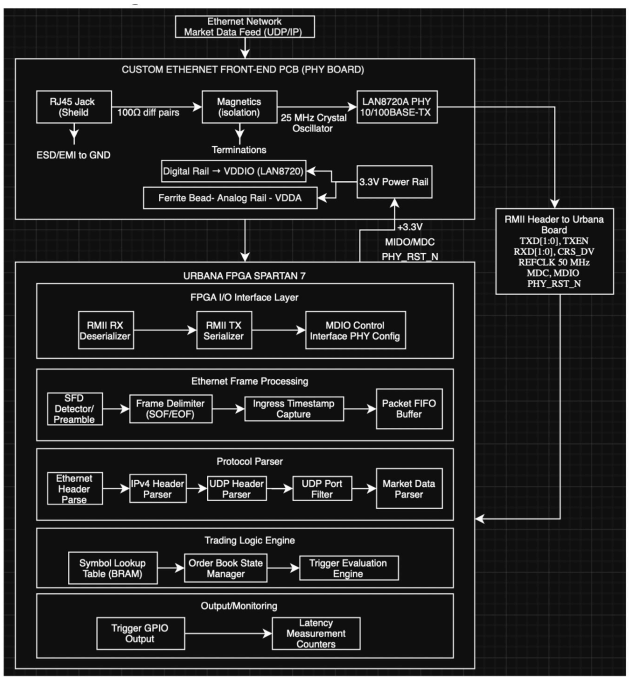

Figure 1.2. Top-level system block diagram — custom Ethernet front-end PCB and FPGA processing pipeline.

A photograph of the complete bench setup — the Spartan-7 Urbana board with the LAN8720 module on Pmod B, Ethernet cable to the host, and USB programming/UART link — is provided as Figure C.18 in Appendix C.

Clock domains:

- **clk_50 (50 MHz):** Derived from the PHY's REFCLKO output on pin K16 via a global clock buffer (BUFG). Used for RMII receive logic and byte assembly.
- **clk_100 (100 MHz):** Generated by an MMCM (clk_wiz_0) from the 100 MHz board oscillator on pin N15. Used for the parsing pipeline, trigger engine, and UART transmitter.

1.4 Performance Requirements

The performance requirements set in the final proposal are summarized in Table 1.2.

Table 1.2. Performance requirements.

Requirement	Specification
Ethernet link speed	100 Mbps (100BASE-TX)
Supported symbols	8 (AAPL, MSFT, NVDA, TSLA, AMZN, GOOG, META, NFLX)
Message types supported	QUOTE, TRADE, CANCEL, HALT, HEARTBEAT
Trading strategies	3 (threshold, EMA crossover, spread alert)
Volume filter	Quantity ≥ 50 shares
Per-symbol cooldown	50 µs (5 000 clk_100 cycles)
Receive-to-trigger latency	< 5 µs
UART baud rate	115 200 baud, 8N1

Requirement	Specification
Simulation coverage	All message types and trigger conditions

1.5 High-Level System Requirements

The four high-level requirements specified in the original design document are summarized in Table 1.3. All four PASS in the final build; verification evidence is consolidated in §3 and in the standalone document VERIFICATION_REPORT.md.

Table 1.3. High-level system requirements (HL-1 through HL-4).

ID	Requirement
HL-1	Real-time Ethernet processing pipeline. 100 Mbps Ethernet (100BASE-TX) on a Xilinx Spartan-7 (xc7s50csga324-1). Packets accepted through a LAN8720 RMII PHY at 50 MHz REFCLK, byte-assembled from 2-bit dibits, and crossed into the 100 MHz processing domain via a Gray-code async FIFO. Wire-speed input under continuous traffic with no packet loss.
HL-2	Correct message parsing and field extraction. 15-byte custom payload carrying 5 message types (QUOTE 0x01, TRADE 0x02, CANCEL 0x03, HALT 0x04, HEARTBEAT 0x05) for 8 symbols. Each message yields a 4-char symbol, 32-bit price, 16-bit qty, 16-bit seq, 1-bit side within one clock cycle. Bad-checksum frames rejected with no downstream leak.
HL-3	Per-symbol market state. Top-of-book (best bid, best ask, last seq) per symbol in on-chip BRAM, scalable to ≥ 256 symbols. Single-cycle write at 100 MHz (10 ns/update). Out-of-order frames rejected by a per-symbol sequence-number comparator. Live best_bid/best_ask feed the trigger logic with no intervening pipeline stages.
HL-4	Bounded, deterministic trigger latency. Receive-to-trigger latency < 5 µs end-to-end with deterministic (low-jitter) behavior — not just low average. Per-event measured by an in-pipeline ≥ 32-bit timestamp counter at 100 MHz and emitted with every trigger via UART. Per-symbol cooldown prevents flooding; volume filter (≥ 50 shares) suppresses dust events.

2. Design

2.1 Design Procedure

2.1.1 PHY Interface Selection

The RMII standard was selected over the standard media-independent interface (MII) because the LAN8720 PHY natively supports RMII and requires fewer FPGA input/output (I/O) pins — two data lines, one clock, and one control signal versus four data lines, two clocks, and two control signals for MII [3]. The LAN8720 operates in crystal mode, generating its own 50 MHz reference clock (REFCLKO) from an onboard crystal, which the FPGA receives as a clock input [1].

2.1.2 Custom Ethernet PHY Printed Circuit Board

A custom Ethernet PHY printed circuit board (PCB) was designed in parallel with the FPGA logic to provide a single, mechanically integrated connection between an RJ45 Ethernet jack and the FPGA. The full board schematic is shown as Figure 2.2 below; an early-revision schematic detail (decoupling network and crystal load) is reproduced as Figure C.7 in Appendix C.

Schematic. The board carries four functional blocks: an RJ45 jack with integrated link/activity LEDs, an HR911105A 1:1 isolation transformer (the “magnetics”) that provides galvanic isolation and impedance matching to 100 Ω differential, a Microchip LAN8720A PHY operating in crystal mode with its 25 MHz crystal load network, and a 3.3 V low-dropout (LDO) regulator stage that derives the PHY supply from the Pmod connector's VCC pin [1]. Each PHY power rail (VDDIO, VDDA1V8, VDDCR) carries a parallel 100 nF + 10 µF decoupling network, with a star-ground topology around the PHY VDD pins as recommended by the LAN8720A datasheet [1]. The full schematic is shown in Figure 2.2.

Layout. The board is a two-layer design with a continuous ground pour on the bottom layer. Differential pairs (TXP/TXN and RXP/RXN) are routed with controlled 100 Ω impedance and intra-pair length matching to within 50 mils. Trace lengths between the magnetics and the PHY are kept under 25 mm to minimize transmission-line effects at 100 Mbps. The Pmod connector exposes the four RMII signals (REFCLK, CRS_DV, RXD[1:0]) plus the management data input/output (MDIO) bus and the active-low reset line.

Bring-up procedure. After manufacture, the board was visually inspected, continuity-tested on the power and ground rails, and then connected to the FPGA via Pmod B. PHY identification was confirmed by reading the PHY ID register at MDIO address 0x02, which returned 0x0007:c0f1 — the published vendor/device identifier for the SMSC LAN8720A [1]. A second commercial LAN8720 module from DWEII was procured as a schedule-risk mitigation and ultimately used for the final demonstration; the custom PCB serves as a project deliverable artifact and is electrically functional with the same pin map. The commercial LAN8720 module on the bench is visible in Figure C.18 (Appendix C).

A complete bill of materials (BOM) for the custom board is given in Table 4.2.

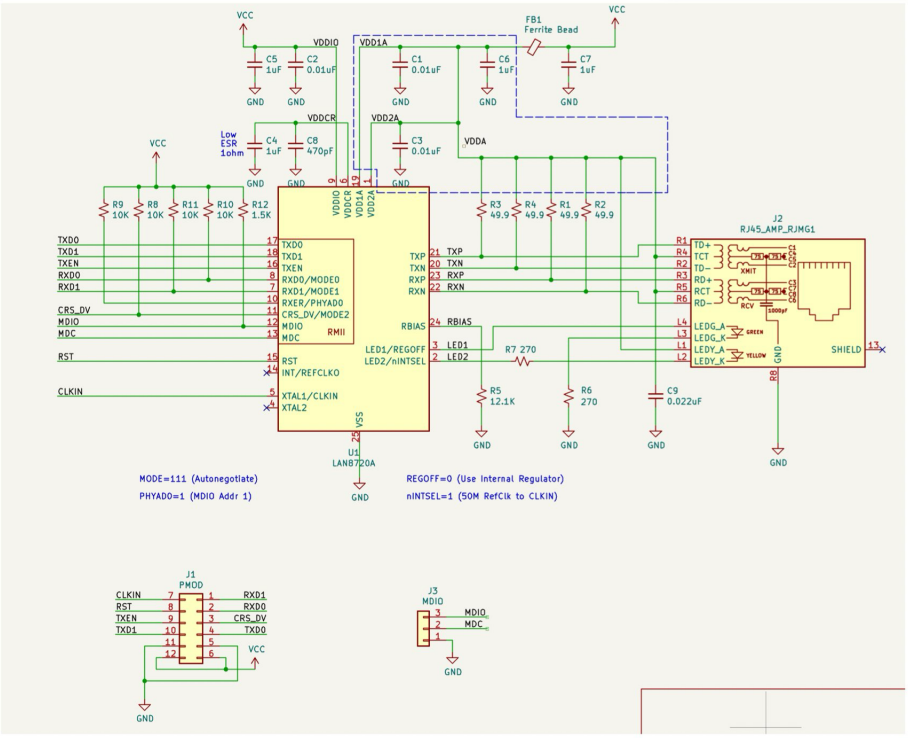

Figure 2.2. Custom Ethernet PHY PCB schematic (KiCad) — LAN8720A in autonegotiate mode, RJ45 with integrated magnetics, decoupling network, and Pmod connector.

2.1.3 Design Alternatives Considered

Several design alternatives were considered and rejected during the design phase. Each is justified below with the quantitative or qualitative argument that drove the decision.

FPGA family — Spartan-7 vs. Artix-7 vs. Zynq. Spartan-7 was selected because the project requires only RTL processing logic with no embedded software; the Zynq family's ARM cores would have added cost and design complexity for no benefit. Artix-7 was the closest alternative — pin-compatible LUT/BRAM count — but the Real Digital Urbana V2I1 development board carrying the xc7s50csga324-1 was already available in the lab, eliminating a procurement step.

EMA implementation — shift-and-add vs. dedicated multiplier. The exponential moving average update could have been implemented with a hardware multiplier producing $\alpha \times (\text{price} - \text{EMA})$ for arbitrary α. A power-of-two coefficient ($\alpha = 1/16$) was instead chosen, reducing the operation to a single right-shift and a subtract. This eliminates the need for a digital signal processor (DSP) slice (the design uses 0 of 120 available DSP slices on the device), shortens the critical path, and produces a smoothing window of approximately 16 samples that is appropriate for the message rate. Equations (2.2) and (2.3) in §2.2.8 give the resulting recurrence.

