FPGA HFT SYSTEMS

Anirudh, Anya, Maanav, Miller, Sanjay, Yiming

Code vs. Digital Logic

Comparing procedural C++ execution with parallel Verilog digital logic

Software: C++ (Sequential)

Executes line-by-line on CPU clock cycles

```cpp
// Executed sequentially
void logic(bool a, bool b, bool c) {
  bool x = a & b; // Step 1
  bool y = x | c; // Step 2
  out = ~y; // Step 3
}
```

Key Paradigm:

- Instruction Pointer advances sequentially.
- Variables represent memory locations.
- Latency depends on total instruction count.

Hardware: Verilog (Concurrent)

Evaluates continuously as physical gates

```verilog
// Behavioral procedural block
always @(*) begin
  x = a & b;
  y = x | c;
  out = ~y;
end
// Triggers on any sensitivity change
```

Key Paradigm:

- All statements execute simultaneously.
- Code defines physical circuit topology.
- Latency bounded only by gate propagation delay.

Logic Synthesis: Code to Circuit

How Verilog procedural code maps directly into physical hardware logic gates

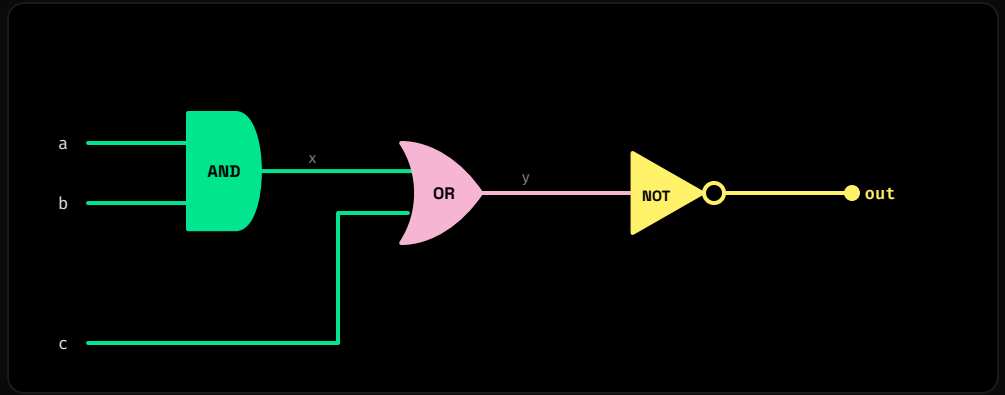
1. Verilog Input

Behavioral description

```
always @(*) begin
  x = a & b;
  y = x | c;
  out = ~y;
end
```

2. Gate-Level Netlist Circuit

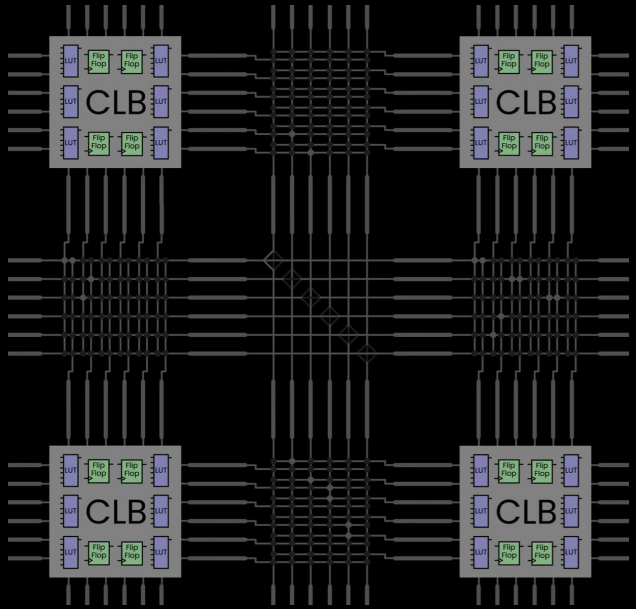
Continuous parallel dataflow through physical silicon logic



Synthesis Tool Impact

The compiler analyzes boolean expressions and infers physical hardware primitives without any CPU execution steps.

What’s in an FPGA?

