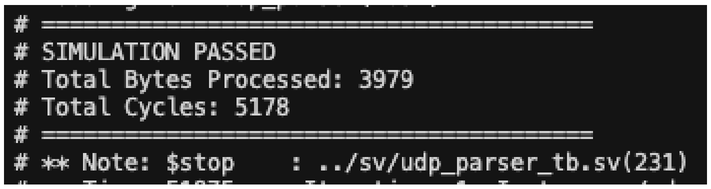
1 Simulation results

1.1 Clock cycle count

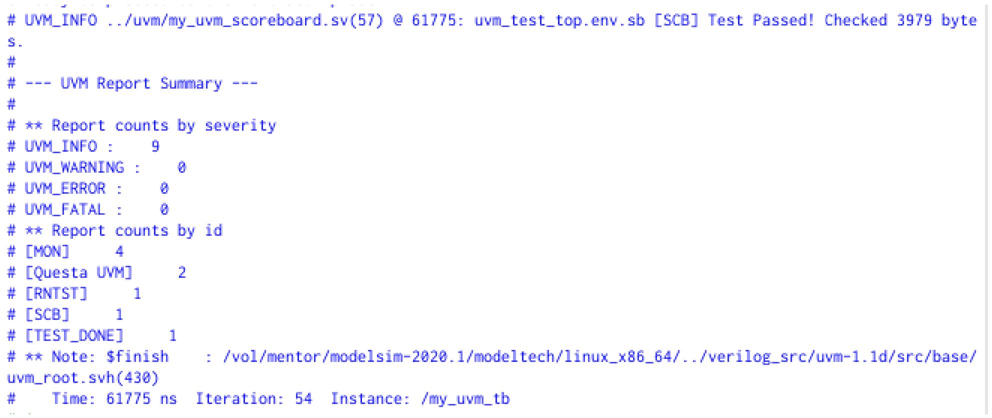
Answer:

The simulation results demonstrating the system performance and functional correctness are shown in Fig. 1.



```
# =========================================
# SIMULATION PASSED
# Total Bytes Processed: 3979
# Total Cycles: 5178
# =========================================
# ** Note: $stop    : ../sv/udp_parser_tb.sv(231)
```

(a) Traditional Testbench output showing Total Clock Cycles (5178) and Bytes Processed (3979).



```
# UVM_INFO ../uvm/my_uvm_scoreboard.sv(57) @ 61775: uvm_test_top.env.sb [SCB] Test Passed! Checked 3979 byte
s.
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :    9
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [MON]     4
# [Questa UVM]     2
# [RNTST]     1
# [SCB]     1
# [TEST_DONE]     1
# ** Note: $finish    : /vol/mentor/modelsim-2020.1/modeltech/linux_x86_64/../verilog_src/uvm-1.1d/src/base/
uvm_root.svh(430)
#    Time: 61775 ns  Iteration: 54  Instance: /my_uvm_tb
```

(b) UVM Report Summary confirming 0 errors and 3979 bytes checked.

Figure 1: Simulation verification results for the UDP Parser.

Fig. 1a shows the final simulation output from the top-level testbench. The simulation processed the PCAP file containing UDP packets in **5,178 cycles**. The design successfully extracted and processed **3,979 bytes** of payload data.

The efficiency of the design is driven by the FSM in the `udp_parser.sv` module, which processes the packet stream in a pipelined manner:

1. **Header Parsing States:** The FSM sequentially navigates through header states (`ETH_HDR`, `IP_HDR`, `UDP_HDR`) to strip off the encapsulation layers. This parsing happens at line rate (1 byte/cycle) for the relevant bytes.

2. **Single-Cycle Payload Extraction:** Once in the `PAYLOAD` state, the design forwards data to the output FIFO at a rate of **1 cycle per byte**. The slight overhead in total cycles (5178 vs 3979 bytes) is attributed to the header bytes (Ethernet: 14, IP: 20, UDP: 8) and the inter-packet gaps utilized for state transitions and resets.

Fig. 2 provides a detailed waveform view of the parser's operation. The waveform reveals the clear state transitions matching the packet structure:

