High Frequency Trade Book Builder using FPGA

CSEE 4840 Embedded System

Presenters:
Choka Thenappan (ct3185)
Shivam Shekhar (ss6960)
Ameya Keshava Mallya (am6024)

May 12, 2024

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Goal

- High Frequency Trade Book building using FPGA and simulated NASDAQ ITCH message protocol.

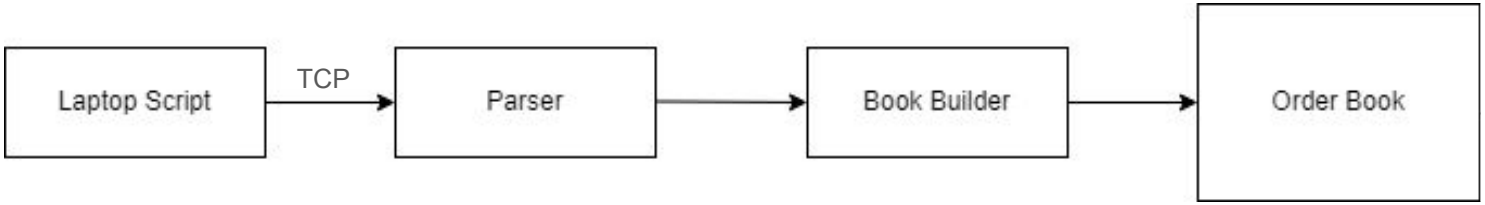
- Main Components -
    - Simulated Market
    - Hardware component
    - HW/SW interface

Background

- Average amount of data coming from the market is around 32 Gb/s. Which means if the data transfer is down even for a millisecond we lose 10000 packets of data.
- Hence we cannot use software methods to parse the data due to latency issues.
- This is why we decided to use FPGA to parse market data to cut latency issues and take advantages of parallel processing speeds.

- Order types :
    - Ask: The term "ask" refers to the lowest price at which a seller will sell the stock.
    - Bid : The term "bid" refers to the highest price a buyer will pay to buy a specified number of shares of a stock at any given time.
    - Sell : Sell refers to the stock being purchased between a buyer and seller.

Project Flow



- We get market data from the simulated market script because of current FPGA limitations.
- The received market data is then processed by a hardware software integrator which sends the data received to the hardware through registers for parsing and book building.
- The parser parses the data and builds and keeps track of the max price.

Incoming Message

Field name	Offset	Length	Format	Binary	Decoded	Notes
Message Type	0	1	Alphanumeric	`0x41`	`A`	Add Order message
Timestamp	1	8	Numeric	`0x0000000000000000`	0	Nanoseconds
Order Reference Number	9	4	Numeric	`0x000003EA`	1002	
Transaction ID	13	4	Numeric	`0x00000000`	0	
Order book ID	17	4	Numeric	`0x000500CD`	327885	
Side	21	1	Alphanumeric	`0x42`	B	“B” = Buy order “S” = Sell order
Quantity	22	4	Numeric	`0x00000001`	1	
Price	26	8	Price	`0x0000000000000001`	1	
Yield	34	4	Price	`0x00000000`	0	

Simulated Market

```
FPGA_IP = "128.59.19.114"
FPGA_PORT = 42000

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

sock.connect((FPGA_IP, FPGA_PORT))
```

Simulated Market

```
message = struct.pack(
    "!BIIIIBIQQ8s",
    msg_type,
    time_stamp,
    order_reference_number,
    transaction_id,
    order_book_id,
    side,
    quantity,
    price,
    yield_value,
    stock
)
return message
```

Data Flow

- Data coming in from Market sim gets into software queue.
- Software Queue communicates with the Hardware with the defined registers.

HW/SW Interface - Ethernet data

- Reading the network data from ethernet and storing that in a queue.

```
void *network_thread_f(void *arg) {
    int sockfd = *(int *)arg;
    char recvBuf[BUFFER_SIZE];
    while (1) {
        int n = read(sockfd, recvBuf, BUFFER_SIZE);
        if (n < 0) {
            perror("Error reading from socket");
            exit(1);
        } else if (n == 0) {
            // printf("Connection closed by client\n");
            close(sockfd);
            break;
        }
        QueueData newData;
        strcpy(newData.data, data);
        enqueue(&dataQueue, newData);
    }
    return NULL;
}
```

HW/SW Interface - Writing in registers

Defining registers for all the data in the message and the required flag.

