6.111 final: HFT Accelerator

Natnael Kahssay, Endrias Kahssay, Ziheng Wang

December 2019

Contents

1 Background & Motivation

1.1 Terminology

Stock: a virtual object which someone can sell or buy.
Order: a request to buy or sell a stock.
Bids: a request to buy a stock at a particular price
Ask: a request to sell a stock at a particular price
Exchange: a third party entity that matches groups interested in buying / selling stocks. Sends market updates to market participants over network.
Automated trading: a computer strategy that connects to an exchange and submits orders with the objective of making money.

1.2 HFT: High frequency trading:

HFT or High frequency trading, a subset of automated trading where the objective is to react quickly to changes in the market, and submit trades. The flow of information is as follows: the exchange sends market data through Ethernet (describing the current price of the stock, who is interested in buying and selling). The HFT engine is responsible for receiving this data, parsing it with the protocol provided by the exchange, updating is internal state about the market, and submitting orders back to the exchange in reaction to market updates.
In summary:
HFT firms and market makers need ultra-low latency solutions to quickly:
1) parse the market data stream from the exchange.
2) update their knowledge of the market to keep track of best prices for stocks
(building an order book).
3) Submit trades to the exchange based on the information

1.3 Our project:

For our 6.111 final project, we built an HFT (High Frequency Trading) accelerator in FPGA. In our implementation, we have an Ethernet stack (TCP), which receives the market data from the laptop. It forwards it to the parser module, which is responsible for implementing the exchange protocol and parsing what orders were submitted / canceled, or what trades happened on the market.

The parser module forwards the parsed information into the order book module, which is used to represent the current state of the market.

The order book module forwards the best price for each stock to the trading module. The trading module does calculations based on the best price of the stocks at this time step, as well as in the previous time step to decide what orders to submit to the market.

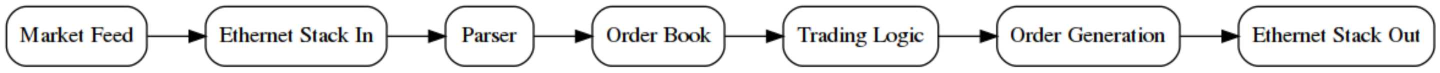


Figure 1: Block diagram for overall module.

1.4 Status:

We were able to get everything working except integration with the Ethernet stack for receiving and sending updates from the order book / trading module. We explain the issue we run into below in the Ethernet section.

1.4.1 Commitment / Basic:

Team: Each individual component works in isolation. Demonstrates functional correctness in simulation, in comparison to C++ or Python test script. Individual components: Ethernet/Parser:

- ✓ Capable of receiving structured message complying to Nasdaq ITCH protocol over Ethernet and grab relevant information Correctly.
- ✓ Supports outputs pertaining to add and cancel order messages.
- ✓ Correctness is verified by cross checking output over Ethernet.

Book building:

- ✓ Correctly supports adding, canceling orders, and market trades in simulation.
- ✓ State of the order book is verified with the output of a python script.

Trading logic:

- ✓ Has logic to update the covariance matrix and returns vector to update linear system to solve.
- ✓ Correctly solves a N by N linear system, subject to certain constraints (no fixed point overflow during the QR and upper triangular solve, bad condition number arising from fixed point error etc.) in simulation. Compare solutions to Python test bench.
- ✓ Correctly generates orders to submit over the Ethernet stack.

1.4.2 Goal:

Team:

- ✓ Make sure components work on FPGA.
- ✓ Integration of order book and trading module.
- × Integration of Ethernet and order book / trading.

Individual components: Parser:

- ✓ Capable of receiving structured message complying to NASDAQ ITCH protocol over Ethernet and grab relevant information Correctly.
- ✓ Capable of sending structured message complying to Nasdaq OTCH protocol over Ethernet by utilizing outputs from the Trading logic.
- ✓ Correctness is verified by cross checking output in over Ethernet.

Book building:

- ✓ Correct functionality when synthesized on FPGA.
- ✓ Interfaces with Parser module to get orders / cancels / trades from the market.
- ✓ Interfaces with trading module to expose best bid and ask prices for all the stocks.
- × Periodically forwards state of the order book (price ladder per stock, how many quantities per order) to laptop through Ethernet so that it can be visually displayed on laptop. (Ethernet integration failed due to reasons explained in the Ethernet section).
- ✓ Optimize resource usage to fit with other modules (Parser, and Trading Logic).

Trading logic:

- ✓ Correct functionality when synthesized on FPGA.
- ✓ Correctly interfaces with the order book module, to take snapshots of the market state.
- × Correctly interfaces with the Ethernet stack to actually send those orders over TCP.
- ✓ Optimizes resource usage so that it can fit alongside with the other modules.

1.4.3 Stretch goal:

Team:

- ✓ System performance. Latency measurements across the entire system (latency from getting an order to parsing it, from parsing to building the book, then from book to trade updates). Resource measurements of our current system in terms of logic units used, how much memory used. Compare with state-of-the-art commercial solutions when normalized to resource usage.

Individual components: Ethernet/Parser:

- ✓ Supports back-pressure from Book without dropping packets.
- × Bypass trading logic operating on select information directly from the parsing module. (We found this not to be worthwhile since there we implemented a very complex trading strategy, and if we bypass it our system is likely to lose money)
- × Replace The Microblaze softcore IP used for abstracting TCP/IP protocol with light-weight Ethernet stack. This will enable a tight pipeline from packet to parser
- ✓ Performance and latency measurement of the parser

Book building:

- ✓ Analyzing the theoretical and empirical memory usage, and latency.
- ✓ Explore faster implementations of order book / performance engineering of order book (went with a different approach that what was described since I found that to translate to verilog better).

Trading logic:

- ✓ Explore different designs of linear solver module (folded vs pipeline) and assess the area/latency tradeoff.
- × Merge the rotator and arctan cordic IP modules in a custom cordic module. (Probably quite difficult, but described in literature.)

2 Global Constants

N = 4, number of stocks
B = 16, number of bits in stock price

3 Ethernet Stack and Parser (Natnael)

3.1 Background

Market exchanges format their data feed according to published specification and send it out for their customers over TCP/IP to consume. They also expose a protocol for receiving data from their clients also according to a specification. The goal is to receive data feed from the exchange as fast as possible, process it and extract meaningful information, and build a data structure commonly refereed to as an order book to maintain information that has arrived. The order book is connected with a trading module which to generate orders to send out.

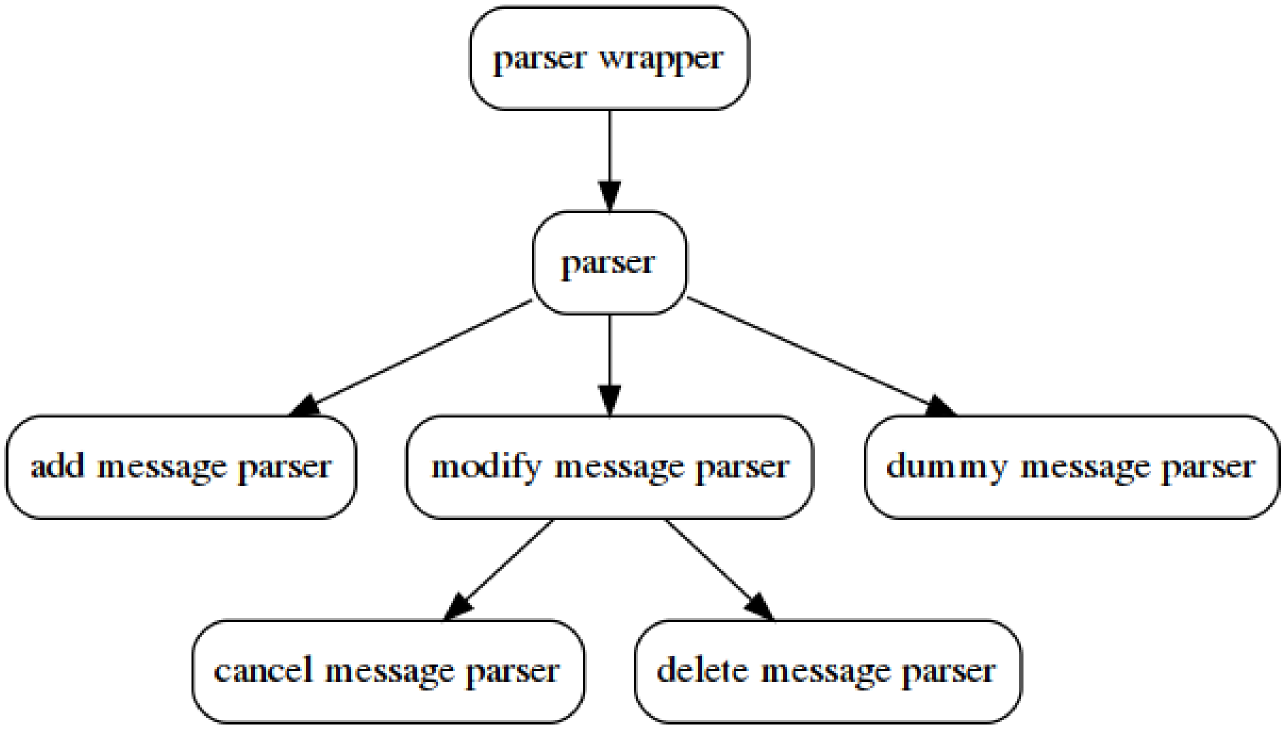
NASDAQ, the exchange of interest, packages their data feed according to Itch protocol. An itch protocol message on a high level consists of the messages length, message type and message content. An example message is provided below. Pertinent messages either add new orders or update the status of currently existing orders.

Message	Message size in bytes	Message meaning
0x13	1	size of message
0x44 = ”D”	1	this message is to delete a previous order
0x01_13	2	which stock this order refers to
0x10_32	2	NASDAQ’s internal tracking number for this message
0x00_20_30_00_40_0a	6	message arrived at exchange this many nano seconds after the exchange opened today
0x00_00_32_12_df_ab_01	8	the reference number of the order this message pertains to

3.2 Implementation

3.3 parser architecture

The parser is broken up into different modules as specified below:



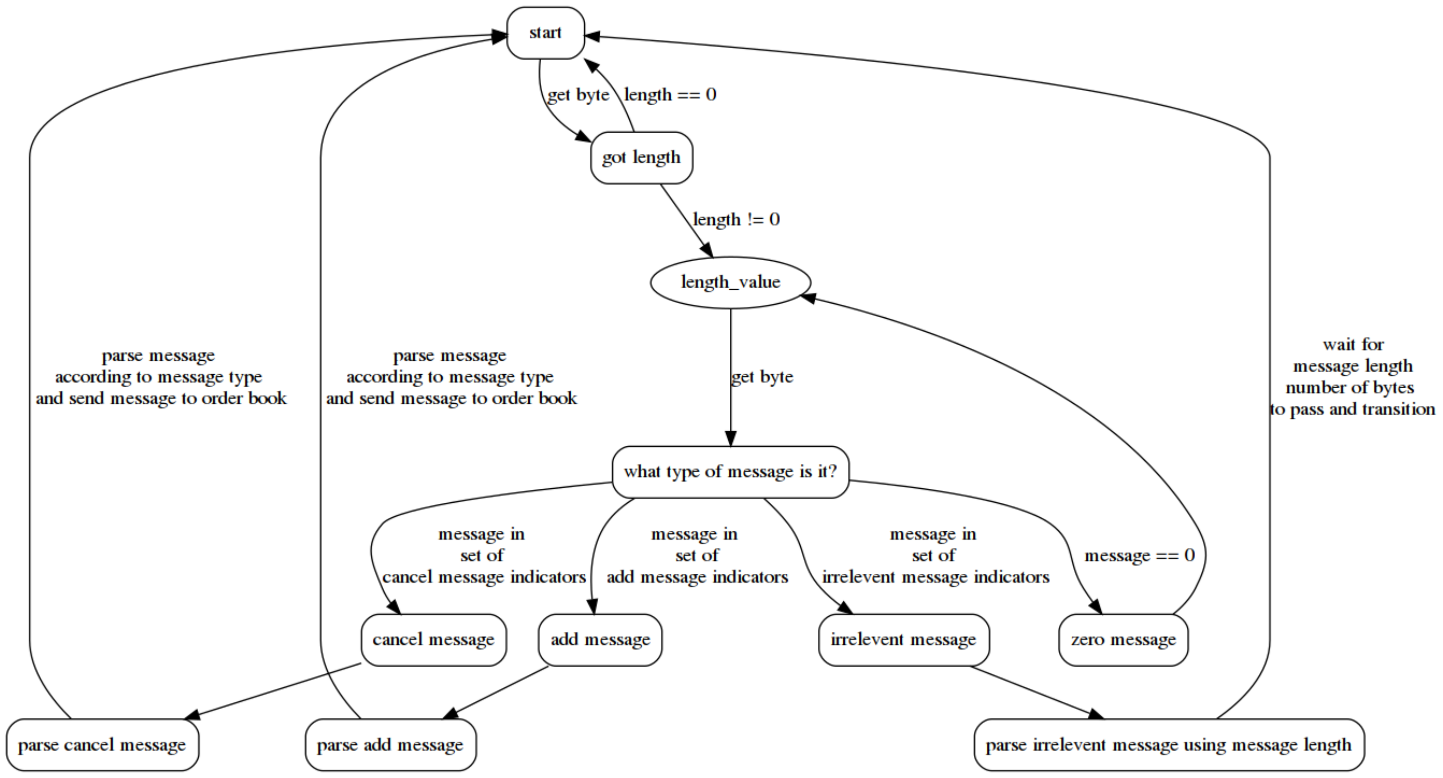
The parser only makes progress when it gets a valid value through AXIS protocol. If not, it keeps its current state until it does.

The parser enables its inner modules according to the message type it receives. It also passes in the message length to the ”irrelevant” message parser.

3.3.1 parser implementation

The parser is implemented as a giant state machine based on the FSM provided below:

1. parser FSM


start
get byte
length == 0
got length
length != 0
length_value
get byte
parse message according to message type and send message to order book
parse message according to message type and send message to order book
wait for message length number of bytes to pass and transition
what type of message is it?
message in set of cancel message indicators
message in set of add message indicators
message in set of irrelevent message indicators
message == 0
cancel message
add message
irrelevent message
zero message
parse cancel message
parse add message
parse irrelevent message using message length

2. trade-off between bit sizes and FSM complexity

   The advantage of structuring the data exchange at granularity of bits between the Ethernet stack and the parser is that the finer the granularity, the more opportunities to pipeline the system. However, when it comes to building the FSM that parses the data, matching the data feed specifications granularity allows for a much simpler FSM to be implemented. In the case of NASDAQ ITCH protocol, because the message type is structured in bytes, it would be natural to pick a granularity of a byte to use as the connection between the parser and the Ethernet stack. Some of the advantages are:

   (a) It simplifies the job of constructing a working parser because the FSM matches the structure of the protocol.

   (b) if structured at the granularity of a byte and more pipe-lining is required, the finite state machine for this can be implemented by using the byte based FSM as a sub module. This is still optimal on the Parsing side because of the natural structure of the data feed messages.

   All in all, the ease and generalizability of byte based protocol between the parser and Ethernet stack make it an ideal candidate for an initial implementation.

3. pertinent messages

   for building and maintaining an order book, the relevant messages are of type as follows.

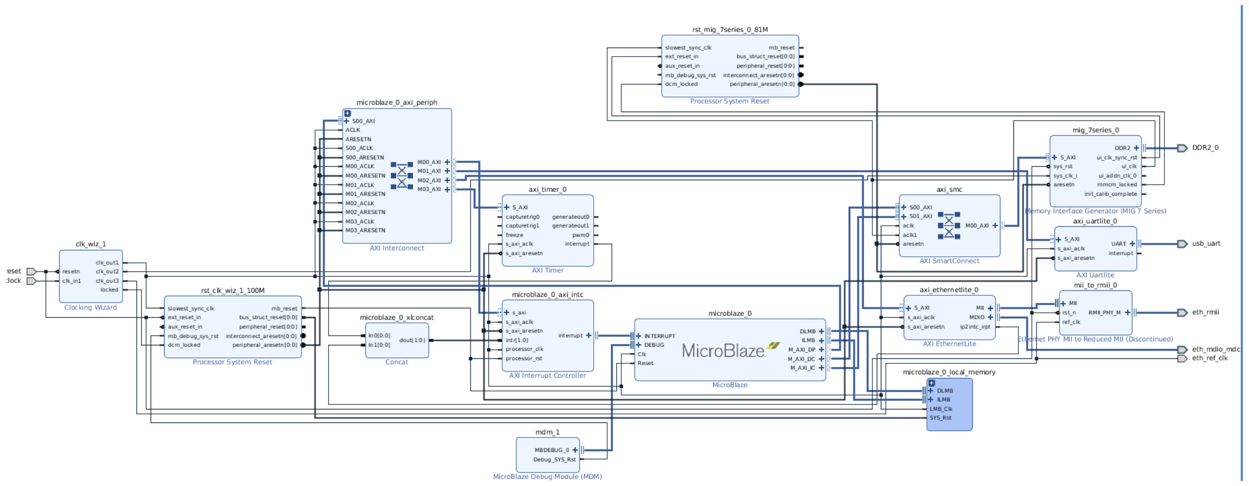
message catagory	type	message value
Add orders	add order	"A" = 0x41
Add orders	add order with mipid attribution	"F" = 0x46
Cancel orders	order executed	"E" = 0x45
Cancel orders	order executed with price message	"C" = 0x43
Cancel orders	order cancel	"X" = 0x58
Cancel orders	order delete	"D" = 0x44
Cancel orders	order replace	"U" = 0x55

   Therefore, the task of the parser is to pass these messages to the order book while filtering out every other message. In our implementation, we support all messages except `order replace`. `order replace` message changes the order id of a previous order, which is effectively a cancel and add order happening atomically. So, to a first order approximation, this can be implemented by replacing an `order replace` message with a consecutive `cancel` and `add` order messages on the parser side. The only issue with this approach is that the trading logic may pick up illegal intermediate state.

3.3.2 Ethernet implementation

The Ethernet stack is built upon the micro-blaze soft core IP. The attractive features of using micro blaze IP in the implementation is that it exposes a mature Ethernet stack that can be manipulated within Xilinux's hardware SDK. We developed a C implementation of the Ethernet stack on the micro-blaze to facilitate a TCP/IP network communication between the FPGA and exchanged. In our setup, the exchange is simulated by generating market orders from laptop that adhere to the ITCH protocol and communicate with the FPGA over Ethernet. We modified and wrote a good amount of python software to support this.

1. Microblaze IP

   Communicating between The Microblaze IP and other logic on The FPGA fabric requires the axi stream protocol. Therefore the parser is implmeneted to adhere to the axi stream protocol when receiving data.

   It is also perfectly possible for the data input path to be invalid for some number of cycles. Therefore the parser is implemented so that it doesn't make progress during such a time. The main observatiion to acheieve this is to only transition the FSM when we valid input that satsify the axi stream protocol.

2. AXI Stream protocol

   The AXI Stream protocol is the protocol the Microblaze exposes for communication with other FPGA logic. At its core, it consists of a valid, ready, data, last signal. It is a Master, slave protocol where data flows from the master to the slave.

   The protocol specification stipulates that both valid and ready must be high in the same clock cycle for data transfer to occur, i.e. the master to know that the slave is ready to receive it's data and for the slave to know that the master is sending it valid data.

   The Microblaze exposes at most 15 slave, master pairs of AXI Stream ports. It also exposes the FSL instruction set to send data to and from these AXI Stream ports.

3. FSL instruction set

   The FSL instruction set allows for communication over The Microblaze's AXI Stream ports. It is a standard that takes it name from a depreciated protocol Xilinx used to ship with instead of ARM's AXI stream (AXIS), but for compatibility reasons, Xilinux now included AXI Stream as the default choice. vestigially named instruction set has been extend to support AXI stream when Xilinux decised to adopt AXI as their de facto inter-IP communication.

   The FSL instruction set is exposed as a set of macros that can be placed within the set of instructions to be run on the Microblaze. It's canonical form takes in a piece of data to write or read from and an AXI port name. More advanced versions of the instruction set flags that set valid, control and/or error registers upon execution of instruction. It can also be configured to be blocking or non-blocking. The blocking type blocks until valid data can be transmitted into or from the FPGA fabric, where as the non-blocking type executes and sets a valid bit upon execution indicating that whether the data was sent or it read valid data.

   For our purposes, we need a connection that greedy attempts to send to and receive data from the exchange. This disqualifies the use of the blocking version for our final version.

   The non-blocking FSL instructions, `getfsl(var, id, FSL_NONBLOCKING)` and `setfsl(var, id, FSL_NONBLOCKING)` both set a value that specifies if they are able to execute or not while completing a valid AXIS communication. this can be checked using `fsl_isinvalid(invalid)` instruction that sets the variable invalid to 1 if the transaction is invalid. We make use of this instructions to implement the C side of the Microblaze.

4. interacting between microblaze and internal logic

   to interact between the internal logic and the Microblaze, the Microblaze exposes a few macros. These macros essentially take data from the Microblaze's internal memory and make it available over an AXI Stream port.

   There are also instruction to read from an axi stream port and write to the Microblaze's Internal memory.

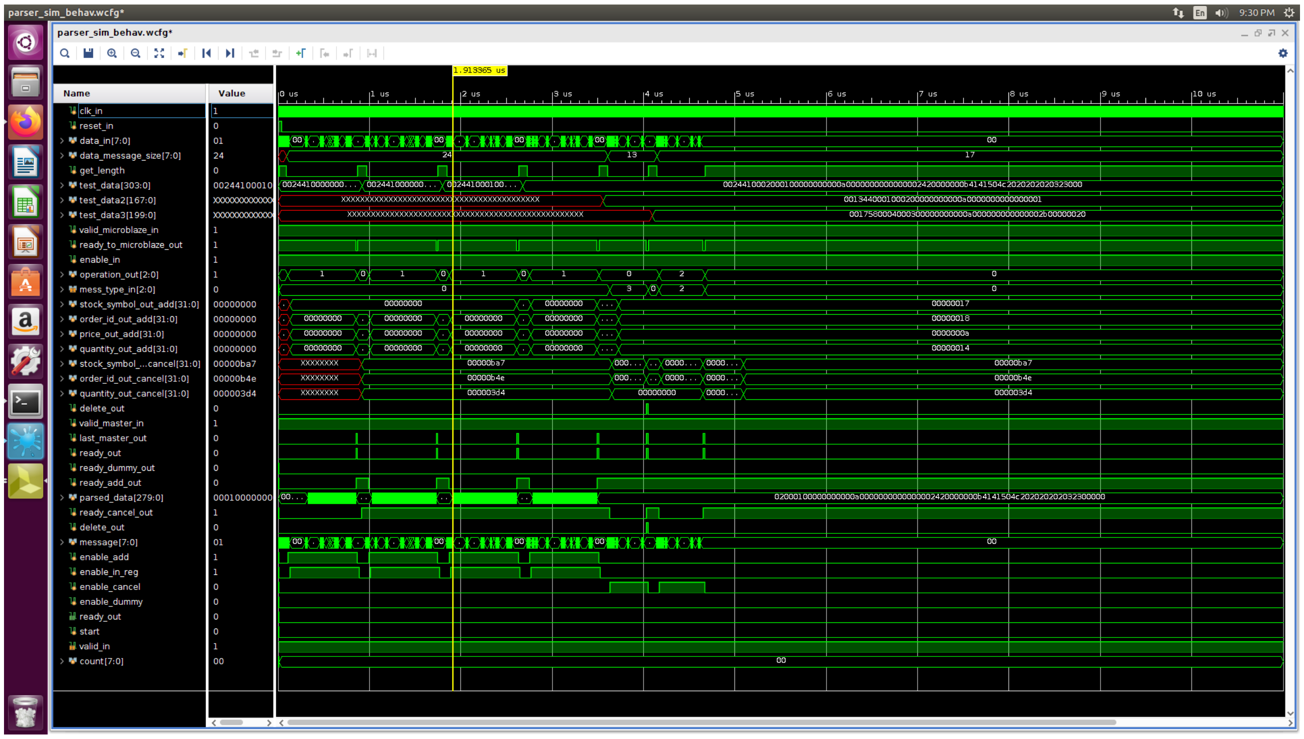
   These instructions come with blocking and non-blocking flavors that alter how the Microblaze should read the AXI stream ports. the blocking instructions wait until an external ready signal goes high while non-blocking instructions grab what ever is on the AXI ports output and set a flag if the value read also came with a valid external ready signal.

   This instructions as named `FSL instructions`, a mostly depreciated protocol replaced in favor for `AXI Stream`.

   In the case of the parser, it is getting data feed from the 2nd Master AXIS port while sending parsed output on the 2nd Slave AXIS port of the Microblaze.

3.3.3 parser Correctness:

We demoed input / output going through the parser through the FPGA and Microblaze Ethernet. Below is a simulation result where we have encoded multiple messages that satsify the ITCH protocol to be fed to the parser. We compared them to the python version that generated them to verify that they they were correct.



```
parser_sim_behav.wcfg*

Name                            Value
clk_in                          1
reset_in                        0
data_in[7:0]                    01
data_message_size[7:0]          24
get_length                      0
test_data[303:0]                00244100010
test_data2[167:0]               XXXXXXXXXXXX
test_data3[199:0]               XXXXXXXXXXXX
valid_microblaze_in             1
ready_to_microblaze_out         1
enable_in                       1
operation_out[2:0]              1
mess_type_in[2:0]               0
stock_symbol_out_add[31:0]      00000000
order_id_out_add[31:0]          00000000
price_out_add[31:0]             00000000
quantity_out_add[31:0]          00000000
stock_symbol_...cancel[31:0]    00000ba7
order_id_out_cancel[31:0]       00000b4e
quantity_out_cancel[31:0]       000003d4
delete_out                      0
valid_master_in                 1
last_master_out                 0
ready_out                       0
ready_dummy_out                 0
ready_add_out                   0
parsed_data[279:0]              00010000000
ready_cancel_out                1
delete_out                      0
message[7:0]                    01
enable_add                      1
enable_in_reg                   1
enable_cancel                   0
enable_dummy                    0
ready_out                       0
start                           0
valid_in                        1
count[7:0]                      00
```

3.4 Performance Benchmark

The current implementation of the Parser is very efficient. The Byte based Parser can optimally parse ITCH messages since messages are structured with Byte alignment in mind.

For a message of length N bytes, The parser takes $N + 1$ cycles to parse. The extra cycle comes from the fact that the current implementation of the parser doesn't bypass the ready signal for it's inner modules when they set their ready signal high.

During operation, this doesn't cause any inefficiency since the ITCH protocol reserves two bytes for the message length and according to specification, there is no message that currently requires two bytes to describe. So effectively, The first byte of the message can safely be ignored.

If a message length matching implementation of the parser is required, it is possible to modify the full stack to use a bypassed ready signal. This would require modifying the inner most ready_out signal like so:

```
always_comb begin
  ready_out = ready_comb ? ready_comb : ready_reg;
end
```

and subsequent upper modules like so:

```
always_comb begin
  case(select_between_inner)
     1: ready_out = ready_inner_1_out;
     default:  ready_out = 0;
  endcase
end
```

3.5 Circular buffer

Market data has an average data arrival rate that the parser plus book system have to match in order for correctness. If the parse plus book system don't have the required throughput to match the average data rate of the data feed, their is no possible way to use the FPGA to do this type of computation. In systems like this, what determines throughput is the book since not only can the parser be parallelized with little programming cost, it also can be structured to match cycle by cycle performance of the Ethernet stack as parsing what ever was transmitted over the Ethernet stack within the same cycle is within the clock speed of the machine.

At times though, Market data can and does arrive with a higher density of pertinent to irrelevant messages. During times where burst activity happens on the market, there needs to be a structure that handles packets so they are not dropped.

The initial implementation achieves this by essentially removing perturbations in the data feed rate that exceed the throughput of the book and has no mechanism to handle this phenomena. As a result, it is susceptible to dropping packets whenever the rate exceeds the throughput of the book.

To mitigate this problem, a data storage block needs to exists in either the micro blaze IP, which is used to serve as a connection point from the outside and the FPGA. Another attractive solution is to implement such data structure within the FPGA fabric.

1. design exploration for Circular Buffer

   (a) FPGA implementation

   the FPGA implementation consists of implementing a ring buffer in hardware. This means that it takes data in a places it in a well size-ed buffer. It then attempts to deque from this queue every cycle and send to the parser if the parser isn't stalling because of back pressure from the book.

   This implementation can also be utilized for sending back data over the Microblaze because the current implementation only sends out data when a packet arrives but if a message is ready to be send, it would have to wait in some memory location so it is not lost.

   (b) Microblaze implementation

   the micro blaze implementation consists of defining a buffer of some fixed length and implementing a circular buffer in C. This results in an implementation that is consists of an array with a head and tail pointer that are updated when new messages come in to be sent and old messages are sent, respectively.

3.6 lessons learned & Challenges

- Documentation: The Microblaze is quite difficult to work on because lack of proper documentation and a lot of experimentation was required to get the different components to work.
- integrate early: We should have tried to integrate earlier in the project with the other parts (Order Book and Trading Module) – we find the Microblaze tool very finicky to work with often not compiling properly when we added other IP modules, and the micro-blaze not at be as reliable in terms of sending messages over multiple cycles.

3.7 gotchas

microblaze FSL instructions

the consistency of the microblaze's `FSL instructions` is questionable. More than once have I seen a `getfslx` instruction have a valid value, when it shouldn't.

ILA and SDK

there is little documentation on how the hardware SDK should be set up so that newly generated bit streams can examined through ILA and be programmed through the SDK. This was very frustarting as for a while there was no way to observe internal signals when running, which made debugging staring at the void. The trick I found through experimentation is to make projects point to the newly generated bit streams.

AXI stream

The reset signals used for an AXI stream port ought to be active low. This constraint is there if working with AXI ports through the Block Design menu in Vivado.

4 Order Book Builder (Endrias)

4.1 Background

One of the most important aspects of a trading system is having an internal succinct representation of what is happening on the market: this data structure is typically referred to as an Order Book, and a per stock data structure. The order book has two sides: bid and ask. In this report, we will focus on the bid side of the book. The objective keeps track of the max price on the market as different market participants add new orders or cancel existing orders. There is three types of messages we receive from the exchange:
ADD_ORDER: the exchange specifies the order id, the price, and quantity of an order. The order book should accept this as a new order and appropriately update the size and best price on the market for that stock.

CANCEL_ORDER: the exchange specifies an order id, and the order should be removed from the market.

EXECUTE_TRADE: the exchange specifies an order id, and a quantity to decrease the order by.

In my code, I represent an order entry as follows:

```
parameter ADD_ORDER = 3'b001;
parameter CANCEL_ORDER = 3'b000;
parameter EXECUTE_ORDER = 3'b010;

typedef struct packed {
 logic [PRICE_INDEX:0] price;  // price for order to add
 logic [ORDER_INDEX:0] order_id; // order id
 logic [QUANTITY_INDEX:0] quantity; //quanttity
} book_entry;
```

4.2 Design

The high level design of the order book is as follows: there is a top level module order book wrapper, which is the interface that the order book exposes to the Ethernet stack.

4.2.1 Order Book wrapper:

```
order_book_wrapper(
        input clk_in,
        input rst_in,
        input [STOCK_INDEX:0] stock_to_add, //which stock to add
        input book_entry order_to_add, // what order to add
        input start,  //start signal
        input delete,  //whether to delete order from book
        input [QUANTITY_INDEX:0] quantity, //quantity to decrease order by
        input [2:0] request, // ADD_ORDER or CANCEL_ORDER or // EXECUTE_ORDER
        input [ORDER_INDEX:0] order_id, //should be supplied for a cancel / trade
        output logic is_busy, //high if th book is busy
        output logic best_price_valid, // high is best price is valid
        output logic [PRICE_INDEX:0] best_price_stocks  [0:NUM_STOCK_INDEX] // best price of //all the
        stocks ,
        output logic [0:NUM_STOCK_INDEX] best_prices_valid,
        output logic [SIZE_INDEX:0] size_of_stocks [0:NUM_STOCK_INDEX]
    );
```

This module is responsible for taking in a request to update the order book for a particular stock, and outputting the relevant market events (best prices of all the stocks, and the sizes) to the trading module. It also outputs a valid signal per stock to indicate if the best price of the stock is valid. The three operations supported are *CANCEL_ORDER*, *ADD_ORDER*, *EXECUTE_TRADE*. It outputs the current best price for all the stocks, and the size of the book for the different stocks. The protocol for communication is as follows: if the $order_book\_wrapper$ is not busy, and *start* signal is asserted, it will latch on the request that is incoming, and the signal is_busy goes high after a cycle. When the request is completed, the *is_busy* signal goes low. The reason we chose this protocol is because different operations in the order book have different latencies based on the input (the latencies will be discussed in detail) and this abstraction makes it easy to integrate with other modules.

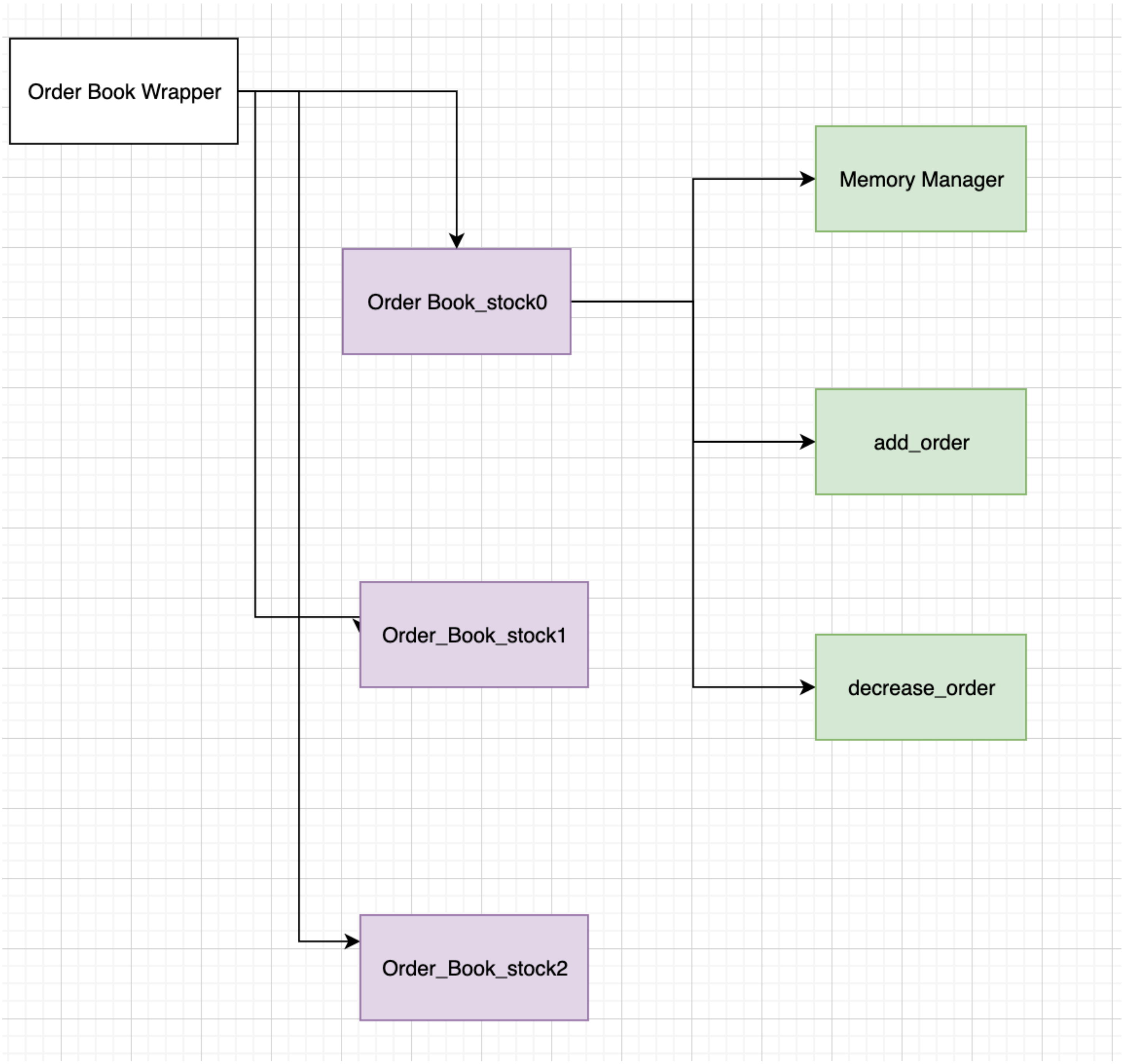


Figure 2: Schemtic of order book wrapper (with 3 stocks). Each each order book (there is one book per stock) has its own memory module, and add and decrease order modules.

One thing I found useful in implementing this module is to leverage ”generate statements” in verilog to parametrize the number of stocks the order book wrapper supports.

4.3 Basic Order book:

The order book at high level is implemented as follows: there is a BRAM that is managed by the memory module and is responsible for storing all the orders on the market. The add order, and decrease order module manipulate this BRAM through signals that are routed to the memory module through the order_book module, beging careful to not violate the property of the BRAM that there only one operation per port at any point in time.

```
order_book (
    input clk_in,
    input rst_in,
    input book_entry order_to_add,
    input start_book,
    input delete,
    input [2:0] request,
    input [ORDER_INDEX:0] order_id, //should be supplied for a cancel / trade
    input [QUANTITY_INDEX:0] quantity,  // should be supplied for a trade
    output logic is_busy_o,
    output logic [CANCEL_UPDATE_INDEX:0] cancel_update,
    output logic [PRICE_INDEX:0] best_price_o,
    output logic best_price_valid,
    output logic [SIZE_INDEX:0] size_book
);
```

The order book module is responsible for maintaining the best price of a particular stock. It also follows the ready-input protocol. Cancel and Execute Trade are implemented using the decrease module, which can decrease the quantity of an order or delete it. Adding an order is done by the add order module. Internally, it starts the add order or decrease order module based on the request, and has a mux to combinatonally holds the signal outputs from add order or decrease order module to feed into the memory module (based on the request). One tricky aspect is both add order and cancel order modify the signals coming into the memory module, so care has to be taken to make sure only one of them is able to feed their output to the memory module.

4.3.1 Add order:

```
 add_order#(parameter IS_MAX=MAX)
              (input clk_in, input book_entry order, input start, input valid,
               input   [QUANTITY_INDEX:0] price_distr ,
               input [SIZE_INDEX:0] size, input [PRICE_INDEX:0] best_price, input price_valid,
               output logic [ADDRESS_INDEX:0] addr,  output logic mem_start,
               output book_entry data_w,
               output logic price_update,
               output logic quantity_update,
               output quantity,
               output logic is_write, output logic ready, output logic [SIZE_INDEX:0] size_update_o,
               output logic  [PRICE_INDEX:0] add_best_price )
```

The add order module is responsible for adding a new order from the market the order book. It implements the ready-start protocol. When it gets a request, it checks if there is sufficient space on the BRAM to store the order. If there is, then the proper memory signals are asserted to commit the order to the BRAM. It waits until the signal that indicates that BRAM write has been committed is high before indicating that it is no longer busy. It appropriately updates the new size of the book, and the best price of the stock, taking into account if there was a valid best price before.

