# Assignment 5: UDP Network Parser

**UDP Parser**

In this assignment you will be implementing a UDP network packet parser  in RTL to extract the datagram from the packet.  You will be implementing a streaming architecture using FIFOs. The FIFO architecture should be augmented to include synchronization signals start-of-frame (sof) and end-of-frame (eof) on both the read/write paths.

Begin by downloading and installing Wireshark.  You should be able to open up the provided pcap file in Wireshark to view the UDP packets in the file. Make sure you understand the packet structure.

Next, compile and run the udp_parser.cpp code provide to you. The program will read in the pcap network file and output the datagrams from the UDP packets.  A sample test.pcap and resulting test.txt files are provided for you.

Implement the UDP parser in SystemVerilog using an FSM. Each state should read individual UDP packet segments, validating the packet as needed. Your implementation must use UVM for verification.

Important note about the PCAP files: each file includes both global file headers (24 bytes) and individual packet headers (16 bytes). These headers should be extracted in the testbench. However, you should use the packet header to determine the size of the packet so that you can appropriately set the start and end of frame signals. The packet length is specified in bytes 8-12 in the packet header, as shown below. You can learn more about the Wireshark packet formats here.

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp seconds */
        guint32 ts_usec;        /* timestamp microseconds */
        guint32 incl_len;       /* number of octets of packet saved in file */
        guint32 orig_len;       /* actual length of packet */
} pcaprec_hdr_t;
```

**Goals:**

- Create a streaming architecture that parses UDP packets and extracts the datagram.
- Create a SystemVerilog architecture for the UDP parser using an FSM.

- Augment the FIFO architecture with SOF and EOF signals to synchronize start and end of frames.
- Implement the UVM simulation architecture to verify your design is bit-true accurate.
- Implement UVM functional coverage to report on your RTL design coverage
- Report on performance results and improvements

**Setup:**

- Compile and run the C-code to generate the pcap output for your simulation.
  - **g++ udp_parser.cpp -o upd_parser**
  - **./udp_parser test.pcap > test_output.txt**
- Download and install Wireshark. Open up the test. pcap file in Wireshark to view the UDP packets in the file.

**Implementation:**

- Create a new module called fifo_ctrl with sof and eof signals for both the read and write paths. The simplest way to do this is to instantiate 2 FIFOs in the fifo_ctrl module: one 8-bit data fifo, and one 2-bit control fifo.
- Develop a UDP parser using an FSM to extract the datagram. You will need to extract the datagram from the UDP packet by parsing the packet layers as follows:

  1. Ethernet frame
  2. IP frame
  3. UDP frame

**Verification:**

- Create a new UVM model that reads in the 8-bit wide pcap data and outputs the data to a FIFO. The UVM Driver should extract the pcap headers and only stream the UDP packet data. Likewise, it should read the datagram from the output FIFO and compare results against the known data from your C program. You should do a byte-wise comparison in simulation. Use the test_output.txt file generated from the C code to verify your simulation.
- Implement UVM functional coverage and report on states covered and successful packets.
- Create a udp_parser_sim.do file that compiles the SystemVerilog and UVM files, sets up the wave form, and runs the simulation.

- Run the simulation in modelsim and verify the design is bit-true accurate.
- Compile the design in Synplify Premier to get high-level resource results. You don't need to go through Place and Route. You should not synthesize the UVM files.

**Reporting:**

- Submit a PDF file with simulation and synthesis results with the following:
    - Simulation results:
        - Clock cycle count
        - Errors reported (if any)
        - Functional coverage
    - Synthesis results (some might not apply):
        - Maximum frequency
        - Registers / LUTs / Logic Elements
        - Memory utilization
        - Multipliers (DSPs)
        - Worst path (timing analysis)
        - Schematic architecture (RTL)
        - Performance / Speedup
        - Throughput (Frames-per-second)
        - Include pictures and a brief description of your architecture from Synplify Premier.

Turn in your designs with the report. Your file should be zipped, and should include the SystemVerilog files, synthesis, simulation, and input/output files. **PLEASE MAKE SURE TO REMOVE ALL THE INTERMEDIATE WORK DIRECTORIES.**