MDP3.0 Tickerplant and Order Book Generator

Final Report - CSEE 4840 Spring 2014

Daron Lin (dl2573) Jonathan Liu (jl3516)
Giovanni Ortuno (gdo2103) Mirza Ali (maa2205)

Introductions

The MDP3.0 tickerplant generates order-books using the incremental market refresh MDP3.0 datafeed and the parallelizable attributes of FPGAs. We have created a hardware alternative to software book building, significantly increasing the speed and efficiency with which market data can be processed and broadcast.

Project Overview

Our tickerplant will take in data in MDP3.0 form, a lower latency and less CPU intensive alternative to FAST that provides not only increased timestamp granularity, but also increased Market state granularity. Once received, the data will be packetized, passed on to the parser, and then sent to the correct location for tracking with a FIFO queue between each stage.

Software Overview:

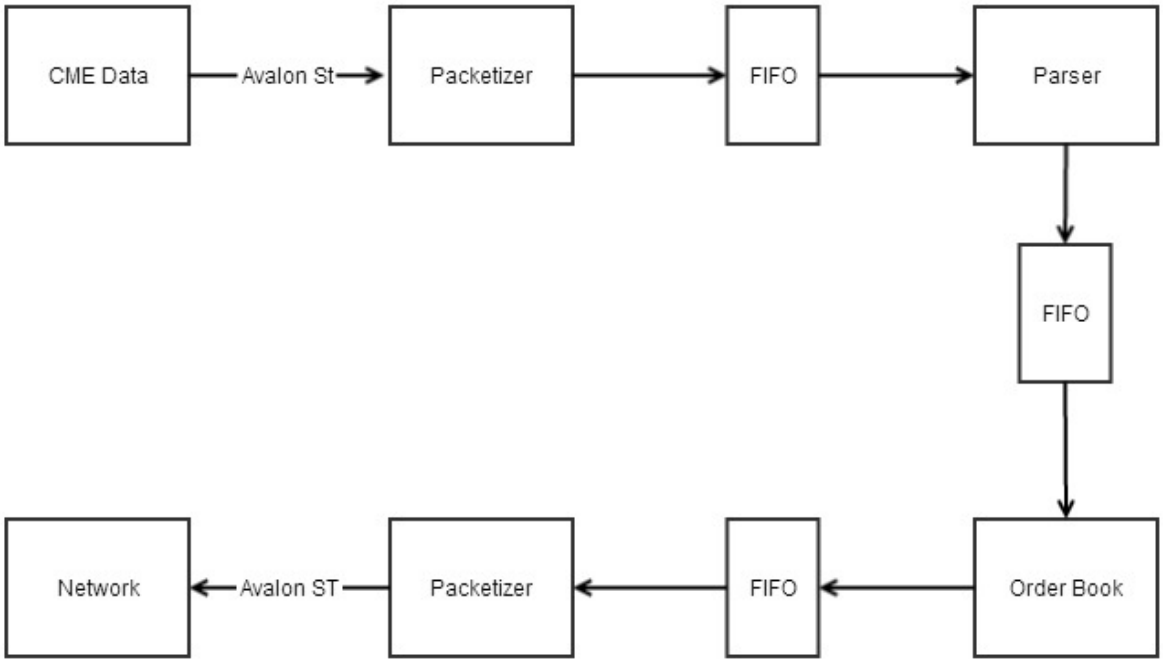
Before we received a sample of real MDP3 data, we wrote a python parser and book builder based on pseudo-MDP3 hex strings we had constructed based off the specifications of the CME website. Although this software protoype was not built to process real MDP3 data, it helped to model the hardware components of our parser and book builder. After we received the sample MDP3 data, a more accurate and robust python was produced by Professor David Lariviere. (Please see appendix for reference)

Algorithm

The core logic of our project relies on quick and efficient order book generation. However the prerequisites are correct data processing and validation by the packetizer and parser. CME data consists of a number of extra headers namely TCP/UDP and Security tags. The first part of our algorithm packetizes input from the CME Group and then stores blocks of data in our implementation of

a FIFO. Then our parser is able to read from the FIFO and feed it through another FIFO that the Order Book Module reads from. Ultimately the logic built into the Order Book allows for speedy addition, updation and deletion of order in the order book. This is reflected via changes in price, quantity, number and type of stocks. Currently we support 10 distinct price levels for N instantiated order book stocks. We plan to be able to scale this for N price levels as well as support the generation of implied order books by storing a series past months order books. The remainder of the report dives deep into the framework of our project.

Architecture

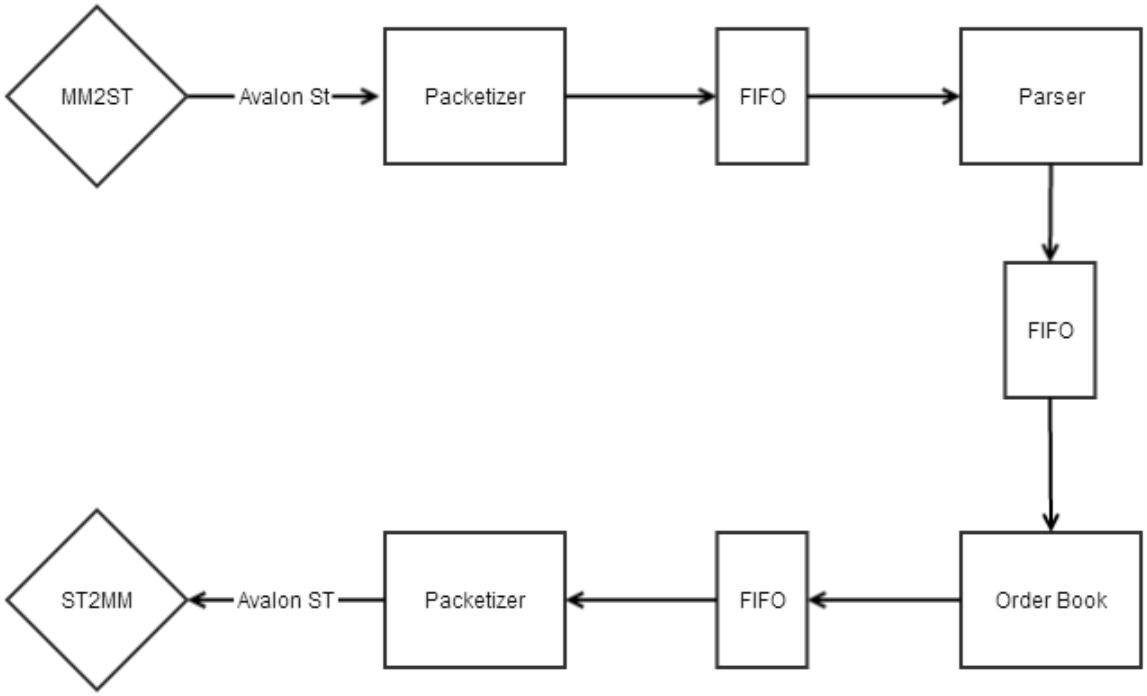


High Level Block Diagram

The high-level architecture of our project is shown in the diagram above. We begin with reading data broadcast by the CME group. It is assumed that the data is passed to us in 8-byte packets using an Avalon ST interface. Our Packetizer implements Avalon ST and strips away our network headers before putting it on a 64 bit wide FIFO queue. The Parser then picks data off the queue and extracts the relevant information which it then passes through another FIFO to the Order Book. When the Order Book receives the data, it first checks the SECURITY_ID field. If it the SECURITY_ID matches, then

the Order Book proceeds by either adding, deleting, or updating its contents according to the contents of the message.

In order to check whether our order book is functioning properly while testing, we also have the Order Book check the RptSeq number contained within the data. If that number matches a pre-specified hardcoded value, the Order Book then writes the entire contents of it's book, 16-bytes at a time to another FIFO. The ST_OUT module then takes this data off the FIFO and writes it to an ST2MM module which maps it to memory, allowing for software validation.



Test Environment using Avalon On-Chip Memory FIFOs (MM2ST, ST2MM)

Hardware Overview:

The MDP 3.0 Order Book Generator relies on a number of hardware components working in sync to be able to read data from the CME Group and create a up-to-date working order book. All our hardware logic is written in SystemVerilog and compiles on Quartus.

A detailed description of individual components is below:

FIFO

The Mega-Wizard generated FIFO queue implementations allows us to decouple all the components in our system. This way, a long operation on one component involving multiple clock cycles would not affect any other components.

We support the required enqueue and dequeue implementation of the FIFO queue on the rising edge of the clock cycle as shown below:

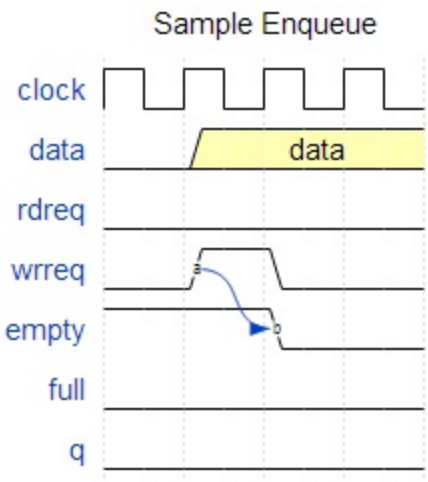


Figure 1

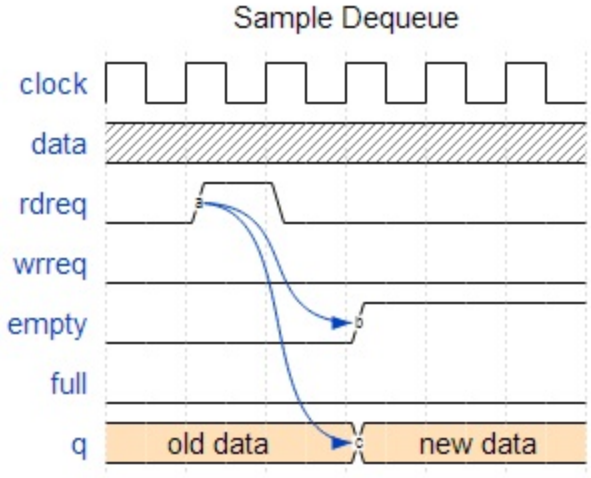


Figure 2

Also to be able to continuously read in data and update the parser simultaneously we implement the logic required for enqueuing and dequeuing at the same time. This happens at the rising clock edge and is shown below as well:

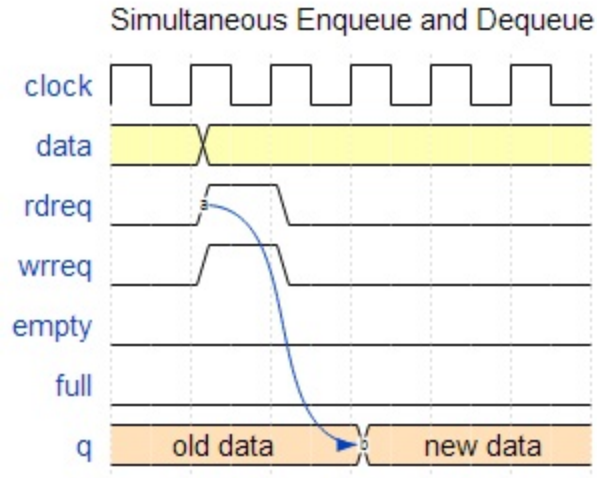


Figure 3

Parser

Our MDP Parser is able to read Message inputs from the FIFO queue. The FIFO queue stores 64bits x 256 Data values and on READ_SIG high it pulls each 64 bit chunk and parses it. The parsing algorithm is then able to decipher the Message into the following blocks which it then feeds into the Order Book Generator:

- MsgSeqNum
- SendingTime
- MsgSize
- BlockLength
- TemplateID
- SchemaID
- Version
- RawTransactTime
- MatchEventIndicator
- BlockLengthEntry
- NumMdEntries

In the intermediate steps the parser stores temporary offset values in buffers and then finally stores these values in its output logic. The follwing represents the input required for the order book algorithm:

- ACTION
- ENTRY_TYPE
- SECURITY_ID
- RptSeq
- PRICE
- QUANTITY
- NUM_ORDERS

The FSM for our parser is shown below:

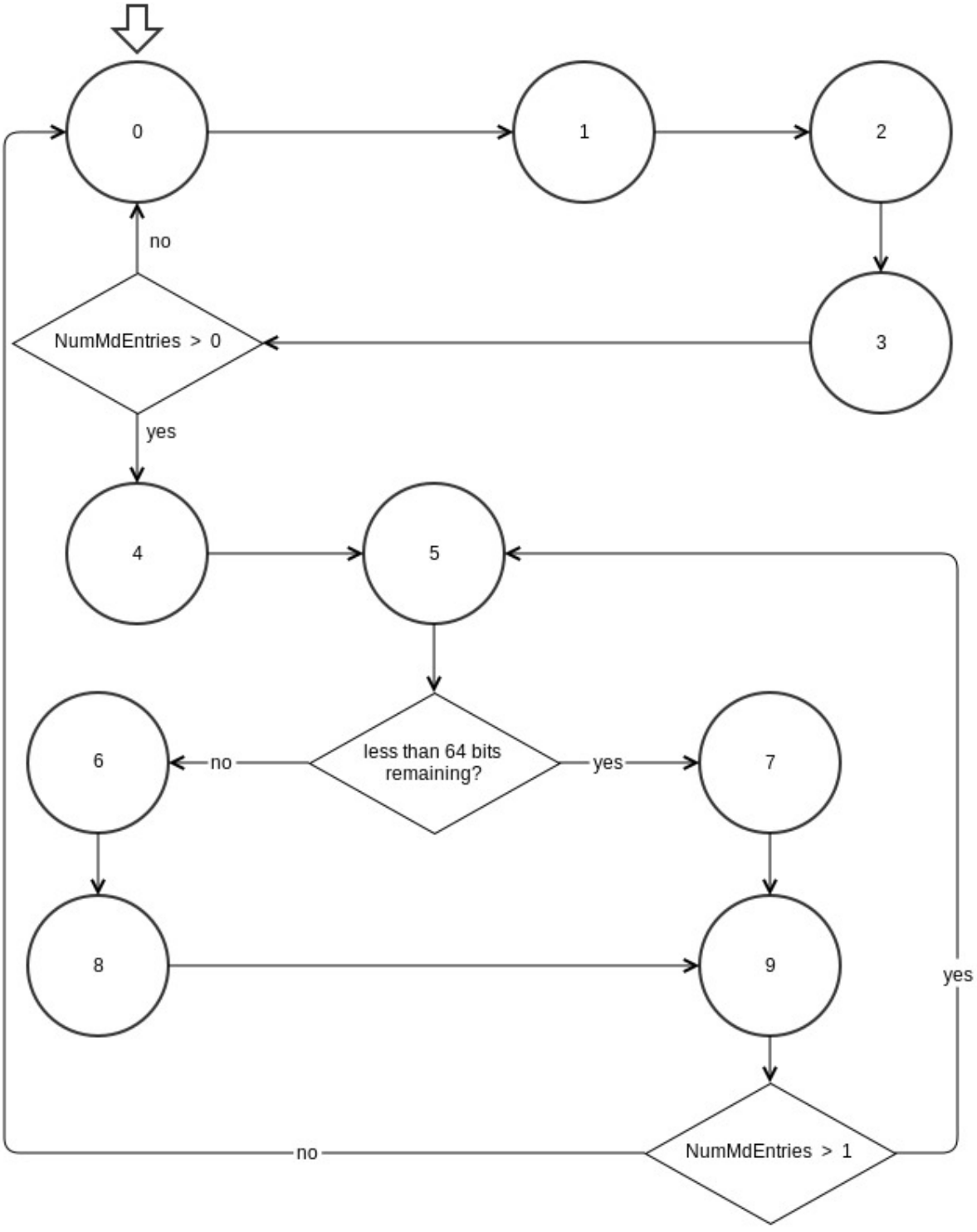

0
1
2
3
4
5
6
7
8
9
no
NumMdEntries > 0
yes
less than 64 bits remaining?
no
yes
NumMdEntries > 1
yes
no

Figure 4

Although message headers are always multiples of 64 bits, the actual message entries themselves are 214 bits long. However, data is always passed in 64-bit segments, which means that if there are multiple entries in a message, the start bit of every entry would exist at a different offset for each subsequent entry.

In order to calculate this offset, we use the following equation:

```
offset = (n*40)%64
```

where n is the nth entry in the message, starting with an index of 0. (In our implementation, we avoid using the multiplication operator and instead just increment offset by 40 each subsequent message).

```
offset = (offset+40)%64
```

We then use this offset value to read and write to a buffer where we store temporarily buffer the incoming messages.

Packetizer

Our Packetizer strips network headers off of incoming messages, and passes it into a FIFO connected the parser. The input ports of the Packetizer adhere to the Avalon ST and the output ports connect directly to a 64x256 bit FIFO.
The FSM for our packetizer is shown below:

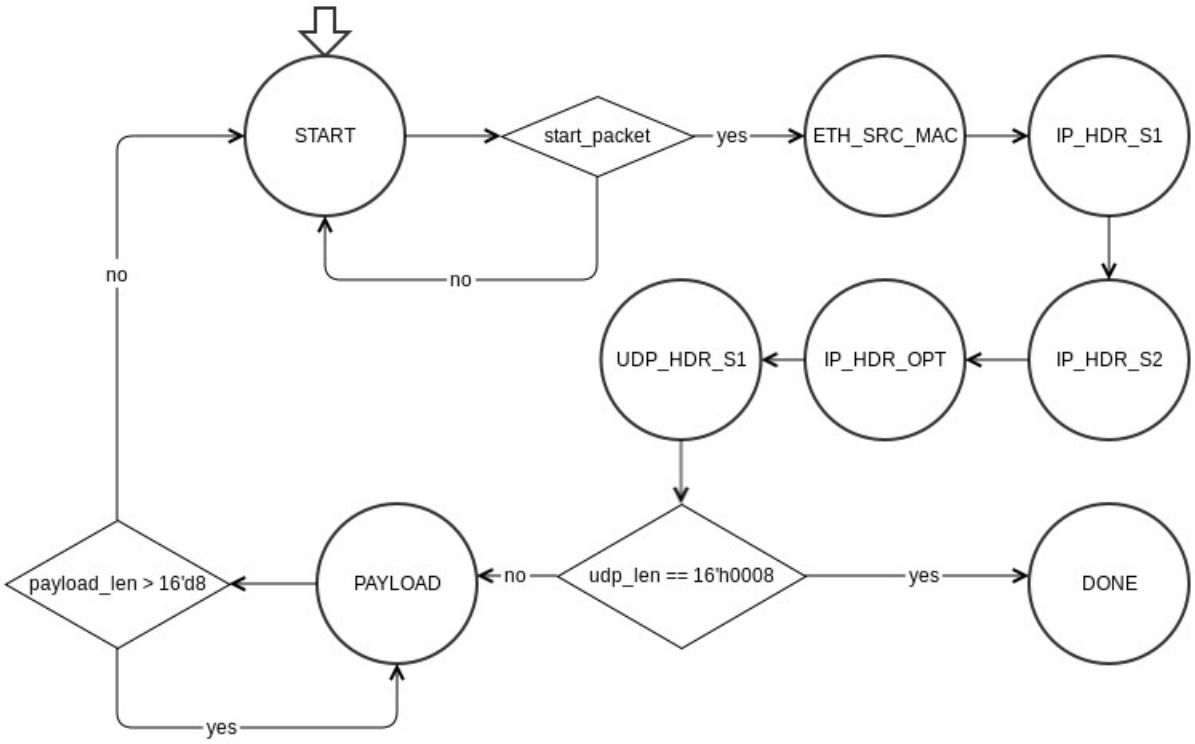


Figure 5

Order Book Generation

Our order book works via incremental market refresh updates. The order book acts as a functional module that reads in messages from the parser and the updates the inbuilt order books. Message inputs direct our algorithm to update a particular stock index with changes in pre-levels, quantity and number of orders. We are currently storing our order book in a series of registers, which we realize would never scale well. However, since we are only keeping track of a few securities for this prototype, the area blow up is not drastic.

The market refresh message sent to us from the CME group and decoded by the parser contains the following key data points.

