Tick to Trade in High Frequency Trading - High Performance, Ultra-Low-Latency FPGA Packet Processing and Routing Pipeline

Yunus Emre Aslan
E-mail: yunusemre.aslan@yahoo.com

1 Introduction

An FPGA-based packet processing system was initially developed to establish baseline functionality, and then progressively optimized for ultra-low-latency performance in high-frequency trading (HFT) environments. The primary objective was to detect predefined patterns within high-speed packet streams and tag them in real time, all while operating across a dual-clock domain with minimal processing delay.

Project Link : `https://github.com/yeaslan/packet_processing_hft`

Tools : Modelsim + Vivado + Aldec (Riviera PRO) + Cocotb (Automated Self-Checking Testbenches - Regression - Python) + VSCode + SV Test-bench + Github Actions - Gitlab CI/CD + AWS

1.1 Problem Statement

Modern HFT systems require ultra-low-latency processing of network packets. The incoming data packets carry critical trading information which requires real-time analysis to identify specific (PACKET-TYPE, SYMBOL) patterns as they stream through the system. Upon detecting a match, the system tags each packet with a corresponding buffer ID. This buffer ID is subsequently used by the host system to determine the appropriate handling or processing route for the packet. The design challenge is to identify specific data patterns in incoming packets, encode these matches into a buffer value, and forward the processed data to a host interface. This must be achieved with strict timing guarantees and across different clock domains.

1.2 Design Goals

The design implements the following key processing steps to meet the system's performance and reliability requirements :

1. SOP/EOP-Based Framing : Packets are drawn using Start-of-Packet (SOP) and End-of-Packet (EOP) markers to ensure accurate framing throughout the data stream.

2. Pattern Matching at Defined Offsets : The system performs precise matching of (PACKET TYPE, SYMBOL) fields located at specific byte offsets within each packet.

3. Buffer ID Generation at EOP : Buffer IDs are generated only upon detection of the EOP signal, ensuring that the full packet content is considered before classification.

4. Cross-Clock Domain Reliability : A robust synchronization mechanism ensures glitch-free data transfer between asynchronous clock domains, preserving data integrity.

5. Low Latency and Timing Closure : The design is carefully optimized to minimize processing latency while achieving reliable timing closure across all critical paths.

2 System Design Overview

The project started with a traditional buffered architecture. It included a receiver interface packet buffer module to accumulate bytes using SOP/EOP framing, store the complete packet, and later process the buffered data using a pattern-detector module. Although this method ensured functionality, early work-load profiling revealed latency bottlenecks caused by post-packet processing, Finite State Machine (FSM) stalls and memory access delays. To address these limitations in the context of HFT where every nanosecond counts, the design restructured to a pipelined, streaming architecture. This new approach enables pattern detection in-flight, significantly reducing end-to-end latency, area and improving throughput under adaptive traffic profiles conditions.

2.1 Design Considerations – Functional to Ultra-Low-Latency

The project evolved through three progressively optimized designs aimed at reducing latency, improving timing closure, and maximizing FPGA efficiency for HFT workloads.

- Design 1: Baseline - Functional (buffered, FSM)
- Design 2: Streaming (simplified, FSM reduced)
- Design 3: HFT (streaming, pipelined, optimized for latency/Fmax)

Metric	**Design 1: Baseline**	**Design 2: Streaming**	**Design 3: HFT**
Latency (SOP $\rightarrow$ buffer)	variable (EOP + FSM)	$\sim$1-2 cycles	$\sim$1-2 cycles (sustained)
Pattern match timing	post-buffer	during stream	stream + pipelined + parallel
BRAM used	1 large mem[]	none	none
FSM stages	3+ states	1 counter	1 counter
Stalls (backpressure)	yes	possible	none (pipelined)
Clock crossing depth	async FIFO	async FIFO	async FIFO
Host replay delay	moderate	fast	ultra-fast ($\leq$ 2 clk)
Logic depth	medium	shallow	very shallow (Fmax Higher)
Power/perf efficiency	low	better	best per watt

Table 1: Design Progression Summary

- Design 1 began with a traditional FSM-buffered approach, where packet data was stored and processed after full reception. While functionally complete, this introduced stalls, increased BRAM usage, and delayed pattern detection.