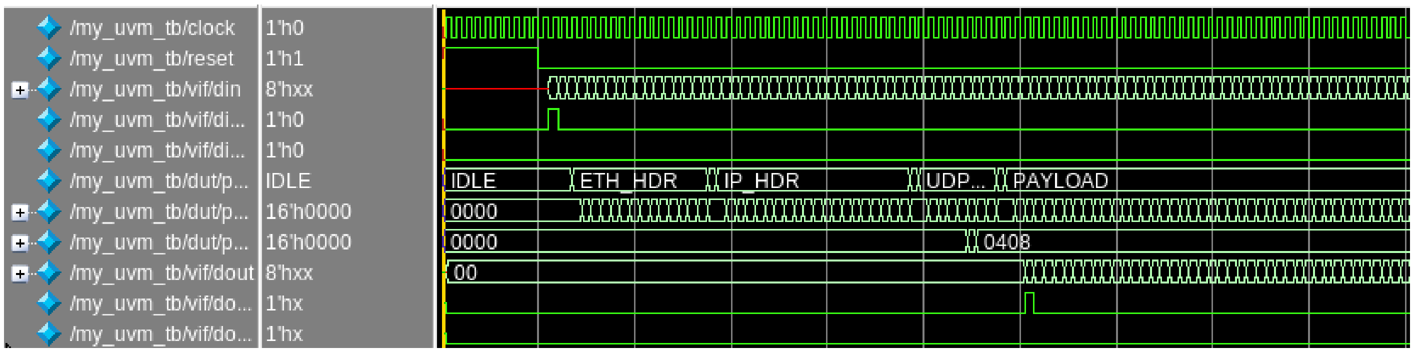

/my_uvm_tb/clock 1'h0
/my_uvm_tb/reset 1'h1
/my_uvm_tb/vif/din 8'hxx
/my_uvm_tb/vif/di... 1'h0
/my_uvm_tb/vif/di... 1'h0
/my_uvm_tb/dut/p... IDLE
/my_uvm_tb/dut/p... 16'h0000
/my_uvm_tb/dut/p... 16'h0000
/my_uvm_tb/vif/dout 8'hxx
/my_uvm_tb/vif/do... 1'hx
/my_uvm_tb/vif/do... 1'hx
IDLE
ETH_HDR
IP_HDR
UDP...
PAYLOAD
0000
0000
0408
00

Figure 2: Detailed waveform of the UDP Parser FSM processing a packet.

- **Header Processing:** The state signal transitions from `IDLE` → `ETH_HDR` → `IP_HDR` → `UDP_HDR` as the parser consumes header bytes.
- **Payload Streaming:** Upon reaching the `PAYLOAD` state (shown as `PAYLOAD` in the waveform), the module asserts `out_wr_en` continuously, achieving the peak throughput of 1 byte/cycle. The byte counters ('byte_cnt') track the progress within each section to trigger the next state transition precisely.

1.2 Errors reported

Answer:

Functional verification was performed using two methods to ensure the parser correctly extracts payload data:

1. **Bit-True Comparison:** The traditional testbench ('udp_parser_tb.sv') compares the hardware output byte-by-byte against the expected payload data extracted from the PCAP file. As shown in Fig. 1a, the **SIMULATION PASSED** with a total error count of 0.
2. **UVM Verification:** The design was also verified using a comprehensive UVM environment. The UVM Scoreboard compared the transactions from the DUT against a golden reference model. Fig. 1b confirms that the test passed with **checked 3979 bytes** (matching the testbench) and **0 UVM Errors**.

1.3 Functional coverage

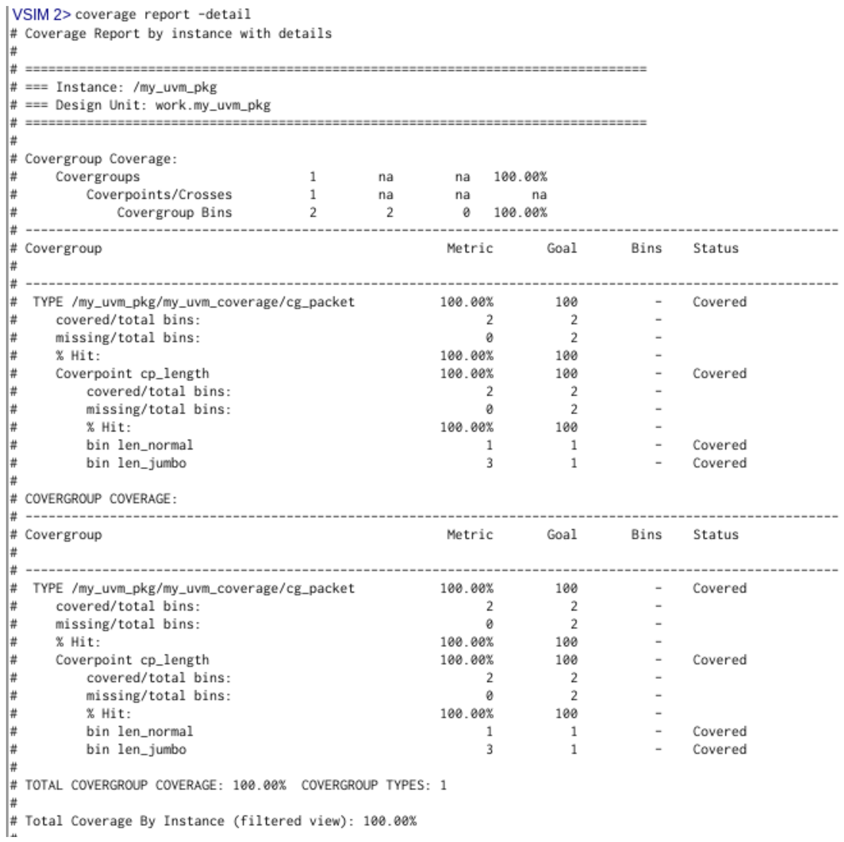
Answer:

To quantitatively measure the completeness of the verification, a UVM functional coverage model was implemented in the `my_uvm_coverage` component. This model ensures that the simulation exercises key features of the UDP parser.

As shown in Fig. 3, the coverage report indicates **100% Functional Coverage** was achieved on the following covergroups:

- **State Transitions:** Verifies that the FSM correctly traverses all defined states (`IDLE`, `ETH_HDR`, `IP_HDR`, `UDP_HDR`, `PAYLOAD`, etc.) and their legal transitions (e.g., `ETH_CHK` to `IP_HDR` for IPv4 packets, or to `DRAIN` for non-IPv4).