- NUM_ORDERS
- QUANTITY
- PRICE
- ACTION, ENTRY_TYPE
- SECURITY_ID

Given the data we construct the following:

- 10 LEVELS OF ASK & BID PRICE LEVELS
- STORE DATA FOR IMPLIED ORDER BOOK CONSTRUCTION

Name	**Description**
NUM_ORDERS	number of repeating orders
QUANTITY	the order quantity
PRICE	the price level of the order
ACTION	the associated action 0 = New 1 = Change 2 = Delete
ENTRY_TYPE	the type of entry 0 = Bid 1 = Offer
SECURITY_ID	Unique Instrument ID

The modular and functional design allows us to instantiate a number of order books (so long as there is memory on the board) We hope that in the future we can expand the size of order books to size N to be more scalable. After processing input data and calculating refresh updates via the algorithm we store the newly created order book in memory to help us construct the implied order book.

Our Order_Book then writes to a 128 bit wide FIFO in order to transfer it's contents to ST_OUT

ST_OUT

ST_OUT reads the Order Book data off a FIFO and uses an Avalon ST interface to broadcast its data to the network in 8-byte packets.

Implied Order Book Generation

An implied order is an order created from individual outright orders available in the market place. Implied IN/OUT spreading occurs when the trading engine simultaneously works synthetic spread orders in spread markets and synthetic orders in the individual leg markets without the risk to the trader/broker of being double filled or filled on one leg and not on the other leg. By linking the spread and outright markets, implied spread trading substantially increases market liquidity.

Implied Orders can be created at multiple levels via first, second generation etc. Our future work on the project aims to be able to read the saved Order Books across different months to create Implied books. First Generation Implied requires us to read in 2 histories of order books and generate an implied version of IN's & OUT's. The complexity associated with Implied Books is that updating it requires us to update the past and present books as well.

Credits and Acknowledgements

We'd like to thank Professor Edwards, Professor Larivier, and Qiushi Ding for lending their skills, expertise, and time to this project.

Source Code:

packetizer.sv:

```
module packetizer(data_in,clk, reset_n, start_packet, end_packet, EN,
writeReq, reset, done, data_out, ready, empty);

      parameter START = 4'b0000;

      parameter DROP_PACKET = 4'b0001;

      parameter PREAMBLE = 4'b0010;

      parameter ETH_MAC = 4'b0011;

      parameter ETH_SRC_MAC = 4'b0100;

      parameter ETH_VLAN = 4'b0101;

      parameter IP_HDR_S1 = 4'b0110;

      parameter IP_HDR_S2 = 4'b0111;

      parameter IP_HDR_OPT = 4'b1000;

      parameter UDP_HDR_S1 = 4'b1001;

      parameter PAYLOAD = 4'b1010;

      parameter DONE = 4'b1011;



      parameter INPUT_WIDTH = 64;

      parameter BIT_WIDTH = 48;


      input wire [63:0] data_in;
      input wire clk;
      input wire reset_n;
      input wire start_packet;
      input  wire end_packet;
```

```
    input wire EN;
    input wire [2:0] empty;
    output wire ready; //packetizer read to take in data
    output reg writeReq; //not in use right now
    output reg reset;
    output logic done; //when done is high, start passing data_out,
write to FIFO
    output wire [63:0] data_out;


    //-------Ethernet Header Fields----
    reg [47:0] eth_dst;
    reg [47:0] eth_src;
    reg [15:0] eth_type;


    //-------IP Header Files----------
    reg [3:0] ip_ver;
    reg [3:0] ip_hlen;
    reg [7:0] ip_tos;
    reg [15:0] ip_len;
    reg [15:0] ip_id;
    reg [15:0] ip_flag_frag_off;
    reg [7:0] ip_ttl;
    reg [7:0] ip_protocol;
    reg [15:0] ip_checksum;
    reg [31:0] ip_src_addr;
    reg [31:0] ip_dst_addr;
    reg [31:0] ip_options;


    //-------UDP Header Files----------
    reg [15:0] udp_src_port_opt;
    reg [15:0] udp_dst_port;
    reg [15:0] udp_len;
    reg [15:0] udp_checksum_opt;


    //--------other buses and regs----
    reg [63:0] payload_data;
    reg [63:0] previous_data0;
    reg [63:0] previous_data1;
    reg [63:0] previous_data2;
```

```
reg [63:0] data_aligned;
reg [15:0] payload_len;
reg [3:0] offset = 4'b00_00;


reg [3:0] next_state;
wire [3:0] state;
reg flag;
reg done_final;
reg write_delay;
reg packet_ended;
reg [63:0] packet_count = 0;

initial begin
offset = 4'b00_00;

end

assign ready = 1'b1;


always_ff@(posedge clk)  begin
     if (offset == 0) begin
          data_aligned = data_in;
     end
     else if (offset == 2) begin
          data_aligned [63:48] = previous_data0 [15:0];
          data_aligned [47:0] = data_in[63:16];
     end
     else if (offset == 4) begin
          data_aligned [63:32] = previous_data0 [31:0];
          data_aligned [47:0] = data_in[63:32];
     end
     else if (offset == 6) begin
          data_aligned [63:16] = previous_data0 [47:0];
          data_aligned [15:0] = data_in[63:48];
     end
     else if (offset == 10) begin
          data_aligned [63:48] = previous_data0 [15:0];
          data_aligned [47:0] = data_in[63:16];
     end
     else if (offset == 14) begin
          data_aligned [63:16] = previous_data0 [47:0];
```

```
            data_aligned [15:0] = data_in[63:48];
      end
      else begin
            data_aligned = data_in;
      end
end


assign state = next_state;
assign data_out = payload_data;


always_ff @(posedge clk) begin
      if (!reset_n) begin
            payload_data <= 'b0;
            done <= 'b0;
      end else begin

      if (EN) begin
            previous_data0<=data_in;
            previous_data1<=previous_data0;
            previous_data2<=previous_data1;
      end



      case (state)
            START: begin
                  done <= 1'b0;
                  if (EN) begin
                        if (start_packet)begin
                              packet_count <= packet_count + 1;
                              packet_ended <= 1'b0;
                              next_state <= ETH_SRC_MAC;
                              offset <= 4'b0000;
                        end
                        else begin
                              next_state <= START;
                              done <=1'b0;
                        end
                  end
                  else begin
                        next_state <= START;
```

```
     end
end
ETH_MAC: begin //3

end
ETH_SRC_MAC: begin //4
     if (EN) begin
          done <= 0;
               next_state <= IP_HDR_S1;
     end
end
ETH_VLAN: begin //5
     if (EN) begin
          next_state <= IP_HDR_S1;
     end
end
IP_HDR_S1: begin           //6
     if (EN) begin
               next_state <= IP_HDR_S2;
     end
end
IP_HDR_S2: begin //7
     if (EN) begin
          next_state <= IP_HDR_OPT;
     end
end
IP_HDR_OPT: begin //8
     if (EN) begin
          udp_len <= data_in[15:0];
               next_state <= UDP_HDR_S1;
     end
end
UDP_HDR_S1: begin //9
     if (EN) begin
          if (udp_len == 16'h0008 )begin
               payload_len <= 16'h0000;
               next_state <= DONE;
          end
          else begin
               payload_len <= udp_len - 16'h0008;
               //payload_len <= 16'd40;
               next_state <= PAYLOAD;
               //offset <= 4'd6;
```

```
                    end
               end
          end
          PAYLOAD: begin //10
               if (end_packet && EN) begin
                    packet_ended <= 1'b1;
                    next_state <= START;
               end
               if ((!packet_ended && EN) || packet_ended &&!EN)
begin
                    if (payload_len > 16'd8) begin
                         payload_data <= data_aligned;
                         payload_len <= payload_len-
16'b0000_0000_0000_1000;
                         next_state <= PAYLOAD;
                         done <= 1'b1;
                    end
                    else begin
                         case (payload_len)
                              16'h0001: begin
                                   payload_data <=
{data_aligned [63:56], 56'h00000000000000};
                                   payload_len <=
payload_len-1;
                                   done <= 1'b1;
                              end
                              16'h0002: begin
                                   payload_data <=
{data_aligned [63:48], 48'h000000000000};
                                   payload_len <=
payload_len-2;
                                   done <= 1'b1;
                              end
                              16'h0003: begin
                                   payload_data <=
{data_aligned [63:40], 40'h0000000000};
                                   payload_len <=
payload_len-3;
                                   done <= 1'b1;
                              end
                              16'h0004: begin
                                   payload_data <=
{data_aligned [63:32], 32'h00000000};
```

```
                                        payload_len <=
payload_len-4;
                                        done <= 1'b1;
                                    end
                                    16'h0005: begin
                                        payload_data <=
{data_aligned [63:24], 24'h000000};
                                        payload_len <=
payload_len-5;
                                        done <= 1'b1;
                                    end
                                    16'h0006: begin
                                        payload_data <=
{data_aligned [63:16], 16'h0000};
                                        payload_len <=
payload_len-6;
                                        done <= 1'b1;
                                    end
                                    16'h0007: begin
                                        payload_data <=
{data_aligned [63:8], 8'h00};
                                        payload_len <=
payload_len-7;
                                        done <= 1'b1;
                                    end
                                    16'h0008: begin
                                        payload_data <=
data_aligned [63:0];
                                        payload_len <=
payload_len-8;
                                        done <= 1'b1;
                                    end
                                    default: payload_data <=
64'h00000000_00000000;
                                endcase
                                done_final <= 1'b1;
                                next_state <= START;
                                offset <= 4'b0000;
                            end
                        end else begin
                            done <= 1'b0;
                        end
                    end
```

```
                default: next_state <= START;
            endcase
        end
        end

always_ff@ (posedge clk) begin
        if (!reset_n) begin
            writeReq <= 1'b0;
            write_delay <= 1'b0;
        end else begin
            write_delay <= done;
            if (state == PAYLOAD) begin
                writeReq <= 1'b1;
            end else begin
                writeReq <= write_delay;
            end
        end
end


endmodule
```

