Hardware Accelerated Decoding of FIX/FAST and Book Building of Market Data

Final Report - CSEE 4840 Spring 2013

Danqing Hua

Junkang Ren

Chang Liu

Raghavan Santhanam

1 Introduction

1.1 FIX/FAST: A Brief Overview

The Financial Information eXchange (FIX) Protocol is a series of messaging specifications for the electronic communication of trade-related messages. Because of the huge increase in volume of data transferred in today's markets, the FIX Adapted for STreaming (FAST) Protocol has been developed as part of the FIX Market Data Optimization Working Group. FIX messages, like any self-describing message syntax, have a relatively high overhead of message descriptor. The FAST Protocol is a way of eliminating this overhead by exchanging the message description separately from the message. For example, in traditional FIX messages each field takes the form "Tag=Value<SOH>", FAST eliminates redundancy with a template that describes the message structure. This technique is known as implicit tagging as the FIX tags become implicit in the data.

1.2 Book Builder

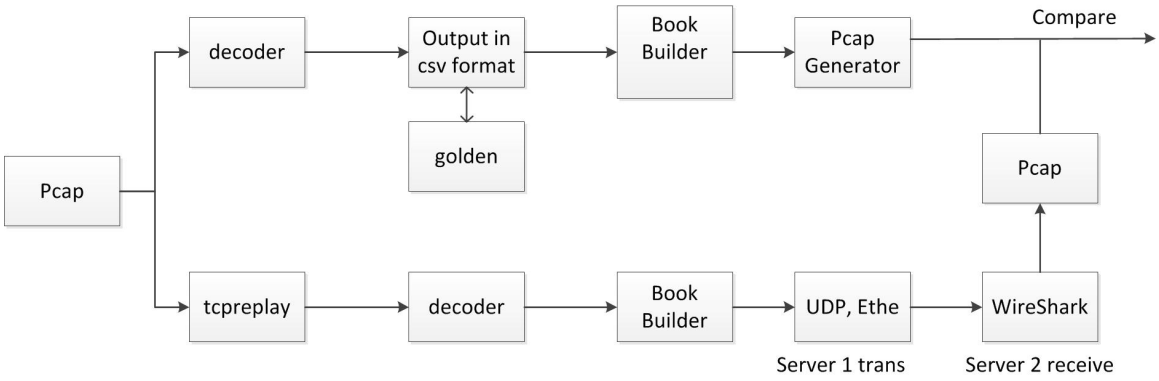
Book is records of bid and ask information in trading activity. Here are some important variables related to book builder:
MDEntryType: Decides whether we are working on book with bidding information or on book with asking information. (0 for bid, 1 for ask)
MDUpdateAction: 3 actions in total. "0" means add a new level (item) in the book; "1" means modify a certain level in the book; "2" means delete an existing level in the book.
MDPriceLevel: Decides which item of the book we are working on.
MDEntryPx: Price of a stock
MDEntrySize: The amount of a certain stock

1.3 Contribution of this project

In this project, we are using Solarflare AoE board to accelerate decoding of FIX/FAST, a protocol used for efficiently compressing market data, and build a book to record such data. Thanks to this high-speed board, we can implement such functionality on hardware that is much faster than software. By decoding the raw data sending from source, we use the predefined

templates with default values and delta-type operations that rely on previous values from other entries to get a much more compressed format. After that, we would build a book to store such information and update the book by three types of command such as update, insert, and delete. The book reports summarized order quantities and order counts at a given price level. The depth represents the number of price levels that are supported in the feed. Finally, the book is transmitted as a new UDP packet to the PC in which the AoE is installed.

2 Design Flow Overview



The design flow of the entire project is shown in figure above. We are provided with a pcap file that packs Ethernet packets including raw market data. This file is used to generate input for both Software Verification and Hardware Implementation.

The path on the top shows the flow of Software Verification. The decoder is implemented in C programming language. We take advantage of libpcap library to parse the pcap file and pear off Ethernet header, IP header and UDP header of each packet so as to derive the payload, which is the raw data encoded by Fix-Fast protocol. Then the trading information is decoded in this block and printed out to a CSV file as output. Then we can compare with the golden output the check if our algorithm is correct. Once we find the correct algorithm to decode it, we can implement the same algorithm in software. Then the book builder, which is implemented in Python, will take the CSV file as input, generate bid book and ask book which records the decoded information and finally output snapshot of book in pcap format.

The path on the bottom shows the flow of our Hardware Implementation. For this part we use “tcpreplay” on the server to read the input pcap file and sending out Ethernet packets packed inside that file. Such packets are sent to the AOE boards through an Internet cable where our hardware design is loaded. We separate our hardware design into two modules as decoder and book builder that are similar as what we have done in software. So during simulation, we can compare the intermediate output with the CSV golden output as shown by the double-sided arrow in the figure above. Our hardware modules at last will pack snapshot of books into Ethernet packets and send it back to the server.

Finally, on the sever we will run Wireshark to capture packets passing the Ethernet interface we are using and packets the received packets into another pcap file. We can now compare the content of pcap files generated by software and hardware to see if they are the same.

3 Hardware Design

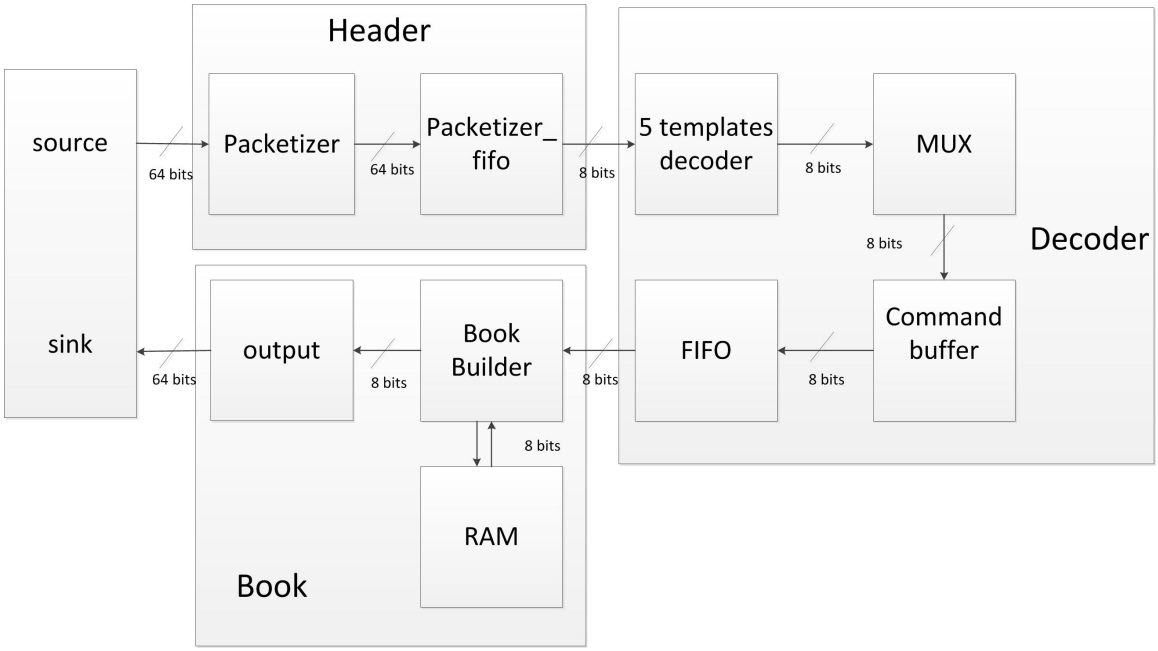


Figure 1 High Level Block Diagram

The whole hardware part is made of source/sink and three big modules (Header, Decoder and Book). Figure 1 shows the high-level block diagram of our project. Streaming source sends packets to the board. A packet consists of

IP header, UDP header and UDP payload. The Packetizer receives the raw data and rips always the IP header and UDP header, only passes along the payload to next module. Since the receiving data is 64 bits, however, our decoder process 8 bits per clock cycle. Here we have a packetizer_fifo to convert data transferring from 64 bits per clock cycle to 8 bits per cycle. And also it can store the data which hasn't been processed in case of big amount of input data. Then 8-bits flit enters decoder modules. There are 5 templates decoders corresponding to 5 potential templates. The MUX would choose the data from these 5 decoders by detecting the template number in the packet. When the seven fields which we are interested in have all been arrived, the command buffer would combine them to a piece of command. Here we add a fifo to prevent the case that book builder processes slower than the speed of generation of commands. Then the book builder updates the book according to each command. Finally, sink receives packet produced by our hardware modules, the packet consists header and the snapshot of book. Hardware modules of our project would be introduced in next sections.

3.1 Packetizer

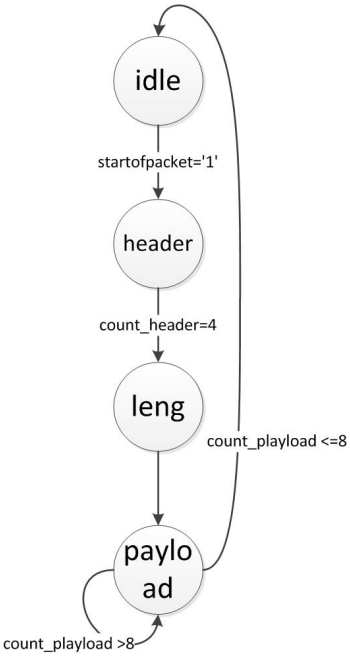


Figure 2 FSM of Packetizer

The aim of Packetizer is to rips always the IP header and UDP header which contains 42 bytes, only passes along the UDP payload. It also calculates the checksum and tosses the flit away in case the checksum is bad.

3.2 Packetizer_fifo

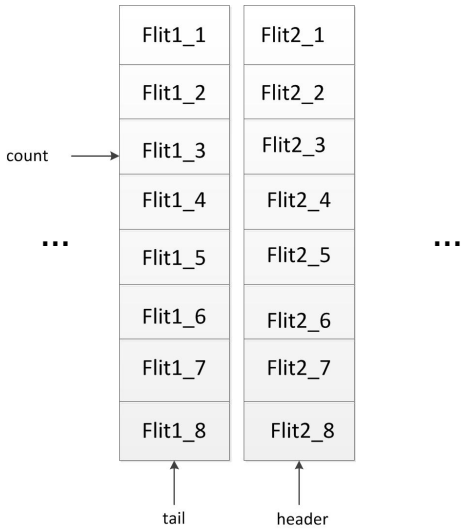


Figure 3 FIFO Diagram

To convert data to a 8-bits flit, we build a fifo and use flag tail to choose the 64-bits data which would be processed and then use flag count to choose the 8-bits flit which would be outputted. Also the valid_out should be modified. Before the fifo there is a one bit valid signal reflecting the validity of the 64-bits data. Now for each 8-bits flit, we need a new valid signal.

3.3 5_template_decoder

Here we have five templates decoders for template 117, 122, 125, 129, 131. Take decoder of template 117 for example, other four decoders are similar.

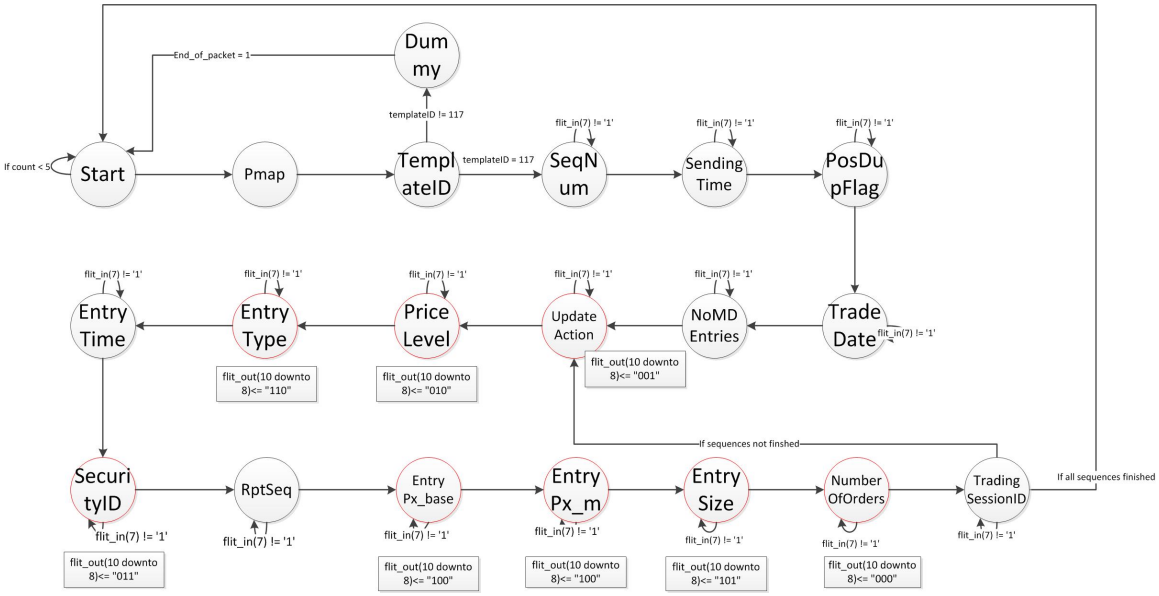


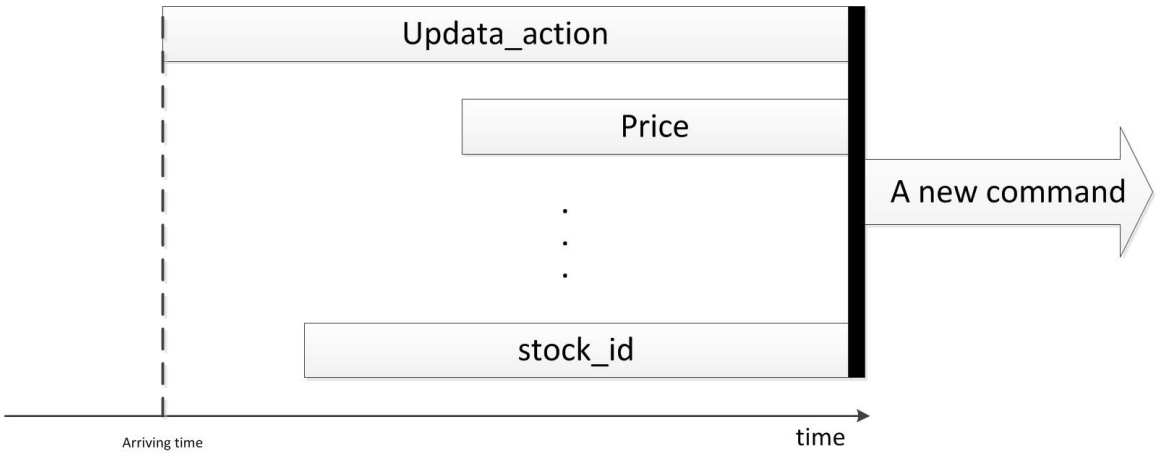
Figure 4 FSM of decoder of template 117

In Start state, it counts to five to toss away first five bytes which are sequence number and channel number. These five bytes occur in every template thus we can just ignore them. In Pmap state, we check whether it contains template and whether the optional field exists, actually there is another way to check the optional field's appearance which would be discussed later. Then in TemplateID state, the FSM compare the receiving template ID with the reserved template ID. If they are the same, it means that such packet should be processed by current decoder and goes to next states, otherwise it goes to Dummy state and waits for ending of this packet and then back to Start state. After the template ID is matched, the FSM will enter a sequence for N times, where N is indicated by NoMDEntries. As Figure 4 shows, the red states' input data is what we are really interested in, i.e. MDUpdateAction, MDEntryType, MDPriceLevel, SecurityID, MDEntrySize, MDEntryPx and NumberOfOrders. In particular, field MDEntryPx is separated as two fields MDEntryPx_base and MDEntryPx_m due to the presentation of this field. When FSM goes into these specified states, the flit_out(10 downto 8) would be assigned 3-bits flag to indicates which field it is. And such flag can be detected by the next module Command Buffer. When in TradingSessionID state which is the last state, it would go to Start state or the beginning of sequence depending on the count flag which indicates how many times of sequence it has completed. The flit_out(7 downto 0) is always the same as the input, we use the flit_out(11) (valid bit) and the flit_out(10 downto 8) (flag) to tell next module that

such output is useful.

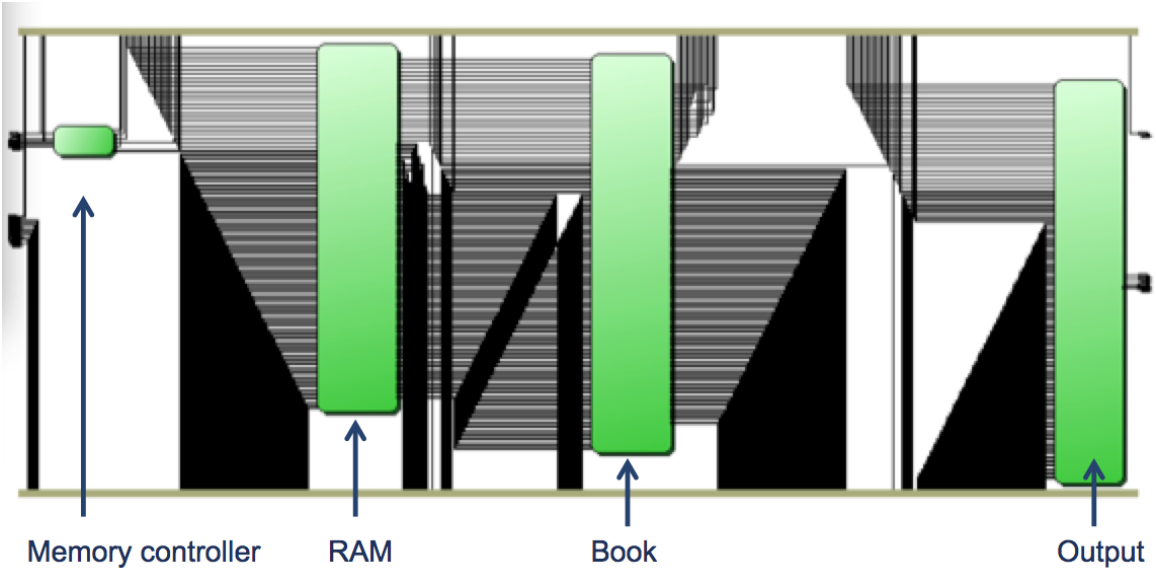
Other templates are more complicated than template 117 since there are optional fields in sequence, however, the process of decoding is quite similar.

3.4 Command Buffer



The command buffer combines specified information to a piece of command. The command buffer cannot output the command until it receives all the specified fields in a sequence. However, the orders of these fields are different in different templates, we should come up with a uniform method to output the command. To do this, we use seven registers in this module to restore the seven specified fields' data. Each time it detects the special 3-bits flag signals, the corresponding data would be stored in the register. When seven registers' highest bits are all high which means all seven useful fields arrive, it's the time to combine these data and output it.

3.5 Book Builder



The book building unit consists of three modules, the memory controller, the book builder and the RAM. The purpose of book builder is to carry out the decoded command from the FSM and build the real data for each "symbol"(which represents a stock).

Inside the RAM, each book for a particular symbol is stored at certain locations. For every individual command, the memory controller looks at the security ID and decode the ID into address. Then the memory controller sends out a "fetch" signal in the next cycle for the Book builder to load new data from RAM. In the following cycle, the Book builder operates modify, insert or delete command on either bid/ask side of the book based on the decoded command from FSM. When the operations finishes, the memory controller sends out a "write back" signal to tell the RAM to accept updated book content to its certain address locations, and to tell the book to reset itself. Meanwhile, the memory controller goes back to initial state and gets ready to execute next command.

It takes 3 cycle to finish executing a decoded command. Since the FSM needs more than 20 cycle to decode a command, this fetch/execute/writeback method is fast enough.

3.6 Output Module

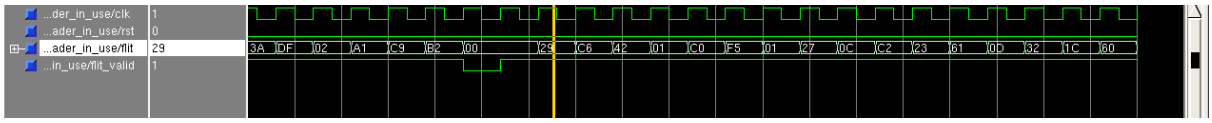
The output unit overlooks the execution of the command. It looks at the content of the book. When a certain level in a certain book has been changed, it records the book symbol(the security ID) and the level number that has been changed. When a packet is finished, the output unit moves all the level changing information to a output buffer and moves to detect further modification. Once the buffer receives an complete set of level change information, it begins to send out packets to the output port, that is the Avalon Bus.

The data transmission follows startofpacket, 8byte data/ cycle, endofpacket protocol. Since our output is fixed length, we gradually send out IP header, UDP header, and our real payload one cycle after one cycle. The data is sent to avalon bus and transmitted to Ethernet module.

3.7 Verification of Functionality

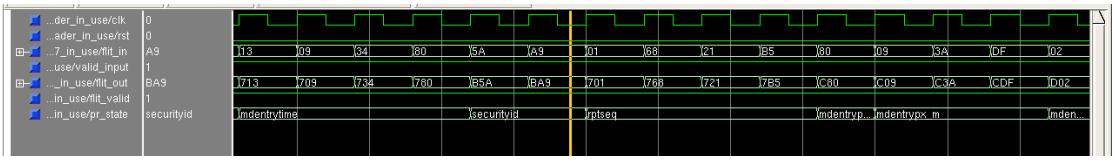
The functionality of all modules is verified by using testbench which is made of ten thousands commands. We use ModelSim to simulate our hardware design. By using “diff”

Header Module (Packetizer + Packetizer_fifo)



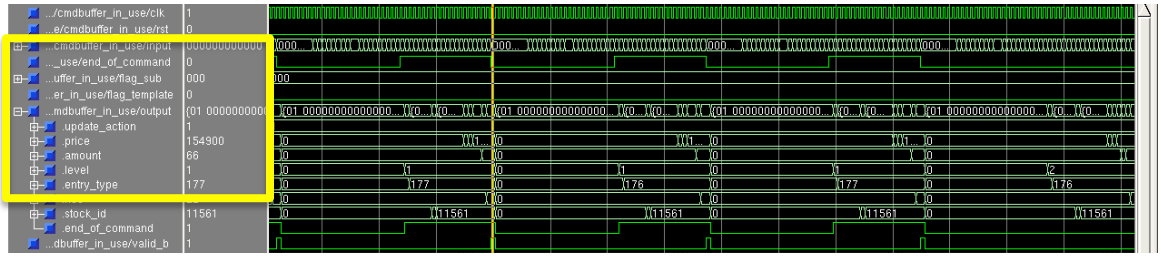
As we can see that the output “flit” is the UDP payload which tosses away the header and begins with “0029”.

Template Decoder Module:



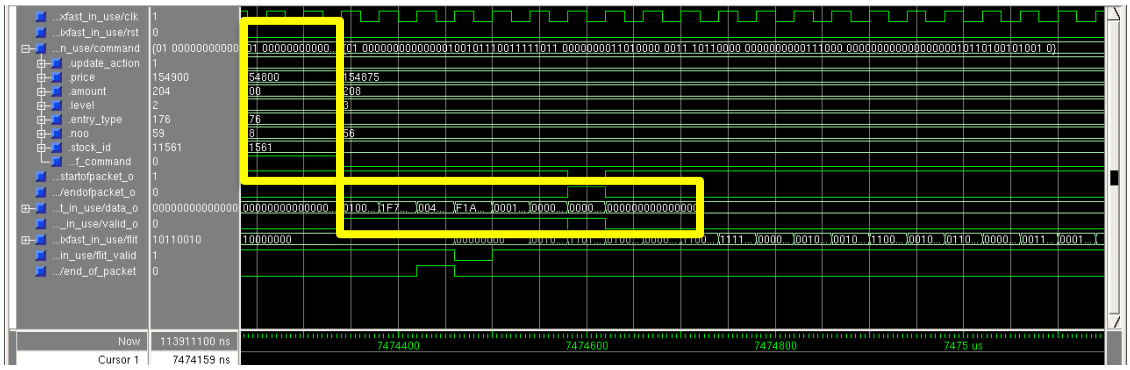
The flit_out has four additional bits in the head of the original flit_in, which indicates that whether the flit is useful and which useful flit it is. In the above figure, when the additional 4 bits is “7” in hexademinal, it means that flit is not useful; other values indicate different useful field. For example, “B” means SecurityID , “C” means MDEntryPx.

Command Buffer Module:



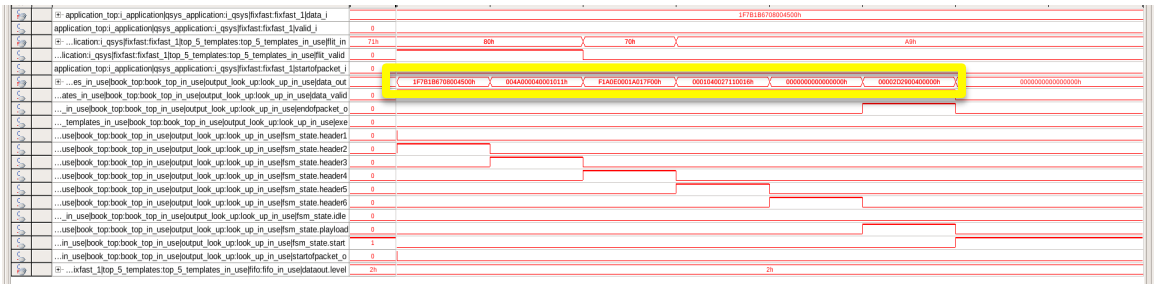
When all specified flits arrive, the valid_b would go high and output it. In the next cycle they would be reset and wait for new data.

Whole hardware modules:

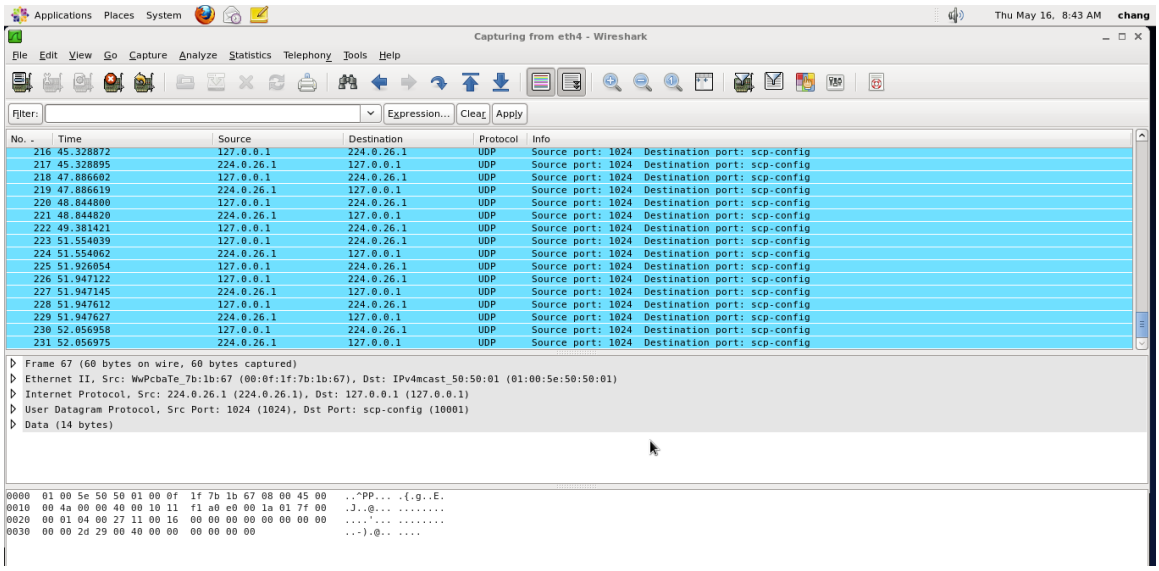


When a new command is formed, after several clocks the data_o will output the correct snapshot of the book with IP header and UDP header.

3.8 Runtime Verification: Signal Tap



Here we tested on the Solarflare AoE board. The highlight data_out is sent to Ethernet module, and is exactly what we expected (the same as result of simulation in Modelsim).



From the above figure we can see that the sink can receive the correct data come from the board (source 224.0.26.1, destination 127.0.0.1) .

4 Software Design

The software design is divided into two parts: Software Verification and Software Support. Software Verification implement the same algorithm used in Hardware Design and compare the outputs to golden output so as to make sure we are using the correct decoding algorithm and the final output is valid. Software Support is mainly about using script languages such as Python, Perl and Shell Script to help us parse files, auto generate VHDL testbench and simplify the operations in compile, simulation and result comparison.

4.1 Software Verification—Decoder

4.1.1Introduction

Software validation and verification involved building a software equivalent to the hardware decoder to decode the financial market data which would have been encoded using the standard FIX/FAST standard protocol which has been widely adapted for exchange of Financial Information and has been proved to be efficient and hence successful over the years.

FIX/FAST– **a quick introduction**

Message format

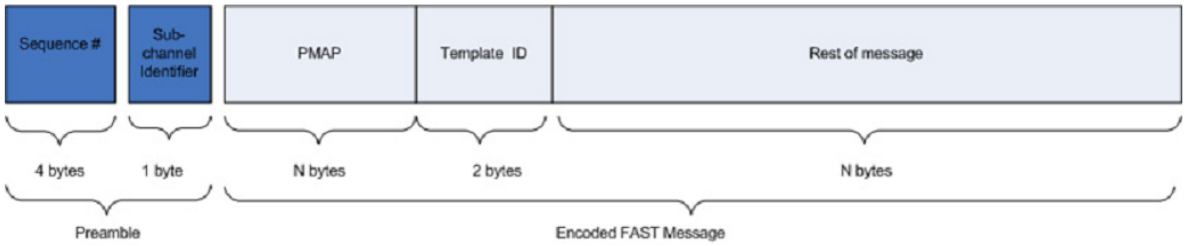


Fig1 - Fundamental FIX/FAST Message structure

Each FIX/FAST message would be prepended with a message-sequence-number that uniquely identifies a FIX/FAST message within a network packet. Here, PMAP and template-id are the two important fields. PMAP or presence map is basically a sequence of one or more bytes wherein the least seven significant bits of each of those bytes correspond to the presence/absence of a specific optional field in a template uniquely identified by the field, template-id. Length of the fields are as indicated in the diagram above – some vary and some other, fixed.

Given that fields can be basically of most common types, integers and string, the length of such fields are expressed in terms of bit-widths, 16-bit, 32-bit, and 64-bit talking about integers. And for strings, a byte of special meaning is used. This byte is termed as **Stop-Byte**. A **Stop-Byte** is a byte with the value 0x80. Essentially, we will be interested in the least significant 7-bits of every byte in the packet when extracting data but will examine the bytes with only their MSB set i.e., 0x80 marking the end of a particular FIX/FAST message field. There can also sequences of fields within a template. These sequences are limited by the sequence length present at the beginning of the sequence.