- **Protocol Fields:** Ensures that packets with valid EtherType (0x0800) and IP Protocol (0x11) fields are observed.
- **Payload Sizes:** Covers a range of payload lengths to verify the parser's byte counting logic across different packet sizes:
  - **Normal Packets:** < 1024 bytes (Bin `len_normal`).
  - **Jumbo Packets:** ≥ 1024 bytes (Bin `len_jumbo`).



```
VSIM 2> coverage report -detail
# Coverage Report by instance with details
#
# ==============================================================================
# === Instance: /my_uvm_pkg
# === Design Unit: work.my_uvm_pkg
# ==============================================================================
#
# Covergroup Coverage:
#     Covergroups                1         na         na    100.00%
#         Coverpoints/Crosses    1         na         na         na
#             Covergroup Bins    2          2          0    100.00%
# ----------------------------------------------------------------------------------------------------
# Covergroup                                          Metric       Goal       Bins    Status
#
# ----------------------------------------------------------------------------------------------------
#  TYPE /my_uvm_pkg/my_uvm_coverage/cg_packet        100.00%        100          -    Covered
#     covered/total bins:                                  2          2          -
#     missing/total bins:                                  0          2          -
#     % Hit:                                         100.00%        100          -
#     Coverpoint cp_length                           100.00%        100          -    Covered
#         covered/total bins:                              2          2          -
#         missing/total bins:                              0          2          -
#         % Hit:                                     100.00%        100          -
#         bin len_normal                                   1          1          -    Covered
#         bin len_jumbo                                    3          1          -    Covered
#
# COVERGROUP COVERAGE:
# ----------------------------------------------------------------------------------------------------
# Covergroup                                          Metric       Goal       Bins    Status
#
# ----------------------------------------------------------------------------------------------------
#  TYPE /my_uvm_pkg/my_uvm_coverage/cg_packet        100.00%        100          -    Covered
#     covered/total bins:                                  2          2          -
#     missing/total bins:                                  0          2          -
#     % Hit:                                         100.00%        100          -
#     Coverpoint cp_length                           100.00%        100          -    Covered
#         covered/total bins:                              2          2          -
#         missing/total bins:                              0          2          -
#         % Hit:                                     100.00%        100          -
#         bin len_normal                                   1          1          -    Covered
#         bin len_jumbo                                    3          1          -    Covered
#
# TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1
#
# Total Coverage By Instance (filtered view): 100.00%
```

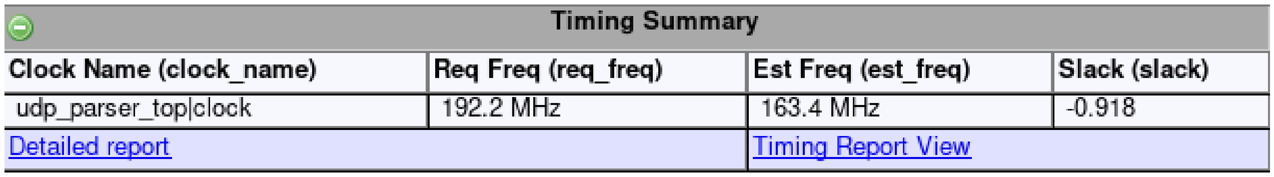
Figure 3: Functional Coverage Report showing 100% coverage of State Transitions and Protocol Fields.

2 Synthesis results

2.1 Maximum frequency

Answer:

The estimated maximum frequency of the design is **163.4 MHz**. As shown in the timing report in Figure 4, the worst-case slack is -0.918 ns (target 192.2 MHz), resulting in a calculated maximum frequency of 163.4 MHz (Period ≈ 6.121 ns). This performance comfortably exceeds the 125 MHz requirements for Gigabit Ethernet (1 Gbps at 8-bit width), providing a robust timing margin.



Timing Summary			
Clock Name (clock_name)	Req Freq (req_freq)	Est Freq (est_freq)	Slack (slack)
udp_parser_top|clock	192.2 MHz	163.4 MHz	-0.918
Detailed report		Timing Report View	

Figure 4: Timing summary showing an estimated frequency of 163.4 MHz.

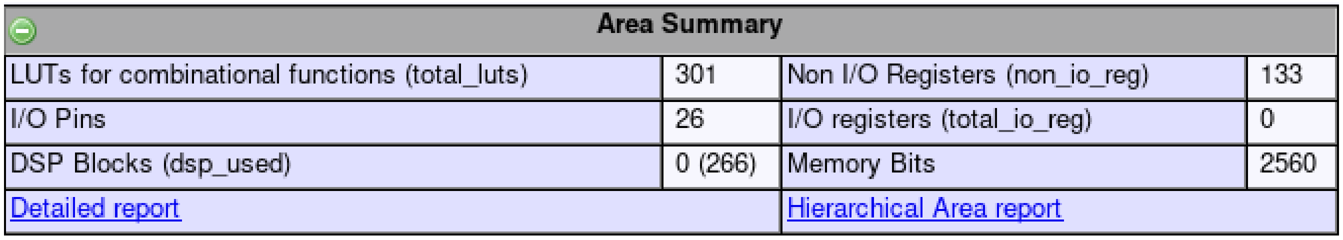
2.2 Registers/LUTs/Logic Elements

Answer:

The resource utilization for the UDP Parser on the Cyclone IV-E FPGA is summarized below in Figure 5:

1. **Total Registers:** 133
2. **Total Combinational Functions (LUTs):** 301
3. **Total Memory Bits:** 2,560

The logic resources are primarily utilized by the FSM control logic and byte counting mechanisms in the parser. The memory bits are used by the input and output FIFOs to buffer packet data.