MDP3_Parser.sv

```
module MDP3_Parser(
      input clk,
      input reset,
      input logic not_empty, //tells parser when to start reading
      input logic [63:0] MESSAGE, //assume each message is 8 bytes
      output logic[31:0] NUM_ORDERS,
      output logic[31:0] QUANTITY,
      output logic[63:0] PRICE,
      output logic[7:0] ACTION, ENTRY_TYPE,
      output logic [31:0] SECURITY_ID,
      output logic message_ready,//let next block know message is
ready
      output logic parser_ready,
      output logic enable_order_book //halts the reading of orderbook
if low
      );

      int bus_count = 0;
      logic data_valid;
      int entries_seen = 0;
      logic test = 1'b0;
      logic [63:0] PRICE_TEMP, SendingTime, SendingTimeTemp,
RawTransactTime, temp;
      logic [31:0] SECURITY_ID_TEMP, MsgSeqNum, RptSeq, EntrySize,
NumOrdersTemp;
      logic processing = 1'b0;
      logic [15:0] MsgSize, BlockLength, Version, BlockLengthEntry;
      logic [7:0] TemplateID, SchemaID, MatchEventIndicator,
NumMdEntries, priceLevel;
      logic [215:0] buffer;
      int offset = 0;
      //if reset, reset all data
      initial begin
            message_ready <= 1'b0;
            bus_count = 0;
            parser_ready <= 1'b1;
      end

      always_ff @(posedge clk) begin

            if(not_empty && parser_ready)begin
```

```
                data_valid <= 1'b1;
            end else begin
                data_valid <= 1'b0;
            end

            if(data_valid && parser_ready || data_valid && processing)
begin
                enable_order_book <= 1'b1;

                case(bus_count)
                    0: begin
                        bus_count += 1;
                        processing <= 1'b1;
                        parser_ready <= 1'b1; //right now parser
is ready for next input each clock cycle
                        message_ready <= 1'b0;
                        MsgSeqNum <=
changeEndian32(MESSAGE[63-:32]);
                        SendingTimeTemp[63-:32] <=
MESSAGE[31-:32];
                    end
                    1: begin
                        SendingTimeTemp[31-:32] <=
MESSAGE[63-:32];
                        MsgSize <=
changeEndian16(MESSAGE[31-:16]);
                        BlockLength <=
changeEndian16(MESSAGE[15-:16]);
                        bus_count += 1;
                    end
                    2: begin
                        SendingTime <=
changeEndian64(SendingTimeTemp);
                        TemplateID <= MESSAGE[63-:8];
                        SchemaID <= MESSAGE[55-:8];
                        Version <=
changeEndian16(MESSAGE[47-:16]);
                        RawTransactTime[63-:32] <=
MESSAGE[31-:32];
                        bus_count += 1;
                    end
                    3: begin
```

```
                        RawTransactTime[31-:32] <=
MESSAGE[63-:32];
                        MatchEventIndicator <= MESSAGE[31-:8];
                        BlockLengthEntry <=
changeEndian16(MESSAGE[23-:16]);
                        NumMdEntries <= MESSAGE[7-:8];
                        if(MESSAGE[7-:8] > 0)
                              bus_count += 1;
                        else begin
                              message_ready <= 1'b0;
                              processing <= 1'b0;
                              parser_ready <= 1'b1;
                              bus_count = 0;
                        end
                  end
                  4: begin // Start processing entriesa
                        RawTransactTime <=
changeEndian64(RawTransactTime);
                        buffer[(215-offset)-:64] <= MESSAGE;
                        bus_count = 5;
                  end
                  5: begin
                        message_ready <= 1'b0;
                        buffer[(151-offset)-:64] <= MESSAGE;
                        if(87 - offset < 64)
                              bus_count = 7;
                        else
                              bus_count = 6;
                  end
                  6: begin
                        buffer[(87-offset)-:64] <= MESSAGE;
                        bus_count = 8;
                  end
                  7: begin
                        case(offset)
                              24: begin
                                    buffer[63-:64] <= MESSAGE;
                              end
                              32: begin
                                    buffer[55-:56] <=
MESSAGE[63-:56];
                                    temp[63-:8] <= MESSAGE[7-:8];
                              end
```

```
                              40: begin
                                   buffer[47-:48] <=
MESSAGE[63-:48];
                                   temp[63-:16] <= MESSAGE[15-:16];
                              end
                              48: begin
                                   buffer[39-:40] <=
MESSAGE[63-:40];
                                   temp[63-:24] <= MESSAGE[23-:24];
                              end
                              56: begin
                                   buffer[31-:32] <=
MESSAGE[63-:32];
                                   temp[63-:32] <= MESSAGE[31-:32];
                              end
                         endcase
                         offset <= (offset+40)%64;
                         bus_count = 9;
                    end
                    8: begin
                         case(offset)
                              0: begin
                                   buffer[23-:24] <=
MESSAGE[63-:24];
                                   temp[63-:40] <= MESSAGE[39-:40];
                              end
                              8: begin
                                   buffer[15-:16] <=
MESSAGE[63-:16];
                                   temp[63-:48] <= MESSAGE[47-:48];
                              end
                              16: begin
                                   buffer[7-:8] <= MESSAGE[63-:8];
                                   temp[63-:56] <= MESSAGE[55-:56];
                              end
                         endcase
                         offset <= (offset+40)%64;
                         bus_count = 9;
                    end
                    9: begin
                         // OUTPUT ALL values
                         ACTION <= buffer[215 -: 8];
                         ENTRY_TYPE <= buffer[207 -: 8];
```

```
                        SECURITY_ID <=
changeEndian32(buffer[199-:32]);
                        RptSeq <= changeEndian32(buffer[167-:32]);
                        priceLevel <= buffer[135-:8];
                        PRICE <= changeEndian64(buffer[127 -:
64]);
                        QUANTITY <=
changeEndian32(buffer[63-:32]);
                        NUM_ORDERS <=
changeEndian32(buffer[31-:32]);
                        message_ready <= 1'b1;

                        if(NumMdEntries - 1> 0) begin
                              NumMdEntries <= NumMdEntries - 1;
                              entries_seen <= entries_seen + 1;
                              case(offset)
                                    8:
                                          buffer[215-:8] <=
temp[63-:8];
                                    16:
                                          buffer[215-:16] <=
temp[63-:16];
                                    24:
                                          buffer[215-:24] <=
temp[63-:24];
                                    32:
                                          buffer[215-:32] <=
temp[63-:32];
                                    40:
                                          buffer[215-:40] <=
temp[63-:40];
                                    48:
                                          buffer[215-:48] <=
temp[63-:48];
                                    56:
                                          buffer[215-:56] <=
temp[63-:56];
                              endcase
                              buffer[(215-offset)-:64] <= MESSAGE;
                              bus_count = 5;
                        end
                        else begin
```

```
                                    //Executed if valid is high right
after finished
                                    processing <= 1'b0;
                                    parser_ready <= 1'b1;
                                    offset <= 1'b0;
                                    bus_count = 0;
                                    NumMdEntries <= 1'b0;
                               end
                          end
                     endcase
               end else if(processing && (bus_count == 9)) begin
                          ACTION <= buffer[215 -: 8];
                          ENTRY_TYPE <= buffer[207 -: 8];
                          SECURITY_ID <= changeEndian32(buffer[199-:32]);
                          RptSeq <= changeEndian32(buffer[167-:32]);
                          priceLevel <= buffer[135-:8];
                          PRICE <= changeEndian64(buffer[127 -: 64]);
                          QUANTITY <= changeEndian32(buffer[63-:32]);
                          NUM_ORDERS <= changeEndian32(buffer[31-:32]);
                          message_ready <= 1'b1;

                          if(NumMdEntries - 1> 0) begin
                               NumMdEntries <= NumMdEntries - 1;
                               entries_seen <= entries_seen + 1;
                               case(offset)
                                    8:
                                         buffer[215-:8] <= temp[63-:8];
                                    16:
                                         buffer[215-:16] <= temp[63-:16];
                                    24:
                                         buffer[215-:24] <= temp[63-:24];
                                    32:
                                         buffer[215-:32] <= temp[63-:32];
                                    40:
                                         buffer[215-:40] <= temp[63-:40];
                                    48:
                                         buffer[215-:48] <= temp[63-:48];
                                    56:
                                         buffer[215-:56] <= temp[63-:56];
                               endcase
                               buffer[(215-offset)-:64] <= MESSAGE;
                               bus_count = 5;
                          end
```

```
                    else begin
                         //Executed if valid is high right after
finished
                         processing <= 1'b0;
                         parser_ready <= 1'b1;
                         offset <= 1'b0;
                         bus_count = 0;
                         NumMdEntries <= 1'b0;
                    end
          end else if(data_valid == 1'b0) begin //end if
               enable_order_book <= 1'b0;
          end //end else if
     end //end always_ff


     function [15 : 0] changeEndian16;
          input [15:0] value;
          changeEndian16 = {value[7 -: 8], value[15 -: 8]};
     endfunction

     function [23:0] changeEndian24;
          input [23:0] value;
          changeEndian24 = {value[7 -: 8], value[15 -: 8], value[23
-: 8]};
     endfunction

     function [31:0] changeEndian32;
          input [31:0] value;
          changeEndian32 = {value[7 -: 8], value[15 -: 8], value[23
-: 8], value[31 -: 8]};
     endfunction

     function [39:0] changeEndian40;
          input [39:0] value;
          changeEndian40 = {value[7 -: 8], value[15 -: 8], value[23
-: 8], value[31 -: 8], value[39 -: 8]};
     endfunction

     function [47:0] changeEndian48;
          input [47:0] value;
          changeEndian48 = {value[7 -: 8], value[15 -: 8], value[23
-: 8], value[31 -: 8], value[39 -: 8], value [47 -: 8]};
     endfunction
```

```
     function [55:0] changeEndian56;
          input [55:0] value;
          changeEndian56 = {value[7 -: 8], value[15 -: 8], value[23
-: 8], value[31 -: 8], value[39 -: 8], value [47 -: 8], value[55 -:
8]};
     endfunction

     function [63:0] changeEndian64;
          input [63:0] value;
          changeEndian64 = {value[7 -: 8], value[15 -: 8], value[23
-: 8], value[31 -: 8], value[39 -: 8], value [47 -: 8], value[55 -:
8], value[63 -: 8]};
     endfunction

endmodule
```