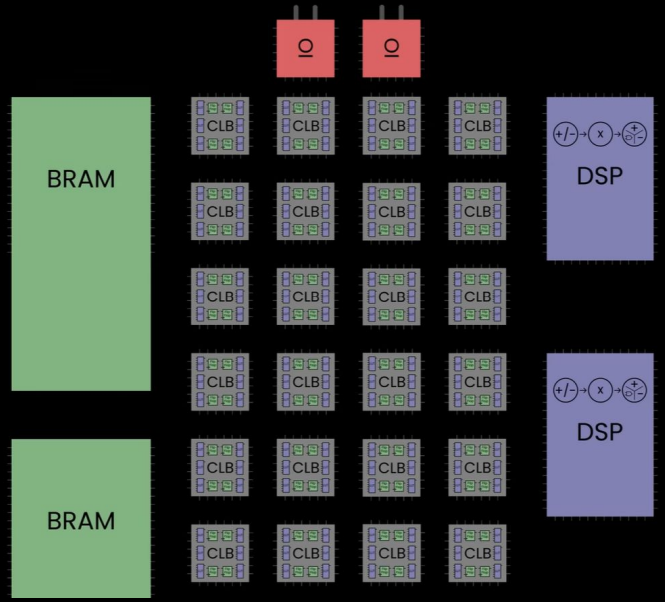


CLBs, LUTS, and Routing

LUTs have programmable truth tables

CLBS contain LUTS and Flip Flops

Programmable routing fills the grid



RAM and DSPs on the mesh

BRAM for efficient memory

DSP for efficient computation

Implementation

Vivado solves the layout and routing for the synthesized logic

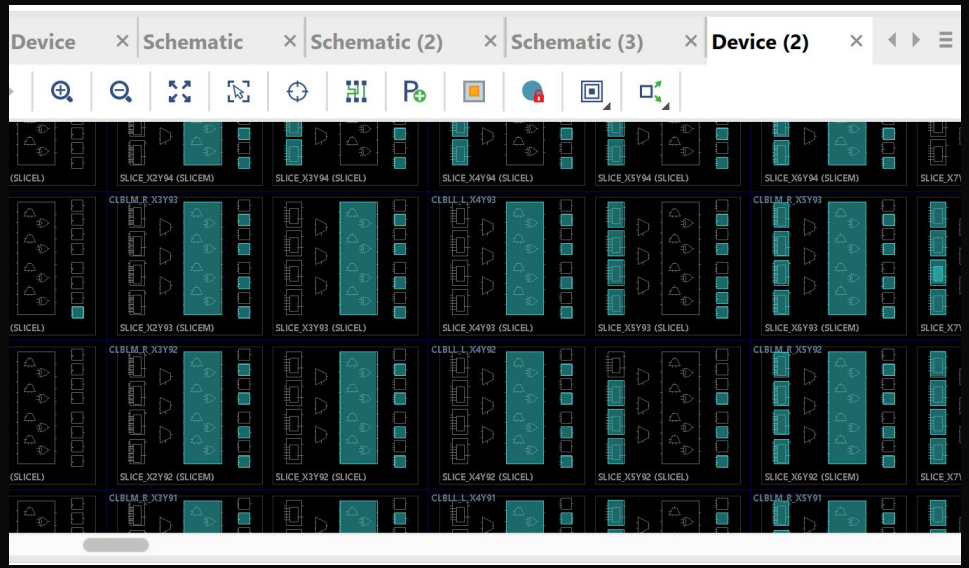
01 / Placement

Maps synthesized logic primitives directly onto physical CLB tiles, LUTs, and dedicated hardware blocks across the FPGA die.

02 / Routing

Configures programmable interconnect wires to establish low-latency signal paths while satisfying strict timing closure constraints.

VIVADO DEVICE VIEW // ROUTING MATRIX

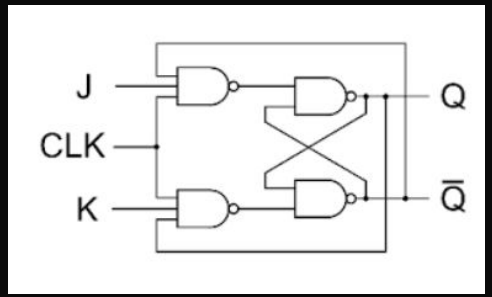


Clocks

Complex behavior is enabled by clock logic — latches and flip-flops allow building state machines and timing-sensitive logic

01 / Sequential Circuits

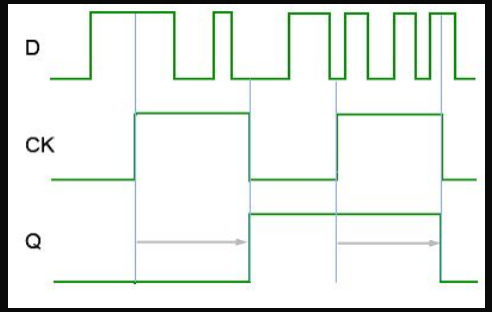
Cross-coupled NAND gates form fundamental SR latch state storage



SCHEMATIC: SR LATCH LOGIC GATES

02 / Timing & Waveforms

Synchronous state changes aligned with clock signal transitions



TIMING DIAGRAM: SIGNAL TRANSITIONS

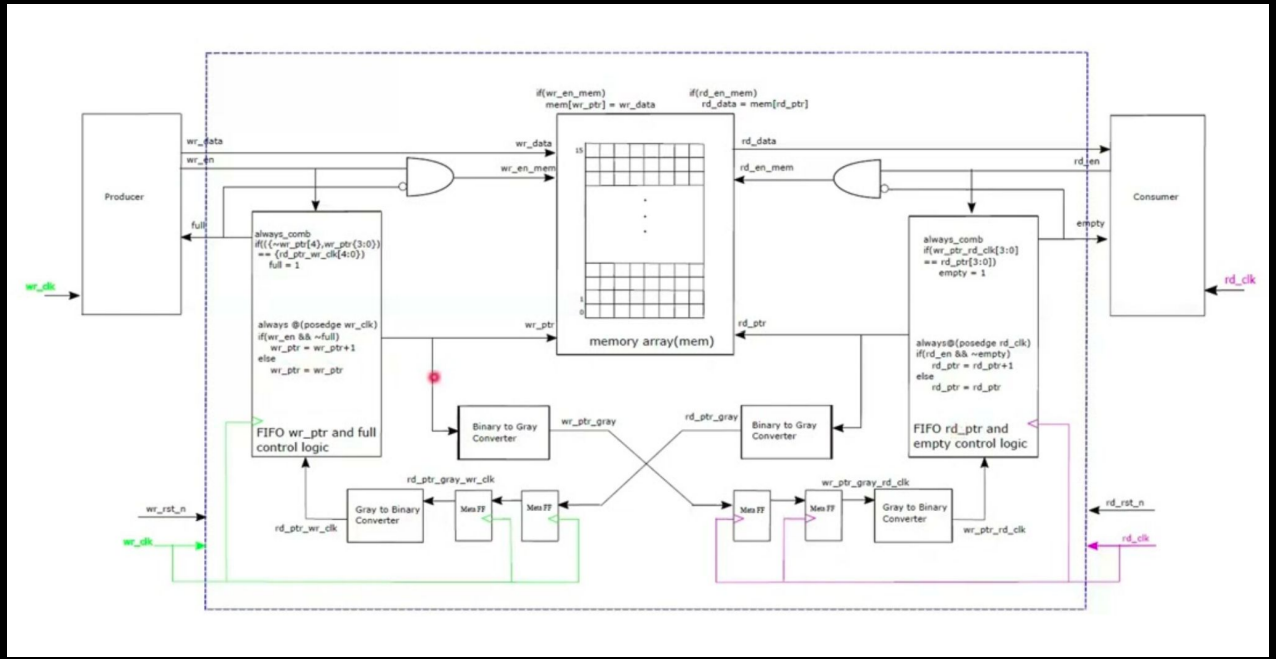
03 / Verilog Implementation

Edge-triggered always block for timing-sensitive register logic

```
// D Flip-Flop w/ Reset
always @(posedge clk or posedge
rst) begin
    if (rst) q <= 1'b0;
    else q <= d;
end
```

RTL: SEQUENTIAL ALWAYS BLOCK

Async FIFO Buffer



Allows for data to cross clock domains

Simply reading and writing into a buffer would result in synchronization issues
The board clock and each ethernet module have their own clock domain
Metastability issues

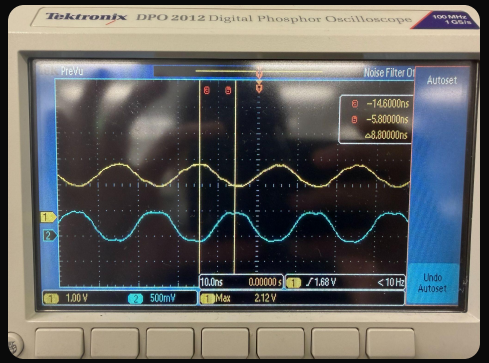
Real Examples

01 / Clock Phase & Frequency Drift

Clock domains out of phase and frequency drift despite both being 50 MHz crystal oscillators