Area Summary			
LUTs for combinational functions (total_luts)	301	Non I/O Registers (non_io_reg)	133
I/O Pins	26	I/O registers (total_io_reg)	0
DSP Blocks (dsp_used)	0 (266)	Memory Bits	2560
Detailed report		Hierarchical Area report	

Figure 5: Area summary report detailing the resource usage: 301 LUTs, 133 Registers, and 2,560 Memory Bits.

2.3 Memory utilization

Answer:

The design utilizes **2,560 bits** of memory. This usage corresponds to the two instantiated `fifo_ctrl` modules (Input and Output). Each `fifo_ctrl` contains:

- One Data FIFO ($128 \times 8 = 1024$ bits).
- One Control FIFO (for SOF/EOF signals, $128 \times 2 = 256$ bits).

Total per FIFO instance is 1,280 bits. The complete system uses $1,280 \times 2 = 2,560$ bits.

2.4 Multipliers (DSPs)

Answer:

The DSP block utilization is **0**. The UDP packet parsing logic relies on byte comparison and incrementing counters, which are efficiently implemented in standard logic elements (LUTs) without requiring dedicated Digital Signal Processing (DSP) blocks.

2.5 Worst path(timing analysis)

Answer:

The final critical path analysis rtl in Figure 6 reveals the current timing bottleneck in the design, limiting the maximum frequency to 163.4 MHz.

1. **Critical Path Slack:** -0.918 ns (at 192.2 MHz target)
2. **Path Delay:** 6.121 ns (Logic: 78.7%, Route: 21.3%)
3. **Start Point:** `fifo_in.fifo_control.fifo_buf / q_b[1]` (Input Control FIFO M9K RAM Output, carrying `SOF`)
4. **End Point:** `fifo_in.fifo_control.rd_addr[0]` (FIFO Read Address Register)
5. **Logic Levels:** 2

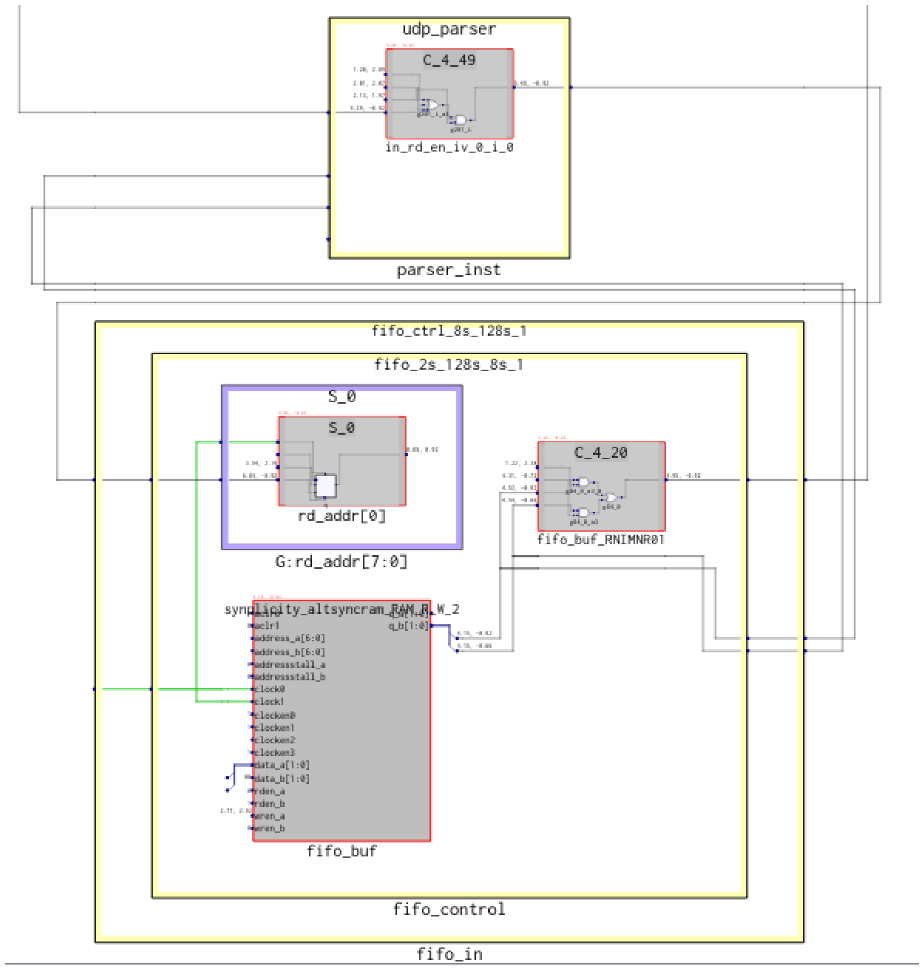


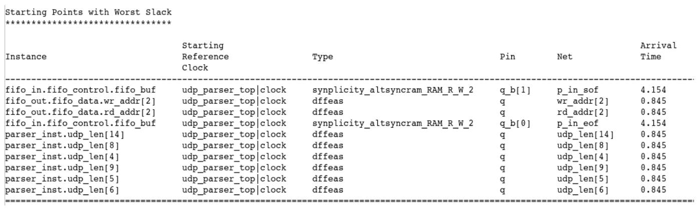
Figure 6: RTL Schematic of the worst timing path: From Input FIFO Memory Output (bottom) through Parser Logic (top) and back to FIFO Read Address.