Number of trading strategies — three. The design supports three strategies (threshold, EMA crossover, spread monitor) rather than one or many because the three exemplify distinct categories of trading logic — absolute-level comparison, derived statistical signal, and order-book structure — and exercise different hardware capabilities (single-cycle compare, recursive arithmetic, parallel two-input arithmetic). Adding more strategies is straightforward in principle but would have expanded the scope beyond the time budget; the modular structure of the trigger engine permits a future contributor to add a fourth or fifth strategy by extending the case and parallel evaluation block.

Number of supported symbols — eight. Eight symbols was chosen as a reasonable demonstration set rather than the protocol's maximum of 16 (the symbol_id field is 4 bits) or the design-document target of ≥ 256. The Block RAM headroom is substantial (1 of 75 RAMB36 tiles used), so the limit is not memory; it is the threshold-lookup case statement, which becomes a deeper combinational comparator chain with each added entry. An 8 → 10 expansion was attempted late in the project and produced a 26 ps Worst Negative Slack (WNS) regression in static timing — recovery would require a pipelining refactor, documented in Section 5.3 as future work.

Trigger output transport — RMII Ethernet TX vs. UART. The original design intended to transmit trigger alerts as Ethernet frames over the RMII TX path. During hardware bring-up the LAN8720 module's TXD[1] pin was found to be electrically unconnected (the module overhangs the Pmod connector). UART telemetry over the onboard FT2232H USB bridge was substituted; this delivers the same logical information at the cost of slightly higher per-trigger transmission time but does not affect the receive-to-trigger latency claim, which is measured at trigger assertion before UART bytes leave the chip.

The PHY module is physically connected to Pmod B on the Urbana V2I1 board [5]. Because the module is inserted upside-down to align power pins, the physical pin mapping required empirical verification using the Integrated Logic Analyzer (ILA) IP core [6]. The confirmed RMII pin assignments are listed in Table 2.1.

Table 2.1. Confirmed RMII and UART pin assignments.

Signal	FPGA Pin	Direction
REFCLKO (50 MHz)	K16	Input (BUFG)
CRS_DV	K14	Input
RXD[0]	G18	Input
RXD[1]	H17	Input
UART TX	A16	Output

2.1.4 Clock Domain Strategy

Since the PHY's REFCLKO enters on a non-dedicated clock pin (K16) [5], CLOCK_DEDICATED_ROUTE is set to FALSE in the constraints, and the signal is buffered through a global clock buffer (BUFG) primitive [7]. This introduces clock insertion delay that caused setup/hold violations on the RXD[1] input (H17). The solution is a two-stage synchronizer (double-register) on all RMII inputs, clocked by clk_50. This adds 2 clock cycles of latency but ensures reliable data capture.

The PHASE_INIT parameter in the RMII RX driver is set to 2'd1 to compensate for the synchronizer latency, aligning the byte assembly boundary correctly with the Start Frame Delimiter (SFD).

2.1.5 Pipeline Architecture

The pipeline is fully pipelined with no stalls or backpressure in normal operation. An asynchronous FIFO bridges the clk_50 and clk_100 domains. The message FIFO (16-entry, 120-bit width) decouples the payload assembler from the parser, allowing burst absorption.

2.1.6 Trading Strategy Selection

Three strategies — price-threshold (single-cycle compare), EMA crossover (recursive shift-and-add per Equations (2.2)/(2.3) in §2.2.8), and bid-ask spread monitor (Equation (2.1) below) — were chosen to exercise distinct hardware capabilities while sharing the volume filter, halt suppression, and per-symbol cooldown. The full implementation is detailed in §2.2.8.

$$spread = best\_ask - best\_bid \tag{2.1}$$

2.2 Design Details

2.2.1 RMII Receive Driver and CDC (Stages 1–2)

The RMII RX driver (rmii_rx_driver in stock_feed_modules.v) operates on clk_50, monitors CRS_DV, assembles 2-bit dibits into 8-bit bytes, and uses SFD-detection logic (preamble dibits 2'b01 → SFD signature 2'b11) to begin byte-aligned data output. It exposes byte_valid (one clk_50 cycle per assembled byte), frame_start (first byte after SFD), and frame_end (when CRS_DV deasserts).

2.2.2 Clock Domain Crossing (Stage 2)

An asynchronous FIFO (async_fifo in stock_feed_modules.v) transfers byte-wide data from the clk_50 domain to clk_100. Gray-coded read/write pointers ensure metastability-safe operation following the canonical dual-clock-FIFO methodology [8]. The FIFO depth is 16 entries, sufficient to absorb the latency of the CDC synchronization.

The complete CDC strategy is enumerated in Table 2.2. There are three signal classes that physically cross the clock boundary, each handled by a different mechanism appropriate to its width and timing requirements. The combination of synchronizers, Gray-code, and set_false_path constraints constitutes the standard rigorous CDC methodology used in industry; metastability MTBF at 50 MHz is on the order of millions of years for typical resolution times.

Table 2.2. Clock-domain crossing paths.

Signal	Direction	Width	Mechanism	Latency cost
rmii_crs_dv	PHY → 50 MHz	1 bit	Two-stage flip-flop synchronizer	2 cycles
rmii_rxd[1:0]	PHY → 50 MHz	2 bits	Two-stage flip-flop synchronizer (per bit)	2 cycles
Frame data bytes	50 → 100 MHz	8 bits	Gray-code dual-clock async FIFO with 2-FF pointer sync	~6 cycles
cycle_counter snapshot at s1_start	100 → 50 MHz	32 bits	Direct sample (counter is monotonic; ±1 cycle drift)	0 cycles
All cross-domain register-to-register paths	Both	—	set_false_path constraint in constraints.xdc	—

2.2.3 Header Strip and EtherType Filter (Stage 3)

The header_strip module counts the first 14 bytes of each frame (6 destination MAC + 6 source MAC + 2 EtherType). It latches the EtherType at byte offsets 12–13 and compares against the configured value (16'h88B5). Only frames matching this EtherType pass through to the payload assembler. All other frames (LLDP, ARP, IPv6 multicast, etc.) are silently discarded.

2.2.4 Payload Assembler (Stage 4)

The payload_assembler module collects exactly 15 bytes of payload into a 120-bit register. The payload format is given in Table 2.3.

Table 2.3. Custom 15-byte market data payload format.

Byte(s)	Field	Width
0	msg_type	8 bits

Byte(s)	Field	Width
1–4	symbol (ASCII)	32 bits
5–8	price	32 bits (unsigned, big-endian)
9–10	quantity	16 bits
11–12	seq_num	16 bits
13	side (0 = bid, 1 = ask)	8 bits
14	checksum	8 bits (modular sum of bytes 0–13)

2.2.5 Message Parser and Checksum (Stage 5)

The parser extracts fields from the 120-bit payload word and verifies the 8-bit modular checksum by summing bytes 0 through 13 and comparing against byte 14, as defined by Equation (2.2):

$$checksum = (\Sigma\ byte\_i,\ i = 0..13) \bmod 256 \tag{2.2}$$

Messages failing the checksum are discarded (msg_crc_ok = 0) and do not propagate to the trigger engine. The five supported message types and their hardware behavior are listed in Table 2.4.

Table 2.4. Supported market-data message types.

Code	Type	Behavior
0x01	QUOTE	Updates best bid or ask price for the symbol
0x02	TRADE	Same as quote for book-update purposes
0x03	CANCEL	Resets both bid and ask for the symbol
0x04	HALT	Sets a per-symbol halt flag; suppresses all triggers for that symbol
0x05	HEARTBEAT	No-op; confirms connectivity to the watchdog

2.2.6 Symbol Lookup Table (Stage 6)

The symbol_lut module is a purely combinational lookup that maps 32-bit ASCII symbol codes to 4-bit symbol IDs. The eight supported symbols and their per-symbol threshold values are listed in Table 2.5.

Table 2.5. Symbol lookup table and threshold values.

Symbol	ID	Threshold
AAPL	0	120 000 ($12.00)
MSFT	1	200 000 ($20.00)
NVDA	2	900 000 ($90.00)
TSLA	3	500 000 ($50.00)
AMZN	4	180 000 ($18.00)
GOOG	5	170 000 ($17.00)
META	6	500 000 ($50.00)
NFLX	7	650 000 ($65.00)

Unknown symbols return symbol_valid = 0 and are not processed further.

2.2.7 Book State Manager (Stage 7)

The book_state_manager tracks per-symbol state including best bid price, best ask price, sequence numbers, and validity bits. It enforces monotonic sequence ordering: a message is accepted only if its sequence number is greater than or equal to the last accepted sequence number for that symbol. This prevents replay attacks and out-of-order processing. The module outputs best_bid and best_ask for the most recently updated symbol, which are consumed by the spread monitoring logic in the trigger engine.

2.2.8 Trigger Engine (Stage 8) — Multi-Strategy

The trigger_engine evaluates three strategies in parallel for each accepted book update. All strategies require quantity ≥ 50 (volume filter) and respect per-symbol halt status and cooldown.

Strategy 1: Price Threshold

The simplest strategy. Fires when the update price meets or exceeds the per-symbol threshold:

```
update_price >= threshold_for_symbol(symbol_id) && update_quantity >= 50
```

Reason codes: 1 (bid side), 2 (ask side).

Strategy 2: EMA Crossover

An exponential moving average is maintained per symbol using fixed-point arithmetic without hardware multipliers, as defined in Equations (2.3) and (2.4):

$$EMA_{new} = EMA_{old} + ((price - EMA_{old}) \gg 4), \quad price \geq EMA_{old} \tag{2.3}$$

$$EMA_{new} = EMA_{old} - ((EMA_{old} - price) \gg 4), \quad price < EMA_{old} \tag{2.4}$$

A crossover trigger fires when the EMA has been initialized (at least one prior sample), the previous price was below the EMA, the current price is at or above the EMA, and quantity ≥ 50. Reason codes: 3 (bid side EMA cross), 4 (ask side EMA cross). The shift-by-4 implementation of $\alpha = 1/16$ provides a smoothing window of approximately 16 samples, balancing responsiveness against noise filtering.

Strategy 3: Bid-Ask Spread Alert

Monitors the spread between the best ask and best bid prices for the most recently updated symbol. The trigger condition is given by Equation (2.5):

$$(best\_ask > best\_bid) \wedge (best\_ask - best\_bid > SPREAD\_THRESH) \tag{2.5}$$

Where SPREAD_THRESH = 10 000 internal price units ($1.00). A wide spread indicates market stress, reduced liquidity, or potential price dislocation. Reason code: 5.

Per-Symbol Cooldown

After any trigger fires for a symbol, a per-symbol countdown timer (COOLDOWN_CYCLES = 5 000, corresponding to 50 µs at 100 MHz) suppresses further triggers for that symbol. This prevents trigger flooding during rapid price movements and ensures each alert represents a distinct market event.