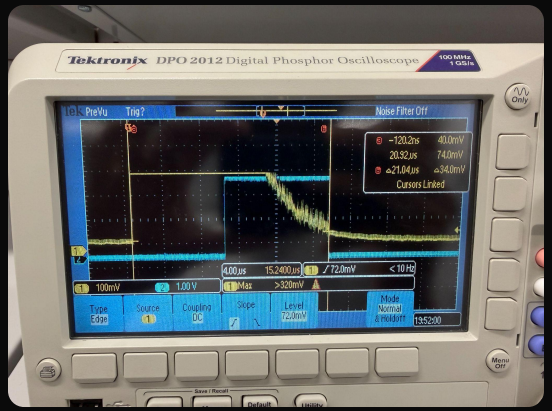
TRACE A: Frequency Drift



TRACE B: Phase Offset

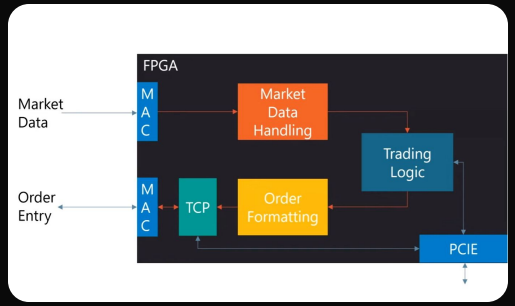
02 / Metastability

Metastability in the wild captured on digital scope



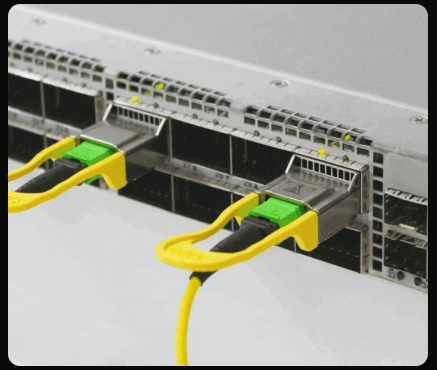
Non-deterministic output state

FPGAs In Industry



01 / Typical System Architecture

Example of typical setup, connects to compute through PCIE for programming and data exchange



02 / Physical Layer & Latency

IRL, the Physical Layer of the network stack is handled through many mediums: fiber, coax, microwave, ethernet, etc. FPGAs are used with various mediums for various strategies at various firms.

Co-location/shortest cable is a big deal because speed of light adds lots of latency

What Makes Computers Slow?

CPUs

Unpredictable response times

- Cache misses, branch mispredictions, interrupts
- Sequential processing

Interpreted vs. compiled languages

- Interpreted languages evaluate source code at runtime; slower than compiled languages, which compile source code into machine code

OS limitations

- Process priority, preemption, context-switching, energy saving modes, resource contention

Network limitations

- NIC → RAM → Packet processing → user program

FPGAs

Reliable with lower clock speed

- Requires fewer steps, fewer data transfers

Parallelism

- Executes concurrent logic pipelines directly in hardware

More direct I/O Access

- Bypasses kernel & memory bus latency

Our Design — Creating a Fair Comparison

GOAL

Compare FPGA and computer timing on the same strategy

01 / Standard Physical Layer

Standard interface between laptop and board - the Physical Layer should be set up such that the packet is sent the same way and with reliable timing

02 / Ethernet Medium

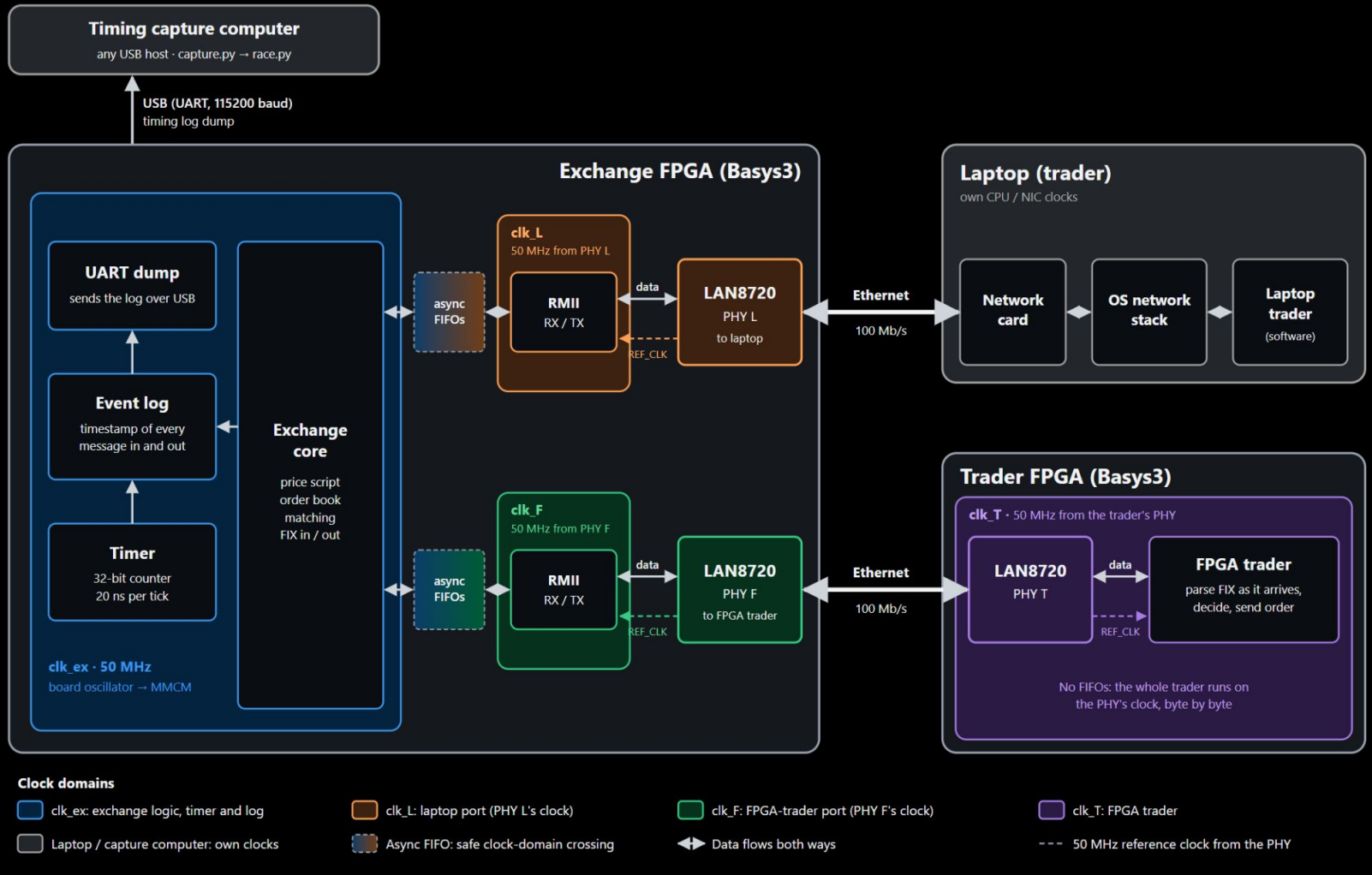
USB is slow and inconsistent, laptops can't wire in directly, needed to use ethernet