Using the ioctl calls to write/read data.

```
/* Device Registers */
#define MESSAGE_TYPE(x) (x)
#define TIMESTAMP(x) ((x)+1)
#define ORDER_REF_NUMBER(x) ((x)+2)
#define TRANS_ID(x) ((x)+3)
#define ORDER_BOOK_ID(x) ((x)+4)
#define SIDE(x) ((x)+5)
#define QTY(x) ((x)+6)
#define PRICE(x) ((x)+7)
#define YIELD(x) ((x)+8)
#define BUFFER_NOT_EMPTY(x) ((x)+9)
#define READPORTT(x) ((x)+10)
```

HW/SW Interface

```
static void write_data(vga_ball_color_t *message)
{
    iowrite8(message->msg_type, MESSAGE_TYPE(dev.virtbase) );
    iowrite8(message->timestamp, TIMESTAMP(dev.virtbase) );
    iowrite8(message->order_ref_number, ORDER_REF_NUMBER(dev.virtbase) );
    iowrite8(message->trans_id, TRANS_ID(dev.virtbase) );
    iowrite8(message->order_book_id, ORDER_BOOK_ID(dev.virtbase) );
    iowrite8(message->side, SIDE(dev.virtbase) );
    iowrite8(message->qty, QTY(dev.virtbase) );
    iowrite8(message->price, PRICE(dev.virtbase) );
    iowrite8(message->yield, YIELD(dev.virtbase) );
    iowrite8(message->buffer_not_empty, BUFFER_NOT_EMPTY(dev.virtbase));
    dev.message = *message;
}
```

HW/SW Interface

Sensing the hardware to send next data.

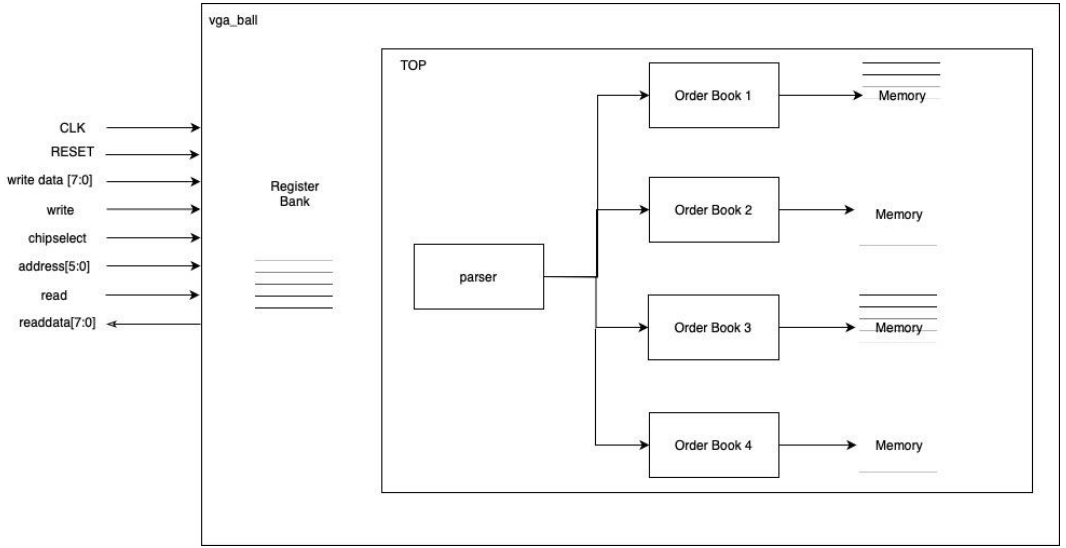
```c
void write_message(const vga_ball_color_t *c) {
    vga_ball_arg_t vla;
    vla.message = *c;
    unsigned char bufferNotEmpty = ioctl(vga_ball_fd, VGA_BALL_READ_DATA, &vla);
    unsigned char readPort = ioctl(vga_ball_fd, VGA_BALL_READ_DATA, &vla);
    if (bufferNotEmpty && readPort) {

    if (ioctl(vga_ball_fd, VGA_BALL_WRITE_DATA, &vla)) {
        perror("ioctl(VGA_BALL_WRITE_DATA) failed");
        return;
    }
    printf("Msg Type: %02x, Timestamp: %016llx, Order Ref Number: %08x, Order Book ID: %08x, Qty: %08x, Price: %016llx \n",
           vla.message.msg_type, vla.message.timestamp, vla.message.order_ref_number,
           vla.message.order_book_id, vla.message.qty, vla.message.price);
    printf("Done!\n");

    }
    else {
        printf("Waiting for ready...\n");
    }
}
```