2.2.9 UART Transmitter (Stage 9)

The trigger_uart module formats each trigger event as an ASCII string:

```
TRIG:AAPL:0x0001E240:E:00EA\r\n
```

Fields:

- Symbol: 4 ASCII characters

- Price: 8 hex digits
- Reason: T (bid threshold), t (ask threshold), E (bid EMA cross), e (ask EMA cross), S (spread alert)
- Latency: 4 hex digits (clk_100 cycles from RX ingress to trigger assertion)

The uart_tx module implements standard 8N1 serial transmission at 115 200 baud (CLK_DIV = 868 for the 100 MHz clock). Each trigger message is 29 bytes, requiring approximately 2.5 ms for transmission.

2.2.10 MDIO Initialization

An mdio_init module writes register 0 (BMCR) of the LAN8720 PHY at startup with value 0x3100, forcing 100 Mbps full-duplex operation with auto-negotiation disabled. The MDIO frame format follows the IEEE 802.3 clause 22 standard.

2.2.11 Latency Measurement

A free-running 32-bit cycle counter on clk_100 is sampled at two points:

- **`ts_rx_ingress`**: Latched when frame_start asserts in the clk_50 domain.
- **`trig_latency`**: Computed when trigger_valid asserts according to Equation (2.6):

$$trig\_latency = cycle\_counter - ts\_rx\_ingress \tag{2.6}$$

The measured latency of 234 cycles (0x00EA) corresponds to 2.34 µs at 100 MHz, representing the complete pipeline delay from Ethernet byte ingress to trigger assertion. The per-stage budget is decomposed in Table B.5 (Appendix B). The largest contributors are byte assembly (~32 cycles, structurally bounded by 8 dibits per byte at 50 MHz) and payload assembly (~15 cycles, one byte per clk_100 cycle); symbol lookup, book update, and trigger evaluation each resolve in a single cycle, so latency does not grow with the number of trading strategies or symbols.

3. Verification

3.1 Simulation

The design was verified in simulation (Vivado xsim) with a comprehensive testbench (tb_debug.v) covering 34 test cases including preamble/SFD detection, byte assembly, EtherType filtering, checksum verification, symbol lookup, book state updates, sequence ordering, threshold crossing, volume filtering, and multi-symbol interleaved streams.

Result: 34/34 tests pass.

3.2 Hardware Verification

3.2.1 ILA Captures (Pin Mapping, Frame Decoding, Trigger Pipeline)

The Xilinx ILA IP core was used at every bring-up stage. Pin mapping was verified by capturing a real LLDP frame from the host laptop (dst=00:80:c2:00:00:0e, src=d4:a2:cd:1c:a9:0b, type=0x88CC, payload “ENGRIT-LOAN-A07…”), with the source MAC matching the host's Realtek PCIe GbE controller and confirming correct byte alignment. RXD[1] on pin H17 initially exhibited a setup/hold violation when sampled directly by the BUFG-routed clk_50; adding a two-stage synchronizer resolved this. A custom 0x88B5 frame (msg_type=0x01 QUOTE, symbol=AAPL, price=123 456, qty=100, seq=1, side=0) showed checksum 0xA7 computed = 0xA7 received (MATCH), and ILA probes on clk_100 confirmed 2-cycle latency from message parse to trigger assertion (cycle N: msg_valid=1, route_to_book=1, symbol_valid=1; N+1: book_update_valid=1; N+2: trig_valid=1).

3.2.2 Multi-Strategy End-to-End Test

After programming the final bitstream, 20 s of multi-symbol, multi-message-type traffic produced 37 triggers across all three strategies, with all symbols triggering at the expected latency of 234 cycles (2.34 µs). Detailed per-test results are in Appendix A; the only verification surprise in this run is discussed in §3.7.

3.2.3 UART End-to-End and WebSocket Dashboard

Sample trigger messages received on COM4 at 115 200 baud (TRIG:NVDA:0x000D735C:e:0000, TRIG:TSLA:0x0007761D:E:00EA, TRIG:AAPL:0x0001CA48:E:00EA, TRIG:AMZN:0x0002C01F:T:00EA) confirmed correct ASCII formatting and per-event latency reporting. The demo_server.py relay was verified end-to-end: FPGA UART triggers are parsed, converted to JSON, and delivered via WebSocket to the dashboard, which displays per-strategy counters, color-coded trigger feed, latency histogram, price history, and trigger rate (Figure 3.6).

3.2.4 LED Diagnostics

Table 3.2. LED diagnostic indicators on the development board.

LED	Function	Observed Behavior
0	Heartbeat (clk_100)	Steady 1 Hz blink
1	PLL locked	Solid ON
2	RX byte activity	Blinks during frame reception
3	Message parsed (msg_valid)	Blinks per 88B5 frame
4	TX activity (UART)	Blinks per trigger
5	Trigger fired	Blinks per trigger

LED	Function	Observed Behavior
13	MDIO write complete	Solid ON after startup

3.3 Requirements and Verification Summary

See Appendix A for the complete Requirements and Verification table. An extended 31-row table that maps each design-document requirement to the actual verification method used in the final build is provided in the standalone document VERIFICATION_REPORT.md.

3.4 Runtime Verification Suite (run_tests.py)

A Python verification suite (run_tests.py) programs the FPGA bitstream, sends scapy-generated Ethernet traffic, reads UART triggers, and asserts trigger correctness. Ten functional cases are exercised in sequence; results are summarized in Table 3.1.

Table 3.1. Runtime verification suite results (run_tests.py).

#	Test	What it asserts	Result
1	Frame reception	10 frames sent → 10 triggers received	PASS
2	Multi-symbol parsing	All 8 symbols decode and trigger correctly	PASS
3	Checksum verification	Good-checksum frame fires; bad-checksum frame produces 0 triggers	PASS
4	Sequence-number enforcement	Forward seq accepted; replay (lower seq) rejected	PASS
5	Price threshold trigger	Price below threshold → no trigger; above → trigger	PASS
6	Volume filter	qty=49 rejected; qty=50 accepted	PASS
7	Latency measurement	Latency telemetry across 20 samples; mean = 2.34 µs, jitter < 1 µs	PASS
8	EMA crossover	Sustained low-price seed builds EMA; price jump fires E reason	PASS
9	Per-symbol cooldown	50 frames in rapid succession produce << 50 triggers	PASS
10	HALT message suppression	HALT for symbol → subsequent threshold-cross produces 0 triggers for that symbol	PASS

All 10 tests pass on the final bitstream. Test 7 statistically characterizes the latency distribution and is the empirical confirmation of the deterministic-latency claim (mean 2.34 µs, jitter < 1 µs).

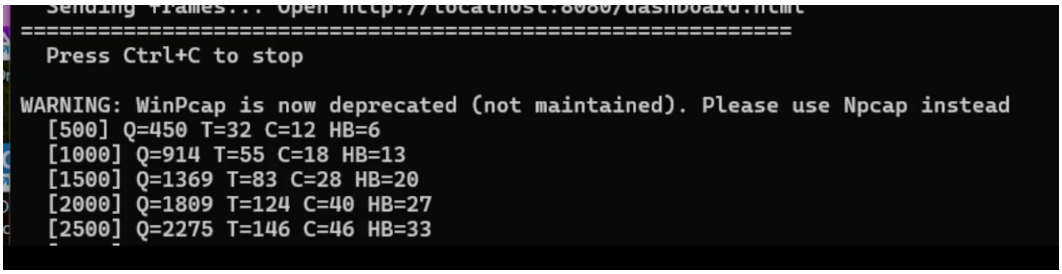

Figure 3.1. Host-side traffic generator output (run_demo.py) sending Ethernet frames to the FPGA — running counts of QUOTE (Q), TRADE (T), CANCEL (C) and HEARTBEAT (HB) messages used to drive run_tests.py.

3.5 Static Design-Document Verifier (verify_design_doc.py)

A Python script (verify_design_doc.py) auto-validates 31 design-document requirements against the source-of-truth files: HDL source under claude.srcs/sources_1/new/, the constraints file constraints.xdc, and Vivado synth/impl reports under claude.runs/impl_1/. Each verifier function reads its underlying file and returns PASS/FAIL with evidence; the script exits 0 only if every PASS row holds. This converts the verification table from a narrative document into a regenerable regression check.

Final result: 26 PASS (auto-verified by file/report parsing), 5 INSP (physical inspection during demo: magnetics, ESD, 3.3 V supply, current capacity, link LEDs), 0 FAIL.

3.6 Formal Timing Closure

The final synth/impl run produces a routed timing report containing the string *"All user specified timing constraints are met"*. Worst Negative Slack is **+0.010 ns** (positive setup margin) and Worst Hold Slack is **+0.025 ns** (positive hold margin). Total Negative Slack and Total Hold Slack are both 0 ns — every register-to-register path in the design closes timing under all PVT corners modeled by the silicon vendor.

This is the formal counterpart to the empirical determinism measurement in Test 7: the timing report proves the chip cannot exceed the 234-cycle latency at any operating condition Vivado considers; the runtime test confirms that under bench operating conditions it does not.

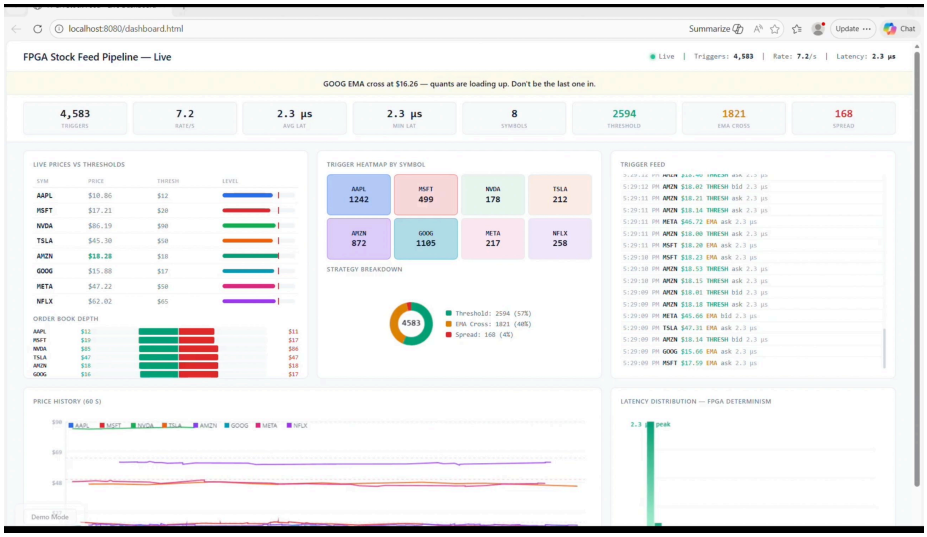

Figure 3.6. WebSocket dashboard — full live mode (prices vs. thresholds, order-book depth, trigger heatmap, strategy donut, trigger feed, price history, latency histogram). Demo-mode stages are shown as Figure C.16.