03 / Rate Equalization

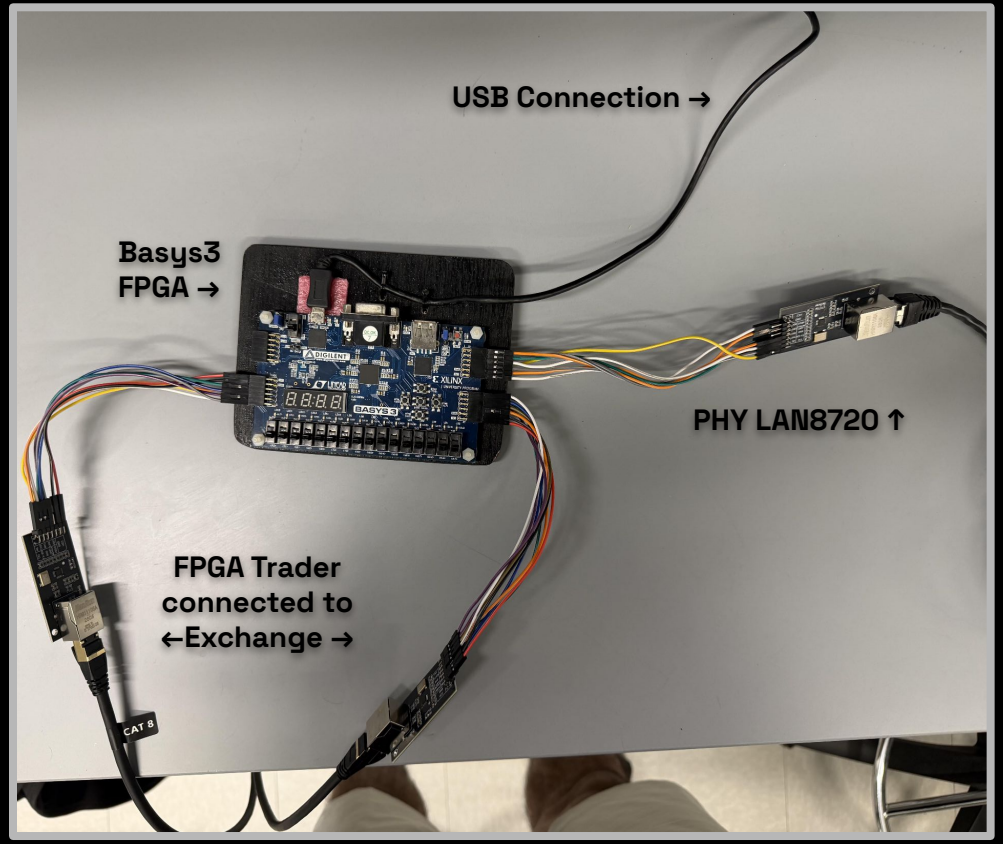
Can't afford FPGA with Gigabit ethernet, 100 MBPs pinned on both for fair comparison

04 / Master Reference Clock

Timing needs to be reliable and consistent, one reliable clock to measure both


Timing capture computer
any USB host · capture.py → race.py
USB (UART, 115200 baud)
timing log dump
Exchange FPGA (Basys3)
UART dump
sends the log over USB
Event log
timestamp of every message in and out
Timer
32-bit counter
20 ns per tick
clk_ex · 50 MHz
board oscillator → MMCM
Exchange core
price script
order book
matching
FIX in / out
async FIFOs
clk_L
50 MHz from PHY L
RMII
RX / TX
data
REF_CLK
LAN8720
PHY L
to laptop
clk_F
50 MHz from PHY F
RMII
RX / TX
data
REF_CLK
LAN8720
PHY F
to FPGA trader
async FIFOs
Ethernet
100 Mb/s
Ethernet
100 Mb/s
Laptop (trader)
own CPU / NIC clocks
Network card
OS network stack
Laptop trader
(software)
Trader FPGA (Basys3)
clk_T · 50 MHz from the trader's PHY
LAN8720
PHY T
data
REF_CLK
FPGA trader
parse FIX as it arrives, decide, send order
No FIFOs: the whole trader runs on the PHY's clock, byte by byte
Clock domains
clk_ex: exchange logic, timer and log
clk_L: laptop port (PHY L's clock)
clk_F: FPGA-trader port (PHY F's clock)
clk_T: FPGA trader
Laptop / capture computer: own clocks
Async FIFO: safe clock-domain crossing
Data flows both ways
50 MHz reference clock from the PHY

Physical Components Hardware Architecture & Setup



EXPERIMENTAL VALIDITY

Since separate digital logic is truly separate, we can put the exchange and the trader on the same chip. The logic and timing don't interact, so it's still a fair experiment.

Hardware Annotations

- **Basys3 FPGA:** Primary board hosting isolated trader & exchange logic.
- **PHY LAN8720:** Ethernet to RMII interface with 50 MHz clock oscillator.
- **USB Connection:** High-speed interface to laptop for programming & data dumps.
- **Trader-Exchange Link:** Direct on-chip interconnects eliminating external latency.

Our Design - Data FIX Protocol & Specification

FIX (Financial Information eXchange): Protocol for exchanging trading information between exchanges, firms, and market participants.

- Exchanges offer a FIX gateway for order placement & market data feeds.
- We utilize a streamlined, low-latency version of FIX for exchange-trader communication.

Messages Sent by Exchange

- **Incremental Market Update** (downloaded to FPGA)
- **Order / Cancel ACK**
- **Fill**

Messages Sent by Trader

- **New Order**
- **Cancel**

Sample FIX Message Format

New Order Single (35=D)		
34 MsgSeqNum	Sequence number	34=00001
11 ClOrdID	Client order ID	11=00001
55 Symbol	Instrument ticker	55=AAPL
54 Side	Buy or sell	54=1
38 OrderQty	Number of shares	38=00100
40 OrdType	Order type	40=2
44 Price	Limit price	44=0100.00

Software Tech Stack & Execution Environments

FPGA Exchange Verilog

Hardware-based exchange simulation component for fast market order processing.