Order_Book.sv: *Note this is not the final version of Order Book, substantial changes have been made since this version.

```
module Order_Book(

     input clk,
     input logic message_ready, //when the parser has a message ready
for us
     input logic enable_order_book,
     input logic[31:0] NUM_ORDERS,
     input logic[31:0] QUANTITY,
     input logic[63:0] PRICE,
     input logic[7:0] ACTION, ENTRY_TYPE,
     input logic[31:0] SECURITY_ID,
     output logic[127:0] ASK0, ASK1, ASK2, ASK3, ASK4, ASK5, ASK6,
ASK7, ASK8, ASK9,
     output logic[127:0] BID0, BID1, BID2, BID3, BID4, BID5, BID6,
BID7, BID8, BID9,
     output logic orderbook_ready //let next block know message is
ready might need more?
     );
// Parameters
parameter DG_SECURITY_ID = 0;
parameter MAX_CONTRACTS = 10;
parameter PRICE_IDX = 63;
parameter QUANTITY_IDX = 95;
parameter NUM_ORDERS_IDX = 127;
     // 87-:16 -> Quantity; 71-:8 -> NUM_ORDERS; 63-:64 -> PRICE;
     logic [127:0] ask [MAX_CONTRACTS-1:0];
     logic [127:0] bid [MAX_CONTRACTS-1:0];
     assign ASK0 = ask[0];
     assign ASK1 = ask[1];
     assign ASK2 = ask[2];
     assign ASK3 = ask[3];
     assign ASK4 = ask[4];
     assign ASK5 = ask[5];
     assign ASK6 = ask[6];
     assign ASK7 = ask[7];
     assign ASK8 = ask[8];
     assign ASK9 = ask[9];
     assign BID0 = bid[0];
     assign BID1 = bid[1];
     assign BID2 = bid[2];
```

```
assign BID3 = bid[3];
assign BID4 = bid[4];
assign BID5 = bid[5];
assign BID6 = bid[6];
assign BID7 = bid[7];
assign BID8 = bid[8];
assign BID9 = bid[9];
logic reset = 1'b0; //if reset is 1 reset to initial state

/* Create a sample initial state of book with fake entries */
initial begin
      // set price
      bid[0][PRICE_IDX-:64] <= 64'd11;
      bid[1][PRICE_IDX-:64] <= 64'd10;
      bid[2][PRICE_IDX-:64] <= 64'd8;
      bid[3][PRICE_IDX-:64] <= 64'd7;
      bid[4][PRICE_IDX-:64] <= 64'd6;
      bid[5][PRICE_IDX-:64] <= 64'd5;
      bid[6][PRICE_IDX-:64] <= 64'd4;
      bid[7][PRICE_IDX-:64] <= 64'd3;
      bid[8][PRICE_IDX-:64] <= 64'd2;
      bid[9][PRICE_IDX-:64] <= 64'd1;
      // Set quantity and
      bid[0][QUANTITY_IDX-:32] <= 16'd8;
      bid[1][QUANTITY_IDX-:32] <= 16'd8;
      bid[2][QUANTITY_IDX-:32] <= 16'd8;
      bid[3][QUANTITY_IDX-:32] <= 16'd8;
      bid[4][QUANTITY_IDX-:32] <= 16'd8;
      bid[5][QUANTITY_IDX-:32] <= 16'd8;
      bid[6][QUANTITY_IDX-:32] <= 16'd8;
      bid[7][QUANTITY_IDX-:32] <= 16'd8;
      bid[8][QUANTITY_IDX-:32] <= 16'd8;
      bid[9][QUANTITY_IDX-:32] <= 16'd8;
      // Set num orders
      bid[0][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[1][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[2][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[3][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[4][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[5][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[6][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[7][NUM_ORDERS_IDX-:32] <= 8'd8;
      bid[8][NUM_ORDERS_IDX-:32] <= 8'd8;
```

```
      bid[9][NUM_ORDERS_IDX-:32] <= 8'd8;

      //ASK0 contains lowest ask.
      // set price
      ask[0][PRICE_IDX-:64] <= 64'd1;
      ask[1][PRICE_IDX-:64] <= 64'd2;
      ask[2][PRICE_IDX-:64] <= 64'd4;
      ask[3][PRICE_IDX-:64] <= 64'd5;
      ask[4][PRICE_IDX-:64] <= 64'd6;
      ask[5][PRICE_IDX-:64] <= 64'd7;
      ask[6][PRICE_IDX-:64] <= 64'd8;
      ask[7][PRICE_IDX-:64] <= 64'd9;
      ask[8][PRICE_IDX-:64] <= 64'd10;
      ask[9][PRICE_IDX-:64] <= 64'd11;
      //Set quantity and
      ask[0][QUANTITY_IDX-:32] <= 16'd8;
      ask[1][QUANTITY_IDX-:32] <= 16'd8;
      ask[2][QUANTITY_IDX-:32] <= 16'd8;
      ask[3][QUANTITY_IDX-:32] <= 16'd8;
      ask[4][QUANTITY_IDX-:32] <= 16'd8;
      ask[5][QUANTITY_IDX-:32] <= 16'd8;
      ask[6][QUANTITY_IDX-:32] <= 16'd8;
      ask[7][QUANTITY_IDX-:32] <= 16'd8;
      ask[8][QUANTITY_IDX-:32] <= 16'd8;
      ask[9][QUANTITY_IDX-:32] <= 16'd8;
      //Set num orders
      ask[0][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[1][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[2][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[3][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[4][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[5][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[6][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[7][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[8][NUM_ORDERS_IDX-:32] <= 8'd8;
      ask[9][NUM_ORDERS_IDX-:32] <= 8'd8;

end
always_ff @(posedge clk) begin
      if(reset) begin
            reset <= 'b0;

      end else begin
```

```
                if(message_ready && enable_order_book &&
DG_SECURITY_ID == SECURITY_ID) begin

                // New
                if(ACTION == 2'd0) begin
                     // ENTRY_TYPE == bid
                     if(ENTRY_TYPE == 2'd0) begin
                          for(int i = 0; i < MAX_CONTRACTS; i++)
begin
                               if(bid[i][PRICE_IDX-:64] == PRICE)
begin
                                    bid[i][QUANTITY_IDX-:32] <=
bid[i][QUANTITY_IDX-:32] + QUANTITY;
                                    bid[i][NUM_ORDERS_IDX-:32] <=
bid[i][NUM_ORDERS_IDX-:32] + NUM_ORDERS;
                                    break;
                               end else if(bid[i][PRICE_IDX -:64] <
PRICE) begin
                                    for(int j = MAX_CONTRACTS-1; j >
i; j--) begin
                                         bid[j] <= bid[j-1];
                                    end
                                    bid[i] <= {QUANTITY, NUM_ORDERS,
PRICE};
                                    break;
                               end
                          end
                     end else if(ENTRY_TYPE == 2'd1) begin
//ENTRY_TYPE == ask
                          for(int i = 0; i < MAX_CONTRACTS; i++)
begin
                               if(ask[i][PRICE_IDX-:64] == PRICE)
begin
                                    ask[i][QUANTITY_IDX-:32] <=
ask[i][QUANTITY_IDX-:32] + QUANTITY;
                                    ask[i][NUM_ORDERS_IDX-:32] <=
ask[i][NUM_ORDERS_IDX-:32] + NUM_ORDERS;
                                    break;
                               end
                               else if(ask[i][PRICE_IDX-:64] >
PRICE) begin
                                    for(int j = MAX_CONTRACTS-1; j >
i; j--) begin
```

```
                                        ask[j] <= ask[j-1];
                                    end
                                    ask[i] <= {QUANTITY, NUM_ORDERS,
PRICE};
                                    break;
                               end
                          end
                     end
                end else if (ACTION == 2'd1)begin //update existing
                     if(ENTRY_TYPE == 2'd0)begin
                          for(int i = 0; i < MAX_CONTRACTS; i++)
begin
                               if(bid[i][PRICE_IDX-:64] == PRICE)
begin
                                    bid[i][QUANTITY_IDX-:32] <=
bid[i][QUANTITY_IDX-:32] + QUANTITY;
                                    bid[i][NUM_ORDERS_IDX-:32] <=
bid[i][NUM_ORDERS_IDX-:32] + NUM_ORDERS;
                                    break;
                               end
                          end
                     end else if(ENTRY_TYPE == 2'd1)begin
                          for(int i = 0; i < MAX_CONTRACTS; i++)
begin
                               if(ask[i][PRICE_IDX-:64] == PRICE)
begin
                                    ask[i][QUANTITY_IDX-:32] <=
ask[i][QUANTITY_IDX-:32] + QUANTITY;
                                    ask[i][NUM_ORDERS_IDX-:32] <=
ask[i][NUM_ORDERS_IDX-:32] + NUM_ORDERS;
                                    break;
                               end
                          end
                     end
                end else if( ACTION == 2'd2) begin //delete from book
action
                     if(ENTRY_TYPE == 2'd0) begin
                          for(int i=0; i<MAX_CONTRACTS; i++) begin
                               if(bid[i][PRICE_IDX -:64] == PRICE)
begin
                                    bid[i][QUANTITY_IDX-:32] <=
bid[i][QUANTITY_IDX-:32] - QUANTITY;
```

```
                                        bid[i][NUM_ORDERS_IDX-:32] <=
bid[i][NUM_ORDERS_IDX-:32] - NUM_ORDERS;

                                        //if no more orders left, delete
entry from book
                                        //we check if it's equal to
QUANTITY so that we can do this operation in 1 clock cycle
                                        if(bid[i][QUANTITY_IDX-:32] ==
QUANTITY && bid[i][NUM_ORDERS_IDX-:32] == NUM_ORDERS)begin
                                             for(int j = i; j <
MAX_CONTRACTS-1; j++) begin
                                                  bid[j] <= bid[j+1];
                                             end //end for
                                        end
                                        //NEED TO FIGURE WHAT TO DO WITH
LAST ORDER IN BID BOOK
                                        break;
                                   end
                              end//end for-loop
                         end else if(ENTRY_TYPE == 2'd1) begin
                              for(int i=0; i<MAX_CONTRACTS; i++) begin
                                   if(ask[i][PRICE_IDX -:64] == PRICE)
begin
                                        ask[i][QUANTITY_IDX-:32] <=
ask[i][QUANTITY_IDX-:32] - QUANTITY;
                                        ask[i][NUM_ORDERS_IDX-:32] <=
ask[i][NUM_ORDERS_IDX-:32] - NUM_ORDERS;

                                        //if no more orders left, delete
entry from book
                                        //we check if it's equal to
QUANTITY so that we can do this operation in 1 clock cycle
                                        if(ask[i][QUANTITY_IDX-:32] ==
QUANTITY && ask[i][NUM_ORDERS_IDX-:32] == NUM_ORDERS)begin
                                             for(int j = i; j <
MAX_CONTRACTS-1; j++) begin
                                                  ask[j] <= ask[j+1];
                                             end //end for
                                        end
                                        //NEED TO FIGURE WHAT TO DO WITH
LAST ORDER IN BID BOOK
                                        break;
                                   end
```

```
                        end//end for-loop
                    end//end else if entRY_TYPE
                end
                end //end if(message_ready)
            end //end if-else reset
        end //end always_ff
endmodule
```