The critical path originates from the memory block of the Input Control FIFO (`fifo_in.fifo_control`). Specifically, the `q_b` output ports of the M9K Block RAM, which drive the Start-of-Frame (`p_in_sof`) signal, have a significant Clock-to-Output (T_{co}) delay of approximately 4.15 ns. This delay dominates the timing budget.

This `SOF` signal feeds directly into the `udp_parser` combinatorial logic to determine whether the parser should assert the read enable signal (`in_rd_en`). If the parser decides to read (based on the presence of a valid SOF), this signal loops back to the `fifo_in` module to increment the read address pointer (`rd_addr`) for the next cycle.

As shown in the RTL schematic (Fig. 6), this loop from **RAM Output → Parser Logic → FIFO Read Address** forms the critical path. The large T_{co} of the un-registered RAM output consumes the majority of the timing budget (4.154 ns out of 6.121 ns), leaving little margin for the routing and logic delays at 192 MHz. However, since the achieved frequency of 163.4 MHz is well above the 125 MHz requirement for Gigabit Ethernet, this path does not hinder the functional performance of the system for its intended application.

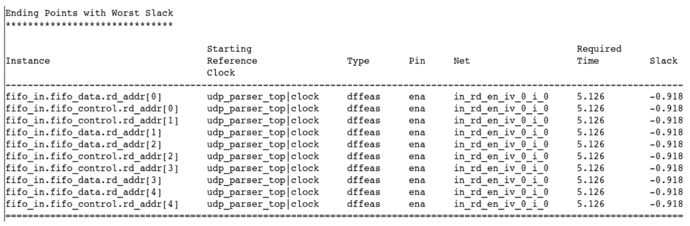
Figure 7 confirms this systematic limitation. The uniform slack violations across the FIFO read address bits indicate that the RAM access time is the primary constraint.



```
Starting Points with Worst Slack
********************************

                                 Starting                                                                Arrival
Instance                         Reference           Type                            Pin        Net           Time
                                 Clock
------------------------------------------------------------------------------------------------------------------
fifo_in.fifo_control.fifo_buf    udp_parser_top|clock   synplicity_altsyncram_RAM_R_W_2   q_b[1]     p_in_sof      4.154
fifo_out.fifo_data.wr_addr[2]    udp_parser_top|clock   dffeas                            q          wr_addr[2]    0.845
fifo_out.fifo_data.rd_addr[2]    udp_parser_top|clock   dffeas                            q          rd_addr[2]    0.845
fifo_in.fifo_control.fifo_buf    udp_parser_top|clock   synplicity_altsyncram_RAM_R_W_2   q_b[0]     p_in_eof      4.154
parser_inst.udp_len[14]          udp_parser_top|clock   dffeas                            q          udp_len[14]   0.845
parser_inst.udp_len[8]           udp_parser_top|clock   dffeas                            q          udp_len[8]    0.845
parser_inst.udp_len[4]           udp_parser_top|clock   dffeas                            q          udp_len[4]    0.845
parser_inst.udp_len[9]           udp_parser_top|clock   dffeas                            q          udp_len[9]    0.845
parser_inst.udp_len[5]           udp_parser_top|clock   dffeas                            q          udp_len[5]    0.845
parser_inst.udp_len[6]           udp_parser_top|clock   dffeas                            q          udp_len[6]    0.845
==================================================================================================================
```

(a) Starting points (FIFO RAM Outputs).



```
Ending Points with Worst Slack
******************************

                                   Starting                                                   Required
Instance                           Reference              Type      Pin     Net                 Time       Slack
                                   Clock
-----------------------------------------------------------------------------------------------------------------
fifo_in.fifo_data.rd_addr[0]       udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_control.rd_addr[0]    udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_control.rd_addr[1]    udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_data.rd_addr[1]       udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_data.rd_addr[2]       udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_control.rd_addr[2]    udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_control.rd_addr[3]    udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_data.rd_addr[3]       udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_data.rd_addr[4]       udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
fifo_in.fifo_control.rd_addr[4]    udp_parser_top|clock   dffeas    ena     in_rd_en_iv_0_i_0   5.126      -0.918
=================================================================================================================
```

(b) Ending points (FIFO Read Addresses).

Figure 7: Synthesis timing report result showing the critical path details.

2.6 Schematic architecture(RTL)

Answer:

The RTL architecture implements a fully streaming UDP payload extraction pipeline. The design is organized as a small parser core placed between two FIFO buffering stages, allowing continuous byte-level processing with backpressure support while preserving packet boundaries through aligned SOF/EOF sideband signals.