FPGA Strategy Verilog

Low-latency trading logic implemented directly in hardware for sub-microsecond decisioning.

Unoptimized CPU Trader Python

High-level software implementation serving as a functional baseline for performance testing.

Optimized CPU Trader C++

Highly optimized software execution engine built for software-only benchmarking.

System Environment Config Laptop Settings

Laptop Settings: Energy Saving Mode Off

Prevents CPU throttling and frequency scaling during benchmarking to maintain deterministic execution performance.

FPGA Logic Low-Latency Execution Pipeline

Pipeline Flow: Listen → Check → Decide → Fire | Reads price byte-by-byte; Decides what stored packet from RAM to transmit back

01

WAIT

Idle state monitoring incoming traffic.

- Wait for packet start signal on RMII interface.
- Configuration changes only applied between packets.

02

READ & CHECK

On-the-fly byte parsing (Bytes 0–96).

- Validate destination & FIX message format.
- Bytes 66–70: Store packet sequence number.
- Bytes 75–78: Verify ticker symbol match.
- Byte 84: Confirm sell offer tag ('1').
- Bytes 90–96: Extract price digit-by-digit.

03

DECIDE

Instant trigger evaluation at Byte 96.

Latency: 20 ns
(1 Clock Cycle)

- **BUY Condition:** Packet valid + Ticker match + Sell offer + Price ≤ Limit.
- Otherwise, abort and take no action.

04

SEND

Return packet from memory

- Iterates through stored return packet
- Populates stored template with sequence number, price, & checksums.
- Returns to WAIT state upon completion.

Speculative Transmission

INCOMING ETHERNET FRAME arrives byte by byte

Preamble + SFD	Ethernet header	IP / UDP headers	Payload: ticker + price	FCS

01 / Identify frame
Recognize headers and protocol

02 / Parse live data
Extract ticker and price as the payload arrives

03 / Evaluate trade
Compare price against threshold while transmitting

OUTGOING RESPONSE starts before the decision is complete

Response headers + payload stream out speculatively	Final 4 bytes: FCS

ACCEPT → correct FCS → frame accepted

REJECT → corrupt FCS → frame dropped

Checksum: a value calculated from the frame's bytes to detect errors. The final 4-byte FCS carries it; corrupting the FCS makes the receiver drop the frame.

Timing Results

Python, Computer

MEDIAN LATENCY

249 µs

- **95% faster than 472 µs**
- Mean latency: 313 µs

C++, Computer

MEDIAN LATENCY

124 µs

- **95% faster than 158 µs**
- Mean latency: 126 µs
- Maximum observed: 5.334 ms

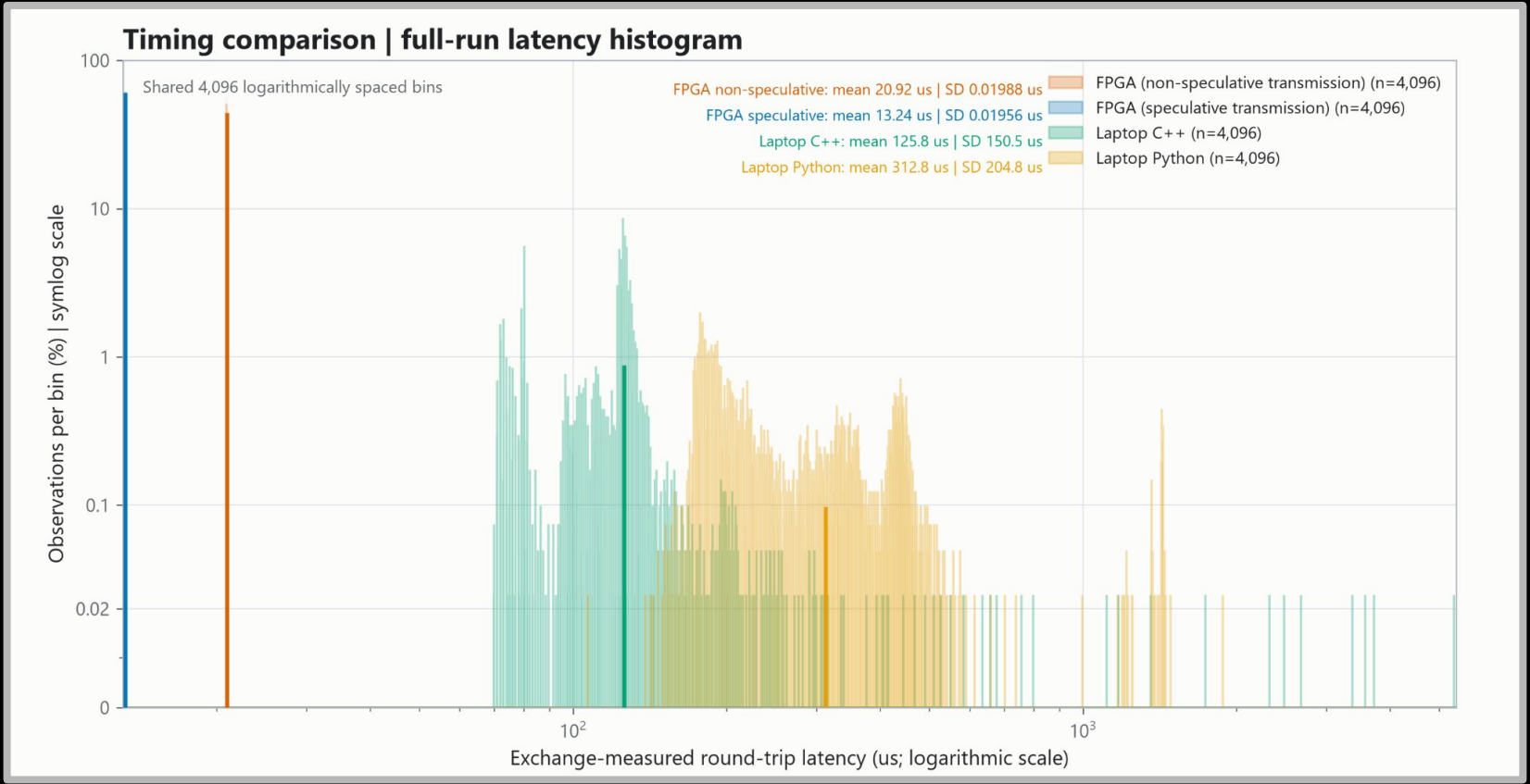
FPGA

SPECULATIVE LATENCY

13.26 µs

- **20.90 µs non-speculative**
- **37% latency reduction via speculative transmission**
- Only 2 observed latency values per capture