MDP3_STREAMER_TOP.sv

```
module MDP3_STREAMER_TOP (
          input wire start_packet,
          input wire end_packet,
          input wire valid,
          input wire [63:0] data_in,
          input wire [2:0]empty,
          input wire clk,
          input wire reset_n,
          output wire ready,
          output wire [63:0] data_out,
          output reg writeReq, //not in use right now
          output reg reset,
          output wire [63:0] q,
          //output logic done, //when done is high, start passing
data_out, write to FIFO

          //Parser Output
          output logic[31:0] NUM_ORDERS,
          output logic[31:0] QUANTITY,
          output logic[63:0] PRICE,
          output logic[7:0] ACTION, ENTRY_TYPE,
          output logic [31:0] SECURITY_ID,
          output logic message_ready,//let next block know message
is ready
          //output logic parser_ready,
          output logic enable_order_book, //halts the reading of
orderbook if low

          //FIFO output
          output logic [63:0] message_packetizer_to_fifo,
          output logic fifo_wrreq, //write request from packetizer
to fifo
          output logic fifo_empty, //high if fifo is empty
          output logic read_fifo
     );

logic[127:0] ASK0, ASK1, ASK2, ASK3, ASK4, ASK5, ASK6, ASK7, ASK8,
ASK9;
logic[127:0] BID0, BID1, BID2, BID3, BID4, BID5, BID6, BID7, BID8,
BID9;
```

```
logic orderbook_ready; //let next block know message is ready might
need more?

logic[127:0] message_parser_to_ob_fifo_in;
logic wrreq_ob_fifo_in;
logic rdreq_ob_fifo_in;
logic empty_ob_fifo_in;

packetizer mdp3_packetizer(
                .EN(valid),
                .data_out(message_packetizer_to_fifo),
                .done(fifo_wrreq),
                .*
                );


scfifo64x256 fifo(
                .data(message_packetizer_to_fifo),
                .wrreq(fifo_wrreq),
                .clock(clk),
                .q(q),
                .rdreq(read_fifo),
                .empty(fifo_empty),
                .full(),
                .usedw(),
                .*
                );

MDP3_Parser parser (.MESSAGE(q), .not_empty(!fifo_empty),
.reset(reset_n), .parser_ready(read_fifo), .*);


Order_Book #(123,10) book(.*);

endmodule
```

streamer_top.v:

```
//
=============================================================================
=======
// Copyright (c) 2013 by Terasic Technologies Inc.
//
=============================================================================
=======
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
Development
//   Kits made by Terasic.  Other use of this code, including the
selling
//   ,duplication, or modification of any portion is strictly
prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
reference
//   which illustrates how these types of functions can be
implemented.
//   It is the user's responsibility to verify their design for
//   consistency and functionality through the use of formal
//   verification methods.  Terasic provides no warranty regarding
the use
//   or functionality of this code.
//
//
=============================================================================
=======
//
//  Terasic Technologies Inc
//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City,
30070. Taiwan
//
//
//                     web: http://www.terasic.com/
```

```
//                     email: support@terasic.com
//
//
=============================================================================
=======
//Date:  Mon Jul  1 14:21:10 2013
//
=============================================================================
=======


//`define ENABLE_DDR3
//`define ENABLE_HPS
//`define ENABLE_HSMC

module streamer_top(

      ///////// AUD /////////
      input              AUD_ADCDAT,
      inout              AUD_ADCLRCK,
      inout              AUD_BCLK,
      output             AUD_DACDAT,
      inout              AUD_DACLRCK,
      output             AUD_I2C_SCLK,
      inout              AUD_I2C_SDAT,
      output             AUD_MUTE,
      output             AUD_XCK,

`ifdef ENABLE_DDR3
      ///////// DDR3 /////////
      output      [14:0] DDR3_A,
      output      [2:0]  DDR3_BA,
      output             DDR3_CAS_n,
      output             DDR3_CKE,
      output             DDR3_CK_n,
      output             DDR3_CK_p,
      output             DDR3_CS_n,
      output      [3:0]  DDR3_DM,
      inout       [31:0] DDR3_DQ,
      inout       [3:0]  DDR3_DQS_n,
      inout       [3:0]  DDR3_DQS_p,
      output             DDR3_ODT,
      output             DDR3_RAS_n,
```

```
      output             DDR3_RESET_n,
      input              DDR3_RZQ,
      output             DDR3_WE_n,
`endif /*ENABLE_DDR3*/

      ///////// FAN /////////
      output             FAN_CTRL,

`ifdef ENABLE_HPS
      ///////// HPS /////////
      input              HPS_CONV_USB_n,
      output      [14:0] HPS_DDR3_A,
      output      [2:0]  HPS_DDR3_BA,
      output             HPS_DDR3_CAS_n,
      output             HPS_DDR3_CKE,
      output             HPS_DDR3_CK_n,
      output             HPS_DDR3_CK_p,
      output             HPS_DDR3_CS_n,
      output      [3:0]  HPS_DDR3_DM,
      inout       [31:0] HPS_DDR3_DQ,
      inout       [3:0]  HPS_DDR3_DQS_n,
      inout       [3:0]  HPS_DDR3_DQS_p,
      output             HPS_DDR3_ODT,
      output             HPS_DDR3_RAS_n,
      output             HPS_DDR3_RESET_n,
      input              HPS_DDR3_RZQ,
      output             HPS_DDR3_WE_n,
      output             HPS_ENET_GTX_CLK,
      inout              HPS_ENET_INT_n,
      output             HPS_ENET_MDC,
      inout              HPS_ENET_MDIO,
      input              HPS_ENET_RX_CLK,
      input       [3:0]  HPS_ENET_RX_DATA,
      input              HPS_ENET_RX_DV,
      output      [3:0]  HPS_ENET_TX_DATA,
      output             HPS_ENET_TX_EN,
      inout       [3:0]  HPS_FLASH_DATA,
      output             HPS_FLASH_DCLK,
      output             HPS_FLASH_NCSO,
      inout              HPS_GSENSOR_INT,
      inout              HPS_I2C_CLK,
      inout              HPS_I2C_SDA,
      inout       [3:0]  HPS_KEY,
```

```
      inout              HPS_LCM_BK,
      output             HPS_LCM_D_C,
      output             HPS_LCM_RST_N,
      input              HPS_LCM_SPIM_CLK,
      output             HPS_LCM_SPIM_MOSI,
      output             HPS_LCM_SPIM_SS,
      output      [3:0]  HPS_LED,
      inout              HPS_LTC_GPIO,
      output             HPS_SD_CLK,
      inout              HPS_SD_CMD,
      inout       [3:0]  HPS_SD_DATA,
      output             HPS_SPIM_CLK,
      input              HPS_SPIM_MISO,
      output             HPS_SPIM_MOSI,
      output             HPS_SPIM_SS,
      input       [3:0]  HPS_SW,
      input              HPS_UART_RX,
      output             HPS_UART_TX,
      input              HPS_USB_CLKOUT,
      inout       [7:0]  HPS_USB_DATA,
      input              HPS_USB_DIR,
      input              HPS_USB_NXT,
      output             HPS_USB_STP,
`endif /*ENABLE_HPS*/

`ifdef ENABLE_HSMC
      ///////// HSMC /////////
      input       [2:1]  HSMC_CLKIN_n,
      input       [2:1]  HSMC_CLKIN_p,
      output      [2:1]  HSMC_CLKOUT_n,
      output      [2:1]  HSMC_CLKOUT_p,
      output             HSMC_CLK_IN0,
      output             HSMC_CLK_OUT0,
      inout       [3:0]  HSMC_D,
      input       [7:0]  HSMC_GXB_RX_p,
      output      [7:0]  HSMC_GXB_TX_p,
      input              HSMC_REF_CLK_p,
      inout       [16:0] HSMC_RX_n,
      inout       [16:0] HSMC_RX_p,
      output             HSMC_SCL,
      inout              HSMC_SDA,
      inout       [16:0] HSMC_TX_n,
      inout       [16:0] HSMC_TX_p,
```

```
`endif /*ENABLE_HSMC*/

      ///////// IRDA /////////
      input              IRDA_RXD,

      ///////// KEY /////////
      input       [3:0]  KEY,

      ///////// LED /////////
      output      [3:0]  LED,

      ///////// OSC /////////
      input              OSC_50_B3B,
      input              OSC_50_B4A,
      input              OSC_50_B5B,
      input              OSC_50_B8A,

      ///////// PCIE /////////
      input              PCIE_PERST_n,
      output             PCIE_WAKE_n,

      ///////// RESET /////////
      input              RESET_n,

      ///////// SI5338 /////////
      inout              SI5338_SCL,
      inout              SI5338_SDA,

      ///////// SW /////////
      input       [3:0]  SW,

      ///////// TEMP /////////
      output             TEMP_CS_n,
      output             TEMP_DIN,
      input              TEMP_DOUT,
      output             TEMP_SCLK,

      ///////// USB /////////
      input              USB_B2_CLK,
      inout       [7:0]  USB_B2_DATA,
      output             USB_EMPTY,
      output             USB_FULL,
      input              USB_OE_n,
```

```
      input              USB_RD_n,
      input              USB_RESET_n,
      inout              USB_SCL,
      inout              USB_SDA,
      input              USB_WR_n,

      ///////// VGA /////////
      output      [7:0]  VGA_B,
      output             VGA_BLANK_n,
      output             VGA_CLK,
      output      [7:0]  VGA_G,
      output             VGA_HS,
      output      [7:0]  VGA_R,
      output             VGA_SYNC_n,
      output             VGA_VS



);


//=======================================================
//  REG/WIRE declarations
//=======================================================
wire [63:0]                          lb_data          ;
wire                                 lb_valid         ;
wire                                 lb_ready         ;
wire                                 lb_startofpacket ;
wire                                 lb_endofpacket   ;
wire[2:0]                            lb_empty         ;
//=======================================================
//  Structural coding
//=======================================================



    streamer_qsys u0 (
        .clk_50_clk      (OSC_50_B4A),      //    clk_50.clk
        .reset_in0_reset (~RESET_n),  // reset_in0.reset
        .stream_src_data           (lb_data),           //
stream_src.data
        .stream_src_valid          (lb_valid),          //
.valid
```

```
        .stream_src_ready           (lb_ready),           //
.ready
        //.stream_src_ready           (1),           //
.ready

            .stream_src_startofpacket  (lb_startofpacket),  //
.startofpacket
        .stream_src_endofpacket    (lb_endofpacket),    //
.endofpacket
        .stream_src_empty           (lb_empty),           //
.empty

            .stream_sink_data           (),           //
stream_sink.data
        .stream_sink_valid          (),          //             .valid
        .stream_sink_ready          (),          //             .ready
        .stream_sink_startofpacket (), //             .startofpacket
        .stream_sink_endofpacket    (),   //             .endofpacket
        .stream_sink_empty          ()           //             .empty

    );


     MDP3_STREAMER_TOP mdp3_packetizer_top(
          .start_packet(lb_startofpacket),
          .end_packet(lb_endofpacket),
          .valid(lb_valid),
          .data_in(lb_data),
          .ready(lb_ready),
          .empty(lb_empty),
          .clk(OSC_50_B4A),
          .reset_n(RESET_n),
          .data_out()
          );




endmodule
```