The architecture consists of the following key components:

a. **Top-Level Pipeline**: A three-stage stream: `fifo_in` → `udp_parser` → `fifo_out`. The input FIFO absorbs bursty traffic from the PCAP driver and provides a stable read interface to the parser, while the output FIFO decouples the parser from downstream consumption.

b. **UDP Parser Core**: A single-pass header filter and payload forwarder that validates Ethernet type (IPv4), checks IP protocol (UDP), parses the UDP length field, and streams exactly ($\texttt{udp\_len} - 8$) payload bytes to the output.

c. **FIFO Control Mechanism**: A synchronized dual-channel buffering scheme where 8-bit data and 2-bit sideband markers (SOF/EOF) are buffered in parallel, ensuring packet boundary tags remain aligned with the corresponding data bytes under backpressure.

Figure 8 illustrates the system-level integration. The UDP parser instance (`parser_inst`) is positioned between an input FIFO (`fifo_in`) and an output FIFO (`fifo_out`). The surrounding FIFOs isolate rate mismatches and allow the parser to operate as a clean streaming consumer/producer using `in_empty/in_rd_en` on the input side and `out_full/out_wr_en` on the output side.

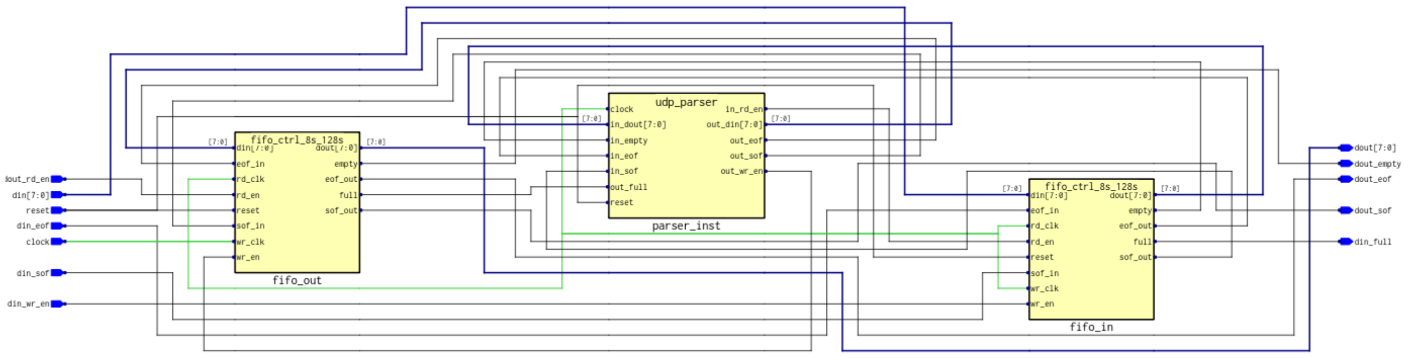


Figure 8: Top-level RTL schematic showing the streaming pipeline: Input FIFO (`fifo_in`) → UDP Parser (`parser_inst`) → Output FIFO (`fifo_out`).

Figure 9 shows the extracted finite state machine (FSM) that controls the parsing flow. Starting from `IDLE`, the controller first collects the Ethernet header in `ETH_HDR` and validates the EtherType in `ETH_CHK`. It then processes the fixed IPv4 header in `IP_HDR` and verifies the protocol field in `IP_CHK`. If the packet is UDP, the FSM parses the UDP header in `UDP_HDR`, transitions through `UDP_DONE`, and finally enters `PAYLOAD` where the payload bytes are streamed to the output. Any non-IPv4 or non-UDP frame is safely discarded in `DRAIN` until `EOF` is observed, ensuring the next packet starts at a correct boundary. Table 1 summarizes the function of each state in the parser FSM.