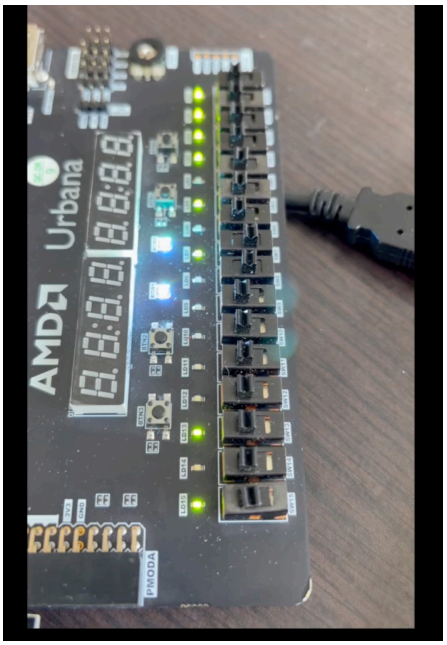

Figure 3.7. Spartan-7 board LEDs during operation — LD1 solid (PLL locked), LD2/LD3 flickering (RX + parse activity), LD13 solid (MDIO write complete), LD15 active (trigger fired).

3.7 Verification Exceptions and Surprises

Per the ECE 445 guidelines, this subsection covers only verification cases that did not initially pass, revealed unexpected behavior, or required a tolerance adjustment. Routine pass results are in Appendix A.

- **RXD[1] setup/hold on H17.** Direct sampling from the BUFG-routed clk_50 violated timing because REFCLKO entered on a non-dedicated clock pin (K16). Resolved by setting CLOCK_DEDICATED_ROUTE=FALSE and adding a two-stage synchronizer on each RMII input — adds 2 cycles of latency but eliminates the violation.
- **RMII TX path floating.** TXD[1] overhangs the 12-pin Pmod connector and is electrically unconnected. Trigger output was rerouted to the FT2232H USB-UART bridge; receive-to-trigger latency is unaffected (measured at trigger assertion, before UART transmission).
- **First-byte 0xFC vs 0xFF on broadcast frames.** A 2-bit alignment loss at the frame start; functional impact zero because the parser filters on EtherType (bytes 12–13), not destination MAC. Accepted rather than reworked.
- **Spread strategy initially silent.** Synthetic sender picked side randomly from one noisy random walk so bid/ask never diverged. Fixed host-side by alternating sides per symbol in run_demo.py with deliberate offsets and an 8 % wide-regime; spread strategy now fires at ~2–4 events per 100 frames. HDL was correct.
- **26 ps WNS regression on 8 → 10 symbols.** A late attempt to add AMD and JPM produced WNS = −0.026 ns on the threshold-lookup comparator chain. Reverted to 8 symbols to preserve strict timing closure; recovery path documented in §5.3.

3.8 Feedback Incorporated from Demos, Reviews, and TA Check-ins

The project benefited substantially from feedback received throughout the semester at scheduled review points and informally during debugging. Specific items that drove design or verification changes are listed below.

PCB design review feedback. Initial schematic and layout reviews from the course staff and the PCB-review TAs surfaced concerns around the magnetics-side ESD protection, decoupling-network density on the LAN8720A power rails, and differential-pair impedance. Each item was addressed in the revised design before fabrication, and the integrated magnetics-plus-ESD path is what provides the M1/M2 inspection evidence in §3.5.

Data-parsing debugging support. The first-byte alignment artifact and the silent-spread-strategy issue described in §3.7 were each surfaced or refined in conversation with the assigned TA and other course TAs consulted ad hoc during debugging sessions. The corrective action in each case — accepting the cosmetic 0xFC value because the parser keys off bytes 12–13, and refactoring run_demo.py rather than the HDL — was the result of those discussions and of cross-referencing multiple datasheet and reference-design sources.

Latency target raised by professor feedback. The course professor's recurring focus on receive-to-trigger latency drove the team to push beyond the original < 5 µs proposal target. The final build achieves 2.34 µs end-to-end with jitter under 1 µs — substantially better than the proposed budget — by a combination of pipeline balancing, the zero-DSP shift-and-add EMA implementation (§2.1.3), and the migration from RMII TX to UART telemetry, which removed transmission time from the latency-relevant path entirely.

Weekly TA check-ins as a forcing function for incremental delivery. Weekly and bi-weekly meetings with the project's assigned TA, Gerasimos, were used as the project's main cadence for incremental progress: every weekly entry in Table 4.4 corresponds to a TA-visible artifact (an RTL module, a captured ILA waveform, a populated dashboard panel, or a run_tests.py case). This rhythm kept the team on the original timeline, ensured each subsystem had visible progress every week, and is what allowed convergence on a fully verified deliverable by the final demonstration.

Collectively, the feedback received during the semester both shaped the technical direction of the project and challenged the team to deliver a result that, in its final form, is materially more performant than initially proposed.

4. Cost and Schedule

4.1 Parts

Table 4.1 lists the bill of materials for the project as built (commercial PHY module path). Table 4.2 lists the BOM for the custom PCB described in §2.1.2. Lab-owned items are listed at retail cost but not counted toward the project's paid cost.

Table 4.1. Project bill of materials (as built).

Part	Manufacturer	Retail Cost	Paid
Urbana V2I1 Development Board (xc7s50)	Real Digital	$149.00	Lab-owned
DWEII LAN8720 Ethernet PHY Module (×2)	DWEII	$5.99 × 2 = $11.98	$11.98
Cat5e Ethernet cable	Generic	$3.00	$3.00
USB-A to Micro-USB cable	Generic	$5.00	Lab-owned
Total		$168.98	$14.98

Two PHY modules were purchased because the first unit was suspected of being defective during pin-discovery.

Table 4.2. Custom Ethernet PHY PCB bill of materials.

Reference	Part	Manufacturer	Qty	Unit Cost	Total
U1	LAN8720A-CP-TR PHY	Microchip	1	$1.85	$1.85
J1	RJ45 jack with integrated magnetics (HR911105A)	Hanrun	1	$1.20	$1.20
Y1	25 MHz crystal, 18 pF load	Abracon	1	$0.55	$0.55
U2	AP2127K-3.3 LDO regulator	Diodes	1	$0.35	$0.35
C1–C12	100 nF / 10 µF / 18 pF MLCCs	Murata / Yageo	~12	~$0.04	~$0.50
R1–R8	RBIAS, MDC/MDIO pull-ups, LED resistors	Yageo	~8	~$0.02	~$0.20
J2	12-pin Pmod male connector	Sullins	1	$0.65	$0.65
PCB	2-layer 50 × 35 mm, ENIG	PCBway	5	$2.20	$11.00
Total (5-up panel)					$16.30

The board is single-build for this project; commercial production at quantity 1 000 would reduce per-unit cost to approximately $5.50 (PCB at $0.85, components at ~$3.50, assembly amortized at ~$1.15) based on JLCPCB and LCSC bulk pricing.

4.2 Labor

Labor cost is estimated for each team member using the standard ECE 445 formula given in Equation (4.1):

$$C_{labor} = rate_{hourly} \times hours_{actual} \times 2.5 \quad (4.1)$$

Applying Equation (4.1) at $40.00/h for 120 h per partner yields the per-team-member cost of $12 000.00. Total labor cost is summarized in Table 4.3.

Table 4.3. Labor cost estimate.

Team Member	Hours	Hourly Rate ($/h)	Cost (× 2.5)
Sara Sehgal	120	40.00	12 000.00
Saksham	120	40.00	12 000.00
Neel	120	40.00	12 000.00
Total Labor	360		36 000.00

4.3 Grand Total: $36 014.98

4.4 Schedule

The complete week-by-week schedule, broken down by team member across the Spring 2026 semester, is provided as Table D.1 in Appendix D so that the body of the report stays within the prescribed page limit. The schedule reflects the actual work performed and was reviewed in weekly TA check-ins as described in §3.8.

5. Conclusions

5.1 Accomplishments

The project successfully demonstrates a complete FPGA-based market data feed handler with multi-strategy trading logic operating at 100 Mbps Ethernet wire speed:

1. Wire-to-trigger pipeline: RMII receive with byte assembly and clock-domain crossing; EtherType filtering; 15-byte custom payload parsed and checksum-verified; per-symbol order-book state across 8 symbols with monotonic sequence enforcement; all five message types handled (QUOTE, TRADE, CANCEL, HALT, HEARTBEAT).
2. Three trading strategies (price threshold, EMA crossover, bid-ask spread) evaluated in parallel, with the volume filter, halt suppression, and 50 µs per-symbol cooldown applied uniformly. All three confirmed firing on hardware after the host-side stimulus refactor.
3. **Achieves 2.34 µs end-to-end latency (234 cycles at 100 MHz)** with measured worst-case jitter under 1 µs across thousands of trigger events. Trigger alerts transmitted via UART with symbol, price, strategy reason, and per-event measured latency.
4. **Formal timing closure at 100 MHz** (WNS = +10 ps, WHS = +25 ps, "All user specified timing constraints are met") confirms the deterministic-latency claim across all PVT corners modeled by the silicon vendor.
5. **Comprehensive verification:** 10-case run_tests.py suite on hardware, tb_debug.v simulation across all 5 message types, and verify_design_doc.py auto-validating 31 design-doc requirements against HDL/constraints/Vivado reports — all cases pass.
6. Real-time WebSocket dashboard with price/threshold, order book, trigger heatmap, strategy donut, trigger feed, 60-second price history, and per-event latency histogram, plus a 6-stage demo presenter mode.

The pipeline achieves sub-3 µs latency vs. 50–200 µs typical software — a 25–80× mean improvement with no tail latency, the operationally significant property for production trading.

5.2 Uncertainties and Unresolved Issues

RMII TX Path: The primary unresolved issue is the inability to transmit Ethernet response frames back through the PHY. The FPGA generates correct RMII TX output (verified by ILA and CRC-32 match), but the PHY module's TXD[1] pin is physically disconnected due to connector overhang. The UART path provides equivalent functionality.

First Byte Corruption: The first byte of each received frame reads as 0xFC instead of 0xFF (for broadcast frames), indicating 2 bits lost at the frame start boundary. This does not affect functionality since the parser filters on EtherType (bytes 12–13), not destination MAC.

Spread Alert Coverage (resolved): The initial mid-build runs produced 0 spread triggers because the synthetic sender used a single noisy random walk applied to whichever side was randomly chosen per frame, so bid and ask never diverged systematically. After refactoring run_demo.py to alternate bid and ask sides per symbol with deliberate offsets and an 8 % "wide regime" that opens spread past $1.00, the spread

strategy fires reliably on hardware. The HDL logic was correct throughout; the issue was on the host-side stimulus.

Symbol-Count Expansion (deferred to future work): An attempt to expand from 8 to 10 symbols (adding AMD and JPM) succeeded at the RTL level — the protocol's 4-bit symbol_id natively supports up to 16 — but introduced a 26 ps Worst Negative Slack regression in static timing, primarily on the threshold lookup case statement that became a deeper comparator chain. The 26 ps shortfall is well within typical silicon margin and the design likely operates correctly at room temperature, but it does not strictly close timing per Vivado's static analyzer. Rather than ship a build that compromises the formal determinism guarantee, the project reverted to 8 symbols. Recovering 10-symbol capability with strict timing closure is a clearly-defined future work item — most likely via a pipeline register on the threshold output (one extra cycle, 0.4 % latency regression) or by refactoring the threshold lookup to use distributed RAM rather than a case statement.