HW Design - Implementation

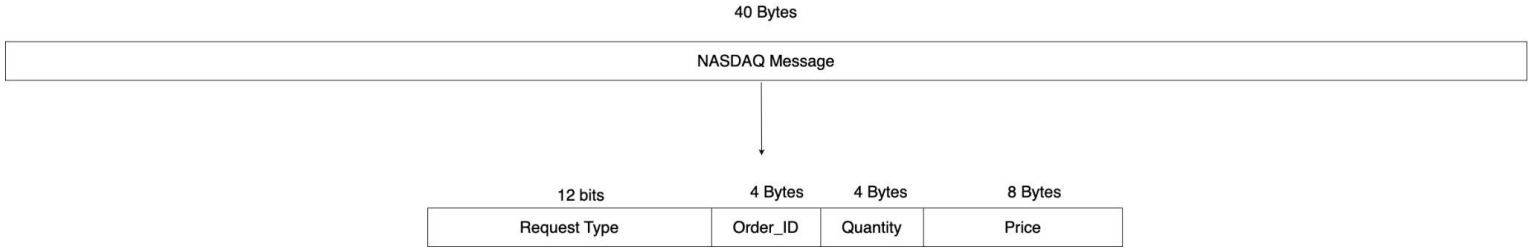
4 Bytes	4 Bytes	8 Bytes
Order_ID	Quantity	Price



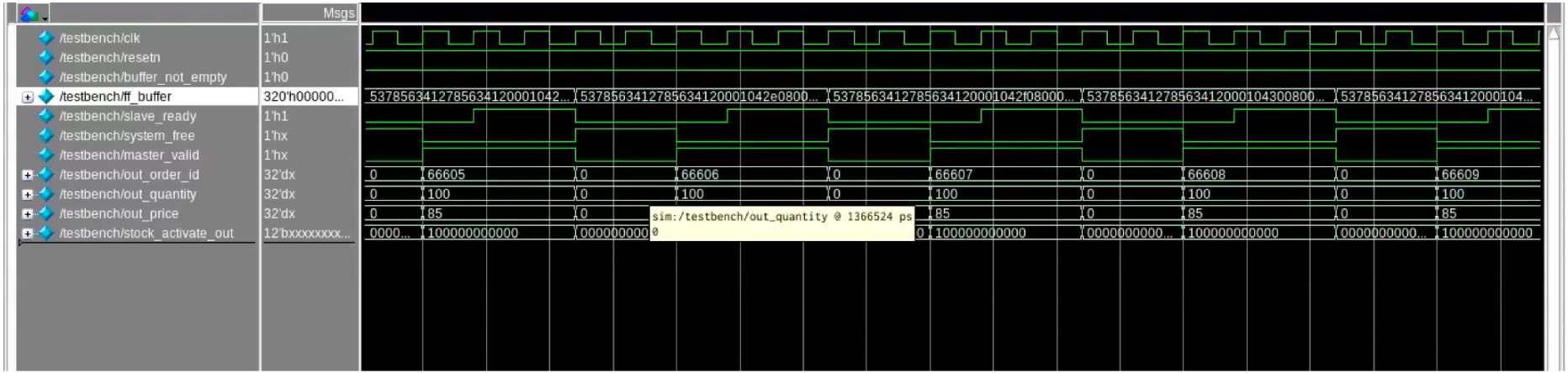
HW Design - Parser

```
module parser(
input [319:0] ff_buffer,
output reg [31:0] out_order_id,
output reg [31:0] out_quantity,
output reg [63:0] out_price,
output reg [11:0] stock_activate
);
```