4.3.2 Decrease Order:

```
      decrease_order#(parameter SIDE=BUY_SIDE)
              (input clk_in,
               input logic [ORDER_INDEX:0] id,
               input [QUANTITY_INDEX:0] quantity,
               input [PRICE_INDEX:0] best_price,
               input delete,
               input mem_valid,
               input [SIZE_INDEX:0] size,
               input start,
               input read_result data_r,
               output  mem_struct mem_control,
               output book_entry data_w,
               output logic ready,
               output logic [SIZE_INDEX:0] size_update_o,
               output logic[CANCEL_UPDATE_INDEX:0] update
    );
```

The decrease order module is responsible for canceling an order / or reducing the quantity associated on the market. It takes as input the the order id of the order to modify, and the quantity to modify it by it. If instead the order needs to be deleted (remove all the quantity) to the order, the delete signal must be set high. It outputs signals to controls the memory module which are routed by the order book module.

The high level implementation is as follows: there is a state machine that scans the BRAM from 0 to the size of the current book, looking for the order ID. Because the BRAM has parametrizable latency, care is taken to wait until the data from BRAM is valid as we can scan through it. If we don't find the order, then the order must not have been stored in the BRAM, so no updates are necessary.

If the order is found, we update the quantity into the BRAM and see if its still non zero (if the request was to instead to delete the order, we don't have to check the quantity). If these condition is satsified, then we need to delete it from BRAM. We have two choices: we can incorporate a valid bit with each entry into the BRAM, and just mark that valid bit to 0. The other choice is move the elements that are part of the book but come after the entry we want so that the gap left by the deleted element is filled. This approach turns out to incurs more latency for delete, but it has nice properties that give a faster *add_order* implementation, so I went with the approach. I will discuss this in more detail in the latency section.

To delete an order after we identify its index in the BRAM (let say index i), there is an intricate state machine that grabs the element at index $i + 1$, moves it to index i, and repeats this operation for index $i + 1$ until it reach the current size of the book (we don't have to scan the entire BRAM, just the parts that are valid). The complexity of this comes from trying to minimize the number of cycles we have to wait for each BRAM write /read while maintaining correctness as we iterate through the BRAM.

4.4 Correctness

Correctness: I rigorously tested correctness by simulating different sequences events into the order book (I had over 13 simulation tests), and compared the state of the order book to a python script that I wrote that computes the order book in python and displays the state.

In simulation, I was able to see the entire state of BRAM by clicking on the BRAM memory unit and opening mem_unit, which shows the memory stored as registers. On the actual FPGA, I used ILA to make sure everything was working. To inspect the BRAM, I used a module to sweep the address space and see the memory (I piggybacked on the memory module I wrote).

4.5 Performance Analysis:

The whole point of building HFT using FPGA is to get low latency. Since all data coming from the exchange goes through the parser, and then to the order book, its crucial for the order book to be low latency. To analyze the latency of "*memory_manger* module, *order_book* module, we first analyze the latency of the *add_order* and *decrease_order* module.

4.5.1 memory manager Latency:

Once a request to the BRAM is sent, we have to wait $BRAM\_LATENCY$ before we are guaranteed that the result has been written / or retrieved from BRAM. Thus, the latency of this module is $BRAM_LATENCY$.

4.5.2 Add Order Latency:

The latency after the start signal is set to high corresponds to writing the new entry of the order book to the end of BRAM, and waiting until it gets committed + the cost of state machine transitions in the memory manager. Because there is a cycle delay before start signal is set (an inherent issue with the ready start protocol), there is one cycle wasted.

4.5.3 Delete Latency:

To analyze the latency of this module, we consider the different states of the FSM at a high level. The first state of the FSM corresponds to finding the location of the order in the book. Lets assume the location of the element is at location X in the BRAM. It will take us $C * X$ cycles to find the element in the BRAM, where C is a constant related to $BRAM\_LATENCY$ and cost of internal state transitions in the FSM. Once we find the element, we have to shift all the elements that come after it one by one. This takes $C * (N - X)$ cycles, where N is the size of the order book. Summing up, we find the number of cycles total is $C * N$, which is proportional to the size of the order book.

4.5.4 Overall Analysis:

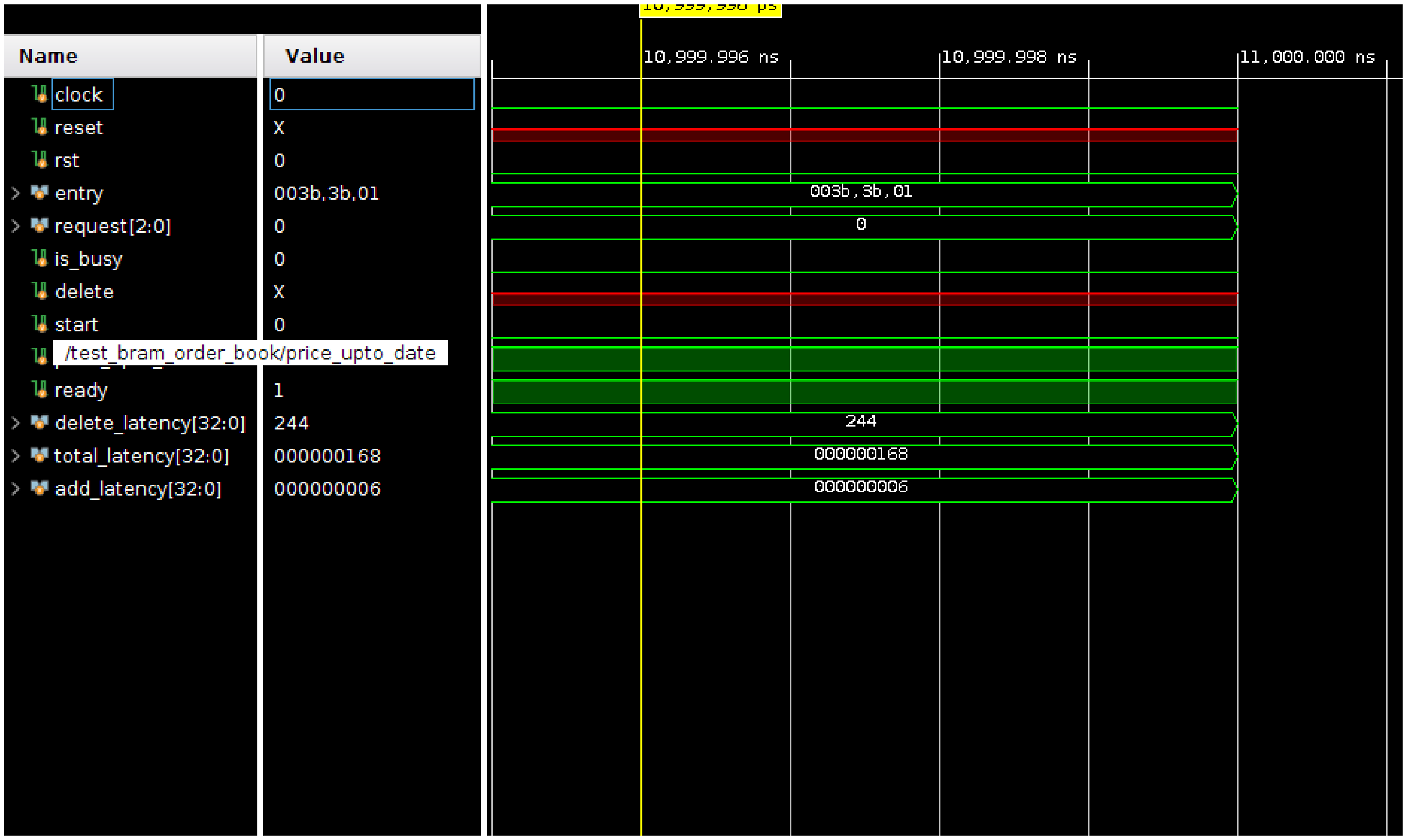
From the above analysis, we can conclude that adding orders takes constant number of cycles regardless of the number of orders. This is quite desirable for an HFT system as it guarantees constant and low latency for adding orders. The throughout of the system is 1/latency, as only one operation can proceed at a time (because all operations access the BRAM). However, deleting an order is not constant. If there is a lot of open orders on the market, the performance deteriorates. This is not desirable, and in the next section, we describe an elegant order book that can achieve adding and deleting orders with O($\log_2 n$) latency, and O(1) throughput.

4.5.5 Hardware usage:

When the number of entries in the BRAM is set to 200,000 (which means we can support 200,000 orders at the same time, the resource usage was as follows; 13.7% block rom, and 1.43% LUT. We can increase the resource usage by increasing the number of bits used for storing an order, or creating a larger BRAM tahn can support more orders, but overall the BRAM module is not heavy in terms of space usage.

4.6 Empirical Performance:

To measure the empirical performance, I wrote a test that submitted 50 orders to the market, with increasing prices, and then canceled the last order. I computed the latency of the add module and decrease module by counting how many cycles the busy signal is high when these operations are in progress.



Name	Value
clock	0
reset	X
rst	0
entry	003b,3b,01
request[2:0]	0
is_busy	0
delete	X
start	0
/test_bram_order_book/price_upto_date	
ready	1
delete_latency[32:0]	244
total_latency[32:0]	000000168
add_latency[32:0]	000000006

Figure 3: Basic Order Book Results.

We can see that the latency of an add is small; 6 cycles. However, the latency of deleting an order is 244 cycles, because we have to scan a BRAM that has 50 active orders. This latency is not predictable and depends on how many active orders on the market. For an HFT company, its critical to have low and deterministic latency. Thus, in the next section, we describe an implementation that is able to circumvent this problem.

4.7 Tree OrderBook (Stretch goal)

: An interesting observation about the range of prices in a stock market throughout the day is that they fall in a very right range: its uncommon for a stock to increase by more than a few dollars each day. In fact, the whole point of HFT is to keep the price range very tight around its ”true” value. Further, each stock has an associated ticker. The stock can only have prices that are multiples of that ticker. For example, if the ticker is 0.1, then the stock can only have prices 0.1, 0.2 , 0.3 , ...1 , 1.1 ... We can take advantage of this observation to build an optimized data structure for the order book. A similar approach was taken by a paper that I saw but couldn't find again.

The high level design is as follows:
Lets assume the price of the stock lies between [X, Y].

Have a mapping module that maps prices to indexes. This would scale the price by subtracting the price range of that stock, and scaling it by the ticker value.

Maintain a bank of price registers (say P_Q), which store the aggregate quantity for each price.

Then build a reducer tree like circuit on top of price registers to compute the best price for the stock. In terms of notation, lets say the first level of the tree is "level" 0, second "level" 1, etc. Then the number of nodes at each level is 2^{level}. The tree has $\log_2 n$ height, where n is the number of distinct prices (the number leaves of the trees). The number of distinct prices is (Y - X)/ticker.

On the side, a BRAM that is indexed by the order ID is stored.

4.7.1 Implementing the tree:

The tree can be implemented in Verilog by representing the different layers of the tree as contagious registers that map prices to quantities, and connecting level i with level $i+1$ of the tree. We can use the identify that the node at level i and location j is connected to node $2*j$ and $2*j+1$ from level $i+1$, and because we want the maximum price, if the the node at $2*j+1$ has non zero quantity, the node at level i takes it value. Otherwise, it takes the value of the $2*j$ node at level $i+1$. The last layer of the tree is connected to the bank of price registers (P_Q) in a similar way.

The best price will be stored at level 0. There is some interesting possibilities in how to feed values from level i to level $i+1$: should it be done combinatiaonally or be clocked? The issue with making it combational is the circuit will violate timing since the maximum propagation delay in the circuit is related to $\log_2 n$*$TPDofRegister$. So if we have a lot of levels, the circuit can't update itself in one clock cycle. Pipeling this would require clocks. Pipe-ling would increase the latency of getting an update to $\log_2 n$ cycles. We will discuss in the performance section ways to mitigate that.

4.7.2 Adding an order:

Adding an order corresponds indexing the BRAM, and storing the order. The mapper is used to identify which price register to update. That price register is updated with the appropriate quantity. The interesting observation.

4.7.3 Canceling / reducing the quantity of an order:

Canceling an order corresponds retrieving the quantity from BRAM by indexing it using the order ID, and update the quantity for that price in P_M.

4.8 Latency Analysis:

Add and cancel operations have to update / query one location in BRAM, so they incur $BRAM\_LATENCY$, plus one update to the register that holds the quantity of the price that is being updated. Thus the have $O(1/BRAM\_LATENCY)$ throughput. Assuming the different layers of the tree are pipelined in terms of updates, we incur $\log_2 n + BRAM\_LATENCY$ clocks in latency for the best price to change. Note that however, the throughout doesn't depend on the the height of the price tree, which is desirable for high scaliability.

4.9 Implementation

We prototype this fast order book implementation. We stored the order ids in an array of registers, and used an identify mapper as a slight simplification. Every cycle, the data from the level i of the tree is propagated to level i-1 using always ff blocks. We tested with 6 levels for prices (so 64 distinct prices, although this is parameterizable).

4.10 Performance

We re-rerun the performance test we used for the order book implementation given above. The test constituted of 59 adds to the market with different prices, and one cancel operation. 6 levels are used for the tree, to support 64 distinct prices.

4.11 Parametrizing everything

Since I wanted to build an extensible system for the order book, I defined a constants.sv folder which fully parameterized the order in terms of the size of the order id, the price, the quantity, and the number of total entries in the order book, etc. This make integration with the trading

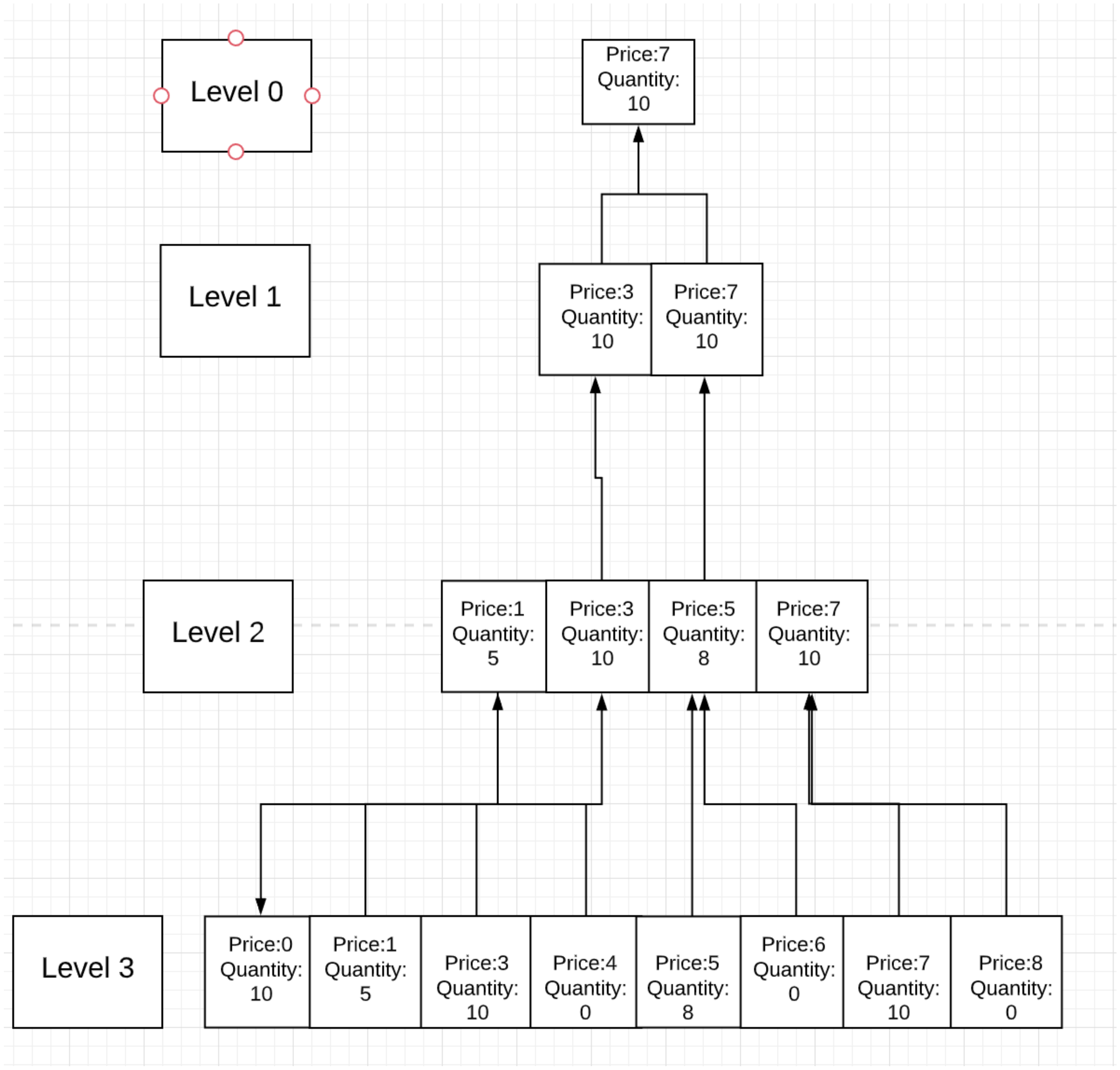
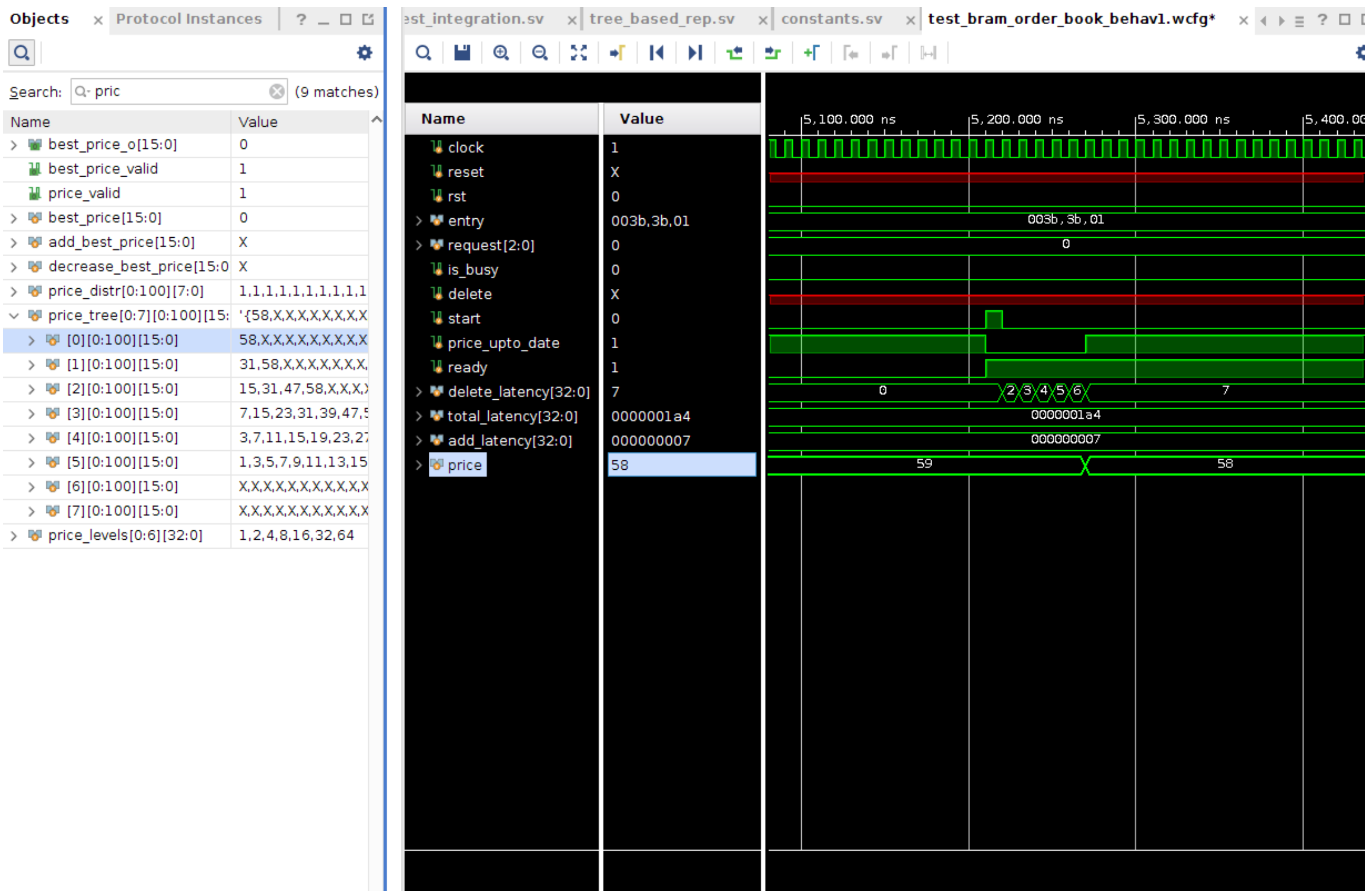


Figure 4: A depiction of the tree. Each node is associated with a price and quantity. The leaves have the aggregate quantity for each price. The nodes above them take the value of the right node if its quantity is non zero, otherwise it takes the price of the left node. This encodes the relationship that "right" child has a higher price, and the non zero quantity tells us if the price is valid. (for example node 0 at level 3 takes the value of its right child, while node 1 at level 3 takes the value of its left child

4.12 Tricky aspect

4.12.1 Building the right level of abstraction

: One major issue of headache was interfacing with the same BRAM as quickly as possible while respecting a parameterizable delay for the BRAM, and not violating that only read / write request will be satisfied per port. I had a lot of error prone code I was writing, and I took a step back and I created a module that managed all accesses to the BRAM which simplified a lot of my code.



Name	Value
best_price_o[15:0]	0
best_price_valid	1
price_valid	1
best_price[15:0]	0
add_best_price[15:0]	X
decrease_best_price[15:0	X
price_distr[0:100][7:0]	1,1,1,1,1,1,1,1,1,1
price_tree[0:7][0:100][15:	'{58,X,X,X,X,X,X,X,X
[0][0:100][15:0]	58,X,X,X,X,X,X,X,X,X
[1][0:100][15:0]	31,58,X,X,X,X,X,X,X,
[2][0:100][15:0]	15,31,47,58,X,X,X,X
[3][0:100][15:0]	7,15,23,31,39,47,5
[4][0:100][15:0]	3,7,11,15,19,23,27
[5][0:100][15:0]	1,3,5,7,9,11,13,15
[6][0:100][15:0]	X,X,X,X,X,X,X,X,X,X,X
[7][0:100][15:0]	X,X,X,X,X,X,X,X,X,X,X
price_levels[0:6][32:0]	1,2,4,8,16,32,64

Name	Value
clock	1
reset	X
rst	0
entry	003b,3b,01
request[2:0]	0
is_busy	0
delete	X
start	0
price_upto_date	1
ready	1
delete_latency[32:0]	7
total_latency[32:0]	0000001a4
add_latency[32:0]	000000007
price	58

Figure 5: Performance results for tree represnetation. The signal *price_upto_date* goes high when the propagation delay of the tree has passed. We note that We can see that the latency of both delete and add is 7 cycles, which is very close to the predicted latency (which is the height of the tree or 6). This is a massive speedup compared to the order book described in the previous section.

4.12.2 Wiring the Tree in verilog

: I started with wiring the tree combinationally, but run into timing violations,and was forced to switch to a cloked version. It was also difficult to get the exact schematics of the tree right because it required a 3d array which I haven't used in verilog.

4.13 Learning Aspect Advice

4.13.1 Getting pigeonholed in optimization

: I thought a lot about performance as I was designing the system, which makes sense for this project. But I also obsessed over details instead of focusing on getting a working implementation. Joe advise regarding this was very helpful.

4.13.2 Think FGPA

As someone will a lot of *C* background, particularly in performance engineering, I tried to port the data structures I would use for this problem in C. I realized that they didn't port well because the abstraction and target hardware is radically different. My final iteartion of the order book was made after thinking about what design of order book is well suited for the FPGA's parallelism.

4.13.3 Look at Warning

Multiple times, I didn't bother looking at warnings and spent time debugging issues that could have been easily resolved by a close insepction of the warnnings.

5 Trading Logic (Ziheng)

5.1 Background

For our trading system, we are going to try to maintain a minimum variance portfolio, where market risk is minimized. This is a classic trading strategy. Computing the necessary target positions for the stocks in this portfolio is important for an HFT firm because it offers valuable information about the risk of its current positions, and how to adjust them to minimize risk. This should be done as quickly as possible, since this might indicate an abrupt change in market structure that could evolve rapidly. A primer on this can be found in this reference [5].

We build a system on FPGA to generate these target positions from price updates. Not only is the latency low (compared to same logic on CPU), it is deterministic, another coveted characteristic for HFT firms.

The key to the trading logic is the covariance matrix of returns of the stocks. Given a covariance matrix K, the target minimum variance portfolio can be defined by the position weights w, where:

$$w = \frac{K^{-1}\mathbf{1}}{(K^{-1}\mathbf{1})^T\mathbf{1}} \tag{1}$$

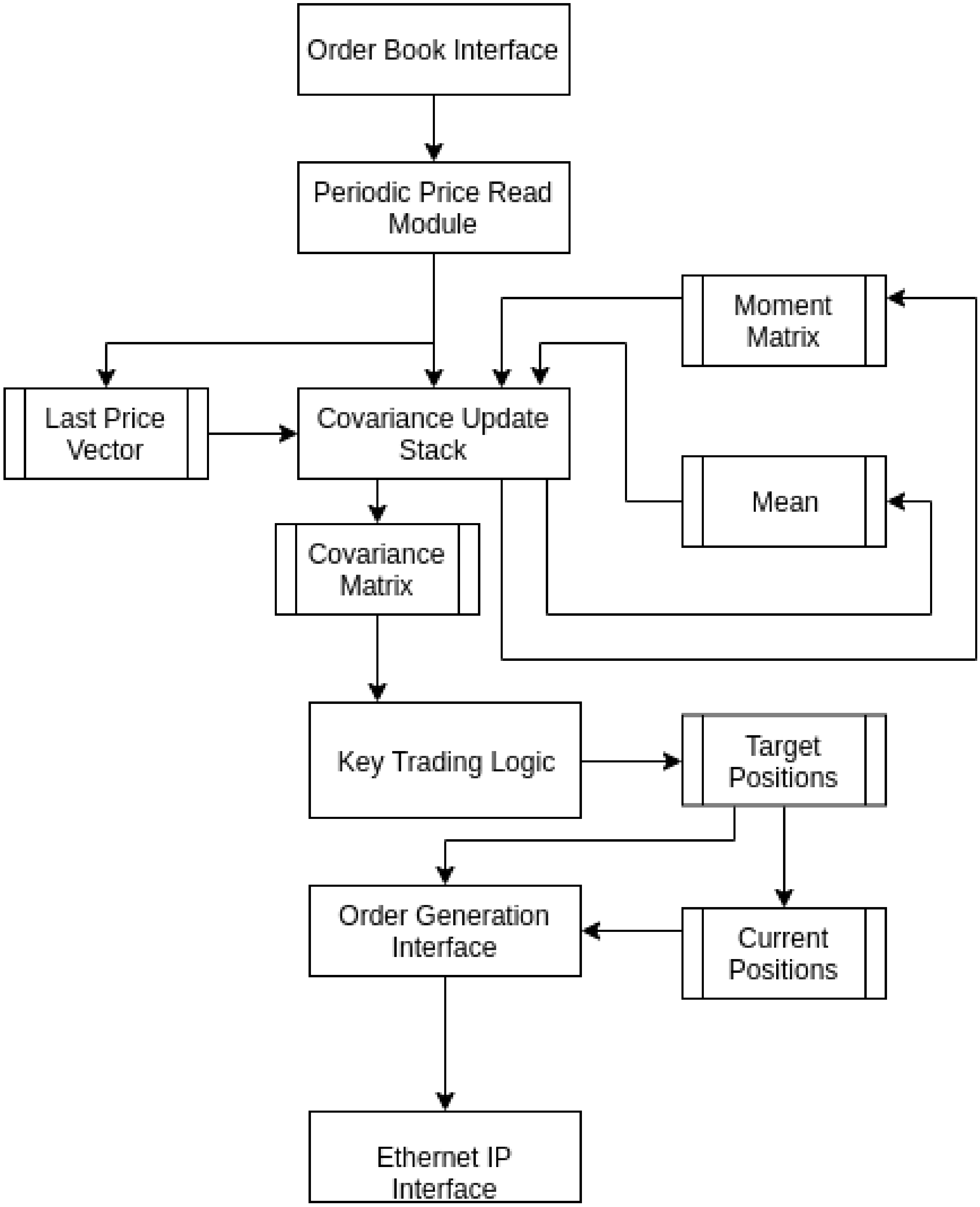


Figure 6: Block diagram for overall trading logic module and how it would interface with the Ethernet IP. In the integration test with the order book, we just display the latest prices and target positions using the hex display.

The overall system diagram for the trading logic is shown in Figure 6.

We first need to estimate the covariance matrix K from market data, given from the book building module. The formula for the covariance matrix, K, is defined as:

$$K = \mathbb{E}[XX^T] - \mathbb{E}[X]\mathbb{E}[X]^T \tag{2}$$

X is a random vector (vector of random variables), where element i corresponds to the return of stock i. X has to be estimated from market data. In our integrated system, we make the measurement every time we press a button, indicating that a new price is available for computation: for every stock, we take the current price p, the price a millisecond ago q, and calculate the return r as below:

$$r = (q - p)/p - 1 \tag{3}$$

We can thus obtain an observation of the random vector. We can use those observations to compute the expectations required for an estimate of the covariance matrix. In real life, this measurement would be made at a fixed time interval, say every second. (Any reasonable update interval used in industry should be well above the latency of our system, which is only 400 microseconds. Our system meets design spec in this sense.)

The overall system diagram is displayed below. A module periodically reads from a bank of registers maintained by the order book builder module to get the latest price. Then we update the covariance matrix estimate, which feeds into the trading logic module. After the target positions are computed, we generate the necessary information to form outgoing order messages in the OUCH protocol, handled by the Ethernet IP stack.

Now let's focus on how to update the covariance matrix from observations of the stock returns. Let's use superscript, K^N to denote the estimate of K at time step N.

$$K^N = \mathbb{E}^N[XX^T] - \mathbb{E}^N[X]\mathbb{E}^N[X]^T \tag{4}$$

If we denote the observation of the random vector X at time i as x_i , then we have the following formula for the two expectation terms in the above equation.

$$\mathbb{E}^N[XX^T] = \frac{1}{N}\Sigma_{i=1}^N(x_i x_i^T) \tag{5}$$

$$\mathbb{E}^N[X]\mathbb{E}^N[X]^T = \frac{1}{N}(\Sigma_{i=1}^N x_i)\frac{1}{N}(\Sigma_{i=1}^N x_i^T) \tag{6}$$

Now the key question is how does K change as we get a new observation. This computation needs to be done as quickly and resource-efficiently as possible. Let's start with the formula for the new K. We have to update the two expectation terms with the new observation. The first is called the second moment (which we just call the moment matrix), and the second is an outer product of the first moments (i.e. mean).

$$K^{N+1} = \mathbb{E}^{N+1}[XX^T] - \mathbb{E}^{N+1}[X]\mathbb{E}^{N+1}[X]^T \tag{7}$$

After looking at the formula for the first expectation above, we can see that given the new observation x_{N+1}, the new moment matrix can be calculated as follows:

$$\mathbb{E}^{N+1}[XX^T] = \frac{N \times \mathbb{E}^N[XX^T] + x_{N+1}x_{N+1}^T}{N+1} \tag{8}$$

As for the outer product of the mean, instead of trying to calculate the difference with the previous term in K^N, it's more efficient to just update the mean:

$$\mathbb{E}^{N+1}[X] = \frac{N}{N+1}\mathbb{E}^N[X] + \frac{x_{N+1}}{N+1} \tag{9}$$

Then we do an outer product between $\mathbb{E}^{N+1}[X]$ to get what we need in the second term of Equation 7. This means that at each update step N, we have to save $\mathbb{E}^N[XX^T]$ and $\mathbb{E}^N[X]$ for computation at the next time step.

5.2 Updating the Covariance Matrix on FPGA

We need to compute two sets of outer products, one in Equation 8 and another in Equation 9, as well as numerous other mathematical expressions, such as vector multiply and accumulate and matrix addition. We accomplish that through heavily using DSPs. In the design, I write a matrix arithmetic module to instantiate a grid of DSPs and exposes matrix functions, such as matrix addition and outer product. I also write vector arithmetic modules that expose vector division and vector multiply and accumulate functionality.

To trade off parallelism and resource usage, we divide the computations listed above in sub-computations, which can be topologically sorted based on dependencies (i.e. sub-computations that need to occur before others need to have precedence in the topological order). We can then use the topological sort to divide the computation into stages, where the subcomputations within a stage can be scheduled in parallel. We insert pipeline registers of appropriate size between the stages to reduce combinational path and to potentially reuse some of those registers. For example, the returns vector is reused for different stages. However, some of those pipelining registers are quite large (as large as the matrix for example), which can result in substantial flip-flop overhead. The overall system diagram is shown in Figure 7.

We can easily reason about the latency of each stage based on what subcomputations occur in this stage. For example, stage 0 will take as long as the vector division. Stage 2's runtime will be the maximum of the runtimes of vector multiply-accumulate, matrix outer product and matrix element-wise multiplication. In this design, stage 0 and stage 1 will take $D + N - 1$ cycles, assuming the vector division is performed by pipelining a single divider. Stage 2, 3 and 4 will each take two cycles in my current DSP setup. Since stage inputs and outputs are registered, we will introduce another 4 cycles latency. In total, it should take $2D + 2N + 8$ cycles to finish the computation. Since $N = 4$, it should take $2D + 16$ cycles. In practice, the observed latency is around 70 cycles when D is set to 20. The difference is attributed to some of my state transitioning logic.

We can also easily reason about resource utilization, assuming that different stages can reuse resources. The total amount of resources required is $2N^2 + N = 36$ DSPs for stage 2,3,4 and N dividers for stage 1. The matrix element-wise multiply, subtraction and addition can all use the same grid of DSPs, with different instructions. This estimate is roughly correct. Vivado decides to use 48 DSPs for this pipeline. The extra DSPs were used to calculate some overflow information. (Even though the DSP manual shows that there is an overflow port, there is no such port in the Macro. As such, I manually compute the overflow.)

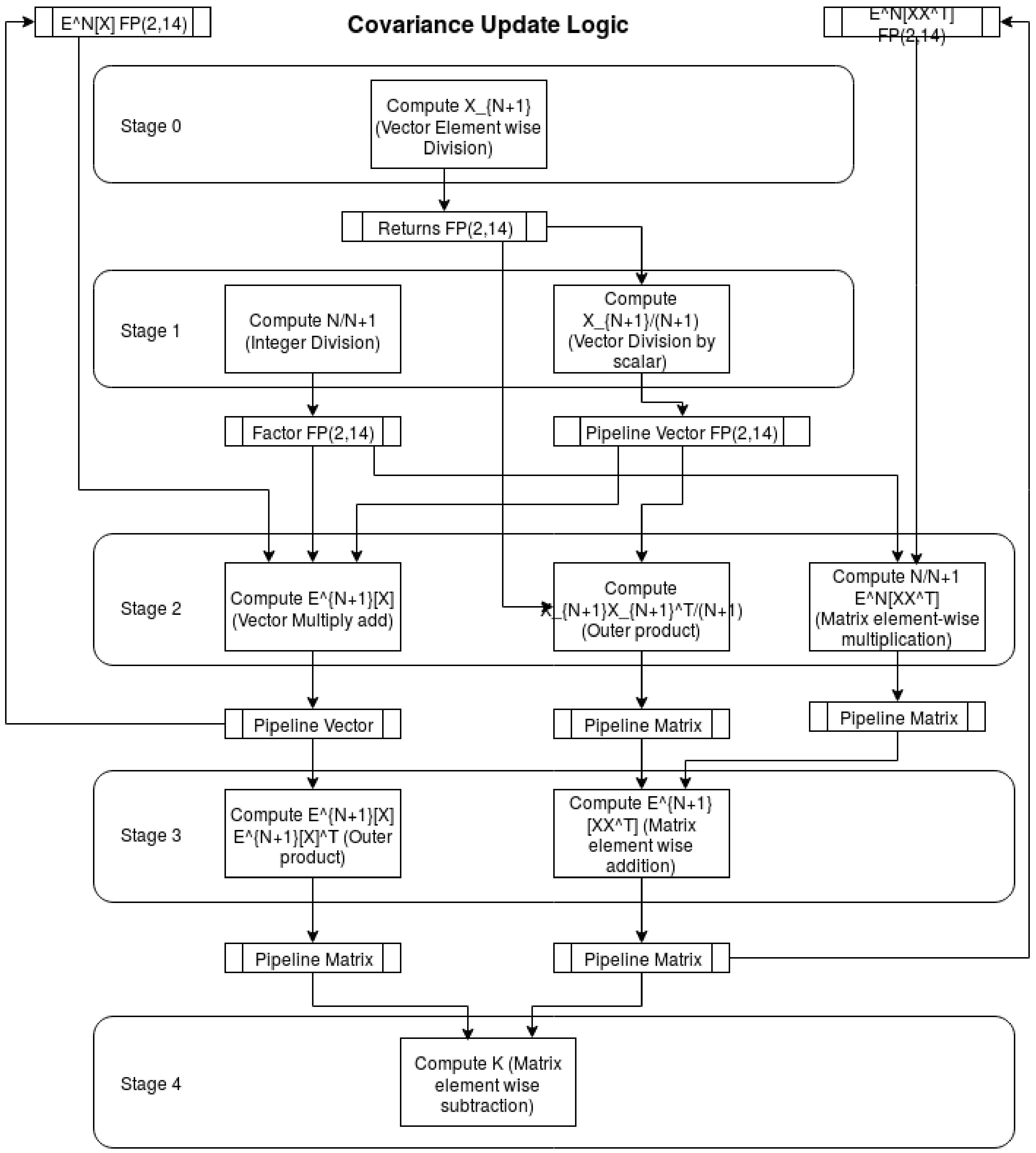

Covariance Update Logic
E^N[X] FP(2,14)
E^N[XX^T] FP(2,14)
Stage 0
Compute X_{N+1} (Vector Element wise Division)
Returns FP(2,14)
Stage 1
Compute N/N+1 (Integer Division)
Compute X_{N+1}/(N+1) (Vector Division by scalar)
Factor FP(2,14)
Pipeline Vector FP(2,14)
Stage 2
Compute E^{N+1}[X] (Vector Multiply add)
Compute X_{N+1}X_{N+1}^T/(N+1) (Outer product)
Compute N/N+1 E^N[XX^T] (Matrix element-wise multiplication)
Pipeline Vector
Pipeline Matrix
Pipeline Matrix
Stage 3
Compute E^{N+1}[X] E^{N+1}[X]^T (Outer product)
Compute E^{N+1} [XX^T] (Matrix element wise addition)
Pipeline Matrix
Pipeline Matrix
Stage 4
Compute K (Matrix element wise subtraction)

Figure 7: Block diagram for the module to update the covariance matrix. Pipelined registers and registered inputs are also shown. The computation is broken into stages, where multiple subcomputations can occur in parallel at each stage.

5.3 Design Choices: Folded vs Combinational vs Pipelined

Here I will briefly take a digression into my design choice. There are typically two designs available for a stage-based FSM: folded and pipelined. In a folded design a single circuit is generated which is reused by different stages. In a pipelined design different circuits are generated for different stages with optional pipelined registers in between. Folded designs are evidently much smaller than pipelined designs. However pipelined designs can achieve higher throughput as it can process a new input every cycle. However, the latency of a single input, which is what we care about here, is the same. We do not need to pipeline our trading logic. There's a small caveat that you might be able to clock the system faster due to logic simplification in the pipelined design since each stage has specialized circuit. However, considering the resource limitations on the Nexys 4, we choose a folded design philosophy here.