- Design 2 transitioned to a lightweight streaming model, eliminating packet buffering and performing parsing during the data stream. This simplified control logic and improved throughput but still relied on discrete stages and control handoffs. The design optimized into a tightly streaming pipeline:
  - Developed a streaming-pattern-detector that operates during streaming, parses SOP/EOP, and performs real-time comparisons at offset locations
  - Matches are encoded into a buffer value exactly at EOP, in a fully pipelined, one-cycle deterministic path

  This shift eliminated full-packet storage, area usage and reducing decision latency by a full clock cycle. The system now streams packets directly from the receiver interface through a pipelined pattern detector, into an asynchronous FIFO, and finally to the host domain all without stalling or buffering delays. Entire path from SOP to pattern match to buffer generation kept inside the same module. This was to:
  - Avoid latency penalties from handshaking across multiple FSMs
  - Maintain a consistent pipeline for high Fmax and predictable timing
  - Keep pattern detection co-located with streaming logic to aid place-and-route
- Design 3, the final architecture, fully integrates pattern detection into the streaming data path using a pipelined, deterministic comparator tree. The streaming pattern detector parses SOP/EOP inline, performs real-time pattern matching at programmable byte offsets (e.g., PACKET TYPE OFFSET, SYMBOL OFFSET), and generates a buffer tag exactly on the EOP cycle. The final architecture minimizes critical path depth through pipelined and unrolled comparators, and avoids BRAM usage entirely. This match logic is fully pipelined, with unrolled comparisons and stage balancing for high fmax, to minimize cross-clock domain latency, an asynchronous FIFO bridges the clk net and faster clk host domains (2×), enabling low-latency replay into the host interface. Those are allowing decisions to be made in a single deterministic cycle (1–2 cycles).

The pattern detection, buffer tagging, and EOP logic are colocated in a single pipelined block to avoid boundary penalties and assist place-and-route tools with logic locality. This final design achieves:

- Deterministic latency (1–2 cycles SOP→buffer)
- No stalling or full-packet buffering
- Zero BRAM and shallow combinational depth
- High fmax via pipeline staging and comparator unrolling

Category	Optimization Method
Pipelining	Dual-stage match pipeline inside `streaming_pattern_detector`
Unrolling	Manual comparator unrolling for fixed patterns (buffer = 1-4)
CDC Handling	77-bit wide `async_fifo` (buffer + length + sop + eop + data)
Timing Closure	Buffer outputs at EOP only, minimizing toggles across clock edge
Floorplanning (Vivado)	Match logic (core) grouped in one quadrant for efficient placement
Latency Awareness	Buffer asserted only on EOP $\rightarrow$ zero extra cycle latency
Synthesis-Aware	All logic is FSM-free, stall-free, and placement-localized

Table 2: Optimization Techniques

- Excellent performance-per-watt for real-time HFT systems

It reflects industry best practices, as outlined in Xilinx HFT whitepapers: "In low-latency FPGA designs, module boundaries = latency penalties. Minimize crossing them where high-speed decisions are made."

Metric	Outcome
Latency	$\sim$1 cycle from match to buffer assertion
Area	Reduced due to removed buffers/FSMs
Fmax	Improved due to pipelining and unrolling
Scalability	Additional patterns easy to insert
Suitability for HFT	Excellent — low latency, deterministic
Timing Closure	Clean paths with set_max_delay scoped

Table 3: Final Outcome

2.2 Functional Architecture

- Receiver logic parses packet stream
- Streaming pattern detector identifies patterns live
- Encoded buffer ID pushed via async FIFO
- Host reads buffer-tagged packet with near-zero delay