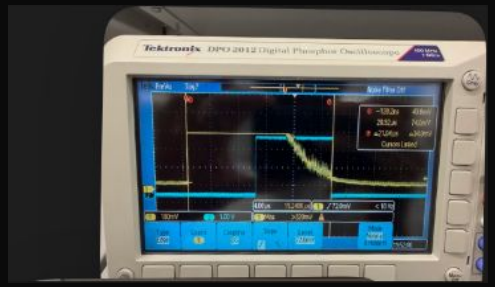
Timing Results



Oscilloscope Measurements

What each capture measures, and what its timing tells us

01 / Full exchange

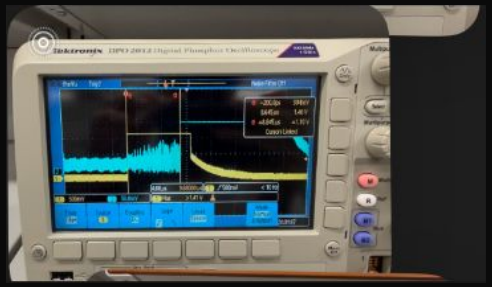


START Exchange TX_EN rise
END Last reply CRS_DV fall

≈ 21.04 µs

Both frames, FPGA processing and PHY delay, end to end.

02 / Trader response

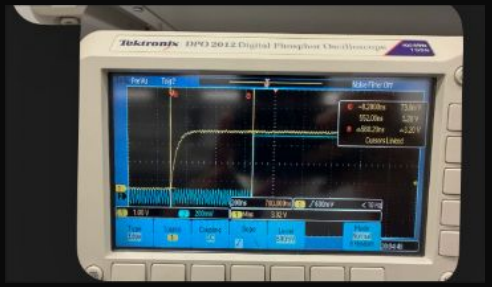


START Trader CRS_DV rise
END Trader TX_EN rise

≈ 8.845 µs

Time spent receiving and evaluating data before the reply starts.

03 / PHY + cable



START Exchange TX_EN rise
END Trader CRS_DV rise

≈ 560.2 ns

One-way delay across both PHYs and the cable.

Timing Math

RMII at 100 Mb/s · 50 MHz clock · 2 bits per clock = 80 ns per byte

01 / Transmission time

Market data: (8 + 119) bytes × 80 ns **10.16 µs**

Reply: (8 + 130) bytes × 80 ns **11.04 µs**

02 / FPGA + PHY

Decision logic: ~40 ns (2 cycles)

PHY + cable: ~560 ns each way

03 / Why speculative is faster



Measured round trip: **20.90 µs** non-speculative → **13.26 µs** speculative (37% lower)

100 Mbps vs. 1 Gbps: Scaled Calculation

Wire serialization drops 10× at 1 Gb/s, while PHY and hardware delays scale non-linearly

1 Gb/s Scaling Factors

Wire Serialization

At 1 Gb/s a byte takes 8 ns, dropping wire transmission time by exactly 10×.

PHY Delay Overhead

PHY delay does not scale 10×. Skips 4B5B coding but adds DSP (echo cancellation, equalization). Typical pairs add ~300–600 ns one way.

FPGA Execution

Runs at 125 MHz (vs. 50 MHz), shortening fixed logic cycle times.

Segment	**100 Mb/s** (measured)	**1 Gb/s** (estimate)
PHY + cable, one way	**0.56 µs**	**0.3–0.6 µs**
Trader response (CRS_DV → TX_EN)	**8.845 µs**	**0.95–1.3 µs**
PHY + cable, return	**0.56 µs**	**0.3–0.6 µs**
Reply frame	**11.04 µs**	**1.10 µs**
Full exchange	**21.04 µs**	**≈ 2.6–3.5 µs** (~3 µs)

How we can optimize the (linux) OS further

CPU Pinning & Core Isolation

Dedicate CPU cores to trading threads, reducing interference from background processes, context switching, and scheduling causing delays

Network IRQ Affinity

Control which CPU cores handle network interrupts, minimizing disruption to latency focused trading processes.

Kernel Network Tuning

Optimize packet processing, TCP configurations, and socket behavior to reduce network overhead

Dedicate logging cores

Keeping the logging writes from the trading path will prevent there being scheduling for logs.

Key Impact: This would achieve more consistent and faster execution times, and decrease tail latency.

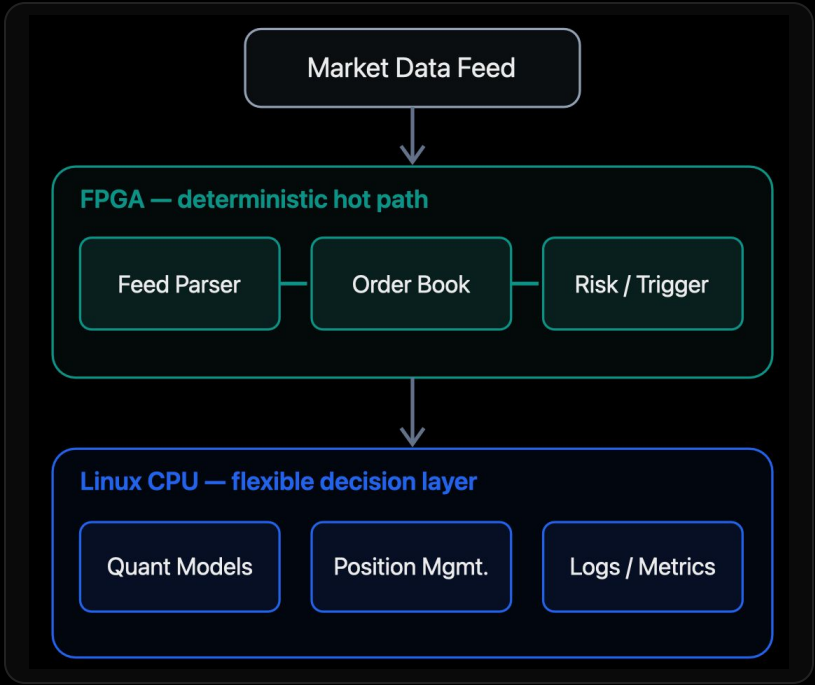
Hybrid Linux + FPGA Architecture

FPGA

Parses FIX market data feed, updates orderbook, and executes immediate risk/trigger logic with zero OS jitter which bypassing the Linux CPU completely. This should be deterministic always.