5.4 How to get K^{-1}

Now that we have K, we are going to have to compute this quantity:

$$w = \frac{K^{-1}\mathbf{1}}{(K^{-1}\mathbf{1})^T\mathbf{1}} \tag{10}$$

Notice that we don't actually have to invert the matrix. We just have to solve for the linear system $K^{-1}\mathbf{1}$. After we have the solution vector, we just normalize it so that all the elements sum up to 1. Typically, in cases like this, we have to perform QR decomposition to improve the numerical stability of the linear solver. This is more important on FPGA since we are using fixed point representations, leaving more room for error.

Suppose $v = K^{-1}\mathbf{1}$, i.e. $Kv = \mathbf{1}$. Then if we decompose K into QR, (where Q is an orthogonal matrix and R is an upper triangular matrix) we have:

$$QRv = \mathbf{1}, Rv = Q^{-1}\mathbf{1} \tag{11}$$

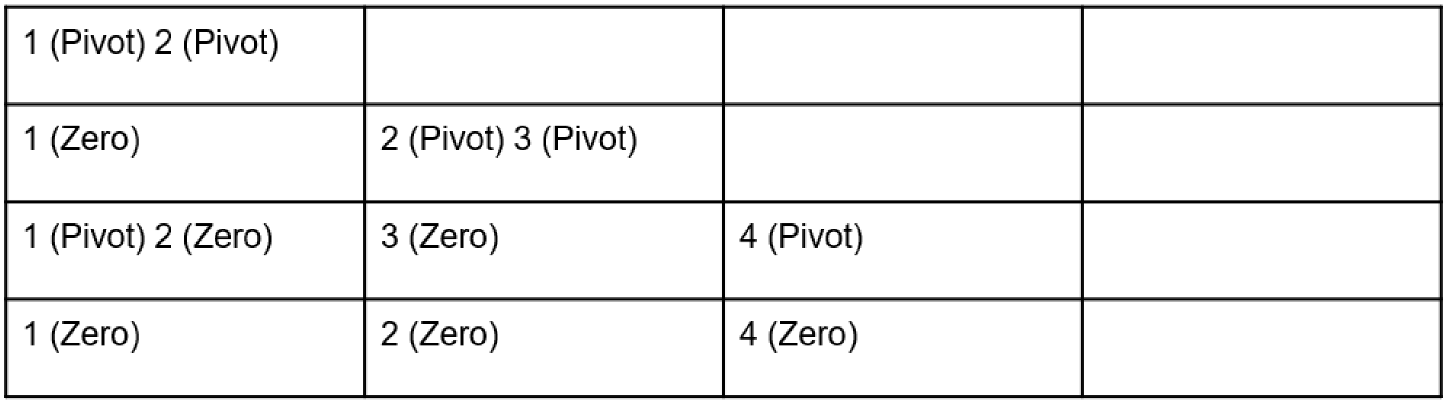
Note that $R = Q^{-1}K$. Thus we would like to apply Q^{-1} to both K and the $\mathbf{1}$ vector. Then we would like to solve the linear system $Rx = Q^{-1}\mathbf{1}$ with upper triangular matrix R and vector $Q^{-1}\mathbf{1}$.

There are three algorithms to do QR decomposition. The first is Gram Schmidt, suitable for sequential processors or FPGAs with a lot of hardended DSPs [3]. On FPGAs with constrained DSP resources, this is typically done through Givens rotations using the CORDIC algorithms [4, 2]. A Givens rotation operates on two rows of the K matrix and rotates them such that one element in one of the rows become zero. To make all the lower left triangular elements of K zero to get the QR decomposition, we need to apply $N(N-1)/2$ Givens rotations. The composition of all these Givens rotations results in an upper triangular matrix R from K, thus, the composition of the rotations is equivalent to the multiplication by Q^{-1}. To obtain $Q^{-1}\mathbf{1}$, we need to apply the same rotations to a $\mathbf{1}$ vector. In practice, this is achieved by simply concatenating an all 1s column at the end of K and applying all those rotations to that column as well.

This Wikipedia page offers a very good explanation of what a Givens rotation does [1]. Briefly, we select two rows i, j of the matrix A, and we pick a column k where both rows contain a nonzero value at that column. We then rotate both rows such that $A[j, k] = 0$ after the rotation. $A[i, k]$ is called the pivot. This involves computing $arctan(A[i, k], A[j, k])$, then rotating all the pairs of values in the rows by that angle. (i.e. rotate $(A[i, 0], A[j, 0])$, $(A[i, 1], A[j, 1])$ etc.) This rotation is an orthogonal transformation. If we compose such rotations we get another orthogonal transformation, which eventually will become our Q^{-1}.

On FPGA, this can be accomplished using two CORDIC blocks, one for the computation of arctan and the other for the rotation. (Since the rotator is pipelined, we can use one for all N elements in the row with only N cycles latency overhead.) Two Givens rotations can be done in parallel if they don't share any rows. Thus, the algorithm can be divided into stages, where in each stage, multiple Givens rotations are done in parallel. An interesting implementation in [2] merges the CORDIC blocks for arctan and rotation so that they can be done concurrently. The Xilinx IP does not support this operation so we don't explore this opportunity here.

The algorithm is illustrated for the 4 by 4 matrix used in this project in Figure 8.



1 (Pivot) 2 (Pivot)			
1 (Zero)	2 (Pivot) 3 (Pivot)		
1 (Pivot) 2 (Zero)	3 (Zero)	4 (Pivot)	
1 (Zero)	2 (Zero)	4 (Zero)	

Figure 8: Demonstration of CORDIC QR algorithm for a 4 by 4 matrix. The numbers correspond to stages. In each stage, multiple pairs of rows can be rotated at once. One of the rows is chosen to be a pivot and the other is chosen to be a zero. The leading nonzero in the zero row is zeroed out in that stage.

In the first stage, we do two two Givens rotations in parallel. The first one operates on the first two rows. It uses the A[0,0] as pivot and zeros out A[1,0]. The second one operates on the third and fourth row, using A[2,0] to zero out A[3,0]. In the second stage, we use A[0,0] again as pivot to zero out A[2,0]. Note that we couldn't have done this in the first stage since A[0,0] was used as pivot for another row. In the second stage, we also use A[1,1] as pivot to zero out A[3,1].

On FPGA, this is implemented again as a folded design. The key hardware module used for each stage consists of a single arctan CORDIC block feeding several rotator CORDIC blocks. We instantiate as many rotators as required in the stage with the maximum parallelism. In the example above, stage 1 and 2 can do two rotations in parallel, whereas stage 3 and 4 can only do 1. Thus, we instantiate two rotators. All the CORDIC blocks are fully pipelined. The latency of each CORDIC block depends on the desired precision and can be tuned for the application. For our analysis, let's assume it is P.

As illustrated in the QR Decomposition FSM portion of Figure 9, in each stage, we feed all the pairs of selected arctan arguments in a pipelined manner. This will incur latency $P + S$, where S is the number of pairs in that stage. The arctan outputs are routed to the appropriate input ports of each of the S rotators. When all the arctan outputs are available, we start feeding all the rotators the elements in their responsible rows. The rotators will incur latency $P + N$. Since there are four stages, we will incur total latency $8P + 4S + 4N$. This works out to be around 160 cycles with the default CORDIC latency. In practice, we observe around 210 cycles due to FSM control logic overhead.

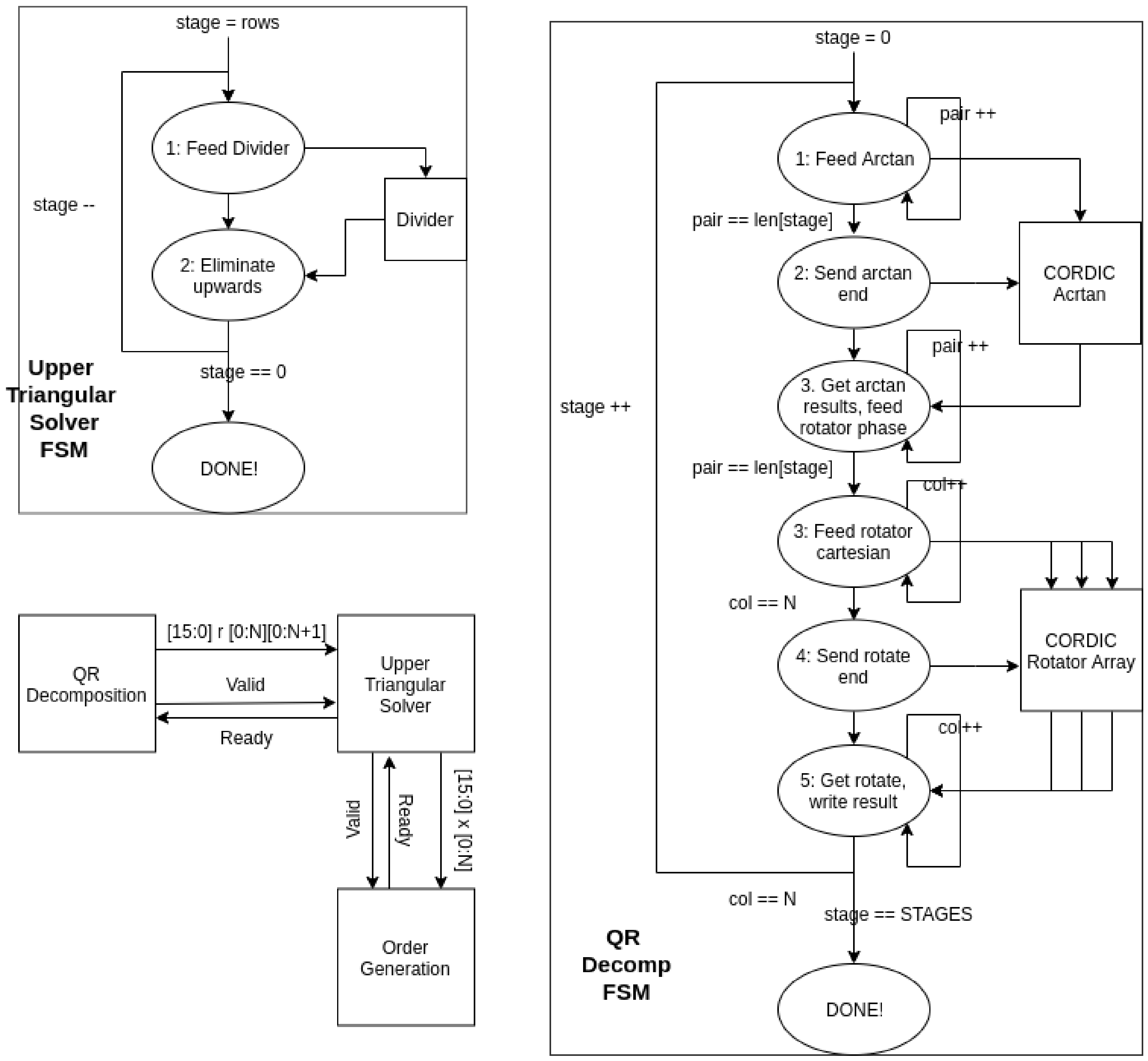


Figure 9: Block diagram of the linear system solver using QR decomposition.

After we perform the QR decomposition, we will have to solve the upper triangular linear system. This is much easier than solving a general linear system, and uses standard back-substitution. At each stage, we alternate between solving for the unknown and substituting it in all other equations. The latency is roughly equal to N divider latencies, which is around 80 cycles by default. In practice we observe 90 cycles.

5.5 System Resource Usage

For the trading logic alone, we use 11.0K (18%) LUTs, 14.1K (11%) FFs and 52 (20%) DSPs. This should leave ample room for other parts of the trading system on the same FPGA.

5.6 System Performance

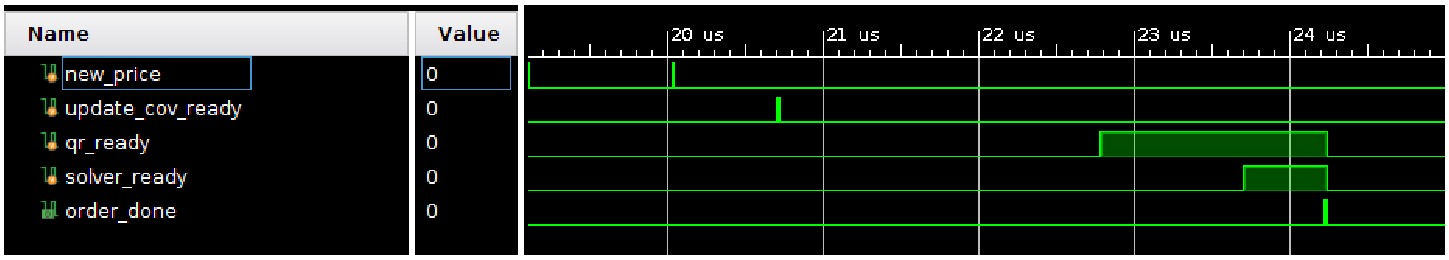


Figure 10: The overall latency of the trading system broken down into the latencies of the components, as seen by when their ready signals are asserted after a new price pulse occurs. We see that the QR decomposition represents the bulk of the latency.

The overall trading logic stack takes just over 400 cycles to generate a trade from a new price update. If we break down the latency as shown in Figure 10, we discover that the logic to update the covariance matrix takes about 70 cycles, the logic to perform the QR decomposition takes about 210 cycles, the logic to solve the upper triangular system takes 90 cycles and the logic to normalize the target position and generate the order takes 50 cycles.

The latencies of the covariance matrix update and upper triangular solver modules are dictated by the divider module, which is currently set to 20 to target 80 Mhz. We can lower this number at the risk of increasing the combinational path. For example, we can shave all of it away by setting the divider latency to 1, but that results in a max clock frequency of only 15 Mhz. Since we use a common clock for all our modules, this might not be worth it. However, at the current clock rate target, we can reduce it to 10 and still barely meet timing with a WNS < 0.1ns. The resulting overall latency is only 340 cycles, as shown in Figure 11.

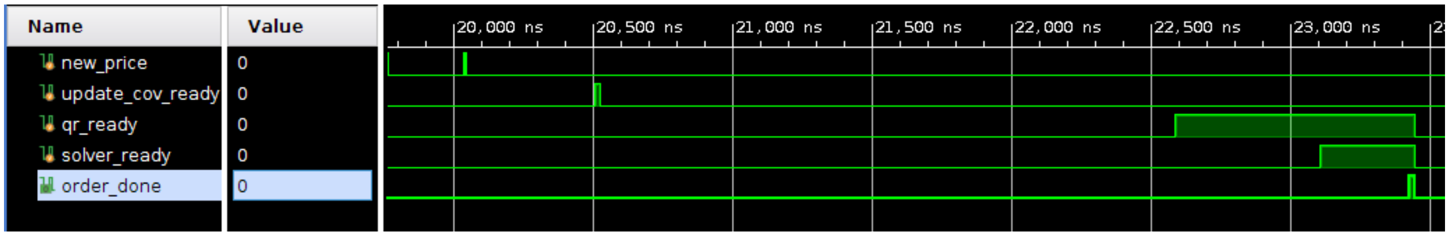


Figure 11: Tuning the divider latency decreases the total trading system latency from around 400 cycles to around 340 cycles.

The latency of the QR decomposition module is the bottleneck of the system. It is dictated by the runtime of the CORDIC blocks, as shown in Figure 12. Higher number of iterations in CORDIC give better accuracy and vice versa. We could greatly speed up this module if we sacrifice some accuracy of the final result. This design choice was also explored in [2] in their system. In our design we could tune the iteration count for either the arctan CORDIC or the rotator CORDIC. Currently in our system they are both set to the default value (14 iterations). The MSE of the final target positions of the four stocks after 50 trade updates is 0.039 (3.9%) due to fixed point error accumulation in our computations. If we change either the rotator or arctan iteration count to 5, the latency drops to 170 cycles from the current 210 for the QR. However, the final target position MSE goes up to about 0.055. If we change both CORDIC iterations to 5, then we drop the latency to 130 cycles, but the MSE goes up to 0.13 (13%), which is probably unacceptable.

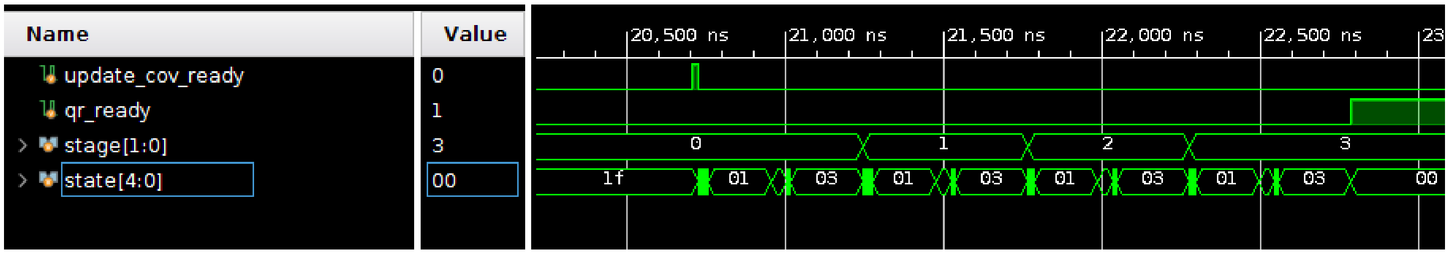


Figure 12: The breakdown of runtime for the QR Decomposition into stages. In each stage, state 1 is the arctan CORDIC and state 3 is the rotate CORDIC. As we can see they dominate total execution time.

Without these approximations, at 80 Mhz system clock, the system takes 340 cycles, which is 425 microseconds at 80Mhz. The clock rate can be higher than 100 Mhz for just the trading logic. However with other parts of the system consuming resources on the FPGA, the routing delay becomes significant and lowers the operating frequency. This is still roughly more than twice as fast as the average runtime in an optimized single threaded CPU implementation using numpy. (CPU is Intel Xeon E5-2450L at 1.80 Ghz) However, our system's latency is deterministic, whereas the CPU implementation's runtime is variable even at the same clock rate since memory systems have non deterministic latency. In practice, we observe that the runtime fluctuates between 800 microseconds to 1.6 milliseconds, which is unacceptable in a high frequency trading scenario.

5.7 Testing for Correctness

Each component of the trading logic is rigorously tested. To test the QR decomposition and upper triangular solver module, I randomly generate test matrices in Python and automatically generate the Systems Verilog test bench. I compare the simulation results against the Python output given by numpy linear solver. To test the covariance update module, I randomly generate price vector updates and automatically generate the Systems Verilog test bench. I compare the simulation results against the same computation carried out in Python. (The Python code has also been attached in Appendix.)

The overall system is also tested, from a price path consisting of 50 updates. The results are presented in a later section describing the integration between the order book module and the trading logic module.

5.8 General Observations

5.8.1 Latency insensitive design

In digital systems, oftentimes we find ourselves waiting for a particular module to finish, where the latency is parameterizable or uncertain. In our HDL code, we have to figure out exactly at which cycle the module terminates computation. This is compounded by the fact that the module might be pipelined. We need to figure out for each input when the module terminates computation.

We employ two methods in our design. The first method assumes no pipelined inputs. In this case, we use the reset-ready protocol. Before we start the module, we always reset it. When it finishes computation, it asserts high on a ready output port. It should also assert other auxiliary information at the same time, i.e. fixed point overflow. Subsequent logic dependent on the completion of the module can then rely on the ready signal.

The second method accommodates pipelined inputs. This is the tuser protocol championed by AXI-stream interface of most Xilinx IPs. (I find it the most useful part of the entire AXI interface.) We assert a unique tuser_in token for each input and wait for that token at the output. Importantly, waiting on the ready signal does not work, since you won't be able to tell different inputs in the pipelined stream from another. For my own modules, I had the option to replicate this functionality, but I chose to go with the reset-ready protocol for simplicity.

Of course, one could decide not to match latency at all and just put results into a FIFO which the consumer can dequeue from. However this method uses extra registers and wastes cycles in FIFO operations.

5.9 Lessons Learned

5.9.1 Always read the manual

One should always read the Xilinx manual from cover to cover before using any Xilinx IP. I spent hours trying to debug the divider IP, only to discover that unbeknownst to me and clearly described in the manual, both the integer and fractional parts have a sign bit! (However, sometimes the manual doesn't match up with what's offered in Vivado: e.g. no overflow port on DSPs.)

5.9.2 Fixed Point Hell

It is quite annoying to work with fixed point, or Q numbers, in Verilog, at least a lot more annoying than I previously imagined. Firstly, I'd recommend anybody working with Fixed Point to parameterize their fractional and integer width, as it is almost impossible to keep track of all the places where this information must surface (typically in scaling factors for multiplications). Improper handling of rounding might also lead to systematically wrong results and numerical instability, which luckily did not happen here. Perhaps I should have used a Fixed Point math library. However those have their own problems, such as nontransparent operations.

5.9.3 Limitations of Vivado

A DSP can be programmed to do different instructions using a select wire. If we write $a * b = c$ in Verilog, it will be inferred to DSP. If we write:

```
if (sel == 0)
    a * b = c
elif (sel == 1)
    a + b = c
```

It should use only 1 DSP with 2 instructions. However, it will infer to 2 DSPs, each with 1 instruction. I guess it's a bit much to ask Vivado to infer multiple DSP instructions. This shows that sometimes one can do more efficient things than the compiler.

6 On Board Integration (Trading & Order Book)

Since the integration between our Ethernet stack and our book builder did not end up working for demo, we built logic to directly generate the necessary orders on the FPGA. The order stream is fed into the order book, and is generated to match a target price. The target price path is generated from first fixing a initial price vector $p[0]$, a mean return vector u and a covariance matrix K. Then at each step i we sample from the multivariate normal $\mathbb{N}(u, K)$ to obtain the returns for that step, r. The price vector at that step becomes $p[i-1] \times (1+r)$. This is a commonly used model for stock price, used for example, in the Markowitz Portfolio Theory and the Black-Scholes option pricing models. We generate 50 prices for 50 steps, and run our trading logic on those 50 updates.

We see that the order book builder module is able to accurately reflect the true price path, which is shown in Figure Figure 13 and Figure Figure 14. The subsequent trading logic is also able to converge quickly to the expected target positions, with some small fixed point error. The target position after 50 price updates is 0.49, 0.27, 0.08, 0.16 from our trading system. The reference target positions is 0.49, 0.30, 0.07, 0.14. This small error has no real effect on the trading. We generate 120 new price updates, and compare how our computed positions does vs. reference Python implementation in Figure Figure 16. We see that there is little if no difference at all.

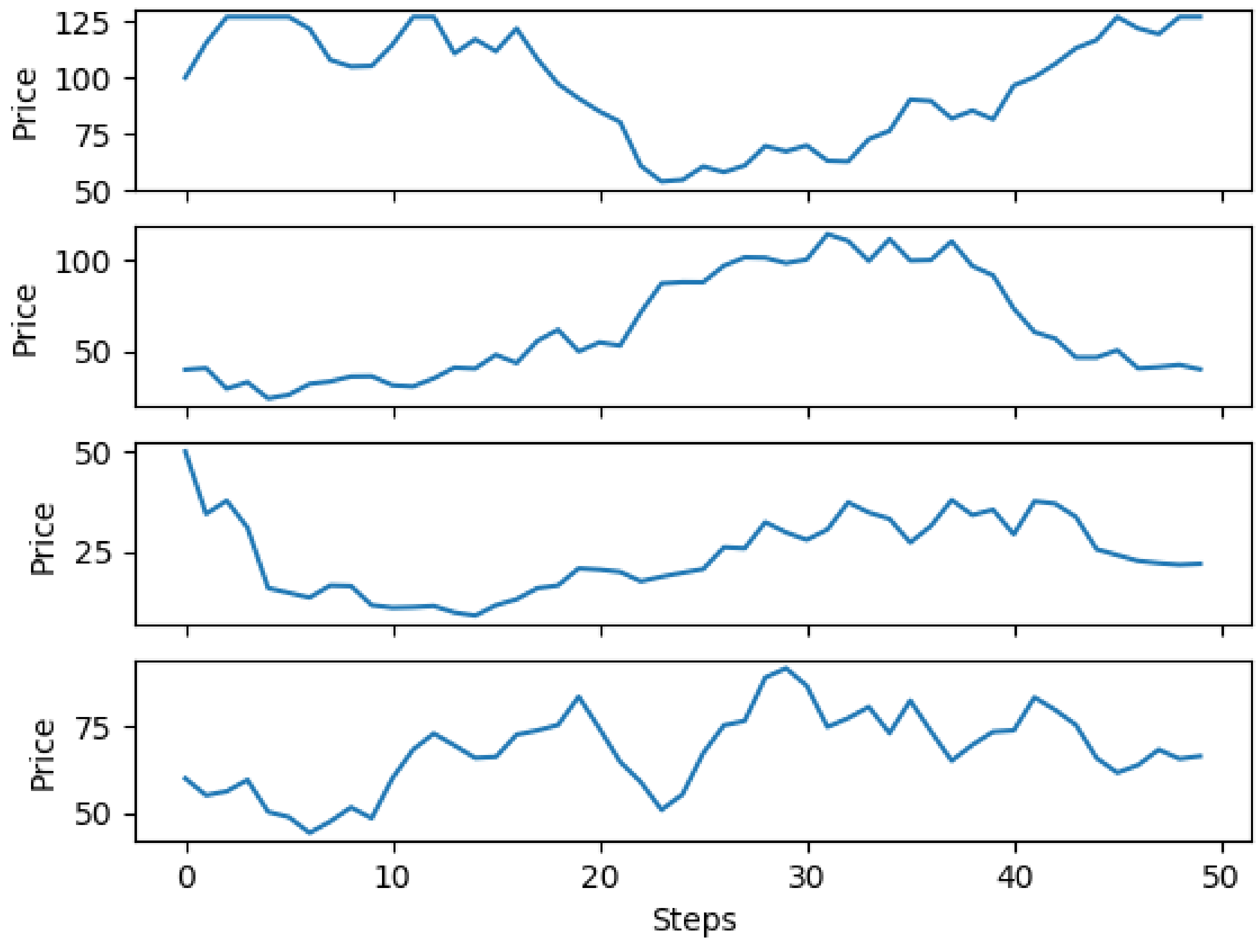
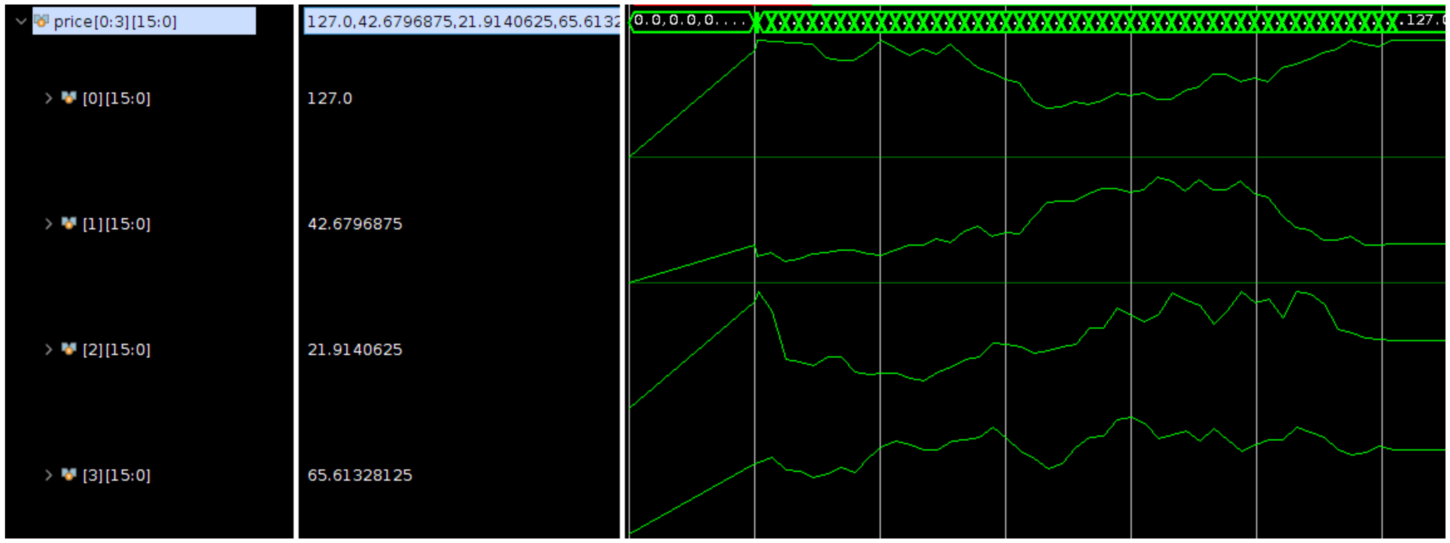


Figure 13: Target price paths.



```
price[0:3][15:0]   127.0,42.6796875,21.9140625,65.6132
  [0][15:0]        127.0
  [1][15:0]        42.6796875
  [2][15:0]        21.9140625
  [3][15:0]        65.61328125
```

Figure 14: Price path given by book builder.

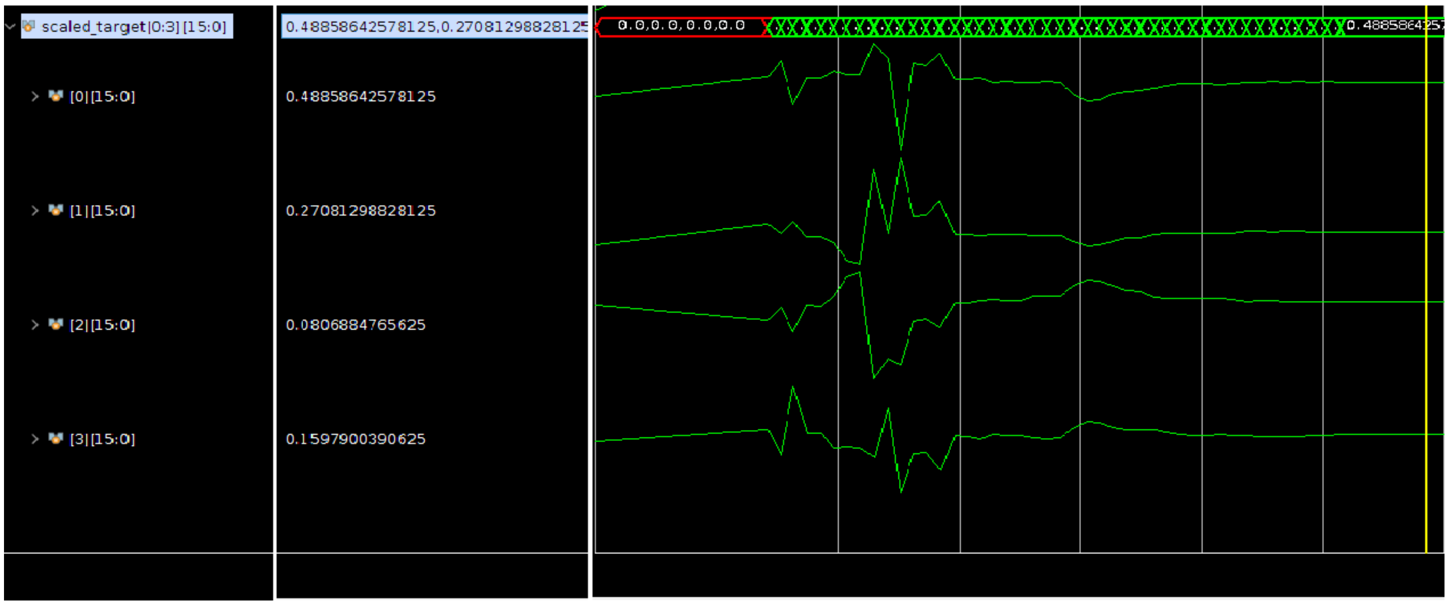


Figure 15: Convergence of scaled target positions.

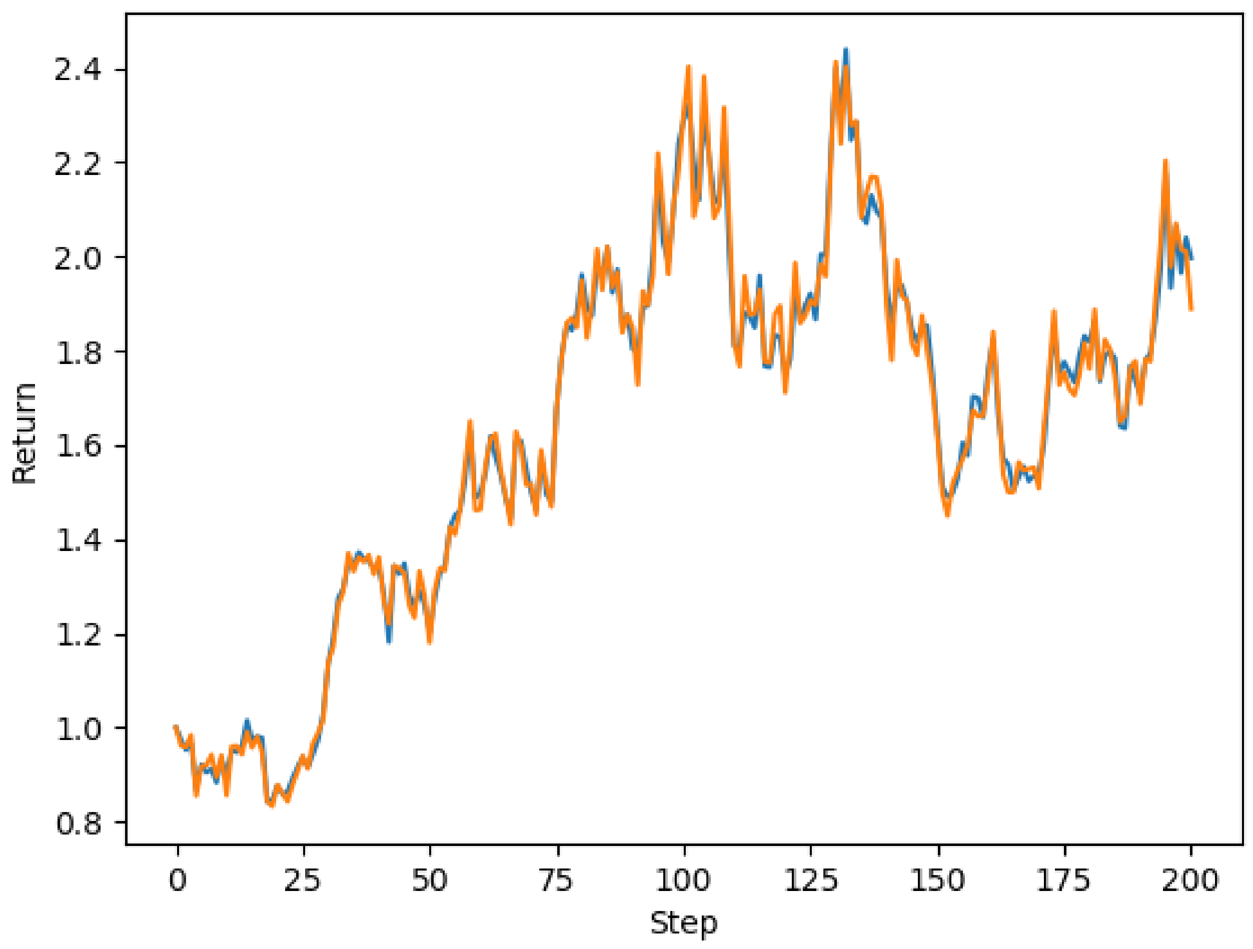


Figure 16: No difference in returns profile between FPGA computed positions and Python computed positions.

6.1 Parser and Book Integration

We were able to get a working Parse book implementation that respects the ITCH protocol. The setup is that the Parse parses and outputs relevant values for the order book to update it's state. Figure 17

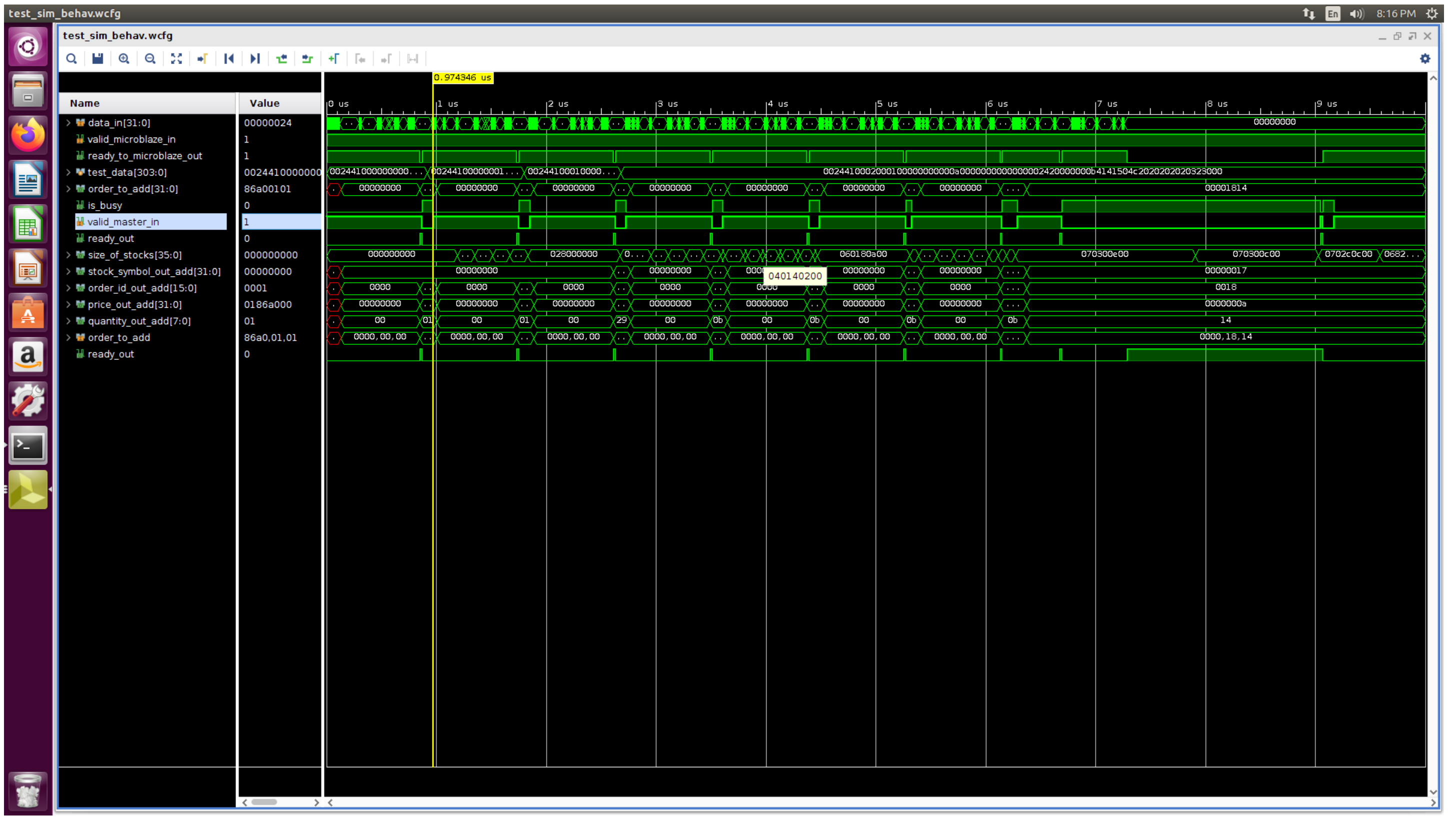

test_sim_behav.wcfg
0.974346 us
Name
Value
data_in[31:0] 00000024
valid_microblaze_in 1
ready_to_microblaze_out 1
test_data[303:0] 0024410000000
order_to_add[31:0] 86a00101
is_busy 0
valid_master_in 1
ready_out 0
size_of_stocks[35:0] 000000000
stock_symbol_out_add[31:0] 00000000
order_id_out_add[15:0] 0001
price_out_add[31:0] 0186a000
quantity_out_add[7:0] 01
order_to_add 86a0,01,01
ready_out 0

Figure 17: parser and book working in tandem. we see at the end that ready signal to Microblaze is set to false because the book is busy.