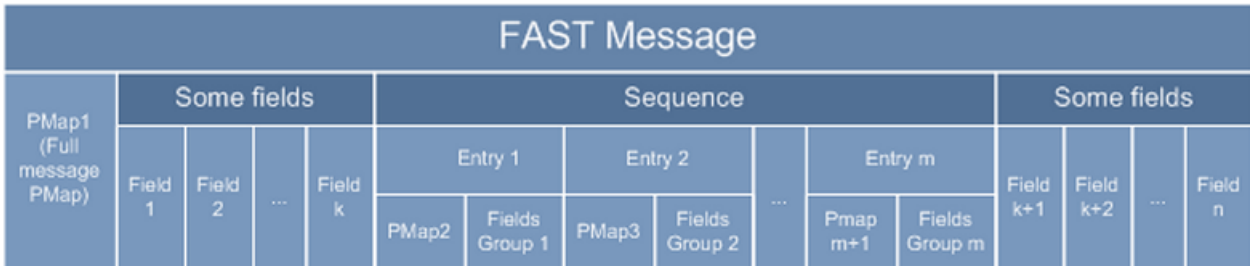


Fig 2 – Detailed FIX/FAST Message structure

Sample template and the significance of fields within – **a brief description**

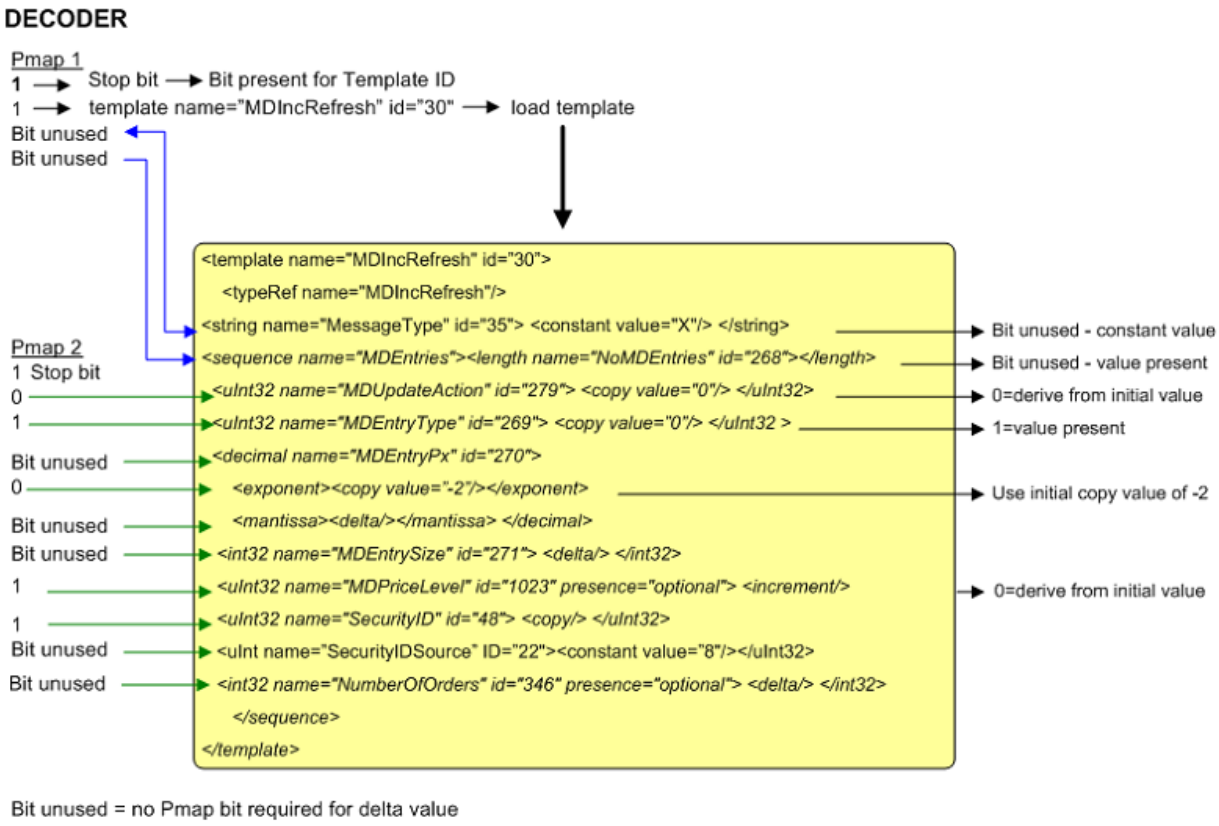


Fig 3 – Typical representation of a template and the meaning of fields within

4.1.2 Decoder

The software decoder has been implemented for a specific scope of templates namely, 117, 122, 125, 129, and 131. Out of these template-117 stands out as the most frequently occurring template. And out of these, template-131 has been most comprehensive and challenging one to decode due to its considerable number of optional fields.

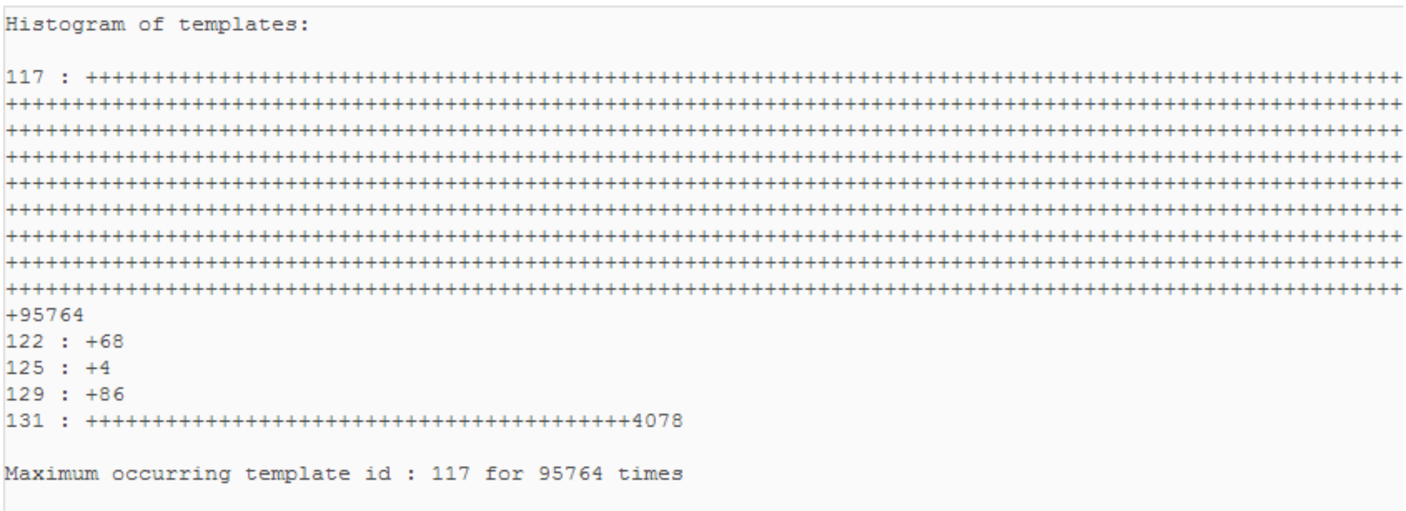


Fig 4 – Histogram of occurrences of templates

The software decoder has been implemented in C for the sake of performance and flexibility in accessing bytes as low as the individual bits, over any high-level and/or interpreted language.

A typical execution of the software decoder implemented in ***decode.c*** is depicted below.

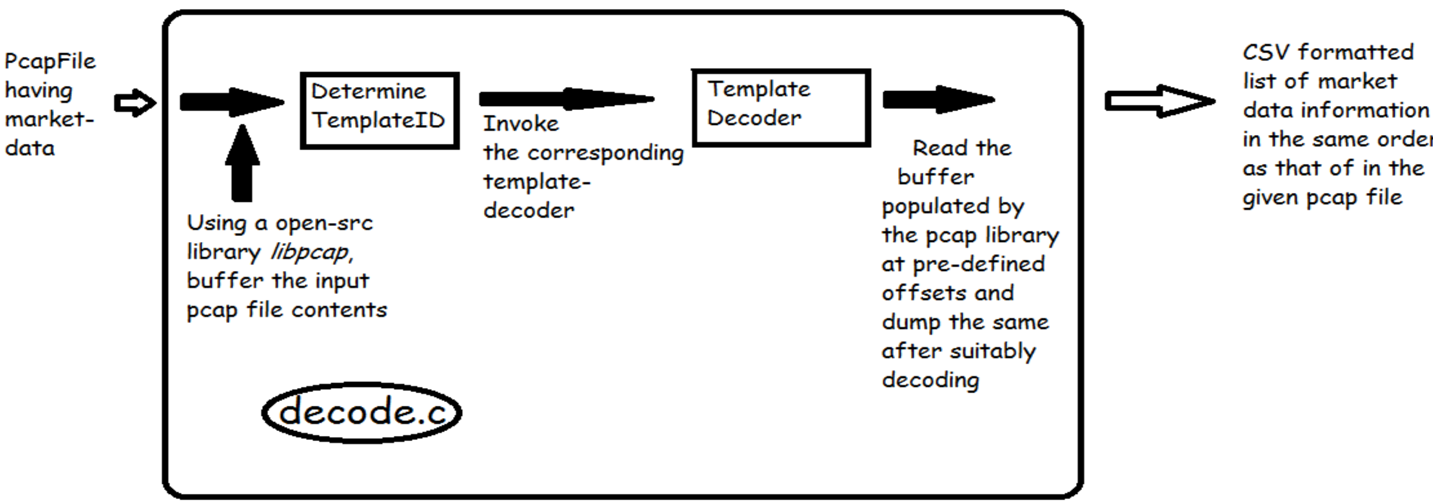


Fig – 5 A typical execution of software decoder

4.1.3 Working

The above diagram Fig-5 is very much self-explanatory in conveying the intrinsic of the software decoding part. In addition, I would like to mention that the CSV generated by such a decoder will be fed into a Python script developed by Chang which extracts the needed fields to form a Book which ultimately is transmitted over a network in a pcap format. So, the decoder provide a means of verifying the **algorithm** being used for decoding each of the template being consistent with that of the hardware decoder implemented in VHDL designated to run on the actual board

4.1.4 Experience

The overall implementation of the software decoder has been quite an experience in itself.

Background work

I and Chang started off discussing with Prof. David Lariviere. In addition, we spent several days and nights brainstorming the significance of the decoding logic. Especially, understanding the logic behind the pmap took a huge amount of time and thereby rendered the implementation challenging. Once we got a bird's eye-view of what has to be accomplished we moved ahead with the actual coding.

Actual work

Chang implemented initially the decoder for the template-117. Thereafter, I implemented all of the decoders for the remaining templates and it was quite challenging and time-consuming, at the same time educative as well. I am saying it as challenging because the other templates, especially template 131 has quite a few optional fields and to decode them, examining the PMAP was more or less problematic as some packets lacked pmaps. Later on, as a result of core discussion and severe introspection, I could complete all of the decoders based on the fact that whenever a byte with a value 0x80 is seen in the data sequence, then the corresponding field in the respective sequence of a template is termed as absent. And also, it took a while to figure out there were indeed a sequence of optional fields within a single packet. This sequence-logic was earlier not thought of by me. But, after comparing the output with the given golden reference CSV file output by David, I could completely list the entire set of sequences of fields within a template.

Lessons learned

My involvement in the project has been quite enthusiastic as I used to actively partake in the lively discussions with my entire team of FIX/FAST project as well as with David. There were few instances wherein we literally argued which method/language/logic has to be used for solving or understanding or interpreting a particular problem. However, the outcomes of these were fruitful toward ensuring active involvement of each team member rather than being always silent and careless which is undesirable in any aspect of life.

4.1.5 Conclusion

The developed C decoder was very useful in verifying the sanity of the hardware decoder in written in VHDL and vice-versa.

Overall, it was a really nice and great learning experience to me.

4.2 Software Verification—Book Builder

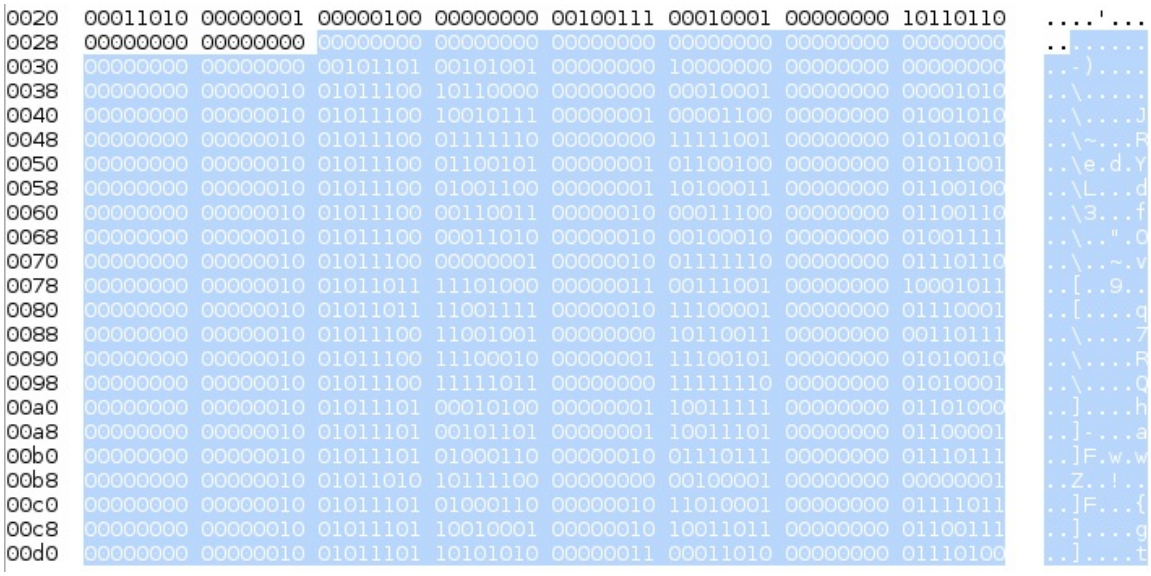
In software Book Builder are wrote in Python. We use the golden output as input and take advantage of “csv” library to help us extract information such as “MDUpdateAction” “MDPriceLevel” and so on to build the book. As for output, we import “pack” function from “struct” library to help us pack data in the UDP payload and “dpkt” library to finally put all generated packets into one pcap file.

In our program bid book and ask book are implemented separately. Each of them is implemented as a list of dictionary structure. Each dictionary in the list represent a certain price level with information we need, shown as figure below:

```
{'NoO': '11', 'isValid': 1, 'EntryPx': '154800', 'EntrySize': '23'}
{'NoO': '74', 'isValid': 1, 'EntryPx': '154775', 'EntrySize': '268'}
{'NoO': '82', 'isValid': 1, 'EntryPx': '154750', 'EntrySize': '249'}
{'NoO': '89', 'isValid': 1, 'EntryPx': '154725', 'EntrySize': '356'}
{'NoO': '100', 'isValid': 1, 'EntryPx': '154700', 'EntrySize': '419'}
{'NoO': '102', 'isValid': 1, 'EntryPx': '154675', 'EntrySize': '540'}
{'NoO': '79', 'isValid': 1, 'EntryPx': '154650', 'EntrySize': '546'}
{'NoO': '118', 'isValid': 1, 'EntryPx': '154625', 'EntrySize': '638'}
{'NoO': '139', 'isValid': 1, 'EntryPx': '154600', 'EntrySize': '825'}
{'NoO': '113', 'isValid': 1, 'EntryPx': '154575', 'EntrySize': '737'}
===>>>===>>>===>>>===>>>===>>>
{'NoO': '55', 'isValid': 1, 'EntryPx': '154825', 'EntrySize': '179'}
{'NoO': '82', 'isValid': 1, 'EntryPx': '154850', 'EntrySize': '485'}
{'NoO': '81', 'isValid': 1, 'EntryPx': '154875', 'EntrySize': '254'}
{'NoO': '104', 'isValid': 1, 'EntryPx': '154900', 'EntrySize': '415'}
{'NoO': '97', 'isValid': 1, 'EntryPx': '154925', 'EntrySize': '413'}
{'NoO': '119', 'isValid': 1, 'EntryPx': '154950', 'EntrySize': '631'}
{'NoO': '1', 'isValid': 1, 'EntryPx': '154300', 'EntrySize': '33'}
{'NoO': '123', 'isValid': 1, 'EntryPx': '154950', 'EntrySize': '721'}
{'NoO': '103', 'isValid': 1, 'EntryPx': '155025', 'EntrySize': '667'}
{'NoO': '116', 'isValid': 1, 'EntryPx': '155050', 'EntrySize': '794'}
```

List above the arrow is the bid book and list below the arrow is the ask book.

The content of book snapshot is shown as below:



The data in blue is the payload. The first six bytes are useless. The reason we add them at the beginning of the payload is because the Avalon bus sends out eight bytes at a time so we are using this six bytes to fill up two bytes left in the UPD header in

hard ware so as to simplify the processing operation. Then the following four bytes represent the Security ID for each operation. After the Security ID, we have two bytes for bid book and two bytes for ask book (actually for each book only 10 bits are used as we have 10 price levels in each book) to indicate if the corresponding price level has been changed in this operation. If the price level is changed, the corresponding bit will set to 1. So in the figure about, it is "00000000 10000000" for bid book and "00000000 00000000" for ask book, which means that the 9th price level in bid book is changed in this operation. The rest of the payload is the detailed content of bid book and ask book. For each price level we use 8 bytes to represent its content. Thus, in our example where bid book and ask book each has 10 price levels, we have 160 bytes to represent contents of the book.

For each price level, the consistence of the 8 bytes is as follow:

EntryPx	MDEntrySize	NumberOfOrders
4 bytes	2 bytes	2 bytes

4.3 Software Support

4.3.1 Auto Generation of VHDL testbench

Before we can combine the Decoder module and Book Builder Module together. We are designing them in parallel. To verify the correctness of Book Builder independently, we need generate input for the simulation. As we have more then 50 thousands inputs, it's a huge project to write it manually. So we write a Perl script to parse the golden output file to get the values of input and using the script to auto generate VHDL testbench according to the values we get.

4.3.2 Auto Generation of VHDL Decoder for All Templates

In our project, we are faced with 5 fix-fast templates. We wrote the Decoder for the most frequent used templates template 117 and template 131 manually. It needs a lot of time if we wish to write a Decoder for each template that we encounter. Moreover, fix-fast in total has more than 100 templates. So once we have the correct decoding algorithm, we use a Python script to parse the XML file which includes information of all templates and auto generate the VHDL Decoders for all templates accordingly.

4.3.3 Auto Comparison of Decoder Result & Golden Output

As the output of hardware are merely consisted of digit 1 or 0. The data format in the Decoder output and Golden output are not exactly the same. Moreover, the Golden output includes information that is not needed in our design. So we wrote Perl scripts to convert the simulation output into the same format of the Golden output and filter the data provided in the Golden output so as we can use "diff" command directly. Then we wrote a shell script that helps us do all the operations in one time. If the simulation result is exactly the same as the golden output, we get a "Perfect!" in the console.

4.3.4 TCL Script to Enable Simulation in One Command

We also write a TCL Script which help use compile all the VHDL codes and run the simulation in ModelSim in one command.

5 Contribution

In this project, Danqing Hua and Junkang Ren are responsible for the hardware part and software part is done by Chang Liu and Raghavan Santhanam.

6 Source Code Listing

6.1 Hardware Part

----

packetizer.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity pack_ff is
port(
clk: in std_logic;
rst : in std_logic;

valid_i: in std_logic;
data_i: in std_logic_vector(63 downto 0);                    --- input data (8 bytes)
                                                             -- clock input
startofpacket: in std_logic;                                 -- indicates when packet start arriving
endofpacket: in std_logic;                                   -- indicates when packet stops receving

valid_o : out std_logic_vector(7 downto 0);
        ----- Goes low till the point fresh data comes in from next packet
data_o : out std_logic_vector (63 downto 0)                  ----Payload Data ( Kevin's Data)
);

end pack_ff;


architecture rtl of pack_ff is
type states is (idle, header, leng, playload);
signal fsm_state: states;
signal count_header: integer;
signal count_playload: integer;
signal valid_temp: std_logic_vector(7 downto 0);
signal valid_o_temp: std_logic_vector(7 downto 0);
begin
process(clk, rst)
begin
if clk'event and clk='1' then
    if rst='1' then
        fsm_state<=idle;
        count_header<=0;
        count_playload<=0;
else
    case fsm_state is
-------------------------------------------------------------------------------------------------
      when idle=>
```

```
                if startofpacket='1' then
                        fsm_state<=header;
                    else
                        fsm_state<=idle;
                    end if;
                    count_header<=0;
                    count_playload<=0;
-------------------------------------------------------------------------------------------------
        when    header=>
                if valid_i='1' then
                        count_header<=count_header+1;
                        end if;

                if (count_header=4 and valid_i='1') then
                        fsm_state<=leng;
                        count_playload<= conv_integer(data_i(15 downto 0)) -8;
                        end if;



-------------------------------------------------------------------------------------------------
         when leng=>
                    if    valid_i='1' then
                            fsm_state<=playload;
                                    count_playload<=count_playload -6;
                        end if;

-------------------------------------------------------------------------------------------------
         when    playload=>
                    if valid_i='1' then
                       if (count_playload <=8 ) then
                            fsm_state<=idle;
                        else
                            fsm_state<=playload;
                        end if;
                count_playload<=count_playload -8;
                        end if;
                end case;
end if;
end if;
end process;
```

```
data_o(15 downto 0)<=data_i(15 downto 0);

with fsm_state select
data_o(63 downto 16)<= data_i(47 downto 0) when leng,
                       data_i(63 downto 16) when others;



with count_playload select
valid_temp<= "00000000" when 0,
             "10000000" when 1,
             "11000000" when 2,
             "11100000" when 3,
                    "11110000" when 4,
                    "11111000" when 5,
                    "11111100" when 6,
                    "11111110" when 7,
                    "11111111" when 8,
                    "11111111" when others;


with fsm_state select
valid_o_temp<=   "00000000" when idle,
            "00000000" when header,
                 "11111100" when leng,
                       valid_temp when playload,
                       "00000000" when others;


valid_out_gen:
for i in 0 to 7 generate
valid_o(i)<= valid_o_temp(i) and valid_i;
end generate;
end rtl;
```

---

packetizer_fifo.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity packet_buffer is
port (
clk : in std_logic;
rst : in std_logic;
data_valid :in std_logic_vector(7 downto 0);        ---1 valid bit for each byte
data_8byte : in std_logic_vector(63 downto 0);
flit : out std_logic_vector(7 downto 0);
flit_valid: out std_logic;
empty: out std_logic;
end_p:          in std_logic;
end_of_packet:      out std_logic
);
end entity;

architecture rtl of packet_buffer is
type byte_valid is array (127 downto 0) of std_logic_vector(7 downto 0);
type local_memory is array (127 downto 0) of std_logic_vector(63 downto 0);
type end_packet_memory is array (127 downto 0) of std_logic;
signal reg_valid : byte_valid;
signal reg_data    : local_memory;
signal reg_eop         : end_packet_memory;
type tp_empty is array (127 downto 0) of std_logic;
signal empty_local : tp_empty;
signal count              : natural range 0 to 7;
signal tail               : natural range 0 to 127;
signal head               : natural range 0 to 127;



begin
process (clk, rst) begin
if clk'event and clk='1' then
    if rst='1' then                                --reset everything
      for i in 0 to 127 loop
      reg_valid(i)<=(others=>'0');
      reg_data(i)<=(others=>'0');
      reg_eop(i) <= '0';
```

```
        count<=0;
        end loop;
        head<=0;
        tail<=0;


else
        if (empty_local(head)='1' and data_valid/="00000000") then                --the write process

          for i in 0 to 7 loop
              reg_valid(head)(7 - i) <= data_valid(0 + i);
              reg_data(head)(63 - 8 * i downto 56 - 8 * i) <= data_8byte(7 + 8 * i downto 0 + 8 * i);
          end loop;

          reg_eop(head) <= end_p;

              if head=127 then head<=0;
              else    head<=head+1;
              end if;
        end if;

        if(empty_local(tail)='0' ) then                          --read process
              if (count/=7 ) then count<=count+1;            --always move the pointer every clk
              else count<=0;
              end if;
              reg_valid(tail)(count)<='0';
        end if;

        if (count/=0 and    empty_local(tail)='1') then
          if (tail=127) then tail<=0;
          else tail<=tail+1;
          end if;
          count<=0;
        end if;
end if;
end if;
end process;

empty<= empty_local(head);

empty_gen:
```

```
for i in 0 to 127 generate
with reg_valid(i) select
empty_local(i)    <= '1' when "00000000",
                     '1' when "10000000",
                     '0' when others;
end generate;

flit<= reg_data(tail)((8*count +7 ) downto 8*count);
flit_valid<=reg_valid(tail)(count);

process (tail, count, reg_valid, reg_eop)

begin
if ( reg_valid(tail)(count) = '0' ) then
        end_of_packet <= '0';
elsif( count = 7 ) then
        end_of_packet <= reg_eop(tail);
elsif( reg_valid(tail)(count + 1) = '0') then
        end_of_packet <= '1';
else
        end_of_packet <= '0';
end if;
end process;



end rtl;
```

---

top_header.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
library std;
```

```
use std.standard.string;
use std.textio.all;

use work.global_constants.all;
use work.header_fields.all;

entity top_header is
port(
                clk             : in std_logic;
                rst             : in std_logic;
                    startofpacket_i: in std_logic;
                    endofpacket_i: in std_logic;
                data_i : in std_logic_vector(63 downto 0);
                    valid_i : in std_logic;
                flit            : out std_logic_vector(7 downto 0);
                flit_valid      : out std_logic);
end entity;

architecture rtl of top_header is

component packetizer_alt is
generic(
input_width: integer:=64;
bit_width48:integer:=48
);
port(
valid_i : in std_logic;
data_i: in std_logic_vector(input_width-1 downto 0);
clk: in std_logic;
start_packet: in std_logic;
end_packet: in std_logic;
data_valid : out std_logic_vector ( 7 downto 0);
data_out : out std_logic_vector (63 downto 0);
end_of_packet: out std_logic;
rst         : in std_logic );
end component;


component pack_ff is
port(
clk: in std_logic;
```

```
rst : in std_logic;

valid_i: in std_logic;
data_i: in std_logic_vector(63 downto 0);                    --- input data (8 bytes)
                                                             -- clock input
startofpacket: in std_logic;                                 -- indicates when packet start arriving
endofpacket: in std_logic;                                   -- indicates when packet stops receving

valid_o : out std_logic_vector(7 downto 0);
        ----- Goes low till the point fresh data comes in from next packet
data_o : out std_logic_vector (63 downto 0)                  ----Payload Data ( Kevin's Data)
);

end component;




component packet_buffer is
port (
        clk             : in std_logic;
        rst             : in std_logic;
        data_valid      : in std_logic_vector(7 downto 0);      ---1 valid bit for each byte
        data_8byte      : in std_logic_vector(63 downto 0);
        flit            : out std_logic_vector(7 downto 0);
        flit_valid      : out std_logic;
        empty           : out std_logic;
        end_p           : in std_logic;
        end_of_packet:out std_logic
        );
end component;

signal data_valid   :   std_logic_vector(7 downto 0);       ---1 valid bit for each byte
signal data_8byte       :   std_logic_vector(63 downto 0);
signal end_of_packet : std_logic;
signal end_of_packet_m : std_logic;
signal empty            : std_logic;
begin

packetizer_in_use : pack_ff port map
```

```
        ( valid_i=>valid_i,
          data_i=>data_i,
          clk=>clk,
          startofpacket=>startofpacket_i,
          endofpacket=>endofpacket_i,
          valid_o=>data_valid,
          data_o=>data_8byte,
          rst=>rst);

packet_buffer_in_use: packet_buffer port map
        (
        clk => clk,
        rst => rst,
        data_valid => data_valid,
        data_8byte => data_8byte,
        flit => flit,
        flit_valid => flit_valid,
        empty => empty,
        end_p => end_of_packet_m,
        end_of_packet => end_of_packet);