Linux CPU

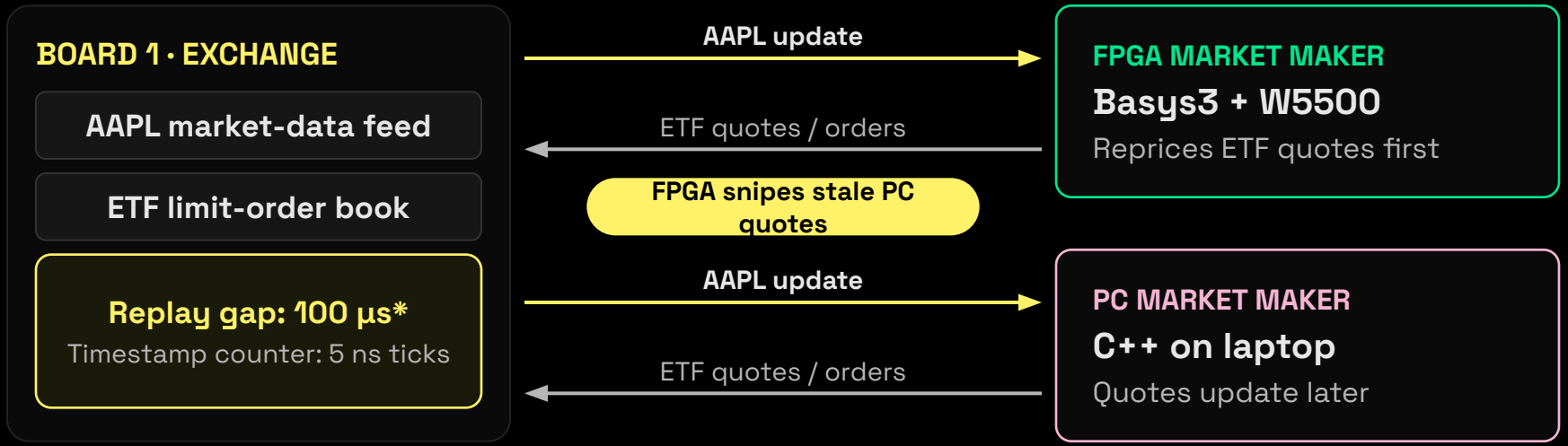
Runs complex quantitative models, manages long-term position state, and handles asynchronous telemetry and system logging off the critical latency path.



Key Metrics to Track: Tick-to-trade latency · p99 / p99.9 execution jitter

Market Making with Latency

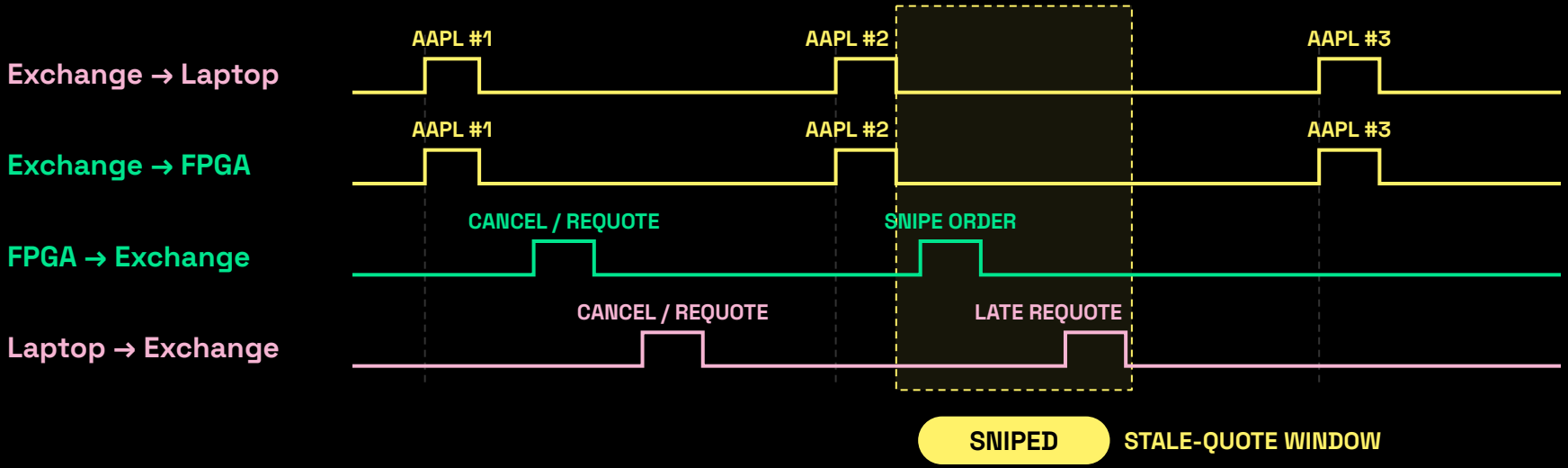
Two market makers receive the same AAPL update; the FPGA reacts first to stale ETF quotes



* 100 µs is the configurable gap after each completed replay exchange; a simultaneous two-trader broadcast is assumed.

The Packet Race

Time flows left → right · illustrative event sequence, not a measured latency trace