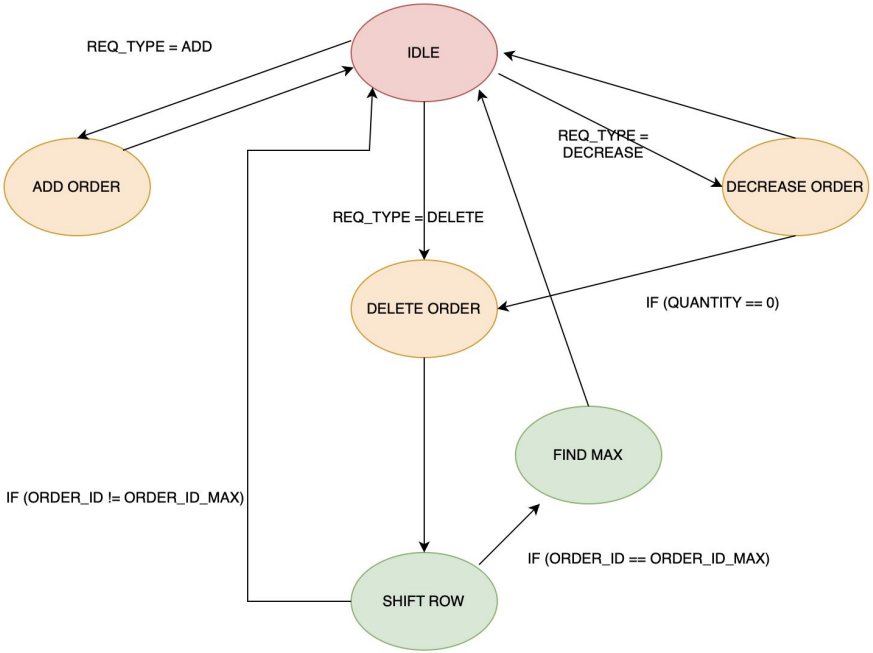
```
`define ADD1 12'b100000000000
`define DELETE1 12'b010000000000
`define DECREASE1 12'b001000000000
`define ADD2 12'b000100000000
`define DELETE2 12'b000010000000
`define DECREASE2 12'b000001000000
`define ADD3 12'b000000100000
`define DELETE3 12'b000000010000
`define DECREASE3 12'b000000001000
`define ADD4 12'b000000000100
`define DELETE4 12'b000000000010
`define DECREASE4 12'b000000000001
```