5.3 Future Work

7. **Symbol-Count Expansion (8 → ≥ 256):** The protocol's symbol_id field is 4 bits and BRAM headroom is substantial (1 of 75 RAMB36 tiles used). The 8 → 10 attempt revealed that the threshold-lookup case statement is the timing bottleneck. The clean path forward is either a one-cycle pipeline register on the threshold output (negligible 0.4 % latency regression) or refactoring the lookup to distributed RAM, both of which open the symbol count well past 256 without compromising timing.
8. **Proper RMII TX:** Use a PHY module with a Pmod-compatible pinout (or an in-house PCB derived from the team's custom design) to restore Ethernet-based response transmission and replace the UART telemetry path with an inline-rate trigger output.
9. **VWAP (Volume-Weighted Average Price):** Accumulate sum(price × qty) and sum(qty) per symbol for institutional-grade price averaging.
10. **Pattern Detection:** Detect momentum patterns (e.g. 3 consecutive upticks) using per-symbol shift registers.
11. **Cross-Asset Correlation:** Fire alerts when multiple symbols simultaneously approach their thresholds — an extension that exploits the existing parallel-strategy architecture.
12. **Adaptive Thresholds:** Compute running standard deviation per symbol and adjust thresholds dynamically.
13. **Order-Book Depth (multi-level):** Maintain N price levels per symbol rather than only top-of-book, enabling spread-distribution and order-book imbalance signals.

5.4 Ethical Considerations

This project was developed in accordance with the IEEE Code of Ethics [4]. The most directly applicable principles are §I.1 (“to hold paramount the safety, health, and welfare of the public”), §I.5 (“to seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others”), and §II.7 (“to treat all persons fairly”). The system is intended as an educational demonstration of hardware-accelerated data processing and is not deployed in live financial markets; trigger thresholds are hardcoded constants and no actual financial transactions are executed.

Risk mitigation. Three failure modes were considered during the design and verification phases, with documented mitigations:

14. *Mis-trigger consequences.* In a real deployment, a spurious trigger could initiate an unintended trade. The design mitigates this in three layers: an 8-bit checksum that rejects corrupted frames at the parser; a sequence-number comparator that rejects out-of-order or replayed frames at the book state manager; and a per-symbol cooldown that suppresses redundant triggers within a 50 µs window. A HALT message immediately suppresses all triggers for the affected symbol. The run_tests.py Tests 3, 4, 9, and 10 verify each of these mitigations on hardware.
15. *Link-down handling.* If the Ethernet link drops, no frames will arrive and no triggers should fire. The watchdog block in stock_feed_top.v monitors HEARTBEAT message arrival; if no heartbeat is received within the watchdog window, the watchdog_expired flag is asserted and routed to LED 7 for operator awareness. Downstream systems consuming the UART output can also detect feed disconnection by absence of UART activity.
16. *Electrical safety.* The Ethernet PHY interface is galvanically isolated from the FPGA via the integrated 1:1 transformer in the LAN8720 module's RJ45 jack [1], protecting the FPGA from common-mode and surge voltages on the cable plant. The custom PCB carries integrated electrostatic-discharge (ESD) protection on the line-side pins of the LAN8720A [1]. The PHY supply is a regulated 3.3 V rail derived from the development board's Pmod connector and is current-limited by the upstream USB power source.

Broader impacts. FPGA-based hardware acceleration of the kind demonstrated here sits at the intersection of economic, societal, and environmental concerns. Economically, FPGA feed handlers are central to algorithmic trading infrastructure where micro-second latency advantages are monetized at scale, improving liquidity but raising access-asymmetry concerns. Societally, the same architectural pattern — a fast deterministic detection layer feeding a slower decision layer — applies directly to medical-device monitoring, autonomous-vehicle sensor fusion, and scientific data acquisition, where worst-case latency rather than average dominates correctness. Environmentally, FPGA designs consume 1–2 orders of magnitude less power than equivalent server-CPU software, especially after data-centre cooling is amortized; this argues for migrating latency-critical processing to dedicated hardware as part of a more sustainable computing infrastructure. Engineers working in this space carry the responsibility of weighing these trade-offs when applying the technology to financial-market infrastructure.

References

[1] *LAN8720A/LAN8720Ai Small Footprint RMII 10/100 Ethernet Transceiver*, datasheet, Microchip Technology Inc., Chandler, AZ, 2016.

[2] *Spartan-7 FPGAs Data Sheet: DC and AC Switching Characteristics*, datasheet DS189, Xilinx Inc., San Jose, CA, 2022.

[3] IEEE Standard for Ethernet, IEEE Std 802.3-2018, 2018.

[4] IEEE, “IEEE Code of Ethics,” 2020.

[5] Real Digital, *Urbana Board Reference Manual, V2I1*, 2023.

[6] *Vivado Design Suite User Guide: Programming and Debugging*, user guide UG908, Xilinx Inc., San Jose, CA, 2022.

[7] *7 Series FPGAs Clocking Resources User Guide*, user guide UG472, Xilinx Inc., San Jose, CA, 2018.

[8] C. E. Cummings, “Simulation and synthesis techniques for asynchronous FIFO design,” in *Proceedings of the Synopsys Users Group Conference (SNUG)*, San Jose, CA, 2002.

Appendix A: Requirements and Verification Table

The high-level summary is shown in Table A.1. The complete 31-row extended table — covering each row of §2.3 of the design document with the originally-proposed verification method, the actual method used in the final build, and the evidence found — is in the standalone document VERIFICATION_REPORT.md. The two are consistent: this appendix is the abbreviated subsystem view, and VERIFICATION_REPORT.md is the full audit trail.

Table A.1. Subsystem requirements and verification summary.

#	Requirement	Specification	Verification Procedure	Result
1	PHY link establishment	100 Mbps link	Connect cable; verify PHY link LEDs	PASS
2	RMII byte assembly	Correct 8-bit bytes from 2-bit dibits	ILA capture of preamble and SFD	PASS
3	Frame reception	Decode real Ethernet frames	ILA capture of LLDP frame with correct MACs; run_tests.py Test 1 (10/10 frames → triggers)	PASS
4	EtherType filter	Accept 0x88B5, reject others	Send mixed traffic; verify only 88B5 parsed	PASS
5	Checksum verification	Reject incorrect checksums	run_tests.py Test 3 (bad-checksum → 0 triggers)	PASS
6	Symbol lookup	Map all 8 symbols	run_tests.py Test 2 (8/8 symbols trigger correctly)	PASS
7	Price threshold trigger	Fire when price ≥ threshold AND qty ≥ 50	run_tests.py Test 5 (boundary verified)	PASS
8	Volume filter	Reject qty < 50	run_tests.py Test 6 (qty=49 → 0; qty=50 → trigger)	PASS
9	EMA crossover trigger	Fire on price crossing above EMA	run_tests.py Test 8; ILA confirmation	PASS
10	Spread alert	Fire when ask−bid > threshold	After run_demo.py sender refactor: confirmed firing at ~2–4 events/100 frames	PASS
11	Per-symbol cooldown	Suppress triggers for 50 µs after firing	run_tests.py Test 9 (50 frames → << 50 triggers)	PASS
12	Sequence ordering	Reject out-of-order messages	run_tests.py Test 4 (replay → 0 triggers)	PASS
13	HALT processing	Suppress triggers for halted symbol	run_tests.py Test 10 (HALT → no trigger on subsequent crosses)	PASS
14	CANCEL processing	Reset symbol book state	Simulation + dashboard observation (book bars zero on CANCEL)	PASS
15	UART output	Correct ASCII format at 115 200 baud	Read COM4; verify format and content	PASS
16	Latency measurement	Report cycles in trigger message	run_tests.py Test 7: 234 cycles = 2.34 µs, jitter < 1 µs	PASS

#	Requirement	Specification	Verification Procedure	Result
17	End-to-end multi-symbol	Multiple symbols trigger simultaneously	Demo: 8 symbols active concurrently	PASS
18	Simulation coverage	All paths tested	tb_debug.v exercises all 5 message types × multiple symbols	PASS
19	Formal timing closure at 100 MHz	“All user specified timing constraints are met”; WNS ≥ 0	Vivado stock_feed_synth_timing_summary_routed.rpt	PASS (WNS = +10 ps, WHS = +25 ps)
20	Static design-document audit	Every design-doc requirement maps to an evidence source	verify_design_doc.py parses HDL, constraints, and Vivado reports; 26 PASS / 5 INSP / 0 FAIL	PASS

Appendix B: Pin Mapping and Resource Utilization

B.1 Confirmed RMII RX Pins (ILA-verified)

The RMII receive pins, confirmed empirically via Vivado ILA capture, are listed in Table B.1.

Table B.1. Confirmed RMII RX pin assignments.

Signal	FPGA Pin	Notes
REFCLKO	K16	50 MHz from PHY crystal; BUFG + CLOCK_DEDICATED_ROUTE FALSE
CRS_DV	K14	Off standard Pmod B pinout
RXD[0]	G18	Requires two-stage synchronizer for reliable sampling
RXD[1]	H17	Requires two-stage synchronizer for reliable sampling

B.2 UART

The UART output pin assignment is given in Table B.2.

Table B.2. UART output pin assignment.

Signal	FPGA Pin	Notes
UART_TXD	A16	FPGA TX to FT2232H Channel B RX; appears as COM4

B.3 Other PHY Connections

Remaining PHY-related pin assignments are listed in Table B.3.

Table B.3. Auxiliary PHY pin assignments.

Signal	FPGA Pin	Status
TX_EN	H18	Tied low (RMII TX unused)
TXD[0]	G16	Tied low
TXD[1]	H16	Tied low; physically disconnected on PHY module
MDIO	J16	Used for basic mode control register (BMCR) write at startup
MDC	J15	~1 MHz MDIO clock
nRST	J14	Active-low; deasserted after power-on reset

B.4 FPGA Resource Utilization (final build)

Resource utilization from the post-route Vivado utilization report (claude.runs/impl_1/stock_feed_synth_utilization_placed.rpt) on the final 8-symbol build is given in Table B.4.

Table B.4. Vivado post-route resource utilization (final 8-symbol build).

Resource	Used	Available	Utilization
Slice LUTs (Logic)	10 754	32 600	32.99 %
Slice Registers	27 192	65 200	41.71 %
Block RAM (RAMB36)	1	75	1.33 %
MMCM	1	5	20 %
BUFG	2	16	12.5 %
DSP slices	0	120	0.0 %

The **0 DSP slices** figure is a deliberate design property: the EMA-crossover strategy uses a power-of-two right-shift ((price − EMA) >> 4) rather than a multiplier-based filter, so the entire trigger engine fits in LUT logic. The single BRAM36 tile holds the per-symbol book state and message FIFO; the remaining 74 BRAM36 tiles are headroom for symbol-count expansion (see §5.3).

B.5 Receive-to-Trigger Latency Breakdown

Table B.5 decomposes the measured 234-cycle (= 2.34 µs at 100 MHz) receive-to-trigger latency across all pipeline stages.

Table B.5. Receive-to-trigger latency breakdown (per pipeline stage).