References

[1] Givens rotation, https://en.wikipedia.org/wiki/givens_rotation, Mar 2019.

[2] Dongdong Chen and Mihai Sima. Fixed-point cordic-based qr decomposition by givens rotations on fpga. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 327–332. IEEE, 2011.

[3] Martin Langhammer and Bogdan Pasca. High-performance qr decomposition for fpgas. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 183–188. ACM, 2018.

[4] Sergio D Muñoz and Javier Hormigo. High-throughput fpga implementation of qr decomposition. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(9):861–865, 2015.

[5] Scott Rome. Eigen-vesting ii. optimize your portfolio with optimization, Mar 2016.

7 Code:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/21/2019 07:20:05 PM
// Design Name:
// Module Name: cov_stack
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

// TODO: trading logic requires global reset every time you want to trade.
// seems reasonable.
`include "constants.sv"

module update_cov(
    input clk_100mhz,
    input signed [15:0] old_p [0:N-1],
    input signed [15:0] new_p [0:N-1],
    input signed [15:0] old_moment [0:N-1][0:N-1],
    input signed [15:0] old_mean [0:N-1],
    input signed [15:0] T,
    input valid,
    input reset,

    output logic signed [15:0] new_moment [0:N-1][0:N-1],
    output logic signed [15:0] new_cov [0:N-1][0:N-1],
    output logic signed [15:0] new_mean [0:N-1],
    output logic ready,
    output logic overflow

);

    logic signed [15:0] dsp_0_a [0:N-1];
    logic signed [15:0] dsp_0_b [0:N-1];
    logic signed [15:0] dsp_0_c [0:N-1][0:N-1];
    logic signed [15:0] dsp_0_d;
    logic signed [15:0] dsp_0_e [0:N-1][0:N-1];
    logic signed [15:0] dsp_0_p2 [0:N-1][0:N-1];
    logic dsp_0_valid;
    logic [2:0] dsp_0_sel;
    logic dsp_0_overflow;
    logic dsp_0_ready;
    logic dsp_0_reset;

    logic signed [15:0] dsp_1_a [0:N-1];
    logic signed [15:0] dsp_1_b [0:N-1];
    logic signed [15:0] dsp_1_c [0:N-1][0:N-1];
    logic signed [15:0] dsp_1_d;
    logic signed [15:0] dsp_1_e [0:N-1][0:N-1];
    logic signed [15:0] dsp_1_p2 [0:N-1][0:N-1];
    logic dsp_1_valid;
    logic [2:0] dsp_1_sel;
    logic dsp_1_overflow;
    logic dsp_1_ready;
    logic dsp_1_reset;

    logic signed [15:0] dsp_2_a [0:N-1];
    logic signed [15:0] dsp_2_b [0:N-1];
    logic signed [15:0] dsp_2_d;
    logic signed [15:0] dsp_2_p1 [0:N-1];
    logic dsp_2_valid;
    logic dsp_2_overflow;
    logic dsp_2_ready;
    logic dsp_2_reset;

    logic signed [15:0] vec_div_a [0:N-1];
    logic signed [15:0] vec_div_b [0:N-1];
    logic signed [15:0] vec_div_f;
    logic signed [15:0] vec_div_p1 [0:N-1];

    logic vec_div_reset;
    logic vec_div_valid;
    logic [2:0] vec_div_sel;
    logic vec_div_ready;
    logic vec_div_scale;

    logic signed [15:0] sca_dividend;
    logic signed [15:0] sca_divisor;
    logic signed [31:0] sca_div_p0;
    logic signed [15:0] sca_div_p0_clean;

    assign sca_div_p0_clean = treat_divider(sca_div_p0);

    logic sca_div_reset;
    logic sca_div_valid;
    logic sca_div_ready;
    logic [3:0] sca_div_tuser_in;
    logic [3:0] sca_div_tuser_out;

    logic signed [15:0] temp_scalar;
    logic signed [15:0] temp_vector [0:N-1];
    logic signed [15:0] returns [0:N-1];

    logic signed [15:0] temp_matrix_0 [0:N-1][0:N-1];
    logic signed [15:0] temp_matrix_1 [0:N-1][0:N-1];


    real_matrix_dsp dsp_0 (

    .clk_100mhz(clk_100mhz),
    .a(dsp_0_a), .b(dsp_0_b), .c(dsp_0_c), .d(dsp_0_d), .e(dsp_0_e),
    .reset(dsp_0_reset),.valid(dsp_0_valid),
    .sel(dsp_0_sel),
    .p2(dsp_0_p2),
    .overflow(dsp_0_overflow),
    .ready(dsp_0_ready)

    );

    real_matrix_dsp dsp_1 (

    .clk_100mhz(clk_100mhz),
    .a(dsp_1_a), .b(dsp_1_b), .c(dsp_1_c), .d(dsp_1_d), .e(dsp_1_e),
    .reset(dsp_1_reset),.valid(dsp_1_valid),
    .sel(dsp_1_sel),
    .p2(dsp_1_p2),
    .overflow(dsp_1_overflow),
    .ready(dsp_1_ready)

    );
```

```
vector_dsp dsp_2_(

.clk_100mhz(clk_100mhz),
.a(dsp_2_a), .b(dsp_2_b), .d(dsp_2_d),
.reset(dsp_2_reset),.valid(dsp_2_valid),
.p1(dsp_2_p1),
.overflow(dsp_2_overflow),
.ready(dsp_2_ready)

);

vector_division vec_divider(
.clk_100mhz(clk_100mhz),
.reset(vec_div_reset),
.valid(vec_div_valid),
.a(vec_div_a),
.b(vec_div_b),
.f(vec_div_f),
.scale(vec_div_scale),
.sel(vec_div_sel),
.d(vec_div_p1),
.ready(vec_div_ready)
);

div_gen_0 divider(
.aclk(clk_100mhz),
.s_axis_dividend_tvalid(sca_div_valid),
.s_axis_dividend_tdata(sca_dividend),
.s_axis_dividend_tuser(sca_div_tuser_in),
.s_axis_divisor_tvalid(sca_div_valid),
.s_axis_divisor_tdata(sca_divisor),
.m_axis_dout_tdata(sca_div_p0),
.m_axis_dout_tvalid(sca_div_ready),
.m_axis_dout_tuser(sca_div_tuser_out)
);

logic signed [4:0] stage;
always_ff @(posedge clk_100mhz) begin
    if(reset) begin
        stage <= -1;
        dsp_0_reset <= 1;
        dsp_1_reset <= 1;
        vec_div_reset <= 1;
        ready <= 0;
        overflow <= 0;

    end else if (valid == 1) begin
        if(stage == -1) begin

            if(T == 0) begin
                new_moment <= '{default:0};
                new_cov <= '{default:0};
                new_mean <= '{default:0};
                ready <= 1;
                overflow <= 0;
            end else begin
                vec_div_reset <= 0;
                dsp_0_reset <= 0;
                dsp_1_reset <= 0;
                vec_div_sel <= 1;
                vec_div_scale <= 1;
                vec_div_valid <= 1;

                vec_div_a <= new_p;
                vec_div_b <= old_p;
                stage <= 0;
            end
        end else if (stage == 0) begin
            if(vec_div_ready == 1) begin
                returns <= vec_div_p1;
                vec_div_reset <= 1;
                stage <= 1;
            end
        end else if (stage == 1) begin
            vec_div_reset <= 0;
            vec_div_a <= returns;
            vec_div_f <= (T + 1) ;
            vec_div_valid <= 1;
            vec_div_sel <= 0;
            vec_div_scale <= 0;

            sca_dividend <= T;
            sca_divisor <= T+1; // no need to scale both of these up
            sca_div_valid <= 1;


            stage <= 2;
        end else if (stage == 2) begin
            if(vec_div_ready && sca_div_ready) begin
                temp_vector <= vec_div_p1;
                temp_scalar <= sca_div_p0_clean;
                dsp_0_reset <= 1;
                dsp_1_reset <= 1;
                dsp_2_reset <= 1;
                // no need to reset the scalar divider since we only use it once
                // we have two different protoocls going on here:
                // one is axi and the other is reset. gotta think about this more

                stage <= 3;
             end
        end else if (stage == 3) begin
            dsp_0_reset <= 0;
            dsp_1_reset <= 0;
            dsp_2_reset <= 0;

            // b * a^T
            dsp_0_a <= returns;
            dsp_0_b <= temp_vector;
            dsp_0_valid <= 1;
            dsp_0_sel <= 0; //outer product

            // c * d
            dsp_1_c <= old_moment;
            dsp_1_d <= temp_scalar;
            dsp_1_sel <= 1;
            dsp_1_valid <= 1;

            // a* d + b
            dsp_2_a <= old_mean;
            dsp_2_d <= temp_scalar;
            dsp_2_b <= temp_vector;
            dsp_2_valid <= 1;

            stage <= 4;
        end else if (stage == 4) begin
            if(dsp_0_overflow || dsp_1_overflow || dsp_2_overflow) begin
                ready <= 1;
                overflow <= 1;
                stage <= 9;
            end if(dsp_0_ready && dsp_1_ready && dsp_2_ready) begin
                temp_vector <= dsp_2_p1;
                new_mean <= dsp_2_p1;
                temp_matrix_0 <= dsp_0_p2;
                temp_matrix_1 <= dsp_1_p2;
                dsp_0_reset <= 1;
                dsp_1_reset <= 1;
                stage <= 5;
            end

        end else if (stage == 5) begin
            dsp_0_reset <= 0;
            dsp_1_reset <= 0;
            dsp_0_valid <= 1;
            dsp_1_valid <= 1;
            dsp_0_a <= temp_vector;
            dsp_0_b <= temp_vector;
            dsp_0_sel <= 0;

            dsp_1_c <= temp_matrix_0;
            dsp_1_e <= temp_matrix_1;
            dsp_1_sel <= 2;
            stage <= 6;

        end else if (stage == 6) begin
            if(dsp_0_overflow || dsp_1_overflow) begin
                ready <= 1;
                overflow <= 1;
                stage <= 9;
            end else if(dsp_0_ready && dsp_1_ready) begin
                temp_matrix_0 <= dsp_0_p2;
                temp_matrix_1 <= dsp_1_p2;
                new_moment <= dsp_1_p2;
                dsp_0_reset <= 1;
                stage <= 7;
            end

        end else if (stage == 7) begin

            dsp_0_reset <= 0;
            dsp_0_valid <= 1;
            dsp_0_sel <= 3;
            dsp_0_c <= temp_matrix_1;
            dsp_0_e <= temp_matrix_0;
            stage <= 8;

        end else if (stage == 8) begin
            if(dsp_0_overflow) begin
                ready <= 1;
                overflow <= 1;
                stage <= 9;
```

```
                end else if(dsp_0_ready) begin
                    new_cov <= dsp_0_p2;
                    ready <= 1;
                    overflow <= 0;
                    stage <= 9;
                end

            end
        end
    end

endmodule


module vector_dsp(

    input clk_100mhz,
    input signed [15:0] a [0:N-1],
    input signed [15:0] b [0:N-1],
    input signed [15:0] d,
    input reset,
    input valid,
    output logic signed [15:0] p1 [0:N-1],
    output logic overflow,
    output logic ready

);

    always_ff @(posedge clk_100mhz) begin // vector multiply add d * a + b

        if(reset) begin
            ready <= 0;
            overflow <= 0;
            p1 <= '{default:0};
        end else if (valid == 1) begin
            for(int i = 0; i < N; i ++) begin
                p1[i] <= d * a[i] / $signed(2**14) + b[i];
                overflow <= mult_overflow(d,a[i]) || add_overflow(d*a[i] / $signed(2**14),b[i]);
                ready <= 1;
            end
        end
    end

endmodule

function automatic dsp_mult_overflow;
    input signed [32:0] c;
    if(c > $signed(2**30-1) || c < $signed(-(2**30))) begin
        $display("Multiplication Result: %b",c);
        $display("Range: %b",$signed(2**30-1) );
        dsp_mult_overflow = 1;
    end else begin
        dsp_mult_overflow = 0;
    end
endfunction

function automatic dsp_add_overflow;
    input signed [32:0] c;
    if(c > $signed(2**16-1) || c < $signed(-(2**16))) begin
        $display("Addition Result: %b",c);
        $display("Range: %b",$signed(2**16-1) );
        dsp_add_overflow = 1;
    end else begin
        dsp_add_overflow = 0;
    end
endfunction

module real_matrix_dsp(

    input clk_100mhz,
    input signed [15:0] a [0:N-1],
    input signed [15:0] b [0:N-1],
    input signed [15:0] c [0:N-1][0:N-1],
    input signed [15:0] d,
    input signed [15:0] e [0:N-1][0:N-1],
    input reset,
    input valid,
    input [2:0] tuser_in,
    input [2:0] sel,
    output logic signed [15:0] p2 [0:N-1][0:N-1],
    output logic overflow,
    output logic ready,
    output logic [2:0] tuser_out

);


    logic [1:0] dsp_sels [0:N-1][0:N-1];
    logic signed [15:0] dsp_a [0:N-1][0:N-1];
    logic signed [15:0] dsp_b [0:N-1][0:N-1];
    logic signed [15:0] dsp_c [0:N-1][0:N-1];
    logic signed [32:0] dsp_p [0:N-1][0:N-1];
    logic [0:N-1][0:N-1] mult_overflow ;
    logic [0:N-1][0:N-1] add_overflow ;

    wire any_mult_overflow;
    wire any_add_overflow; // also detects subtraction as well.
    // we register all the inputs and outputs of the DSP grid
    // as such we turn off all internal pipelining
    // how high are we going to crank the clock anyways on this -3 sg chip
    logic [1:0] state;

    genvar i,j;
    generate
        for(i = 0; i < N; i ++) begin
            for(j = 0; j < N; j++) begin
                xbip_dsp48_macro_0 dut(
                    .A(dsp_a[i][j]),
                    .B(dsp_b[i][j]),
                    .C(dsp_c[i][j]),
                    .P(dsp_p[i][j]),
                    .SEL(dsp_sels[i][j])
                );
                // need to do this stuff because I (and Joe) can't find the overflow port
                // on the DSP wizard lol
                assign mult_overflow[i][j] = dsp_mult_overflow(dsp_p[i][j]);
                assign add_overflow[i][j] = dsp_add_overflow(dsp_p[i][j]);

            end
        end
```

```
endgenerate


assign any_mult_overflow = |mult_overflow;
assign any_add_overflow = |add_overflow;


always_ff @(posedge clk_100mhz) begin

    if(reset) begin
        ready <= 0;
        overflow <= 0;
        state <= 0;
        p2 <= '{default:0};
    end else begin
        if(state == 0) begin
            if(valid == 1) begin
                case(sel)
                0: begin // vector outer product a[i] * b[j]
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            dsp_a[i][j] <= a[i];
                            dsp_b[i][j] <= b[j];
                            dsp_sels[i][j] <= 0;
                            // who produced our results here
                        end
                    end
                end
                1: begin // matrix element wise multiplication by scalar
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            dsp_a[i][j] <= c[i][j];
                            dsp_b[i][j] <= d;
                            dsp_sels[i][j] <= 0;
                        end
                    end
                end
                2: begin // matrix element wise addition
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            dsp_a[i][j] <= c[i][j];
                            dsp_c[i][j] <= e[i][j];
                            dsp_sels[i][j] <= 1;
                        end
                    end
                end
                3: begin // matrix element wise subtraction
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            dsp_a[i][j] <= c[i][j];
                            dsp_c[i][j] <= e[i][j];
                            dsp_sels[i][j] <= 2;
                        end
                    end
                end

                default: begin
                    overflow <= 1;
                    ready <= 0;
                    end
                    endcase
                    state <= 1;
                end

            end else if (state == 1) begin
                case(sel)
                    0: begin // vector outer product a[i] * b[j]
                        overflow <= any_mult_overflow;
                        for(int i = 0; i < N; i ++) begin
                            for(int j =0; j < N; j++) begin
                                p2[i][j] <= dsp_p[i][j] / $signed(2**14);
                            end
                        end

                    end
                    1: begin // matrix element wise multiplication by scalar
                        overflow <= any_mult_overflow;
                        for(int i = 0; i < N; i ++) begin
                            for(int j =0; j < N; j++) begin
                                p2[i][j] <= dsp_p[i][j] / $signed(2**14);
                            end
                        end

                    end
                    2: begin // matrix element wise addition
                        overflow <= any_add_overflow;
                        for(int i = 0; i < N; i ++) begin
                            for(int j =0; j < N; j++) begin
                                p2[i][j] <= dsp_p[i][j];
                            end
                        end
                    end
                    3: begin // matrix element wise subtraction
                        overflow <= any_add_overflow;
                        for(int i = 0; i < N; i ++) begin
                            for(int j =0; j < N; j++) begin
                                p2[i][j] <= dsp_p[i][j];
                            end
                        end
                    end

                    default: begin
                        overflow <= 1;
                        ready <= 0;
                    end
                endcase
                ready <= 1;
                state <= 2; //need to be reset before accepting new computation.
            end

        end

    end

endmodule

module matrix_dsp(

    input clk_100mhz,
    input signed [15:0] a [0:N-1],
    input signed [15:0] b [0:N-1],
    input signed [15:0] c [0:N-1][0:N-1],
    input signed [15:0] d,
    input signed [15:0] e [0:N-1][0:N-1],
    input reset,
    input valid,
    input [2:0] tuser_in,
    input [2:0] sel,
    output logic signed [15:0] p2 [0:N-1][0:N-1],
    output logic overflow,
    output logic ready,
    output logic [2:0] tuser_out

);


    // to be replaced by genvar of DSP array if required
    always_ff @(posedge clk_100mhz) begin

        if(reset) begin
            ready <= 0;
            overflow <= 0;
            p2 <= '{default:0};
        end else if (valid == 1) begin
            case(sel)
                0: begin // vector outer product a[i] * b[j]
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            p2[i][j] <= a[i] * b[j] / $signed(2**14);

                            // VERY DANGEROUS logic, just for simulation.

                            overflow <= mult_overflow(a[i],b[j]);
                            ready <= 1;
                            tuser_out <= tuser_in; // no need to keep track of
                            // who produced our results here
                        end
                    end
                end
                1: begin // matrix element wise multiplication by scalar
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            p2[i][j] <= c[i][j] * d / $signed(2**14);
                            overflow <= mult_overflow(c[i][j], d);
                            ready <= 1;
                            tuser_out <= tuser_in;
                        end
                    end
                end
                2: begin // matrix element wise addition
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            p2[i][j] <= c[i][j] + e[i][j];
                            overflow <= add_overflow(c[i][j], e[i][j]);
                            ready <= 1;
                            tuser_out <= tuser_in;
                        end
                    end
                end
                3: begin // matrix element wise subtraction
                    for(int i = 0; i < N; i++) begin
                        for(int j = 0; j < N; j ++) begin
                            p2[i][j] <= c[i][j] - e[i][j];
                            overflow <= add_overflow(c[i][j], -e[i][j]);
                            ready <= 1;
                            tuser_out <= tuser_in;
                        end
                    end
                end

                default: begin
                    overflow <= 1;
                    ready <= 0;
                end
            endcase
        end

    end

endmodule


module vector_division(
    input clk_100mhz,
    input reset,
    input valid,
    input signed [15:0] a [0:N-1],
    input signed [15:0] b [0:N-1],
    input signed [15:0] f,
    input scale,
    input [2:0] sel,
    input [2:0] vec_tuser_in,
    output logic signed [15:0] d [0:N-1],
    output logic ready,
    output logic [2:0] vec_tuser_out,
    output logic overflow
);

    logic signed [15:0] dividend;
    logic signed [15:0] divisor;
    logic signed [31:0] c;

    logic dividend_valid;
    logic divisor_valid;
    logic result_valid;
    logic [LOGN+1:0] tuser_in;
    logic [LOGN+1:0] tuser_out;


    div_gen_0 divider(
    .aclk(clk_100mhz),
    .s_axis_dividend_tvalid(dividend_valid),
    .s_axis_dividend_tdata(dividend),
    .s_axis_dividend_tuser(tuser_in),
    .s_axis_divisor_tvalid(divisor_valid),
    .s_axis_divisor_tdata(divisor),
    .m_axis_dout_tdata(c),
    .m_axis_dout_tvalid(result_valid),
    .m_axis_dout_tuser(tuser_out)
    );

    logic state;
    logic [LOGN+1:0] processing;
    logic [LOGN+1:0] getting;


    logic signed [15:0] result;
    logic signed [30:0] long_result;

    assign long_result = $signed(c[30:15]) * $signed(2**14) + $signed(c[14:0]);

    always_comb begin
        if(scale == 1) begin
//            if(£signed(long_result[29:14]) > 1) begin
//                result = {£signed(1),14'b0};
//            end else if (£signed(long_result[29:14]) < -2) begin
//                result = {£signed(-2),14'b0};
//            end else begin
//                result = £signed(long_result[15:0]);
//            end
            result = $signed(long_result[15:0]);

        end else begin
            result = $signed(long_result[29:14]);
        end
    end


    always_ff @(posedge clk_100mhz) begin

        if(reset==1) begin
            dividend <= 0;
            divisor <= 0;
            tuser_in <= 0;
            state <= 0;
            processing <= 0;
            getting <= 0;
            ready <= 0;
            overflow <= 0;
            dividend_valid <= 0;
            divisor_valid <= 0;
        end else if (valid==1) begin
            if(state == 0) begin
                state <= 1;

                case(sel)
                    0: begin
                        dividend <= a[processing];
                        divisor <= f;
                        dividend_valid <= 1;
                        divisor_valid <= 1;
                    end
                    1: begin
                        dividend <= a[processing] - b[processing];
                        divisor <= b[processing];
                        dividend_valid <= 1;
                        divisor_valid <= 1;
                    end
                    2: begin
                        dividend <= a[processing];
                        divisor <= b[processing];
                        dividend_valid <= 1;
                        divisor_valid <= 1;
                    end
                endcase
                // tuser_in is 1 indexed. This so we not confuse 0, which is default with 0 index.
                tuser_in <= processing + 1;
                processing <= processing + 1;
            end else if (state == 1) begin
                if(processing < N) begin
                    case(sel)
                    0: begin
                        dividend <= a[processing];
                        divisor <= f;
                        dividend_valid <= 1;
                        divisor_valid <= 1;
                    end
                    1: begin
                        dividend <= a[processing] - b[processing];
                        divisor <= b[processing];
                        dividend_valid <= 1;
                        divisor_valid <= 1;
                    end
                    2: begin
                        dividend <= a[processing];
                        divisor <= b[processing];

                        dividend_valid <= 1;
                        divisor_valid <= 1;
                    end
                endcase
                    tuser_in <= processing + 1;
                    processing <= processing + 1;
                end
                if(tuser_out == getting + 1) begin
                    d[getting] <= result;
                    if(result == {$signed(1),14'b0} || result == {$signed(-2),14'b0}) begin
                        overflow <= 1;
                    end
                    if(getting == N-1) begin
                        state <= 2;
                        ready <= 1;
                    end else begin
                        getting <= getting + 1;
                    end
                end

            end else if (state == 2) begin
                ready <= 1;
                dividend_valid <= 0;
                divisor_valid <= 0;
            end
        end
    end

endmodule


`include "constants.sv"
```

```
module test_integration(
 input sys_clk,
 input [15:0] sw,
 input btnd,
 output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
output[7:0] an,
output led16_b,
output led16_g,
output led16_r
    );

    logic rst;
    logic [STOCK_INDEX:0] stock_to_add;
    book_entry entry;
    logic start;
    logic [2:0] request;
    logic delete;
    logic is_busy;
    logic [ORDER_INDEX:0] order_id;
    logic [QUANTITY_INDEX:0] quantity;

    logic book_busy;

    wire clk_80mhz;
    logic pick_next;
    logic pick_next_r;

    clk_wiz_0 bump(.clk_in1(sys_clk),.clk_out1(clk_80mhz));
    debounce d1(.clock_in(clk_80mhz), .reset_in(sw[15]), .noisy_in(btnd), .clean_out(pick_next));


 top_level dut(
    .clk_100mhz(clk_80mhz),
    .sw(sw),
    .stock_to_add(stock_to_add),
    .entry(entry),
    .start(start),
    .request(request),
    .order_id(order_id),
    .delete(delete), //should be one if cancel
    .quantity(quantity),
    .book_busy(book_busy),
    .ca(ca), .cb(cb), .cc(cc), .cd(cd), .ce(ce), .cf(cf), .cg(cg), .dp(dp),  // segments a-g, dp
    .an(an),
    .led16_b(led16_b),
    .led16_g(led16_g),
    .led16_r(led16_r)
);

assign is_busy = book_busy;
localparam END = 48;

localparam STOCKS = 4;
logic [6:0] price_index  = 0;

logic [5:0] state;
logic [5:0] state2;
logic [20:0] counter;
logic [5:0] i;
logic  [PRICE_INDEX:0]  d_array_ [0:48] [0:NUM_STOCK_INDEX];


assign d_array_[0] = '{115.355298916*2**8,40.9465656921*2**8,34.4863845528*2**8,55.1660735136*2**8};
assign d_array_[1] = '{127.0*2**8,29.5824195882*2**8,37.7681143599*2**8,56.3017956783*2**8};
assign d_array_[2] = '{127.0*2**8,33.1636790203*2**8,31.1068030893*2**8,59.575229658*2**8};
assign d_array_[3] = '{127.0*2**8,24.2162177991*2**8,16.112152735*2**8,50.4074439653*2**8};
assign d_array_[4] = '{127.0*2**8,26.2766803139*2**8,14.9420497843*2**8,48.922379298*2**8};
assign d_array_[5] = '{121.714203685*2**8,32.1975154513*2**8,13.7534866892*2**8,44.4035407743*2**8};
assign d_array_[6] = '{107.831157802*2**8,33.5667962178*2**8,16.7753312399*2**8,47.6596497209*2**8};
assign d_array_[7] = '{104.862285374*2**8,36.329730527*2**8,16.6409127224*2**8,51.7086851658*2**8};
assign d_array_[8] = '{105.119810494*2**8,36.4632174103*2**8,11.8835222024*2**8,48.5692606977*2**8};
 assign d_array_[9] = '{114.597976689*2**8,31.5138766363*2**8,11.2625986153*2**8,60.0325608116*2**8};
assign d_array_[10] = '{127.0*2**8,30.8718742797*2**8,11.3880525765*2**8,68.2524073185*2**8};
assign d_array_[11] = '{127.0*2**8,35.2823975615*2**8,11.6782023548*2**8,72.8426985594*2**8};
assign d_array_[12] = '{110.719361169*2**8,41.2466444431*2**8,10.0730285769*2**8,69.4478890888*2**8};
assign d_array_[13] = '{117.036119074*2**8,40.7110493239*2**8,9.28277202102*2**8,65.8450732847*2**8};
assign d_array_[14] = '{111.715096564*2**8,48.1988851077*2**8,11.8730974208*2**8,66.1485055281*2**8};
assign d_array_[15] = '{121.865032382*2**8,43.7055950132*2**8,13.3517863552*2**8,72.524286021*2**8};
assign d_array_[16] = '{108.302428202*2**8,55.8442320488*2**8,16.0832484483*2**8,73.7296488882*2**8};
assign d_array_[17] = '{97.226952325*2**8,62.014963434*2**8,16.7774795442*2**8,75.2919034407*2**8};
assign d_array_[18] = '{90.6678104747*2**8,50.0936828466*2**8,21.0118773381*2**8,83.4179169717*2**8};
assign d_array_[19] = '{84.8955530082*2**8,55.0117560956*2**8,20.6890245493*2**8,74.2816540892*2**8};
assign d_array_[20] = '{80.1419504053*2**8,53.3106375326*2**8,20.137454168*2**8,64.7584964188*2**8};
assign d_array_[21] = '{60.7513387387*2**8,71.6125388503*2**8,17.7900295254*2**8,58.9446616845*2**8};
assign d_array_[22] = '{53.7268766339*2**8,87.4878206377*2**8,18.9229693044*2**8,50.9871838095*2**8};
assign d_array_[23] = '{54.4787402579*2**8,88.2286718086*2**8,19.9184967643*2**8,55.413255289 7*2**8};
assign d_array_[24] = '{60.4511616641*2**8,88.1367642681*2**8,20.8200638912*2**8,67.2460681644*2**8};
assign d_array_[25] = '{57.9115691272*2**8,97.1932589329*2**8,26.2062138427*2**8,75.2154847462*2**8};
assign d_array_[26] = '{60.8418186729*2**8,101.752719816*2**8,25.9507034878*2**8,76.4600177024*2**8};
assign d_array_[27] = '{69.4445333909*2**8,101.528164165*2**8,32.4440127955*2**8,88.8203943876*2**8};
assign d_array_[28] = '{67.1445591665*2**8,98.6093829304*2**8,29.9111268138*2**8,91.4736035158*2**8};
assign d_array_[29] = '{69.6854153859*2**8,100.625107418*2**8,28.064677332*2**8,86.4577312171*2**8};
assign d_array_[30] = '{63.0214546891*2**8,114.547429973*2**8,30.6303494529*2**8,74.7318935931*2**8};
assign d_array_[31] = '{62.7039351437*2**8,110.804146475*2**8,37.3565836673*2**8,77.239585318*2**8};
assign d_array_[32] = '{72.5134151997*2**8,99.7846919829*2**8,34.8330094562*2**8,80.4745304686*2**8};
assign d_array_[33] = '{76.2436407022*2**8,111.841025794*2**8,33.2757954451*2**8,72.9691931419*2**8};
assign d_array_[34] = '{90.2206143745*2**8,100.061234541*2**8,27.4101689838*2**8,82.2846129524*2**8};
assign d_array_[35] = '{89.5106913138*2**8,100.380392776*2**8,31.5142202558*2**8,73.4638548513*2**8};
assign d_array_[36] = '{81.7403296696*2**8,110.582419879*2**8,37.9382660578*2**8,65.0549725876*2**8};
assign d_array_[37] = '{85.2385067308*2**8,97.0056728205*2**8,34.2251549301*2**8,69.5930200369*2**8};
assign d_array_[38] = '{81.3976054959*2**8,91.8966607002*2**8,35.5322675347*2**8,73.3630103629*2**8};
assign d_array_[39] = '{96.4392373603*2**8,73.5818473341*2**8,29.4327838287*2**8,73.7432268743*2**8};
assign d_array_[40] = '{100.173689874*2**8,60.7258793598*2**8,37.6357014651*2**8,83.1968419319*2**8};
assign d_array_[41] = '{106.07352731*2**8,57.1030912601*2**8,37.0329010041*2**8,79.6106047598*2**8};
assign d_array_[42] = '{113.061032135*2**8,46.765730082*2**8,33.7549827641*2**8,75.3121698778*2**8};
assign d_array_[43] = '{116.737698198*2**8,46.7820686982*2**8,25.7475344197*2**8,65.830989642*2**8};
assign d_array_[44] = '{127.0*2**8,50.7438975216*2**8,24.2887476739*2**8,61.6278785072*2**8};
assign d_array_[45] = '{121.930017526*2**8,40.8344653949*2**8,22.8432596129*2**8,63.7982933554*2**8};
assign d_array_[46] = '{119.410717373*2**8,41.513271546*2**8,22.3017840614*2**8,68.2367308458*2**8};
assign d_array_[47] = '{127.0*2**8,42.6778194476*2**8,21.9146782117*2**8,65.612051776*2**8};
assign d_array_[48] = '{127.0*2**8,40.2419214303*2**8,22.1645451001*2**8,66.3635141004*2**8};



always_ff @(posedge clk_80mhz) begin

    if(sw[3] == 1) begin
        counter <= 0;
        state <= 0;
        state2 <= 0;
        i <= 0;
        price_index <= 0;
    end else begin

        pick_next_r <= pick_next;

        if(state == 0) begin
            if(price_index != 0) begin
                if(state2 == 0) begin
                    stock_to_add <= i;
                    start <= 1;
                    order_id <= 1;
                    request <= CANCEL_ORDER;
                    delete <= 1;
                    state2 <= 5;
                end else if (state2 == 5) begin
                    state2 <= 1;
                end else if (state2 == 1) begin
                    start <= 0;
                    if(is_busy) begin
                        state2 <= 1;
                    end else begin
                        if(i == NUM_STOCK_INDEX) begin
                            i <= 0;
                            state2 <= 2;
                        end else begin
                            state2 <= 0;
                            i <= i + 1;
                        end
                    end
                end else if (state2 == 2) begin
                    stock_to_add <= i;
                    start <= 1;
                    entry <= '{price:d_array_[price_index][i], order_id:1, quantity:1};
                        request <= ADD_ORDER;
                    state2 <= 6;
                end else if (state2 == 6) begin
                    state2 <= 3;
                end else if (state2 == 3) begin
                    start <= 0;
                    if(is_busy) begin
                        state2 <= 3;
                    end else begin
                        if(i == NUM_STOCK_INDEX) begin
                            i <= 0;
                            state2 <= 4;
                        end else begin
                            state2 <= 2;
                            i <= i + 1;
                        end
                    end
                end else if (state2 == 4) begin
                    state <= 1;
                    counter <= 0;
                end
            end else begin
                if (state2 == 0) begin
                    stock_to_add <= i;
```

```
                    start <= 1;
                    entry <= '{price:d_array_[price_index][i], order_id:1, quantity:1};
                    request <= ADD_ORDER;
                    state2 <= 5;
                end else if (state2 == 5) begin
                    state2 <= 1;
                end else if (state2 == 1) begin
                    start <= 0;
                    if(is_busy) begin
                        state2 <= 1;
                    end else begin
                        if(i == NUM_STOCK_INDEX) begin
                            i <= 0;
                            state2 <= 2;
                        end else begin
                            state2 <= 0;
                            i <= i + 1;
                        end
                    end
                end else if (state2 == 2) begin
                    state <= 1;
                    counter <= 0;
                end
            end


        end else if (state == 1) begin
//            if(counter == 1000) begin
//                if(price_index == 48 - 1) begin
//                    state <= 2;
//                end else begin
//                    state <= 0;
//                    state2 <= 0;
//                    i <= 0;
//                    price_index <= price_index + 1;
//                end
//            end else begin
//                counter <= counter + 1;
//            end
            if(pick_next_r && !pick_next) begin
                if(price_index == 48 - 1) begin
                    state <= 2;
                end else begin
                    state <= 0;
                    state2 <= 0;
                    i <= 0;
                    price_index <= price_index + 1;
                end
            end else begin
                state <= 1;
            end
        end
    end
end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
```

```
//
// Create Date: 11/05/2019 11:23:18 AM
// Design Name:
// Module Name: top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

`include "constants.sv"

function automatic signed [15:0] treat_divider;
    input [31:0] c;
    logic signed [30:0] long_result;

    long_result = $signed(c[30:15]) * $signed(2**14) + $signed(c[14:0]);

    if($signed(long_result[29:14]) > 1) begin
        treat_divider = {$signed(1),14'b0};
    end else if ($signed(long_result[29:14]) < -2) begin
        treat_divider = {$signed(-2),14'b0};
    end else begin
        treat_divider = $signed(long_result[15:0]);
    end

endfunction

function automatic mult_overflow;
    input signed [15:0] a;
    input signed [15:0] b;
    logic signed [31:0] c = a * b;
    if(c > $signed(2**30-1) || c < $signed(-(2**30))) begin
//        £display("Multiplication Result: %b",c);
//        £display("Range: %b",£signed(2**30-1) );
        mult_overflow = 1;
    end else begin
        mult_overflow = 0;
    end
endfunction

function automatic add_overflow;
    input signed [15:0] a;
    input signed [15:0] b;
    logic signed [16:0] sum = a+b;
    if(sum > $signed(2**16-1) || sum < $signed(-(2**16))) begin
        $display("Addition Result: %b",sum);
        $display("Range: %b",$signed(2**16-1) );
        add_overflow = 1;
    end else begin
        add_overflow = 0;
    end
endfunction
```

```
module send_order(
    input clk_100mhz,
    input signed [15:0] positions [0:N-1],
    input signed [15:0] target [0:N-1],
    input signed [15:0] curr_price [0:N-1],
    input reset,
    input valid,
    output logic [0:18*N-1] bus,
    output logic done,
    output logic error
);

  parameter signed [15:0] account = 1024;

  logic signed [15:0] shares [0:N-1];

  logic signed [15:0] difference [0:N-1];
  logic div_start;
  logic div_ready;
  logic [2:0] state;

  logic [0:N] overflow;
  logic any_overflow;
  logic div_overflow;

  genvar i;
  generate
    for(i = 0 ; i < N; i ++) begin
        assign difference[i] = target[i] - positions[i];
        assign overflow[i] = mult_overflow(shares[i],account) || add_overflow(target[i],-positions[i]);
        assign bus[18*i:18*i+1] = (shares[i] > 0) ? 2'b01 : 2'b00;
        assign bus[18*i+2:18*i+17] = (shares[i] > 0) ? shares[i] * account / $signed(2**8): (-shares[i]
         ↪  * account)/$signed(2**8); // we can only trade integer shares!
    end
  endgenerate

  assign any_overflow = (|overflow) || div_overflow;

  vector_division divider(
    .clk_100mhz(clk_100mhz),
    .reset(reset),
    .valid(div_start),
    .a(difference),
    .b(curr_price),
    .sel(2),
    .scale(1),
    .d(shares),
    .ready(div_ready),
    .overflow(div_overflow));


    always_ff@(posedge clk_100mhz) begin
        if(reset) begin
            state <= 0;
            div_start <= 0;
            error <= 0;
            done <= 0;
        end else begin
            if(state == 0) begin
```

```
                if(valid) begin
                    div_start <= 1;
                    state <= 1;
                end
            end else if(state == 1) begin
                if(div_ready) begin
                    if(any_overflow) begin
                        error <= 1;
                        done <= 1;
                    end else begin
                        done <= 1;
                    end
                end
            end
        end
    end


endmodule




module get_price(
    input clk_100mhz,
    input reset,
    input signed [15:0] latest_price [0:N-1],
    input latest_price_valid,
    output logic signed [15:0] reg_latest_price [0:N-1],
    output logic new_price // this will be a pulse
);

    parameter DELAY = 100;

    logic [20:0] counter;
    always_ff @(posedge clk_100mhz) begin
        if(reset) begin
            counter <= DELAY-1;
            new_price <= 0;
            reg_latest_price <= '{default:0};
        end else begin
            if(counter == DELAY-1) begin

            // TODO: in production system, we need to remove the condition latest_price !=
            ↪  reg_latest_price
            // this is to avoid simulation problems where the price is the same, driving covariance
            ↪  matrix to zero
            // in production, if the price is the same, then the covariance matrix should be zero!
            // then we will have a ton of overflows, but that's okay.