HW Design - Parser Wave Window

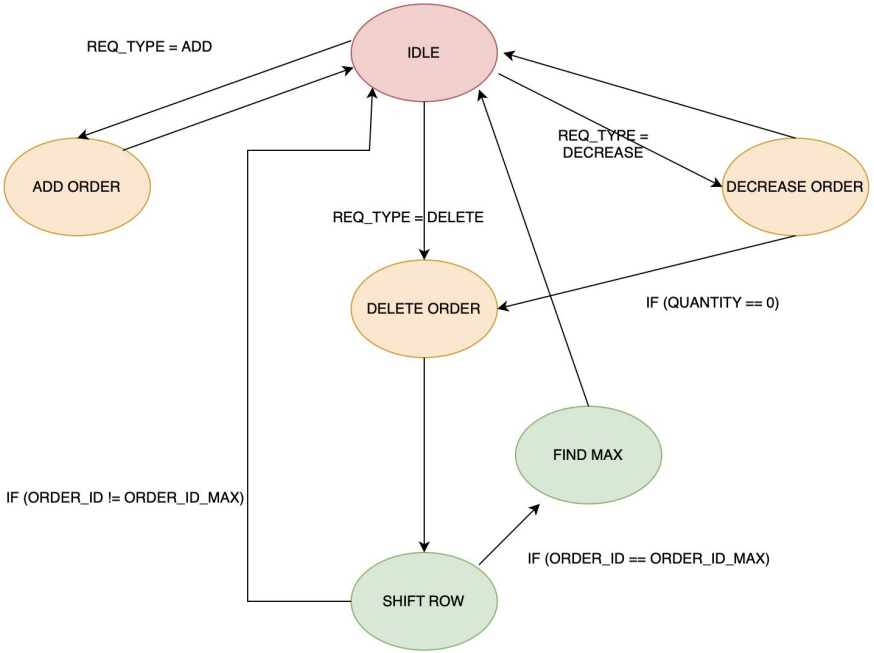


HW Design - Order Book



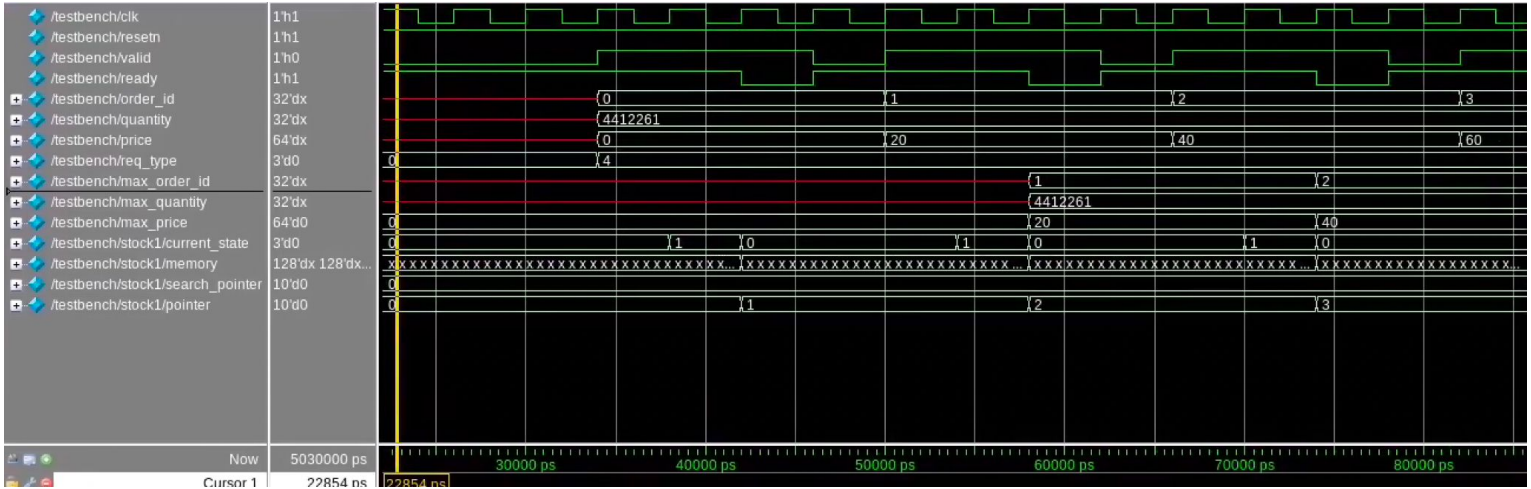

```
module order_book (
input clk,
input resetn,
input valid,
input [31:0] order_id,
input [31:0] quantity,
input [63:0] price,
input [2:0] req_type,
output reg [31:0] max_order_id,
output reg [31:0] max_quantity,
output reg [63:0] max_price,
output reg ready
);
```