end rtl;
```

---

decoder_117.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity decoder_117 is
port(                   flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input  : in std_logic;
--              template_ID         : out std_logic_vector ( 7 downto 0);
--              valid_ID        : out std_logic;
                flit_out        : out std_logic_vector ( 11 downto 0);
```

```
                end_of_packet      : in std_logic;
                end_of_command  : out std_logic;
                end_p_local : out std_logic;
                select_en              : out std_logic);
end decoder_117;

architecture FSM of decoder_117 is
-------------------------------------------------------------------------------------------------------------------
type state is (
                  START,
                  Pmap,
                  templateID,
                  MsgSeqNum,
                  SendingTime,
                  PosDupFlag,
                  TradeDate,
                  NoMDEntries,
                  MDUpdateAction,
                  MDPriceLevel,
                  MDEntryType,
                  MDEntryTime,
                  SecurityID,
                  RptSeq,
                  MDEntryPx_base,
                       MDEntryPx_m,
                  MDEntrySize,
                  NumberOfOrders,
                  TradingSessionID,
                  dummy);                                                --all possible states--
-------------------------------------------------------------------------------------------------------------------
signal pr_state, nx_state: state;
signal count : std_logic_vector ( 2 downto 0 ):= "000";
signal reg_templateID : std_logic_vector ( 7 downto 0):= (others => '0');
signal reg_Pmap :std_logic_vector (7 downto 0):= (others => '0');    ---control signals----
signal reg_valid_ID: std_logic;
signal state_no : std_logic_vector (3 downto 0);



-------------------------------------------------------------------------------------------------------------------
signal reg_MsgSeqNum: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SendingTime: std_logic_vector (15 downto 0):=(others => '0');
```

```
signal reg_PosDupFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NoMDEntries: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDUpdateAction: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDPriceLevel: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryType: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SecurityID: std_logic_vector (31 downto 0):=(others => '0');
signal reg_RptSeq: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryPx_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_MDEntryPx_base: std_logic_vector(7 downto 0):=(others=>'0');
signal reg_MDEntrySize: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NumberOfOrders: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradingSessionID: std_logic_vector (15 downto 0):=(others => '0');
-----------------------------------------------------------------------------------------------------------------------


begin
process (clk)
        begin
                if ( clk'event and clk = '1' ) then
                        if ( rst = '1') then
                        pr_state <= START;
                        count <= "000";
                        reg_valid_ID <= '0';
                        reg_templateID <= (others => '0');
                        reg_MsgSeqNum<= (others=>'0');
                        reg_SendingTime<= (others=>'0');
                        reg_PosDupFlag<= (others=>'0');
                        reg_TradeDate<= (others=>'0');
                        reg_NoMDEntries<= (others=>'0');
                        reg_MDUpdateAction<= (others=>'0');
                        reg_MDPriceLevel<= (others=>'0');
                        reg_MDEntryType<= (others=>'0');
                        reg_MDEntryTime<= (others=>'0');
                        reg_SecurityID<= (others=>'0');
                        reg_RptSeq<= (others=>'0');
                        reg_MDEntryPx_base<= (others=>'0');
                        reg_MDEntryPx_m<= (others=>'0');
                        reg_MDEntrySize<= (others=>'0');
                        reg_NumberOfOrders<= (others=>'0');
                        reg_TradingSessionID<= (others=>'0');
```

```
                end_of_command <= '0';


        elsif    valid_input = '1' then
           if (pr_state=START )    then count<=count + "001";
        end if;

           case pr_state is
-------------------------------------------------------------------------------------------------------------------------------------------------------
                when START =>                -- throw away first five bytes sequence number and channel number.
                        if (count = "100") then
                                pr_state <= Pmap;
                        else
                                pr_state <= START;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

                when Pmap =>                                    -- find Pmap.
                        reg_Pmap <= flit_in;
                        if (flit_in(6) = '1' ) then                     -- has templateID;
                                pr_state <= templateID;
                        elsif(reg_templateID ="11110101" and reg_valid_ID='1') then
                               pr_state <=MsgSeqNum;
                        else
                                        pr_state<=dummy;
                        end if;
                        count<=(others=>'0');
-------------------------------------------------------------------------------------------------------------------------------------------------------
                when templateID =>                                              -- find templateID
                       reg_templateID <= flit_in;               -- save the templateID by clock rising edge
                        reg_valid_ID<='1';
                        if ( flit_in = "11110101") then
                               pr_state <=MsgSeqNum;
                        else
                               pr_state<=dummy;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
           when dummy    =>
                if ( end_of_packet='1' ) then pr_state<=START;
                end_of_command <= '0';
```

```
                else pr_state<=dummy;
                        end if;


-------------------------------------------------------------------------------------------------------------------------------------------------------------------

        when MsgSeqNum=>
                reg_MsgSeqNum( 7 downto 0) <= flit_in;
                reg_MsgSeqNum(15 downto 8) <= reg_MsgSeqNum(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=SendingTime ;
                   else
                        pr_state <=MsgSeqNum;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

        when SendingTime=>
                reg_SendingTime( 7 downto 0) <= flit_in;
                reg_SendingTime(15 downto 8) <= reg_SendingTime(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=PosDupFlag ;
                   else
                        pr_state <=SendingTime;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

        when PosDupFlag=>
                reg_PosDupFlag( 7 downto 0) <= flit_in;
                reg_PosDupFlag(15 downto 8) <= reg_PosDupFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=TradeDate ;
                   else
                        pr_state <=PosDupFlag;
                   end if;

        when TradeDate=>
                reg_TradeDate( 7 downto 0) <= flit_in;
                reg_TradeDate(15 downto 8) <= reg_TradeDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=NoMDEntries ;
                   else
                        pr_state <=TradeDate;
```

```
            end if;

when NoMDEntries=>
        reg_NoMDEntries( 7 downto 0) <= flit_in;
        reg_NoMDEntries(15 downto 8) <= reg_NoMDEntries(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDUpdateAction ;
           else
                pr_state <=NoMDEntries;
           end if;

when MDUpdateAction=>
        reg_MDUpdateAction( 7 downto 0) <= flit_in;
        reg_MDUpdateAction(15 downto 8) <= reg_MDUpdateAction(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDPriceLevel ;
           else
                pr_state <=MDUpdateAction;
           end if;
        if(reg_NoMDEntries(7 downto 0)="10000001") then
                end_of_command <= '1';
        end if;

when MDPriceLevel=>
        reg_MDPriceLevel( 7 downto 0) <= flit_in;
        reg_MDPriceLevel(15 downto 8) <= reg_MDPriceLevel(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDEntryType ;
           else
                pr_state <=MDPriceLevel;
           end if;

when MDEntryType=>
        reg_MDEntryType( 7 downto 0) <= flit_in;
        reg_MDEntryType(15 downto 8) <= reg_MDEntryType(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDEntryTime ;
           else
                pr_state <=MDEntryType;
           end if;
```

```
when MDEntryTime=>
        reg_MDEntryTime( 7 downto 0) <= flit_in;
        reg_MDEntryTime(15 downto 8) <= reg_MDEntryTime(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=SecurityID ;
           else
                pr_state <=MDEntryTime;
           end if;

when SecurityID=>
        reg_SecurityID( 7 downto 0) <= flit_in;
        reg_SecurityID(31 downto 8) <= reg_SecurityID(23 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=RptSeq ;
           else
                pr_state <=SecurityID;
           end if;

when RptSeq=>
        reg_RptSeq( 7 downto 0) <= flit_in;
        reg_RptSeq(15 downto 8) <= reg_RptSeq(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDEntryPx_base ;
           else
                pr_state <=RptSeq;
           end if;

when MDEntryPx_base=>
        reg_MDEntryPx_base( 7 downto 0) <= flit_in;
        if ( flit_in(7) = '1') then
          pr_state <=MDEntryPx_m ;
           else
                pr_state <=MDEntryPx_base;
           end if;

when MDEntryPx_m=>
        reg_MDEntryPx_m( 7 downto 0) <= flit_in;
        reg_MDEntryPx_m( 15 downto 8) <= flit_in;
        reg_MDEntryPx_m( 23 downto 16) <= flit_in;
        if ( flit_in(7) = '1') then
          pr_state <=MDEntrySize ;
```

```
                else
                    pr_state <=MDEntryPx_m;
                end if;

            when MDEntrySize=>
                reg_MDEntrySize( 7 downto 0) <= flit_in;
                reg_MDEntrySize(15 downto 8) <= reg_MDEntrySize(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=NumberOfOrders ;
                  else
                    pr_state <=MDEntrySize;
                  end if;

            when NumberOfOrders=>
                reg_NumberOfOrders( 7 downto 0) <= flit_in;
                reg_NumberOfOrders(15 downto 8) <= reg_NumberOfOrders(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=TradingSessionID ;
                  else
                    pr_state <=NumberOfOrders;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------
            when TradingSessionID=>
                reg_TradingSessionID( 7 downto 0) <= flit_in;
                reg_TradingSessionID(15 downto 8) <= reg_TradingSessionID(7 downto 0);
                if ( flit_in(7) = '1') then
              reg_NoMDEntries <=reg_NoMDEntries -"0000000000000001";

                  if (reg_NoMDEntries (7 downto 0) = "10000001") then
                    pr_state<=START;
                    end_of_command <= '0';

                   reg_NoMDEntries<= (others=>'0');
                  else
                    pr_state<= MDUpdateAction;
                  end if;
                else
                    pr_state<= TradingSessionID;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------
```

```
                    end case;
                end if;
        end if;

        end process;


flit_out(7 downto 0)<= flit_in;
---pass the 8 bit flit directly without cycle delay
With pr_state select
flit_out(11)<= valid_input when NumberOfOrders ,
                    valid_input when MDEntrySize,
                    valid_input when MDEntryPx_base,
                    valid_input when MDEntryPx_m,
                    valid_input when SecurityID,
                    valid_input when MDEntryType,
                    valid_input when MDPriceLevel,
                    valid_input when MDUpdateAction,
                     '0'            when others;

with pr_state select
flit_out(10 downto 8)<= "101" when MDEntrySize,
                                "000" when NumberOfOrders,
                                        "100" when MDEntryPx_m,
                                        "100" when MDEntryPx_base,
                                        "011" when SecurityID,
                                "010" when MDPriceLevel,
                                        "001" when MDUpdateAction,
                                  "110" when MDEntryType,
                                        "111" when others;
---add 4 bit identifier for every bit according to the FSM state



--template_ID<=reg_templateID;

with pr_state select
select_en<= '0' when START,
                '0' when Pmap,
                '0' when templateID,
                '0' when dummy,
```

```
                    '1' when others;

with pr_state select
state_no<= "0001" when START,
                                "0010" when MsgSeqNum,
                                "0011" when NoMDEntries,
                                "0100" when MDUpdateAction,
                                "0000" when others;

process(pr_state, reg_NoMDEntries)
begin
if (( pr_state = TradingSessionID) and ( reg_NoMDEntries (7 downto 0) = "10000001" )) then
end_p_local <= '1' ;
else
end_p_local <= '0' ;
end if;
end process;




end FSM;

----------------------------------------------------------
decoder_122
----------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity decoder_122 is
port(                   flit_in          : in std_logic_vector ( 7 downto 0 );
                clk, rst         : in std_logic;
                valid_input  : in std_logic;
--              template_ID          : out std_logic_vector ( 7 downto 0);
--              valid_ID       : out std_logic;
                flit_out         : out std_logic_vector ( 11 downto 0);
                end_of_packet      : in std_logic;
                end_p_local          : out std_logic;
                end_of_command  : out std_logic;
```

```
        select_en     : out std_logic;
        flag          : out std_logic;                    -- indicates whether template 122 or 125
        flags             : out std_logic_vector(2 downto 0));    -- indicates optional specified fields

end decoder_122;

architecture FSM of decoder_122 is
-------------------------------------------------------------------------------------------------------------
type state is (
            START,
            Pmap,
            templateID,
            MsgSeqNum,
            SendingTime,
            PosDupFlag,
            TradeDate,
            NoMDEntries,
            MDUpdateAction,
            MDEntryType,
            SecurityID,
            RptSeq,
            MDEntryPx_base,
        MDEntryPx_m,
            MDEntrySize,
            NetChgPrevDa_base,
        NetChgPrevDa_m,
            TradeVolume,
            TickDirection,
            TradeCondition,
            MDEntryTime,
            AggressorSide,
            MatchEventIndicator,
            dummy);                                         --all possible states--
-------------------------------------------------------------------------------------------------------------
signal pr_state, nx_state: state;
signal count : std_logic_vector ( 2 downto 0 ):= "000";
signal reg_templateID : std_logic_vector ( 7 downto 0):= (others => '0');
signal reg_Pmap :std_logic_vector (7 downto 0):= (others => '0');    ---control signals----
signal reg_valid_ID: std_logic;
-------------------------------------------------------------------------------------------------------------
signal reg_MsgSeqNum: std_logic_vector (15 downto 0):=(others => '0');
```

```
signal reg_SendingTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_PosDupFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NoMDEntries: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDUpdateAction: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryType: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SecurityID: std_logic_vector (31 downto 0):=(others => '0');
signal reg_RptSeq: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryPx_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_MDEntryPx_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_MDEntrySize: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NetChgPrevDa_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_NetChgPrevDa_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_TradeVolume: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TickDirection: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeCondition: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_AggressorSide: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MatchEventIndicator: std_logic_vector (15 downto 0):=(others => '0');
-----------------------------------------------------------------------------------------------------------------------


begin
process (clk)
        begin
                if ( clk'event and clk = '1' ) then
                        if ( rst = '1') then
                        pr_state <= START;
                        count <= "000";
                        reg_valid_ID <= '0';
                        reg_templateID <= (others => '0');
                        reg_MsgSeqNum<= (others=>'0');
                        reg_SendingTime<= (others=>'0');
                        reg_PosDupFlag<= (others=>'0');
                        reg_TradeDate<= (others=>'0');
                        reg_NoMDEntries<= (others=>'0');
                        reg_MDUpdateAction<= (others=>'0');
                        reg_MDEntryType<= (others=>'0');
                        reg_SecurityID<= (others=>'0');
                        reg_RptSeq<= (others=>'0');
                        reg_MDEntryPx_base<= (others=>'0');
                        reg_MDEntryPx_m<= (others=>'0');
```

```
                reg_MDEntrySize<= (others=>'0');
                reg_NetChgPrevDa_base<= (others=>'0');
                reg_NetChgPrevDa_m<= (others=>'0');
                reg_TradeVolume<= (others=>'0');
                reg_TickDirection<= (others=>'0');
                reg_TradeCondition<= (others=>'0');
                reg_MDEntryTime<= (others=>'0');
                reg_AggressorSide<= (others=>'0');
                reg_MatchEventIndicator<= (others=>'0');
                end_of_command <= '0';
                flag <= '0';

            elsif    valid_input = '1'    then
               if (pr_state=START)    then count<=count + "001";
            end if;
               case pr_state is
-------------------------------------------------------------------------------------------------------------------------------------------------
                when START =>                -- throw away first five bytes sequence number and channel number.
                        if (count = "100") then
                                pr_state <= Pmap;
                        else
                                pr_state <= START;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------

                when Pmap =>                                    -- find Pmap.
               reg_Pmap <= flit_in;
                        if (flit_in(6) = '1' ) then                      -- has templateID;
                                pr_state <= templateID;
                        elsif(reg_templateID ="11111010" and reg_valid_ID='1') then
                              pr_state <=MsgSeqNum;
                           else
                         pr_state<=dummy;
                        end if;
                        count<=(others=>'0');
-------------------------------------------------------------------------------------------------------------------------------------------------
                when templateID =>                                              -- find templateID
                      reg_templateID <= flit_in;                -- save the templateID by clock rising edge
                        reg_valid_ID<='1';
                        if ( flit_in = "11111010") then
                                pr_state <=MsgSeqNum;
```

```
                    flag <= '1';
                else
                    pr_state<=dummy;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------------

          when dummy    =>
              if ( end_of_packet='1' ) then pr_state<=START;
              end_of_command <= '0';
              else pr_state<=dummy;
                    end if;


-------------------------------------------------------------------------------------------------------------------------------------------------------------

          when MsgSeqNum=>
              reg_MsgSeqNum( 7 downto 0) <= flit_in;
              reg_MsgSeqNum(15 downto 8) <= reg_MsgSeqNum(7 downto 0);
              if ( flit_in(7) = '1') then
                pr_state <=SendingTime ;
                 else
                    pr_state <=MsgSeqNum;
                 end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

          when SendingTime=>
              reg_SendingTime( 7 downto 0) <= flit_in;
              reg_SendingTime(15 downto 8) <= reg_SendingTime(7 downto 0);
              if ( flit_in(7) = '1') then
                    pr_state <=PosDupFlag ;
              else
                    pr_state <=SendingTime;
                    end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

          when PosDupFlag=>
              reg_PosDupFlag( 7 downto 0) <= flit_in;
              reg_PosDupFlag(15 downto 8) <= reg_PosDupFlag(7 downto 0);
              if ( flit_in(7) = '1') then
                pr_state <=TradeDate ;
                 else
                    pr_state <=PosDupFlag;
                 end if;
```

```
-----------------------------------------------------------------------------------------------------------------------------------------------------------

            when TradeDate=>
                reg_TradeDate( 7 downto 0) <= flit_in;
                reg_TradeDate(15 downto 8) <= reg_TradeDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=NoMDEntries ;
                   else
                        pr_state <=TradeDate;
                   end if;

-----------------------------------------------------------------------------------------------------------------------------------------------------------




            when NoMDEntries=>
                reg_NoMDEntries( 7 downto 0) <= flit_in;
                reg_NoMDEntries(15 downto 8) <= reg_NoMDEntries(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDUpdateAction ;
                   else
                        pr_state <=NoMDEntries;
                   end if;

-----------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDUpdateAction=>
                reg_MDUpdateAction( 7 downto 0) <= flit_in;
                reg_MDUpdateAction(15 downto 8) <= reg_MDUpdateAction(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryType ;
                   else
                        pr_state <=MDUpdateAction;
                   end if;
                if(reg_NoMDEntries(7 downto 0)="10000001") then
                        end_of_command <= '1';
                end if;

-----------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDEntryType=>
                reg_MDEntryType( 7 downto 0) <= flit_in;
                reg_MDEntryType(15 downto 8) <= reg_MDEntryType(7 downto 0);
```

```
                if ( flit_in(7) = '1') then
                  pr_state <=SecurityID ;
                   else
                        pr_state <=MDEntryType;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

            when SecurityID=>
                reg_SecurityID( 7 downto 0) <= flit_in;
                reg_SecurityID(31 downto 8) <= reg_SecurityID(23 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=RptSeq ;
                   else
                        pr_state <=SecurityID;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

            when RptSeq=>
                reg_RptSeq( 7 downto 0) <= flit_in;
                reg_RptSeq(15 downto 8) <= reg_RptSeq(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryPx_base ;
                   else
                        pr_state <=RptSeq;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDEntryPx_base=>
                reg_MDEntryPx_base( 7 downto 0) <= flit_in;
                if ( flit_in(7) = '1') then
                                pr_state <=MDEntryPx_m ;
                  else
                        pr_state <=MDEntryPx_base;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDEntryPx_m=>
                reg_MDEntryPx_m( 7 downto 0) <= flit_in;
                reg_MDEntryPx_m(15 downto 8) <= reg_MDEntryPx_m(7 downto 0);
                if ( flit_in(7) = '1') then
                                pr_state <=MDEntrySize ;
```

```
                else
                    pr_state <=MDEntryPx_m;
                end if;


-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDEntrySize=>
                reg_MDEntrySize( 7 downto 0) <= flit_in;
                reg_MDEntrySize(15 downto 8) <= reg_MDEntrySize(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=NetChgPrevDa_base;
                else
                        pr_state <=MDEntrySize;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when NetChgPrevDa_base=>
                reg_NetChgPrevDa_base( 7 downto 0) <= flit_in;
                if ( flit_in = "10000000") then
                                    pr_state<=TradeVolume;
                        elsif(flit_in(7) = '1') then
                                    pr_state<=NetChgPrevDa_m;
                else
                        pr_state <=NetChgPrevDa_base;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
            when NetChgPrevDa_m=>
                reg_NetChgPrevDa_m( 7 downto 0) <= flit_in;
                reg_NetChgPrevDa_m( 15 downto 8) <= reg_NetChgPrevDa_m(7 downto 0);
                reg_NetChgPrevDa_m( 23 downto 16) <= reg_NetChgPrevDa_m(15 downto 8);
                if ( flit_in(7) = '1') then
                                    pr_state<=TradeVolume;
                else
                        pr_state <=NetChgPrevDa_m;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when TradeVolume=>
                reg_TradeVolume( 7 downto 0) <= flit_in;
                reg_TradeVolume(15 downto 8) <= reg_TradeVolume(7 downto 0);
                if ( flit_in(7) = '1') then
```

```
                pr_state<=TickDirection;
                else
                        pr_state <=TradeVolume;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TickDirection=>
                reg_TickDirection( 7 downto 0) <= flit_in;
                reg_TickDirection(15 downto 8) <= reg_TickDirection(7 downto 0);
                if ( flit_in(7) = '1') then
                pr_state<=TradeCondition;
                else
                        pr_state <=TickDirection;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TradeCondition=>
                reg_TradeCondition( 7 downto 0) <= flit_in;
                reg_TradeCondition(15 downto 8) <= reg_TradeCondition(7 downto 0);
                if ( flit_in(7) = '1') then
                pr_state<=MDEntryTime;
                else
                        pr_state <=TradeCondition;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when MDEntryTime=>
                reg_MDEntryTime( 7 downto 0) <= flit_in;
                reg_MDEntryTime(15 downto 8) <= reg_MDEntryTime(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=AggressorSide;
                else
                        pr_state <=MDEntryTime;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when AggressorSide=>
                reg_AggressorSide( 7 downto 0) <= flit_in;
                reg_AggressorSide(15 downto 8) <= reg_AggressorSide(7 downto 0);
                if ( flit_in(7) = '1') then
                pr_state<=MatchEventIndicator;
```

```
                    else
                        pr_state <=AggressorSide;
                    end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MatchEventIndicator=>
                    reg_MatchEventIndicator( 7 downto 0) <= flit_in;
                    reg_MatchEventIndicator(15 downto 8) <= reg_MatchEventIndicator(7 downto 0);
                    if ( flit_in(7) = '1') then
                      if(reg_NoMDEntries(7  downto  0)/="10000001")  then                                    --end  of
sequence
                                pr_state<=MDUpdateAction;
                                  reg_NoMDEntries<=reg_NoMDEntries - "00000001";
                           else                                                                              --end of packet
                                pr_state<=START;
                                  reg_Pmap<=(others=>'0');
                                  end_of_command <= '0';
                                  flag <= '0';

                                  reg_NoMDEntries<=reg_NoMDEntries - "00000001";
                           end if;
                      else
                          pr_state <=MatchEventIndicator;
                      end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------
                    end case;
            end if;
            end if;
        end process;


flit_out(7 downto 0)<= flit_in;
---pass the 8 bit flit directly without cycle delay
With pr_state select
flit_out(11)<= --valid_input when NumberOfOrders ,
                    valid_input when MDEntrySize,
                    valid_input when MDEntryPx_m,
                    valid_input when MDEntryPx_base,
                    valid_input when SecurityID,
```

```
                valid_input when MDEntryType,
                --valid_input when MDPriceLevel,
                valid_input when MDUpdateAction,
                 '0'              when others;

with pr_state select
flit_out(10 downto 8)<= "101" when MDEntrySize,
                        --"000" when NumberOfOrders,
                "100" when MDEntryPx_base,
                "100" when MDEntryPx_m,
                "011" when SecurityID,
                --"010" when MDPriceLevel,
                    "001" when MDUpdateAction,
                        "110" when MDEntryType,
                "111" when others;
---add 4 bit identifier for every bit according to the FSM state



--template_ID<=reg_templateID;

with pr_state select
select_en <= '0' when START,
                 '0' when Pmap,
                 '0' when templateID,
                 '0' when dummy,
                 '1' when others;

--with reg_templateID select
--flag <= '1' when "11111010",
--       '0' when others;



process(pr_state, reg_NoMDEntries)
begin
if (( pr_state = MatchEventIndicator) and ( reg_NoMDEntries (7 downto 0) = "10000001" )) then
end_p_local <= '1' ;
else
end_p_local <= '0' ;
end if;
```

```
end process;

process(clk) begin
if (clk'event and clk='1') then
if (pr_state=MDEntrySize and flit_in(7)='1') then flags<="001";
else flags<="000";
end if;
end if;
end process;

end FSM;



------------------------------------------------------------
decoder_125
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity decoder_125 is
port(                   flit_in          : in std_logic_vector ( 7 downto 0 );
                clk, rst         : in std_logic;
                valid_input  : in std_logic;
--              template_ID          : out std_logic_vector ( 7 downto 0);
--              valid_ID       : out std_logic;
                flit_out         : out std_logic_vector ( 11 downto 0);
                end_of_packet      : in std_logic;
                end_p_local          : out std_logic;
                end_of_command  : out std_logic;
                select_en      : out std_logic;
                flag             : out std_logic;                        -- indicates whether template 122 or 125
                flags                   : out std_logic_vector(2 downto 0));    -- indicates optional specified fields
end decoder_125;

architecture FSM of decoder_125 is
-------------------------------------------------------------------------------------------------------------------
type state is (
                  START,
                  Pmap,
```

```
                templateID,
                MsgSeqNum,
                SendingTime,
                PosDupFlag,
                TradeDate,
                NoMDEntries,
                MDUpdateAction,
                SecurityID,
                RptSeq,
                MDEntryType,
                MDEntryPx_base,
            MDEntryPx_m,
                MDEntrySize,
                MDEntryTime,
                OpenCloseSettleFlag,
                SettlDate,
                FixingBracket,
                dummy);                                         --all possible states--
-------------------------------------------------------------------------------------------------------------------
signal pr_state, nx_state: state;
signal count : std_logic_vector ( 2 downto 0 ):= "000";
signal reg_templateID : std_logic_vector ( 7 downto 0):= (others => '0');
signal reg_Pmap :std_logic_vector (7 downto 0):= (others => '0');    ---control signals----
signal reg_valid_ID: std_logic;
-------------------------------------------------------------------------------------------------------------------
signal reg_MsgSeqNum: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SendingTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_PosDupFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NoMDEntries: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDUpdateAction: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SecurityID: std_logic_vector (31 downto 0):=(others => '0');
signal reg_RptSeq: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryType: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryPx_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_MDEntryPx_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_MDEntrySize: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_OpenCloseSettleFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SettlDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_FixingBracket: std_logic_vector (15 downto 0):=(others => '0');
```

```
-------------------------------------------------------------------------------------------------------------------

begin
process (rst, clk)
        begin
                if ( clk'event and clk = '1' ) then
                        if ( rst = '1') then
                        pr_state <= START;
                        count <= "000";
                        reg_valid_ID <= '0';
                        reg_templateID <= (others => '0');
                        reg_MsgSeqNum<= (others=>'0');
                        reg_SendingTime<= (others=>'0');
                        reg_PosDupFlag<= (others=>'0');
                        reg_TradeDate<= (others=>'0');
                        reg_NoMDEntries<= (others=>'0');
                        reg_MDUpdateAction<= (others=>'0');
                        reg_SecurityID<= (others=>'0');
                        reg_RptSeq<= (others=>'0');
                        reg_MDEntryType<= (others=>'0');
                        reg_MDEntryPx_base<= (others=>'0');
                        reg_MDEntryPx_m<= (others=>'0');
                        reg_MDEntrySize<= (others=>'0');
                        reg_MDEntryTime<= (others=>'0');
                        reg_OpenCloseSettleFlag<= (others=>'0');
                        reg_SettlDate<= (others=>'0');
                        reg_FixingBracket<= (others=>'0');
                        end_of_command <= '0';

                elsif (valid_input = '1' ) then
                   if (pr_state=START)    then count<=count + "001";
                end if;
                   case pr_state is
-------------------------------------------------------------------------------------------------------------------------------------------------
                        when START =>                -- throw away first five bytes sequence number and channel number.
                                if (count = "100") then
                                        pr_state <= Pmap;
                                else
                                        pr_state <= START;
                                end if;
```

```
---------------------------------------------------------------------------------------------------------------------------------------------------

                when Pmap =>                                    -- find Pmap.
                    reg_Pmap <= flit_in;
                    if (flit_in(6) = '1' ) then                  -- has templateID;
                          pr_state <= templateID;
                    elsif(reg_templateID ="11111101" and reg_valid_ID='1') then
                          pr_state <=MsgSeqNum;
                       else
                     pr_state<=dummy;
                     end if;
                     count<=(others=>'0');
---------------------------------------------------------------------------------------------------------------------------------------------------

                when templateID =>                                      -- find templateID
                    reg_templateID <= flit_in;                  -- save the templateID by clock rising edge
                     reg_valid_ID<='1';
                     if ( flit_in = "11111101") then
                          pr_state <=MsgSeqNum;
                     else
                          pr_state<=dummy;
                     end if;
---------------------------------------------------------------------------------------------------------------------------------------------------

              when dummy    =>
                   if ( end_of_packet='1' ) then pr_state<=START;
                   else pr_state<=dummy;
                          end if;



---------------------------------------------------------------------------------------------------------------------------------------------------

            when MsgSeqNum=>
                   reg_MsgSeqNum( 7 downto 0) <= flit_in;
                   reg_MsgSeqNum(15 downto 8) <= reg_MsgSeqNum(7 downto 0);
                   if ( flit_in(7) = '1') then
                     pr_state <=SendingTime ;
                      else
                          pr_state <=MsgSeqNum;
                     end if;


---------------------------------------------------------------------------------------------------------------------------------------------------

            when SendingTime=>
                   reg_SendingTime( 7 downto 0) <= flit_in;
```

```
                reg_SendingTime(15 downto 8) <= reg_SendingTime(7 downto 0);
                if ( flit_in(7) = '1') then
                        pr_state <=PosDupFlag ;
                else
                        pr_state <=SendingTime;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when PosDupFlag=>
                reg_PosDupFlag( 7 downto 0) <= flit_in;
                reg_PosDupFlag(15 downto 8) <= reg_PosDupFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=TradeDate ;
                   else
                        pr_state <=PosDupFlag;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when TradeDate=>
                reg_TradeDate( 7 downto 0) <= flit_in;
                reg_TradeDate(15 downto 8) <= reg_TradeDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=NoMDEntries ;
                   else
                        pr_state <=TradeDate;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when NoMDEntries=>
                reg_NoMDEntries( 7 downto 0) <= flit_in;
                reg_NoMDEntries(15 downto 8) <= reg_NoMDEntries(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDUpdateAction ;
                   else
                        pr_state <=NoMDEntries;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDUpdateAction=>
                reg_MDUpdateAction( 7 downto 0) <= flit_in;
                reg_MDUpdateAction(15 downto 8) <= reg_MDUpdateAction(7 downto 0);
```

```
                if ( flit_in(7) = '1') then
                  pr_state <=SecurityID ;
                   else
                        pr_state <=MDUpdateAction;
                   end if;
                if(reg_NoMDEntries(7 downto 0)="10000001") then
                        end_of_command <= '1';
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when SecurityID=>
                reg_SecurityID( 7 downto 0) <= flit_in;
                reg_SecurityID(31 downto 8) <= reg_SecurityID(23 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=RptSeq ;
                   else
                        pr_state <=SecurityID;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when RptSeq=>
                reg_RptSeq( 7 downto 0) <= flit_in;
                reg_RptSeq(15 downto 8) <= reg_RptSeq(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryType ;
                   else
                        pr_state <=RptSeq;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDEntryType=>
                reg_MDEntryType( 7 downto 0) <= flit_in;
                reg_MDEntryType(15 downto 8) <= reg_MDEntryType(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=SecurityID ;
                   else
                        pr_state <=MDEntryPx_base;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDEntryPx_base=>
```

```
                reg_MDEntryPx_base( 7 downto 0) <= flit_in;
                if ( flit_in = "10000000") then
                                        pr_state<=MDEntrySize;
                elsif ( flit_in(7) = '1') then
                                    pr_state <=MDEntryPx_m ;
                  else
                       pr_state <=MDEntryPx_base;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDEntryPx_m=>
                reg_MDEntryPx_m( 7 downto 0) <= flit_in;
                reg_MDEntryPx_m(15 downto 8) <= reg_MDEntryPx_m(7 downto 0);
                if ( flit_in(7) = '1') then
                                pr_state <=MDEntrySize ;
                  else
                       pr_state <=MDEntryPx_m;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDEntrySize=>
                reg_MDEntrySize( 7 downto 0) <= flit_in;
                reg_MDEntrySize(15 downto 8) <= reg_MDEntrySize(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryTime ;
                   else
                       pr_state <=MDEntrySize;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDEntryTime=>
                reg_MDEntryTime( 7 downto 0) <= flit_in;
                reg_MDEntryTime(15 downto 8) <= reg_MDEntryTime(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=OpenCloseSettleFlag;
                else
                       pr_state <=MDEntryTime;
                       end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when OpenCloseSettleFlag=>
```

```
                reg_OpenCloseSettleFlag( 7 downto 0) <= flit_in;
                reg_OpenCloseSettleFlag(15 downto 8) <= reg_OpenCloseSettleFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=SettlDate;
                else
                        pr_state <=OpenCloseSettleFlag;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when SettlDate=>
                reg_SettlDate( 7 downto 0) <= flit_in;
                reg_SettlDate(15 downto 8) <= reg_SettlDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=FixingBracket;
                else
                        pr_state <=SettlDate;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when FixingBracket=>
                reg_FixingBracket( 7 downto 0) <= flit_in;
                reg_FixingBracket(15 downto 8) <= reg_FixingBracket(7 downto 0);
                if ( flit_in(7) = '1') then
                        reg_NoMDEntries <=reg_NoMDEntries -"0000000000000001";
                                if(reg_NoMDEntries(7 downto 0)/="10000001") then
--end of sequence
                                pr_state<=MDUpdateAction;

                        else pr_state<= START;
                                reg_Pmap<=(others=>'0');
                                end_of_command <= '0';
                        end if;

                  else
                        pr_state <=FixingBracket;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------
                end case;
```

```
                end if;
                end if;

        end process;


flit_out(7 downto 0)<= flit_in;
---pass the 8 bit flit directly without cycle delay
With pr_state select
flit_out(11)<=-- valid_input when NumberOfOrders ,
                        valid_input when MDEntrySize,
                        valid_input when MDEntryPx_m,
                        valid_input when MDEntryPx_base,
                        valid_input when SecurityID,
                        valid_input when MDEntryType,
                       -- valid_input when MDPriceLevel,
                        valid_input when MDUpdateAction,
                         '0'              when others;

with pr_state select
flit_out(10 downto 8)<= "101" when MDEntrySize,
                                -- "000" when NumberOfOrders,
                       "100" when MDEntryPx_base,
                       "100" when MDEntryPx_m,
                       "011" when SecurityID,
                                -- "010" when MDPriceLevel,
                          "001" when MDUpdateAction,
                              "110" when MDEntryType,
                       "111" when others;
---add 4 bit identifier for every bit according to the FSM state

--template_ID<=reg_templateID;

with pr_state select
select_en <= '0' when START,
                        '0' when Pmap,
                        '0' when templateID,
                        '0' when dummy,
                        '1' when others;

with reg_templateID select
```

```
flag <= '1' when "11111101",
        '0' when others;

process(pr_state, reg_NoMDEntries)
begin
if (( pr_state = FixingBracket) and ( reg_NoMDEntries (7 downto 0) = "10000001" )) then
end_p_local <= '1' ;
else
end_p_local <= '0' ;
end if;
end process;

process(clk) begin
if (clk'event and clk='1') then
if (pr_state=MDEntrySize and flit_in(7)='1') then flags<="001";
else flags<="000";
end if;
end if;
end process;



end FSM;




----------------------------------------------------------
decoder_129
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity decoder_129 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input : in std_logic;
--              template_ID : out std_logic_vector ( 7 downto 0);
--              valid_ID        : out std_logic;
                flit_out                : out std_logic_vector ( 11 downto 0);
                end_of_packet       : in std_logic;
```

```
                end_p_local      : out std_logic;
                end_of_command  : out std_logic;
                select_en            : out std_logic;
                flags                  : out std_logic_vector(2 downto 0));
end decoder_129;

architecture FSM of decoder_129 is
-------------------------------------------------------------------------------------------------------------------
type state is (
                  START,
                  Pmap,
                  templateID,
                  MsgSeqNum,
                  SendingTime,
                  PosDupFlag,
                  TradeDate,
                  NoMDEntries,
                  MDUpdateAction,
                  MDPriceLevel,
                  MDEntryType,
                  SecurityID,
                  RptSeq,
                  QuoteCondition,
                  MDEntryPx_base,
              MDEntryPx_m,
                  NumberOfOrders,
                  MDEntryTime,
                  MDEntrySize,
                  TradingSessionID,
                  NetChgPrevDa_base,
              NetChgPrevDa_m,
                  TickDirection,
                  OpenCloseSettleFlag,
                  SettlDate,
                  dummy);                                          --all possible states--
-------------------------------------------------------------------------------------------------------------------
signal pr_state, nx_state: state;
signal count : std_logic_vector ( 2 downto 0 ):= "000";
signal reg_templateID : std_logic_vector ( 15 downto 0):= (others => '0');
signal reg_Pmap :std_logic_vector (7 downto 0):= (others => '0');    ---control signals----
signal reg_valid_ID: std_logic;
```

```
---------------------------------------------------------------------------------------------------------------
signal reg_MsgSeqNum: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SendingTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_PosDupFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NoMDEntries: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDUpdateAction: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDPriceLevel: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryType: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SecurityID: std_logic_vector (31 downto 0):=(others => '0');
signal reg_RptSeq: std_logic_vector (15 downto 0):=(others => '0');
signal reg_QuoteCondition: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryPx_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_MDEntryPx_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_NumberOfOrders: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntrySize: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradingSessionID: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NetChgPrevDa_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_NetChgPrevDa_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_TickDirection: std_logic_vector (15 downto 0):=(others => '0');
signal reg_OpenCloseSettleFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SettlDate: std_logic_vector (15 downto 0):=(others => '0');

---------------------------------------------------------------------------------------------------------------

begin
process (rst, clk)
        begin
                if ( clk'event and clk = '1' ) then
                        if ( rst = '1') then
                        pr_state <= START;
                        count <= "000";
                        reg_valid_ID <= '0';
                        reg_templateID <= (others => '0');
                        reg_MsgSeqNum<= (others=>'0');
                        reg_SendingTime<= (others=>'0');
                        reg_PosDupFlag<= (others=>'0');
                        reg_TradeDate<= (others=>'0');
                        reg_NoMDEntries<= (others=>'0');
                        reg_MDUpdateAction<= (others=>'0');
```

```
                reg_MDPriceLevel<= (others=>'0');
                reg_MDEntryType<= (others=>'0');
                reg_SecurityID<= (others=>'0');
                reg_RptSeq<= (others=>'0');
                reg_QuoteCondition<= (others=>'0');
                reg_MDEntryPx_base<= (others=>'0');
                reg_MDEntryPx_m<= (others=>'0');
                reg_NumberOfOrders<= (others=>'0');
                reg_MDEntryTime<= (others=>'0');
                reg_MDEntrySize<= (others=>'0');