                if(latest_price != reg_latest_price && latest_price_valid) begin
                    reg_latest_price <= latest_price;
                    new_price <= 1;
                    counter <= 0;
                end else begin
                    counter <= 0;
                end
            end else begin
                counter <= counter + 1;
                new_price <= 0;
            end

        end

    end

endmodule

module debounce (input reset_in, clock_in, noisy_in,
                 output reg clean_out);

   reg [19:0] count;
   reg new_input;

   always_ff @(posedge clock_in)
     if (reset_in) begin
        new_input <= noisy_in;
        clean_out <= noisy_in;
        count <= 0; end
     else if (noisy_in != new_input) begin new_input<=noisy_in; count <= 0; end
     else if (count == 60) clean_out <= new_input;
     else count <= count+1;


endmodule

module top_level_ip(
    input clk_100mhz,
    input [15:0] sw,
    input [STOCK_INDEX:0] stock_to_add,
    input book_entry entry,
    input start,
    input [2:0] request,
    input [ORDER_INDEX:0] order_id, //should be supplied for a cancel / trade
    input delete, //should be one if cancel
    input [QUANTITY_INDEX:0] quantity,  // sho1uld be supplied for a trade

    output logic book_busy,
    output logic [SIZE_INDEX:0] size_of_stocks [0:NUM_STOCK_INDEX],
    output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
    output[7:0] an,
    output led16_b,
    output led16_g,
    output led16_r,
    output logic [71:0] bus,
    output logic done,
    output logic error
);
    logic done, error;
    assign led16_g = done;
    assign led16_r = error;
    logic signed [15:0] price [0:N-1];
    logic [5:0] counter;
    wire global_reset;
    wire new_p;
    logic new_p_r;
    logic [15:0] price_u[0:N-1];

    logic price_valid;
    logic signed [15:0] prices [0:8][0:N-1];

    genvar i;
    generate
        for(i = 0; i < N; i++) begin
            assign price[i] = price_u[i];
        end
    endgenerate


    order_book_wrapper top_(
    .clk_in(clk_100mhz),
    .rst_in(sw[15]),
    .stock_to_add(stock_to_add),
    .order_to_add(entry),
    .start(start),
    .request(request),
    .order_id(order_id),
    .delete(delete), //should be one if cancel
    .quantity(quantity),
    .is_busy(book_busy),
    .best_price_stocks(price_u),
    .best_prices_valid(price_valid),
    .size_of_stocks(size_of_stocks)
    );

    assign global_reset = sw[0];

    logic signed [15:0] scaled_target [0:N-1];

    trading_top dut(
    .clk_100mhz(clk_100mhz),
    .latest_price(price),
    .latest_price_valid(price_valid),
    .global_reset(global_reset),
    .scaled_target(scaled_target),
    .bus(bus),
    .global_overflow(led16_b),
    .order_done(done),
    .order_error(error)
    );


//    serial_tx comm_unit(.clk_in(clk_100mhz),
//                    .rst_in(global_reset),
//                    .trigger_in(send_out_r && !send_out),
//                    .val_in(scaled_target[0][7:0]),
//                   .done(done),
//                    .data_out(ja[0]));

    wire [6:0] segments;
    assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];

    logic [31:0] display_bits;

    always_comb begin
        if(sw[14]) begin
            display_bits = {scaled_target[0],scaled_target[1]};
        end else if (sw[13]) begin
```

```
            display_bits = {scaled_target[2],scaled_target[3]};
        end else if (sw[12]) begin
            display_bits = {price[0],price[1]};
        end else if (sw[11]) begin
            display_bits = {price[2],price[3]};
        end else if (sw[10]) begin
            display_bits = bus[71:32];
        end else begin
            display_bits = bus[31:0];
        end
    end

    display_8hex display(.clk_in(clk_100mhz),.data_in(display_bits), .seg_out(segments),
    ↪  .strobe_out(an));

    always_ff@(posedge clk_100mhz) begin

        if(global_reset == 1) begin
            //new_p <= 0;
            new_p_r <= 0;
            //bus <= 0;
            counter <= 0;

        end else if(new_p == 1 && new_p_r == 0) begin
            counter <= counter + 1;
            new_p_r <= new_p;
        end else begin
            new_p_r <= new_p;

        end

//        if(new_p && !new_p_r) begin
//            counter <= counter + 1;
//            new_p_r <= new_p;
//        end else begin
//            new_p_r <= new_p;
//        end
    end

endmodule

module top_level_trade(
    input clk_100mhz,
    input sw[15:0],
    input btnc,
    input btnd,
    input btnu,
    output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
    output[7:0] an,
    output led16_b,
    output led16_g,
    output led16_r
//    output logic [71:0] bus,
//    output logic done,
//    output logic error
);
    logic done, error;
    assign led16_g = done;
    assign led16_r = error;
    logic[71:0] bus;
    logic signed [15:0] price [0:N-1];
    logic [5:0] counter;
    wire global_reset;
    wire new_p;
    logic new_p_r;
    logic [15:0] price_u[0:N-1];

    logic price_valid;

    logic signed [15:0] prices [0:8][0:N-1];

    genvar i;
    generate
        for(i = 0; i < N; i++) begin
            assign price[i] = price_u[i];
        end
    endgenerate


    logic book_busy;

       logic pick_next;
       debounce d1(.clock_in(clk_100mhz), .reset_in(sw[15]), .noisy_in(btnu), .clean_out(pick_next));
       logic pick_next_previous;
       logic change;
       always_ff @(posedge clk_100mhz) begin
           pick_next_previous <= pick_next;
           change <= pick_next_previous == 0 && pick_next == 1;
       end

    top_level_simulation top_level_sim (.clk_in(clk_100mhz),
    .next(change),
    .reset(sw[15]),
     .busy(book_busy),
      .best_price_stocks (price_u),
     .stocks_valid(price_valid));
//    assign prices[0] =
↪   '{115.355298916*2**8,40.9465656921*2**8,34.4863845528*2**8,55.1660735136*2**8};
//    assign prices[1] = '{127.0*2**8,29.5824195882*2**8,37.7681143599*2**8,56.3017956783*2**8};
//    assign prices[2] = '{127.0*2**8,33.1636790203*2**8,31.1068030893*2**8,59.575229658*2**8};
//    assign prices[3] = '{127.0*2**8,24.2162177991*2**8,16.112152735*2**8,50.4074439653*2**8};
//    assign prices[4] = '{127.0*2**8,26.2766803139*2**8,14.9420497843*2**8,48.922379298*2**8};
//    assign prices[5] =
↪   '{121.714203685*2**8,32.1975154513*2**8,13.7534866892*2**8,44.4035407743*2**8};
//    assign prices[6] =
↪   '{107.831157802*2**8,33.5667962178*2**8,16.7753312399*2**8,47.6596497209*2**8};
//    assign prices[7] =
↪   '{104.862285374*2**8,36.329730527*2**8,16.6409127224*2**8,51.7086851658*2**8};
//    assign prices[8] =
↪   '{105.119810494*2**8,36.4632174103*2**8,11.8835222024*2**8,48.5692606977*2**8};

//    assign price = prices[counter];
    assign global_reset = sw[0];

    debounce db2(.reset_in(global_reset),.clock_in(clk_100mhz),.noisy_in(btnd),.clean_out(new_p));

    trading_top dut(
    .clk_100mhz(clk_100mhz),
    .latest_price(price),
    .latest_price_valid(price_valid),
    .global_reset(global_reset),
    .bus(bus),
    .global_overflow(led16_b),
    .order_done(done),
    .order_error(error)
    );

    wire [6:0] segments;
    assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];
    display_8hex display(.clk_in(clk_100mhz),.data_in(bus[31:0]), .seg_out(segments), .strobe_out(an));

    always_ff@(posedge clk_100mhz) begin

        if(global_reset == 1) begin
            //new_p <= 0;
            new_p_r <= 0;
            //bus <= 0;
            counter <= 0;

        end else if(new_p == 1 && new_p_r == 0) begin
            counter <= counter + 1;
            new_p_r <= new_p;
        end else begin
            new_p_r <= new_p;

        end

//          if(new_p && !new_p_r) begin
//              counter <= counter + 1;
//              new_p_r <= new_p;
//          end else begin
//              new_p_r <= new_p;
//          end
    end

endmodule

module trading_top(
    input clk_100mhz,
    input signed [15:0] latest_price [0:N-1],
    input latest_price_valid,
    input global_reset,
    output logic [71:0] bus,
    output logic  signed [15:0] scaled_target [0:N-1],
    output logic global_overflow,
    output logic order_done,
    output logic order_error
);

    logic new_price;

    logic curr_price_valid;
    logic signed [15:0] last_price [0:N-1];
    logic signed [15:0] curr_price [0:N-1];
    logic signed [15:0] old_moment [0:N-1][0:N-1];
    logic signed [15:0] old_mean [0:N-1];
    logic signed [15:0] T;

    logic signed [15:0] new_moment [0:N-1][0:N-1];
    logic signed [15:0] new_mean [0:N-1];
    logic signed [15:0] cov_matrix [0:N-1][0:N-1];
    logic update_cov_ready;
    logic update_cov_reset;
    logic update_cov_overflow;

    logic solver_valid;
    logic solver_reset;
    logic signed [15:0] unscaled_target [0:N-1];
    logic solver_ready;
    logic solver_overflow;

    logic normalize_ready;
    logic normalize_overflow;

    logic signed [15:0] positions [0:N-1];
    logic [3:0] stage;


    always_ff@(posedge clk_100mhz) begin
        if(global_reset) begin
            stage <= 0;
            update_cov_reset <= 1; // remember to do them at the end too
            solver_reset <= 1; //remember to do them at the end too
            curr_price_valid <= 0;
            old_moment <= '{default:0};
            old_mean <= '{default:0};
            last_price <= '{default:0};
            positions <= '{default:$signed(0.25) * $signed(2**8)};
            global_overflow <= 0;

            T <= 0;

        end else begin
            if(stage == 0) begin
                if(new_price) begin
                    stage <= 1;
                    curr_price_valid <= 1;
                    update_cov_reset <= 0;
                    global_overflow <= 0;
                end
            end else if (stage == 1) begin
                if(update_cov_overflow) begin
                    global_overflow <= 1;
                    stage <= 0;
                    update_cov_reset <= 1;

                end else if (update_cov_ready) begin
                    stage <= 2;
                    old_moment <= new_moment;
                    old_mean <= new_mean;
                    last_price <= curr_price;
                    T <= T+1;
                    update_cov_reset <= 1;
                    solver_reset <= 0;
                end

            end else if (stage == 2) begin
                if(solver_overflow) begin
                    global_overflow <= 1;
                    stage <= 0;
                    solver_reset <= 1;
                end else if (solver_ready) begin
                    stage <= 3;
                end

            end else if (stage == 3) begin
                if(normalize_overflow) begin
                    global_overflow <= 1;
                    stage <= 0;
                    solver_reset <= 1;

                end else if (normalize_ready) begin
                    stage <= 4;
                end

            end else if (stage == 4) begin
                if(order_error) begin
                    global_overflow <= 1;
                    stage <= 0;
                    solver_reset <= 1;

                end else if (order_done) begin
                    positions <= scaled_target;
                    stage <= 0;
                    solver_reset <= 1;
                end
            end
        end
    end

    get_price my_pull_price(
    .clk_100mhz(clk_100mhz),
    .reset(global_reset),
    .latest_price(latest_price),
    .latest_price_valid(latest_price_valid),
    .reg_latest_price(curr_price),
    .new_price(new_price) // this will be a pulse
    );

    update_cov my_update_cov(
    .clk_100mhz(clk_100mhz),
    .old_p(last_price),
    .new_p(curr_price),
    .old_moment(old_moment),
    .old_mean(old_mean),
    .T(T),
    .valid(curr_price_valid),
    .reset(update_cov_reset),

    .new_moment(new_moment),
    .new_cov(cov_matrix),
    .new_mean(new_mean),
    .ready(update_cov_ready),
    .overflow(update_cov_overflow)
    );

    linear_solver my_solver(
    .clk_100mhz(clk_100mhz),
    .reset(solver_reset),
    .valid(update_cov_ready && !update_cov_overflow),
    .cov(cov_matrix),
    .x(unscaled_target),
    .ready(solver_ready),
    .overflow(solver_overflow)
    );

    normalize_w my_normalize(
    .clk_100mhz(clk_100mhz),
    .reset(solver_reset),
    .valid(solver_ready && (!solver_overflow)),
    .x(unscaled_target),
    .w(scaled_target),
    .overflow(normalize_overflow),
    .ready(normalize_ready)
    );

    send_order my_sender(
    .clk_100mhz(clk_100mhz),
    .positions(positions),
    .target(scaled_target),
    .curr_price(curr_price),
    .reset(solver_reset),
    .valid(normalize_ready && (!normalize_overflow)),
    .bus(bus),
    .done(order_done),
    .error(order_error)

    );


endmodule

module linear_solver(
    input clk_100mhz,
    input reset,
    input valid,
    input signed [15:0] cov [0:N-1][0:N-1],
    output logic signed [15:0] x [0:N-1],
    output logic ready,
    output logic overflow
);


    logic signed [15:0] r [0:N-1][0:N];
    // the last column of the matrix needs be all ones

    logic signed [15:0] cov_one [0:N-1][0:N];
    logic qr_ready;
    logic solver_overflow;
    logic solver_ready;

    assign overflow = solver_overflow;
    assign ready = solver_ready;

    always_comb begin
        for(int i =0; i < N; i++) begin
            for(int j=0; j < N; j++) begin
                cov_one[i][j] = cov[i][j];
            end
            cov_one[i][N] = 2**8;
        end
    end

    qr my_qr(
    .clk_100mhz(clk_100mhz),
    .reset(reset),
    .valid(valid),
    .matrix(cov_one),
    .r(r),
    .ready(qr_ready));

    pipe_solve_r my_solver(
    .clk_100mhz(clk_100mhz),
    .reset(reset),
    .valid(qr_ready),
    .r(r),
    .x(x),
    .overflow(solver_overflow),
    .ready(solver_ready)
);

endmodule



module qr(

    input clk_100mhz,
    input reset,
    input valid,
    input signed [15:0] matrix [0:N-1][0:N],
    output logic signed [15:0] r [0:N-1][0:N],
    output logic ready
    );

    // 4 bits should be enough for everything
    parameter [3:0] LOGN = 1; // actually LOGN - 1
    parameter [3:0] STAGES = 4;
    parameter [3:0] LOGS = 1; // actually LOGSTAGE - 1
    parameter [3:0] STAGE_LEN [0:3] = {2,2,1,1};
    parameter [3:0] BAD = N+1;

    // we are going to prescribe stages

    logic signed [4:0] state;
    // 0: feed arctan
    // 1: wait arctan, put into registers
    // 2: feed rotate

    // these things need to be manually specified, ideally by a python script
    logic [LOGN:0] stage_x_row [0:3][0:1] = '{'{2'd0,2'd2},'{2'd0,2'd1},'{2'd1,2'd0},'{2'd2,2'd0}};
    logic [LOGN:0] stage_x_col [0:3][0:1] = '{'{2'd0,2'd0},'{2'd0,2'd1},'{2'd1,2'd0},'{2'd2,2'd0}};
    logic [LOGN:0] stage_y_row [0:3][0:1] = '{'{2'd1,2'd3},'{2'd2,2'd3},'{2'd2,2'd0},'{2'd3,2'd0}};
    logic [LOGN:0] stage_y_col [0:3][0:1] = '{'{2'd0,2'd0},'{2'd0,2'd1},'{2'd1,2'd0},'{2'd2,2'd0}};
    //logic [LOGN:0] stage1_x_row [0:1] = {0,1};
//    logic [LOGN:0] stage1_x_col [0:1] = {0,1};
//    logic [LOGN:0] stage1_y_row [0:1] = {2,3};
//    logic [LOGN:0] stage1_y_col [0:1] = {0,1};

    // cannot declare 0 length arrays in system verilog, we will rely on stage len to deal with this
    // for now pad this array
//    logic [LOGN:0] stage2_x_row [0:1] = {1,0};
//    logic [LOGN:0] stage2_x_col [0:1] = {1,0};
//    logic [LOGN:0] stage2_y_row [0:1] = {2,0};
//    logic [LOGN:0] stage2_y_col [0:1] = {1,0};
//    logic [LOGN:0] stage3_x_row [0:1] = {2,0};
//    logic [LOGN:0] stage3_x_col [0:1] = {2,0};
//    logic [LOGN:0] stage3_y_row [0:1] = {3,0};
//    logic [LOGN:0] stage3_y_col [0:1] = {2,0};

    logic [LOGS:0] stage;

    logic signed [15:0] arctan_tin_data_y;
    logic signed [15:0] arctan_tin_data_x;
    logic [31:0] arctan_tin_data;
    assign arctan_tin_data = {arctan_tin_data_y,arctan_tin_data_x};

    logic arctan_tin_valid;
    logic [15:0] arctan_tout_angle;
    logic arctan_tout_valid;
    logic [3:0] arctan_tin_user;
    logic [3:0] arctan_tout_user;

    cordic_1 dut1(.aclk(clk_100mhz),
        .s_axis_cartesian_tdata(arctan_tin_data),
        .s_axis_cartesian_tvalid(arctan_tin_valid),
        .s_axis_cartesian_tuser(arctan_tin_user),
        .m_axis_dout_tdata(arctan_tout_angle),
        .m_axis_dout_tvalid(arctan_tout_valid),
        .m_axis_dout_tuser(arctan_tout_user)
    );

    logic signed [15:0] rotate_tin_data_y[R-1:0];
    logic signed [15:0] rotate_tin_data_x[R-1:0];
    logic [31:0] rotate_tin_data[R-1:0];



    logic rotate_tin_valid;
    logic [15:0] rotate_tin_angle[1:0];
    logic [31:0] rotate_tout_data[1:0];
    logic rotate_tout_valid[1:0];

    logic [LOGN+1:0] rotate_tin_user[1:0];
    logic [LOGN+1:0] rotate_tout_user[1:0];

    // all the valid signals are tied: this is a synchronous design!
    // the output valid signals are supposed to be tied but we can't drive
    // the same wire with two inputs


    // can't use a for loop for this, should be okay though since we can't use a
    // for loop for all the different rotators either.
    // need to write a code generator for this

    assign rotate_tin_data[0] = {rotate_tin_data_y[0],rotate_tin_data_x[0]};
    assign rotate_tin_data[1] = {rotate_tin_data_y[1],rotate_tin_data_x[1]};


    cordic_0 rotate0
    (.aclk(clk_100mhz),
     .s_axis_cartesian_tdata(rotate_tin_data[0]),
     .s_axis_cartesian_tvalid(rotate_tin_valid),
     .s_axis_cartesian_tuser(rotate_tin_user[0]),
     .s_axis_phase_tdata(rotate_tin_angle[0]),
     .s_axis_phase_tvalid(rotate_tin_valid),
     .m_axis_dout_tdata(rotate_tout_data[0]),
     .m_axis_dout_tvalid(rotate_tout_valid[0]),
     .m_axis_dout_tuser(rotate_tout_user[0]));

  cordic_0 rotate1
    (.aclk(clk_100mhz),
     .s_axis_cartesian_tdata(rotate_tin_data[1]),
     .s_axis_cartesian_tvalid(rotate_tin_valid),
     .s_axis_cartesian_tuser(rotate_tin_user[1]),
     .s_axis_phase_tdata(rotate_tin_angle[1]),
     .s_axis_phase_tvalid(rotate_tin_valid),
     .m_axis_dout_tdata(rotate_tout_data[1]),
     .m_axis_dout_tvalid(rotate_tout_valid[1]),
     .m_axis_dout_tuser(rotate_tout_user[1]));

logic [LOGN+1:0] pair;
logic [LOGN+1:0] col;
logic done;


// first we will arctan
always_ff @(posedge clk_100mhz) begin
    if(reset) begin
        stage <= 0;
        state <= -1;
        pair <= 0;
        arctan_tin_valid <= 0;
        rotate_tin_valid <= 0;
        done <= 0;
        ready <= 0;

    end else begin
        if(done == 0) begin

            if(state == -1) begin
                if(valid == 1) begin
                    for(int i = 0; i < N; i ++) begin
                        for(int j =0; j < N+1; j++) begin
                            r[i][j] <= matrix[i][j];
                        end

                    end
                    state <= 0;
                end
            end else if(state == 0) begin
                arctan_tin_valid <= 1;
                // feed in new input every cycle
                //arctan_tin_data <= 31'b0011_0011_0000_0000_0001_0101_0000_0000;
                //arctan_tin_data <= {matrix[0][0],matrix[0][1]};

                arctan_tin_data_y <= r[stage_y_row[stage][pair]][stage_y_col[stage][pair]]
                ↪  /$signed(2);
                arctan_tin_data_x <= r[stage_x_row[stage][pair]][stage_x_col[stage][pair]]/
                ↪  $signed(2);
                arctan_tin_user <= pair;
                if(pair < STAGE_LEN[stage] - 1) begin
                    pair <= pair + 1;
                end else begin
                    state <= 4;
                    pair <= 0;
                end
            end else if (state == 4) begin
                state <= 1;
                arctan_tin_user <= BAD;
            end else if (state == 1) begin
                if(arctan_tout_user != BAD && arctan_tout_valid == 1) begin
                    // fully pipelined, should give one new output per cycle
                    rotate_tin_angle[pair] <= -arctan_tout_angle;
                    if(pair < STAGE_LEN[stage] - 1) begin
                        pair <= pair + 1;
                    end else begin
                        state <= 2;
                        pair <= 0;
                        col <= 0;
                    end
                end
            end else if (state == 2) begin
                // we are gonna rotate the entire row, even the leading zeros. this way it's easier
                ↪  to reason about.
                // in stage 0 this is gonna be true anyways
                // this doesn't give the right results! Vivado is stupid.
                //rotate_tin_data[0] <=
                ↪  {matrix[stage0_y_row[0]][col]/2,matrix[stage0_x_row[0]][col]/2};
                //rotate_tin_data[1] <=
                ↪  {matrix[stage0_y_row[1]][col]/2,matrix[stage0_x_row[1]][col]/2};

                // the rotate_tin_user of unused rotators in a stage will be forever bad. Lol.
                for(int i = 0; i < STAGE_LEN[stage]; i ++) begin
                    rotate_tin_data_x[i] <= r[stage_x_row[stage][i]][col]/ $signed(2);
                    rotate_tin_data_y[i] <= r[stage_y_row[stage][i]][col]/ $signed(2);
                    rotate_tin_user[i] <= col;
                end
//                rotate_tin_data_x[1] <= r[stage_x_row[stage][1]][col]/ $signed(2);
//                rotate_tin_data_y[1] <= r[stage_y_row[stage][1]][col]/ $signed(2);
//                rotate_tin_user[1] <= col;

                rotate_tin_valid <= 1;
                if(col < N) begin
                    col <= col + 1;
                end else begin
                    col <= 0;
                    state <= 5;
                end

            end else if (state == 5) begin
                state <= 3;
                for(int i = 0; i < STAGE_LEN[stage]; i ++) begin
                   rotate_tin_user[i] <= BAD;
                end
               //tout_valid <= 1;
            end else if (state == 3) begin

//                for(int i = 0; i < STAGE_LEN[stage]; i ++) begin
//                    tout_valid = tout_valid && (rotate_tout_valid[i] == 1);
//                      end

                    //if(rotate_tout_valid[0] == 1 && rotate_tout_valid[1] == 1 && rotate_tout_user[0]
                    ↪   != BAD && rotate_tout_user[1] != BAD) begin
                    // we are doing a dangerous thing here. We are assuming all the rotators are
                    ↪   synchronous
                    // if we want to check for each rotator's tuser we will need a for loop, hard to
                    ↪   do.
                    if(rotate_tout_valid[0] == 1 && rotate_tout_user[0] != BAD ) begin

                        // fully pipelined, should give one new output per cycle

                        for(int i = 0; i < STAGE_LEN[stage]; i ++) begin
                            r[stage_y_row[stage][i]][col] <= rotate_tout_data[i][31:16] * 2;
                            r[stage_x_row[stage][i]][col] <= -rotate_tout_data[i][15:0] * 2;
                        end
//                        r[stage_y_row[stage][1]][col] <= rotate_tout_data[1][31:16] * 2;
//                        r[stage_x_row[stage][1]][col] <= rotate_tout_data[1][15:0] * 2;
                        if(col < N ) begin
                            col <= col + 1;
                        end else begin
                            col <= 0;
                            state <= 0;
                            if(stage < STAGES -1) begin
                                stage <= stage + 1;
                            end else begin
                                done <= 1;
                                ready <= 1;
                            end
                        end
                    end

                end

            end
        end
    end

endmodule



module solve_r(
    input clk_100mhz,
    input reset,
    input valid,
    input signed [15:0] r[0:N-1][0:N],
    output logic signed [15:0] x [0:N-1],
    output logic overflow,
    output logic ready

);
    logic signed [15:0] dividend;
    logic signed [15:0] divisor;
    logic signed [31:0] c;
    logic user;
    logic dividend_valid;
    logic divisor_valid;
    logic result_valid;
    logic [3:0] tuser_in;
    logic [3:0] tuser_out;

    parameter [2:0] ROWS = 4;
    parameter [2:0] ONE_COL = 4;
    div_gen_0 divider(
    .aclk(clk_100mhz),
    .s_axis_dividend_tvalid(dividend_valid),
    .s_axis_dividend_tdata(dividend),
    .s_axis_dividend_tuser(tuser_in),
    .s_axis_divisor_tvalid(divisor_valid),
    .s_axis_divisor_tdata(divisor),
    .m_axis_dout_tdata(c),
    .m_axis_dout_tvalid(result_valid),
    .m_axis_dout_tuser(tuser_out)
    );

    logic [2:0] stage;
    logic [2:0] state;
    // 0: feed divisor
    // 1: wait divisor
    logic working;
    logic signed [15:0] cache [0:ROWS-1][0:ONE_COL];
    logic signed [15:0] result;
    logic signed [30:0] long_result;

    // we are going to make result FP[8,8]

    assign long_result = $signed(c[30:15]) * $signed(2**14) + $signed(c[14:0]);
    //assign long_result = £signed(c[30:15]) << 14 + £signed(c[14:0]);

    assign result = $signed(long_result[21:6]);

//    always_comb begin
//        if(£signed(long_result[29:14]) > 2**7-1) begin
//            £display("solver overflow");
//            //£display(£signed(long_result[29:14]));
//            //result = {£signed(127),8'b0};
//        end else if (£signed(long_result[29:14]) < -2**8) begin
//            £display("solver overflow");
//            //£display(£signed(long_result[29:14]));

//            //result = {£signed(-128),8'b0};
//        end else begin
//            £display("no overflow?");
//        end
//    end

    always_ff @(posedge clk_100mhz) begin

        if(reset==1) begin
            working <= 0;
            dividend <= 0;
            divisor <= 0;
            tuser_in <= 0;
            overflow <= 0;
            ready <= 0;
        end else if (valid==1) begin
            if(working == 0) begin
                working <= 1;
                stage <= ROWS - 1;
                state <= 0;
                for(int i = 0; i < ROWS; i ++) begin
                    for(int j = 0; j < ONE_COL + 1; j ++) begin
                        cache[i][j] <= r[i][j];
                    end
                end
            end else if (working==1) begin
                if(state == 0) begin
                    dividend <= cache[stage][ONE_COL];
                    divisor <= cache[stage][stage];
                    dividend_valid <= 1;
                    divisor_valid <= 1;
                    tuser_in <= stage;
                    state <= 1;
                    if(cache[stage][stage] == 0) begin //division by zero!
                        overflow <= 1;
                    end
                end else if (state == 1) begin
                    if(tuser_out == stage) begin
                    // change data type here! what a waste

                        for(int i = 0; i < stage; i ++) begin
                            cache[i][stage] <= 0;
                            cache[i][ONE_COL] <= cache[i][ONE_COL] - cache[i][stage] * result /
                            ↪  $signed(2 ** 8);

                            overflow <= mult_overflow(cache[i][stage], result);
                        end
                        x[stage] <= result;


                        if(stage == 0) begin
                            state <= 2;
                            ready <= 1;
                        end else begin
                            stage <= stage - 1;
                            state <= 0;
                        end
                    end
                end
            end
        end
    end

endmodule

module pipe_solve_r(
    input clk_100mhz,
    input reset,
    input valid,
    input signed [15:0] r[0:N-1][0:N],
    output logic signed [15:0] x [0:N-1],
    output logic overflow,
    output logic ready

);
    logic signed [15:0] dividend;
    logic signed [15:0] divisor;
    logic signed [31:0] c;
    logic user;
    logic dividend_valid;
    logic divisor_valid;
    logic result_valid;
    logic [3:0] tuser_in;
    logic [3:0] tuser_out;

    parameter [2:0] ROWS = 4;
    parameter [2:0] ONE_COL = 4;
    div_gen_0 divider(
    .aclk(clk_100mhz),
    .s_axis_dividend_tvalid(dividend_valid),
    .s_axis_dividend_tdata(dividend),
    .s_axis_dividend_tuser(tuser_in),
    .s_axis_divisor_tvalid(divisor_valid),
    .s_axis_divisor_tdata(divisor),
    .m_axis_dout_tdata(c),
    .m_axis_dout_tvalid(result_valid),
    .m_axis_dout_tuser(tuser_out)
    );

    logic [3:0] stage;
    logic [3:0] state;
    // 0: feed divisor
    // 1: wait divisor
    logic working;
    logic signed [15:0] cache [0:ROWS-1][0:ONE_COL];
    logic signed [15:0] cached_results ;
    logic signed [15:0] result;
    logic signed [30:0] long_result;

    // we are going to make result FP[8,8]

    assign long_result = $signed(c[30:15]) * $signed(2**14) + $signed(c[14:0]);
    //assign long_result = £signed(c[30:15]) << 14 + £signed(c[14:0]);

    assign result = $signed(long_result[21:6]);

    always_ff @(posedge clk_100mhz) begin

        if(reset==1) begin
            working <= 0;
            dividend <= 0;
            divisor <= 0;
            tuser_in <= 0;
            overflow <= 0;
            ready <= 0;
        end else if (valid==1) begin
            if(working == 0) begin
                working <= 1;
                stage <= ROWS - 1;
                state <= 0;
                for(int i = 0; i < ROWS; i ++) begin
                    for(int j = 0; j < ONE_COL + 1; j ++) begin
                        cache[i][j] <= r[i][j];
                    end
                end
            end else if (working==1) begin
                if(state == 0) begin
                    dividend <= cache[stage][ONE_COL];
                    divisor <= cache[stage][stage];
```

```
                    dividend_valid <= 1;
                    divisor_valid <= 1;
                    tuser_in <= stage;
                    state <= 1;
                    if(cache[stage][stage] == 0) begin //division by zero!
                        overflow <= 1;
                    end
                end else if (state == 1) begin
                    if(tuser_out == stage) begin
                    // change data type here! what a waste

                        cached_results <= result;
                            //cache[i][ONE_COL] <= cache[i][ONE_COL] - cache[i][stage] * result /
                            ↪  £signed(2 ** 8);

                        x[stage] <= result;

                        if(stage == 0) begin
                            state <= 3;
                            ready <= 1;
                        end else begin
                            state <= 2;
                        end
                    end
                end else if (state == 2) begin

//                  for(int i = 0; i < stage; i ++) begin
//                      cache[i][stage] <= 0;

//                      cache[i][ONE_COL] <= cache[i][ONE_COL] - cache[i][stage] * cached_results /
↪   £signed(2 ** 8);

//                      overflow <= mult_overflow(cache[i][stage], cached_results[i]);
//                  end

                    case(stage)

                        1: begin
                            cache[0][1] <= 0;
                            cache[0][ONE_COL] <= cache[0][ONE_COL] - cache[0][1] * cached_results /
                            ↪  $signed(2 ** 8);
                            overflow <= mult_overflow(cache[0][1], cached_results[0]);
                        end
                        2: begin
                            cache[0][2] <= 0;
                            cache[0][ONE_COL] <= cache[0][ONE_COL] - cache[0][2] * cached_results /
                            ↪  $signed(2 ** 8);
                            overflow <= mult_overflow(cache[0][2], cached_results[0]);
                             cache[1][2] <= 0;
                            cache[1][ONE_COL] <= cache[1][ONE_COL] - cache[1][2] * cached_results /
                            ↪  $signed(2 ** 8);
                            overflow <= mult_overflow(cache[1][2], cached_results[1]);
                        end
                        3: begin
                            cache[0][3] <= 0;
                            cache[0][ONE_COL] <= cache[0][ONE_COL] - cache[0][3] * cached_results /
                            ↪  $signed(2 ** 8);
                            overflow <= mult_overflow(cache[0][3], cached_results[0]);
                             cache[1][3] <= 0;
```

```
                                cache[1][ONE_COL] <= cache[1][ONE_COL] - cache[1][3] * cached_results /
                                ↪  $signed(2 ** 8);
                                overflow <= mult_overflow(cache[1][3], cached_results[1]);
                                cache[2][3] <= 0;
                                cache[2][ONE_COL] <= cache[1][ONE_COL] - cache[2][3] * cached_results /
                                ↪  $signed(2 ** 8);
                                overflow <= mult_overflow(cache[2][3], cached_results[2]);
                            end

                        endcase

                        state <= 0;
                        stage <= stage - 1;

                    end
                end
            end
        end

endmodule

module normalize_w(
    input clk_100mhz,
    input reset,
    input valid,
    input signed [15:0] x [0:N-1],
    output signed [15:0] w [0:N-1],
    output logic overflow,
    output logic ready
);

    logic signed [15:0] sum;
    always_comb begin
        sum = 0;
        overflow = 0;
        for(int i = 0; i < N; i ++) begin
            sum = sum + x[i];
            overflow = overflow || add_overflow(sum,x[i]);
        end
    end

    logic signed [15:0] dividend;
    logic signed [15:0] divisor;
    logic signed [31:0] c;
    logic dividend_valid;
    logic divisor_valid;
    logic result_valid;
    logic [3:0] tuser_in;
    logic [3:0] tuser_out;

    logic working;

    vector_division vec_div(
    .clk_100mhz(clk_100mhz),
    .reset(reset),
    .valid(valid),
    .a(x),
    .scale(1),
    .f(sum),
    .sel(0),
    .d(w),
    .ready(ready)
);


endmodule

\\constants.sv

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/06/2019 05:25:11 PM
// Design Name:
// Module Name: constants
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////

`ifndef CONSTANTS
`define CONSTANTS

parameter N = 4; // number of rows/cols
parameter LOGN = 2;
parameter R = 2; // number of rotators

parameter NUM_STOCKS = 1;
parameter NUM_STOCK_INDEX = NUM_STOCKS - 1;
parameter STOCK_INDEX = 1;
parameter MAX_INDEX = 100;
parameter NUM_PRICES = 200;

//3 index
parameter PRICE_INDEX = 15;
parameter ORDER_INDEX = 7;
parameter QUANTITY_INDEX = 7;

parameter TOTAL_BITS = PRICE_INDEX +1 + ORDER_INDEX + 1 + QUANTITY_INDEX + 1;

parameter ADDRESS_INDEX = 7;

parameter ENTRY_INDEX = TOTAL_BITS - 1;
parameter BRAM_LATENCY = 2;
parameter SIZE_INDEX = 8;
parameter CANCEL_UPDATE_INDEX = 2;

parameter MAX = 1;

parameter BUY_SIDE = 1; //interested in max price
parameter SELL_SIDE = 0; // interested in min price


parameter ADD_ORDER = 3'b001;
parameter CANCEL_ORDER = 3'b000;
parameter EXECUTE_ORDER = 3'b010;






typedef struct packed {
 logic [PRICE_INDEX:0] price;
 logic [ORDER_INDEX:0] order_id;
 logic [QUANTITY_INDEX:0] quantity;
} book_entry;





typedef struct packed {
  logic [ADDRESS_INDEX:0] addr;
  logic is_write;
  logic start;
} mem_struct;

typedef struct packed {
  book_entry first;
}read_result;
 `endif
```

order_book.sv

```
`timescale 1ns / 1ps
//`default_nettype none
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/19/2019 07:14:49 PM
// Design Name:
// Module Name: order_book
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
```

```
//////////////////////////////////////////////////////////////////////////////////


`include "constants.sv"


module memory_manager_for_tree_based_rep(
    input clk_in,
    input logic start,
    input logic is_write,
    input logic [ADDRESS_INDEX:0] addr,
    input book_entry data_i,
    output book_entry data_o,
    output logic valid);


    logic [10:0] counter = 0;

    logic write;
    localparam WAITING = 0;
    localparam STARTED = 1;

    logic [2:0] state = WAITING;
    logic enable = 0;

    blk_mem_gen_0 mem (.clka(clk_in), .addra(addr), .douta(data_o), .dina(data_i), .ena(enable),
    ↪  .wea(write));
    //ila_0 my_ila(.clk(clk_in), .probe0(start_book), .probe1(best_price_o), .probe2(current_size));
    always_ff @(posedge clk_in) begin
        case(state)
            WAITING: begin
                valid <= 0;
                if(start) begin
                    write <= is_write;
                    state <= STARTED;
                    counter <= 1;
                    enable <= 1;
                end

            end
            STARTED: begin
                if(counter < BRAM_LATENCY + 1) begin
                    counter <= counter + 1;
                end
                else begin
                    state <= WAITING;
                    counter <= 0;
                    valid <= 1;
                    enable <= 0;
                    write <= 0;
                end
             end
         endcase

    end

endmodule
```

```
module memory_manager(
    input clk_in,
    input logic start,
    input logic is_write,
    input logic [ADDRESS_INDEX:0] addr,
    input book_entry data_i,
    output book_entry data_o,
    output logic valid);


    logic [10:0] counter = 0;

    logic write;
    localparam WAITING = 0;
    localparam STARTED = 1;

    logic [2:0] state = WAITING;
    logic enable = 0;

    blk_mem_gen_0 mem (.clka(clk_in), .addra(addr), .douta(data_o), .dina(data_i), .ena(enable),
    ↪  .wea(write));
    //ila_0 my_ila(.clk(clk_in), .probe0(start_book), .probe1(best_price_o), .probe2(current_size));
    always_ff @(posedge clk_in) begin
        case(state)
            WAITING: begin
                valid <= 0;
                if(start) begin
                    write <= is_write;
                    state <= STARTED;
                    counter <= 1;
                    enable <= 1;
                end

            end
            STARTED: begin
                if(counter < BRAM_LATENCY) begin
                    counter <= counter + 1;
                end
                else begin
                    state <= WAITING;
                    counter <= 0;
                    valid <= 1;
                    enable <= 0;
                    write <= 0;
                end
             end
         endcase

    end

endmodule




module order_book #(parameter IS_MAX = MAX) (
        input clk_in,
        input rst_in,
        input book_entry order_to_add,
        input start_book,
        input delete,
        input [2:0] request,
        input [ORDER_INDEX:0] order_id, //should be supplied for a cancel / trade
        input [QUANTITY_INDEX:0] quantity,  // should be supplied for a trade
        output logic is_busy_o,
        output logic [CANCEL_UPDATE_INDEX:0] cancel_update,
        output logic [PRICE_INDEX:0] best_price_o,
        output logic best_price_valid,
        output logic [SIZE_INDEX:0] size_book
    );

    localparam MAX_INDEX = 0;



    localparam START = 0;
    localparam PROGRESS = 1;

    logic start;
    logic [ADDRESS_INDEX:0] addr;

    logic [ADDRESS_INDEX:0] add_addr;



    logic [PRICE_INDEX:0] best_price = 0;

    assign best_price_o = best_price;
    logic [PRICE_INDEX:0] add_best_price;
    logic [PRICE_INDEX:0] decrease_best_price;

    book_entry data_i;
    book_entry data_o;

    book_entry add_data_w;
    book_entry decrease_w;

    logic mem_start;



    logic valid;
    logic is_write;

    logic is_write_add;
    logic add_start;
    logic decrease_start;
    logic add_ready;
    logic decrease_ready;
    logic add_mem_start;


    logic valid_mem;
    logic units_busy;

    logic [QUANTITY_INDEX:0] price_distr [0:MAX_INDEX];

    localparam WAITING_STATE= 2'b00;
    localparam PROGRESS_STATE = 2'b01;
```

```
logic [2:0] request_latched = 0;
memory_manager book_mem(.clk_in(clk_in), .start(mem_start), .is_write(is_write), .addr(addr),
↪  .data_i(data_i), .data_o(data_o), .valid(valid_mem));




 logic is_busy = 0;
assign is_busy_o = is_busy;
 logic [SIZE_INDEX:0] current_size = 0;
 logic [SIZE_INDEX:0] add_size;
 logic [SIZE_INDEX:0] decrease_size;

assign size_book = current_size;
assign best_price_valid = current_size > 0;

add_order order_adder(.clk_in(clk_in), .order(order_to_add), .start(add_start), .valid(valid_mem),
↪  .price_valid(best_price_valid),
.best_price(best_price),
                      .size(current_size),.addr(add_addr), .mem_start(add_mem_start),
                      ↪  .is_write(is_write_add),
                      .ready(add_ready), .size_update_o(add_size), .data_w(add_data_w),
                      ↪  .add_best_price(add_best_price));


 read_result read_output;
 mem_struct mem_control;

 assign read_output.first = data_o;
 logic delete_actual;
decrease_order order_decreaser(.clk_in(clk_in), .id(order_id), .quantity(quantity),
↪  .delete(delete_actual),  .best_price(best_price),
                               .mem_valid(valid_mem), .size(current_size), .start(decrease_start),
                               .data_r(read_output), .mem_control(mem_control),
                               ↪  .data_w(decrease_w),
                               .ready(decrease_ready), .size_update_o(decrease_size),
                               ↪  .update(cancel_update));


 always_comb begin
    if(is_busy ) begin
        case(request_latched)
            ADD_ORDER: begin
                addr = add_addr;
                is_write = is_write_add;
                mem_start = add_mem_start;
                data_i =  add_data_w;
                units_busy = !add_ready;
            end
            //R, EXECUTE_ORDER:
            CANCEL_ORDER, EXECUTE_ORDER: begin
               addr = mem_control.addr;
               is_write = mem_control.is_write;
               mem_start = mem_control.start;
               data_i = decrease_w;
               units_busy = !decrease_ready;
            end
            default: begin
              addr = 0;
```