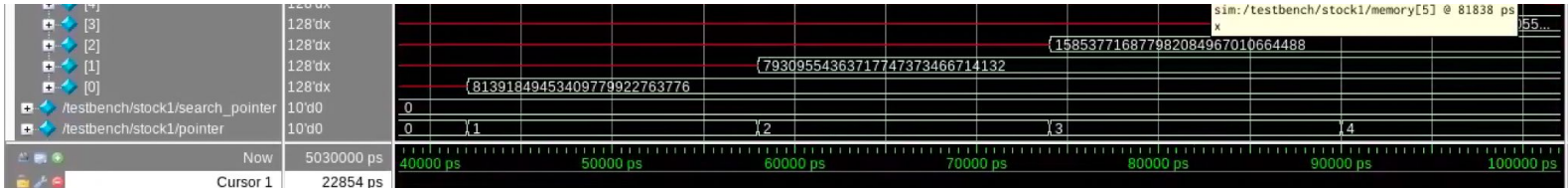
HW Design - Order Book - Add Order



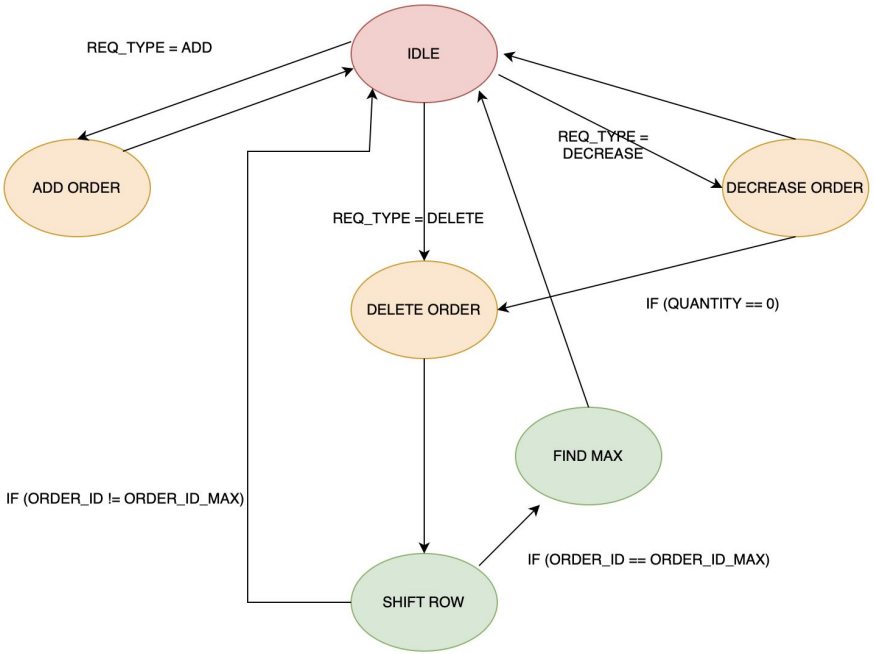
Memory	
Address	Data
1	XXXXXXXXXXXXX
2	XXXXXXXXXXXXX
3	XXXXXXXXXXXXX
4	XXXXXXXXXXXXX
5	XXXXXXXXXXXXX
6	XXXXXXXXXXXXX
7	XXXXXXXXXXXXX
8	XXXXXXXXXXXXX
9	
10	
11	
12	
.	
.	
.	
.	
992	
993	
994	
995	
996	
997	
998	
999	
1000	