                reg_TradingSessionID<= (others=>'0');
                reg_NetChgPrevDa_base<= (others=>'0');
                reg_NetChgPrevDa_m<= (others=>'0');
                reg_TickDirection<= (others=>'0');
                reg_OpenCloseSettleFlag<= (others=>'0');
                reg_SettlDate<= (others=>'0');
                end_of_command <= '0';

            elsif (valid_input = '1' ) then
               if (pr_state=START) then    count<=count + "001";
            end if;
               case pr_state is
-------------------------------------------------------------------------------------------------------------------------------------------------------
                when START =>               -- throw away first five bytes sequence number and channel number.
                        if (count = "100") then
                                pr_state <= Pmap;
                        else
                                pr_state <= START;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------

                when Pmap =>                                    -- find Pmap.
                        reg_Pmap <= flit_in;
                        if (flit_in(6) = '1' ) then                 -- has templateID;
                                pr_state <= templateID;
                        elsif(reg_templateID ="11110101" and reg_valid_ID='1') then
                              pr_state <=MsgSeqNum;
                           else
                          pr_state<=dummy;
                        end if;
                        count<=(others=>'0');
```

```
---------------------------------------------------------------------------------------------------------------------------------------------------------
                when templateID =>                                          -- find templateID
                    reg_templateID(7 downto 0) <= flit_in;                 -- save the templateID by clock rising edge
                    reg_templateID(15 downto 8) <= reg_templateID(7 downto 0);
                    if (flit_in(7) = '1') then
                      reg_valid_ID<='1';
                      if ( flit_in = "10000001" and reg_templateID(7 downto 0) = "00000001") then
                            pr_state <=MsgSeqNum;
                      else
                            pr_state<=dummy;
                      end if;
                    end if;
---------------------------------------------------------------------------------------------------------------------------------------------------------
              when dummy    =>
                  if ( end_of_packet='1' ) then pr_state<=START;
                  else pr_state<=dummy;
                        end if;



---------------------------------------------------------------------------------------------------------------------------------------------------------
            when MsgSeqNum=>
                  reg_MsgSeqNum( 7 downto 0) <= flit_in;
                  reg_MsgSeqNum(15 downto 8) <= reg_MsgSeqNum(7 downto 0);
                  if ( flit_in(7) = '1') then
                    pr_state <=SendingTime ;
                    else
                        pr_state <=MsgSeqNum;
                    end if;


---------------------------------------------------------------------------------------------------------------------------------------------------------
            when SendingTime=>
                  reg_SendingTime( 7 downto 0) <= flit_in;
                  reg_SendingTime(15 downto 8) <= reg_SendingTime(7 downto 0);
                  if ( flit_in(7) = '1') then
                        pr_state <=PosDupFlag ;
                  else
                        pr_state <=SendingTime;
                        end if;


---------------------------------------------------------------------------------------------------------------------------------------------------------
            when PosDupFlag=>
```

```
                reg_PosDupFlag( 7 downto 0) <= flit_in;
                reg_PosDupFlag(15 downto 8) <= reg_PosDupFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=TradeDate ;
                   else
                        pr_state <=PosDupFlag;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when TradeDate=>
                reg_TradeDate( 7 downto 0) <= flit_in;
                reg_TradeDate(15 downto 8) <= reg_TradeDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=NoMDEntries ;
                   else
                        pr_state <=TradeDate;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when NoMDEntries=>
                reg_NoMDEntries( 7 downto 0) <= flit_in;
                reg_NoMDEntries(15 downto 8) <= reg_NoMDEntries(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDUpdateAction ;
                   else
                        pr_state <=NoMDEntries;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDUpdateAction=>
                reg_MDUpdateAction( 7 downto 0) <= flit_in;
                reg_MDUpdateAction(15 downto 8) <= reg_MDUpdateAction(7 downto 0);
                if ( flit_in(7) = '1') then
                            pr_state <=MDPriceLevel ;
                else
                        pr_state <=MDUpdateAction;
                        end if;

                if(reg_NoMDEntries(7 downto 0)="10000001") then
                        end_of_command <= '1';
                end if;
```

```
-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDPriceLevel=>
                reg_MDPriceLevel( 7 downto 0) <= flit_in;
                reg_MDPriceLevel(15 downto 8) <= reg_MDPriceLevel(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryType ;
                   else
                        pr_state <=MDPriceLevel;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDEntryType=>
                reg_MDEntryType( 7 downto 0) <= flit_in;
                reg_MDEntryType(15 downto 8) <= reg_MDEntryType(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=SecurityID ;
                   else
                        pr_state <=MDEntryType;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when SecurityID=>
                reg_SecurityID( 7 downto 0) <= flit_in;
                reg_SecurityID(31 downto 8) <= reg_SecurityID(23 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=RptSeq ;
                   else
                        pr_state <=SecurityID;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when RptSeq=>
                reg_RptSeq( 7 downto 0) <= flit_in;
                reg_RptSeq(15 downto 8) <= reg_RptSeq(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <= QuoteCondition ;
                   else
                        pr_state <=RptSeq;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
when QuoteCondition=>
        reg_QuoteCondition( 7 downto 0) <= flit_in;
        reg_QuoteCondition(15 downto 8) <= reg_QuoteCondition(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDEntryPx_base ;
           else
                pr_state <=QuoteCondition;
           end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDEntryPx_base=>
        reg_MDEntryPx_base( 7 downto 0) <= flit_in;
        if ( flit_in(7) = '1') then
                              pr_state <=MDEntryPx_m ;
          else
                pr_state <=MDEntryPx_base;
           end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDEntryPx_m=>
        reg_MDEntryPx_m( 7 downto 0) <= flit_in;
        reg_MDEntryPx_m(15 downto 8) <= reg_MDEntryPx_m(7 downto 0);
        if ( flit_in(7) = '1') then
                          pr_state <=NumberOfOrders ;
          else
                pr_state <=MDEntryPx_m;
           end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when NumberOfOrders=>
        reg_NumberOfOrders( 7 downto 0) <= flit_in;
        reg_NumberOfOrders(15 downto 8) <= reg_NumberOfOrders(7 downto 0);
        if ( flit_in(7) = '1') then
          pr_state <=MDEntryTime ;
           else
                pr_state <=NumberOfOrders;
           end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDEntryTime=>
        reg_MDEntryTime( 7 downto 0) <= flit_in;
```

```
                reg_MDEntryTime(15 downto 8) <= reg_MDEntryTime(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=MDEntrySize;
                else
                        pr_state <=MDEntryTime;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when MDEntrySize=>
                reg_MDEntrySize( 7 downto 0) <= flit_in;
                reg_MDEntrySize(15 downto 8) <= reg_MDEntrySize(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=TradingSessionID;
                else
                        pr_state <=MDEntrySize;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TradingSessionID=>
                reg_TradingSessionID( 7 downto 0) <= flit_in;
                reg_TradingSessionID(15 downto 8) <= reg_TradingSessionID(7 downto 0);
                if ( flit_in(7) = '1') then
                        pr_state<=NetChgPrevDa_base;
                else
                        pr_state <=TradingSessionID;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when NetChgPrevDa_base=>
                reg_NetChgPrevDa_base( 7 downto 0) <= flit_in;
                if ( flit_in = "10000000") then
                                        pr_state<=TickDirection;
                    elsif(flit_in(7) = '1') then
                                        pr_state<=NetChgPrevDa_m;
                else
                        pr_state <=NetChgPrevDa_base;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when NetChgPrevDa_m=>
                reg_NetChgPrevDa_m( 7 downto 0) <= flit_in;
                reg_NetChgPrevDa_m( 15 downto 8) <= reg_NetChgPrevDa_m(7 downto 0);
```

```
                reg_NetChgPrevDa_m( 23 downto 16) <= reg_NetChgPrevDa_m(15 downto 8);
                if ( flit_in(7) = '1') then
                                pr_state<=TickDirection;
                else
                        pr_state <=NetChgPrevDa_m;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TickDirection=>
                reg_TickDirection( 7 downto 0) <= flit_in;
                reg_TickDirection(15 downto 8) <= reg_TickDirection(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=OpenCloseSettleFlag;
                else
                        pr_state <=TickDirection;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when OpenCloseSettleFlag=>
                reg_OpenCloseSettleFlag( 7 downto 0) <= flit_in;
                reg_OpenCloseSettleFlag(15 downto 8) <= reg_OpenCloseSettleFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=SettlDate;
                else
                        pr_state <=OpenCloseSettleFlag;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when SettlDate=>
                reg_SettlDate( 7 downto 0) <= flit_in;
                reg_SettlDate(15 downto 8) <= reg_SettlDate(7 downto 0);
                if ( flit_in(7) = '1') then
                 if(reg_NoMDEntries(7  downto  0)/="10000001")  then                                    --end  of
sequence
                                pr_state<=MDUpdateAction;
                                  reg_NoMDEntries<=reg_NoMDEntries - "00000001";
                        else                                                                            --end of packet
                                pr_state<=START;
                                  reg_Pmap<=(others=>'0');
                                  end_of_command <= '0';
                                  reg_NoMDEntries<=reg_NoMDEntries - "00000001";
```

```
                    end if;
                else
                    pr_state <=SettlDate;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
                end case;
            end if;
            end if;

        end process;


flit_out(7 downto 0)<= flit_in;
---pass the 8 bit flit directly without cycle delay
With pr_state select
flit_out(11)<= valid_input when NumberOfOrders ,
                valid_input when MDEntrySize,
                valid_input when MDEntryPx_m,
                valid_input when MDEntryPx_base,
                valid_input when SecurityID,
                valid_input when MDEntryType,
                valid_input when MDPriceLevel,
                valid_input when MDUpdateAction,
                 '0'            when others;

with pr_state select
flit_out(10 downto 8)<= "101" when MDEntrySize,
                        "000" when NumberOfOrders,
                "100" when MDEntryPx_base,
                "100" when MDEntryPx_m,
                "011" when SecurityID,
                        "010" when MDPriceLevel,
                    "001" when MDUpdateAction,
                        "110" when MDEntryType,
                "111" when others;
---add 4 bit identifier for every bit according to the FSM state


--template_ID<=reg_templateID;
```

```
with pr_state select
select_en<= '0' when START,
                    '0' when Pmap,
                    '0' when templateID,
                    '0' when dummy,
                    '1' when others;

process(pr_state, reg_NoMDEntries)
begin
if (( pr_state = SettlDate) and ( reg_NoMDEntries (7 downto 0) = "10000001" )) then
end_p_local <= '1' ;
else
end_p_local <= '0' ;
end if;
end process;

process(clk) begin
if (clk'event and clk='1') then
if (pr_state=MDPriceLevel and flit_in(7)='1' ) then flags<="100";
elsif (pr_state=NumberOfOrders and flit_in(7)='1') then flags<="010";
elsif (pr_state=MDEntrySize and flit_in(7)='1') then flags<="001";
else flags<="000";
end if;
end if;
end process;

end FSM;



----------------------------------------------------------------
decoder_131
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity decoder_131 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input : in std_logic;
```

```
--                template_ID : out std_logic_vector ( 15 downto 0);
--                valid_ID       : out std_logic;
                  flit_out                  : out std_logic_vector ( 11 downto 0);
                  end_of_packet      : in std_logic;
--                end_of_packet_local: out std_logic;
                  end_of_command  : out std_logic;
                  end_p_local : out std_logic;
                  select_en             : out std_logic;
                        flags                   : out std_logic_vector(2 downto 0));
end decoder_131;

architecture FSM of decoder_131 is
-------------------------------------------------------------------------------------------------------------------
type state is (
                    START,
                    Pmap,
                    templateID,
                    MsgSeqNum,
                    SendingTime,
                    PosDupFlag,
                    TradeDate,
                    NoMDEntries,
                    MDUpdateAction,
                    MDPriceLevel,
                    MDEntryType,
                    OpenCloseSettleFlag,
                    SettlDate,
                    SecurityID,
                    RptSeq,
                    MDEntryPx_base,
                MDEntryPx_m,
                    MDEntryTime,
                    MDEntrySize,
                    NumberOfOrders,
                    TradingSessionID,
                    NetChgPrevDa_base,
                    NetChgPrevDa_m,
                    TradeVolume,
                    TradeCondition,
                    TickDirection,
                    QuoteCondition,
```

```
                AggressorSide,
                MatchEventIndicator,
                dummy);                                         --all possible states--
-------------------------------------------------------------------------------------------------------------------------
signal pr_state, nx_state: state;
signal count : std_logic_vector ( 2 downto 0 ):= "000";
signal reg_templateID : std_logic_vector ( 15 downto 0):= (others => '0');
signal reg_Pmap :std_logic_vector (7 downto 0):= (others => '0');    ---control signals----
signal reg_valid_ID: std_logic;
signal state_no : std_logic_vector (3 downto 0);

-------------------------------------------------------------------------------------------------------------------------
signal reg_MsgSeqNum: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SendingTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_PosDupFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NoMDEntries: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDUpdateAction: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDPriceLevel: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryType: std_logic_vector (15 downto 0):=(others => '0');
signal reg_OpenCloseSettleFlag: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SettlDate: std_logic_vector (15 downto 0):=(others => '0');
signal reg_SecurityID: std_logic_vector (31 downto 0):=(others => '0');
signal reg_RptSeq: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntryPx_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_MDEntryPx_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_MDEntryTime: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MDEntrySize: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NumberOfOrders: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradingSessionID: std_logic_vector (15 downto 0):=(others => '0');
signal reg_NetChgPrevDa_base: std_logic_vector (7 downto 0):=(others => '0');
signal reg_NetChgPrevDa_m: std_logic_vector (23 downto 0):=(others => '0');
signal reg_TradeVolume: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TradeCondition: std_logic_vector (15 downto 0):=(others => '0');
signal reg_TickDirection: std_logic_vector (15 downto 0):=(others => '0');
signal reg_QuoteCondition: std_logic_vector (15 downto 0):=(others => '0');
signal reg_AggressorSide: std_logic_vector (15 downto 0):=(others => '0');
signal reg_MatchEventIndicator: std_logic_vector (15 downto 0):=(others => '0');

-------------------------------------------------------------------------------------------------------------------------
```

```
begin
process ( clk)
        begin
                if ( clk'event and clk = '1' ) then
                        if (rst = '1') then
                        pr_state <= START;
                        count <= "000";
                        reg_valid_ID <= '0';
                        reg_templateID <= (others => '0');
                        reg_MsgSeqNum<= (others=>'0');
                        reg_SendingTime<= (others=>'0');
                        reg_PosDupFlag<= (others=>'0');
                        reg_TradeDate<= (others=>'0');
                        reg_NoMDEntries<= (others=>'0');
                        reg_MDUpdateAction<= (others=>'0');
                        reg_MDPriceLevel<= (others=>'0');
                        reg_MDEntryType<= (others=>'0');
                        reg_OpenCloseSettleFlag<= (others=>'0');
                        reg_SettlDate<= (others=>'0');
                        reg_SecurityID<= (others=>'0');
                        reg_RptSeq<= (others=>'0');
                        reg_MDEntryPx_base<= (others=>'0');
                        reg_MDEntryPx_m<= (others=>'0');
                        reg_MDEntryTime<= (others=>'0');
                        reg_MDEntrySize<= (others=>'0');
                        reg_NumberOfOrders<= (others=>'0');
                        reg_TradingSessionID<= (others=>'0');
                        reg_NetChgPrevDa_base<= (others=>'0');
                        reg_NetChgPrevDa_m<= (others=>'0');
                        reg_TradeVolume<= (others=>'0');
                        reg_TradeCondition<= (others=>'0');
                        reg_TickDirection<= (others=>'0');
                        reg_QuoteCondition<= (others=>'0');
                        reg_AggressorSide<= (others=>'0');
                        reg_MatchEventIndicator<= (others=>'0');
                        end_of_command <= '0';

                elsif ( valid_input = '1' ) then
                   if (pr_state=START) then count<=count + "001"; end if;
                   case pr_state is
```

```
                when START =>                -- throw away first five bytes sequence number and channel number.
                        if (count = "100") then
                                pr_state <= Pmap;
                        else
                                pr_state <= START;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

                when Pmap =>                                    -- find Pmap.
                        reg_Pmap <= flit_in;
                        if (flit_in(6) = '1' ) then              -- has templateID;
                                pr_state <= templateID;
                        elsif(reg_templateID ="11110101" and reg_valid_ID='1') then
                                pr_state <=MsgSeqNum;
                           else
                       pr_state<=dummy;
                        end if;
                        count<=(others=>'0');
-------------------------------------------------------------------------------------------------------------------------------------------------------
                when templateID =>                                              -- find templateID
                      reg_templateID(7 downto 0) <= flit_in;                    -- save the templateID by clock rising edge
                      reg_templateID(15 downto 8) <= reg_templateID(7 downto 0);
                      if (flit_in(7) = '1') then
                        reg_valid_ID<='1';
                        if ( flit_in = "10000011" and reg_templateID(7 downto 0) = "00000001") then
                                pr_state <=MsgSeqNum;
                        else
                                pr_state<=dummy;
                        end if;
                      end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
              when dummy    =>
                    if ( end_of_packet='1' ) then pr_state<=START;
                    else pr_state<=dummy;
                            end if;



-------------------------------------------------------------------------------------------------------------------------------------------------------
             when MsgSeqNum=>
                    reg_MsgSeqNum( 7 downto 0) <= flit_in;
```

```
                reg_MsgSeqNum(15 downto 8) <= reg_MsgSeqNum(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=SendingTime ;
                   else
                        pr_state <=MsgSeqNum;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when SendingTime=>
                reg_SendingTime( 7 downto 0) <= flit_in;
                reg_SendingTime(15 downto 8) <= reg_SendingTime(7 downto 0);
                if ( flit_in(7) = '1') then
                        pr_state <=PosDupFlag ;
                else
                        pr_state <=SendingTime;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when PosDupFlag=>
                reg_PosDupFlag( 7 downto 0) <= flit_in;
                reg_PosDupFlag(15 downto 8) <= reg_PosDupFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=TradeDate ;
                   else
                        pr_state <=PosDupFlag;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when TradeDate=>
                reg_TradeDate( 7 downto 0) <= flit_in;
                reg_TradeDate(15 downto 8) <= reg_TradeDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <= NoMDEntries;
                   else
                        pr_state <=TradeDate;
                   end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------------

            when NoMDEntries=>
                reg_NoMDEntries( 7 downto 0) <= flit_in;
                reg_NoMDEntries(15 downto 8) <= reg_NoMDEntries(7 downto 0);
```

```
                if ( flit_in(7) = '1') then
                  pr_state <=MDUpdateAction ;
                   else
                        pr_state <=NoMDEntries;
                   end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDUpdateAction=>
                reg_MDUpdateAction( 7 downto 0) <= flit_in;
                reg_MDUpdateAction(15 downto 8) <= reg_MDUpdateAction(7 downto 0);
                if ( flit_in(7) = '1') then
                            pr_state <=MDPriceLevel ;
                else
                        pr_state <=MDUpdateAction;
                        end if;

                if(reg_NoMDEntries(7 downto 0)="10000001") then
                        end_of_command <= '1';
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDPriceLevel=>
                reg_MDPriceLevel( 7 downto 0) <= flit_in;
                reg_MDPriceLevel(15 downto 8) <= reg_MDPriceLevel(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryType ;
                   else
                        pr_state <=MDPriceLevel;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when MDEntryType=>
                reg_MDEntryType( 7 downto 0) <= flit_in;
                reg_MDEntryType(15 downto 8) <= reg_MDEntryType(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=OpenCloseSettleFlag;
                else
                        pr_state <=MDEntryType;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
            when OpenCloseSettleFlag=>
```

```
                reg_OpenCloseSettleFlag( 7 downto 0) <= flit_in;
                reg_OpenCloseSettleFlag(15 downto 8) <= reg_OpenCloseSettleFlag(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=SettlDate;
                else
                        pr_state <=OpenCloseSettleFlag;
                end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when SettlDate=>
                reg_SettlDate( 7 downto 0) <= flit_in;
                reg_SettlDate(15 downto 8) <= reg_SettlDate(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=SecurityID ;
                   else
                        pr_state <=SettlDate;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when SecurityID=>
                reg_SecurityID( 7 downto 0) <= flit_in;
                reg_SecurityID(31 downto 8) <= reg_SecurityID(23 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=RptSeq ;
                   else
                        pr_state <=SecurityID;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when RptSeq=>
                reg_RptSeq( 7 downto 0) <= flit_in;
                reg_RptSeq(15 downto 8) <= reg_RptSeq(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state <=MDEntryPx_base ;
                   else
                        pr_state <=RptSeq;
                   end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------

        when MDEntryPx_base=>
                reg_MDEntryPx_base( 7 downto 0) <= flit_in;
```

```
                if ( flit_in(7) = '1') then
                                pr_state <=MDEntryPx_m ;
                  else
                      pr_state <=MDEntryPx_base;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDEntryPx_m=>
                reg_MDEntryPx_m( 7 downto 0) <= flit_in;
                reg_MDEntryPx_m(15 downto 8) <= reg_MDEntryPx_m(7 downto 0);
                if ( flit_in(7) = '1') then
                                pr_state <=MDEntryTime ;
                  else
                      pr_state <=MDEntryPx_m;
                  end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDEntryTime=>
                reg_MDEntryTime( 7 downto 0) <= flit_in;
                reg_MDEntryTime(15 downto 8) <= reg_MDEntryTime(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=MDEntrySize;
                else
                      pr_state <=MDEntryTime;
                      end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------------------

            when MDEntrySize=>
                reg_MDEntrySize( 7 downto 0) <= flit_in;
                reg_MDEntrySize(15 downto 8) <= reg_MDEntrySize(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=NumberOfOrders;
                else
                      pr_state <=MDEntrySize;
                      end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------------------

            when NumberOfOrders=>
                reg_NumberOfOrders( 7 downto 0) <= flit_in;
                reg_NumberOfOrders(15 downto 8) <= reg_NumberOfOrders(7 downto 0);
                if ( flit_in(7) = '1') then
                  pr_state<=TradingSessionID;
```

```
                else
                        pr_state <=NumberOfOrders;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TradingSessionID=>
                reg_TradingSessionID( 7 downto 0) <= flit_in;
                reg_TradingSessionID(15 downto 8) <= reg_TradingSessionID(7 downto 0);
                if ( flit_in(7) = '1') then
                        pr_state<=NetChgPrevDa_base;
                else
                        pr_state <=TradingSessionID;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
        when NetChgPrevDa_base=>
                reg_NetChgPrevDa_base( 7 downto 0) <= flit_in;
                if ( flit_in = "10000000") then
                                        pr_state<=TradeVolume;
                        elsif(flit_in(7) = '1') then
                                        pr_state<=NetChgPrevDa_m;
                else
                        pr_state <=NetChgPrevDa_base;
                        end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when NetChgPrevDa_m=>
                reg_NetChgPrevDa_m( 7 downto 0) <= flit_in;
                reg_NetChgPrevDa_m( 15 downto 8) <= reg_NetChgPrevDa_m(7 downto 0);
                reg_NetChgPrevDa_m( 23 downto 16) <= reg_NetChgPrevDa_m(15 downto 8);
                if ( flit_in(7) = '1') then
                                        pr_state<=TradeVolume;
                else
                        pr_state <=NetChgPrevDa_m;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TradeVolume=>
                reg_TradeVolume( 7 downto 0) <= flit_in;
                reg_TradeVolume(15 downto 8) <= reg_TradeVolume(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=TradeCondition;
                else
                        pr_state <=TradeVolume;
```

```
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TradeCondition=>
                reg_TradeCondition( 7 downto 0) <= flit_in;
                reg_TradeCondition(15 downto 8) <= reg_TradeCondition(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=TickDirection;
                else
                        pr_state <=TradeCondition;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when TickDirection=>
                reg_TickDirection( 7 downto 0) <= flit_in;
                reg_TickDirection(15 downto 8) <= reg_TickDirection(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=QuoteCondition;
                else
                        pr_state <=TickDirection;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when QuoteCondition=>
                reg_QuoteCondition( 7 downto 0) <= flit_in;
                reg_QuoteCondition(15 downto 8) <= reg_QuoteCondition(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=AggressorSide;
                else
                        pr_state <=QuoteCondition;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when AggressorSide=>
                reg_AggressorSide( 7 downto 0) <= flit_in;
                reg_AggressorSide(15 downto 8) <= reg_AggressorSide(7 downto 0);
                if ( flit_in(7) = '1') then
                 pr_state<=MatchEventIndicator;
                else
                        pr_state <=AggressorSide;
                end if;
-------------------------------------------------------------------------------------------------------------------------------------------------------
        when MatchEventIndicator=>
                reg_MatchEventIndicator( 7 downto 0) <= flit_in;
                reg_MatchEventIndicator(15 downto 8) <= reg_MatchEventIndicator(7 downto 0);
```

```
                    if ( flit_in(7) = '1') then
                      if(reg_NoMDEntries(7  downto  0)/="10000001")  then                                    --end  of
sequence
                                pr_state<=MDUpdateAction;
                                  reg_NoMDEntries<=reg_NoMDEntries - "00000001";
                          else                                                                              --end of packet
                                pr_state<=START;
                                  reg_Pmap<=(others=>'0');
                                  end_of_command <= '0';

                                  reg_NoMDEntries<=reg_NoMDEntries - "00000001";
                          end if;
                        else
                            pr_state <=MatchEventIndicator;
                        end if;

-------------------------------------------------------------------------------------------------------------------------------------------------------
                    end case;

              end if;
                 end if;

        end process;


flit_out(7 downto 0)<= flit_in;
---pass the 8 bit flit directly without cycle delay
With pr_state select
flit_out(11)<= valid_input when NumberOfOrders ,
                    valid_input when MDEntrySize,
                    valid_input when MDEntryPx_m,
                    valid_input when MDEntryPx_base,
                    valid_input when SecurityID,
                    valid_input when MDEntryType,
                    valid_input when MDPriceLevel,
                    valid_input when MDUpdateAction,
                     '0'              when others;

with pr_state select
flit_out(10 downto 8)<= "101" when MDEntrySize,
                              "000" when NumberOfOrders,
```

```
                    "100" when MDEntryPx_base,
                    "100" when MDEntryPx_m,
                    "011" when SecurityID,
                              "010" when MDPriceLevel,
                         "001" when MDUpdateAction,
                              "110" when MDEntryType,
                    "111" when others;
---add 4 bit identifier for every bit according to the FSM state


with pr_state select
select_en<= '0' when START,
                '0' when Pmap,
                '0' when templateID,
                '0' when dummy,
                '1' when others;


with pr_state select
state_no<= "0001" when START,
                          "0010" when MsgSeqNum,
                          "0011" when NoMDEntries,
                          "0100" when MDUpdateAction,
                          "0000" when others;

------this is the flag generator
process(clk) begin
if (clk'event and clk='1') then
if (pr_state=MDPriceLevel and flit_in(7)='1' ) then flags<="100";
elsif (pr_state=NumberOfOrders and flit_in(7)='1') then flags<="010";
elsif (pr_state=MDEntrySize and flit_in(7)='1') then flags<="001";
else flags<="000";
end if;
end if;
end process;

process(pr_state, reg_NoMDEntries)
begin
if (( pr_state = MatchEventIndicator) and ( reg_NoMDEntries (7 downto 0) = "10000001" )) then
end_p_local <= '1' ;
else
```

```
end_p_local <= '0' ;
end if;
end process;

end FSM;


----------------------------------------------------------------
mux
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity mux is
port (
flit_out_117          : in std_logic_vector(11 downto 0);
flit_out_129          : in std_logic_vector(11 downto 0);
flit_out_125          : in std_logic_vector(11 downto 0);
flit_out_131          : in std_logic_vector(11 downto 0);
flit_out_122          : in std_logic_vector(11 downto 0);
select_en_117       : in std_logic;
select_en_129       : in std_logic;
select_en_125       : in std_logic;
select_en_131       : in std_logic;
select_en_122       : in std_logic;
flags_122     : in std_logic_vector(2 downto 0);
flags_125     : in std_logic_vector(2 downto 0);
flags_129     : in std_logic_vector(2 downto 0);
flags_131     : in std_logic_vector(2 downto 0);
end_of_command_117    : in std_logic;
end_of_command_122    : in std_logic;
end_of_command_125    : in std_logic;
end_of_command_129    : in std_logic;
end_of_command_131    : in std_logic;
flit_out                  : out std_logic_vector(11 downto 0);
end_of_command  : out std_logic;
flags             : out std_logic_vector(2 downto 0)
);
end entity;

architecture rtl of mux is
signal select5 : std_logic_vector(4 downto 0);
```

```
begin
select5<= select_en_117 & select_en_122 & select_en_125 & select_en_129 & select_en_131;

with select5 select
flit_out <= flit_out_117 when "10000" ,
                    flit_out_122 when "01000",
                    flit_out_125 when "00100",
                    flit_out_129 when "00010",
                    flit_out_131 when "00001",
                  (others=>'0') when others;

with select5 select
end_of_command <=      end_of_command_117    when "10000" ,
                        end_of_command_122    when "01000",
                        end_of_command_125    when "00100",
                        end_of_command_129    when "00010",
                        end_of_command_131    when "00001",
                        '0'                         when others;

with select5 select
flags <=       flags_122    when "01000",
                  flags_125    when "00100",
                  flags_129    when "00010",
                  flags_131    when "00001",
                  "000"          when others;

end architecture;



----------------------------------------------------------------
command buffer
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use work.type_pkg.all;

entity cmdbuffer is
        generic
        (
```

```
        CMD_WIDTH       : natural := 2;
        LEVEL_WIDTH     : natural := 4;
        NAME_WIDTH      : natural := 32;
        PRICE_WIDTH     : natural := 32;
        AMOUNT_WIDTH : natural := 16;
        SB_WIDTH   : natural := 8;
        DATA_WIDTH      : natural := 8;
        NOO_WIDTH       : natural := 16
        );

        port
        (
        clk             : in std_logic;
        rst             : in std_logic;
        input           : in std_logic_vector ( 11 downto 0);
        end_of_command  : in std_logic;
        flag_sub        : in std_logic_vector ( 2 downto 0); -- 2: level 1: noo 0: entrysize
        flag_template       : in std_logic;                             -- indicate template 122 and 125 which don't have entries of noo
and level.
        output              : out command_record;
        valid_b             : out std_logic
        );
end entity;

architecture rtl of cmdbuffer is
signal tmp_price:   std_logic_vector(PRICE_WIDTH downto 0):=(others => '0');
signal tmp_cmd:     std_logic_vector(CMD_WIDTH downto 0):=(others => '0');
signal tmp_level:   std_logic_vector(LEVEL_WIDTH downto 0):=(others => '0');
signal tmp_name:            std_logic_vector(NAME_WIDTH downto 0):=(others => '0');
signal tmp_amount:          std_logic_vector(AMOUNT_WIDTH downto 0):=(others => '0');
signal tmp_sb:              std_logic_vector(SB_WIDTH downto 0):=(others => '0');
signal tmp_noo :    std_logic_vector (NOO_WIDTH downto 0):=(others => '0');
signal visited :        std_logic_vector( 2 downto 0);
signal op :             std_logic_vector( 2 downto 0);
signal count:               natural range 0 to 10001;
signal debug:               std_logic;


begin
```

```
output.price(27 downto 0) <= tmp_price(PRICE_WIDTH - 5 downto 0);
output.price(31 downto 28) <="0000";
output.update_action <= tmp_cmd(CMD_WIDTH - 1 downto 0);
output.level <= tmp_level(LEVEL_WIDTH - 1 downto 0);
output.stock_id(27 downto 0) <= tmp_name(NAME_WIDTH - 5 downto 0);
output.stock_id(31 downto 28) <= "0000";
output.amount(13 downto 0) <= tmp_amount(AMOUNT_WIDTH - 3 downto 0);
output.amount(15 downto 14) <= "00";
output.entry_type <= tmp_sb( 7 downto 0);
output.NOO (13 downto 0) <= tmp_noo(NOO_WIDTH - 3 downto 0);
output.NOO (15 downto 14) <= "00";


process(clk)
begin

if ( rising_edge(clk)) then

if (rst = '1') then
                tmp_cmd(CMD_WIDTH) <= '0';
                tmp_level(LEVEL_WIDTH) <= '0';
                tmp_name(NAME_WIDTH) <= '0';
                tmp_price(PRICE_WIDTH) <= '0';
                tmp_amount(AMOUNT_WIDTH) <= '0';
                tmp_sb(SB_WIDTH) <= '0';
                tmp_noo(NOO_WIDTH) <= '0';
                valid_b <= '0';
                visited <= (others => '0');
                op <= (others => '0');
                count <= 0;
                debug <= '0';

else

        output.end_of_command <= end_of_command;

        if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_level(LEVEL_WIDTH)='1') and (tmp_name(NAME_WIDTH)='1')
                 and (tmp_price(PRICE_WIDTH)='1') and (tmp_amount(AMOUNT_WIDTH)='1')
                 and (tmp_sb(SB_WIDTH)='1') and (tmp_noo(NOO_WIDTH) = '1') ) then
                count <= count + 1;
                if (count = 857 ) then debug <= '1'; else debug <= '0'; end if;
```

```
                valid_b <= '0';
                tmp_cmd(CMD_WIDTH) <= '0';
                tmp_level(LEVEL_WIDTH) <= '0';
                tmp_name(NAME_WIDTH) <= '0';
                tmp_price(PRICE_WIDTH) <= '0';
                tmp_amount(AMOUNT_WIDTH) <= '0';
                tmp_sb(SB_WIDTH) <= '0';
                tmp_noo(NOO_WIDTH) <= '0';
--              tmp_cmd <= (others =>'0');
                tmp_level <= (others =>'0');
                tmp_name <= (others =>'0');
                tmp_price <= (others =>'0');
                tmp_amount <= (others =>'0');
                tmp_sb <= (others =>'0');
                tmp_noo <= (others =>'0');
        else
                valid_b <= '0';
        end if;

    if ( visited(2) = '1') then
        if (flag_sub(2) = '1' and op(2) = '0') then
                tmp_level(LEVEL_WIDTH-1 downto 0) <= tmp_level(LEVEL_WIDTH-1 downto 0) - '1';
        end if;
        tmp_level(LEVEL_WIDTH) <= '1';
        if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_name(NAME_WIDTH)='1') and (tmp_price(PRICE_WIDTH)='1')
              and (tmp_amount(AMOUNT_WIDTH)='1') and (tmp_sb(SB_WIDTH)='1')
              and (tmp_noo(NOO_WIDTH) = '1') ) then
                valid_b <= '1';
        end if;
        visited(2) <= '0';
    end if;

    if ( visited(1) = '1') then
        if ( flag_sub(1) = '1' and op(1) = '0') then
                tmp_noo(NOO_WIDTH - 3 downto 0) <= tmp_noo(NOO_WIDTH - 3 downto 0) - '1';
        end if;
        tmp_noo(NOO_WIDTH) <= '1';
        if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_name(NAME_WIDTH)='1') and (tmp_price(PRICE_WIDTH)='1')
              and (tmp_amount(AMOUNT_WIDTH)='1') and (tmp_sb(SB_WIDTH)='1')
              and (tmp_level(LEVEL_WIDTH) = '1') ) then
                valid_b <= '1';
```

```
            end if;
            visited(1) <= '0';
        end if;

        if ( visited(0) = '1') then
            if ( flag_sub(0) = '1' and op(0) = '0' ) then
                tmp_amount(AMOUNT_WIDTH - 3 downto 0) <= tmp_amount(AMOUNT_WIDTH - 3 downto 0) - '1';
            end if;
            tmp_amount(AMOUNT_WIDTH) <= '1';
            if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_name(NAME_WIDTH)='1') and (tmp_price(PRICE_WIDTH)='1')
                 and (tmp_noo(NOO_WIDTH)='1') and (tmp_sb(SB_WIDTH)='1')
                 and (tmp_level(LEVEL_WIDTH) = '1') ) then
                valid_b <= '1';
            end if;
            visited(0) <= '0';
        end if;

            if (input(DATA_WIDTH + 3) = '1' ) then

                case input(10 downto 8) is

                when "001" =>
                    tmp_cmd(CMD_WIDTH-1 downto 0) <= input(CMD_WIDTH-1 downto 0);
                    tmp_cmd(CMD_WIDTH) <= '1';

                when "011" =>
                    tmp_name(NAME_WIDTH - 5 downto DATA_WIDTH - 1) <= tmp_name(NAME_WIDTH- DATA_WIDTH
- 4 downto 0);
                    tmp_name(DATA_WIDTH - 2 downto 0) <= input(DATA_WIDTH - 2 downto 0);
                    if (input(DATA_WIDTH - 1) = '1') then
                    tmp_name(NAME_WIDTH) <= '1';
                        if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_level(LEVEL_WIDTH)='1') and
(tmp_price(PRICE_WIDTH)='1')
                            and (tmp_amount(AMOUNT_WIDTH)='1') and (tmp_sb(SB_WIDTH)='1')
                            and (tmp_noo(NOO_WIDTH) = '1') ) then
                        valid_b <= '1';
                        end if;
                    else
                    tmp_name(NAME_WIDTH) <= '0';
                    end if;
                when "100" =>
```

```
                tmp_price(PRICE_WIDTH - 5 downto DATA_WIDTH - 1) <= tmp_price(PRICE_WIDTH- DATA_WIDTH -
4 downto 0);
                tmp_price(DATA_WIDTH - 2 downto 0) <= input(DATA_WIDTH - 2 downto 0);
                if (input(DATA_WIDTH - 1) = '1') then
                tmp_price(PRICE_WIDTH) <= '1';
                    if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_level(LEVEL_WIDTH)='1') and
(tmp_name(NAME_WIDTH)='1')
                        and (tmp_amount(AMOUNT_WIDTH)='1') and (tmp_sb(SB_WIDTH)='1')
                        and (tmp_noo(NOO_WIDTH) = '1') ) then
                    valid_b <= '1';
                    end if;
                else
                tmp_price(PRICE_WIDTH) <= '0';
                end if;
            when "101" =>
                tmp_amount(AMOUNT_WIDTH - 3 downto DATA_WIDTH - 1) <= tmp_amount(AMOUNT_WIDTH -
DATA_WIDTH - 2 downto 0);
                tmp_amount(DATA_WIDTH - 2 downto 0) <= input (DATA_WIDTH - 2 downto 0);
                if (input(DATA_WIDTH - 1) = '1') then
                visited(0) <= '1';
                end if;

                if ( input(7 downto 0) = "10000000" ) then
                    op(0) <= '1';
                else
                    op(0) <= '0';
                end if;

            when "110" =>
                tmp_sb(SB_WIDTH-1 downto 0) <= input(SB_WIDTH-1 downto 0);
                tmp_sb(SB_WIDTH) <= '1';
                    if ((tmp_cmd(CMD_WIDTH)='1') and (tmp_level(LEVEL_WIDTH)='1') and
(tmp_name(NAME_WIDTH)='1')
                        and (tmp_price(PRICE_WIDTH)='1') and (tmp_amount(AMOUNT_WIDTH)='1')
                        and (tmp_noo(NOO_WIDTH) = '1') ) then
                    valid_b <= '1';
                    end if;
            when others =>

                debug <= debug;
            end case;
```

```
                if (flag_template = '0' ) then
            case input(10 downto 8) is


                when "010" =>
                    tmp_level(LEVEL_WIDTH-1 downto 0) <= input(LEVEL_WIDTH-1 downto 0);
                    visited(2) <= '1';
                    if ( input(7 downto 0) = "10000000" )    then
                        op(2) <= '1';
                    else
                        op(2) <= '0';
                    end if;

                when "000" =>
                    tmp_noo(NOO_WIDTH - 3 downto DATA_WIDTH - 1) <= tmp_noo(NOO_WIDTH - DATA_WIDTH - 2
downto 0);
                    tmp_noo(DATA_WIDTH - 2 downto 0) <= input (DATA_WIDTH - 2 downto 0);
                    if (input(DATA_WIDTH - 1) = '1') then
                    visited(1) <= '1';
                    end if;

                    if ( input(7 downto 0) = "10000000" ) then
                        op(1) <= '1';
                    else
                        op(1) <= '0';
                    end if;

                when others =>

                    debug <= debug;
                end case;
            else
                    tmp_level(LEVEL_WIDTH-1 downto 0) <= (others => '0');
                    visited(2) <= '1';
                    op(2) <= '0';
                    tmp_noo(NOO_WIDTH - 3 downto 0) <= (others => '0');
                    visited(1) <= '1';
                    op(1) <= '0';
            end if;
```

```
                end if;
        end if;
        end if;
        end process;
end rtl;

----------------------------------------------------------------
FIFO
----------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.type_pkg.all;

entity fifo is
GENERIC
(
ADDRESS_WIDTH          : integer:=2; -- 2 bit
DATA_WIDTH      : integer:=87 -- 87 bit
);

port ( clk     : in std_logic;
                rst     : in std_logic;
                enr     : in std_logic;                         --enable read,should be '0' when not in use.
                enw     : in std_logic;                         --enable write,should be '0' when not in use.
                dataout : out command_record;                           --output data
                valid   : out std_logic_vector(1 downto 0);    --valid bits for output
                datain          : in command_record                     --input data

        );
end fifo;

architecture Behavioral of fifo is

type memory_type is array (0 to ((2**ADDRESS_WIDTH)-1)) of command_record;
type memory_type2 is array (0 to ((2**ADDRESS_WIDTH)-1)) of std_logic_vector(1 downto 0);


-----distributed-------
signal memory : memory_type;                                            --memory for queue
```

```
signal valid_memory: memory_type2:=(others => (others => '0'));              --memory for valid bits
signal readptr,writeptr : std_logic_vector(ADDRESS_WIDTH-1 downto 0);          --read and write pointers.
signal full0 : std_logic;
signal empty0 : std_logic;
constant default_command : command_record := (                                --use to reset memory
        update_action                => (others => '0'),
        price                         => (others => '0'),
        amount                          => (others => '0'),
        level                         => (others => '0'),
        entry_type                    => (others => '0'),
        NOO                             => (others => '0'),
        stock_id                      => (others => '0'),
        end_of_command                => '0'
        );

begin

valid <= valid_memory (conv_integer(readptr));
dataout <= memory (conv_integer(readptr));

fifo0: process(clk)
begin
        if rising_edge(clk) then
        if rst='1' then
        readptr <= (others => '0');
        writeptr <= (others => '0');
        valid_memory <= (others => (others => '0'));
        memory <= (others => default_command);
        empty0 <='1';
        full0<='0';
        --err<='0';

        else

                if (writeptr + '1' = readptr) then                    --configure full empty indicator
                full0<='1';
                else
                full0<='0';
                end if ;

                if (readptr = writeptr ) then
```

```
                empty0<='1';
                else
                empty0<='0';
                end if ;

                --if (empty0='1' and enr='1') or (full0='1' and enw='1') then
                --err<='1';
                --end if ;

                if enw='1' and full0='0' then                    -- write data to fifo
                memory (conv_integer(writeptr)) <= datain ;
                valid_memory(conv_integer(writeptr)) <= "11";
                writeptr <= writeptr + '1' ;
                end if ;

                --if enr='0' and empty0='0' then
                --if enr='0' and (readptr /= writeptr) then
                --dataout <= memory (conv_integer(readptr));
                --if (empty0 = '0') then
                --valid_memory(conv_integer(readptr)) <= "00";
                if enr = '1' then                                -- read data from fifo
                valid_memory(conv_integer(readptr)) <= "00";
                readptr <= readptr + '1' ;
                end if ;

        end if;
        end if;
end process;
end Behavioral;

----------------------------------------------------------
book
----------------------------------------------------------
----Third edition---
----by Danqing Hua--
----4/15/2013-------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