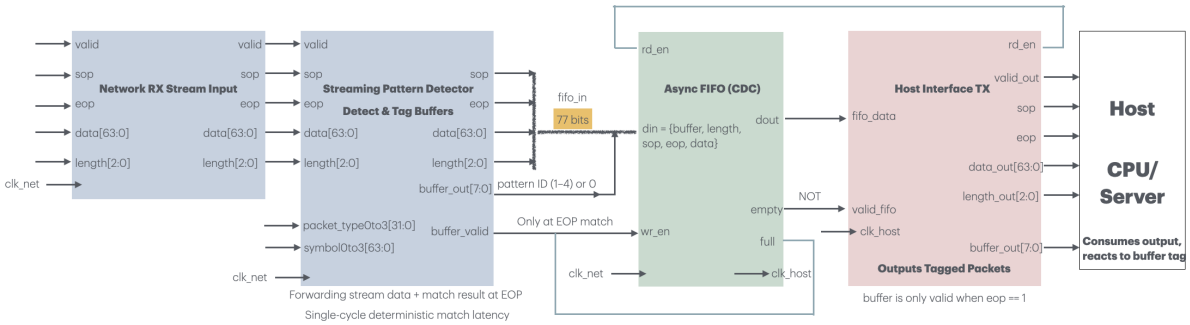


Figure 1: Top Level - Block Diagram

2.3 Design Flow Diagram

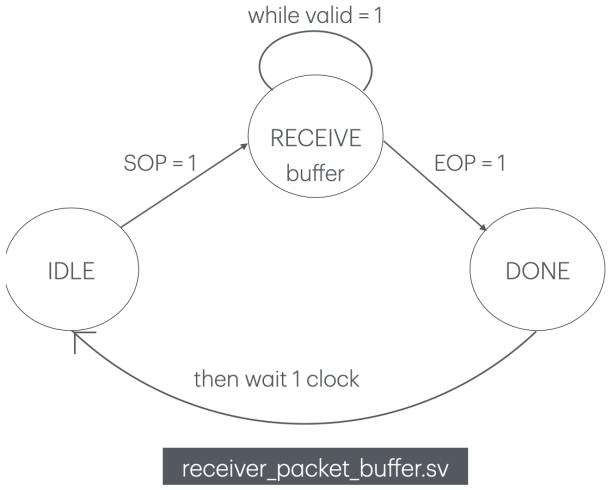
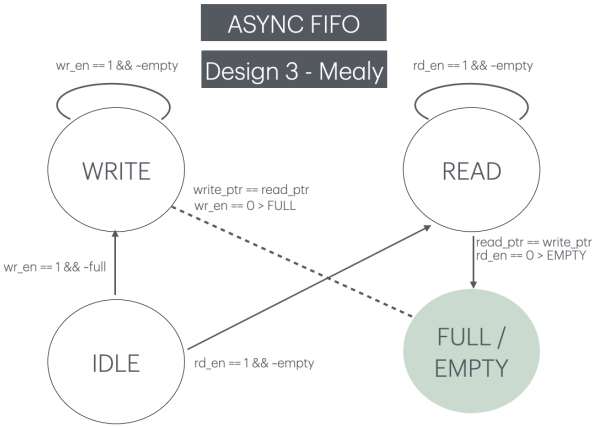


Figure 3: Design 1 - Receiver Interface (Moore)