```
                units_busy = 0;
                is_write = 0;
                mem_start = 0;
                data_i = 0;
                units_busy = 0;
              end
          endcase
      end
      else begin
        addr = 0;
        units_busy = 0;
        is_write = 0;
        mem_start = 0;
        data_i = 0;
        units_busy = 0;
      end

  end

  logic [3:0] add_state;


  logic [1:0] add_mem_state;

  logic [1:0] current_state =  WAITING_STATE;

 //ila_0 book_ila(.clk(clk_in), .probe0(start_book), .probe1(best_price_o), .probe2(current_size));
  always_ff @(posedge clk_in) begin

      if(rst_in)  begin
          current_size <= 0;
          //add_size <= 0;
          is_busy <= 0;
          for(integer i = 0; i < MAX_INDEX; i ++) begin
             price_distr[i] <= 0;
          end
        //  decrease_size <= 0;
      end
      else begin

          if(is_busy) begin
              add_start <= 0;
              decrease_start <= 0;
              if(!units_busy) begin
                 is_busy <= 0;
                 case(request_latched)
                    ADD_ORDER: begin
                       current_size <= add_size;
                       best_price <= add_best_price;
                    end
                    CANCEL_ORDER, EXECUTE_ORDER: begin
                       current_size <= decrease_size;
                    end

                 endcase
              end
          end
          else begin
              if(start_book) begin
                  request_latched <= request;
```

```
                    is_busy <= 1;
                    case(request)
                        ADD_ORDER: begin
                            add_start <= 1;
                              price_distr[order_to_add.price] <= price_distr[order_to_add.price] +
                               ↪  order_to_add.quantity;
                        end
                        CANCEL_ORDER: begin
                            decrease_start <= 1;
                            delete_actual <= 1;
                         end
                        EXECUTE_ORDER: begin
                           decrease_start <= 1;
                           delete_actual <= 0;
                        end
                    endcase
                end
        end
     end


    end



endmodule





module add_order#(parameter IS_MAX=MAX)
               (input clk_in, input book_entry order, input start, input valid,
                input   [QUANTITY_INDEX:0] price_distr ,
                input [SIZE_INDEX:0] size, input [PRICE_INDEX:0] best_price, input price_valid,
                output logic [ADDRESS_INDEX:0] addr,  output logic mem_start,
                output book_entry data_w,
                output logic price_update,
                output logic quantity_update,
                output quantity,
                output logic is_write, output logic ready, output logic [SIZE_INDEX:0] size_update_o,
                output logic  [PRICE_INDEX:0] add_best_price );


    logic [1:0] add_mem_state = 0;

    localparam START = 0;
    localparam PROGRESS = 1;
    logic [SIZE_INDEX:0] size_update = 0;


    assign size_update_o = size_update;


    always_ff @(posedge clk_in) begin
       case(add_mem_state)
       START: begin
            ready <= 0;
            if(start) begin
                if(size < MAX_INDEX) begin
                    addr <= size;
                    is_write <= 1;
                    size_update <= size + 1;
                    add_mem_state <= PROGRESS;
                    data_w <= order;
                    mem_start <= 1;
                    price_update <= order.price;
                    quantity_update <= order.quantity;
                    if(!price_valid || (order.price > best_price) == IS_MAX) begin
                       add_best_price <= order.price;
                    end
                    else begin
                        add_best_price <= best_price;
                    end

                 end
            end
        end
        PROGRESS: begin
            mem_start <= 0;
            if(valid) begin
                ready <= 1;
                add_mem_state <= START;
            end
        end
    endcase
     end


endmodule




module decrease_order#(parameter SIDE=BUY_SIDE)
               (input clk_in,
                input logic [ORDER_INDEX:0] id,
                input [QUANTITY_INDEX:0] quantity,
                input [PRICE_INDEX:0] best_price,
                input delete,
                input mem_valid,
                input [SIZE_INDEX:0] size,
                input start,
                input read_result data_r,
                output  mem_struct mem_control,
                output book_entry data_w,
                output logic ready,
                output logic [SIZE_INDEX:0] size_update_o,
                output logic[CANCEL_UPDATE_INDEX:0] update
    );
    logic [SIZE_INDEX:0] index;

    logic [SIZE_INDEX:0] size_latched;

    localparam WAITING = 3'b000;
    localparam FIND = 3'b001;
    localparam DELETE = 3'b010;
    localparam UPDATE = 3'b110;
    localparam DONE = 3'b011;
    localparam NOT_FOUND = 3'b111;
    localparam COPY = 2'b00;
    localparam MOVE = 2'b01;


    localparam MEM_IDLE = 0 ;
    localparam MEM_PROGRESS = 1;
    logic [2:0] mem_state = MEM_IDLE;



    logic [2:0] state = WAITING;

    logic [SIZE_INDEX:0] update_index;
    logic [2:0] delete_state;
    logic [QUANTITY_INDEX:0] quantity_latched;
    logic delete_latched;
    book_entry copy_entry;


    assign size_update_o = size_latched;

    //need start signal, entry in order book, addr, delete
//    ila_1 my_ila(.clk(clk_in), .probe0(0), .probe1(data_r), .probe2(mem_valid), .probe3(start),
↪   .probe4(mem_control.addr));

    always_ff @(posedge clk_in) begin
          case (state)

            WAITING: begin
                data_w <= 0;
                update <= WAITING;
                if(start) begin
                    index <= 0;
                    state <= FIND;
                    mem_state <= MEM_IDLE;
                    size_latched <= size;
                    quantity_latched <= quantity;
                    delete_latched <= delete;
                    ready <= 0;
                end
                else begin
                    ready <= 0;
                end
            end
            UPDATE: begin
                    case(mem_state)
                        MEM_IDLE: begin
                          mem_control.addr <= update_index;
                          mem_control.is_write <= 1;
                          mem_control.start <= 1;
                          data_w <= '{price:copy_entry.price, order_id:copy_entry.order_id,
                           ↪  quantity:copy_entry.quantity - quantity_latched};
                          mem_state <= MEM_PROGRESS;
                        end
                        MEM_PROGRESS: begin
                           mem_control.start <= 0;
                            if(mem_valid) begin
                                mem_state <= MEM_IDLE;
                                state <= WAITING;
                                update <= UPDATE;
                                ready <= 1;
                            end
                       end
                    endcase

               end
            FIND: begin
              case(mem_state)
                  MEM_IDLE:  begin
                    if(index < size_latched) begin
                        mem_control <= '{addr: index, is_write:0, start: 1};
                        mem_state <= MEM_PROGRESS;
                    end
                    else begin
                       state <= WAITING;
                       update <= NOT_FOUND;
                       ready <= 1;

                    end
                  end
                  MEM_PROGRESS: begin
                   mem_control.start <= 0;
                    if(mem_valid) begin
                       mem_state <= MEM_IDLE;

                        if(data_r.first.order_id == id) begin
                            update_index <= index;
                            if(data_r.first.quantity <= quantity || delete_latched) begin
                                state <= DELETE;
                                delete_state <= COPY;
                            end
                            else begin
                               state <= UPDATE;
                               copy_entry <= data_r.first;
                            end
                        end
                        else begin
                            index <= index + 1;
                        end


                    end
                  end
              endcase
            end
            DELETE: begin
              case(delete_state)

                COPY: begin
                    case(mem_state)
                      MEM_IDLE: begin
                        if(update_index + 1 < size_latched) begin

                          mem_control.addr <= update_index + 1;
                          mem_control.is_write <= 0;
                          mem_control.start <= 1;
                          mem_state <= MEM_PROGRESS;
                        end
                        else begin
                           size_latched <= size_latched - 1;
```

```
                            state <= WAITING;
                            ready <= 1;
                            update <= DELETE;
                         end
                        end

                       MEM_PROGRESS: begin
                          mem_control.start <= 0;
                         if(mem_valid) begin
                             copy_entry <= data_r.first;

                             delete_state <= MOVE;
                             mem_state <= MEM_IDLE;

                         end
                       end
                     endcase
                end
                MOVE: begin
                     case(mem_state)
                         MEM_IDLE: begin

                           mem_control.addr <= update_index;
                           mem_control.is_write <= 1;
                           mem_control.start <= 1;
                           data_w <= copy_entry;
                           mem_state <= MEM_PROGRESS;

                         end
                         MEM_PROGRESS: begin
                            mem_control.start <= 0;
                             if(mem_valid) begin
                                 mem_state <= MEM_IDLE;
                                 delete_state <= COPY;
                                 update_index <= update_index + 1;
                             end
                        end
                     endcase
                 end

              endcase


            end

           endcase

    end

endmodule

#order_book_top_level.sv

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/02/2019 06:14:07 PM
// Design Name:
// Module Name: top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

`include "constants.sv"



module debounce (input reset_in, clock_in, noisy_in,
                 output reg clean_out);

   reg [19:0] count;
   reg new_input;

   always_ff @(posedge clock_in)
     if (reset_in) begin
        new_input <= noisy_in;
        clean_out <= noisy_in;
        count <= 0; end
     else if (noisy_in != new_input) begin new_input<=noisy_in; count <= 0; end
     else if (count == 1000000) clean_out <= new_input;
     else count <= count+1;


endmodule


module top_level_simulation(input clk_in, input next, input reset, output busy,
    output logic[PRICE_INDEX:0] best_price_stocks  [0:NUM_STOCK_INDEX] ,
    output logic stocks_valid
);
    logic [STOCK_INDEX:0] stock_to_add;
    book_entry entry;
    reg start;
    logic [2:0] request;
    logic delete;
    logic is_busy;
    logic [ORDER_INDEX:0] order_id;
    logic [QUANTITY_INDEX:0] quantity;





 order_book_wrapper top_(
        .clk_in(clk_in),
        .rst_in(reset),
        .stock_to_add(stock_to_add),
        .order_to_add(entry),
        .start(start),
        .request(request),
        .order_id(order_id),
        .delete(delete), //should be one if cancel
        .quantity(quantity),
        .is_busy(is_busy),
        .best_price_stocks(best_price_stocks),
        .best_prices_valid(stocks_valid));   // should be supplied for a trade


       localparam FIRST = 0;
       localparam END = 5;


       localparam WAIT_FIRST = 2'b00;
       localparam WAIT_NEXT = 2'b01;
       localparam PROGRESS =  2'b11;
       logic [1:0] top_level = WAIT_NEXT;

       logic [7:0] state = 0;

       reg pick_next_previous;
       logic [PRICE_INDEX:0] best_price;
       assign best_price = best_price_stocks[0];
      ila_0 top_level_ila(.clk(clk_in), .probe0(best_price), .probe1(top_level), .probe2(state));

       logic change_signal;

       assign busy = (top_level == PROGRESS) || is_busy;

       logic  [PRICE_INDEX:0]  d_array_ [0:END] [0:NUM_STOCK_INDEX] = '{'{  100*2**8, 106*2**8,
        ↪  107*2**8, 108*2**8},
                                                                     '{ 101*2**8, 107*2**8, 108*2**8,
                                                                     ↪  109*2**8},
                                                                      '{104*2**8, 110*2**8, 109*2**8,
                                                                      ↪  110*2**8},
                                                                      '{106*2**8, 111*2**8, 111*2**8,
                                                                      ↪  114*2**8},
                                                                      '{110*2**8, 113*2**8, 114*2**8,
                                                                      ↪  116*2**8},
                                                                      '{110*2**8, 113*2**8, 114*2**8,
                                                                      ↪  116*2**8}};
       logic  [PRICE_INDEX:0]  d_array [1:0] [NUM_STOCK_INDEX:0] = '{'{
        ↪  100.0*2**8,40.0*2**8,50.0*2**8,60.0*2**8},
                                                                       '{
                                                                       ↪  115.355298916*2**8,40.9465656921*2**

       logic  [PRICE_INDEX:0]  price_ [NUM_STOCK_INDEX:0]= { 100.0*2**8,40.0*2**8,50.0*2**8,60.0*2**8};

      logic [3:0]  stock_index = 0;
      logic [3:0] price_index = 0;
      parameter INITIAL = 0;
      parameter ADD = 1;
      parameter DELETE = 2;
      always_ff @(posedge clk_in) begin
         case(top_level)
         PROGRESS: begin
           case(state)
                ADD: begin
                    stock_to_add <= stock_index;
                    start <= 1;
```

```
            entry <= '{price:d_array_[price_index][stock_index], order_id:2, quantity:2};
            request <= ADD_ORDER;
            top_level <= WAIT_FIRST;
        end
      /*
       2: begin
            stock_to_add <= 0;
            start <= 1;
            entry <= '{price:4, order_id:3, quantity:2};

            request <= ADD_ORDER;
           top_level <= WAIT_FIRST;

         end
       3: begin
          stock_to_add <= 0;
            start <= 1;
            order_id <= 3;
            delete <= 1;
            request <= CANCEL_ORDER;
           top_level <= WAIT_FIRST;
       end  */


    endcase
end
WAIT_FIRST: begin
   top_level <= WAIT_NEXT;

end

WAIT_NEXT: begin

  start <= 0;

   if(!is_busy && price_index <= END )  begin
       if(next && (state == INITIAL)) begin
        stock_index <= 0;
        state <= ADD;
        top_level <= PROGRESS;
       end
       else if(stock_index < NUM_STOCK_INDEX) begin
          stock_index <= stock_index + 1;
          state <= ADD;
          top_level <= PROGRESS;
       end
      else if(next && price_index < END) begin

          stock_index <= 0;
          state <= ADD;
          price_index <= price_index + 1;
          top_level <= PROGRESS;
        end

   end
end
endcase

end
```

```
endmodule


module top_level_fpga_tester(input clk_in, input next, input reset, output busy);
    logic [STOCK_INDEX:0] stock_to_add;
    book_entry entry;
    reg start;
    logic [2:0] request;
    logic delete;
    logic is_busy;
    logic [ORDER_INDEX:0] order_id;
    logic [QUANTITY_INDEX:0] quantity;




    logic[PRICE_INDEX:0] best_price_stocks  [NUM_STOCK_INDEX:0];
 order_book_wrapper top_(
        .clk_in(clk_in),
        .rst_in(reset),
        .stock_to_add(stock_to_add),
        .order_to_add(entry),
        .start(start),
        .request(request),
        .order_id(order_id),
        .delete(delete), //should be one if cancel
        .quantity(quantity),
        .is_busy(is_busy),
        .best_price_stocks(best_price_stocks));   // should be supplied for a trade


       localparam FIRST = 0;
       localparam END = 8;


       localparam WAIT_FIRST = 2'b00;
       localparam WAIT_NEXT = 2'b01;
       localparam PROGRESS =  2'b11;
       logic [1:0] top_level = WAIT_NEXT;
       logic [7:0] state = 0;

       reg pick_next_previous;
       logic [PRICE_INDEX:0] best_price;
       assign best_price = best_price_stocks[1];
       ila_0 top_level_ila(.clk(clk_in), .probe0(best_price), .probe1(top_level), .probe2(state));

       logic change_signal;

       assign busy = (top_level == PROGRESS) || is_busy;

      always_ff @(posedge clk_in) begin
         case(top_level)
         PROGRESS: begin
           case(state)
                1: begin
                    stock_to_add <= 0;
                    start <= 1;
                    entry <= '{price:2, order_id:2, quantity:2};
                    request <= ADD_ORDER;
                    top_level <= WAIT_FIRST;
                end

               2: begin
                    stock_to_add <= 0;
                    start <= 1;
                    entry <= '{price:4, order_id:3, quantity:2};

                    request <= ADD_ORDER;
                   top_level <= WAIT_FIRST;

                 end
               3: begin
                   stock_to_add <= 0;
                    start <= 1;
                    entry <= '{price:4, order_id:4, quantity:2};
                    request <= ADD_ORDER;
                   top_level <= WAIT_FIRST;

               end
               4: begin
                    stock_to_add <= 0;
                    start <= 1;
                    order_id <= 4;
                    delete <= 1;
                    request <= CANCEL_ORDER;
                   top_level <= WAIT_FIRST;
               end
               5: begin
                    stock_to_add <= 0;
                    start <= 1;
                    order_id <= 3;
                    delete <= 1;
                    request <= CANCEL_ORDER;
                   top_level <= WAIT_FIRST;

               end



            endcase
        end
        WAIT_FIRST: begin

           top_level <= WAIT_NEXT;

        end
        WAIT_NEXT: begin

          start <= 0;

           if(!is_busy && state < END && next)  begin
               state <= state + 1;
               top_level <= PROGRESS;
           end
        end
        endcase

       end

endmodule
```

```
module top_level(input clk_100mhz, input btnu, input [15:0] sw);

       logic pick_next;
       debounce d1(.clock_in(clk_100mhz), .reset_in(sw[15]), .noisy_in(btnu), .clean_out(pick_next));
       logic pick_next_previous;
       logic change;
       always_ff @(posedge clk_100mhz) begin
           pick_next_previous <= pick_next;
           change <= pick_next_previous == 0 && pick_next == 1;
       end

       logic busy;
       top_level_fpga_tester fpga_tester(.clk_in(clk_100mhz), .next(change), .reset(sw[15]),
        ↪  .busy(busy));
endmodule






module top_level_fpga_tester_demo(input clk_in, input next, input reset, output busy);
    logic [STOCK_INDEX:0] stock_to_add;
    book_entry entry;
    reg start;
    logic [2:0] request;
    logic delete;
    logic is_busy;
    logic [ORDER_INDEX:0] order_id;
    logic [QUANTITY_INDEX:0] quantity;



        localparam  num_prices = 1;
        logic [31:0] price_index;


       logic  [PRICE_INDEX:0]  d_array [num_prices:0] [NUM_STOCK_INDEX:0] = '{'{101, 102, 103, 104},
                                                                           '{ 105, 107, 108, 110}};

    logic[PRICE_INDEX:0] best_price_stocks  [NUM_STOCK_INDEX:0]    ;
 order_book_wrapper top_(
        .clk_in(clk_in),
        .rst_in(reset),
        .stock_to_add(stock_to_add),
        .order_to_add(entry),
        .start(start),
        .request(request),
        .order_id(order_id),
        .delete(delete), //should be one if cancel
        .quantity(quantity),
        .is_busy(is_busy),
        .best_price_stocks(best_price_stocks));   // should be supplied for a trade


       localparam FIRST = 0;
       localparam END = 8;
       logic price_valid = 0;

       localparam WAIT_FIRST = 2'b00;
       localparam WAIT_NEXT = 2'b01;
       localparam PROGRESS =  2'b11;
       logic [1:0] top_level = WAIT_NEXT;
       logic [7:0] state = 0;

       reg pick_next_previous;
       logic [PRICE_INDEX:0] best_price;
       assign best_price = best_price_stocks[1];
       ila_0 top_level_ila(.clk(clk_in), .probe0(best_price), .probe1(top_level), .probe2(state));

       logic change_signal;

       assign busy = (top_level == PROGRESS) || is_busy;

       localparam events = 4;
      always_ff @(posedge clk_in) begin
         case(top_level)
         PROGRESS: begin
           case(state)
                1: begin
                    stock_to_add <= 0;

                    start <= 1;
                    entry <= '{price:2, order_id:2, quantity:2};
                    request <= ADD_ORDER;
                    top_level <= WAIT_FIRST;
                end
               2: begin
                    stock_to_add <= 1;
                    start <= 1;
                    entry <= '{price:4, order_id:2, quantity:2};

                    request <= ADD_ORDER;
                    top_level <= WAIT_FIRST;

                 end
               3:begin
                    stock_to_add <= 0;
                    start <= 1;
                    order_id <= 2;
                    delete <= 1;
                    request <= CANCEL_ORDER;
                   top_level <= WAIT_FIRST;
                  end

               4: begin
                    stock_to_add <= 1;
                    start <= 1;
                    order_id <= 2;
                    delete <= 1;
                    request <= CANCEL_ORDER;
                   top_level <= WAIT_FIRST;

               end

            endcase
        end
```

```
        WAIT_FIRST: begin

           top_level <= WAIT_NEXT;

        end
        WAIT_NEXT: begin

          start <= 0;

           if(!is_busy &&  next)  begin
               if(state == events) begin
                  price_index <= price_index + 1;
                  price_valid <= 1;
               end
               else begin
                    state <= state + 1;
                    price_valid <= 0;
               end
               top_level <= PROGRESS;
           end
        end
        endcase

       end

endmodule


module order_book_wrapper(
        input clk_in,
        input rst_in,
        input [STOCK_INDEX:0] stock_to_add,
        input book_entry order_to_add,
        input start,
        input delete,
        input [QUANTITY_INDEX:0] quantity,
        input [2:0] request,
        input [ORDER_INDEX:0] order_id, //should be supplied for a cancel / trade
        output logic is_busy,
        output logic best_price_valid,
        output logic [CANCEL_UPDATE_INDEX:0] cancel_update,
        output logic [PRICE_INDEX:0] best_price_stocks [0:NUM_STOCK_INDEX],
        output logic [0:NUM_STOCK_INDEX] best_prices_valid,
        output logic [SIZE_INDEX:0] size_of_stocks [0:NUM_STOCK_INDEX]
    );
    logic [NUM_STOCK_INDEX:0] order_book_start;
    logic [NUM_STOCK_INDEX:0] book_busy;


    logic [STOCK_INDEX:0] stock_latched;
    localparam WAITING = 2'b00;
    localparam INITATE = 2'b01;
    localparam PROGRESS = 2'b10;



    assign best_price_valid = &best_prices_valid;

     logic [2:0] state = WAITING;
         assign is_busy = state != WAITING;
     genvar i;


     generate
        for(i = 0; i < 4; i = i+1)

        begin
          order_book #(.IS_MAX(MAX))  book(
        .clk_in(clk_in),
        .rst_in(rst_in),
        .order_to_add(order_to_add),
        .request(request),
        .start_book(order_book_start[i]),
        .order_id(order_id),
        .delete(delete),
        .quantity(quantity),
        .is_busy_o(book_busy[i]),
        .best_price_o(best_price_stocks[i]),
        .best_price_valid(best_prices_valid[i]),
        .size_book(size_of_stocks[i])
    );
        end
    endgenerate



  //  ila_0 wrapper_ila(.clk(clk_in), .probe0(stock_to_add), .probe1(start), .probe2(state));
     always_ff @(posedge clk_in) begin
        case(state)
            WAITING: begin
               if(start) begin
                 if(stock_to_add < NUM_STOCKS) begin
                   state <= INITATE;
                   stock_latched <= stock_to_add;
                   //stock to add from 0 to 1
                   order_book_start[stock_to_add] <= 1;
                 end
                else begin

                end
               end

            end
            INITATE: begin
               state <= PROGRESS;
               order_book_start[stock_to_add] <= 0;
            end
            PROGRESS: begin
              if(!book_busy[stock_latched]) begin
                state <= WAITING;
              end
            end
        endcase