```
use ieee.std_logic_unsigned.all;
use work.type_pkg.all;

entity book is
generic
( PRICE_WIDTH: natural:=32;                    --this 3 defines how wide and deep the book is
   AMOUNT_WIDTH: natural:=16;
   NOO_WIDTH: natural:=16;
   LEVEL_WIDTH: natural :=4;
   ID_WIDTH: natural:=8;
   Cell_WIDTH : natural:=64
);
port (
          clk              : in std_logic;
          reset            : in std_logic;
          exe               : in std_logic;
          fetch            : in std_logic;
          command          : in command_record;
          book_data_in: in book_record;
          book_data_out : out book_record
      );
end entity;

architecture rtl of book is
  type price_tp is array(1 to 10)of std_logic_vector (PRICE_WIDTH-1 downto 0) ;
  type amount_tp is array(1 to 10) of std_logic_vector (AMOUNT_WIDTH-1 downto 0) ;
  type NOO_tp is array(1 to 10) of std_logic_vector(NOO_WIDTH -1 downto 0);
  type valid_tp is array (1 to 10) of std_logic;
  signal price_sell: price_tp;
  signal price_buy: price_tp;
  signal amount_sell: amount_tp;
  signal amount_buy: amount_tp;
  signal valid_sell: valid_tp;
  signal valid_buy: valid_tp;
  signal NOO_buy : NOO_tp;
  signal NOO_sell:NOO_tp;
  signal flag : std_logic;
  begin
process(clk) begin
  if clk'event and clk='1' then
    if (reset='1') then                                    -------reset all mem----------
```

```
        price_sell(1 to 10)<=(others=>(others=>'0'));
        price_buy(1 to 10)<=(others=>(others=>'0'));
        amount_sell(1 to 10)<=(others=>(others=>'0'));
        amount_buy(1 to 10)<=(others=>(others=>'0'));
        valid_sell(1 to 10)<=(others=>'0');
        valid_buy(1 to 10)<=(others=>'0');
        NOO_sell(1 to 10)<=(others=>(others=>'0'));
        NOO_buy(1 to 10)<=(others=>(others=>'0'));
        flag<='0';
  elsif exe='1' then
        case command.update_action is
  -------------------------delete--------------------------------------------------------------
        when "10"=>
        if command.entry_type="10110000" then      ----sell----
sell_gen:      for    i in 1 to 9    loop
                        if(i >= conv_integer(command.level)) then
                          NOO_sell(i)<=NOO_sell(i+1);
                          price_sell(i)<=price_sell(i+1);
                          amount_sell(i)<=amount_sell(i+1);
                          valid_sell(i)<=valid_sell(i+1);
                          valid_sell(10)<='0';
                          end if;
                    end loop sell_gen;

       elsif command.entry_type="10110001" then                    ----buy--------
buy_gen:      for    i in 1 to 9    loop
                        if(i>= conv_integer(command.level)) then
                          NOO_buy(i)<=NOO_buy(i+1);
                          price_buy(i)<=price_buy(i+1);
                          amount_buy(i)<=amount_buy(i+1);
                          valid_buy(i)<=valid_buy(i+1);
                          valid_buy(10)<='0';
                        end if;
                    end loop buy_gen;
  end if;

-------------------------insert-------------------------------------------------------------------
    when "00"=>
      if command.entry_type="10110000" then      ----sell----
sell_gen_insert:      for    i in 1 to 9    loop
                        if(i >= conv_integer(command.level)) then
```

```
                        NOO_sell(i+1)<=NOO_sell(i);
                        price_sell(i+1)<=price_sell(i);
                        amount_sell(i+1)<=amount_sell(i);
                        valid_sell(i+1)<=valid_sell(i);
                        valid_sell(conv_integer(command.level))<='1';
                        price_sell(conv_integer(command.level))<=command.price;
                        amount_sell(conv_integer(command.level))<=command.amount;
                        NOO_sell(conv_integer(command.level))<=command.NOO;
                        end if;
                          end loop sell_gen_insert;

        elsif command.entry_type="10110001"        then          ----buy--------
buy_gen_insert:      for    i in 1 to 9    loop
                       if(i>= conv_integer(command.level)) then
                        NOO_buy(i+1)<=NOO_buy(i);
                        price_buy(i+1)<=price_buy(i);
                        amount_buy(i+1)<=amount_buy(i);
                        valid_buy(i+1)<=valid_buy(i);
                        valid_buy(conv_integer(command.level))<='1';
                        price_buy(conv_integer(command.level))<=command.price;
                        amount_buy(conv_integer(command.level))<=command.amount;
                        NOO_buy(conv_integer(command.level))<=command.NOO;
                        end if;
   end loop buy_gen_insert;
  end if;
--------------------------modify-----------------------------------------------------------------------
   when "01"=>
     flag<='1';

   if command.entry_type="10110000"    then --sell---
   price_sell(conv_integer(command.level))<=command.price;
   amount_sell(conv_integer(command.level))<=command.amount;
   NOO_sell(conv_integer(command.level))<=command.NOO;
   valid_sell(conv_integer(command.level))<='1';

   elsif command.entry_type="10110001"        then
   NOO_buy(conv_integer(command.level))<=command.NOO;
   price_buy(conv_integer(command.level))<=command.price;
   amount_buy(conv_integer(command.level))<=command.amount;
   valid_buy(conv_integer(command.level))<='1';
   end if;
```

```
  ----------others----------
  when others=>
    --do nothing----maybe we can add sth later-------
  ----------

 end case;
 elsif fetch='1' then
 data_in_gen_1:
 for i in 1 to 10 loop
  NOO_sell(i)<=book_data_in(i-1).NOO;
  price_sell(i)<= book_data_in(i-1).price;
  amount_sell(i)<=book_data_in(i-1).amount;
  valid_sell(i)<=book_data_in(i-1).valid;
  end loop;
 data_in_gen_2:
 for i in 1 to 10 loop
  NOO_buy(i)<=book_data_in(i+9).NOO;
  price_buy(i)<= book_data_in(i+9).price;
  amount_buy(i)<= book_data_in(i+9).amount;
  valid_buy(i)<= book_data_in(i+9).valid;
 end loop;
 end if;
 end if;
 end process;

data_out_gen_1:
   for i in 1 to 10 generate
 book_data_out(i-1).NOO<=NOO_sell(i);
 book_data_out(i-1).price<= price_sell(i);
 book_data_out(i-1).amount<=amount_sell(i);
 book_data_out(i-1).valid <=valid_sell(i);
 end generate;

 data_out_gen_2:
  for    i in 1 to 10 generate
 book_data_out(i+9).NOO<=NOO_buy(i);
 book_data_out(i+9).price<= price_buy(i);
 book_data_out(i+9).amount<=amount_buy(i);
 book_data_out(i+9).valid <=valid_buy(i);
 end generate;
```

```
 end rtl;



----------------------------------------------------------------
memory controller.vhdl
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
entity memory_controller is
generic(
                Cell_WIDTH : natural := 64;
                  Book_count : natural := 10;
                  Cell_count : natural :=20;
              ID_WIDTH: natural:=32
);

port (
      clk                         : in std_logic;
      reset                       : in std_logic;
      stock_id                    : in std_logic_vector(31 downto 0):=(others=>'0');
      command_status       : in std_logic_vector( 1 downto 0);
      ---00 for not valid
      ---01 for writing, stock_id not ready
      ---10 for writing, stock_ide ready
      ---11 for ready to be executed
      addr                        : out natural range 0 to (Book_count*Cell_count -1);
      we                            : out std_logic;
      re                            : out std_logic;
      exe                           : out std_logic;
      fetch_en                        : out std_logic;
      book_reset                : out std_logic);
end entity;

architecture rtl of memory_controller is
type state is (idle, fetch, execute, writeback);
signal fsm_state: state;
signal book_status:    std_logic_vector(1 downto 0);
```

```
---00 for empty
---01 for begin fetch
---10 for ready to execute
---11 for begin to write back
---flow : empty=>begin fetch => ready to execute => ready to write back(execute in this cycle) =>begin to write back => empty
begin
process(clk) begin
if clk'event and clk='1' then
if (reset='1') then
fsm_state<=idle;
else
case fsm_state is
--------------------------------------------------------------------------------
when idle=>                                          --start mode
--------------------------------------------------------------------------------
  if command_status="00" or command_status="01" then fsm_state<=idle; end if;
  if command_status="10" or command_status="11" then fsm_state<=fetch; end if;
--------------------------------------------------------------------------------
when fetch=>                                         --fetch mode
--------------------------------------------------------------------------------
  fsm_state<= execute;
--------------------------------------------------------------------------------
when execute=>                                        --execute mode
--------------------------------------------------------------------------------
if command_status="11" then fsm_state<=writeback;
else fsm_state<=execute;
end if;
--------------------------------------------------------------------------------
when writeback=>                                      --writeback mode
--------------------------------------------------------------------------------
  fsm_state<=idle;
--------------------------------------------------------------------------------
end case;
end if;
end if;
end process;

with fsm_state select
  book_status<= "00" when idle,
                "01" when fetch,
                "10" when execute,
```

```
                        "11" when writeback;

with fsm_state select
  we<= '0' when idle,
           '0' when fetch,
           '0' when execute,
           '1' when writeback;

with fsm_state select
re<= '0' when idle,
         '1' when fetch,
         '0' when execute,
         '0' when writeback;

with fsm_state select
exe<= '0' when idle,
           '0' when fetch,
           '1' when execute,
           '0' when writeback;

with    fsm_state select
fetch_en<= '0' when idle,
              '1' when fetch,
              '0' when execute,
              '0' when writeback;

with fsm_state select
book_reset <= '1' when idle,
                        reset when fetch,
                        reset when execute,
                        reset when writeback;

--the look up table for addr
with stock_id select
addr<= 0 when "00000000000000000010110100101001",    --11561
            1 when "00000000000000000100010100101000",    --17704
            2 when "00000000000000000110110111010000",    --28112
            3 when "00000000000001011101010011111110",    --382206
            4 when "00000000000000000000000101101110",    --366
            5 when "00000000000000000010000010011011", ---8347
            9 when others;
```

```
end rtl;


----------------------------------------------------------
RAM
----------------------------------------------------------

-- a book has 20 levels, a level has 64 bit, a book has 1280 bit
--assume that the RAM can at least handle 10 books-----that is 12800 bit--

library ieee;
use ieee.std_logic_1164.all;
use work.type_pkg.all;

entity RAM is

	generic
	(
		Book_count : natural := 10;
		Cell_count : natural :=20
	);

	port
	(
		clk		: in std_logic;
		reset	: in std_logic;
		addr	: in natural range 0 to Book_count-1;
		book_data_in: in book_record;
		we		: in std_logic := '0';
		re		: in std_logic :='0';
		book_data_out: out book_record
	);

end entity;

architecture rtl of RAM is
type mem is array (0 to Book_count -1) of book_record;
signal memory: mem;
begin
process(clk) begin
if clk'event and clk='1' then
```

```
------reset the rem---------------------------------
if reset='1' then
Reset_gen:
            for i in 0 to (Book_count -1) loop
            for j in 0 to (Cell_count -1) loop
            memory(i)(j).valid<='0';
            memory(i)(j).price<=(others=>'0');
            memory(i)(j).NOO<=(others=>'0');
            memory(i)(j).amount<=(others=>'0');
      end loop;
      end loop;
------write data-----------------------------------
  elsif    we='1' then
            memory(addr)<=book_data_in;
------read data-----------------------------------
  elsif re='1' then
read_valid_clear:
        for i in 0 to Cell_count -1 loop
          memory(addr)(i).valid<='0';
        end loop;
---care: data is always "on line", but only cleared when "re=1"---
        end if;
  end if;
end process;

book_data_out<= memory(addr);
end rtl;