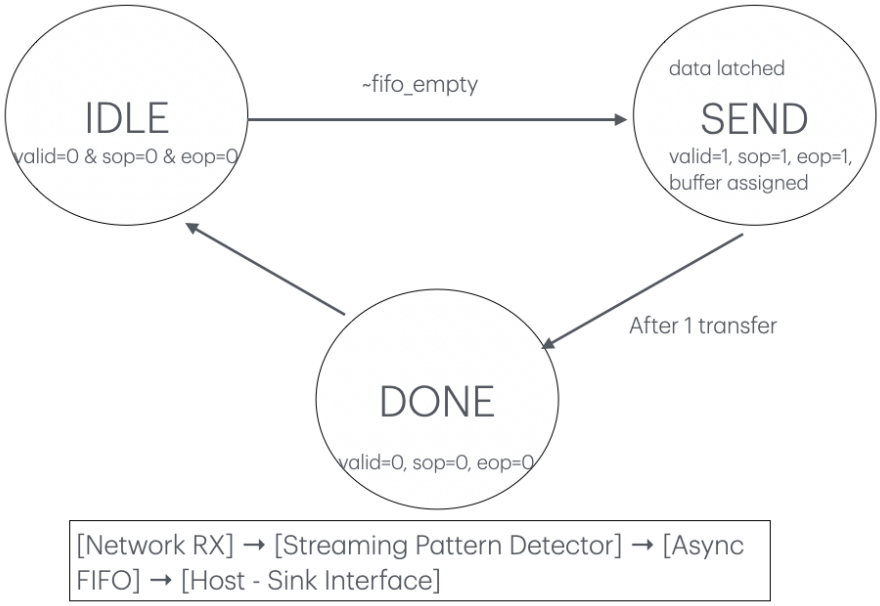


IDLE	No data written or read
WRITE	Data valid & state available
READ	Data valid & read enabled
FULL / EMPTY	Inferred from pointer comparisons / back pressure

Figure 2: Asyncronous FIFO

Current State	**Condition**	**Next State**
IDLE	`wr_en == 1 && ~full`	WRITE
IDLE	`rd_en == 1 && ~empty`	READ
WRITE	`wr_en == 1 && ~full`	WRITE
WRITE	`~wr_en == 0`	
READ	`rd_en == 1 && ~empty`	READ
READ	`~rd_en == 0`	
WRITE	Write causes `write_ptr == read_ptr` after increment	FULL
READ	Read causes `write_ptr == read_ptr` after increment	EMPTY

Table 4: Asynchronous FIFO - State Transition Table



IDLE	Wait for FIFO to have data
SEND	Send data + buffer (with SOP/EOP) to host
DONE	Finish transmission, clear valid signals

Figure 4: Design 3 - Host Interface - Moore

Figure 5: Pipeline

3 RTL Implementation

- RX_streaming_pattern_detector.sv

  Behaviour : FSM eliminated to reduce latency, in-flight matching, fully combinational + pipelined, pattern detection and buffer tagging are driven by valid, SOP, EOP.

- async_fifo.sv

  Behaviour : CDC FIFO, state tracking via pointers and flags, pointer-based buffer control, implicit FSM. Optional Improvements : Gray coded pointers and sync stages, almost full/almost empty flags, built-in FIFO IPs

- host_interface.sv

  Behaviour : Hand-off logic, managing output, no FSM

- topmodule.sv : Structural integration

3.1 Design Decisions

- In-Streaming match, avoid state delays
- Unrolled Match logic + pipelined
- Buffer asserted only at EOP for output alignment
- CDC logic designed to minimize metastability risk

4 Verification and Testbenches

- System Verilog tests for the Isolated Block Level + Integration Level
- Cocotb tests for isolated module + full pipeline (to be completed)
- Edge cases: Partial-multiple packets, Invalid headers, Resets mid-packet, Randomized (full coverage to be completed)
- ModelSim + VCD output for waveform validation
- Automated Verification (to be completed)

4.1 Coverage and Limitations

Test coverage includes:

- Functionality
- Match/no match cases
- CDC behavior
- Latency measurements

Limitations: Static pattern loading; dynamic matching not implemented.

4.2 Checks

- Receiver interface inputs (valid, data, SOP, EOP, length) are correctly parsed in the streaming path.
- Pattern matching uses: A fixed 4-byte PACKET TYPE at a programmable offset, an 8-byte SYMBOL at a second offset
- Both comparisons are done during streaming, not post-buffering
- When a pattern match occurs, a unique buffer value (1–4) is generated.
- If no match occurs, buffer = 0
- Buffer is asserted at the same time as EOP = 1 in the host domain, implemented with latency alignment
- The clk host domain runs at 2× clk_net and async_FIFO crossing is clean
- Host interface output includes all required signals, and signal timing respects the valid + EOP + buffer alignment

5 Conclusion

The final design is optimized for ultra-low-latency packet tagging on FPGAs with a pipelined match engine, async FIFO for CDC, and minimal resource usage. Thus, it achieves a performance trade-off. Future design improvements are planned and will be realized.

5.1 Key Results

- 1-2 cycle latency from SOP to buffer
- Timing closure: 500+ MHz possible in Virtex UltraScale+ (to be tested more in Vivado Synthesis, Implementation, Floorplan)
- Streaming match detection at full-line rate
- Zero stalls, no BRAM, minimal logic