     end

endmodule
```

\\tree_based_rep.sv

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/11/2019 06:31:46 PM
// Design Name:
// Module Name: tree_based_rep
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

`include "constants.sv"

module tree_based_rep_order_book #(parameter IS_MAX = MAX) (
        input clk_in,
        input rst_in,
        input book_entry order_to_add,
        input start_book,
        input delete,
        input [2:0] request,
        input [ORDER_INDEX:0] order_id, //should be supplied for a cancel / trade
        input [QUANTITY_INDEX:0] quantity,  // should be supplied for a trade
        output logic is_busy_o,
        output logic [CANCEL_UPDATE_INDEX:0] cancel_update,
        output logic [PRICE_INDEX:0] best_price_o,
        output logic best_price_valid,
        output logic price_valid,
        output logic [SIZE_INDEX:0] size_book
    );






    localparam START = 0;
    localparam PROGRESS = 1;

    logic start;
    logic [ADDRESS_INDEX:0] addr;

    logic [ADDRESS_INDEX:0] add_addr;



    logic [PRICE_INDEX:0] best_price = 0;
```

```
assign best_price_o = best_price;
logic [PRICE_INDEX:0] add_best_price;
logic [PRICE_INDEX:0] decrease_best_price;

book_entry data_i;
book_entry data_o;

book_entry add_data_w;
book_entry decrease_w;

logic mem_start;



logic valid;
logic is_write;

logic is_write_add;
logic add_start;
logic decrease_start;
logic add_ready;
logic decrease_ready;
logic add_mem_start;


logic valid_mem;
logic units_busy;

logic [QUANTITY_INDEX:0] price_distr [0:MAX_INDEX];

tree_based_rep order_space [0:MAX_INDEX];

localparam WAITING_STATE= 2'b00;
localparam PROGRESS_STATE = 2'b01;

logic [2:0] request_latched = 0;


//memory_manager_for_tree_based book_mem(.clk_in(clk_in), .start(mem_start), .is_write(is_write),
↪  .addr(addr), .data_i(data_i), .data_o(data_o), .valid(valid_mem));


logic [PRICE_INDEX:0] price_tree [0:7][0:100];


 logic is_busy = 0;
assign is_busy_o = is_busy;
 logic [SIZE_INDEX:0] current_size = 0;
 logic [SIZE_INDEX:0] add_size;
 logic [SIZE_INDEX:0] decrease_size;

assign size_book = current_size;
assign best_price_valid = current_size > 0;

 read_result read_output;
 mem_struct mem_control;

 assign read_output.first = data_o;
 logic delete_actual;
 always_comb begin
    if(is_busy ) begin
        case(request_latched)
            //R, EXECUTE_ORDER:
            CANCEL_ORDER, EXECUTE_ORDER: begin
               addr = mem_control.addr;
               is_write = mem_control.is_write;
               mem_start = mem_control.start;
               data_i = decrease_w;
               units_busy = !decrease_ready;
            end
            default: begin
              addr = 0;
              units_busy = 0;
              is_write = 0;
              mem_start = 0;
              data_i = 0;
              units_busy = 0;
            end
        endcase
    end
    else begin
      addr = 0;
      units_busy = 0;
      is_write = 0;
      mem_start = 0;
      data_i = 0;
      units_busy = 0;
    end

end

logic [3:0] add_state;


logic [1:0] add_mem_state;

logic [1:0] current_state =  WAITING_STATE;
localparam MEM_IDLE = 0 ;
localparam MEM_PROGRESS = 1;
logic [2:0] mem_state = MEM_IDLE;


logic [1:0] delete_state;


logic [8:0] counter;

always_ff @(posedge clk_in) begin

    if(start_book) begin
       counter <= 1;
       price_valid <= 0;
    end

    else if(counter <= LEVEL_INDEX) begin
        counter <= counter + 1;
        end
        else begin
            price_valid <= 1;
        end


    end

    logic [32:0] price_levels [0:MAX_LEVEL_INDEX] = {1,2, 4, 8, 16, 32, 64};
    always_ff @(posedge clk_in) begin
        for(int i = 0; i < LEVEL_INDEX; i++)
        begin
           for(int j = 0; j < price_levels[i]; j++) begin

              if(price_tree[i+1][2*j+1] != 0) begin
                 price_tree[i][j] <= price_tree[i+1][2*j + 1];
              end
              else begin
                 price_tree[i][j] <= price_tree[i+1][2*j];
              end


           end
        end

        for(int j = 0; j < price_levels[LEVEL_INDEX]; j++) begin

              if(price_distr[2*j+1] != 0) begin
                 price_tree[LEVEL_INDEX][j]  <= 2*j + 1;
              end
              else if(price_distr[2*j] != 0) begin

                  price_tree[LEVEL_INDEX][j] <= 2*j;
              end
              else begin
                    price_tree[LEVEL_INDEX][j] <= 0;
              end

      end

    end
  // ila_0 book_ila(.clk(clk_in), .probe0(start_book), .probe1(best_price_o), .probe2(current_size));
    always_ff @(posedge clk_in) begin

        if(rst_in)  begin
            current_size <= 0;
            //add_size <= 0;
            is_busy <= 0;
            for(integer i = 0; i < MAX_INDEX; i ++) begin
               price_distr[i] <= 0;
               order_space[i] <= 0;
            end
          //  decrease_size <= 0;
        end
        else begin

            if(is_busy) begin
                add_start <= 0;
                decrease_start <= 0;
                if(!units_busy) begin
              is_busy <= 0;

              case(request_latched)
                 CANCEL_ORDER, EXECUTE_ORDER: begin
                    case(mem_state)

                      MEM_PROGRESS: begin
                         mem_control.start <= 0;

                          if(valid_mem) begin
                              mem_state <= MEM_IDLE;

                              //price_distr[data_r.first.order_id] <=
                              ↪  price_distr[data_r.first.order_id];

                          end
                       end
                     endcase
                 end

               endcase

           end
       end
       else begin
           if(start_book) begin
               request_latched <= request;
              // is_busy <= 1;
               case(request)
                   ADD_ORDER: begin
                    current_size <= current_size  + 1;
                      order_space[order_to_add.order_id].quantity <= order_to_add.quantity;
                         order_space[order_to_add.order_id].price <= order_to_add.price;
                      price_distr[order_to_add.price] <= price_distr[order_to_add.price] +
                       ↪  order_to_add.quantity;
                   end
                   CANCEL_ORDER, EXECUTE_ORDER: begin
                        mem_control.addr <= order_id;

                        mem_control.is_write <= 1;
                        if(order_space[order_id].quantity != 0) begin
                            current_size <= current_size - 1;
                          if(request == CANCEL_ORDER) begin
                             order_space[order_id].quantity <= 0;
                             price_distr[order_space[order_id].price] <=
                              ↪  price_distr[order_space[order_id].price] -
                              ↪  order_space[order_id].quantity;
                          end


                        end
                    end
               endcase
           end
   end
end


end
```

```
endmodule

/*
 * Copyright (C) 2009 - 2019 Xilinx, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
 * SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
 * OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
 * OF SUCH DAMAGE.
 *
 */

#include <stdio.h>

#include "xparameters.h"

#include "netif/xadapter.h"

#include "platform.h"
#include "platform_config.h"
#if defined (__arm__) || defined(__aarch64__)
#include "xil_printf.h"
#endif

#include "lwip/tcp.h"
#include "xil_cache.h"

#if LWIP_IPV6==1
#include "lwip/ip.h"
#else
#if LWIP_DHCP==1
#include "lwip/dhcp.h"
#include "fsl.h"
#endif
#endif

/* defined by each RAW mode application */
void print_app_header();
int start_application();
int transfer_data();
```

```
void tcp_fasttmr(void);
void tcp_slowtmr(void);

/* missing declaration in lwIP */
void lwip_init();

#if LWIP_IPV6==0
#if LWIP_DHCP==1
extern volatile int dhcp_timoutcntr;
err_t dhcp_start(struct netif *netif);
#endif
#endif

extern volatile int TcpFastTmrFlag;
extern volatile int TcpSlowTmrFlag;
static struct netif server_netif;
struct netif *echo_netif;

#if LWIP_IPV6==1
void print_ip6(char *msg, ip_addr_t *ip)
{
  print(msg);
  xil_printf(" %x:%x:%x:%x:%x:%x:%x:%x\n\r",
      IP6_ADDR_BLOCK1(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK2(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK3(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK4(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK5(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK6(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK7(&ip->u_addr.ip6),
      IP6_ADDR_BLOCK8(&ip->u_addr.ip6));

}
#else
void
print_ip(char *msg, ip_addr_t *ip)
{
  print(msg);
  xil_printf("%d.%d.%d.%d\n\r", ip4_addr1(ip), ip4_addr2(ip),
      ip4_addr3(ip), ip4_addr4(ip));
}

void
print_ip_settings(ip_addr_t *ip, ip_addr_t *mask, ip_addr_t *gw)
{

  print_ip("Board IP: ", ip);
  print_ip("Netmask : ", mask);
  print_ip("Gateway : ", gw);
}
#endif

#if defined (__arm__) && !defined (ARMR5)
#if XPAR_GIGE_PCS_PMA_SGMII_CORE_PRESENT == 1 || XPAR_GIGE_PCS_PMA_1000BASEX_CORE_PRESENT == 1
int ProgramSi5324(void);
int ProgramSfpPhy(void);
#endif
#endif

#ifdef XPS_BOARD_ZCU102
```

```
#ifdef XPAR_XIICPS_0_DEVICE_ID
int IicPhyReset(void);
#endif
#endif

int main()
{
#if LWIP_IPV6==0
  ip_addr_t ipaddr, netmask, gw;

#endif
  /* the mac address of the board. this should be unique per board */
  unsigned char mac_ethernet_address[] =
  { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };

  echo_netif = &server_netif;
#if defined (__arm__) && !defined (ARMR5)
#if XPAR_GIGE_PCS_PMA_SGMII_CORE_PRESENT == 1 || XPAR_GIGE_PCS_PMA_1000BASEX_CORE_PRESENT == 1
  ProgramSi5324();
  ProgramSfpPhy();
#endif
#endif

/* Define this board specific macro in order perform PHY reset on ZCU102 */
#ifdef XPS_BOARD_ZCU102
  if(IicPhyReset()) {
    xil_printf("Error performing PHY reset \n\r");
    return -1;
  }
#endif

  init_platform();

#if LWIP_IPV6==0
#if LWIP_DHCP==1
    ipaddr.addr = 0;
  gw.addr = 0;
  netmask.addr = 0;
#else
  /* initliaze IP addresses to be used */
  IP4_ADDR(&ipaddr,  192, 168,   1, 10);
  IP4_ADDR(&netmask, 255, 255, 255,  0);
  IP4_ADDR(&gw,      192, 168,   1,  1);
#endif
#endif
  print_app_header();

  lwip_init();

#if (LWIP_IPV6 == 0)
  /* Add network interface to the netif_list, and set it as default */
  if (!xemac_add(echo_netif, &ipaddr, &netmask,
            &gw, mac_ethernet_address,
            PLATFORM_EMAC_BASEADDR)) {
    xil_printf("Error adding N/W interface\n\r");
    return -1;
  }
#else
  /* Add network interface to the netif_list, and set it as default */
  if (!xemac_add(echo_netif, NULL, NULL, NULL, mac_ethernet_address,
            PLATFORM_EMAC_BASEADDR)) {
    xil_printf("Error adding N/W interface\n\r");
    return -1;
  }
  echo_netif->ip6_autoconfig_enabled = 1;

  netif_create_ip6_linklocal_address(echo_netif, 1);
  netif_ip6_addr_set_state(echo_netif, 0, IP6_ADDR_VALID);

  print_ip6("\n\rBoard IPv6 address ", &echo_netif->ip6_addr[0].u_addr.ip6);

#endif
  netif_set_default(echo_netif);

  /* now enable interrupts */
  platform_enable_interrupts();

  /* specify that the network if is up */
  netif_set_up(echo_netif);

#if (LWIP_IPV6 == 0)
#if (LWIP_DHCP==1)
  /* Create a new DHCP client for this interface.
   * Note: you must call dhcp_fine_tmr() and dhcp_coarse_tmr() at
   * the predefined regular intervals after starting the client.
   */
  dhcp_start(echo_netif);
  dhcp_timoutcntr = 24;

  while(((echo_netif->ip_addr.addr) == 0) && (dhcp_timoutcntr > 0))
    xemacif_input(echo_netif);

  if (dhcp_timoutcntr <= 0) {
    if ((echo_netif->ip_addr.addr) == 0) {
      xil_printf("DHCP Timeout\r\n");
      xil_printf("Configuring default IP of 192.168.1.10\r\n");
      IP4_ADDR(&(echo_netif->ip_addr),  192, 168,   1, 10);
      IP4_ADDR(&(echo_netif->netmask), 255, 255, 255,  0);
      IP4_ADDR(&(echo_netif->gw),      192, 168,   1,  1);
    }
  }

  ipaddr.addr = echo_netif->ip_addr.addr;
  gw.addr = echo_netif->gw.addr;
  netmask.addr = echo_netif->netmask.addr;
#endif

  print_ip_settings(&ipaddr, &netmask, &gw);

#endif
  /* start the application (web server, rxtest, txtest, etc..) */
  start_application();

  /* receive and process packets */
  while (1) {
    if (TcpFastTmrFlag) {
      tcp_fasttmr();
      TcpFastTmrFlag = 0;

    }
```

```c
    if (TcpSlowTmrFlag) {
      tcp_slowtmr();
      TcpSlowTmrFlag = 0;
    }

    xemacif_input(echo_netif);
    transfer_data();
  }

  /* never reached */
  cleanup_platform();

  return 0;
}

/*
 * Copyright (C) 2009 - 2019 Xilinx, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
 * SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
 * OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
 * OF SUCH DAMAGE.
 *
 */
#if __MICROBLAZE__ || __PPC__
#include "arch/cc.h"
#include "platform.h"
#include "platform_config.h"
#include "xil_cache.h"
#include "xparameters.h"
#include "xintc.h"
#include "xil_exception.h"
#include "lwip/tcp.h"
#include "netif/xadapter.h"
#ifdef STDOUT_IS_16550
#include "xuartns550_l.h"
#endif

#include "lwip/tcp.h"

#if LWIP_DHCP==1
volatile int dhcp_timoutcntr = 24;
```

```
void dhcp_fine_tmr();
void dhcp_coarse_tmr();
#endif

volatile int TcpFastTmrFlag = 0;
volatile int TcpSlowTmrFlag = 0;

extern struct netif *echo_netif;

void
timer_callback()
{
  static int DetectEthLinkStatus = 0;
  /* we need to call tcp_fasttmr & tcp_slowtmr at intervals specified by lwIP.
   * It is not important that the timing is absoluetly accurate.
   */
  static int odd = 1;
#if LWIP_DHCP==1
    static int dhcp_timer = 0;
#endif
  DetectEthLinkStatus++;
   TcpFastTmrFlag = 1;

  odd = !odd;
  if (odd) {

#if LWIP_DHCP==1
    dhcp_timer++;
    dhcp_timoutcntr--;
#endif
    TcpSlowTmrFlag = 1;
#if LWIP_DHCP==1
    dhcp_fine_tmr();
    if (dhcp_timer >= 120) {
      dhcp_coarse_tmr();
      dhcp_timer = 0;
    }
#endif
  }

  /* For detecting Ethernet phy link status periodically */
  if (DetectEthLinkStatus == ETH_LINK_DETECT_INTERVAL) {
    eth_link_detect(echo_netif);
    DetectEthLinkStatus = 0;
  }
}

static XIntc intc;

void platform_setup_interrupts()
{
  XIntc *intcp;
  intcp = &intc;

  XIntc_Initialize(intcp, XPAR_INTC_0_DEVICE_ID);
  XIntc_Start(intcp, XIN_REAL_MODE);

  /* Start the interrupt controller */
  XIntc_MasterEnable(XPAR_INTC_0_BASEADDR);
#ifdef __PPC__
  Xil_ExceptionInit();
  Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
      (XExceptionHandler)XIntc_DeviceInterruptHandler,
      (void*) XPAR_INTC_0_DEVICE_ID);
#elif __MICROBLAZE__
  microblaze_register_handler((XInterruptHandler)XIntc_InterruptHandler, intcp);
#endif

  platform_setup_timer();

#ifdef XPAR_ETHERNET_MAC_IP2INTC_IRPT_MASK
  /* Enable timer and EMAC interrupts in the interrupt controller */
  XIntc_EnableIntr(XPAR_INTC_0_BASEADDR,
#ifdef __MICROBLAZE__
      PLATFORM_TIMER_INTERRUPT_MASK |
#endif
      XPAR_ETHERNET_MAC_IP2INTC_IRPT_MASK);
#endif


#ifdef XPAR_INTC_0_LLTEMAC_0_VEC_ID
#ifdef __MICROBLAZE__
  XIntc_Enable(intcp, PLATFORM_TIMER_INTERRUPT_INTR);
#endif
  XIntc_Enable(intcp, XPAR_INTC_0_LLTEMAC_0_VEC_ID);
#endif


#ifdef XPAR_INTC_0_AXIETHERNET_0_VEC_ID
  XIntc_Enable(intcp, PLATFORM_TIMER_INTERRUPT_INTR);
  XIntc_Enable(intcp, XPAR_INTC_0_AXIETHERNET_0_VEC_ID);
#endif


#ifdef XPAR_INTC_0_EMACLITE_0_VEC_ID
#ifdef __MICROBLAZE__
  XIntc_Enable(intcp, PLATFORM_TIMER_INTERRUPT_INTR);
#endif
  XIntc_Enable(intcp, XPAR_INTC_0_EMACLITE_0_VEC_ID);
#endif


}

void
enable_caches()
{
#ifdef __PPC__
  Xil_ICacheEnableRegion(CACHEABLE_REGION_MASK);
  Xil_DCacheEnableRegion(CACHEABLE_REGION_MASK);
#elif __MICROBLAZE__
#ifdef XPAR_MICROBLAZE_USE_ICACHE
  Xil_ICacheEnable();
#endif
#ifdef XPAR_MICROBLAZE_USE_DCACHE
  Xil_DCacheEnable();
#endif
#endif
}
```

```c
void
disable_caches()
{
  Xil_DCacheDisable();
  Xil_ICacheDisable();
}

void init_platform()
{
  enable_caches();

#ifdef STDOUT_IS_16550
  XUartNs550_SetBaud(STDOUT_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, 9600);
  XUartNs550_SetLineControlReg(STDOUT_BASEADDR, XUN_LCR_8_DATA_BITS);
#endif

  platform_setup_interrupts();
}

void cleanup_platform()
{
  disable_caches();
}
#endif

/*
 * Copyright (C) 2009 - 2019 Xilinx, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
 * SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
 * OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
 * IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
 * OF SUCH DAMAGE.
 *
 */

#include <stdio.h>
#include <string.h>

#include "lwip/err.h"
#include "lwip/tcp.h"
#include "fsl.h"
```

```
#if defined (__arm__) || defined (__aarch64__)
#include "xil_printf.h"
#endif

//char buffer [32];
//volatile void * payload = buffer;
struct tcp_pcb *global_pcb;

int transfer_data() {
//  uint val;
//  void * payload = &val;
//    int validity = 0;
//    getfsl(val, 0);
//    fsl_isinvalid(validity);
//    //fsl_iserror(errority);
//
//      if(validity == 0) {
//            tcp_write(global_pcb, payload , 1, 1); //not sure how to get a tcp_pcb
//            tcp_send();
//      }
  return 0;
}

void print_app_header()
{
#if (LWIP_IPV6==0)
  xil_printf("\n\r\n\r-----lwIP TCP echo server ------\n\r");
#else
  xil_printf("\n\r\n\r-----lwIPv6 TCP echo server ------\n\r");
#endif
  xil_printf("TCP packets sent to port 6001 will be echoed back\n\r");
}

err_t recv_callback(void *arg, struct tcp_pcb *tpcb,
                               struct pbuf *p, err_t err)
{

  /* do not read the packet if we are not in ESTABLISHED state */
  if (!p) {
    tcp_close(tpcb);
    tcp_recv(tpcb, NULL);
    return ERR_OK;
  }
  /* indicate that the packet has been received */
  tcp_recved(tpcb, p->len);

  char buffer [32];
  void * payload = buffer;
  int validity = 1;
  int errority = 1;

  /* axi stream get and send 0,1 axis ports */

//          //u32_t var_axi;
//          //  void *payload = &var_axi;
//          //        getfslx(var_axi,0, FSL_DEFAULT);
//          u16_t len;
//          len = 8;//          putfslx(p->payload, 2, FSL_NONBLOCKING);
//            validity = 0;
//            getfslx(buffer[i + 1], 2, FSL_NONBLOCKING);
```

```
//
//            fsl_isinvalid(validity);
//            fsl_iserror(errority);
//
//            buffer[i + 2] = '\n';
//    //
//    //      if(validity == 0) {
//                    tcp_write(tpcb, payload , i + 3, 1);
//    //      }
//
//          u16_t loop_length = 4;
//          int i = 0;
//          for(; i < loop_length; i += 1) {
//            getfsl(buffer[i], 0);//EXCEPTION_CONTROL_ATOMIC);
//          fsl_isinvalid(validity);
//          fsl_iserror(errority);
//          }
//
//      buffer[loop_length] = '\n';
//      i += 1;
//
//      //tcp_write(tpcb, payload, len , 1);
//      int len2 = (p -> len) >> 2;
//      for(; i < loop_length + len2 + 1; i += 1 ) {2
//          putfslx(p->payload , 1, FSL_NONBLOCKING);
//          validity = 1;
//          fsl_isinvalid(validity);
//                  fsl_iserror(errority);
//          getfslx(buffer[i], 1, FSL_NONBLOCKING);
//          validity = 1;
//          fsl_isinvalid(validity);
//          fsl_iserror(errority);
//      }
//      buffer[i] = '\n';
//      tcp_write(tpcb, payload, loop_length  + len2 + 2, 1);
//

  /* last */
    char * vals = p->payload;
    char val = vals[0];
    validity = 1;
    errority = 1;
        for(int i = 0; i < p->len; i++) {
          val = vals[i];
          putfsl(val, 2);
          putfsl(val, 3);
//          fsl_isinvalid(validity);
//        fsl_iserror(errority);
//        if(validity == 0) {
//          i = i+1;
//        }
//        validity = 1;
//        errority = 1;

          getfslx(buffer[0], 3, FSL_NONBLOCKING);
          fsl_isinvalid(validity);
          fsl_iserror(errority);
          if(validity == 0) {
            buffer[1] = '\n';
            tcp_write(tpcb, payload, 2, 1);
          }
          validity = 1;
          errority = 1;
        }

          putfslx(val, 3,FSL_NONBLOCKING);

          validity = 1;
          errority = 1;

          getfslx(buffer[0], 2, FSL_NONBLOCKING);
      fsl_isinvalid(validity);
      fsl_iserror(errority);

      int len = 0;
      if(validity == 0) {
       len += 1;
      }
          validity = 1;
          errority = 1;
          getfslx(buffer[len], 3, FSL_NONBLOCKING);

      if(validity == 0) {
       len += 1;
      }
      if(len != 0) {
        buffer[len + 1] = '\n';
        tcp_write(tpcb, payload, len + 2, 1);
      }




//          putfslx(p->payload, 2, FSL_NONBLOCKING);
//          validity = 0;
//          getfslx(buffer[i + 1], 2, FSL_NONBLOCKING);
//
//          fsl_isinvalid(validity);
//          fsl_iserror(errority);
//
//          buffer[i + 2] = '\n';
//  //
//  //      if(validity == 0) {
//                  tcp_write(tpcb, payload , i + 3, 1);
//  //      }

  pbuf_free(p);

  return ERR_OK;

}

err_t accept_callback(void *arg, struct tcp_pcb *newpcb, err_t err)
{
  static int connection = 1;

  /* set the receive callback for this connection */
  tcp_recv(newpcb, recv_callback);
  /* just use an integer number indicating the connec/afs/athena.mit.edu/user/n/k/nkk/fpga_6115tion id
  as the
     callback argument */
  tcp_arg(newpcb, (void*)(UINTPTR)connection);

  /* increment for subsequent accepted connections */
  connection++;

  return ERR_OK;
}


int start_application()
{
  struct tcp_pcb *pcb;
  err_t err;
  unsigned port = 7;


  /* create new TCP PCB structure */
  pcb = tcp_new_ip_type(IPADDR_TYPE_ANY);
  if (!pcb) {
    xil_printf("Error creating PCB. Out of Memory\n\r");
    return -1;
  }

  /* bind to specified @port */
  err = tcp_bind(pcb, IP_ANY_TYPE, port);
  if (err != ERR_OK) {
    xil_printf("Unable to bind to port %d: err = %d\n\r", port, err);
    return -2;
  }

  /* we do not need any arguments to callback functions */
  tcp_arg(pcb, NULL);

  /* listen for connections */
  pcb = tcp_listen(pcb);
  if (!pcb) {
    xil_printf("Out of memory while tcp_listen\n\r");
    return -3;
  }

  /* specify callback to use for incoming connections */
  tcp_accept(pcb, accept_callback);

  xil_printf("TCP echo server started @ port %d\n\r", port);


  // set global pcb
  global_pcb = pcb;
  // repeatedly attempt to send, this will force this function to not return
//  char buffer [32];
//  volatile void * payload = buffer;
//    int validity;
//    int errority;
  //  for(int i = 0; i < p->len; i += 1 ) {
  //    putfslx(p->payload + i*4, 1, FSL_DEFAULT); //putfslx(p->payload + i*4, 1, FSL_NONBLOCKING);

//  while(1) {
```

```
//      error = mfmsr();
//      error &= ~0x10;
//      mtmstr(error);
//
//  }
//  for(int i = 0; i < 32; i += 1) {
//    buffer[i] = i + 40;
//  }
//    int len = 4; //4 BYTES
//  int validity;
//  int errority;

//  while(1) {
//    for(int i = 0; i < len; i += 1 ) {
//        putfslx(p->payload + i*4, 1, FSL_DEFAULT); //putfslx(p->payload + i*4, 1, FSL_NONBLOCKING);
//
//        fsl_iserror(errority);
//        getfslx(buffer[i], 1, FSL_NONBLOCKING_EXCEPTION_CONTROL_ATOMIC); //getfslx(buffer[i], 1,
FSL_NONBLOCKING);
//    }
//    getfslx(buffer[i], 1, FSL_NONBLOCKING_EXCEPTION_CONTROL_ATOMIC);
//    fsl_isinvalid(validity);
//    if(validity) {
//    tcp_write(pcb,payload, len, 1);
//    }
//  }
  return 0;
}

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/01/2019 04:17:23 PM
// Design Name:
// Module Name: parser_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module parser_sim();
   parameter PRICE_WIDTH=15;
   parameter ID_WIDTH=15;
   parameter QUANT_WIDTH=7;
   parameter STOCK_WIDTH=7;
   parameter DATA_WIDTH=31;
   logic                  clk_in;
   logic                  reset_in;
```

```
    logic [DATA_WIDTH:0]   data_in;

    logic valid_microblaze_in;
    logic                  ready_to_microblaze_out;

    logic                  enable_in;
    logic [2:0]            operation_out;

     logic  [STOCK_WIDTH:0]                  stock_symbol_out_add;
    logic  [ID_WIDTH:0]                     order_id_out_add;
    logic  [PRICE_WIDTH:0]                  price_out_add;
    logic  [QUANT_WIDTH:0]                  quantity_out_add;

    logic [STOCK_WIDTH:0]                    stock_symbol_out_cancel;
    logic [ID_WIDTH:0]                       order_id_out_cancel;
    logic [PRICE_WIDTH:0]                    price_out_cancel;
    logic [QUANT_WIDTH:0]                    quantity_out_cancel;
    logic delete_out;
   // logic [STOCK_WIDTH: 0] stock_symbol_out;
   //  logic [ID_WIDTH: 0]    order_id_out;
   //  logic [PRICE_WIDTH: 0] price_out;
   //  logic [QUANT_WIDTH: 0] quantity_out;

    logic                  valid_master_in;
    logic                  last_master_out;
    logic                  ready_out;

    logic                  enable_parser;

//   assign ready_to_microblaze_out = ~ready_out;
//   assign last_master_out = ready_out;
//   assign enable_parser = 1; //have to think about this
//   assign delete_out = (operation_out == 3'b000) && ready_out;

    parser_top parser(.clk_in(clk_in), .reset_in(reset_in), .data_in(data_in[7:0]),
    .enable_in(enable_in), .valid_microblaze_in(valid_microblaze_in),
    .ready_to_microblaze_out(ready_to_microblaze_out),
     .operation_out(operation_out),

     .stock_symbol_out_add(stock_symbol_out_add), .order_id_out_add(order_id_out_add),
     .price_out_add(price_out_add), .quantity_out_add(quantity_out_add),

     .stock_symbol_out_cancel(stock_symbol_out_cancel), .order_id_out_cancel(order_id_out_cancel),
     .price_out_cancel(price_out_cancel), .quantity_out_cancel(quantity_out_cancel),
     .delete_out(delete_out),
     .valid_master_in(valid_master_in), .last_master_out(last_master_out), .ready_out(ready_out));


   parameter size = 304;
   logic [303: 0] test_data; //36 bytes == add
   //type, stocklocate, trackingnumber, timestamp, order_ref, buySell, shares, stock, price

   logic [7:0] test;
   logic [7:0] test_1;
   logic [7:0] test_2;
   logic [7:0] test_3;

   always begin
   #10
   clk_in = ~clk_in;
```

```
  end

  initial begin
     clk_in = 1'b0;
     reset_in = 1'b1;
     valid_microblaze_in = 1'b1;
     enable_in = 1'b1; //not really needed
     valid_master_in = 1'b1;
     //test_data =
     312'h3253_1238__3253_1238_A242_A242__3253_1238__41__3253_1238_A242_A242__3253_1238_A242_0043__32AD__1400_
     test_data = 304'h2441_0000000000000000000a000000000000000142000000014141504c202020200186a000;
     test = 8'hBA;
     test_1 = test[7-:8];
     test_2 = test[0+:8];
     for(integer i =0; i <= 7; i = i + 1) begin
        test_3[i] = test[7 - i];
     end
     #20

     enable_in = 1'b1;

     #20
     reset_in = 1'b0;

     data_in = test_data[size - 1 -:8];
     for(integer i = size - 1 - 8; i >= 7 ; i = i - 8) begin
        #20
        data_in = test_data[i-:8];
     end
//     #10
//       data_in = 32'hAF_32_13_45;
//     #600
//       data_in = 32'hAF_32_13_41;
//     #600
//       data_in = 32'hAF_32_13_41;
//    #600
//       data_in = 32'hAF_32_13_44;


end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/01/2019 04:17:23 PM
// Design Name:
// Module Name: parser_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
```

```
//
///////////////////////////////////////////////////////////////////////////////////

module parser_top # (parameter PRICE_WIDTH=15,
                 parameter ID_WIDTH=15,
                 parameter QUANT_WIDTH=7,
                 parameter STOCK_WIDTH=7,
                 parameter DATA_WIDTH=31)
      (
    input                          clk_in,
    input                          reset_in,
    input [DATA_WIDTH:0]           data_in,

    input [15:0]                   sw,
    output                         ca, cb, cc, cd, ce, cf, cg, dp, // segments a-g, dp
    output [7:0]                   an,

    input                          valid_microblaze_in,
    output                         ready_to_microblaze_out,
    input                          enable_in,

    output [STOCK_WIDTH:0]         stock_symbol_out,
    output [2:0]                   operation_out, //add = 1, cancel = 2, delete = 0
    // output [STOCK_WIDTH: 0] stock_symbol_out,
    // output [ID_WIDTH: 0]    order_id_out,
       // output [PRICE_WIDTH: 0] price_out,
    // output [QUANT_WIDTH: 0] quantity_out,

    output logic [STOCK_WIDTH:0] stock_symbol_out_add,
    output logic [ID_WIDTH:0]    order_id_out_add,
    output logic [PRICE_WIDTH:0] price_out_add,
    output logic [QUANT_WIDTH:0] quantity_out_add,
    output                       order_type_out_add,

    output logic [STOCK_WIDTH:0] stock_symbol_out_cancel,
    output logic [ID_WIDTH:0]    order_id_out_cancel,
    output logic [PRICE_WIDTH:0] price_out_cancel,
    output logic [QUANT_WIDTH:0] quantity_out_cancel,
    output                       delete_out,

    input                          valid_master_in,
    output                         last_master_out,
    output                         ready_out
    );

   logic                       enable_parser;
   //logic[STOCK_WIDTH: 0] stock_symbol_raw_out;
   //logic[ID_WIDTH: 0]    order_id_raw_out;
   //logic[PRICE_WIDTH: 0] price_raw_out;
   //logic[QUANT_WIDTH: 0] quantity_raw_out;

   logic [STOCK_WIDTH:0]   stock_symbol_raw_out_add;
   logic [ID_WIDTH:0]      order_id_raw_out_add;
   logic [PRICE_WIDTH:0]   price_raw_out_add;
   logic [QUANT_WIDTH:0]   quantity_raw_out_add;

   logic [STOCK_WIDTH:0]   stock_symbol_raw_out_cancel;
   logic [ID_WIDTH:0]      order_id_raw_out_cancel;
   logic [PRICE_WIDTH:0]   price_raw_out_cancel;
   logic [QUANT_WIDTH:0]   quantity_raw_out_cancel;
```

```
 // mk_parser parser(.clk_in(clk_in), .reset_in(reset_in), .data_in(data_in[7:0]),
 // .enable_in(enable_parser), .valid_in(valid_microblaze_in), .valid_master_in(valid_master_in),
 // .operation_out(operation_out),
 // .stock_symbol_out_add(stock_symbol_out_add), .order_id_out_add(order_id_out_add),
 .price_out_add(price_out_add),
 //  .quantity_out_add(quantity_out_add), .order_type_out_add(order_type_out_add),
 //   .stock_symbol_out_cancel(stock_symbol_out_cancel), .order_id_out_cancel(order_id_out_cancel),
 //   .price_out_cancel(price_out_cancel), .quantity_out_cancel(quantity_out_cancel),
 .ready_out(ready_out));

mk_parser parser(.clk_in(clk_in), .reset_in(reset_in), .data_in(data_in[7:0]),
.enable_in(enable_parser), .valid_in(valid_microblaze_in), .valid_master_in(valid_master_in),
.operation_out(operation_out),
.stock_symbol_out_add(stock_symbol_raw_out_add), .order_id_out_add(order_id_raw_out_add),
.price_out_add(price_raw_out_add),
 .quantity_out_add(quantity_raw_out_add), .order_type_out_add(order_type_raw_out_add),
  .stock_symbol_out_cancel(stock_symbol_raw_out_cancel), .order_id_out_cancel(order_id_raw_out_cancel),
  .price_out_cancel(price_raw_out_cancel), .quantity_out_cancel(quantity_raw_out_cancel),
  .ready_out(ready_out));

  always_comb begin
    for(integer i =0; i <= STOCK_WIDTH; i = i + 1) begin
       stock_symbol_out_add[i] = stock_symbol_raw_out_add[STOCK_WIDTH - i];
       stock_symbol_out_cancel[i] = stock_symbol_raw_out_cancel[STOCK_WIDTH - i];
    end
    for(integer i =0; i <= ID_WIDTH; i = i + 1) begin
      order_id_out_add[i] = order_id_raw_out_add[ID_WIDTH - i];
      order_id_out_cancel[i] = order_id_raw_out_cancel[ID_WIDTH - i];
    end
    for(integer i =0; i <= PRICE_WIDTH; i = i + 1) begin
       price_out_add[i] = price_raw_out_add[PRICE_WIDTH - i];
       price_out_cancel[i] = price_raw_out_cancel[PRICE_WIDTH - i];

    end
    for(integer i =0; i <= QUANT_WIDTH; i = i + 1) begin
       quantity_out_add[i] = quantity_raw_out_add[QUANT_WIDTH - i];
       quantity_out_cancel[i] = quantity_raw_out_cancel[QUANT_WIDTH - i];
    end
  end

   assign ready_to_microblaze_out = ~ready_out;
   assign last_master_out = ready_out;
   assign enable_parser = 1; //have to think about this
   assign delete_out = (operation_out == 3'b000) && ready_out;
   assign stock_symbol_out = (operation_out == 3'b000) ? stock_symbol_out_cancel :
   stock_symbol_out_add;
   // always@(posedge clk_in) begin
   //    if(reset_in) begin

   //    end else begin
   //       if(ready_out)ready_to_microblaze_out <= 1;
   // end


  logic [31:0] data_to_display ;
  seg_display dis(.clk_in(clk_in), .rst_in(reset_in), .val_in(data_to_display), .cat_out({cg, cf, ce,
  cd, cc, cb, ca}), .an_out(an));
   logic [15:0] sw_debounced;
   assign  dp = 1'b1;  // turn off the period
```

```
    debounce deb(.clock_in(clk_in), .reset_in(reset_in), .noisy_in(sw), .clean_out(sw_debounced));

    always@(posedge clk_in) begin
    case(operation_out)
    0 :    case(sw)
        16'b0_0_0_0_0_0_0 : data_to_display <= price_out_add;
        16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_add;
        16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_add;
        16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_add;

        16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
        valid_master_in, last_master_out, ready_out};
        16'b0_0_1_0_0_0_0 : data_to_display <= data_in;

        default : data_to_display <= operation_out;
        endcase
    1 :case(sw)
       16'b0_0_0_0_0_0_0 : data_to_display <= price_out_cancel;
       16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_cancel;
       16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_cancel;
       16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_cancel;

       16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
       valid_master_in, last_master_out, ready_out};
       16'b0_0_1_0_0_0_0 : data_to_display <= data_in;
        default : data_to_display <= operation_out;
        endcase

    2 :case(sw)
       16'b0_0_0_0_0_0_0 : data_to_display <= price_out_cancel;
       16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_cancel;
       16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_cancel;
       16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_cancel;

       16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
       valid_master_in, last_master_out, ready_out};
       16'b0_0_1_0_0_0_0 : data_to_display <= data_in;

        default : data_to_display <= operation_out;
        endcase

    default :case(sw)
       16'b0_0_0_0_0_0_0 : data_to_display <= price_out_add;
       16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_add;
       16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_add;
       16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_add;

       16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
       valid_master_in, last_master_out, ready_out};
       16'b0_0_1_0_0_0_0 : data_to_display <= data_in;

        default : data_to_display <= operation_out;
        endcase

    endcase
    end

endmodule

module debounce (input reset_in, clock_in, noisy_in,
                  output reg clean_out);

    reg [19:0] count;
    reg new_input;

 //   always_ff @(posedge clock_in)
 //     if (reset_in) begin new <= noisy_in; clean_out <= noisy_in; count <= 0; end
 //     else if (noisy_in != new) begin new <= noisy_in; count <= 0; end
 //     else if (count == 650000) clean_out <= new;
 //     else count <= count+1;

    always_ff @(posedge clock_in)
      if (reset_in) begin
         new_input <= noisy_in;
         clean_out <= noisy_in;
         count <= 0; end
      else if (noisy_in != new_input) begin new_input<=noisy_in; count <= 0; end
      else if (count == 650000) clean_out <= new_input;
      else count <= count+1;


 endmodule

 module seg_display(input              clk_in,
                    input              rst_in,
                    input [31:0]       val_in,
                    output logic [6:0] cat_out, // was 7:0 for some reason
                    output logic [7:0] an_out
                    );

    logic [7:0]                        segment_state;
    logic [31:0]                       segment_counter;
    logic [3:0]                        routed_vals;
    logic [6:0]                        led_out;

    binary_to_seven_seg my_converter ( .in(routed_vals), .out(led_out));
    assign cat_out = ~led_out;
    assign an_out = ~segment_state;


    always_comb begin
       case(segment_state)
         8'b0000_0001:   routed_vals = val_in[3:0];
         8'b0000_0010:   routed_vals = val_in[7:4];
         8'b0000_0100:   routed_vals = val_in[11:8];
         8'b0000_1000:   routed_vals = val_in[15:12];
         8'b0001_0000:   routed_vals = val_in[19:16];
         8'b0010_0000:   routed_vals = val_in[23:20];
         8'b0100_0000:   routed_vals = val_in[27:24];
         8'b1000_0000:   routed_vals = val_in[31:28];
         default:        routed_vals = val_in[3:0];
       endcase
    end

    always_ff @(posedge clk_in)begin
       if (rst_in)begin
          segment_state <= 8'b0000_0001;
          segment_counter <= 32'b0;
       end else begin
          if (segment_counter == 32'd100_000)begin //changed from 100_000
```

```
            segment_counter <= 32'd0;
            segment_state <= {segment_state[6:0],segment_state[7]};
         end else begin
            segment_counter <= segment_counter +1;
         end
      end
   end

endmodule //seven_seg_controller


//feel free to either include binary_to_seven_seg module here or in its own file!
module binary_to_seven_seg (in,
                            out
                            );

   input [3:0]                 in;
   output logic [6:0]          out;

   assign out[0] = ~((in == 1)  | (in == 4)  | (in == 11) | (in == 13));
   assign out[1] = ~((in == 5)  | (in == 6)  | (in == 11) | (in == 12) | (in == 14) | (in == 15));
   assign out[2] = ~((in == 2)  | (in == 12) | (in == 14) | (in == 15));
   assign out[3] = ~((in == 1)  | (in == 4)  | (in == 7)  | (in == 10) | (in == 15));
   assign out[4] = ~((in == 1)  | (in == 3)  | (in == 4)  | (in == 5)  | (in == 7)  | (in == 9 ));
   assign out[5] = ~((in == 1)  | (in == 2)  | (in == 3)  | (in == 7)  | (in == 13));
   assign out[6] = ~((in == 0)  | (in == 1)  | (in == 7)  | (in == 12));


endmodule

module mk_parser # (parameter PRICE_WIDTH=15,
                 parameter ID_WIDTH=15,
                 parameter QUANT_WIDTH=7,
                 parameter STOCK_WIDTH=7,
                 parameter DATA_WIDTH=7)//31

   (
    input                   clk_in,
    input                   reset_in,
    input [DATA_WIDTH:0]    data_in,
    input                   enable_in,

    input                   valid_in,

    input                   valid_master_in,
    output logic [2:0]      operation_out,
    // output [STOCK_WIDTH: 0] stock_symbol_out,
    // output [ID_WIDTH: 0]    order_id_out,
    // output [PRICE_WIDTH: 0] price_out,
    // output [QUANT_WIDTH: 0] quantity_out,

    output [STOCK_WIDTH:0] stock_symbol_out_add,
    output [ID_WIDTH:0]    order_id_out_add,
    output order_type_out_add,
    output [PRICE_WIDTH:0] price_out_add,
    output [QUANT_WIDTH:0] quantity_out_add,

    output [STOCK_WIDTH:0] stock_symbol_out_cancel,
    output [ID_WIDTH:0]    order_id_out_cancel,
    output [PRICE_WIDTH:0] price_out_cancel,
    output [QUANT_WIDTH:0] quantity_out_cancel,

    output logic           ready_out
    );

   logic                    enable_add;
   logic                    enable_cancel;
   logic                    enable_dummy;
   logic [2:0]              mess_type;

    logic  [2:0]                           operation_out_add;
   // logic  [STOCK_WIDTH:0]                 stock_symbol_out_add;
   // logic  [ID_WIDTH:0]                    order_id_out_add;
   // logic  [PRICE_WIDTH:0]                 price_out_add;
   // logic  [QUANT_WIDTH:0]                 quantity_out_add;

    logic [2:0]                           operation_out_cancel;
   // logic [STOCK_WIDTH:0]                   stock_symbol_out_cancel;
   // logic [ID_WIDTH:0]                      order_id_out_cancel;
   // logic [PRICE_WIDTH:0]                   price_out_cancel;
   // logic [QUANT_WIDTH:0]                   quantity_out_cancel;

   logic [2:0]                              operation_out_dummy;
   logic [STOCK_WIDTH:0]                   stock_symbol_out_dummy;
   logic [ID_WIDTH:0]                      order_id_out_dummy;
   logic [PRICE_WIDTH:0]                   price_out_dummy;
   logic [QUANT_WIDTH:0]                   quantity_out_dummy;

   logic                                   ready_add_out;
   logic                                   ready_cancel_out;
   logic                                   ready_dummy_out;

   logic [DATA_WIDTH:0] data_reg;
      logic [7:0]                             message;
   logic [DATA_WIDTH:0]                    data_last;
   logic [DATA_WIDTH:0]                    data_last2;
   logic [DATA_WIDTH:0]                    data_message_size;

   logic                                   get_length;

      parameter MESSAGE_TYPE = DATA_WIDTH - 7;

   //using data_in not data_reg here to automatically skip it but currently used to pipeline ffor
   correctness
   mkAddMessage #(.PRICE_WIDTH(PRICE_WIDTH), .ID_WIDTH(ID_WIDTH), .QUANT_WIDTH(QUANT_WIDTH),
   .STOCK_WIDTH(STOCK_WIDTH), .DATA_WIDTH(DATA_WIDTH)) addMessage(.clk_in(clk_in), .reset_in(reset_in),
   .data_in(data_reg), .mess_type_in(mess_type), .enable_in(enable_add), .valid_in(valid_in),
   .operation_out(operation_out_add), .stock_symbol_out(stock_symbol_out_add),
   .order_id_out(order_id_out_add), .price_out(price_out_add), .quantity_out(quantity_out_add),
   .order_type_out(order_type_out_add), .ready_out(ready_add_out));

   mkCancelMessage #(.PRICE_WIDTH(PRICE_WIDTH), .ID_WIDTH(ID_WIDTH), .QUANT_WIDTH(QUANT_WIDTH),
   .STOCK_WIDTH(STOCK_WIDTH), .DATA_WIDTH(DATA_WIDTH)) cancelMessage(.clk_in(clk_in),
   .reset_in(reset_in), .data_in(data_reg),.mess_type_in(mess_type), .enable_in(enable_cancel),
   .valid_in(valid_in), .operation_out(operation_out_cancel),
   .stock_symbol_out(stock_symbol_out_cancel), .order_id_out(order_id_out_cancel),
   .price_out(price_out_cancel), .quantity_out(quantity_out_cancel), .ready_out(ready_cancel_out));
```

```
mkDummyMessage #(.PRICE_WIDTH(PRICE_WIDTH), .ID_WIDTH(ID_WIDTH), .QUANT_WIDTH(QUANT_WIDTH),
.STOCK_WIDTH(STOCK_WIDTH), .DATA_WIDTH(DATA_WIDTH)) dummyMessage(.clk_in(clk_in),
.reset_in(reset_in), .data_in(data_message_size), .enable_in(enable_dummy), .valid_in(valid_in),
.operation_out(operation_out_dummy), .stock_symbol_out(stock_symbol_out_dummy),
.order_id_out(order_id_out_dummy), .price_out(price_out_dummy), .quantity_out(quantity_out_dummy),
.ready_out(ready_dummy_out));



always_comb begin
   for(integer i =0; i <= DATA_WIDTH; i = i + 1) begin
      message[i] =  data_in[DATA_WIDTH - i];
   end
end
// always@(posedge clk_in) begin
//    data_last <= data_in;
//    data_last2 <= data_last;
//    data_message_size <= data_last2;

// end
//logic ready_ou
always@(posedge clk_in) begin
   casez({reset_in, valid_in, message, ready_out || ready_dummy_out, valid_master_in ||
   ready_dummy_out, get_length, enable_add, enable_cancel, enable_dummy}) //probably want to make
   this message a bit stream parameterized (actually although that may yeild energy savings, latency
   savings are not there).
     //actualy there can be latency saving for a class of trading strategies or book bulding methods
     (when system can't match but that is unlikley as that would be the excahgne machine is sso good
     that it can send informtation so fast) that are only looking at the msb s and doing computation
     on them.
     { 16'b1_?_????_????_?_?_?_???}: begin  data_reg <= 0; enable_add <= 0; enable_cancel <= 0;
     enable_dummy <= 0; mess_type <= 0; get_length <= 1; end
     // skip 00 messages, they are not really useful
     { 1'b0, 1'b1, 8'h00,  1'b0, 1'b?, 1'b?, 3'b0_0_0  }:  begin data_reg <= data_reg; enable_add <=
     enable_add; enable_cancel <= enable_cancel; enable_dummy <= enable_dummy; mess_type <=
     mess_type; end

     // get the length of the message
     { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b1, 3'b0_0_0  }:  begin data_message_size <= data_in;
     get_length <= 0; end

     // send the message to the appropriate parser
     { 1'b0, 1'b1, 8'h82,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_add <=
     1; mess_type <= 0; end //add //A 41