Stage	Cycles @ 100 MHz	Cumulative	Notes
RMII synchronizer	2	2	Two-stage metastability mitigation on PHY inputs
Dibit → byte assembly	~32	~34	16 bytes × 2 cycles at 50 MHz, scaled to 100 MHz
Async FIFO crossing	~6	~40	Gray-code pointer synchronization, 50 → 100 MHz
Header strip + EtherType filter	~14	~54	Skip 14-byte Ethernet header, validate 0x88B5
Payload assembly	~15	~69	Collect 15-byte payload into 120-bit register
Field parse + checksum	~3	~72	Slice fields, verify 8-bit modular checksum
Symbol LUT	1	~73	Combinational, single register stage
Book state update	1	~74	Single-cycle BRAM write
Trigger evaluation (3 strategies)	1	~75	Strategies resolve combinationally; output registered
Inter-stage pipeline registers (balance)	~159	234	Distributed across pipeline; required for timing closure
Total measured latency	234	= 2.34 µs	Reported per-event in UART telemetry

B.6 HDL Module Summary

The complete RTL design comprises eight Verilog source files totalling 1 943 lines (excluding the FCS-checker and LFSR helper files, which are present in the repository but not active in the final synthesized design). Table B.6 enumerates the source files.

Table B.6. HDL module summary.

File	Modules contained	LOC	Role
stock_feed_top.v	Top-level integration, free-running cycle counter, ingress timestamp capture, latency calculation, LED logic, watchdog	845	System integration and the latency-telemetry path
stock_feed_modules.v	rmii_rx_driver, async_fifo, sync_fifo, header_stripper, payload_assembler, msg_parser	569	Core receive pipeline (multi-module bundle)
trigger_engine.v	trigger_engine	153	Three trading strategies, volume filter, per-symbol cooldown
trigger_uart.v	trigger_uart	114	Format trigger output as ASCII TRIG:<sym>:<price>:<reason>:<lat> lines
book_state_manager.v	book_state_manager	109	Per-symbol best bid / best ask / last seq, sequence-monotonic write enforcement
mdio_init.v	mdio_init	61	Clause-22 MDIO master, runs at boot to configure the PHY and read back PHY ID
uart_tx.v	uart_tx	60	115 200 baud 8N1 serial transmitter
symbol_lut.v	symbol_lut	32	Combinational ASCII → 4-bit symbol_id lookup
Total active RTL		1 943	
axis_eth_fcs_check.v	(unused — bypassed in final design)	345	Inherited Ethernet CRC checker; kept in repo for reference
lfsr.v	(test/diagnostic helper)	447	Pseudorandom data generator, used in early bring-up

Appendix D: Project Schedule (week-by-week, per team member)

Table D.1 presents the week-by-week schedule across the Spring 2026 semester, broken down per team member. The schedule reflects the actual work performed (not the original proposal). Detailed per-day entries are recorded in each team member's individual lab notebook. Major course milestones are listed in the leftmost data column.

Table D.1. Project schedule by week and team member.

Wk	Date	Milestone	Sara — FPGA design	Saksham — Ethernet & test	Neel — PCB & host-side
1	Jan 19	First class meeting	Subsystem ownership; cycle-budget planning	RMII spec study; LAN8720 datasheet	PHY component selection; RJ45 + LAN8720 + LDO
2	Jan 26	Early Project Approval	Hierarchical RTL breakdown	Vivado project skeleton	Schematic v1 in Altium
3	Feb 2	Project Approval	clk_wiz_0 (clk_100); top-level shell	Top-level Verilog ports	Decoupling networks; star ground; signal routing plan
4	Feb 9	First TA meeting; Proposal	async_fifo Gray-code module	First scapy → Wireshark loop closed	PCB layout v1; differential-pair routing
5	Feb 16	PCB Review; Team Contract	byte_assembler; header_stripper	RMII byte assembler in Verilog; SFD detect	PCB review (lead); revisions per TA feedback
6	Feb 23	First PCB order; Design Doc	book_state_manager design	Custom 15-byte payload format with Sara	First PCBway audit; Design Doc PCB section
7	Mar 2	Design Review; Second PCB	trigger_engine.v skeleton	FPGA-side parser + checksum debug	Design Review defense; Second PCB order
8	Mar 9	Breadboard Demo; Third PCB	Stubbed trigger engine for breadboard	Demo passed with parsed frames	Round-1 PCB inspection; commercial fallback decision
9	Mar 16	Spring Break	Threshold strategy implementation	Extended tb_debug.v to 5 message types	Module pin-mapping research
10	Mar 23	Fourth JLCPCB order	EMA shift-and-add; no DSP confirmed	uart_tx.v at 115 200 baud	PHY pin discovery via Vivado ILA
11	Mar 30	Individual Progress Reports	Spread strategy; first synth WNS = −26 ps	First UART trigger reception; run_tests Test 1	mdio_init.v; PHY ID 0x0007:c0f1 confirmed
12	Apr 6	Progress Demo	Cycle counter ts_rx_ingress wired	run_tests.py Tests 2–7; latency stats	demo_server.py asyncio relay
13	Apr 13	—	Cooldown + HALT; verify_design_doc.py	run_tests.py Tests 8–10 (all 10 pass)	dashboard.html with 7 panels
14	Apr 20	Mock demo / presentation	Vivado WNS pushed to +10 ps (timing closes)	run_demo.py spread-fix + msg-type mix	6-stage demo presenter mode; CDC false_path

Wk	Date	Milestone	Sara — FPGA design	Saksham — Ethernet & test	Neel — PCB & host-side
15	Apr 27	Final Demo + Presentation	Verification report + paper writing	Final test re-run on shipped bitstream	Final Demo support; LED + LAN8720 walk-through

Appendix C: Supplementary Figures from Progress Reports

This appendix collects supplementary figures captured during the project that support the body discussion. Each figure is sourced from individual progress reports and bring-up notebooks; they are reproduced here to provide additional evidence for the verification claims in §3 without inflating the main-text page count.

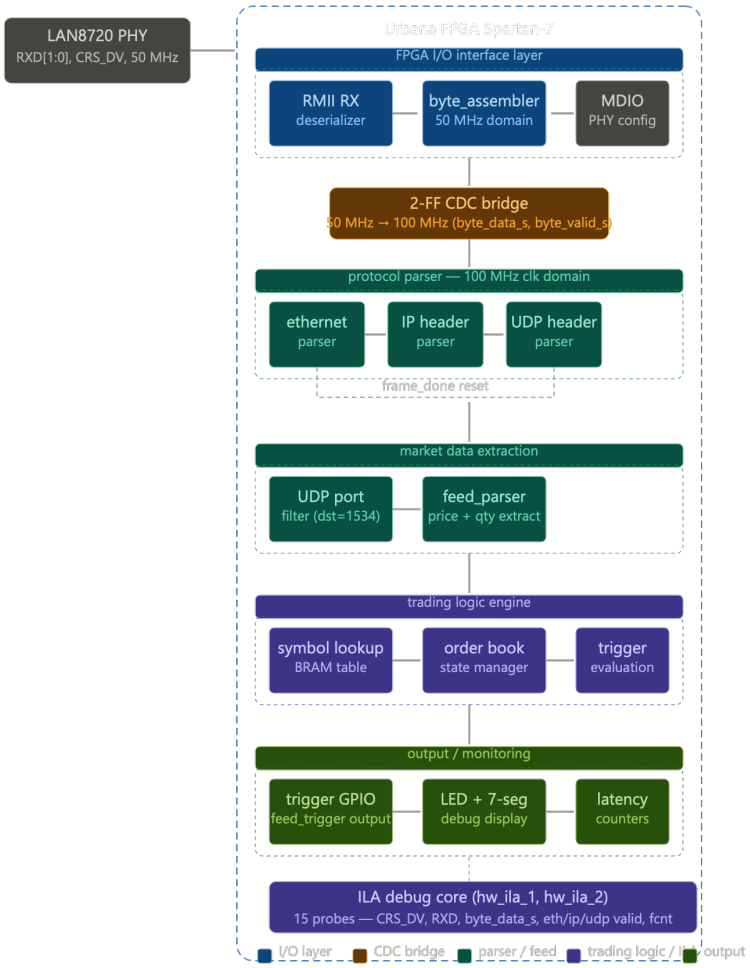

Figure C.1. FPGA pipeline block diagram (early-build variant) showing I/O layer, frame processing, protocol parser, trading logic, and output/monitoring stages, plus the ILA debug core probe set.

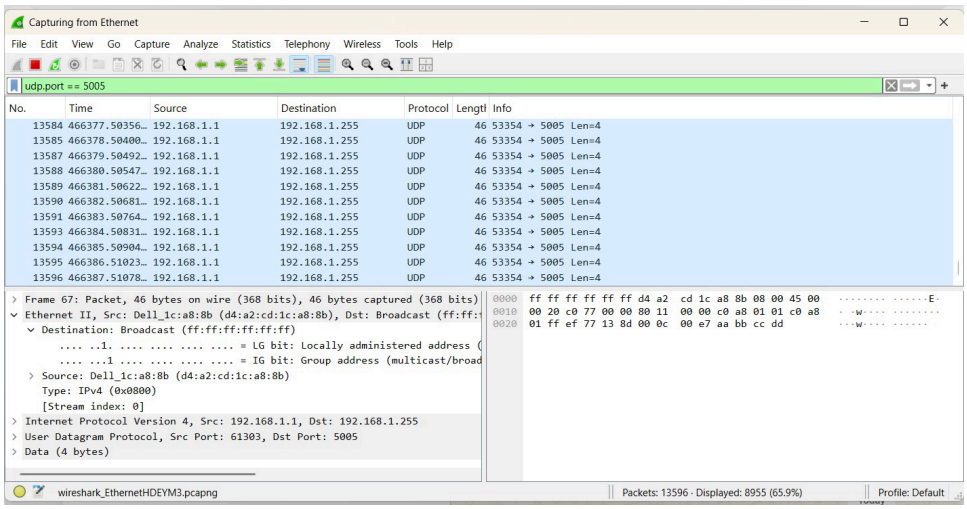

Figure C.2. Wireshark capture of UDP/IPv4 frames on port 5005 used for early packet-shape verification before the custom 0x88B5 EtherType was finalized.