AAPL UDP data

20 B payload; sent after each reply plus a 100 µs gap*

ETF quote updates

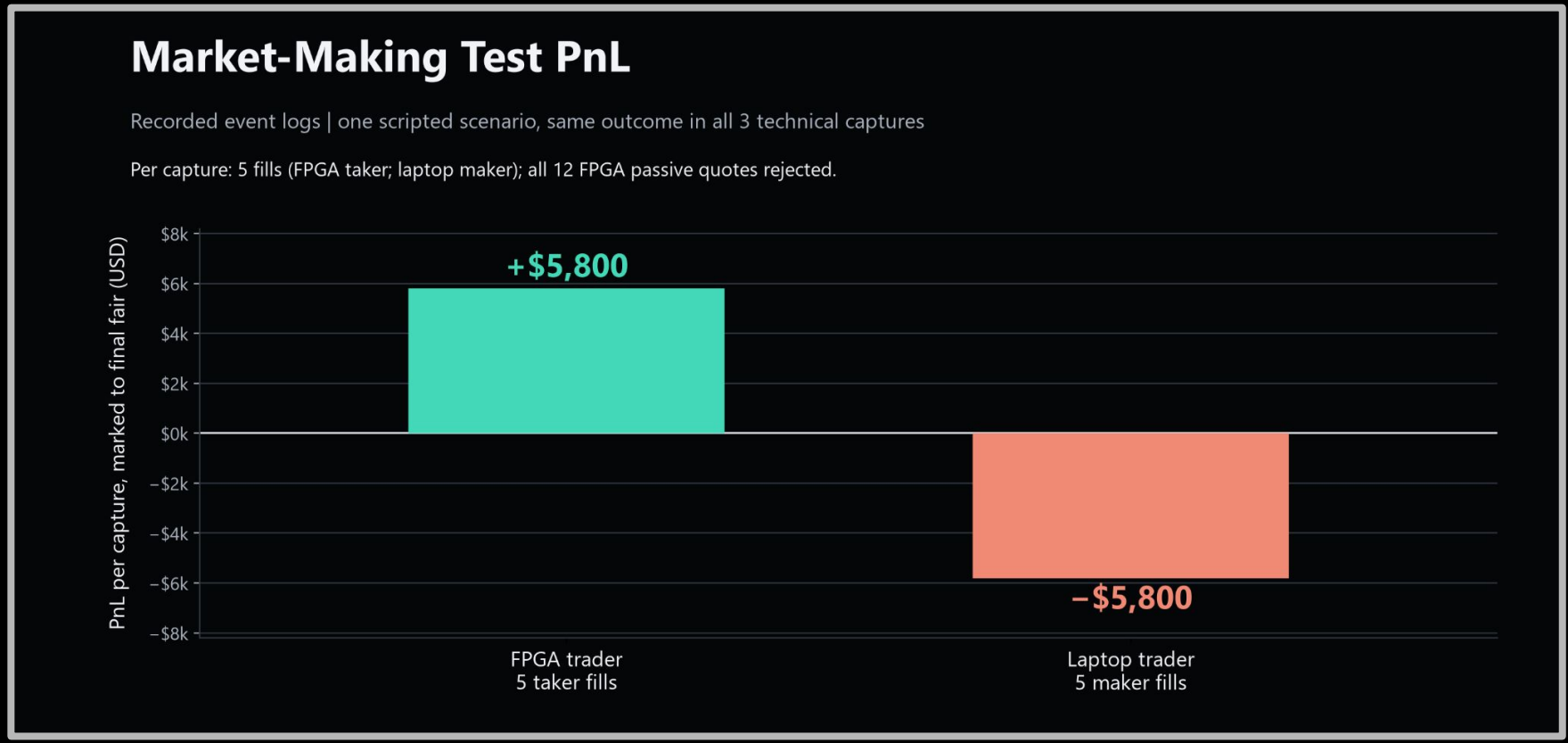
Event-driven after each price update; no fixed rate

Snipe order

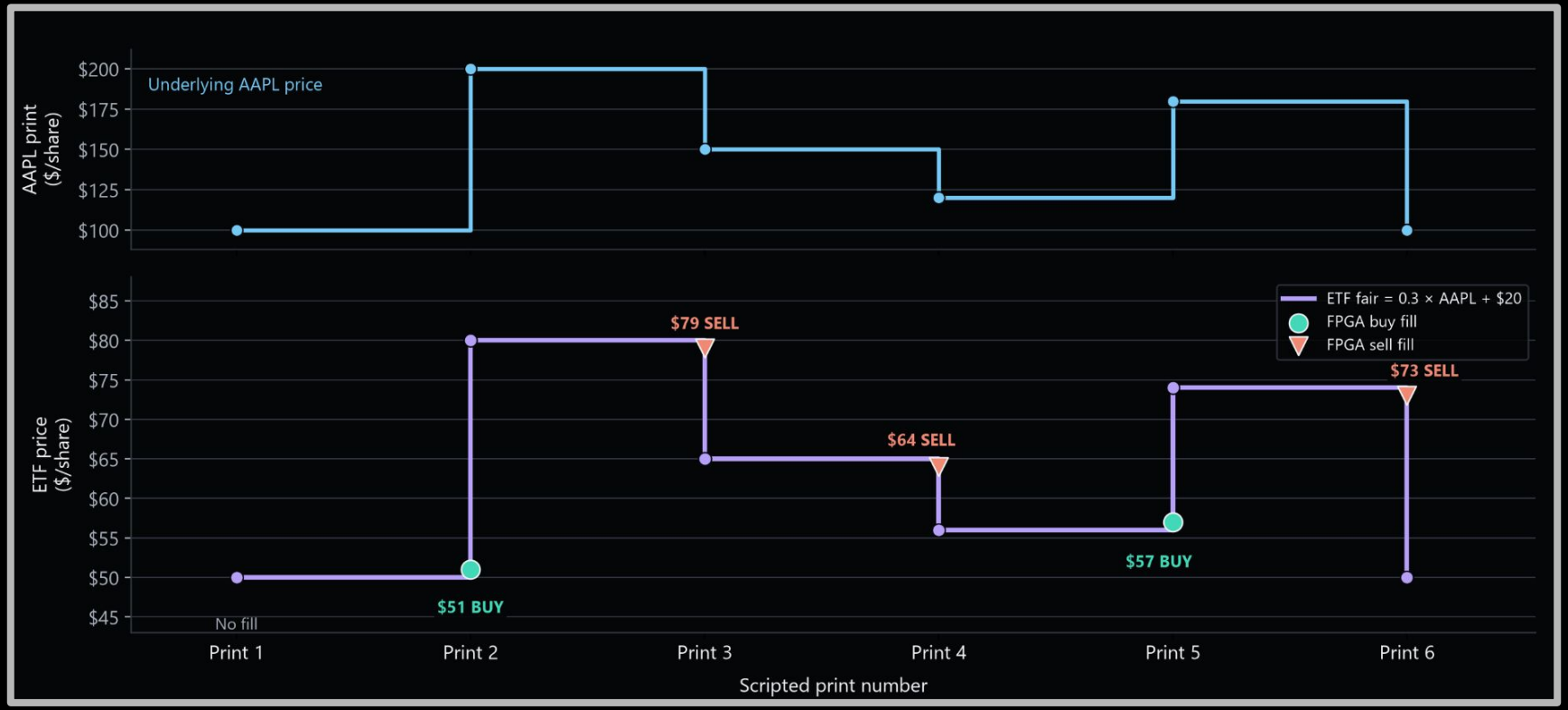
Conditional: only while the PC quote is stale

* GAP_US = 100 µs idle wait after each reply; the effective interval adds round-trip time. ETF packets are conceptual.

PNL and Results



PNL and Results



Thank You!

Questions?

Anirudh • Anya • Maanav • Miller • Sanjay • Yiming

Image creds

- https://www.electronicsforu.com/technology-trends/learn-electronics/flip-flop-rs-jk-t-d
- https://www.youtube.com/watch?v=d3nuepnbmC4
- https://www.youtube.com/watch?v=78thKVhcp-M
- https://learnabout-electronics.org/Digital/dig53.php
- https://ftthfiberoptic.com/what-is-lpo-optical-transceiver-module/
- https://www.youtube.com/watch?v=RCb8PsdipHI