-------------------------------------------------------------
output.vhdl
-------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.type_pkg.all;

entity output_look_up is
port (
clk: in std_logic;
rst: in std_logic;
```

```
command: in command_record;
exe        : in std_logic;
addr      : in     natural range 0 to 9;
data_out: out std_logic_vector (63 downto 0 );
data_valid: out std_logic;
startofpacket_o            : out std_logic;
endofpacket_o                : out std_logic;
book_data                       : in     book_record;
we                                  : in     std_logic
);
end entity;

architecture rtl of output_look_up is
type book_look_up is array(9 downto 0) of std_logic_vector (9 downto 0);
type book_change_table is array(9 downto 0) of std_logic;
signal look_up_bid: book_look_up;
signal look_up_ask: book_look_up;
signal book_changed: book_change_table;
signal look_up_bid_o: book_look_up;
signal look_up_ask_o: book_look_up;
signal book_changed_o: book_change_table;
signal select_out: std_logic_vector(9 downto 0);
signal select_out_natural: natural range 0 to 9;
signal reg_end_of_command: std_logic;
type state is (idle, start, header1,header2,header3,header4,header5,header6, playload,book);
signal fsm_state: state;
signal data_payload : std_logic_vector (63 downto 0 );
signal data_valid_payload : std_logic;
signal select_temp : std_logic_vector(9 downto 0);
signal i: natural range 0 to 20;
signal book_o : std_logic_vector(63 downto 0);
signal book_reg: book_record;
signal flag_end: std_logic;
begin
-------------------------the look up table-----------------------------------------------------------
process(clk, rst)
begin
if (clk'event and clk='1') then
     if(rst='1' OR reg_end_of_command = '1') then
        for i in 0 to 9 loop
        look_up_bid(i)<=(others=>'0');
```

```
        look_up_ask(i)<=(others=>'0');
        book_changed(i)<='0';
        reg_end_of_command<='0';
        end loop;


         for j in 0 to (19) loop
         book_reg(j).valid<='0';
         book_reg(j).price<=(others=>'0');
         book_reg(j).NOO<=(others=>'0');
         book_reg(j).amount<=(others=>'0');
        end loop;

        if (we='1') then book_reg<=book_data; end if;
else


        if (exe='1') then
             reg_end_of_command<=command.end_of_command;

           if (command.update_action = "10" or command.update_action ="00") then        --both insert and delete
              book_changed(addr)<='1';
              if command.entry_type="10110000" then                                      --bid
              for i in 0 to 9 loop
                 if ( i>= conv_integer(command.level)-1) then look_up_bid(addr)(i)<='1'; end if;
              end loop;
              elsif command.entry_type="10110001" then                                  --ask
              for i in 0 to 9 loop
                 if (i>= conv_integer(command.level) -1) then look_up_ask(addr)(i)<='1'; end if;
              end loop;
              end if;
           elsif command.update_action = "01" then
                 book_changed(addr)<='1';
                 if command.entry_type="10110000" then look_up_bid(addr)(conv_integer(command.level)-1)<='1';
                 elsif command.entry_type="10110001" then look_up_bid(addr)(conv_integer(command.level)-1)<='1'; end if;
           end if;
         end if;
  end if;


end if;
```

```
end process;



--------------------the ouput buffer----an exact copy of the look up table----------------------------------------
process(clk,rst)
begin
if (clk='1' and clk'event) then
    if (rst='1' ) then
            for i in 0 to 9 loop
            look_up_bid_o(i) <=(others=>'0');
            look_up_ask_o(i) <=(others=>'0');
            book_changed_o(i)<='0';
            end loop;
    else
       if (reg_end_of_command='1') then
            look_up_bid_o<= look_up_bid;
            look_up_ask_o<=look_up_ask;
            book_changed_o<=book_changed;

       end if;

     for i in 0 to 9 loop
    if select_out(i)='1' then book_changed_o(i)<='0'; end if;
    end loop;
    end if;
end if;
end process;

----------------------this is the priority decoder---------------------------------------
----------------------its function is to select output for each instrument cycle by cyle--
with book_changed_o select
select_temp<= "1000000000" when "1---------",
                        "0100000000" when "01--------",
                        "0010000000" when "001-------",
                        "0001000000" when "0001------",
                        "0000100000" when "00001-----",
                        "0000010000" when "000001----",
                        "0000001000" when "0000001---",
                        "0000000100" when "00000001--",
                        "0000000010" when "000000001-",
                        "0000000001" when "0000000001",
```

```
                    "0000000000" when others;



with fsm_state select
select_out <= select_temp when playload,
                     "0000000000" when others;


----------this is the mux which pack up the output-----------------------------
with select_out select                                  --first decode the select into addr
select_out_natural<= 9 when "1000000000",
                                 8 when "0100000000",
                                 7 when "0010000000",
                                 6 when "0001000000",
                                 5 when "0000100000",
                                 4 when "0000010000",
                                 3 when "0000001000",
                                 2 when "0000000100",
                                 1 when "0000000010",
                                 0 when "0000000001",
                                 9 when others;       ---must have a default, 9 is never used

with select_out_natural select
data_valid_payload<=          '1' when 0,
                              '1' when 1,
                              '1' when 2,
                              '1' when 3,
                              '1' when 4,
                              '1' when 5,
                              '1' when 6,
                              '0' when others;

with select_out_natural select
data_payload(63 downto 32)<=   "00000000000000000010110100101001" when 0,   --11561
                                  "00000000000000000100010100101000" when 1,   --17704
                                  "00000000000000000110110111010000" when 2,   --28112
                                  "00000000000001011101010011111110" when 3,   --382206
                                  "00000000000000000000000101101110" when 4,   --366
                                  "00000000000000000010000010011011" when 5, ---8347
                                  (others=>'0')                              when others;
```

```
data_payload(31 downto 22)<= look_up_bid_0(select_out_natural);
data_payload(21 downto 16)<="000000";
data_payload(15 downto 6)<= look_up_ask_0(select_out_natural);
data_payload(5 downto 0)<="000000";

with fsm_state select
data_out <= x"01005e505001000f" when header1,
                        x"1f7b1b6708004500" when header2,
                        x"004a000040001011" when header3,
                        x"f1a0e0001a017f00" when header4,
                        x"00010400271100B6" when header5,
                        x"0000000000000000" when header6,
                        data_payload          when playload,
                                book_o                  when book,
                        (others => '0')       when others;

with fsm_state select
data_valid <= '1' when header1,
                        '1' when header2,
                        '1' when header3,
                        '1' when header4,
                        '1' when header5,
                        '1' when header6,
                        '1' when playload,
                                        '1' when book,
                        '0' when others;



------this is the output fsm----------------------------------------
process(rst,clk)
begin
if (clk'event and clk='1') then
      if(rst='1') then
            fsm_state<=idle;
            i<=0;
      else
           case fsm_state is
--------------------------------------------------------------------------
                when idle=>
```

```
                        i<=0;
                        if (command.end_of_command='1') then
                    fsm_state<=start;
                          else fsm_state<=idle;
                        end if;
-------------------------------------------------------------------------
                    when start=>
                          i<=0;
                        if(exe='1') then
                        fsm_state<=header1;
                        end if;
-------------------------------------------------------------------------
                    when header1=>
                          fsm_state<=header2;
-------------------------------------------------------------------------
                    when header2=>
                          fsm_state<=header3;
-------------------------------------------------------------------------
                    when header3=>
                          fsm_state<=header4;
-------------------------------------------------------------------------
                    when header4=>
                          fsm_state<=header5;
-------------------------------------------------------------------------
                    when header5=>
                          fsm_state<=header6;
-------------------------------------------------------------------------
                    when header6=>
                          fsm_state<=playload;
-------------------------------------------------------------------------
                    when playload=>
                        fsm_state<=book;

-------------------------------------------------------------------------
                    when book=>
                       if (i<19) then i<=i+1; end if;
                       if( i=19) then fsm_state<=start; end if;
-----------------------------------------------------------------------------------
--                    when endop=>
--                            fsm_state<=idle;
                    end case;
```

```
        end if;
end if;
end process;

with (fsm_state = book and i = 19) select
flag_end <= '1' when true,
            '0' when others;


with flag_end select
startofpacket_o <='0' when '1',
                   '1' when others;



with flag_end select
endofpacket_o<='1' when '1',
                  '0' when others;


book_o<= book_reg(i).price & book_reg(i).amount & book_reg(i).NOO;


end rtl;

-------------------------------------------------------------
book_top.vhdl
-------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use work.type_pkg.all;
entity book_top is
generic(
   PRICE_WIDTH: natural:=32;
   AMOUNT_WIDTH: natural:=16;
   LEVEL_WIDTH: natural :=4;
   ID_WIDTH: natural:=32;
   Cell_WIDTH : natural := 64;
   Book_count : natural := 10;
   Cell_count : natural :=20
```

```
);

port(
        clk                             : in std_logic;
        reset                           : in std_logic;
        command                         : in command_record;
        command_status                  : in std_logic_vector(1 downto 0);
        exe                             : out std_logic;
        data_o                          : out std_logic_vector(63 downto 0);
        valid_o                         : out std_logic;
        startofpacket_o                 : out std_logic;
        endofpacket_o                   : out std_logic);

end entity;

architecture rtl of book_top is

--------------------------------use these 2 types-------------------------------------------
signal book_data: book_record;

component memory_controller
port (
clk: in std_logic;
reset: in std_logic;
stock_id: in std_logic_vector(ID_WIDTH-1 downto 0) :=(others=>'0');
command_status: in std_logic_vector( 1 downto 0);
addr   : out natural range 0 to (Book_count*Cell_count -1);
we             : out std_logic;
re             : out std_logic;
exe          : out std_logic;
fetch_en       : out std_logic;
book_reset : out std_logic);
end component;
-------------------------------------------------------------------------------------------
component RAM is
        port
        (
                clk             : in std_logic;
                reset       : in std_logic;
                addr    : in natural range 0 to Book_count-1;
                book_data_in: in book_record;
```

```
                we              : in std_logic ;
                re          : in std_logic ;
                book_data_out: out book_record
        );
end component;
-------------------------------------------------------------------------------
component book is
port (
        clk             : in std_logic;
        reset           : in std_logic;
        fetch           : in std_logic;
        exe              : in std_logic;
        command         : in command_record;
        book_data_in: in book_record;
        book_data_out: out book_record);
end component;
-------------------------------------------------------------------------------
component output_look_up is
  port (
        clk: in std_logic;
        rst: in std_logic;
        command: in command_record;
        exe     : in std_logic;
        addr    : in    natural range 0 to 9;
        data_out: out std_logic_vector (63 downto 0 );
        data_valid: out std_logic;
        startofpacket_o         : out std_logic;
        endofpacket_o           : out std_logic;
        book_data               : in book_record;
        we                      : in std_logic);
end component;
-------------------------------------------------------------------------------
signal book_data_b2r: book_record;
signal book_data_r2b: book_record ;
signal exe_temp : std_logic:='0';
signal fetch    : std_logic:='0';
signal book_reset: std_logic:='0';
signal we, re   : std_logic;
signal addr     : natural range 0 to book_count -1;
begin
book_in_use: book port map(clk=>clk,
```

```
                                    reset=>book_reset,
                                    fetch=>fetch,
                                    exe=>exe_temp,
                                    command=>command,
                                    book_data_in=>book_data_r2b,
                                    book_data_out=>book_data_b2r);

memory_controller_in_use: memory_controller
                            port map (clk=>clk,
                                    reset=>reset,
                                    stock_id=>command.stock_id,
                                    command_status=>command_status,
                                    addr=>addr,
                                    we=>we,
                                    re=>re,
                                    exe=>exe_temp,
                                    fetch_en=>fetch,
                                    book_reset=>book_reset);

RAM_in_use : RAM            port map(clk=>clk,
                                    reset=>reset,
                                        addr=>addr,
                                        book_data_in=>book_data_b2r,
                                        we=>we,
                                        re=>re,
                                      book_data_out=>book_data_r2b);


look_up_in_use: output_look_up    port map(
                                    clk=>clk,
                                    rst=>reset,
                                    command=>command,
                                    exe=>exe_temp,
                                    addr=>addr,
                                    data_out=>data_o,
                                    data_valid=>valid_o,
                                    startofpacket_o=> startofpacket_o,
                                  endofpacket_o=>endofpacket_o,
                                    book_data=>book_data_b2r,
                                    we=> we);
```

```
exe<=exe_temp;



end rtl;




----------------------------------------------------------------
top_5_templates.vhdl
----------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use work.type_pkg.all;

entity top_5_templates is
port(                   clk             : in    std_logic;
                        rst             : in    std_logic;
                                rst_fsm : in std_logic;
                        flit_in         : in    std_logic_vector(7    downto 0);
                        flit_valid      : in    std_logic;
                        data_o      :           out std_logic_vector(63 downto 0);
                                valid_o    : out std_logic;
                                startofpacket_o: out std_logic;
                                endofpacket_o    : out std_logic);
end entity;

architecture rtl of top_5_templates is

component decoder_117 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input  : in std_logic;
--              template_ID         : out std_logic_vector ( 7 downto 0);
--              valid_ID        : out std_logic;
                flit_out        : out std_logic_vector ( 11 downto 0);
                end_of_packet       : in std_logic;
                end_of_command  : out std_logic;
```

```
                end_p_local : out std_logic;
                select_en           : out std_logic);
end component;

component decoder_122 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input  : in std_logic;
--              template_ID         : out std_logic_vector ( 7 downto 0);
--              valid_ID        : out std_logic;
                flit_out        : out std_logic_vector ( 11 downto 0);
                end_of_packet       : in std_logic;
                end_p_local         : out std_logic;
                end_of_command  : out std_logic;
                select_en           : out std_logic;
                flag            : out std_logic;                        -- indicates whether template 122 or 125
                flags                   : out std_logic_vector(2 downto 0));    -- indicates optional specified fields
end component;

component decoder_125 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input : in std_logic;
--              template_ID : out std_logic_vector ( 7 downto 0);
--              valid_ID        : out std_logic;
                flit_out                : out std_logic_vector ( 11 downto 0);
                end_of_packet       : in std_logic;
                end_p_local : out std_logic;
                end_of_command  : out std_logic;
                select_en           : out std_logic;
                flag            : out std_logic;                        -- indicates whether template 122 or 125
                flags                   : out std_logic_vector(2 downto 0));    -- indicates optional specified fields
end component;

component decoder_129 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input : in std_logic;
--              template_ID : out std_logic_vector ( 7 downto 0);
--              valid_ID        : out std_logic;
                flit_out                : out std_logic_vector ( 11 downto 0);
```

```
                end_of_packet      : in std_logic;
                end_p_local : out std_logic;
                end_of_command  : out std_logic;
                select_en          : out std_logic;
                        flags              : out std_logic_vector(2 downto 0));
end component;

component decoder_131 is
port(           flit_in         : in std_logic_vector ( 7 downto 0 );
                clk, rst        : in std_logic;
                valid_input : in std_logic;
--              template_ID : out std_logic_vector ( 15 downto 0);
--              valid_ID       : out std_logic;
                flit_out            : out std_logic_vector ( 11 downto 0);
                end_of_packet      : in std_logic;
                end_of_command  : out std_logic;
                end_p_local  : out std_logic;
                select_en          : out std_logic;
                        flags              : out std_logic_vector(2 downto 0));
end component;

component cmdbuffer is
        port
        (
          clk                           : in std_logic;
          rst                           : in std_logic;
          input                         : in std_logic_vector ( 11 downto 0);
          end_of_command                : in std_logic;
          flag_sub              : in std_logic_vector ( 2 downto 0); -- 2: level 1: noo 0: entrysize
          flag_template             : in std_logic;
          output                        : out command_record;
          valid_b               : out std_logic
        );
end component;

component fifo is
port ( clk      : in std_logic;
                rst     : in std_logic;
                enr     : in std_logic;                         --enable read,should be '0' when not in use.
                enw     : in std_logic;                         --enable write,should be '0' when not in use.
                dataout : out command_record;                           --output data
```

```
        valid  : out std_logic_vector(1 downto 0);
        datain      : in command_record                    --input data
    );
end component;

component book_top is
port(
    clk                     : in std_logic;
    reset                   : in std_logic;
    command                 : in command_record;
    command_status          : in std_logic_vector(1 downto 0);
    exe                     : out std_logic;
    data_o                  : out std_logic_vector(63 downto 0);
    valid_o                 : out std_logic;
    startofpacket_o         : out std_logic;
    endofpacket_o           : out std_logic
);
end component;

component mux is
port (
flit_out_117        : in std_logic_vector(11 downto 0);
flit_out_129        : in std_logic_vector(11 downto 0);
flit_out_125        : in std_logic_vector(11 downto 0);
flit_out_131        : in std_logic_vector(11 downto 0);
flit_out_122        : in std_logic_vector(11 downto 0);
select_en_117       : in std_logic;
select_en_129       : in std_logic;
select_en_125       : in std_logic;
select_en_131       : in std_logic;
select_en_122       : in std_logic;
flags_122    : in std_logic_vector(2 downto 0);
flags_125    : in std_logic_vector(2 downto 0);
flags_129    : in std_logic_vector(2 downto 0);
flags_131    : in std_logic_vector(2 downto 0);
end_of_command_117    : in std_logic;
end_of_command_122    : in std_logic;
end_of_command_125    : in std_logic;
end_of_command_129    : in std_logic;
end_of_command_131    : in std_logic;
flit_out            : out std_logic_vector(11 downto 0);
```

```
end_of_command  : out std_logic;
flags           : out std_logic_vector(2 downto 0)
);
end component;


--signal end_of_packet                 :std_logic;
--signal template_ID                       :std_logic_vector (7 downto 0 ) := (others => '0');
--signal valid_ID                          :std_logic;
signal flit_out                     :   std_logic_vector ( 11 downto 0);
signal select_en                    :   std_logic;
signal end_of_command               :   std_logic;
signal flit_out_117                 :   std_logic_vector(11 downto 0);
signal flit_out_129                 :   std_logic_vector(11 downto 0);
signal flit_out_125                 :   std_logic_vector(11 downto 0);
signal flit_out_131                 :   std_logic_vector(11 downto 0);
signal flit_out_122                 :   std_logic_vector(11 downto 0);
signal select_en_117                :   std_logic;
signal select_en_129                :   std_logic;
signal select_en_125                :   std_logic;
signal select_en_131                :   std_logic;
signal select_en_122                :   std_logic;
signal end_of_command_117               :   std_logic;
signal end_of_command_129               :   std_logic;
signal end_of_command_125               :   std_logic;
signal end_of_command_131               :   std_logic;
signal end_of_command_122               :   std_logic;
signal flags_122            :   std_logic_vector(2 downto 0);
signal flags_125            :   std_logic_vector(2 downto 0);
signal flags_129            :   std_logic_vector(2 downto 0);
signal flags_131            :   std_logic_vector(2 downto 0);
signal data                             : std_logic_vector (11 downto 0 ) := (others => '0');
signal data2                            : command_record;
signal enw                              : std_logic;
signal enr                              : std_logic;
signal final_output                 : command_record;
signal valid_b                      :   std_logic_vector( 1 downto 0);
signal flags                        :   std_logic_vector(2 downto 0);

signal end_P                        : std_logic;
signal end_p_local_117              : std_logic;
```

```
signal end_p_local_122                  : std_logic;
signal end_p_local_125                  : std_logic;
signal end_p_local_129                  : std_logic;
signal end_p_local_131                  : std_logic;
signal flag_template                    : std_logic;
signal flag_template_122    : std_logic;
signal flag_template_125    : std_logic;


begin
flag_template<=flag_template_122 and flag_template_125;

end_P <= end_p_local_117 or end_p_local_122 or end_p_local_125 or end_p_local_129 or   end_p_local_131;


decoder_117_in_use: decoder_117 port map (
        clk => clk,
        rst => rst_fsm,
        flit_in => flit_in,
        valid_input => flit_valid,
--      template_ID => template_ID,
--      valid_ID => valid_ID,
        flit_out => flit_out_117,
        end_of_packet => end_P,
        end_p_local => end_p_local_117,
        end_of_command => end_of_command_117,
        select_en => select_en_117
);

decoder_122_in_use: decoder_122 port map (
        clk => clk,
        rst => rst_fsm,
        flit_in => flit_in,
        valid_input => flit_valid,
--      template_ID => template_ID,
--      valid_ID => valid_ID,
        flit_out => flit_out_122,
        end_of_packet => end_P,
        end_p_local => end_p_local_122,
        end_of_command => end_of_command_122,
        select_en => select_en_122,
```

```
        flag => flag_template_122,
        flags => flags_122
);

decoder_125_in_use: decoder_125 port map (
        clk => clk,
        rst => rst_fsm,
        flit_in => flit_in,
        valid_input => flit_valid,
--      template_ID => template_ID,
--      valid_ID => valid_ID,
        flit_out => flit_out_125,
        end_of_packet => end_P,
        end_p_local => end_p_local_125,
        end_of_command => end_of_command_125,
        select_en => select_en_125,
        flag => flag_template_125,
        flags => flags_125
);

decoder_129_in_use: decoder_129 port map (
        clk => clk,
        rst => rst_fsm,
        flit_in => flit_in,
        valid_input => flit_valid,
--      template_ID => template_ID,
--      valid_ID => valid_ID,
        flit_out => flit_out_129,
        end_of_packet => end_P,
        end_p_local => end_p_local_129,
        end_of_command => end_of_command_129,
        select_en => select_en_129,
        flags => flags_129
);

decoder_131_in_use: decoder_131 port map (
        clk => clk,
        rst => rst_fsm,
        flit_in => flit_in,
        valid_input => flit_valid,
--      template_ID => template_ID,
```

```
--      valid_ID => valid_ID,
        flit_out => flit_out_131,
        end_of_packet => end_P,
        end_p_local => end_p_local_131,
        end_of_command => end_of_command_131,
        select_en => select_en_131,
        flags => flags_131
);

mux_in_use: mux port map(
flit_out_117 => flit_out_117,
flit_out_122 => flit_out_122,
flit_out_125 => flit_out_125,
flit_out_129 => flit_out_129,
flit_out_131 => flit_out_131,
select_en_117 => select_en_117,
select_en_122 => select_en_122,
select_en_125 => select_en_125,
select_en_129 => select_en_129,
select_en_131 => select_en_131,
end_of_command_117 => end_of_command_117,
end_of_command_122 => end_of_command_122,
end_of_command_125 => end_of_command_125,
end_of_command_129 => end_of_command_129,
end_of_command_131 => end_of_command_131,
flags_122 =>flags_122,
flags_125 =>flags_125,
flags_129 =>flags_129,
flags_131 =>flags_131,
end_of_command => end_of_command,
flit_out        => data,
flags => flags
);

cmdbuffer_in_use : cmdbuffer port map(
        clk => clk,
        rst => rst,
        input => data,
        end_of_command => end_of_command,
        output => data2,
        valid_b => enw,
```

```
        flag_sub => flags,
        flag_template => flag_template
);

fifo_in_use: fifo port map(
        clk => clk,
        rst => rst,
        enr => enr,
        enw => enw,
        datain => data2,
        valid => valid_b,
        dataout => final_output
);

book_top_in_use :book_top      port map(
         clk=> clk,
         reset=>rst,
         command=>final_output,
         command_status=>valid_b,
         exe=> enr,
         data_o=>data_o,
         valid_o=>valid_o,
         startofpacket_o=>startofpacket_o,
         endofpacket_o=>endofpacket_o);


end rtl;
```

---

top_fixfast.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_textio.all;
library std;
```

```
use std.standard.string;
use std.textio.all;

entity fixfast
is port(
        clk             : in std_logic; -- clk input
        rst             : in std_logic;
------------------------------------------------------------interface in
        startofpacket_i: in std_logic;
        endofpacket_i: in std_logic;
        empty_i: in std_logic_vector(2 downto 0);
        ready_i : out std_logic;
        error_i: in std_logic;
      data_i : in std_logic_vector(63 downto 0);
        valid_i : in std_logic;
------------------------------------------------------------interface out
        startofpacket_o: out std_logic;
        endofpacket_o: out std_logic;
        empty_o: out std_logic_vector(2 downto 0);
        ready_o : in std_logic;
        error_o: out std_logic;
      data_o : out std_logic_vector(63 downto 0);
        valid_o : out std_logic);
end entity;

architecture rtl of fixfast is
signal rst_l: std_logic;
signal rst_fsm : std_logic;
component top_5_templates is
port(                   clk             : in    std_logic;
                        rst             : in    std_logic;
                                rst_fsm : in std_logic;
                        flit_in         : in    std_logic_vector(7    downto 0);
                        flit_valid      : in    std_logic;
                        data_o      : out std_logic_vector(63 downto 0);
                                valid_o     : out std_logic;
                                startofpacket_o: out std_logic;
                                endofpacket_o     : out std_logic);
end component;

component top_header is
```

```
port(
                clk             : in std_logic;
                rst             : in std_logic;

              startofpacket_i: in std_logic;
              endofpacket_i: in std_logic;
                data_i : in std_logic_vector(63 downto 0);
              valid_i : in std_logic;

                flit            : out std_logic_vector(7 downto 0);
                flit_valid      : out std_logic);
end component;

signal flit             : std_logic_vector(7    downto 0);
signal flit_valid       : std_logic;



begin
rst_l<= not rst;
rst_fsm <= (not rst) or startofpacket_i ;

top_5_templates_in_use : top_5_templates port map
(       clk => clk,
        rst_fsm => rst_fsm,
        rst => rst_l,
        flit_in => flit,
        flit_valid => flit_valid,
      data_o=>data_o,
        valid_o=>valid_o,
        startofpacket_o=>startofpacket_o,
        endofpacket_o=>endofpacket_o);

top_header_in_use : top_header port map
(       clk => clk,
        rst => rst_l,
        flit => flit,
        flit_valid => flit_valid,
        startofpacket_i=>startofpacket_i,
        endofpacket_i=>endofpacket_i,
        data_i=>data_i,
        valid_i=>valid_i);
```

```
        error_o<='0';
        empty_o<="000";

            ready_i<='1';
end rtl;



------------------------------------------------------------
type.vhdl
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
package type_pkg is
-----------------------------define command type---------------------------------------
type command_record is

        record
        update_action           :   std_logic_vector(1 downto 0);
        price                   :   std_logic_vector(31 downto 0);
        amount                  :   std_logic_vector(15 downto 0);
        level                   :   std_logic_vector(3 downto 0);
        entry_type              :   std_logic_vector( 7 downto 0);
        NOO                     :   std_logic_vector(15 downto 0);
        stock_id                :   std_logic_vector(31 downto 0);
        end_of_command      :   std_logic;
  end record;
-----------------------------define the cell & book type--------------------------------
type cell_record is
        record
        price                   :   std_logic_vector(31 downto 0);
        amount                  :   std_logic_vector(15 downto 0);
        NOO                     :   std_logic_vector(15 downto 0);
        valid                   :   std_logic;
end record;
type book_record    is array (19 downto 0) of cell_record;
-------------------------------------------------------------------------------------------
end type_pkg;
```