scfifo64x256.v:

```
// megafunction wizard: %FIFO%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: scfifo

// ============================================================
// File Name: scfifo64x256.v
// Megafunction Name(s):
//             scfifo
//
// Simulation Library Files(s):
//             altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 13.1.1 Build 166 11/26/2013 SJ Full Version
// ************************************************************


//Copyright (C) 1991-2013 Altera Corporation
//Your use of Altera Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Altera Program License
//Subscription Agreement, Altera MegaCore Function License
//Agreement, or other applicable license agreement, including,
//without limitation, that your use is for the sole purpose of
//programming logic devices manufactured by Altera and sold by
//Altera or its authorized distributors.  Please refer to the
//applicable agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module scfifo64x256 (
     clock,
```

```
data,
rdreq,
wrreq,
empty,
full,
q,
usedw);

input  clock;
input[63:0]  data;
input  rdreq;
input  wrreq;
output      empty;
output      full;
output    [63:0]  q;
output    [7:0]  usedw;

wire [7:0] sub_wire0;
wire  sub_wire1;
wire  sub_wire2;
wire [63:0] sub_wire3;
wire [7:0] usedw = sub_wire0[7:0];
wire  empty = sub_wire1;
wire  full = sub_wire2;
wire [63:0] q = sub_wire3[63:0];

scfifo    scfifo_component (
               .clock (clock),
               .data (data),
               .rdreq (rdreq),
               .wrreq (wrreq),
               .usedw (sub_wire0),
               .empty (sub_wire1),
               .full (sub_wire2),
               .q (sub_wire3),
               .aclr (),
               .almost_empty (),
               .almost_full (),
               .sclr ());
defparam
     scfifo_component.add_ram_output_register = "OFF",
     scfifo_component.intended_device_family = "Cyclone V",
     scfifo_component.lpm_numwords = 256,
```

```
            scfifo_component.lpm_showahead = "OFF",
            scfifo_component.lpm_type = "scfifo",
            scfifo_component.lpm_width = 64,
            scfifo_component.lpm_widthu = 8,
            scfifo_component.overflow_checking = "ON",
            scfifo_component.underflow_checking = "ON",
            scfifo_component.use_eab = "ON";


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: AlmostEmpty NUMERIC "0"
// Retrieval info: PRIVATE: AlmostEmptyThr NUMERIC "-1"
// Retrieval info: PRIVATE: AlmostFull NUMERIC "0"
// Retrieval info: PRIVATE: AlmostFullThr NUMERIC "-1"
// Retrieval info: PRIVATE: CLOCKS_ARE_SYNCHRONIZED NUMERIC "0"
// Retrieval info: PRIVATE: Clock NUMERIC "0"
// Retrieval info: PRIVATE: Depth NUMERIC "256"
// Retrieval info: PRIVATE: Empty NUMERIC "1"
// Retrieval info: PRIVATE: Full NUMERIC "1"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: LE_BasedFIFO NUMERIC "0"
// Retrieval info: PRIVATE: LegacyRREQ NUMERIC "1"
// Retrieval info: PRIVATE: MAX_DEPTH_BY_9 NUMERIC "0"
// Retrieval info: PRIVATE: OVERFLOW_CHECKING NUMERIC "0"
// Retrieval info: PRIVATE: Optimize NUMERIC "0"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: UNDERFLOW_CHECKING NUMERIC "0"
// Retrieval info: PRIVATE: UsedW NUMERIC "1"
// Retrieval info: PRIVATE: Width NUMERIC "64"
// Retrieval info: PRIVATE: dc_aclr NUMERIC "0"
// Retrieval info: PRIVATE: diff_widths NUMERIC "0"
// Retrieval info: PRIVATE: msb_usedw NUMERIC "0"
// Retrieval info: PRIVATE: output_width NUMERIC "64"
// Retrieval info: PRIVATE: rsEmpty NUMERIC "1"
// Retrieval info: PRIVATE: rsFull NUMERIC "0"
// Retrieval info: PRIVATE: rsUsedW NUMERIC "0"
// Retrieval info: PRIVATE: sc_aclr NUMERIC "0"
// Retrieval info: PRIVATE: sc_sclr NUMERIC "0"
```

```
// Retrieval info: PRIVATE: wsEmpty NUMERIC "0"
// Retrieval info: PRIVATE: wsFull NUMERIC "1"
// Retrieval info: PRIVATE: wsUsedW NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf
altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADD_RAM_OUTPUT_REGISTER STRING "OFF"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone
V"
// Retrieval info: CONSTANT: LPM_NUMWORDS NUMERIC "256"
// Retrieval info: CONSTANT: LPM_SHOWAHEAD STRING "OFF"
// Retrieval info: CONSTANT: LPM_TYPE STRING "scfifo"
// Retrieval info: CONSTANT: LPM_WIDTH NUMERIC "64"
// Retrieval info: CONSTANT: LPM_WIDTHU NUMERIC "8"
// Retrieval info: CONSTANT: OVERFLOW_CHECKING STRING "ON"
// Retrieval info: CONSTANT: UNDERFLOW_CHECKING STRING "ON"
// Retrieval info: CONSTANT: USE_EAB STRING "ON"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT NODEFVAL "clock"
// Retrieval info: USED_PORT: data 0 0 64 0 INPUT NODEFVAL
"data[63..0]"
// Retrieval info: USED_PORT: empty 0 0 0 0 OUTPUT NODEFVAL "empty"
// Retrieval info: USED_PORT: full 0 0 0 0 OUTPUT NODEFVAL "full"
// Retrieval info: USED_PORT: q 0 0 64 0 OUTPUT NODEFVAL "q[63..0]"
// Retrieval info: USED_PORT: rdreq 0 0 0 0 INPUT NODEFVAL "rdreq"
// Retrieval info: USED_PORT: usedw 0 0 8 0 OUTPUT NODEFVAL
"usedw[7..0]"
// Retrieval info: USED_PORT: wrreq 0 0 0 0 INPUT NODEFVAL "wrreq"
// Retrieval info: CONNECT: @clock 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data 0 0 64 0 data 0 0 64 0
// Retrieval info: CONNECT: @rdreq 0 0 0 0 rdreq 0 0 0 0
// Retrieval info: CONNECT: @wrreq 0 0 0 0 wrreq 0 0 0 0
// Retrieval info: CONNECT: empty 0 0 0 0 @empty 0 0 0 0
// Retrieval info: CONNECT: full 0 0 0 0 @full 0 0 0 0
// Retrieval info: CONNECT: q 0 0 64 0 @q 0 0 64 0
// Retrieval info: CONNECT: usedw 0 0 8 0 @usedw 0 0 8 0
// Retrieval info: GEN_FILE: TYPE_NORMAL scfifo64x256.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL scfifo64x256.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL scfifo64x256.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL scfifo64x256.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL scfifo64x256_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL scfifo64x256_bb.v FALSE
// Retrieval info: LIB_FILE: altera_mf
```