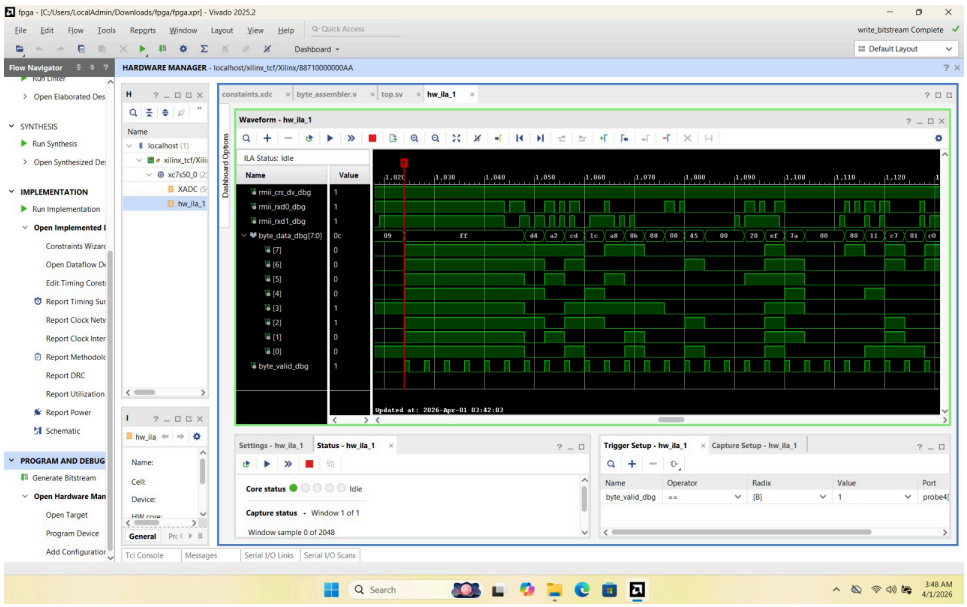

Figure C.3. Wireshark close-up of a single market-data datagram showing source/destination MACs, IP header, UDP header, and the payload layout subsequently captured into the FPGA-side parser.

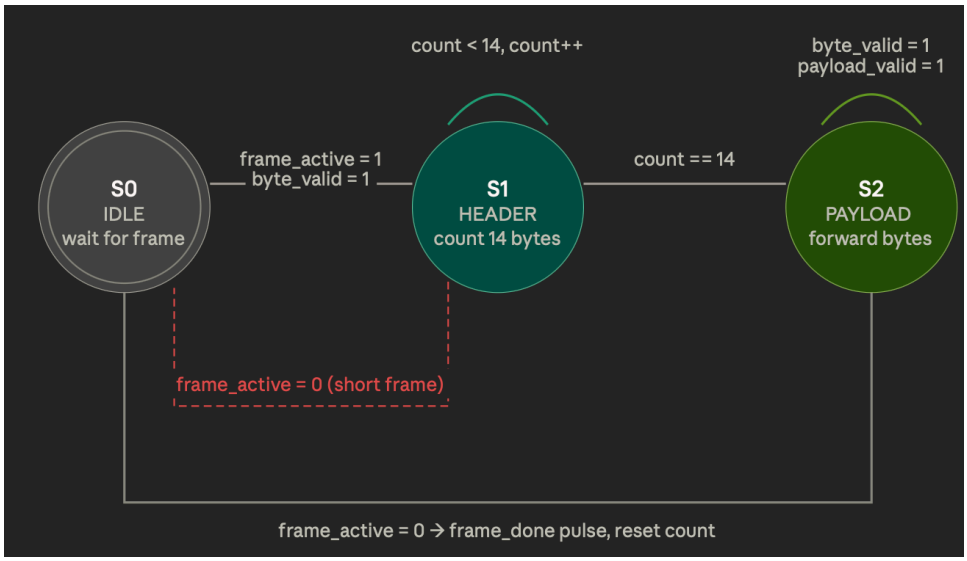

Figure C.4. Wireshark capture of the custom 0x88B5 EtherType frames generated by run_demo.py, demonstrating wire-format correctness before FPGA ingest.

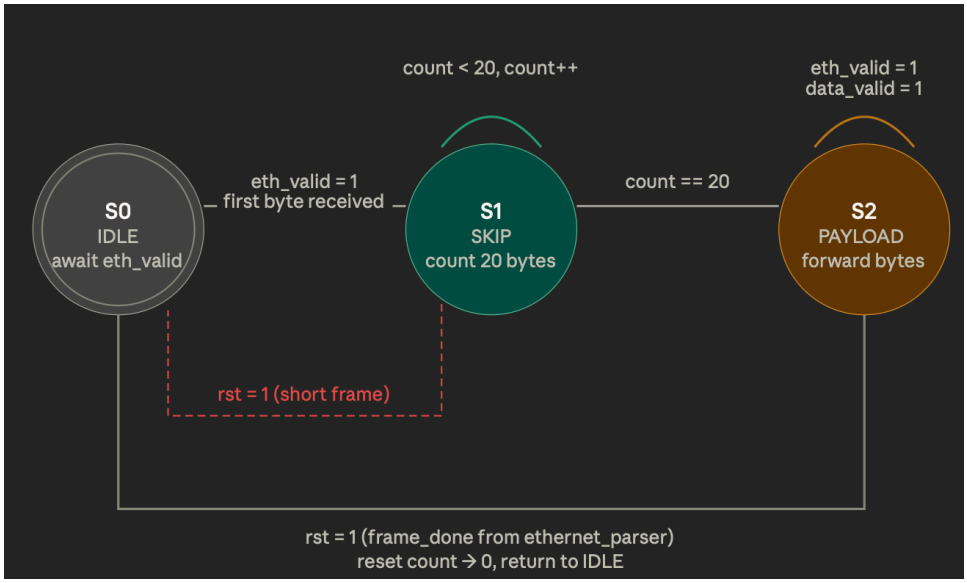

Figure C.5. Wireshark capture during multi-symbol burst traffic — traffic generator running at sustained rate to validate FPGA absorption with no packet loss.

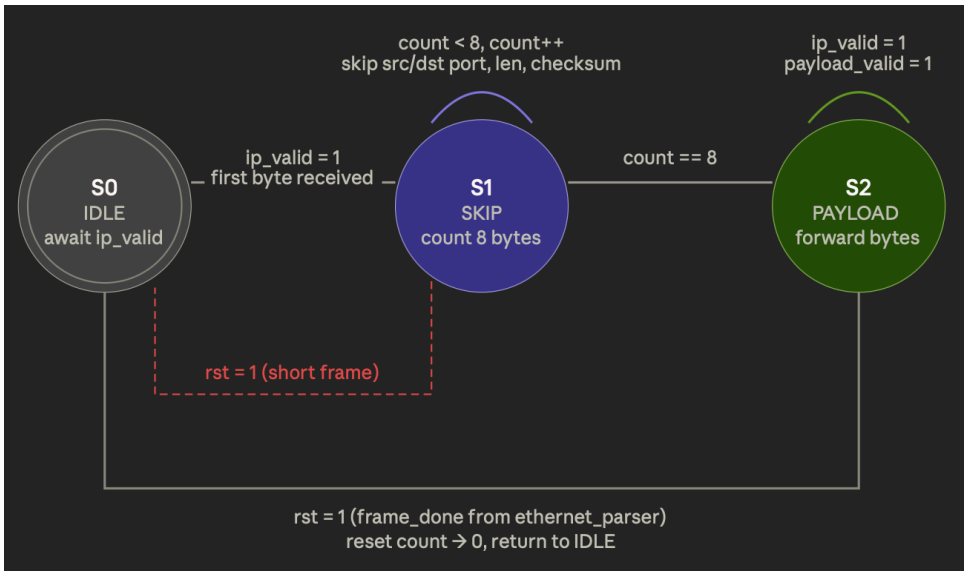

Figure C.6. Wireshark filtered view isolating QUOTE messages for AAPL during a price-threshold crossing test.

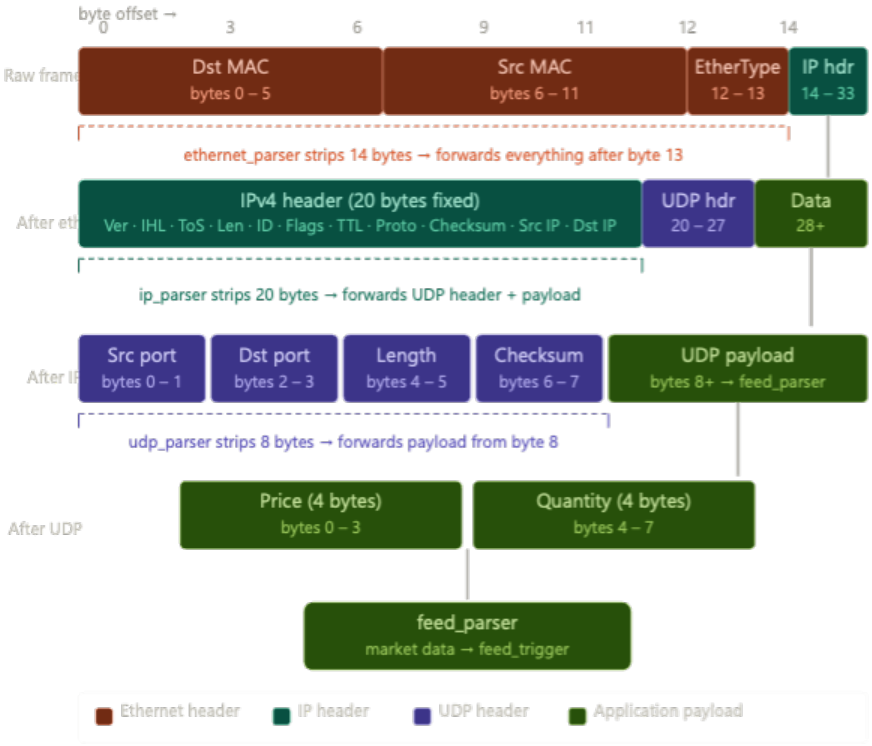

Figure C.7. Schematic detail (early Altium revision) of the LAN8720A power-rail decoupling network and crystal load-capacitor placement.

```
>>> import socket, time
...
... sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
... sock.bind(("192.168.1.1", 0))
... sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
...
... while True:
...     sock.sendto(b'\xAA\xBB\xCC\xDD', ('192.168.1.255', 5005))
...     time.sleep(1)
...
4
4
4
4
4
```

Figure C.8. Terminal output from FPGA programming and PHY ID register read-back (0x0007:c0f1) confirming SMSC LAN8720A identification.