6.2 Software Part

/* Software Verification Decoder

decoder.c

*/

/* Source Code */

```
/*
 * File: decode.c
 * Authors:
 *          Chang Liu(cl3078@columbia.edu),
 *          Raghavan Santhanam(rs3294@columbia.edu)
 */

#include <stdio.h>
#include <stdlib.h>
#include <pcap.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

#include <net/ethernet.h>
#include <netinet/ip.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <arpa/inet.h>

// #define __DEBUG__

#define TOUCH()     do {\
                    printf("%s()\n", __func__);\
      } while ( 0 )


//#define printf(...) do {\
```

```
//              printf("%d: ", __LINE__);\
//              printf(__VA_ARGS__);\
//   } while ( 0 )

#ifdef __DEBUG__
#define DEBUG(...) printf(__VA_ARGS__)
#else
#define DEBUG(...)
#endif

enum MaxTemplates { EMaxTemplateIds = 16384 }; // Maximum value of
                                // 14-bit unsigned
                                // integer.

static unsigned long long int template_ids_presence[ EMaxTemplateIds ];

static void (*template_decoder[ EMaxTemplateIds ])(unsigned char *,
                                    unsigned char *,
                                    unsigned short int);
enum num_of_calc_funcs { ENum_of_calc_functions };
unsigned long long int (*calc_func[ ENum_of_calc_functions ])
                            (unsigned char *,
                                unsigned char *);

void packetHandler(u_char *userData, const struct pcap_pkthdr* pkthdr,
                                const u_char* packet);
static void init_template_decoders_hash_table(void);
static void init_field_calculator_hash_table(void);

int main(int argc, const char *argv[]) {
    pcap_t *descr = NULL;
    char errbuf[ PCAP_ERRBUF_SIZE ] = "";

    if ( argc != 2 )
    {
        fprintf(stderr, "Usage: PcapDecoder PcapFile\n");
        exit(-1);
    }

    init_template_decoders_hash_table();
    init_field_calculator_hash_table();
```

```
// open capture file for offline processing
descr = pcap_open_offline(argv[ 1 ], errbuf);
if ( !descr )
{
	fprintf(stderr, "pcap_open_live() failed: %s\n",
					errbuf);
	exit(1);
}

printf("MsgSeqNum,SendingTime,MDUpdateAction,MDPriceLevel,"
	"MDEntryType,SecurityID,RptSeq,MDEntryPx,MDEntryTime,"
	"MDEntrySize,QuoteCondition,NumberOfOrders,TradingSessionID,"
	"TradeDate,TickDirection,OpenCloseSettleFlag,TradeVolume,"
	"SettlDate,TradeCondition,AggressorSide,"
	"MatchEventIndicator,\n");

// start packet processing loop, just like live capture
if ( pcap_loop(descr, 0, packetHandler, NULL) < 0 )
{
	fprintf(stderr, "pcap_loop() failed: %s\n",
				pcap_geterr(descr));
	exit(1);
}

printf("\n\nHistogram of templates: \n\n");
int i = 0;
unsigned short int max_occur_templ_id = 0xffff;
unsigned long long int max_occur_times = 0;
while ( i < EMaxTemplateIds )
{
	if ( !template_ids_presence[ i ] )
	{
		i++;
		continue;
	}

	if ( max_occur_times < template_ids_presence[ i ] )
	{
		max_occur_times = template_ids_presence[ i ];
		max_occur_templ_id = i;
```

```
            }

            printf("%d : ", i);
            int j = 0;
            while ( j < template_ids_presence[ i ] )
            {
                if ( !(j % 100) )
                {
                    putchar('+');
                }
                j++;
            }
            printf("%llu\n", template_ids_presence[ i ]);
            i++;
        }

        printf("\n\nMaximum occuring template id : %u for %llu times\n",
                                max_occur_templ_id,
                                    max_occur_times);


        return 0;
}


static unsigned char * get_field(unsigned char *cur_ptr,
                                unsigned short int *len_left)
{
        unsigned char *tmp = cur_ptr;
        int l = 1;
        while (!(*tmp & 0x80) && l <= *len_left)
        {
            tmp++;
            l++;
        }
        *len_left -= l;

#ifdef __DEBUG__
            unsigned char *begin_ptr = tmp;
            unsigned int i = 1;
            while (begin_ptr > cur_ptr)
```

```
        {
            printf("[%u]:%02x ", i++, *begin_ptr--);
        }
        printf("[%u]:%02x\n", i, *begin_ptr);
#endif

    return tmp;
}

static unsigned int calc_uint32(unsigned char *stop_ptr,
                                unsigned char *cur_ptr)
{
    // Loop not used for efficiency!
    union h
    {
        // It's known that bit-fields are not
        // portable by default. And no attempt
        // has been made to make it portable
        // either as readability is being
        // preferred over portability here.
        //
        // Assumed: sizeof(unsigned int) = 4
        // as on Ubuntu 64-bit version.
        struct s
        {   // This is more intuitive and simple
            // Just consider always the 7-bits
            // of a byte!
            unsigned int b1 : 7;
            unsigned int b2 : 7;
            unsigned int b3 : 7;
            unsigned int b4 : 7;
        } f;
        unsigned int i;
    } u;

    u.i = 0;
    u.f.b1 = *cur_ptr--; // The first one has to be anyway read
                         // regardless whether is its stop-bit
                         // is set or not.
    if ( cur_ptr > stop_ptr )
    {
```

```
            u.f.b2 = *cur_ptr--;
        }
        if ( cur_ptr > stop_ptr )
        {
            u.f.b3 = *cur_ptr--;
        }
        if ( cur_ptr > stop_ptr )
        {
            u.f.b4 = *cur_ptr;
        }

        return u.i;
}

static unsigned long int calc_uint64(unsigned char *stop_ptr,
                                     unsigned char *cur_ptr)
{
        // Loop not used for efficiency!
        union h
        {
            // It's known that bit-fields are not
            // portable by default. And no attempt
            // has been made to make it portable
            // either as readability is being
            // preferred over portability here.
            //
            // Assumed: sizeof(unsigned long int) = 8
            // as on Ubuntu 64-bit version.
            struct s
            {   // This is more intuitive and simple
                // Just consider always the 7-bits
                // of a byte!
                unsigned long int b1 : 7;
                unsigned long int b2 : 7;
                unsigned long int b3 : 7;
                unsigned long int b4 : 7;
                unsigned long int b5 : 7;
                unsigned long int b6 : 7;
                unsigned long int b7 : 7;
                unsigned long int b8 : 7;
            } f;
```

```
        unsigned long int i;
    } u;

    u.i = 0;
    u.f.b1 = *cur_ptr--;
    if ( cur_ptr > stop_ptr )
    {
        u.f.b2 = *cur_ptr--;
    }
    if ( cur_ptr > stop_ptr )
    {
        u.f.b3 = *cur_ptr--;
    }
    if ( cur_ptr > stop_ptr )
    {
        u.f.b4 = *cur_ptr--;
    }
    if ( cur_ptr > stop_ptr )
    {
        u.f.b5 = *cur_ptr--;
    }
    if ( cur_ptr > stop_ptr )
    {
        u.f.b6 = *cur_ptr--;
    }
    if ( cur_ptr > stop_ptr )
    {
        u.f.b7 = *cur_ptr--;
    }
    if ( cur_ptr > stop_ptr )
    {
        u.f.b8 = *cur_ptr;
    }

    return u.i;
}

static unsigned short int calc_uint16(unsigned char *stop_ptr,
                                      unsigned char *cur_ptr)
{
    union h
```

```
    {
        // It's known that bit-fields are not
        // portable by default. And no attempt
        // has been made to make it portable
        // either as readability is being
        // preferred over portability here.
        //
        // Assumed: sizeof(unsigned short int) = 2
        // as on Ubuntu 64-bit version.
        struct s
        {
            unsigned short int b1: 7;
            unsigned short int b2: 7;
        } f;
        unsigned short int s; // At least 16-bits wide.
    } u;
    u.s = 0;

    u.f.b1 = *cur_ptr--;
    if ( cur_ptr > stop_ptr )
    {
        u.f.b2 = *cur_ptr;
    }

    return u.s;
}

enum decode_width { EUI32, EUI64, EUI16, EString };

static unsigned long long int decode_field(unsigned char **s,
                                           unsigned char **c,
                                           unsigned short int *l,
                                           enum decode_width width)
{
    *s = *c;
    ++*c;
    *c = get_field(*c, l);

    return calc_func[ width ](*s, *c);
}
```

```
enum string_type { EAscii, EUnicode, };
// Caller should free the returned string.
char *extract_string(unsigned char **stop_ptr, unsigned char **cur_ptr,
                                enum string_type t)
{
      (void) t; // Currently the type of the string is assumed to
                // be ASCII.

      char *s = malloc(*cur_ptr - *stop_ptr + 1);
      char *b = s;
      if ( s ) {
            memset(s, '\0', *cur_ptr - *stop_ptr + 1);
            while ( *cur_ptr >= *stop_ptr ) {
                  *s++ = *cur_ptr[ 0 ]-- & 0x7f;
            }
      }

      return b;
}

// Function-call simplifier macro.
#define decode_field(width) decode_field(&stop_ptr, &cur_ptr, \
                                &len_left, width)

struct template_117
{
      unsigned int msg_seq_num;
      unsigned long int sending_time;
      unsigned char pos_dup_flag;
      unsigned int trade_date;
      unsigned int sequence_length;
      unsigned int md_update_action;
      unsigned int md_price_level;
      unsigned int md_entry_type;
      unsigned int md_entry_time;
      unsigned int security_id;
      unsigned int rpt_seq;
      long int md_entry_px;
      int md_entry_size;
      unsigned int number_of_orders;
      unsigned char trading_session_id;
```

```
};


static void template117_decoder(unsigned char *stop_ptr,
                        unsigned char *cur_ptr,
                        unsigned short int len_left)
{
      struct template_117 t = { 0 };

      // Note that the below decode_field macro call expand
      // to make the actual function call to decode_field()
      // with the necessary arguments -- Have defined this
      // as a macro for simpler usage.

      t.msg_seq_num = decode_field(EUI32);
      printf("%u,", t.msg_seq_num);

      t.sending_time = decode_field(EUI64);
      printf("%lu,", t.sending_time);

      cur_ptr++; // Optional byte -- PosDupFlag

      t.trade_date = decode_field(EUI32);

      t.sequence_length = *cur_ptr++;

      // Decode Group Fields
      t.md_update_action = decode_field(EUI32);
      printf("%u,", t.md_update_action);

      t.md_price_level = decode_field(EUI32);
      printf("%u,", t.md_price_level);

      t.md_entry_type = decode_field(EUI32);
      printf("%c,", t.md_entry_type);

      t.md_entry_time = decode_field(EUI32);

      t.security_id = decode_field(EUI32);
      printf("%u,", t.security_id);
```

```
        t.rpt_seq = decode_field(EUI32);
        printf("%u,", t.rpt_seq);

        int exp = decode_field(EUI32);
        long int mantissa = decode_field(EUI64);
        t.md_entry_px = mantissa * pow(10, exp);
        printf("%lu,", t.md_entry_px);

        printf("%u,", t.md_entry_time);

        t.md_entry_size = decode_field(EUI32);
        printf("%u,", t.md_entry_size);

        printf("\\N,"); // QuoteCondition absent in 117

        t.number_of_orders = decode_field(EUI32);
        printf("%u,", t.number_of_orders);

        t.trading_session_id = decode_field(EUI32);
        printf("%c,", t.trading_session_id);

        printf("%u,", t.trade_date);

        printf("\\N,"); // TickDirection absent in 117
        printf("\\N,"); // openCloseSettleFlag absent in 117
        printf("\\N,"); // TradeVolume absent in 117
        printf("\\N,"); // SettlDate absent in 117
        printf("\\N,"); // TradeCondition absent in 117
        printf("\\N,"); // AggressorSide absent in 117
        printf("\\N"); // MatchEventIndicator absent in 117

        printf("\n");
}

static void loop(unsigned char *cur_ptr, int off, int bytes, bool step)
{
        putchar('\n');
        cur_ptr += off;
        int i = off;
        while ( bytes --> 0 ) {
                printf("%p\t[%d]:%#x %d\n", (void *)cur_ptr + i, i,
```

```
                        cur_ptr[ i ], cur_ptr[ i ] & 0x7f);
            i++;
        }
}

struct template_131
{
        unsigned int msg_seq_num;
        unsigned long int sending_time;
        unsigned char pos_dup_flag;
        unsigned int trade_date;
        unsigned int sequence_length;
        unsigned int md_update_action;
        unsigned int md_price_level;
        unsigned char md_entry_type;
        unsigned int open_close_settle_flag;
        unsigned int settle_date;
        unsigned int security_id_source;
        unsigned int security_id;
        unsigned int rpt_seq;
        long int md_entry_px;
        unsigned int md_entry_time;
        int md_entry_size;
        unsigned int number_of_orders;
        unsigned char trading_session_id;
        long int net_chg_prev_day;
        unsigned int trade_volume;
        unsigned char trade_condition;
        char *tick_direction;
        char *quote_condition;
        unsigned int aggressor_side;
        char *match_event_indicator;
};

static void template131_decoder(unsigned char *stop_ptr,
                        unsigned char *cur_ptr,
                        unsigned short int len_left)
{

        struct template_131 t = { 0 };
```

```
t.msg_seq_num = decode_field(EUI32);

t.sending_time = decode_field(EUI64);

cur_ptr++;

t.trade_date = decode_field(EUI32);

t.sequence_length = (*cur_ptr++ & 0x7f) - 13;

bool first_sequence = true;

while ( t.sequence_length --> 0 ) {

bool open_close_settle_flag_present = false;
bool settle_date_present = false;
bool number_of_orders_present = false;
bool trading_session_id_present = false;
bool trade_volume_present = false;
bool trade_condition_present = false;
bool tick_direction_present = false;

printf("%u,", t.msg_seq_num);
printf("%lu,", t.sending_time);

t.md_update_action = decode_field(EUI32);
printf("%u,", t.md_update_action);

if ( *(cur_ptr + 1) != 0x80 ) {
      t.md_price_level = decode_field(EUI32) - 1;
      printf("%u,", t.md_price_level);
} else {
      printf("\\N,");
      cur_ptr++;
}

t.md_entry_type = decode_field(EUI32);
printf("%c,", t.md_entry_type);

if ( *(cur_ptr + 1) != 0x80 ) {
      open_close_settle_flag_present = true;
```

```
        t.open_close_settle_flag = decode_field(EUI32) - 1;
} else {
        cur_ptr++;
}

if ( *cur_ptr != 0x80 ) {
        settle_date_present = true;
        t.settle_date = decode_field(EUI32) - 1;
} else {
        cur_ptr++;
}

t.security_id_source = 8; // Constant - Not present in packet.

t.security_id = decode_field(EUI32);
printf("%u,", t.security_id);

t.rpt_seq = decode_field(EUI32);
printf("%u,", t.rpt_seq);

int exp = decode_field(EUI32);
long int mantissa = decode_field(EUI64);
t.md_entry_px = mantissa * pow(10, exp);
printf("%lu,", t.md_entry_px);

t.md_entry_time = decode_field(EUI32);
printf("%u,", t.md_entry_time);

if ( *++cur_ptr != 0x80 ) {
        cur_ptr--;
        t.md_entry_size = decode_field(EUI32) - 1;
        printf("%d,", t.md_entry_size);
} else {
        printf("\\N,");
        cur_ptr++;
}

if ( *(cur_ptr++ + 1) != 0x80 ) {
        cur_ptr--;
        number_of_orders_present = true;
        t.number_of_orders = decode_field(EUI32) - 1;
```

```
} else {
	cur_ptr++;
}

if ( *cur_ptr++ != 0x80 ) {
	trading_session_id_present = true;
	t.trading_session_id = *cur_ptr & 0x7f;
} else {
	cur_ptr++;
}

if ( *++cur_ptr != 0x80 ) {
	trade_volume_present = true;
	t.trade_volume = decode_field(EUI32) - 1;
} else {
	cur_ptr++;
}

if ( *++cur_ptr != 0x80 ) {
	trade_condition_present = true;
	t.trade_condition = decode_field(EUI32) - 1;
} else {
	cur_ptr++;
}

if ( *cur_ptr != 0x80 ) {
	tick_direction_present = true;
	t.tick_direction =
		extract_string(&stop_ptr, &cur_ptr, EAscii);
} else {
	cur_ptr++;
}

cur_ptr -= 2;
if ( *cur_ptr != 0x80 ) {
	t.quote_condition =
		extract_string(&stop_ptr, &cur_ptr, EAscii);
	if ( t.quote_condition && t.quote_condition[ 0 ] ) {
		printf("%s,", t.quote_condition);
	} else {
		printf("\\N,");
```

```
        }

        free(t.quote_condition);
        t.quote_condition = NULL;
    } else {
        cur_ptr++;
        printf("\\N,");
    }

    if ( number_of_orders_present ) {
        printf("%u,", t.number_of_orders);
    } else {
        printf("\\N,");
    }

    if ( trading_session_id_present ) {
        printf("%c,", t.trading_session_id);
    } else {
        printf("\\N,");
    }

    printf("%u,", t.trade_date);

    if ( tick_direction_present ) {
        printf("%s,", t.tick_direction);
        free(t.tick_direction);
        t.tick_direction = NULL;
    } else {
        printf("\\N,");
    }

    if ( open_close_settle_flag_present ) {
        printf("%u,", t.open_close_settle_flag);
    } else {
        printf("\\N,");
    }

    if ( trade_volume_present ) {
        printf("%d,", t.trade_volume);
    } else {
        printf("\\N,");
```

```
}

if ( settle_date_present ) {
	printf("%u,", t.settle_date);
} else {
	printf("\\N,");
}

if ( trade_condition_present ) {
	printf("%c,", t.trade_condition);
} else {
	printf("\\N,");
}

cur_ptr++;
if ( *++cur_ptr != 0x80 ) {
	cur_ptr = stop_ptr - 1;
	cur_ptr += 7;
	t.aggressor_side = decode_field(EUI32) - 1;
	printf("%u,", t.aggressor_side);
} else {
	// cur_ptr = stop_ptr - 1;
	cur_ptr++;
	printf("\\N,");
}

stop_ptr = cur_ptr + 1;
if ( *cur_ptr++ != 0x80 ) {
	t.match_event_indicator =
		extract_string(&stop_ptr, &cur_ptr, EAscii);
	if ( t.match_event_indicator &&
			t.match_event_indicator[ 0 ] ) {
		printf("%s", t.match_event_indicator);
	} else {
		printf("\\N", t.match_event_indicator);
	}
} else {
	cur_ptr++;
	printf("\\N");
}
```

```
        cur_ptr++;

        printf("\n");
        }
}

struct template_122
{
        unsigned int msg_seq_num;
        unsigned long int sending_time;
        unsigned char pos_dup_flag;
        unsigned int trade_date;
        unsigned char sequence_length;
        unsigned int md_update_action;
        unsigned char md_entry_type;
        unsigned int security_id_source;
        unsigned int security_id;
        unsigned int rpt_seq;
        long int md_entry_px;
        int md_entry_size;
        long int net_chg_prev_day;
        unsigned int trade_volume;
        char *tick_direction;
        unsigned char trade_condition;
        unsigned int md_entry_time;
        unsigned int aggressor_side;
        char *match_event_indicator;
};

static void template122_decoder(unsigned char *stop_ptr,
                        unsigned char *cur_ptr,
                        unsigned short int len_left)
{
        struct template_122 t = { 0 };

        t.msg_seq_num = decode_field(EUI32);

        t.sending_time = decode_field(EUI64);

        cur_ptr++;
```

```
t.trade_date = decode_field(EUI32);

t.sequence_length = (*cur_ptr++ & 0x7f) + 4; // Add 4, dunno why!?

while ( t.sequence_length --> 0 ) { // Yeah, when it tends to 0!

	bool trade_volume_present = false;
	bool trade_condition_present = false;
	bool md_entry_size_present = false;
	bool tick_direction_present = false;

	printf("%u,", t.msg_seq_num);
	printf("%lu,", t.sending_time);

	t.md_update_action = decode_field(EUI32);
	printf("%u,", t.md_update_action);

	printf("\\N,"); // md_price_level absent in 121

	t.md_entry_type = decode_field(EUI32);
	printf("%c,", t.md_entry_type);

	t.security_id_source = 8; // Constant - Not present in
				  // packet.

	t.security_id = decode_field(EUI32);
	printf("%u,", t.security_id);

	t.rpt_seq = decode_field(EUI32);
	printf("%u,", t.rpt_seq);

	int exp = decode_field(EUI32);
	long int mantissa = decode_field(EUI64);
	t.md_entry_px = mantissa * pow(10, exp);
	printf("%lu,", t.md_entry_px);

	if ( *cur_ptr++ != 0x80 ) {
		md_entry_size_present = true;
		cur_ptr--;
		t.md_entry_size = decode_field(EUI32) - 1;
	} else {
```

```
        cur_ptr++;
}

(void)decode_field(EUI32); // Ignore 4 bytes!
(void)decode_field(EUI32); // Again!

if ( *cur_ptr != 0x80 ) {
        trade_volume_present = true;
        t.trade_volume = decode_field(EUI32) - 1;
} else {
        cur_ptr++;
}

stop_ptr = cur_ptr + 1;
if ( *cur_ptr++ != 0x80 ) {
        t.tick_direction =
                extract_string(&stop_ptr, &cur_ptr,
                                        EAscii);
if ( t.tick_direction[ 0 ] ) {
        tick_direction_present = true;
}
} else {
        cur_ptr++;
}

cur_ptr++;
if ( *++cur_ptr != 0x80 ) {
        trade_condition_present = true;
        t.trade_condition = decode_field(EUI32) - 1;
} else {
        cur_ptr++;
}

cur_ptr--;
t.md_entry_time = decode_field(EUI32);
printf("%u,", t.md_entry_time);

if ( md_entry_size_present ) {
        printf("%d,", t.md_entry_size);
} else {
        printf("\\N,");
```

```
}

printf("\\N,"); // QuoteCondition absent in 121
printf("\\N,"); // NumberOfOrders absent in 121
printf("\\N,"); // TradingSessionId absent in 121


printf("%u,", t.trade_date);

if ( tick_direction_present ) {
      printf("%s,", t.tick_direction[ 0 ] ?
                  t.tick_direction : "\\N");
      free(t.tick_direction);
      t.tick_direction = NULL;
} else {
      printf("\\N,");
}

printf("\\N,"); // OpenCloseSettleFlag absent in 121

if ( trade_volume_present ) {
      printf("%u,", t.trade_volume);
} else {
      printf("\\N,\n");
}

printf("\\N,"); // SettlDate absent in 121

if ( trade_condition_present ) {
      printf("%c,", t.trade_condition);
} else {
      printf("\\N,");
}

if ( *cur_ptr != 0x80 ) {
      t.aggressor_side = decode_field(EUI32) - 1;
      printf("%u,", t.aggressor_side);
} else {
      cur_ptr++;
      printf("\\N,");
}
```

```
            stop_ptr = cur_ptr + 1;
            if ( *cur_ptr++ != 0x80 ) {
                t.match_event_indicator =
                    extract_string(&stop_ptr, &cur_ptr,
                                        EAscii);
                printf("%s", t.match_event_indicator[ 0 ] ?
                        t.match_event_indicator : "\\N");
                free(t.match_event_indicator);
                t.match_event_indicator = NULL;
            } else {
                cur_ptr++;
                printf("\\N");
            }
            cur_ptr++;

            printf("\n");
        }

}

struct template_125
{
    unsigned int msg_seq_num;
    unsigned long int sending_time;
    unsigned char pos_dup_flag;
    unsigned int trade_date;
    unsigned char sequence_length;
    unsigned int md_update_action;
    unsigned char md_entry_type;
    unsigned int security_id_source;
    unsigned int security_id;
    unsigned int rpt_seq;
    long int md_entry_px;
    int md_entry_size;
    unsigned int md_entry_time;
};

static void template125_decoder(unsigned char *stop_ptr,
                    unsigned char *cur_ptr,
                    unsigned short int len_left)
```

```
{
    struct template_125 t = { 0 };

    t.msg_seq_num = decode_field(EUI32);

    t.sending_time = decode_field(EUI64);

    cur_ptr++;

    t.trade_date = decode_field(EUI32);

    t.sequence_length = (*cur_ptr++ & 0x7f) - 12; // Dunno why subtract 12!?

    printf("\n\n ** SEQUENCE LENGTH : %d\n", t.sequence_length);

    while ( t.sequence_length --> 0 ) { // Yeah, when it tends to 0!

        bool md_entry_size_present = false;

        printf("%u,", t.msg_seq_num);
        printf("%lu,", t.sending_time);

        t.md_update_action = decode_field(EUI32);
        printf("%u,", t.md_update_action);

        printf("\\N,"); // md_price_level absent in 125

        t.security_id_source = 8; // Constant - Not present in
                                  // packet.

        t.security_id = decode_field(EUI32);

        t.rpt_seq = decode_field(EUI32);

        t.md_entry_type = decode_field(EUI32);
        printf("%c,", t.md_entry_type);
        printf("%u,", t.security_id);
        printf("%u,", t.rpt_seq);

        if ( *(cur_ptr + 1) != 0x80 ) {
            int exp = decode_field(EUI32);
```

```
            long int mantissa = decode_field(EUI64);
            t.md_entry_px = mantissa * pow(10, exp) - 1;
            printf("%lu,", t.md_entry_px);
        } else {
            printf("\\N,");
        }

        if ( *cur_ptr != 0x80 ) {
            decode_field(EUI32); // Ignore 4 bytes!
            md_entry_size_present = true;
            t.md_entry_size = decode_field(EUI32) - 1;
        } else {
            cur_ptr++;
        }

        t.md_entry_time = decode_field(EUI32);
        printf("%u,", t.md_entry_time);

        if ( md_entry_size_present ) {
            printf("%d,", t.md_entry_size);
        } else {
            printf("\\N,");
        }

        printf("\\N,"); // QuoteCondition absent in 125
        printf("\\N,"); // NumberOfOrders absent in 125
        printf("\\N,"); // TradingSessionId absent in 125

        printf("%u,", t.trade_date);

        printf("\\N,"); // TickDirection is absent in 125
        printf("\\N,"); // OpenCloseSettleFlag is absent in 125
        printf("\\N,"); // TradeVolume is absent in 125
        printf("\\N,"); // SettlDate is absent in 125
        printf("\\N,"); // TradeCondition is absent in 125
        printf("\\N,"); // AggressorSide is absent in 125
        printf("\\N,"); // MatchEventIndicator is absent in 125

        cur_ptr += 3;
        printf("\n");
    }
```

```
}


struct template_129
{
	unsigned int msg_seq_num;
	unsigned long int sending_time;
	unsigned char pos_dup_flag;
	unsigned int trade_date;
	unsigned char sequence_length;
	unsigned int md_update_action;
	unsigned int md_price_level;
	unsigned char md_entry_type;
	unsigned int security_id_source;
	unsigned int security_id;
	unsigned int rpt_seq;
	char *quote_condition;
	long int md_entry_px;
	unsigned int number_of_orders;
	unsigned int md_entry_time;
	int md_entry_size;
	char *trading_session_id;
	long int net_chg_prev_day;
	char *tick_direction;
	unsigned int open_close_settle_flag;
	unsigned int settl_date;
};

static void template129_decoder(unsigned char *stop_ptr,
			unsigned char *cur_ptr,
			unsigned short int len_left)
{
	struct template_129 t = { 0 };

	t.msg_seq_num = decode_field(EUI32);

	t.sending_time = decode_field(EUI64);

	cur_ptr++;

	t.trade_date = decode_field(EUI32);
```

```
t.sequence_length = (*cur_ptr++ & 0x7f) - 12;

bool first_sequence = true;

while ( t.sequence_length --> 0 ) {
    bool open_close_settle_flag_present = false;
    bool settle_date_present = false;
    bool number_of_orders_present = false;
    bool trading_session_id_present = false;
    bool tick_direction_present = false;
    bool quote_condition_present = false;

    printf("%u,", t.msg_seq_num);
    printf("%lu,", t.sending_time);

    t.md_update_action = decode_field(EUI32);
    printf("%u,", t.md_update_action);

    if ( *(cur_ptr + 1) != 0x80 ) {
        t.md_price_level = decode_field(EUI32) - 1;
        printf("%u,", t.md_price_level);
    } else {
        printf("\\N,");
        cur_ptr++;
    }

    t.md_entry_type = decode_field(EUI32);
    printf("%c,", t.md_entry_type);

    t.security_id_source = 8; // Constant - Not present in packet.

    t.security_id = decode_field(EUI32);
    printf("%u,", t.security_id);

    t.rpt_seq = decode_field(EUI32);
    printf("%u,", t.rpt_seq);

    unsigned char *s = stop_ptr;
    stop_ptr = ++cur_ptr;
    if ( *cur_ptr != 0x80 ) {
```

```
        t.quote_condition =
                extract_string(&stop_ptr, &cur_ptr, EAscii);
        if ( t.quote_condition[ 0 ] )    {
                        quote_condition_present = true;
        } else {
                free(t.quote_condition);
                t.quote_condition = NULL;
        }
} else {
        cur_ptr--;
}
stop_ptr = s;

cur_ptr += 1;
int exp = decode_field(EUI32);
long int mantissa = decode_field(EUI64);
t.md_entry_px = mantissa * pow(10, exp);
printf("%lu,", t.md_entry_px);

if ( *(cur_ptr + 1) != 0x80 ) {
        number_of_orders_present = true;
        t.number_of_orders = decode_field(EUI32) - 1;
} else {
        cur_ptr++;
}

t.md_entry_time = decode_field(EUI32);
printf("%u,", t.md_entry_time);

if ( *++cur_ptr != 0x80 ) {
        cur_ptr--;
        t.md_entry_size = decode_field(EUI32) - 1;
        printf("%d,", t.md_entry_size);
} else {
        printf("\\N,");
        cur_ptr++;
}

if ( quote_condition_present ) {
        printf("%c,", t.quote_condition[ 0 ]);
        free(t.quote_condition);
```

```
        t.quote_condition = NULL;
    } else {
        printf("\\N,");
    }

    stop_ptr = ++cur_ptr;
    if ( *cur_ptr != 0x80 ) {
        t.trading_session_id = extract_string(&stop_ptr,
                                &cur_ptr, EAscii);
        if ( t.trading_session_id[ 0 ] ) {
            trading_session_id_present = true;
        } else {
            free(t.trading_session_id);
            t.trading_session_id = NULL;
        }
    } else {
        cur_ptr++;
    }

    if ( *cur_ptr != 0x80 ) {
        t.tick_direction =
            extract_string(&stop_ptr, &cur_ptr, EAscii);
        if ( t.tick_direction && t.tick_direction [ 0 ] ) {
            tick_direction_present = true;
        } else {
            free(t.tick_direction);
            t.tick_direction = NULL;
        }
    } else {
        cur_ptr++;
    }

    cur_ptr++;
    if ( *(cur_ptr + 1) != 0x80 ) {
        open_close_settle_flag_present = true;
        t.open_close_settle_flag = decode_field(EUI32) - 1;
    } else {
        cur_ptr++;
    }

    if ( *cur_ptr++ != 0x80 ) {
```

```
        settle_date_present = true;
        t.settl_date = decode_field(EUI32) - 1;
} else {
        cur_ptr++;
}

if ( number_of_orders_present ) {
        printf("%u,", t.number_of_orders);
} else {
        printf("\\N,");
}

if ( trading_session_id_present ) {
        printf("%c,", t.trading_session_id[ 0 ]);
        free(t.trading_session_id);
        t.trading_session_id = NULL;
} else {
        printf("\\N,");
}

printf("%u,", t.trade_date);

if ( tick_direction_present ) {
        printf("%s,", t.tick_direction);
} else {
        printf("\\N,");
}

if ( open_close_settle_flag_present ) {
        printf("%u,", t.open_close_settle_flag);
} else {
        printf("\\N,");
}

if ( settle_date_present ) {
        printf("%u,", t.settl_date);
} else {
        printf("\\N,");
}

printf("\\N,"); // Trade volume absent in 129
```

```
            printf("\\N,"); // TradeCondition absent in 129
            printf("\\N,"); // AggressorSide absent in 129
            printf("\\N"); // MatchEventIndicator absent in 129

            if ( first_sequence ) {
                cur_ptr -= 3;
                first_sequence = false;
            } else {
                cur_ptr++;
            }

            printf("\n");
        }
}

static void init_template_decoders_hash_table(void)
{
    template_decoder[ 117 ] = template117_decoder;
    template_decoder[ 122 ] = template122_decoder;
    template_decoder[ 125 ] = template125_decoder;
    template_decoder[ 129 ] = template129_decoder;
    template_decoder[ 131 ] = template131_decoder;
}

static void init_field_calculator_hash_table(void)
{
    calc_func[ EUI32 ] = (unsigned long long int (*)
                (unsigned char *,
                    unsigned char *))calc_uint32;
    calc_func[ EUI64 ] = (unsigned long long int (*)
                (unsigned char *,
                    unsigned char *))calc_uint64;
    calc_func[ EUI16 ] = (unsigned long long int (*)
                (unsigned char *,
                    unsigned char *))calc_uint16;
}

void packetHandler(u_char *userData, const struct pcap_pkthdr* pkthdr,
                            const u_char* packet)
{
    DEBUG("Inside %s\n", __func__);
```

```
    const struct ether_header* ethernetHeader = NULL;
    const struct ip* ipHeader = NULL;
    const struct tcphdr* tcpHeader = NULL;
    char sourceIp[ INET_ADDRSTRLEN ] = "";
    char destIp[ INET_ADDRSTRLEN ] = "";
    u_int sourcePort = 0, destPort = 0;
    u_char *data = NULL;
    int dataLength = 0;
    char dataStr[ 1000 ] = "";

#ifdef __DEBUG__
    static unsigned long long int n = 0;
#endif

    ethernetHeader = (struct ether_header*)packet;
    if (ntohs(ethernetHeader->ether_type) == ETHERTYPE_IP)
    {
        ipHeader = (struct ip*)(packet +
                    sizeof(struct ether_header));
        inet_ntop(AF_INET, &(ipHeader->ip_src),
                    sourceIp, INET_ADDRSTRLEN);
        inet_ntop(AF_INET, &(ipHeader->ip_dst), destIp,
                        INET_ADDRSTRLEN);

        /**
         * Note: market data (including CME) is UDP not TCP.
         * note the limited sample data provided only contains
         * UDP multi cast data from a single group
         * (channel 7 -> "ES") **/
        if (ipHeader->ip_p == IPPROTO_UDP)
        {
            // printf("\n\n****\n");
#ifdef __DEBUG__
            printf("UDP Packet no. : %llu\n", ++n);
#endif

            struct udphdr *uhdr =
                (struct udphdr *)
                (packet + sizeof(struct ether_header) +
                        sizeof(struct ip));
```

```
        unsigned char *data = (u_char *)((char *)uhdr +
                        sizeof(struct udphdr));

#ifdef __DEBUG__
        printf("UDP payload offset : %ld\n",
                        data - packet);
#endif

        union udp_payload_len
        {
            unsigned char c[ 2 ];
            unsigned short int s;
        } p;
        p.s = 0;
        p.c[ 1 ] = uhdr->len & 0xff;
        p.c[ 0 ] =
            ((unsigned short int)
                uhdr->len & 0xff00) >> 8;

        unsigned short int data_len = p.s;
        data_len -= sizeof(struct udphdr);
        unsigned short int len_left = data_len;
#ifdef __DEBUG__
                        unsigned long long int d = 0;
                        while ( d < data_len )
                        {
                                printf("%02x ", data[ d++ ]);
                                if ( !(d % 10) )
                                {
                                        putchar('\n');
                                }
                        }
        if (d % 10)
            putchar('\n');
#endif

                        union msg_seq_num
                        {
                                unsigned char c[ 4 ];
                                unsigned int s;
```

```
                        } m;

        // Loop unrolled for efficiency!
                        m.c[ 3 ] = *data++;
                        m.c[ 2 ] = *data++;
                        m.c[ 1 ] = *data++;
                        m.c[ 0 ] = *data++;

#ifdef __DEBUG__
        unsigned int msg_seq_num = m.s;
        printf("msg_seq_num : %u\n", msg_seq_num);
#endif

        // printf("Subchannel id : %u\n",
        //              *(unsigned char*)data);

        // Decoding FIX/FAST message
        unsigned char *f_msg =
                (unsigned char *)data + 1;
#ifdef __DEBUG__
        printf("Fix/Fast msg starting addr : %p\n",
                        (void *)f_msg);
#endif

        unsigned char *cur_ptr = f_msg;
        cur_ptr = get_field(cur_ptr, &len_left);

#ifdef __DEBUG__
        printf("Fix/Fast msg length : %hu\n", data_len);
#endif

        unsigned int templ_id_present =
                *cur_ptr & (unsigned char)0x40;
        unsigned char* stop_ptr = cur_ptr;

        cur_ptr++;

        static unsigned short int templ_id = 0xffff;
        if ( templ_id_present )
        {
              cur_ptr = get_field(cur_ptr, &len_left);
```

```
                templ_id = calc_uint16(stop_ptr,
                                cur_ptr);
                template_ids_presence[ templ_id ]++;

                // printf("Template ID: %hu\n", templ_id);
            }
            else if ( templ_id != 0xffff )
            {
                // Get it from the previous packet!
                template_ids_presence[ templ_id ]++;
                // printf("Template ID: %hu\n", templ_id);
            }
            else
            {
                // printf("No template ids seen so "
                //              "far!\n");
            }


            if ( template_decoder[ templ_id ] )
            {
                template_decoder[ templ_id ]
                    (stop_ptr, cur_ptr, len_left);
            }

#ifdef __DEBUG__
            // getchar();
#endif
        }
        else if (ipHeader->ip_p == IPPROTO_TCP)
        {
            exit(-1);

            tcpHeader = (struct tcphdr*)(packet +
                    sizeof(struct ether_header) +
                        sizeof(struct ip));
            sourcePort = ntohs(tcpHeader->source);
            destPort = ntohs(tcpHeader->dest);
            data = (u_char*)(packet +
                    sizeof(struct ether_header)
                        + sizeof(struct ip) +
```

```
                    sizeof(struct tcphdr));
            dataLength = pkthdr->len -
                    (sizeof(struct ether_header)
                        + sizeof(struct ip) +
                        sizeof(struct tcphdr));

            // convert non-printable characters, other
            // than carriag return, line feed, or tab into
            // periods when displayed.
            int i = 0;
            for (i = 0; i < dataLength; i++)
            {
                if ((data[i] >= 32 && data[ i ] <= 126)
                                ||
                        data[ i ] == 10 ||
                        data[ i ] == 11 ||
                        data[ i ] == 13)
                {
                        dataStr[ i ] = (char)data[ i ];
                }
                else
                {
                        dataStr[ i ] = '.';
                }
            }

            // print the results
            printf("sourceIp : %s sourcePort : %u "
                "destIp : %s destPort : %u\n",
                sourceIp, sourcePort, destIp, destPort);
            if (dataLength > 0)
            {
                printf("dataStr : %s\n", dataStr);
            }
        }
    }
}
```