stream.tcl: (Modified from Qiushi Ding's original tcl script)

```
variable infile 0
variable s_path ""
set s_path [lindex [get_service_paths master] 0]
open_service master $s_path
set infile [open tclFormatNoNetwork.txt "r"]
set outfile [open packet_data_out.txt "w"]

set stream_base 0x0
set ready_base 0x8
set mmst_base 0x10

# Set register "ready_reg" in Avalon ST simulator to 0, buffer data
in the fifo "sc_fifo_in"
master_write_8 $s_path $ready_base 0x0

# Send packet data in packet_data.txt
set line 64
set wait 1000
set precursor "0x"

set packet_num 0



while { [eof $infile] != 1 } {
set packet_byte [gets $infile]


if {$packet_byte eq "sop"} {
# Send special characters: 7a00_0000_0000_0000 indicate
"startofpacket"
master_write_32 $s_path $stream_base 0x44332211
puts "sop"
set j 0
set inframe 1
set payload_line "7a7a7a55"
while {[expr $j < 1000] && $inframe} {
     #flag to figure out whether or not to pad with 0s
     set filler_flag 0
     incr j
     for {set i 0} {$i < 4} {incr i} {
```

```
            set packet_byte [gets $infile]

            if {$packet_byte eq "7a"} {
                  set packet_byte "5a7d"
                  set payload_line $packet_byte$payload_line
                  incr i
            } elseif {$packet_byte eq "7b" || $packet_byte eq "7B"} {
                  set packet_byte "5b7d"
                  set payload_line $packet_byte$payload_line
                  incr i
            } elseif {$packet_byte eq "7c"} {
                  set packet_byte "5c7d"
                  set payload_line $packet_byte$payload_line
                  incr i
            } elseif {$packet_byte eq "7d"} {
                  set packet_byte "5d7d"
                  set payload_line $packet_byte$payload_line
                  incr i
            } elseif {$packet_byte == "eop"} {
                  set packet_byte "7b"
                  set inframe 0
                  set payload_line "[string range $payload_line 0
1]$packet_byte[string range $payload_line 2 end]"
                  puts "eop $i"
                  break
            } else {
                set payload_line $packet_byte$payload_line
            }

      }
      puts "payload_line: $payload_line"
      set data_to_stream $precursor[string range $payload_line end-7
end]
      set payload_line [string range $payload_line 0 end-8]
      puts "data_to_stream: $data_to_stream"
      master_write_32 $s_path $stream_base $data_to_stream
}
      set data_to_stream $precursor[string range "000000$payload_line"
end-7 end]
      set payload_line [string range $payload_line 0 end-8]
      puts "data_to_stream: $data_to_stream"
      master_write_32 $s_path $stream_base $data_to_stream
}
```

```
      puts "packet_num: $packet_num"
      incr packet_num
}

master_write_8 $s_path $ready_base 0x1
for {set i 0} {$i < [expr $wait]} {incr i} {
   #puts "[master_read_32 $s_path $mmst_base 1]"
}
master_write_8 $s_path $ready_base 0x0
```

mdp3_parser.py:

```
Author: David Alan Lariviere

#!/usr/bin/env python
import sys
import struct
unpack = struct.unpack
structerror = struct.error
import datetime

#For references:
#ftp://ftp.cmegroup.com/SBEFix/NRCert/Templates/templates_FixBinary.x
ml
#http://www.cmegroup.com/confluence/display/EPICSANDBOX/MDP3

#Note inside base_path are N files numbered 0.dat --> 2000.dat
base_path = "/mnt/vmshared/data/mdp3_data"

class MDP3Parser:
#     def __init__(self):

    #Given a byte describing the match event, create a string
describing its contents: see reference @
http://www.cmegroup.com/confluence/display/EPICSANDBOX/MDP3+-+Market+
Data+Incremental+Refresh under tag 5799
    def match_event_string(self, match_event_byte):
        desc = ""
#         Bit 0: (least significant bit) Last Trade Summary message
for a given event
#         Bit 1: Last electronic volume message for a given event
#         Bit 2: Last real quote message for a given event
#         Bit 3: Last statistic message for a given event
#         Bit 4: Last implied quote message for a given event
#         Bit 5: Message resent during recovery
#         Bit 6: Reserved for future use
#         Bit 7: (most significant bit) Last message for a given
event
        if match_event_byte & 1:
            desc = desc + " last trade for event |"
        if match_event_byte & 2:
            desc = desc + " last volume | "
        if match_event_byte & 4:
            desc = desc + " last quote | "
```

```
        if match_event_byte & 8:
            desc = desc + " last quote | "
        if match_event_byte & 16:
            desc = desc + " last implied quote | "
        if match_event_byte & 32:
            desc = desc + " resent during recovery |"
        if match_event_byte & 64:
            desc = desc + " reserved for future use | "
        if match_event_byte & 128:
            desc = desc + " last msg of an event | "

        return desc


    def parse_file(self, filename):
        self.seenbytes = 0
        self.prevbytes = 0
        f = file(filename)

        num_bytes_to_read = 16
        nbytes = f.read()
        print "Will parse", cur_msg_filename, "\t contains ",
len(nbytes), " bytes"
        #print "\tRaw Frame: ", ':'.join(x.encode('hex') for x in
nbytes)
#             print ' '.join('{:02x}'.format(x) for x in nbytes)


        #TODO: replace so we can handle parsing multiple messages



        #All MDP messages start with "binary packet header" consists
of:
        #   MsgSeqNum: uint32
        #   SendingTime: uint64
        msg_seq_num, raw_epoch_sending_time_ns = unpack('<IQ',
nbytes[:12])
        sending_datetime =
datetime.datetime.fromtimestamp(raw_epoch_sending_time_ns /
1000000000)
```

```
        print "\tBinary Packet Header: Seq Num / Epoch NS Timestamp:
", msg_seq_num, "\t", raw_epoch_sending_time_ns, "\t",
sending_datetime, ".", (raw_epoch_sending_time_ns % 1000000000)

        nbytes = nbytes[12:] # drop the header bytes
        #Message header:
        #   MsgSize: uint16
        #   BlockLength: uint16
        #   TemplateID: uint16
        #   SchemaID: uint16 #note typo on website? last two listed
as uint16s but each only one byte long
        #   Version: uint16
        msg_size, block_length, template_id, schema_id, version =
unpack('<HHHBB', nbytes[:8])
        print "\tMsg Size: ", msg_size
        print "\tBlock Length: ", block_length
        print "\tTemplate ID: ", template_id
        print "\tSchema ID: ", schema_id
        print "\tVersion: ", version

        nbytes = nbytes[8:] #drop the message header bytes

        #FIX Message Header:
        #   MsgType: "int" is this 4 bytes or not?
        #fix_msg_type = unpack('<i', nbytes[:4])
        #print "\tRemaining bytes: ", nbytes
        #print "\tFix Msg Type: ", fix_msg_type

        #right now, only implementing parsing for template 20
        if template_id != 20:
            return

        #Note:
#         <ns2:message name="MDIncrementalRefreshBook" id="20"
description="MDIncrementalRefreshBook" blockLength="9"
semanticType="X">
#         <field name="TransactTime" id="60" type="uInt64"
description="Start of event processing time in number of nanoseconds
since Unix epoch" offset="0" semanticType="UTCTimestamp"/>
#         <field name="MatchEventIndicator" id="5799"
type="MatchEventIndicator" description="Bitmap field of eight Boolean
type indicators reflecting the end of updates for a given Globex
event" offset="8" semanticType="MultipleCharValue"/>
```

```
#         <group name="NoMDEntries" id="268" description="Number of
entries in Market Data message" blockLength="27"
dimensionType="groupSize">
#         <field name="MDUpdateAction" id="279" type="MDUpdateAction"
description=" Market Data update action" offset="0"
semanticType="int"/>
#         <field name="MDEntryType" id="269" type="MDEntryTypeBook"
description="Market Data entry type " offset="1"
semanticType="char"/>
#         <field name="SecurityID" id="48" type="Int32"
description="Security ID" offset="2" semanticType="int"/>
#         <field name="RptSeq" id="83" type="Int32"
description="Market Data entry sequence number per instrument update"
offset="6" semanticType="int"/>
#         <field name="MDPriceLevel" id="1023" type="uInt8"
description="Aggregate book level" offset="10" semanticType="int"/>
#         <field name="MDEntryPx" id="270" type="PRICENULL"
description="Market Data entry price" offset="11"
semanticType="Price"/>
#         <field name="MDEntrySize" id="271" type="Int32NULL"
description="Market Data entry quantity" offset="19"
semanticType="Qty"/>
#         <field name="NumberOfOrders" id="346" type="Int32NULL"
description="In Book entry - aggregate number of orders at given
price level" offset="23" semanticType="int"/>
#         </group>
#         </ns2:message>
#         print "".join("{:02x}".format(ord(c)) for c in nbytes)
        #print "\t", ':'.join(x.encode('hex') for x in nbytes)
        #note "NoMDEntries" is a composite type defined as uint16
blockLength + uin8 numInGroup
        raw_transact_time_ns, match_event_indicator, block_length,
num_md_entries = unpack("<QBHB", nbytes[:12])
        print "\tRaw Transact Time NS: " , raw_transact_time_ns
        print "\tMatch Event Indicator: ", match_event_indicator,
"\t", self.match_event_string(match_event_indicator)
        print "\tNum MD Entries: ", num_md_entries
        num_bytes_left = (len(nbytes) - 12)
        if num_md_entries > 0:
            print "\tNOTE: have ", num_bytes_left, " bytes of
repeating groups: averaging ", (num_bytes_left / num_md_entries), "
bytes per entry"
        nbytes = nbytes[12:]
```

```
        if num_md_entries > 0 and len(nbytes) < 27:
            print "Error: invalid sized byte array of only ",
len(nbytes), "; skipping!"
            return


        cur_entry = 0

        while len(nbytes) >= 27:
            #Begin repeating group of entries
            update_action, entry_type, security_id, rpt_seq,
price_level, entry_price_mantissa, entry_size, num_orders =
unpack("<BcIiBqii", nbytes[:27])
            entry_price_exponent = -7 # hard coded constant
            entry_price = entry_price_mantissa * pow(10,
entry_price_exponent)
            #note for implied liquidity, there is no concept of # of
orders, so field can be null, represented as all 1's except highest
bit (2147483647 in base 10)
            if num_orders == 2147483647:
                num_orders = "NULL"
            #print "\tRaw Bytes for Entry", ':'.join(x.encode('hex')
for x in nbytes)
            print "\t", cur_entry, " entry message: "
            print "\t\tUpdate Action: ", update_action
            print "\t\tEntry Type: ", entry_type
            print "\t\tSecurity ID: ", security_id
            print "\t\tRpt Seq: ", rpt_seq
            print "\t\tPrice Level: ", price_level
            print "\t\tEntry Price Mantissa: " , entry_price_mantissa
            print "\t\tEntry Price Exponent: ", entry_price_exponent
            print "\t\tEntry Price: ", entry_price
            print "\t\tEntry Size: " , entry_size
            print "\t\tNum Orders: ", num_orders
            cur_entry = cur_entry + 1
            nbytes = nbytes[27:]
#         if not nbytes:
#             yield (None, 'EOF at %d' % self.seenbytes)
#             break
#             self.seenbytes += num_bytes_to_read
#             print ''.join('{:02x}'.format(x) for x in nbytes)

            #length = unpack('>H', nbytes)[0]
```

```
parser = MDP3Parser()

for i in range(2000):
    cur_msg_filename = base_path + "/%d.dat" % (i)
    parser.parse_file(cur_msg_filename)
```