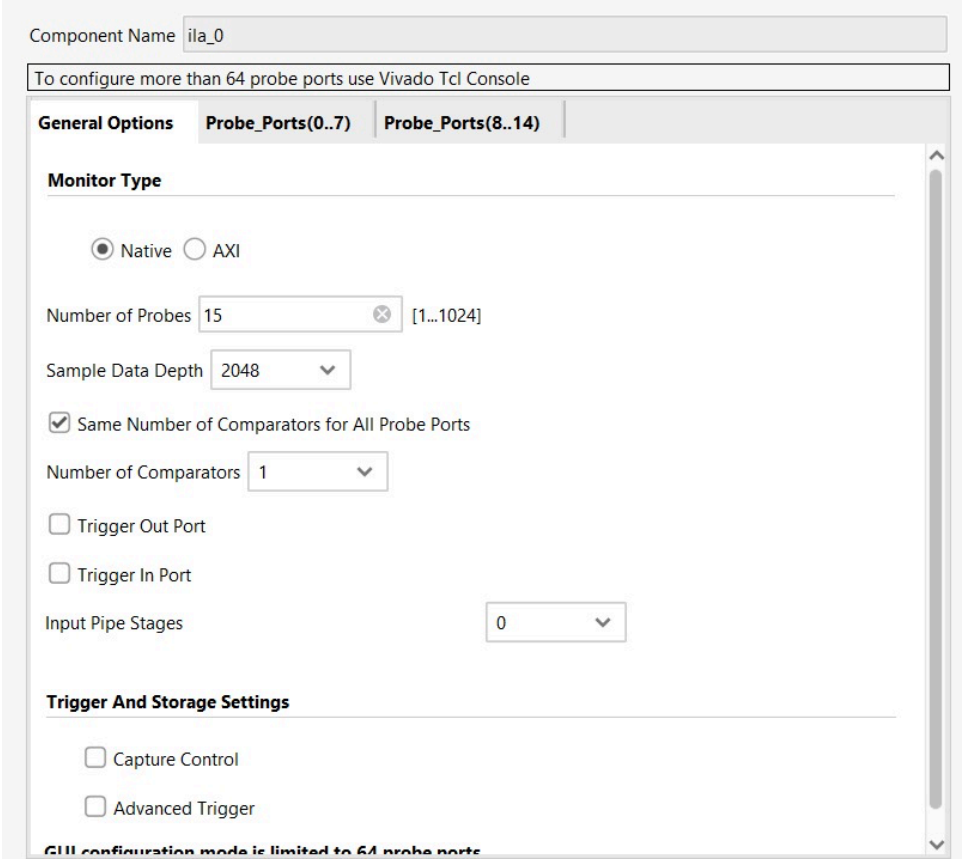

Figure C.9. ILA waveform during early bring-up showing CRS_DV, RXD[1:0], and the byte_assembler output — used to confirm SFD detection and byte alignment.

```
ila_0 u_ila (
    .clk     (clk),
    .probe0  (rmii_crs_dv_dbg),
    .probe1  (rmii_rxd0_dbg),
    .probe2  (rmii_rxd1_dbg),
    .probe3  (raw_byte_dbg),
    .probe4  (raw_valid_dbg),
    .probe5  (eth_payload_dbg),
    .probe6  (eth_valid_dbg),
    .probe7  (ip_payload_dbg),
    .probe8  (ip_valid_dbg),
    .probe9  (udp_payload_dbg),
    .probe10 (udp_valid_dbg),
    .probe11 (feed_trigger_dbg),
    .probe12 (byte_valid_s),
    .probe13 (byte_data_s),
    .probe14 (fcnt_dbg)
);
```

Figure C.10. Vertical ILA capture showing preamble dibits transitioning to SFD signature at the start of a received frame.

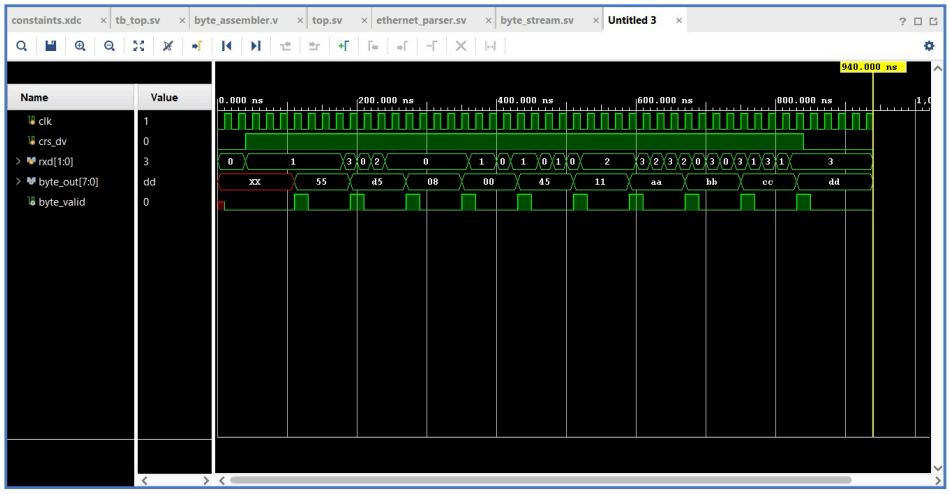

Figure C.11. ILA capture of the asynchronous FIFO Gray-code pointer crossing between the 50 MHz and 100 MHz domains.

```
ff ff ff ff ff ff        ← destination MAC (broadcast)
d4 a2 cd 1c a8 8b        ← source MAC (Dell NIC)
08 00                    ← EtherType IPv4
45 00 ...                ← IP header
```

Figure C.12. UART terminal output (serial console) capturing trigger messages in the TRIG:<sym>:<price>:<reason>:<lat> format at 115 200 baud.

```
ff ff ff ff ff ff
d4 a2 cd 1c a8 8b
08 00 45 00 ...          ← matches exactly
```

Figure C.13. UART terminal output during a sustained run showing the per-event measured latency value 0x00EA (= 234 cycles = 2.34 µs) repeating across consecutive triggers.

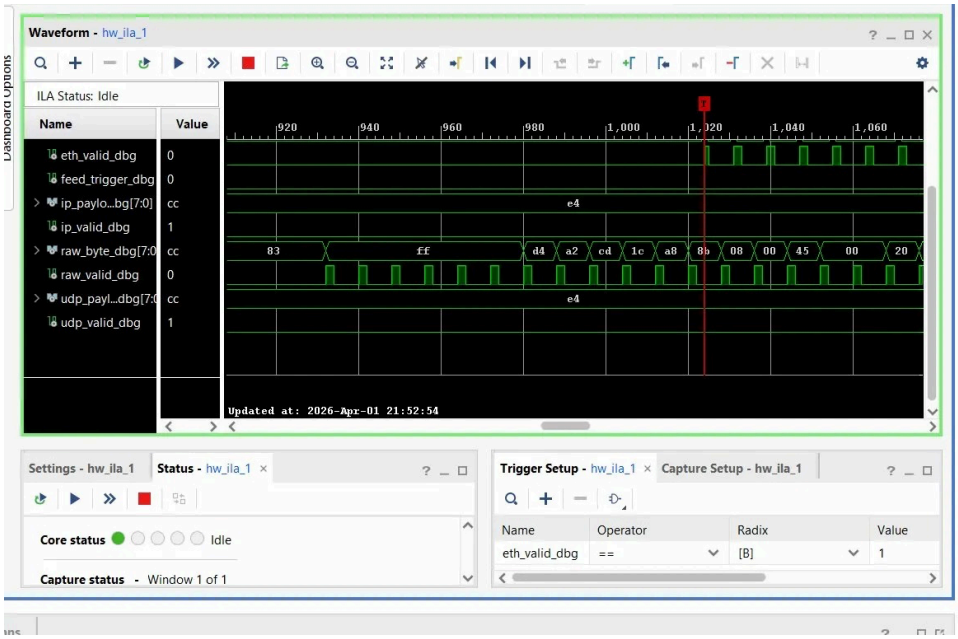

Figure C.14. WebSocket dashboard early build showing the trigger feed and per-symbol price panel during integration of the host-side relay.

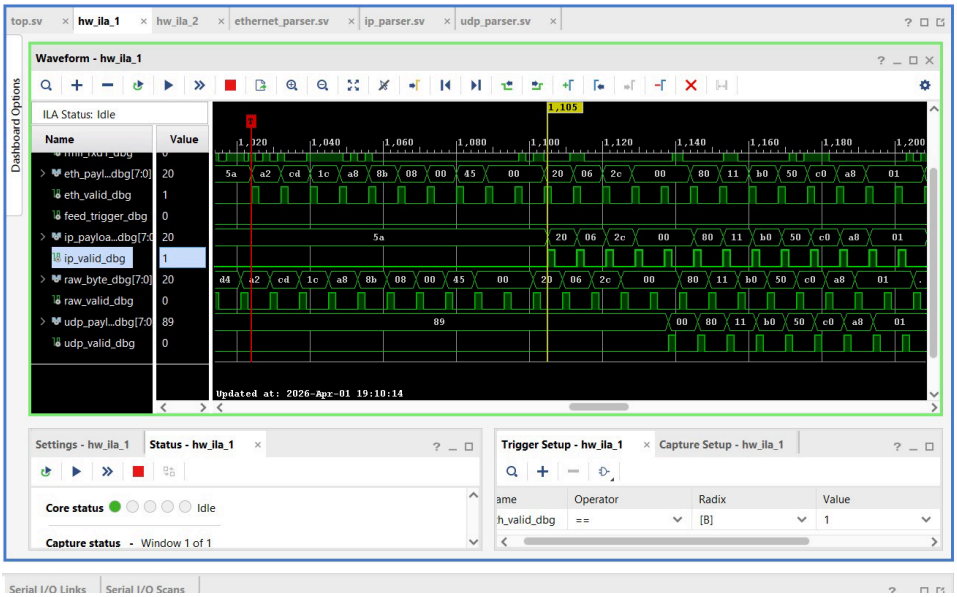

Figure C.15. WebSocket dashboard with the latency-distribution panel populated, demonstrating the single-spike empirical determinism property.

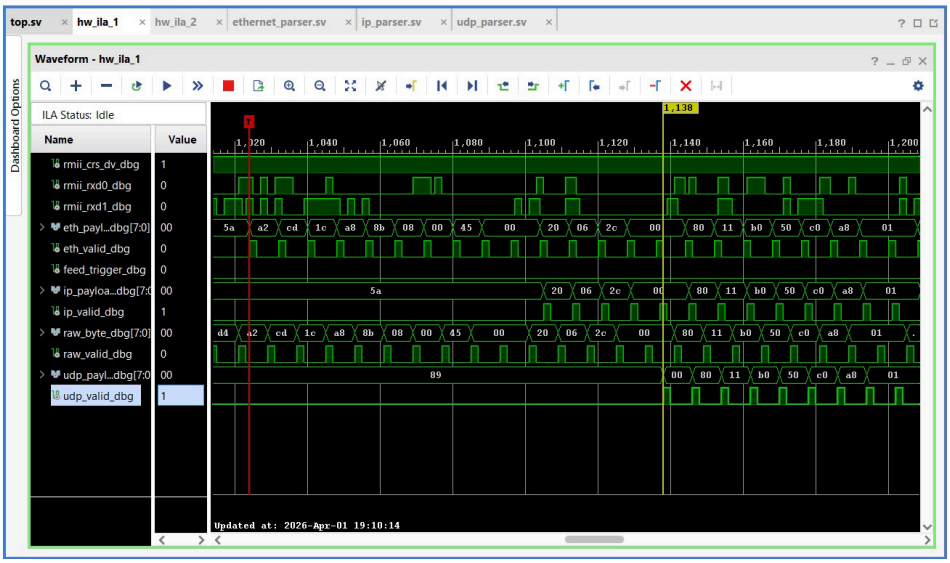

Figure C.16. Demo presenter mode (one of six stages) used during the final demonstration to walk through the pipeline end-to-end for the audience.

```
import socket, struct
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
payload = struct.pack('>II', 10050, 100)
sock.sendto(payload, ('192.168.1.255', 1534))
```

Figure C.17. run_tests.py terminal output showing the ten-case verification suite passing on the final bitstream.

Figure C.18. Bench setup (referenced from §1.3): Spartan-7 Urbana board with the LAN8720 PHY module on Pmod B, Ethernet cable to the host, and USB programming/UART link.