     { 1'b0, 1'b1, 8'h86,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_add <=
     1; mess_type <= 1; end // add with id //F 61

     { 1'b0, 1'b1, 8'h2A,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
     <= 1; mess_type <= 0; end //exec //E 54
     { 1'b0, 1'b1, 8'hC2,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
     <= 1;  mess_type <= 1; end //exec_price //C 43
     { 1'b0, 1'b1, 8'hA1,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
     <= 1;  mess_type <= 2; end //cancel //X 85
     { 1'b0, 1'b1, 8'h22,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
     <= 1;  mess_type <= 3; end //delete //D 44
     { 1'b0, 1'b1, 8'hAA,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
     <= 1;  mess_type <= 4; end //replace //U 55

     { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_dummy
     <= 1;  mess_type <= 4; end //dummy
```

```
        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b1_0_0  }:  begin data_reg <= message;  end //sustain
        input
        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b0_1_0  }:  begin data_reg <= message;  end //sustain
        input
        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b0_0_1  }:  begin data_reg <= message;  end //sustain
        input

        //reset when ready signals are high
        {16'b0_????_????_1_1_0_???}: begin data_reg <= 0; mess_type <= 0; enable_add <= 0;
        enable_cancel <= 0; get_length <= 1; enable_dummy <= 0; end //there is a zero bit entry here so
        all good, no important messages dropped.

        default: begin  data_reg <= data_reg; get_length <= get_length; enable_add <= enable_add;
        enable_cancel <= enable_cancel; enable_dummy <= enable_dummy; mess_type <= mess_type; end
      endcase // casez ({reset_in, message, ready_out, enable_add, enable_cancel, enable_dummy})
   end


   always_comb begin
      case({enable_add, enable_cancel, enable_dummy})
        3'b1_0_0 : begin ready_out = ready_add_out; operation_out = operation_out_add; end
        3'b0_1_0 : begin ready_out = ready_cancel_out; operation_out = operation_out_cancel; end
        3'b0_0_1 : ready_out = 0;
        default: begin ready_out = 0; operation_out = 0; end
      endcase // case ({enable_add, enable_cancel, enable_dummy})
   end

//      always@(posedge clk_in) begin
//         ready_out <= ready_out_this;
//      end


      //   always_comb begin
//      case({enable_add, enable_cancel, enable_dummy})
//        3'b1_0_0 : begin ready_out_this = ready_add_out; end
//        3'b0_1_0 : ready_out_this = ready_cancel_out;
//        3'b0_0_1 : ready_out_this = ready_dummy_out;
//        default: ready_out_this = 0;
//      endcase // case ({enable_add, enable_cancel, enable_dummy})
//   end

//   logic ready_out_2;
//      always@(posedge clk_in) begin
//         ready_out_2 <= ready_out_this;
//         ready_out <= ready_out_2;
//      end
endmodule


module mkAddMessage #(parameter PRICE_WIDTH=15,
                      parameter ID_WIDTH=15,
                      parameter QUANT_WIDTH=7,
                      parameter STOCK_WIDTH=7,
                      parameter DATA_WIDTH=31)

   (
    input                         clk_in,
    input                         reset_in,
    input [DATA_WIDTH:0]          data_in,
 input [2:0]                  mess_type_in,
 input                        enable_in,

 input                        valid_in,

 output logic [2:0]           operation_out,
 output logic [STOCK_WIDTH: 0] stock_symbol_out,
 output logic [ID_WIDTH: 0]    order_id_out,
 output logic order_type_out,
 output logic [PRICE_WIDTH: 0] price_out,
 output logic [QUANT_WIDTH: 0] quantity_out,
 output logic                 ready_out
 );
parameter MESSAGE_TYPE = 0; //because we look at messages a cycle after it comes in. it comes in the
same cycle (~enable_in_reg && enable_in) == 1.
parameter STOCK_LOCATE = 2;
parameter TRACKING_NUMBER = 2;
parameter TIMESTAMP = 6;
parameter ORDER_REF_NUM = 8;
parameter BUY_SELL_IND = 1;
parameter SHARES = 4;
parameter STOCK = 8;
parameter PRICE = 4;
parameter ATTRIBUTION = 4;
parameter TOTAL = MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM +
BUY_SELL_IND + SHARES + STOCK + PRICE;

logic [ (TOTAL) * 8 - 1 :0  ] parsed_data = {0};
logic [10:0]                  count;
logic                         enable_in_reg;
logic                         ready_out_reg;
logic                         mess_type_reg;

assign operation_out = 3'b1;
always_comb begin
   if(~enable_in_reg && enable_in) begin
        ready_out = 0;
   end else begin
      ready_out = ready_out_reg;
   end
end
always@(posedge clk_in) begin
   if(reset_in) begin
      count <= 0; enable_in_reg <=0; ready_out_reg <= 0; mess_type_reg <= 0;
   end else begin
      enable_in_reg <= enable_in;
      if(~enable_in_reg && enable_in) begin
         count <= 0; ready_out_reg <= 0;price_out <= 0; quantity_out <= 0; order_id_out <= 0;
         stock_symbol_out <= 0; mess_type_reg <= mess_type_in;
      end else begin
          casez(mess_type_reg)
            0 : begin
               if(valid_in && ~ready_out_reg) begin //don't update parsed_data until ready_out_reg
               is pulled low
                  count <= count + 1;
                  parsed_data <= {data_in , parsed_data[TOTAL * 8 - 1 : DATA_WIDTH]};
                  //parsed_data[count * 8 +: DATA_WIDTH] <= data_in;
               end
               if(count == MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM
               + BUY_SELL_IND + SHARES + STOCK + PRICE - 1  && ~ready_out_reg) begin
```

```
                    ready_out_reg <= 1;
                    order_id_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP) * 8  +: ORDER_REF_NUM * 8];
                    order_type_out <=  parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP +ORDER_REF_NUM) * 8 +: BUY_SELL_IND * 8] == 8'h41 ? 1'b0 : 1'b1 ;
                    quantity_out <=  parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP +ORDER_REF_NUM + BUY_SELL_IND ) * 8 +: SHARES * 8];
                    stock_symbol_out <=  parsed_data[(MESSAGE_TYPE) * 8 +: STOCK_LOCATE * 8];
                    price_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP +ORDER_REF_NUM + BUY_SELL_IND + SHARES + STOCK) * 8 +: PRICE * 8];

                 end
               end
               default : begin
                 if(count > MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM +
                 BUY_SELL_IND + SHARES + STOCK + PRICE + ATTRIBUTION  ) begin
                    ready_out_reg <= 1;
                    price_out <= 10;
                    quantity_out <= 20;
                    order_id_out <= 24;
                    stock_symbol_out <= 23;
                 end else begin
                    count <= count + 1;
                 end
               end
             endcase
          end
       end
    end

endmodule

module mkCancelMessage #(parameter PRICE_WIDTH=15,
                         parameter ID_WIDTH=15,
                         parameter QUANT_WIDTH=7,
                         parameter STOCK_WIDTH=7,
                         parameter DATA_WIDTH=31)

   (
    input                         clk_in,
    input                         reset_in,
    input [DATA_WIDTH:0]          data_in,
    input [2:0]                   mess_type_in,
    input                         enable_in,

    input                         valid_in,

    output [2:0]                  operation_out,
    output logic [STOCK_WIDTH: 0] stock_symbol_out,
    output logic [ID_WIDTH: 0]    order_id_out,
    output logic [PRICE_WIDTH: 0] price_out,
    output logic [QUANT_WIDTH: 0] quantity_out,
    output logic                  ready_out
    );

   parameter MESSAGE_TYPE = 0; //because we look at messages a cycle after it comes in. it comes in the
   same cycle (~enable_in_reg && enable_in) == 1.
   parameter STOCK_LOCATE = 2;
   parameter TRACKING_NUMBER = 2;
   parameter TIMESTAMP = 6;
parameter ORDER_REF_NUM = 8;
parameter SHARES = 4;
parameter MATCH_NUMBER = 8;
parameter PRINTABLE = 8;
parameter PRICE = 4;
parameter TOTAL = MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP + ORDER_REF_NUM  +
SHARES;

logic [10:0]              count;
   logic [ (TOTAL) * 8 - 1 :0  ] parsed_data = {0};
assign operation_out = {1'b0, ~mess_type_in[0], 0};// currently 3'b010;//mess_type_in[1]}; assign
operation_out = {1'b0, mess_type_in[0], mess_type_in[1]}; //currently only order cancel and order
delete (2,3) //{1'b0, mess_type_in[1] , ~mess_type_in[1]}; //cancel order (01, 10)


logic enable_in_reg;
logic ready_out_reg;
logic mess_type_reg;

always_comb begin
   if(~enable_in_reg && enable_in) begin
        ready_out = 0;
   end else begin
      ready_out = ready_out_reg;
   end
end
always@(posedge clk_in) begin
   if(reset_in) begin
      count <= 0; enable_in_reg <=0; ready_out_reg <= 0; mess_type_reg <= 0;
   end else begin
      enable_in_reg <= enable_in;
      if(~enable_in_reg && enable_in) begin
         count <= 0; ready_out_reg <= 0;price_out <= 0; quantity_out <= 0; order_id_out <= 0;
         stock_symbol_out <= 0; mess_type_reg <= mess_type_in;
      end else begin
          casez(mess_type_reg)
            2 : begin
               if(valid_in && ~ready_out_reg) begin //don't update parsed_data until ready_out_reg
               is pulled low
                  count <= count + 1;
                  parsed_data <= {data_in, parsed_data[TOTAL * 8  - 1:DATA_WIDTH]};
                  //parsed_data[count * 8 +: DATA_WIDTH] <= data_in;
               end
               if(count == MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP + ORDER_REF_NUM
               + SHARES - 1 && ~ready_out_reg ) begin
                  ready_out_reg <= 1;
                  order_id_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                  TIMESTAMP) * 8 +: ORDER_REF_NUM * 8];
                  stock_symbol_out <=  parsed_data[(MESSAGE_TYPE) * 8 +: STOCK_LOCATE * 8];
                  quantity_out <=  parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                  TIMESTAMP +ORDER_REF_NUM  ) * 8 +: SHARES * 8];
              end
              end
            0 : begin
               if(valid_in && ~ready_out_reg) begin //don't update parsed_data until ready_out_reg
               is pulled low
                  count <= count + 1;
                  parsed_data <= {data_in, parsed_data[TOTAL * 8  - 1:DATA_WIDTH]};
                  //parsed_data[count * 8 +: DATA_WIDTH] <= data_in;
               end
```

```
                if(count == MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP + ORDER_REF_NUM
                - 1 && ~ready_out_reg ) begin
                   ready_out_reg <= 1;
                   order_id_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                   TIMESTAMP) * 8 +: ORDER_REF_NUM * 8];
                   stock_symbol_out <=  parsed_data[(MESSAGE_TYPE) * 8 +: STOCK_LOCATE * 8];
                end
             end
             default : begin
                if(count > MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM +
                SHARES + MATCH_NUMBER + PRINTABLE +  PRICE  ) begin
                   ready_out_reg <= 1;
                   price_out <= 56;
                   quantity_out <= 980;
                   order_id_out <= 2894;
                   stock_symbol_out <= 2983;
                end else begin
                   count <= count + 1;
                end
             end
           endcase
       end
    end
 end

endmodule

module mkDummyMessage #(parameter PRICE_WIDTH=15,
                        parameter ID_WIDTH=15,
                        parameter QUANT_WIDTH=7,
                        parameter STOCK_WIDTH=7,
                        parameter DATA_WIDTH=31)

   (
    input                   clk_in,
    input                   reset_in,
    input [DATA_WIDTH:0]    data_in,
    input                   enable_in,

    input                   valid_in,

    output [2:0]            operation_out,
    output [STOCK_WIDTH: 0] stock_symbol_out,
    output [ID_WIDTH: 0]    order_id_out,
    output [PRICE_WIDTH: 0] price_out,
    output [QUANT_WIDTH: 0] quantity_out,
    output logic                 ready_out
    );

   logic                    start;
   logic [7:0]             count;

  always@(posedge clk_in) begin
     if(reset_in) begin
        start <= 0; count <= 0; ready_out <= 0;
     end else begin
        if (~start && enable_in) begin
            start <= 1;
            count <= data_in;
        end else begin
```

```
            if(start && valid_in) begin
                count <= count - 1;
                if(count <= 3) begin
                    start <= 0;
                    ready_out <= 1;
                end
            end else begin
                ready_out <= 0;
            end
        end
     end
  end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/01/2019 04:17:23 PM
// Design Name:
// Module Name: parser_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module parser_top # (parameter PRICE_WIDTH=15,
                 parameter ID_WIDTH=15,
                 parameter QUANT_WIDTH=7,
                 parameter STOCK_WIDTH=7,
                 parameter DATA_WIDTH=31)
      (
    input                        clk_in,
    input                        reset_in,
    input [DATA_WIDTH:0]         data_in,

    input [15:0]                 sw,
    output                       ca, cb, cc, cd, ce, cf, cg, dp, // segments a-g, dp
    output [7:0]                 an,

    input                        valid_microblaze_in,
    output                       ready_to_microblaze_out,
    input                        enable_in,

    output [STOCK_WIDTH:0]       stock_symbol_out,
    output [2:0]                 operation_out, //add = 1, cancel = 2, delete = 0
    // output [STOCK_WIDTH: 0] stock_symbol_out,
    // output [ID_WIDTH: 0]    order_id_out,
       // output [PRICE_WIDTH: 0] price_out,
    // output [QUANT_WIDTH: 0] quantity_out,
    output logic [STOCK_WIDTH:0] stock_symbol_out_add,
    output logic [ID_WIDTH:0]    order_id_out_add,
    output logic [PRICE_WIDTH:0] price_out_add,
    output logic [QUANT_WIDTH:0] quantity_out_add,
    output                       order_type_out_add,

    output logic [STOCK_WIDTH:0] stock_symbol_out_cancel,
    output logic [ID_WIDTH:0]    order_id_out_cancel,
    output logic [PRICE_WIDTH:0] price_out_cancel,
    output logic [QUANT_WIDTH:0] quantity_out_cancel,
    output                       delete_out,

    input                        valid_master_in,
    output                       last_master_out,
    output                       ready_out
    );

   logic                   enable_parser;
   //logic[STOCK_WIDTH: 0] stock_symbol_raw_out;
   //logic[ID_WIDTH: 0]    order_id_raw_out;
   //logic[PRICE_WIDTH: 0] price_raw_out;
   //logic[QUANT_WIDTH: 0] quantity_raw_out;

   logic [STOCK_WIDTH:0]   stock_symbol_raw_out_add;
   logic [ID_WIDTH:0]      order_id_raw_out_add;
   logic [PRICE_WIDTH:0]   price_raw_out_add;
   logic [QUANT_WIDTH:0]   quantity_raw_out_add;

   logic [STOCK_WIDTH:0]   stock_symbol_raw_out_cancel;
   logic [ID_WIDTH:0]      order_id_raw_out_cancel;
   logic [PRICE_WIDTH:0]   price_raw_out_cancel;
   logic [QUANT_WIDTH:0]   quantity_raw_out_cancel;

 // mk_parser parser(.clk_in(clk_in), .reset_in(reset_in), .data_in(data_in[7:0]),
 // .enable_in(enable_parser), .valid_in(valid_microblaze_in), .valid_master_in(valid_master_in),
 // .operation_out(operation_out),
 // .stock_symbol_out_add(stock_symbol_out_add), .order_id_out_add(order_id_out_add),
 .price_out_add(price_out_add),
 //  .quantity_out_add(quantity_out_add), .order_type_out_add(order_type_out_add),
 //   .stock_symbol_out_cancel(stock_symbol_out_cancel), .order_id_out_cancel(order_id_out_cancel),
 //   .price_out_cancel(price_out_cancel), .quantity_out_cancel(quantity_out_cancel),
 .ready_out(ready_out));

mk_parser parser(.clk_in(clk_in), .reset_in(reset_in), .data_in(data_in[7:0]),
.enable_in(enable_parser), .valid_in(valid_microblaze_in), .valid_master_in(valid_master_in),
.operation_out(operation_out),
.stock_symbol_out_add(stock_symbol_raw_out_add), .order_id_out_add(order_id_raw_out_add),
.price_out_add(price_raw_out_add),
 .quantity_out_add(quantity_raw_out_add), .order_type_out_add(order_type_raw_out_add),
  .stock_symbol_out_cancel(stock_symbol_raw_out_cancel), .order_id_out_cancel(order_id_raw_out_cancel),
  .price_out_cancel(price_raw_out_cancel), .quantity_out_cancel(quantity_raw_out_cancel),
  .ready_out(ready_out));

  always_comb begin
    for(integer i =0; i <= STOCK_WIDTH; i = i + 1) begin
       stock_symbol_out_add[i] = stock_symbol_raw_out_add[STOCK_WIDTH - i];
       stock_symbol_out_cancel[i] = stock_symbol_raw_out_cancel[STOCK_WIDTH - i];
    end
    for(integer i =0; i <= ID_WIDTH; i = i + 1) begin
      order_id_out_add[i] = order_id_raw_out_add[ID_WIDTH - i];
      order_id_out_cancel[i] = order_id_raw_out_cancel[ID_WIDTH - i];
    end
    for(integer i =0; i <= PRICE_WIDTH; i = i + 1) begin
       price_out_add[i] = price_raw_out_add[PRICE_WIDTH - i];
       price_out_cancel[i] = price_raw_out_cancel[PRICE_WIDTH - i];

    end
    for(integer i =0; i <= QUANT_WIDTH; i = i + 1) begin
       quantity_out_add[i] = quantity_raw_out_add[QUANT_WIDTH - i];
       quantity_out_cancel[i] = quantity_raw_out_cancel[QUANT_WIDTH - i];
    end
  end

   assign ready_to_microblaze_out = ~ready_out;
   assign last_master_out = ready_out;
   assign enable_parser = 1; //have to think about this
   assign delete_out = (operation_out == 3'b000) && ready_out;
   assign stock_symbol_out = (operation_out == 3'b000) ? stock_symbol_out_cancel :
   stock_symbol_out_add;
   // always@(posedge clk_in) begin
   //    if(reset_in) begin

   //    end else begin
   //       if(ready_out)ready_to_microblaze_out <= 1;
   // end


  logic [31:0] data_to_display ;
  seg_display dis(.clk_in(clk_in), .rst_in(reset_in), .val_in(data_to_display), .cat_out({cg, cf, ce,
  cd, cc, cb, ca}), .an_out(an));
   logic [15:0] sw_debounced;
   assign  dp = 1'b1;  // turn off the period
   debounce deb(.clock_in(clk_in), .reset_in(reset_in), .noisy_in(sw), .clean_out(sw_debounced));

   always@(posedge clk_in) begin
   case(operation_out)
   0 :    case(sw)
       16'b0_0_0_0_0_0_0 : data_to_display <= price_out_add;
       16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_add;
       16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_add;
       16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_add;

       16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
       valid_master_in, last_master_out, ready_out};
       16'b0_0_1_0_0_0_0 : data_to_display <= data_in;

       default : data_to_display <= operation_out;
       endcase
   1 :case(sw)
      16'b0_0_0_0_0_0_0 : data_to_display <= price_out_cancel;
      16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_cancel;
      16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_cancel;
      16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_cancel;

      16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
      valid_master_in, last_master_out, ready_out};
      16'b0_0_1_0_0_0_0 : data_to_display <= data_in;
       default : data_to_display <= operation_out;
       endcase
```

```
   2 :case(sw)
      16'b0_0_0_0_0_0_0 : data_to_display <= price_out_cancel;
      16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_cancel;
      16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_cancel;
      16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_cancel;

      16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
      valid_master_in, last_master_out, ready_out};
      16'b0_0_1_0_0_0_0 : data_to_display <= data_in;

       default : data_to_display <= operation_out;
       endcase

   default :case(sw)
      16'b0_0_0_0_0_0_0 : data_to_display <= price_out_add;
      16'b0_0_0_0_0_0_1 : data_to_display <= quantity_out_add;
      16'b0_0_0_0_0_1_0 : data_to_display <= order_id_out_add;
      16'b0_0_0_0_1_0_0 : data_to_display <= stock_symbol_out_add;

      16'b0_0_0_1_0_0_0 : data_to_display <= {valid_microblaze_in, ready_to_microblaze_out,
      valid_master_in, last_master_out, ready_out};
      16'b0_0_1_0_0_0_0 : data_to_display <= data_in;

       default : data_to_display <= operation_out;
       endcase

   endcase
   end

endmodule

module debounce (input reset_in, clock_in, noisy_in,
                 output reg clean_out);

   reg [19:0] count;
   reg new_input;

//   always_ff @(posedge clock_in)
//     if (reset_in) begin new <= noisy_in; clean_out <= noisy_in; count <= 0; end
//     else if (noisy_in != new) begin new <= noisy_in; count <= 0; end
//     else if (count == 650000) clean_out <= new;
//     else count <= count+1;

   always_ff @(posedge clock_in)
     if (reset_in) begin
        new_input <= noisy_in;
        clean_out <= noisy_in;
        count <= 0; end
     else if (noisy_in != new_input) begin new_input<=noisy_in; count <= 0; end
     else if (count == 650000) clean_out <= new_input;
     else count <= count+1;


endmodule

module seg_display(input              clk_in,
                   input              rst_in,
                   input [31:0]       val_in,
                   output logic [6:0] cat_out, // was 7:0 for some reason
                   output logic [7:0] an_out
```

```
                  );

   logic [7:0]                        segment_state;
   logic [31:0]                       segment_counter;
   logic [3:0]                        routed_vals;
   logic [6:0]                        led_out;

   binary_to_seven_seg my_converter ( .in(routed_vals), .out(led_out));
   assign cat_out = ~led_out;
   assign an_out = ~segment_state;


   always_comb begin
      case(segment_state)
        8'b0000_0001:   routed_vals = val_in[3:0];
        8'b0000_0010:   routed_vals = val_in[7:4];
        8'b0000_0100:   routed_vals = val_in[11:8];
        8'b0000_1000:   routed_vals = val_in[15:12];
        8'b0001_0000:   routed_vals = val_in[19:16];
        8'b0010_0000:   routed_vals = val_in[23:20];
        8'b0100_0000:   routed_vals = val_in[27:24];
        8'b1000_0000:   routed_vals = val_in[31:28];
        default:        routed_vals = val_in[3:0];
      endcase
   end

   always_ff @(posedge clk_in)begin
      if (rst_in)begin
         segment_state <= 8'b0000_0001;
         segment_counter <= 32'b0;
      end else begin
         if (segment_counter == 32'd100_000)begin //changed from 100_000
            segment_counter <= 32'd0;
            segment_state <= {segment_state[6:0],segment_state[7]};
         end else begin
            segment_counter <= segment_counter +1;
         end
      end
   end

endmodule //seven_seg_controller


//feel free to either include binary_to_seven_seg module here or in its own file!
module binary_to_seven_seg (in,
                            out
                            );

   input [3:0]                  in;
   output logic [6:0]           out;

   assign out[0] = ~((in == 1)  | (in == 4)  | (in == 11) | (in == 13));
   assign out[1] = ~((in == 5)  | (in == 6)  | (in == 11) | (in == 12) | (in == 14) | (in == 15));
   assign out[2] = ~((in == 2)  | (in == 12) | (in == 14) | (in == 15));
   assign out[3] = ~((in == 1)  | (in == 4)  | (in == 7)  | (in == 10) | (in == 15));
   assign out[4] = ~((in == 1)  | (in == 3)  | (in == 4)  | (in == 5)  | (in == 7)  | (in == 9 ));
   assign out[5] = ~((in == 1)  | (in == 2)  | (in == 3)  | (in == 7)  | (in == 13));
   assign out[6] = ~((in == 0)  | (in == 1)  | (in == 7)  | (in == 12));
```

```
endmodule

module mk_parser # (parameter PRICE_WIDTH=15,
                    parameter ID_WIDTH=15,
                    parameter QUANT_WIDTH=7,
                    parameter STOCK_WIDTH=7,
                    parameter DATA_WIDTH=7)//31

   (
    input                   clk_in,
    input                   reset_in,
    input [DATA_WIDTH:0]    data_in,
    input                   enable_in,

    input                   valid_in,

    input                   valid_master_in,
    output logic [2:0]      operation_out,
    // output [STOCK_WIDTH: 0] stock_symbol_out,
    // output [ID_WIDTH: 0]    order_id_out,
    // output [PRICE_WIDTH: 0] price_out,
    // output [QUANT_WIDTH: 0] quantity_out,

    output [STOCK_WIDTH:0] stock_symbol_out_add,
    output [ID_WIDTH:0]    order_id_out_add,
    output order_type_out_add,
    output [PRICE_WIDTH:0] price_out_add,
    output [QUANT_WIDTH:0] quantity_out_add,

    output [STOCK_WIDTH:0] stock_symbol_out_cancel,
    output [ID_WIDTH:0]    order_id_out_cancel,
    output [PRICE_WIDTH:0] price_out_cancel,
    output [QUANT_WIDTH:0] quantity_out_cancel,

    output logic           ready_out
    );

   logic                    enable_add;
   logic                    enable_cancel;
   logic                    enable_dummy;
   logic [2:0]              mess_type;

    logic  [2:0]                            operation_out_add;
   // logic  [STOCK_WIDTH:0]                  stock_symbol_out_add;
   // logic  [ID_WIDTH:0]                     order_id_out_add;
   // logic  [PRICE_WIDTH:0]                  price_out_add;
   // logic  [QUANT_WIDTH:0]                  quantity_out_add;

    logic [2:0]                           operation_out_cancel;
   // logic [STOCK_WIDTH:0]                    stock_symbol_out_cancel;
   // logic [ID_WIDTH:0]                       order_id_out_cancel;
   // logic [PRICE_WIDTH:0]                    price_out_cancel;
   // logic [QUANT_WIDTH:0]                    quantity_out_cancel;

   logic [2:0]                              operation_out_dummy;
   logic [STOCK_WIDTH:0]                   stock_symbol_out_dummy;
   logic [ID_WIDTH:0]                      order_id_out_dummy;
   logic [PRICE_WIDTH:0]                   price_out_dummy;
   logic [QUANT_WIDTH:0]                   quantity_out_dummy;
   logic                                   ready_add_out;
   logic                                   ready_cancel_out;
   logic                                   ready_dummy_out;

   logic [DATA_WIDTH:0] data_reg;
      logic [7:0]                             message;
   logic [DATA_WIDTH:0]                    data_last;
   logic [DATA_WIDTH:0]                    data_last2;
   logic [DATA_WIDTH:0]                    data_message_size;

   logic                                   get_length;

      parameter MESSAGE_TYPE = DATA_WIDTH - 7;

   //using data_in not data_reg here to automatically skip it but currently used to pipeline ffor
   correctness
   mkAddMessage #(.PRICE_WIDTH(PRICE_WIDTH), .ID_WIDTH(ID_WIDTH), .QUANT_WIDTH(QUANT_WIDTH),
   .STOCK_WIDTH(STOCK_WIDTH), .DATA_WIDTH(DATA_WIDTH)) addMessage(.clk_in(clk_in), .reset_in(reset_in),
   .data_in(data_reg), .mess_type_in(mess_type), .enable_in(enable_add), .valid_in(valid_in),
   .operation_out(operation_out_add), .stock_symbol_out(stock_symbol_out_add),
   .order_id_out(order_id_out_add), .price_out(price_out_add), .quantity_out(quantity_out_add),
   .order_type_out(order_type_out_add), .ready_out(ready_add_out));

   mkCancelMessage #(.PRICE_WIDTH(PRICE_WIDTH), .ID_WIDTH(ID_WIDTH), .QUANT_WIDTH(QUANT_WIDTH),
   .STOCK_WIDTH(STOCK_WIDTH), .DATA_WIDTH(DATA_WIDTH)) cancelMessage(.clk_in(clk_in),
   .reset_in(reset_in), .data_in(data_reg),.mess_type_in(mess_type), .enable_in(enable_cancel),
   .valid_in(valid_in), .operation_out(operation_out_cancel),
   .stock_symbol_out(stock_symbol_out_cancel), .order_id_out(order_id_out_cancel),
   .price_out(price_out_cancel), .quantity_out(quantity_out_cancel), .ready_out(ready_cancel_out));

   mkDummyMessage #(.PRICE_WIDTH(PRICE_WIDTH), .ID_WIDTH(ID_WIDTH), .QUANT_WIDTH(QUANT_WIDTH),
   .STOCK_WIDTH(STOCK_WIDTH), .DATA_WIDTH(DATA_WIDTH)) dummyMessage(.clk_in(clk_in),
   .reset_in(reset_in), .data_in(data_message_size), .enable_in(enable_dummy), .valid_in(valid_in),
   .operation_out(operation_out_dummy), .stock_symbol_out(stock_symbol_out_dummy),
   .order_id_out(order_id_out_dummy), .price_out(price_out_dummy), .quantity_out(quantity_out_dummy),
   .ready_out(ready_dummy_out));



   always_comb begin
      for(integer i =0; i <= DATA_WIDTH; i = i + 1) begin
         message[i] =  data_in[DATA_WIDTH - i];
      end
   end
   // always@(posedge clk_in) begin
   //    data_last <= data_in;
   //    data_last2 <= data_last;
   //    data_message_size <= data_last2;

   // end
   //logic ready_ou
   always@(posedge clk_in) begin
      casez({reset_in, valid_in, message, ready_out || ready_dummy_out, valid_master_in ||
      ready_dummy_out, get_length, enable_add, enable_cancel, enable_dummy}) //probably want to make
      this message a bit stream parameterized (actually although that may yeild energy savings, latency
      savings are not there).
        //actualy there can be latency saving for a class of trading strategies or book bulding methods
        (when system can't match but that is unlikley as that would be the excahgne machine is sso good
        that it can send informtation so fast) that are only looking at the msb s and doing computation
        on them.
```

```
        { 16'b1_?_????_????_?_?_?_???}: begin  data_reg <= 0; enable_add <= 0; enable_cancel <= 0;
        enable_dummy <= 0; mess_type <= 0; get_length <= 1; end
        // skip 00 messages, they are not really useful
        { 1'b0, 1'b1, 8'h00,  1'b0, 1'b?, 1'b?, 3'b0_0_0  }:  begin data_reg <= data_reg; enable_add <=
        enable_add; enable_cancel <= enable_cancel; enable_dummy <= enable_dummy; mess_type <=
        mess_type; end

        // get the length of the message
        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b1, 3'b0_0_0  }:  begin data_message_size <= data_in;
        get_length <= 0; end

        // send the message to the appropriate parser
        { 1'b0, 1'b1, 8'h82,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_add <=
        1; mess_type <= 0; end //add //A 41
        { 1'b0, 1'b1, 8'h86,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_add <=
        1; mess_type <= 1; end // add with id //F 61

        { 1'b0, 1'b1, 8'h2A,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
        <= 1; mess_type <= 0; end //exec //E 54
        { 1'b0, 1'b1, 8'hC2,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
        <= 1;  mess_type <= 1; end //exec_price //C 43
        { 1'b0, 1'b1, 8'hA1,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
        <= 1;  mess_type <= 2; end //cancel //X 85
        { 1'b0, 1'b1, 8'h22,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
        <= 1;  mess_type <= 3; end //delete //D 44
        { 1'b0, 1'b1, 8'hAA,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_cancel
        <= 1;  mess_type <= 4; end //replace //U 55

        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b0_0_0  }:  begin data_reg <= message; enable_dummy
        <= 1;  mess_type <= 4; end //dummy

        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b1_0_0  }:  begin data_reg <= message;  end //sustain
        input
        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b0_1_0  }:  begin data_reg <= message;  end //sustain
        input
        { 1'b0, 1'b1, 8'h??,  1'b0, 1'b?, 1'b0, 3'b0_0_1  }:  begin data_reg <= message;  end //sustain
        input

        //reset when ready signals are high
        {16'b0_????_????_1_1_0_???}: begin data_reg <= 0; mess_type <= 0; enable_add <= 0;
        enable_cancel <= 0; get_length <= 1; enable_dummy <= 0; end //there is a zero bit entry here so
        all good, no important messages dropped.

        default: begin  data_reg <= data_reg; get_length <= get_length; enable_add <= enable_add;
        enable_cancel <= enable_cancel; enable_dummy <= enable_dummy; mess_type <= mess_type; end
      endcase // casez ({reset_in, message, ready_out, enable_add, enable_cancel, enable_dummy})
   end


   always_comb begin
      case({enable_add, enable_cancel, enable_dummy})
        3'b1_0_0 : begin ready_out = ready_add_out; operation_out = operation_out_add; end
        3'b0_1_0 : begin ready_out = ready_cancel_out; operation_out = operation_out_cancel; end
        3'b0_0_1 : ready_out = 0;
        default: begin ready_out = 0; operation_out = 0; end
      endcase // case ({enable_add, enable_cancel, enable_dummy})
   end

//      always@(posedge clk_in) begin
//         ready_out <= ready_out_this;
//      end


      //   always_comb begin
//      case({enable_add, enable_cancel, enable_dummy})
//        3'b1_0_0 : begin ready_out_this = ready_add_out; end
//        3'b0_1_0 : ready_out_this = ready_cancel_out;
//        3'b0_0_1 : ready_out_this = ready_dummy_out;
//        default: ready_out_this = 0;
//      endcase // case ({enable_add, enable_cancel, enable_dummy})
//   end

//   logic ready_out_2;
//      always@(posedge clk_in) begin
//         ready_out_2 <= ready_out_this;
//         ready_out <= ready_out_2;
//      end
endmodule


module mkAddMessage #(parameter PRICE_WIDTH=15,
                      parameter ID_WIDTH=15,
                      parameter QUANT_WIDTH=7,
                      parameter STOCK_WIDTH=7,
                      parameter DATA_WIDTH=31)

   (
    input                        clk_in,
    input                        reset_in,
    input [DATA_WIDTH:0]         data_in,
    input [2:0]                  mess_type_in,
    input                        enable_in,

    input                        valid_in,

    output logic [2:0]           operation_out,
    output logic [STOCK_WIDTH: 0] stock_symbol_out,
    output logic [ID_WIDTH: 0]    order_id_out,
    output logic order_type_out,
    output logic [PRICE_WIDTH: 0] price_out,
    output logic [QUANT_WIDTH: 0] quantity_out,
    output logic                 ready_out
    );
   parameter MESSAGE_TYPE = 0; //because we look at messages a cycle after it comes in. it comes in the
   same cycle (~enable_in_reg && enable_in) == 1.
   parameter STOCK_LOCATE = 2;
   parameter TRACKING_NUMBER = 2;
   parameter TIMESTAMP = 6;
   parameter ORDER_REF_NUM = 8;
   parameter BUY_SELL_IND = 1;
   parameter SHARES = 4;
   parameter STOCK = 8;
   parameter PRICE = 4;
   parameter ATTRIBUTION = 4;
   parameter TOTAL = MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM +
   BUY_SELL_IND + SHARES + STOCK + PRICE;

   logic [ (TOTAL) * 8 - 1 :0  ] parsed_data = {0};
   logic [10:0]                  count;
   logic                         enable_in_reg;
   logic                          ready_out_reg;
   logic                          mess_type_reg;

   assign operation_out = 3'b1;
   always_comb begin
      if(~enable_in_reg && enable_in) begin
           ready_out = 0;
      end else begin
         ready_out = ready_out_reg;
      end
   end
   always@(posedge clk_in) begin
      if(reset_in) begin
         count <= 0; enable_in_reg <=0; ready_out_reg <= 0; mess_type_reg <= 0;
      end else begin
         enable_in_reg <= enable_in;
         if(~enable_in_reg && enable_in) begin
            count <= 0; ready_out_reg <= 0;price_out <= 0; quantity_out <= 0; order_id_out <= 0;
            stock_symbol_out <= 0; mess_type_reg <= mess_type_in;
         end else begin
             casez(mess_type_reg)
               0 : begin
                  if(valid_in && ~ready_out_reg) begin //don't update parsed_data until ready_out_reg
                  is pulled low
                     count <= count + 1;
                     parsed_data <= {data_in , parsed_data[TOTAL * 8 - 1 : DATA_WIDTH]};
                     //parsed_data[count * 8 +: DATA_WIDTH] <= data_in;
                  end
                  if(count == MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM
                  + BUY_SELL_IND + SHARES + STOCK + PRICE - 1  && ~ready_out_reg) begin
                     ready_out_reg <= 1;
                     order_id_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                     TIMESTAMP) * 8  +: ORDER_REF_NUM * 8];
                     order_type_out <=  parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                     TIMESTAMP +ORDER_REF_NUM) * 8 +: BUY_SELL_IND * 8] == 8'h41 ? 1'b0 : 1'b1 ;
                     quantity_out <=  parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                     TIMESTAMP +ORDER_REF_NUM + BUY_SELL_IND ) * 8 +: SHARES * 8];
                     stock_symbol_out <=  parsed_data[(MESSAGE_TYPE) * 8 +: STOCK_LOCATE * 8];
                     price_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                     TIMESTAMP +ORDER_REF_NUM + BUY_SELL_IND + SHARES + STOCK) * 8 +: PRICE * 8];

                  end
               end
               default : begin
                  if(count > MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM +
                  BUY_SELL_IND + SHARES + STOCK + PRICE + ATTRIBUTION  ) begin
                     ready_out_reg <= 1;
                     price_out <= 10;
                     quantity_out <= 20;
                     order_id_out <= 24;
                     stock_symbol_out <= 23;
                  end else begin
                     count <= count + 1;
                  end
               end
             endcase
         end
      end
   end
```

```
endmodule

module mkCancelMessage #(parameter PRICE_WIDTH=15,
                         parameter ID_WIDTH=15,
                         parameter QUANT_WIDTH=7,
                         parameter STOCK_WIDTH=7,
                         parameter DATA_WIDTH=31)

   (
    input                          clk_in,
    input                          reset_in,
    input [DATA_WIDTH:0]           data_in,
    input [2:0]                    mess_type_in,
    input                          enable_in,

    input                          valid_in,

    output [2:0]                   operation_out,
    output logic [STOCK_WIDTH: 0] stock_symbol_out,
    output logic [ID_WIDTH: 0]    order_id_out,
    output logic [PRICE_WIDTH: 0] price_out,
    output logic [QUANT_WIDTH: 0] quantity_out,
    output logic                   ready_out
    );

   parameter MESSAGE_TYPE = 0; //because we look at messages a cycle after it comes in. it comes in the
   same cycle (~enable_in_reg && enable_in) == 1.
   parameter STOCK_LOCATE = 2;
   parameter TRACKING_NUMBER = 2;
   parameter TIMESTAMP = 6;
   parameter ORDER_REF_NUM = 8;
   parameter SHARES = 4;
   parameter MATCH_NUMBER = 8;
   parameter PRINTABLE = 8;
   parameter PRICE = 4;
   parameter TOTAL = MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP + ORDER_REF_NUM  +
   SHARES;

   logic [10:0]             count;
      logic [ (TOTAL) * 8 - 1 :0  ] parsed_data = {0};
   assign operation_out = {1'b0, ~mess_type_in[0], 0};// currently 3'b010;//mess_type_in[1]}; assign
   operation_out = {1'b0, mess_type_in[0], mess_type_in[1]}; //currently only order cancel and order
   delete (2,3) //{1'b0, mess_type_in[1] , ~mess_type_in[1]}; //cancel order (01, 10)


   logic enable_in_reg;
   logic ready_out_reg;
   logic mess_type_reg;

   always_comb begin
      if(~enable_in_reg && enable_in) begin
           ready_out = 0;
      end else begin
         ready_out = ready_out_reg;
      end
   end
   always@(posedge clk_in) begin
      if(reset_in) begin
         count <= 0; enable_in_reg <=0; ready_out_reg <= 0; mess_type_reg <= 0;
      end else begin
```

```
        enable_in_reg <= enable_in;
        if(~enable_in_reg && enable_in) begin
           count <= 0; ready_out_reg <= 0;price_out <= 0; quantity_out <= 0; order_id_out <= 0;
           stock_symbol_out <= 0; mess_type_reg <= mess_type_in;
        end else begin
            casez(mess_type_reg)
              2 : begin
                 if(valid_in && ~ready_out_reg) begin //don't update parsed_data until ready_out_reg
                 is pulled low
                    count <= count + 1;
                    parsed_data <= {data_in, parsed_data[TOTAL * 8  - 1:DATA_WIDTH]};
                    //parsed_data[count * 8 +: DATA_WIDTH] <= data_in;
                 end
                 if(count == MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP + ORDER_REF_NUM
                 + SHARES - 1 && ~ready_out_reg ) begin
                    ready_out_reg <= 1;
                    order_id_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP) * 8 +: ORDER_REF_NUM * 8];
                    stock_symbol_out <=  parsed_data[(MESSAGE_TYPE) * 8 +: STOCK_LOCATE * 8];
                    quantity_out <=  parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP +ORDER_REF_NUM  ) * 8 +: SHARES * 8];
                end
                end
              0 : begin
                 if(valid_in && ~ready_out_reg) begin //don't update parsed_data until ready_out_reg
                 is pulled low
                    count <= count + 1;
                    parsed_data <= {data_in, parsed_data[TOTAL * 8  - 1:DATA_WIDTH]};
                    //parsed_data[count * 8 +: DATA_WIDTH] <= data_in;
                 end
                 if(count == MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP + ORDER_REF_NUM
                 - 1 && ~ready_out_reg ) begin
                    ready_out_reg <= 1;
                    order_id_out <= parsed_data[(MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER +
                    TIMESTAMP) * 8 +: ORDER_REF_NUM * 8];
                    stock_symbol_out <=  parsed_data[(MESSAGE_TYPE) * 8 +: STOCK_LOCATE * 8];
                 end
              end
              default : begin
                 if(count > MESSAGE_TYPE + STOCK_LOCATE + TRACKING_NUMBER + TIMESTAMP +ORDER_REF_NUM +
                 SHARES + MATCH_NUMBER + PRINTABLE +  PRICE   ) begin
                    ready_out_reg <= 1;
                    price_out <= 56;
                    quantity_out <= 980;
                    order_id_out <= 2894;
                    stock_symbol_out <= 2983;
                 end else begin
                    count <= count + 1;
                 end
              end
            endcase
        end
     end
  end

endmodule

module mkDummyMessage #(parameter PRICE_WIDTH=15,
                        parameter ID_WIDTH=15,
                        parameter QUANT_WIDTH=7,
                        parameter STOCK_WIDTH=7,
                        parameter DATA_WIDTH=31)

   (
    input                   clk_in,
    input                   reset_in,
    input [DATA_WIDTH:0]    data_in,
    input                   enable_in,

    input                   valid_in,

    output [2:0]            operation_out,
    output [STOCK_WIDTH: 0] stock_symbol_out,
    output [ID_WIDTH: 0]    order_id_out,
    output [PRICE_WIDTH: 0] price_out,
    output [QUANT_WIDTH: 0] quantity_out,
    output logic                 ready_out
    );

   logic                    start;
   logic [7:0]             count;

  always@(posedge clk_in) begin
     if(reset_in) begin
        start <= 0; count <= 0; ready_out <= 0;
     end else begin
        if (~start && enable_in) begin
            start <= 1;
            count <= data_in;
        end else begin
            if(start && valid_in) begin
                count <= count - 1;
                if(count <= 3) begin
                    start <= 0;
                    ready_out <= 1;
                end
            end else begin
                ready_out <= 0;
            end
        end
     end
  end
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 12/07/2019 02:26:13 PM
// Design Name:
// Module Name: top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
parameter PRICE_INDEX = 15;
parameter ORDER_INDEX = 7;
parameter QUANTITY_INDEX = 7;

parameter PRICE_WIDTH=15;
parameter ID_WIDTH=15;
parameter QUANT_WIDTH=7;

typedef struct packed {
 logic [PRICE_INDEX:0] price;
 logic [ORDER_INDEX:0] order_id;
 logic [QUANTITY_INDEX:0] quantity;
} book_entry;


module top(
    input [ID_WIDTH:0]     order_id_out_add,
    input [PRICE_WIDTH:0]  price_out_add,
    input [QUANT_WIDTH:0]  quantity_out_add,
    input                  order_type_out_add,

    output book_entry order_to_add
    );

    assign order_to_add.price = price_out_add;
    assign order_to_add.order_id = order_id_out_add;
    assign order_to_add.quantity = quantity_out_add;
endmodule
```

7.1 Final Conclusion:

Thanks for a great semester Joe and Gim and all the TAs!