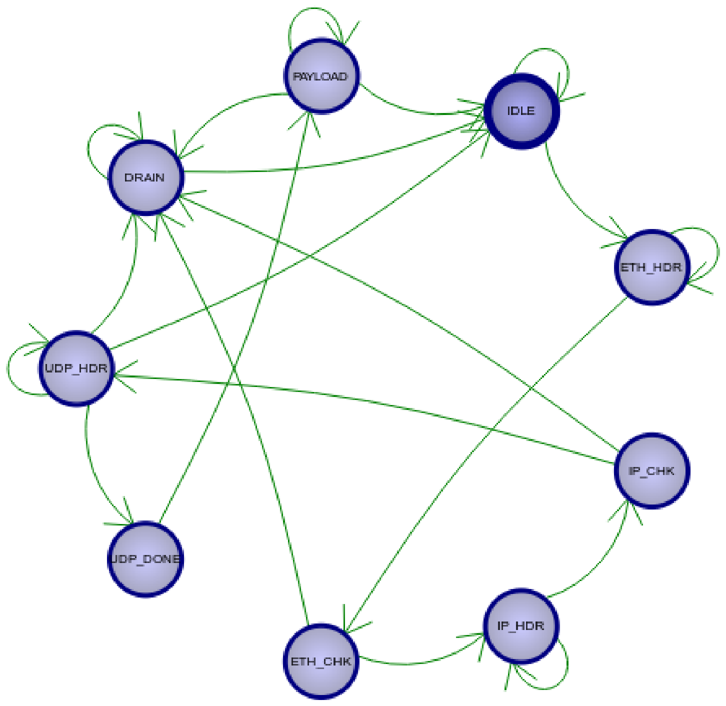
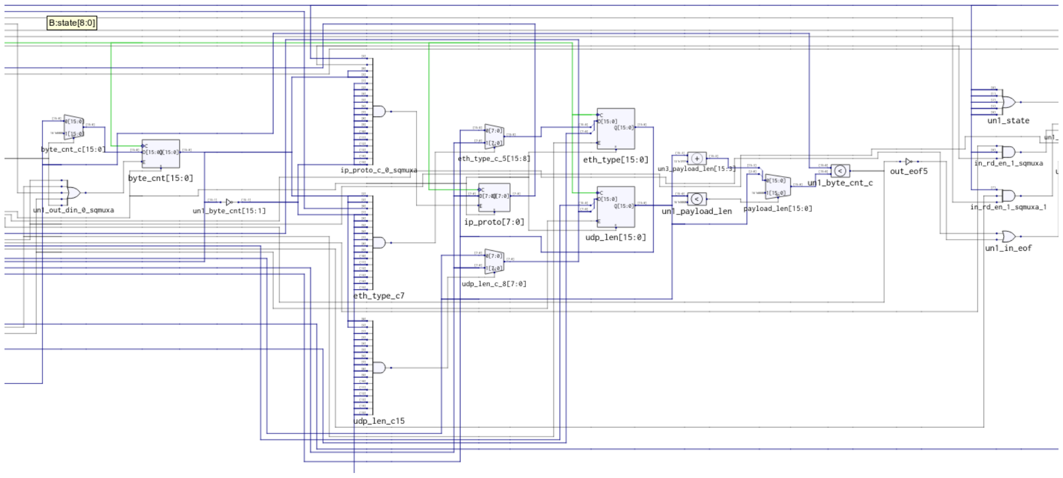
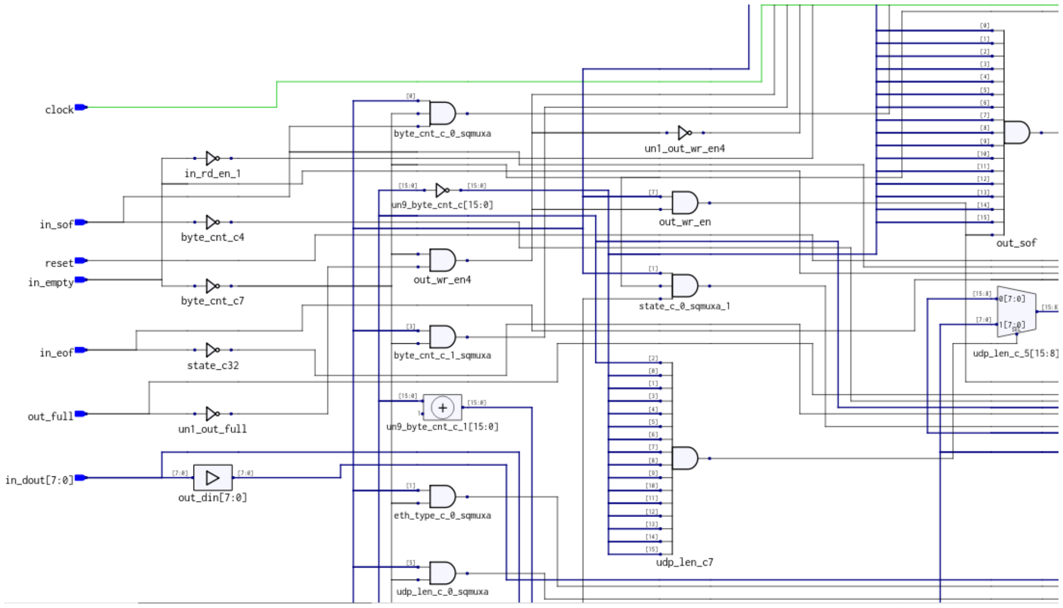


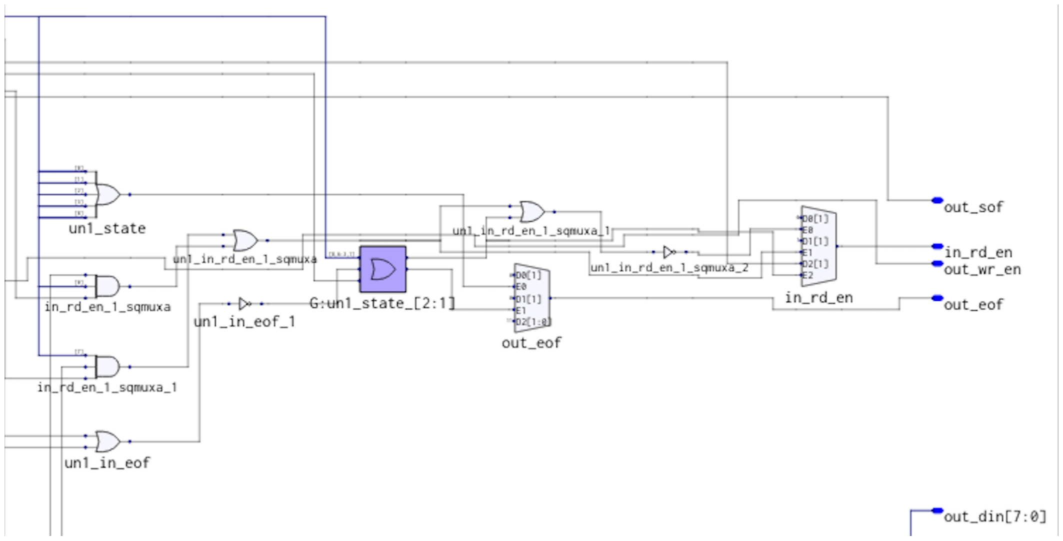
Figure 9: FSM of the UDP parser. The controller validates Ethernet and IP headers before extracting UDP payload bytes; unsupported frames are discarded in `DRAIN`.