Software Verification Decoder

Makefile

```
#taken from http://mrbook.org/tutorials/make/
CC=gcc
CFLAGS=-c -Wall
LDFLAGS=-lpcap -lm
SOURCES=decode.c
OBJECTS=$(SOURCES:.cpp=.o)
EXECUTABLE=PcapDecoder

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
	$(CC) $(LDFLAGS) $(OBJECTS) -o $@

.cpp.o:
	$(CC) $(CFLAGS) $< -o $@

run: all
	./$(EXECUTABLE) ./7IA_2013_03_19.dat.pcap > op
	diff ./op ./7IA_2013_03_19.csv_X_chopped > difference

clean:
	rm PcapDecoder
```

Software Verification Book Builder

book_builder.py

```
import socket
import dpkt
import sys
import struct
import random
import csv

def initBook(numOfLevels):
	emptyDic = {
	'EntryPx' : -1,
	'EntrySize' : -1,
	'NoO' : -1,
	'isValid' : -1
	}
	bookList = []
	for i in range(0, numOfLevels):
		bookList.append(emptyDic)
	return bookList
"""
def packSecurityID(securityIdValue):
	if len(securityIdValue) > 0:
		data = struct.pack(">c", securityIdValue[1])
	else:
		data = struct.pack(">c", "x")
	for i in range(1, 7):
		if i < len(securityIdValue):
			data = data + struct.pack(">c", securityIdValue[i])
		else:
```

```
                data = data + struct.pack(">c", "x")
        return data
"""

def packBookContent(bookBid, bookAsk):
        data = 0
        for i in bookBid:
                price = i['EntryPx']
                price = int(price)
                amount = int(i['EntrySize'])
                noo = int(i['NoO'])
                data = data + struct.pack(">ihh", price, amount, noo)
        for i in bookAsk:
                price = int(i['EntryPx'])
                amount = int(i['EntrySize'])
                noo = int(i['NoO'])
                data = data + struct.pack(">ihh", price, amount, noo)
        return data


f = open('7IA_2013_03_19.dat.pcap', 'rb')
pcapReader = dpkt.pcap.Reader(f)
pcapWriter = dpkt.pcap.Writer(file("snapshot.pcap","wb"))
counter = 0

for ts, packet in pcapReader:
        ether_packet = dpkt.ethernet.Ethernet(packet)
        if ether_packet.type == dpkt.ethernet.ETH_TYPE_IP:
                ip_packet = ether_packet.data
                udp_packet = ip_packet.data
                break

with open('7IA_2013_03_19.csv_X_chopped', 'rb') as csvfile:
        reader = csv.reader(csvfile)
        row = reader.next()
        print row

        """
        Message Sequential Number
        """
        MsgSeqNum = row.index("MsgSeqNum")
```

```
"""
SecurityID
"""
SecurityID = row.index("SecurityID")
"""
Type of Market Data update action.
'0' means insert(new)
'1' means modify(change)
'2' means delete(cancel)
"""
MDUpdateAction = row.index("MDUpdateAction")
"""
Position in the book. Start from 1.
"""
MDPriceLevel = row.index("MDPriceLevel")
"""
Type of Market Data entry.
'0' for bid
'1' for ask
"""
MDEntryType = row.index("MDEntryType")
MDEntryPx = row.index("MDEntryPx")
"""
Quantity or volume represented by the Market Date Entry.
"""
MDEntrySize = row.index("MDEntrySize")
"""
Number of orders in the market.
"""
NumberOfOrders = row.index("NumberOfOrders")

"""
print MDUpdateAction
print MDPriceLevel
print MDEntryType
print MDEntryPx
print MDEntrySize
print NumberOfOrders
"""

emptyDic = {
```

```
    'EntryPx' : -1,
    'EntrySize' : -1,
    'NoO' : -1,
    'isValid' : -1
    }
    bookAsk = initBook(10)
    bookBid = initBook(10)
    securityIdValue = 0
    bidChange = 0
    askChange = 0
    counter = 0

    offset = 2 ** 6
    mask = 2 ** 10 - 1

    #temMsgSeqNum

    for row in reader:
#   for i in range(0, 1299):
#       row = reader.next()
        if counter == 0:
            preMsgSeqNum = row[MsgSeqNum]
            curMsgSeqNum = row[MsgSeqNum]
            counter = counter + 1
        else:
            preMsgSeqNum = curMsgSeqNum
            curMsgSeqNum = row[MsgSeqNum]
        """
        Compare previous MsgSeqNum and current MsgSeqNum
        to see if a packet ends.
        """
        #print preMsgSeqNum, curMsgSeqNum
        #breakpoint = raw_input()
        if preMsgSeqNum != curMsgSeqNum:
            bidChange = bidChange & mask
            askChange = askChange & mask
            if bidChange != 0 or askChange != 0:
                securityIdValue = int(row[SecurityID])
                #length = len(securityIdValue)
                #print length
                bidChange = bidChange * offset
```

```
                askChange = askChange * offset
                udp_packet.data = struct.pack(">iHiHH", 0, 0, securityIdValue,
bidChange, askChange)
                data = ''
                for i in bookBid:
                    price = i['EntryPx']
                    price = int(price)
                    if i['EntrySize'] == '\\N':
                        amount = -1
                    else:
                        amount = int(i['EntrySize'])
                    if i['NoO'] == '\\N':
                        noo = -1
                    else:
                        noo = int(i['NoO'])
                    data = data + struct.pack(">ihh", price, amount, noo)
                for i in bookAsk:
                    price = int(i['EntryPx'])
                    if i['EntrySize'] == '\\N':
                        amount = -1
                    else:
                        amount = int(i['EntrySize'])
                    if i['NoO'] == '\\N':
                        noo = -1
                    else:
                        noo = int(i['NoO'])
                    data = data + struct.pack(">ihh", price, amount, noo)
                udp_packet.data = udp_packet.data + data
                udp_packet.ulen = len(udp_packet.data) + 8
                ip_packet.data = udp_packet
                ip_packet.len = 202
                ether_packet.data = ip_packet
                pcapWriter.writepkt(ether_packet, counter)
                counter = counter + 1
                askChange = 0
                bidChange = 0

        if row[MDEntryType] == '0':
            targetBookList = bookBid
            #print '0'
        elif row[MDEntryType] == '1':
```

```
        targetBookList = bookAsk
        #print '1'
    else:
        #print 'else'
        continue
    if row[MDPriceLevel] == '\\N':
        priceLevelInList = 1
    else:
        priceLevelInList = int(row[MDPriceLevel]) - 1
    bitPostion = 9 - priceLevelInList
    #print priceLevelInList
    if row[MDUpdateAction] == '0':
        insertDic = {
        'EntryPx' : row[MDEntryPx],
        'EntrySize' : row[MDEntrySize],
        'NoO' : row[NumberOfOrders],
        'isValid' : 1
        }
        tempChange = 2 ** (bitPostion + 1) - 1
        targetBookList.insert(priceLevelInList, insertDic)
        targetBookList = targetBookList[0:10]
        #print '0'
    elif row[MDUpdateAction] == '1':
        insertDic = {
        'EntryPx' : row[MDEntryPx],
        'EntrySize' : row[MDEntrySize],
        'NoO' : row[NumberOfOrders],
        'isValid' : 1
        }
        tempChange = 2 ** bitPostion
        targetBookList.pop(priceLevelInList)
        targetBookList.insert(priceLevelInList, insertDic)
        #print '1'
    elif row[MDUpdateAction] == '2':
        tempChange = 2 ** (bitPostion + 1) - 1
        targetBookList.pop(priceLevelInList)
        targetBookList.insert(priceLevelInList, emptyDic)
        #print '2'
    else:
        continue
```

```
            if row[MDEntryType] == '0':
                bidChange = bidChange | tempChange
                bookBid = targetBookList
                #print '0'
            elif row[MDEntryType] == '1':
                askChange = bidChange | tempChange
                bookAsk = targetBookList

        for i in bookBid:
            print i

        print '===>>>===>>>===>>>===>>>===>>>'
        for i in bookAsk:
            print i
"""
udp_packet.data = struct.pack(">hh", 1, 2)
counter = counter + 1
ip_packet.data = udp_packet
ether_packet.data = ip_packet
pcapWriter.writepkt(ether_packet, counter)
"""
f.close()
```

Software Support Auto Generation of VHDL testbench

```
# By Chang Liu and Danqing Hua
# Converts cvs file inot vhdl testbench for bookbuilder

#!/usr/bin/perl
use strict;
use warnings;

# Input file
my $file_name1="7IA_2013_03_19.csv_X_chopped";
# Output file
my $file_name2="testbench.vhd";

my $count = 0;
my $update_action;
my $price_level;
my $entry_type;
my $price;
my $amount;

# If output file exist, delete it.
if (-e $file_name2) {
	`rm -f $file_name2`;
}

open INPUT,"<", $file_name1 or die "Couldn't open",$file_name1,"for reading";
open OUTPUT, ">>",$file_name2 or die "Couldn't open",$file_name2,"for writting";

while (my $line = <INPUT>) {
	chomp $line;
	$count++;
	# Parse CSV file
	my @title_line = split /,/, $line;
	# For the first line, find the index(location) of information we need.
	if ($count == 1) {
		for (my $index = 0; $index < scalar @title_line; $index++) {
			my $tmp = $title_line[$index];
			#print "$tmp\n";
			if ($tmp eq "MDUpdateAction") {
				$update_action = $index;
			} elsif ($tmp eq "MDPriceLevel") {
```

```
                $price_level = $index;
            } elsif ($tmp eq "MDEntryType") {
                $entry_type = $index;
            } elsif ($tmp eq "MDEntryPx") {
                $price = $index;
            } elsif ($tmp eq "MDEntrySize") {
                $amount = $index;
            }
        }
        print OUTPUT << "Lable";
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test_book is
generic(
   PRICE_WIDTH: natural:=32;
   AMOUNT_WIDTH: natural:=8;
   LEVEL_WIDTH: natural :=4;
   ID_WIDTH: natural:=8;
   Cell_WIDTH : natural := 64;
   Book_count : natural := 10;
   Cell_count : natural :=20
);
end entity;

architecture tb of test_book is
signal clk, reset       :    std_logic := '0';
signal entry_type       :    std_logic_vector(1 downto 0):= (others => '0');
signal  price           :    std_logic_vector(PRICE_WIDTH-1 downto 0):= (others =>
'0');
signal amount           :    std_logic_vector(AMOUNT_WIDTH-1 downto 0):= (others
=> '0');
signal level            :    std_logic_vector(LEVEL_WIDTH -1 downto 0):= (others =>
'0');
signal sb               :    std_logic:= '0';
signal stock_id         :    std_logic_vector(ID_WIDTH-1 downto 0):= (others => '0');
signal command_status:    std_logic_vector(1 downto 0):= (others => '0');
signal exe              :    std_logic:= '0';
component book_top is
port(
```

```
        clk                     : in std_logic;
        reset                   : in std_logic;
        entry_type              :   in std_logic_vector(1 downto 0);
        price                   :   in std_logic_vector(PRICE_WIDTH-1 downto 0);
        amount                      :   in  std_logic_vector(AMOUNT_WIDTH-1  downto
0);
        level                   :   in std_logic_vector(LEVEL_WIDTH -1 downto 0);
        sb                      :   in std_logic;
        stock_id                :   in std_logic_vector(ID_WIDTH-1 downto 0);
        command_status          : in std_logic_vector(1 downto 0);
        exe                     : out std_logic
);
end component;

begin

book_top_in_use: book_top port map( clk=>clk,
                                    reset=>reset,
                                        entry_type=>entry_type,
                                        price=>price,
                                        amount=>amount,
                                        level=>level,
                                        sb=>sb,
                                        stock_id=>stock_id,
                                        command_status=>command_status,
                                        exe=>exe);
process
begin
clk <= '0';
wait for 20 ns;
loop
clk <= '0';
wait for 20 ns;
clk <= '1';
wait for 20 ns;
end loop;
end process;

process
begin
reset<='1';
```

```
wait for 80 ns;
reset<='0';

Lable
      # Set the upperbound to 100 for test.
      } elsif ($count > 1 && $count < 100) {
            # Convert data into VHDL input.
            print OUTPUT << "Lable";
----------------------------------------
wait for 160 ns;
Lable
            printf OUTPUT "update_action_action=\"%02b\";\n",
$title_line[$update_action];
#update_action= "$title_line[$update_action]";          --this must be 2 bit binary--
            my $tmp = $title_line[$price_level];
            if ($tmp eq "\\N") {
                  $tmp = 0;
            }
            printf OUTPUT "level=\"%04b\";\n", $tmp;
            printf OUTPUT "sb='1';\n";
            printf OUTPUT "price=\"%032b\";\n", $title_line[$price];
            printf OUTPUT "amount=\"%08b\";\n", $title_line[$amount];
#level="$title_line[$price_level]";                    --this must be 4 bit binary--
#sb='1';
#price="$title_line[$price]";                          --this must be 32 bit binary--
#amount="$title_line[$amount]";                        --this must be 8 bit
binary--
            print OUTPUT << "Lable";
command_status="11";
----------------------------------------
Lable
      }
}

print OUTPUT << "Lable";
wait for 160ns;
command_status="00";
wait;
end process;
end tb;
Lable
```

```
close INPUT;
close OUTPUT; # By Chang Liu and Danqing Hua
# Converts cvs file inot vhdl testbench for bookbuilder

#!/usr/bin/perl
use strict;
use warnings;

# Input file
my $file_name1="7IA_2013_03_19.csv_X_chopped";
# Output file
my $file_name2="testbench.vhd";

my $count = 0;
my $update_action;
my $price_level;
my $entry_type;
my $price;
my $amount;

# If output file exist, delete it.
if (-e $file_name2) {
	`rm -f $file_name2`;
}

open INPUT,"<", $file_name1 or die "Couldn't open",$file_name1,"for reading";
open OUTPUT, ">>",$file_name2 or die "Couldn't open",$file_name2,"for writting";

while (my $line = <INPUT>) {
	chomp $line;
	$count++;
	# Parse CSV file
	my @title_line = split /,/, $line;
	# For the first line, find the index(location) of information we need.
	if ($count == 1) {
		for (my $index = 0; $index < scalar @title_line; $index++) {
			my $tmp = $title_line[$index];
			#print "$tmp\n";
			if ($tmp eq "MDUpdateAction") {
				$update_action = $index;
```

```
                } elsif ($tmp eq "MDPriceLevel") {
                    $price_level = $index;
                } elsif ($tmp eq "MDEntryType") {
                    $entry_type = $index;
                } elsif ($tmp eq "MDEntryPx") {
                    $price = $index;
                } elsif ($tmp eq "MDEntrySize") {
                    $amount = $index;
                }
            }
            print OUTPUT << "Lable";
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test_book is
generic(
   PRICE_WIDTH: natural:=32;
   AMOUNT_WIDTH: natural:=8;
   LEVEL_WIDTH: natural :=4;
   ID_WIDTH: natural:=8;
   Cell_WIDTH : natural := 64;
   Book_count : natural := 10;
   Cell_count : natural :=20
);
end entity;

architecture tb of test_book is
signal clk, reset       :    std_logic := '0';
signal entry_type       :    std_logic_vector(1 downto 0):= (others => '0');
signal  price           :    std_logic_vector(PRICE_WIDTH-1 downto 0):= (others =>
'0');
signal amount           :    std_logic_vector(AMOUNT_WIDTH-1 downto 0):= (others
=> '0');
signal level            :    std_logic_vector(LEVEL_WIDTH -1 downto 0):= (others =>
'0');
signal sb               :    std_logic:= '0';
signal stock_id         :    std_logic_vector(ID_WIDTH-1 downto 0):= (others => '0');
signal command_status:    std_logic_vector(1 downto 0):= (others => '0');
signal exe              :    std_logic:= '0';
component book_top is
```

```
port(
        clk                     : in std_logic;
        reset                   : in std_logic;
        entry_type              :    in std_logic_vector(1 downto 0);
        price                   :    in std_logic_vector(PRICE_WIDTH-1 downto 0);
        amount                  :    in  std_logic_vector(AMOUNT_WIDTH-1  downto
0);
        level                   :    in std_logic_vector(LEVEL_WIDTH -1 downto 0);
        sb                      :    in std_logic;
        stock_id                :    in std_logic_vector(ID_WIDTH-1 downto 0);
        command_status          : in std_logic_vector(1 downto 0);
        exe                     : out std_logic
);
end component;

begin

book_top_in_use: book_top port map( clk=>clk,
                                    reset=>reset,
                                        entry_type=>entry_type,
                                        price=>price,
                                        amount=>amount,
                                        level=>level,
                                        sb=>sb,
                                        stock_id=>stock_id,
                                        command_status=>command_status,
                                        exe=>exe);
process
begin
clk <= '0';
wait for 20 ns;
loop
clk <= '0';
wait for 20 ns;
clk <= '1';
wait for 20 ns;
end loop;
end process;

process
begin
```

```
reset<='1';
wait for 80 ns;
reset<='0';

Lable
      # Set the upperbound to 100 for test.
      } elsif ($count > 1 && $count < 100) {
            # Convert data into VHDL input.
            print OUTPUT << "Lable";
----------------------------------------
wait for 160 ns;
Lable
            printf                OUTPUT                "update_action_action=\"%02b\";\n",
$title_line[$update_action];
#update_action= "$title_line[$update_action]";          --this must be 2 bit binary--
            my $tmp = $title_line[$price_level];
            if ($tmp eq "\\N") {
                  $tmp = 0;
            }
            printf OUTPUT "level=\"%04b\";\n", $tmp;
            printf OUTPUT "sb='1';\n";
            printf OUTPUT "price=\"%032b\";\n", $title_line[$price];
            printf OUTPUT "amount=\"%08b\";\n", $title_line[$amount];
#level="$title_line[$price_level]";                     --this must be 4 bit binary--
#sb='1';
#price="$title_line[$price]";                           --this must be 32 bit binary--
#amount="$title_line[$amount]";                                --this must be 8 bit
binary--
            print OUTPUT << "Lable";
command_status="11";
----------------------------------------
Lable
      }
}

print OUTPUT << "Lable";
wait for 160ns;
command_status="00";
wait;
end process;
end tb;
```

```
Lable

close INPUT;
close OUTPUT;
```

Software Support Auto Generation of Decoder

xml_parser.py

```
"""
XML parser
Created by Chang Liu
"""

from xml.dom import minidom

xmldoc = minidom.parse('templates.xml')

"""
Get nodes of tag "templates"
"""
templates = xmldoc.firstChild
"""
Get list of templates.
Each element of the list is a template.
"""
templateList = templates.getElementsByTagName('template')
templateId = raw_input()

def getElements(templateId, templateList):
    elements = []
    for i in templateList:
        if i.attributes['dictionary'].value == str(templateId):
            elements = i.childNodes
    return elements
```

```
def getFiledName(element):
    return element.attributes['name'].value

def checkConstant(elements):
    isConstant = 0
    for j in elements:
        try:
            if j.tagName == 'constant':
                isConstant = 1
        except:
            pass
    return isConstant

"""
elements is a list of fields in a template
"""
elements = getElements(templateId, templateList)

"""
Each i is a field in template
"""
for i in elements:
    try:
        tag = i.tagName
        name = getFiledName(i)
        if tag != 'sequence':
            """
            For none sequence field
            """
            isConstant = 0
            subElements = i.childNodes
            isConstant = checkConstant(subElements)
            if isConstant == 0:
                if 'presence' in i.attributes.keys():
                    print name + ",1,0,0"
                else:
                    print name + ",0,0,0"
        else:
            """
            For sequence field
            """
```

```
            seqElements = i.childNodes
            tempList = []
            for j in seqElements:
                try:
                    seqTag = j.tagName
                    tempList.append(j)
                except:
                    pass

            for j in tempList:
                isSeqHead = 0
                isSeqTail = 0
                isConstant = 0
                name = getFiledName(j)
                subElements = j.childNodes
                isConstant = checkConstant(subElements)
                if isConstant == 0:
                    if j == tempList[0]:
                        isSeqHead = 1
                    if j == tempList[len(tempList) - 1]:
                        isSeqTail = 1
                    if 'presence' in j.attributes.keys():
                        print name + ",1,%d,%d" %(isSeqHead, isSeqTail)
                    else:
                        print name + ",0,%d,%d" %(isSeqHead, isSeqTail)

    except:
        pass
```

Software Support Auto Comparison

decoder_output_convert.pl

```
# This is the script to convert output from
# VHDL decoder into the same format as the golden output.
# By Chang Liu

#!/usr/bin/perl
```

```
use strict;
use warnings;

sub bin2dec {
	return unpack("N", pack("B32", substr("0" x 32 . shift, -32)));
}

# Input file
my $file_name1 = "res.log";
# Output file
my $file_name2 = "vhdl_output.csv";

if (-e $file_name2) {
	`rm -f $file_name2`;
}

open INPUT, "<", $file_name1 or die "Couldn't open", $file_name1, "for reading";
open OUTPUT, ">>", $file_name2 or die "Couldn't open", $file_name2, "for writting";

<INPUT>;
while (my $line = <INPUT>) {
	chomp $line;
	# Parse CSV file
	my @title_line = split /,/, $line;

	for (my $index = 0; $index < 6; $index++) {
		my $tmp = $title_line[$index];
		if ($tmp != 0 || $index == 0) {
			$tmp = bin2dec($tmp);
			if ($index == 2) {
				$tmp -= 128;
				printf OUTPUT "%c,", $tmp;
			} else {
				print OUTPUT "$tmp,";

			}
		} else {
			print OUTPUT "\\N,";
		}
	}
```

```
    my $tmp = $title_line[6];
    if ($tmp != 0) {
        $tmp = bin2dec($tmp);
        print OUTPUT "$tmp\n";
    } else {
        print OUTPUT "\\N\n";
    }
}
```

#process_raw_golden.pl

```
use strict;
use warnings;

# Input file
my $file_name1 = "raw_golden_output.csv";
# Output file
my $file_name2 = "golden_output.csv";

if (-e $file_name2) {
    `rm -f $file_name2`;
}

open INPUT, "<", $file_name1 or die "Couldn't open", $file_name1, "for reading";
open OUTPUT, ">>", $file_name2 or die "Couldn't open", $file_name2, "for writting";

<INPUT>;
for (my $count = shift; $count > 0; $count--) {
    my $line = <INPUT>;
    if ($line) {
        chomp $line;
        print OUTPUT "$line\n";
    } else {
        print "The index in too large!\n";
    }
}
```

Software Verification Auto Simulation

compile_run.tcl

```
# tcl script used to compile, recompile and simulate our VHDL codes
# Reference: http://www.doulos.com/knowhow/tcltk/examples/modelsim/
# Modified by Chang Liu(cl3078@columbia.edu)

puts {
   ModelSimSE general compile script version 1.1
   Copyright (c) Doulos June 2004, SD
}

# Simply change the project settings in this section
# for each new project. There should be no need to
# modify the rest of the script.

set library_file_list {
                                        design_library {Global_Constants.vhd
                                                        txt_util.vhd
                                                        stream_pcap.vhd
                                                        Header_Fields.vhd
                                                        Packetizer_Alt.vhd
                                                        type.vhd
                                                        file.vhd
                                                        packet_buffer.vhd
                                                        top_header.vhd
                                                        mux.vhd
                                                        decoder_gen_117.vhd
                                                        decoder_gen_122.vhd
                                                        decoder_gen_125.vhd
                                                        decoder_gen_129.vhd
                                                        decoder_gen_131.vhd
                                                        cmdbuffer.vhd
                                                        fifo.vhd
                                                        memory_controller.vhd
                                                        RAM.vhd
                                                        book.vhd
                                                        book_top.vhd
```

```
                                                top_5_templates.vhd
                                                top_fixfast.vhd}
                                    test_library      {test_top_fixfast.vhd}
}
set top_level                       test_library.tb_top_fixfast
set wave_patterns {
                                    /*
}
set wave_radices {
                                    hexadecimal {data q}
}


# After sourcing the script from ModelSim for the
# first time use these commands to recompile.

proc r    {} {uplevel #0 source compile.tcl}
proc rr {} {global last_compile_time
                    set last_compile_time 0
                    r                                   }
proc q    {} {quit -force                       }

#Does this installation support Tk?
set tk_ok 1
if [catch {package require Tk}] {set tk_ok 0}

# Prefer a fixed point font for the transcript
set PrefMain(font) {Courier 10 roman normal}

# Compile out of date files
set time_now [clock seconds]
if [catch {set last_compile_time}] {
   set last_compile_time 0
}
foreach {library file_list} $library_file_list {
   vlib $library
   vmap work $library
   foreach file $file_list {
      if { $last_compile_time < [file mtime $file] } {
         if [regexp {.vhdl?$} $file] {
            vcom -93 $file
```

```
            } else {
                vlog $file
            }
            set last_compile_time 0
        }
    }
}
set last_compile_time $time_now

# Load the simulation
eval vsim $top_level

# If waves are required
if [llength $wave_patterns] {
    noview wave
    foreach pattern $wave_patterns {
        add wave $pattern
    }
    configure wave -signalnamewidth 1
    foreach {radix signals} $wave_radices {
        foreach signal $signals {
            catch {property wave -radix $radix $signal}
        }
    }
    if $tk_ok {wm geometry .wave [winfo screenwidth .]x330+0-20}
}

# Run the simulation
run -all

# If waves are required
if [llength $wave_patterns] {
    if $tk_ok {.wave.tree zoomfull}
}

puts {
    Script commands are:

    r = Recompile changed and dependent files
   rr = Recompile everything
    q = Quit without confirmation
```

```
}

# How long since project began?
if {[file isfile start_time.txt] == 0} {
    set f [open start_time.txt w]
    puts $f "Start time was [clock seconds]"
    close $f
} else {
    set f [open start_time.txt r]
    set line [gets $f]
    close $f
    regexp {\d+} $line start_time
    set total_time [expr ([clock seconds]-$start_time)/60]
    puts "Project time is $total_time minutes"
}
```