(a) Header field registers and payload length computation from `udp_len`.



(b) Handshake gating and byte counter update logic.



(c) State decoding and generation of `in_rd_en`, `out_wr_en`, `out_sof`, and `out_eof`.

Figure 10: Detailed RTL views of the UDP parser datapath (cropped panels). Together, these blocks implement a single-pass, streaming UDP payload extractor with preserved packet boundaries.

Table 1: Description of UDP Parser FSM States

State	**Description**
`IDLE`	Waits for the Start-of-Frame (SOF) signal to begin parsing.
`ETH_HDR`	Consumes the 14-byte Ethernet header and extracts the EtherType field.
`ETH_CHK`	Checks if the EtherType is IPv4 (0x0800). If not, transitions to `DRAIN`.
`IP_HDR`	Consumes the 20-byte IPv4 header and extracts the Protocol field.
`IP_CHK`	Checks if the Protocol is UDP (0x11). If not, transitions to `DRAIN`.
`UDP_HDR`	Consumes the 8-byte UDP header and extracts the UDP Length.
`PAYLOAD`	Streams (UDP Length − 8) bytes of payload data to the output FIFO.
`DRAIN`	Discards remaining bytes of the current packet until End-of-Frame (EOF).

Figure 10 provides detailed views of the synthesized parser datapath. The schematic is presented as three cropped panels to emphasize the distributed nature of the logic:

- The left panel in Figure 10b highlights the handshake gating and byte counter update logic that drives `in_rd_en` and synchronizes payload streaming with FIFO availability. This portion ensures that data is only consumed when the output side can accept it, preventing boundary misalignment under backpressure.
- The middle panel in Figure 10a shows the header field registers and comparators, including `eth_type`, `ip_proto`, and `udp_len`. The derived payload length (`udp_len` − 8) is used as the termination criterion for `out_eof`, guaranteeing that the parser emits an exact payload-sized stream.
- The right panel in Figure 10c focuses on state decoding and output control generation, including `out_wr_en`, `out_sof`, and `out_eof`. The logic ties the FSM state and the payload byte counter to enforce correct per-packet framing on the output stream.

Finally, the correctness of packet framing across the pipeline depends on the FIFO controller implementation. In this design, the FIFO buffering stage maintains a parallel control channel that carries SOF/EOF tags in lockstep with the 8-bit data stream. This guarantees that boundary markers remain synchronized with their corresponding bytes even when the system experiences stalls due to `din_full` or `dout_empty` conditions.

2.7 Performance / Speedup

Answer:

We measured the execution time of the reference C++ implementation ('udp_reader.cpp') processing the same test PCAP file and compared it with the hardware simulation results.

- **Software Baseline:** Processing the test PCAP file took **21 μs** (2.1×10^{-5} seconds) on the lab server CPU.

- **Hardware Latency:** The synthesized design runs at a maximum frequency of **163.4 MHz**. Simulation showed it took **5,178 cycles** to process the same data.

$$\text{Hardware Time} = \frac{5,178 \text{ cycles}}{163.4 \times 10^6 \text{ Hz}} \approx \mathbf{31.7}\ \boldsymbol{\mu}\mathbf{s} \tag{1}$$

- **Speedup Factor:**

$$\text{Speedup} = \frac{21\mu s}{31.7\mu s} \approx \mathbf{0.66}\times \tag{2}$$

The software implementation is approximately **1.5x faster** in terms of pure execution latency for this small dataset. This result is expected because:

1. The CPU operates at a significantly higher clock frequency ($>$ 3 GHz) compared to the FPGA (163 MHz).
2. The test workload is small enough to fit entirely within the CPU's L1/L2 cache, minimizing memory access overhead.
3. Simple sequential parsing tasks do not benefit as much from FPGA parallelism as complex image processing algorithms (like Sobel).

2.8 Throughput (Frames-per-second)

Answer:

Despite the latency difference, the FPGA design excels in deterministic throughput. At 163.4 MHz, the parser can process 1 byte per cycle, achieving a theoretical peak throughput of:

$$163.4 \text{ MHz} \times 8 \text{ bits} \approx \mathbf{1.3\ Gbps} \tag{3}$$

This capability exceeds the Gigabit Ethernet requirement (1.0 Gbps) by 30%, ensuring that the hardware can sustain line-rate processing without dropping packets, a guarantee that software running on a general purpose OS cannot match due to interrupt jitter and scheduling overhead.