HW Design - Order Book - Add Order - Wave



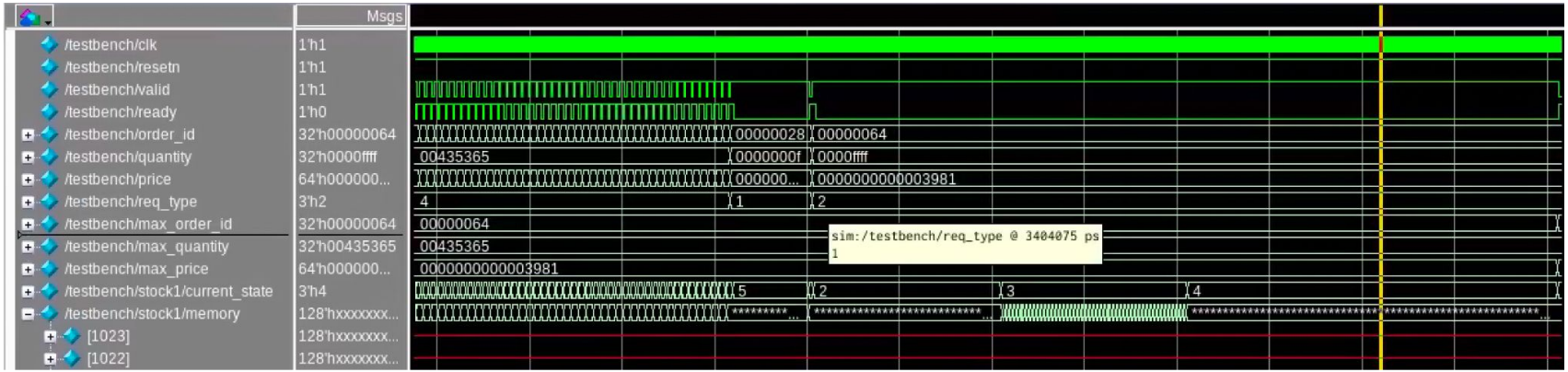


HW Design - Order Book - Decrease Order - Wave

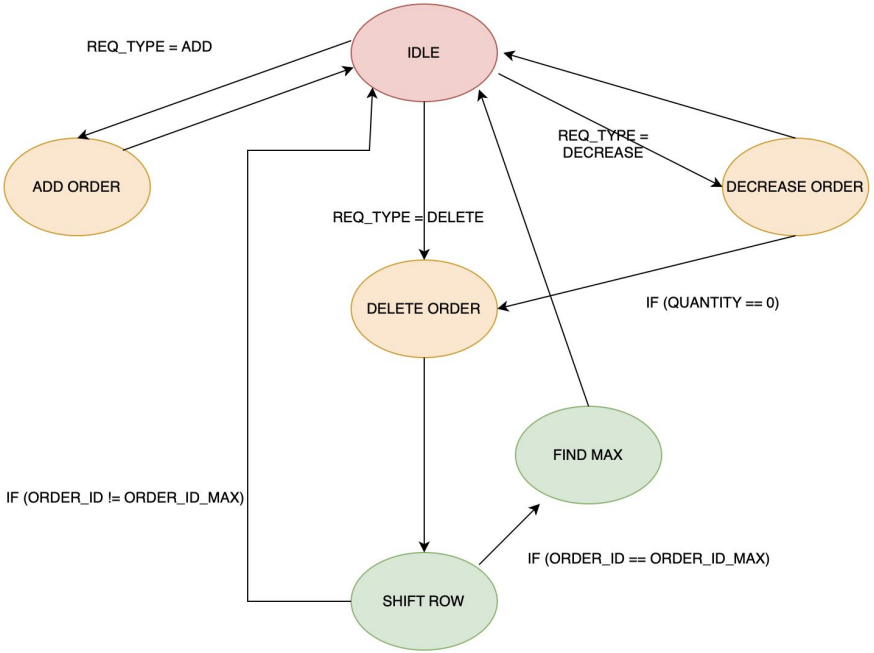


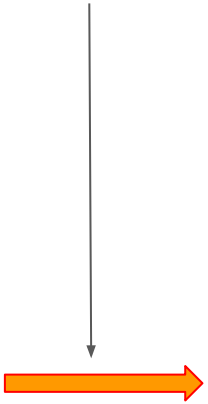
Memory	
Address	Data
1	XXXXXXXXXXXXX
2	XXXXXXXXXXXXX
3	XXXXXXXXXXXXX
4	XXXXXXXXXXXXX
5	XXXXXXXXXXXXX
6	XXXXXXXXXXXXX
7	XXXXXXXXXXXXX
8	XXXXXXXXXXXXX
9	XXXXXXXXXXXXX
10	XXXXXXXXXXXXX
11	XXXXXXXXXXXXX
12	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
992	XXXXXXXXXXXXX
993	
994	
995	
996	
997	
998	
999	
1000	

HW Design - Order Book - Decrease Order - Wave



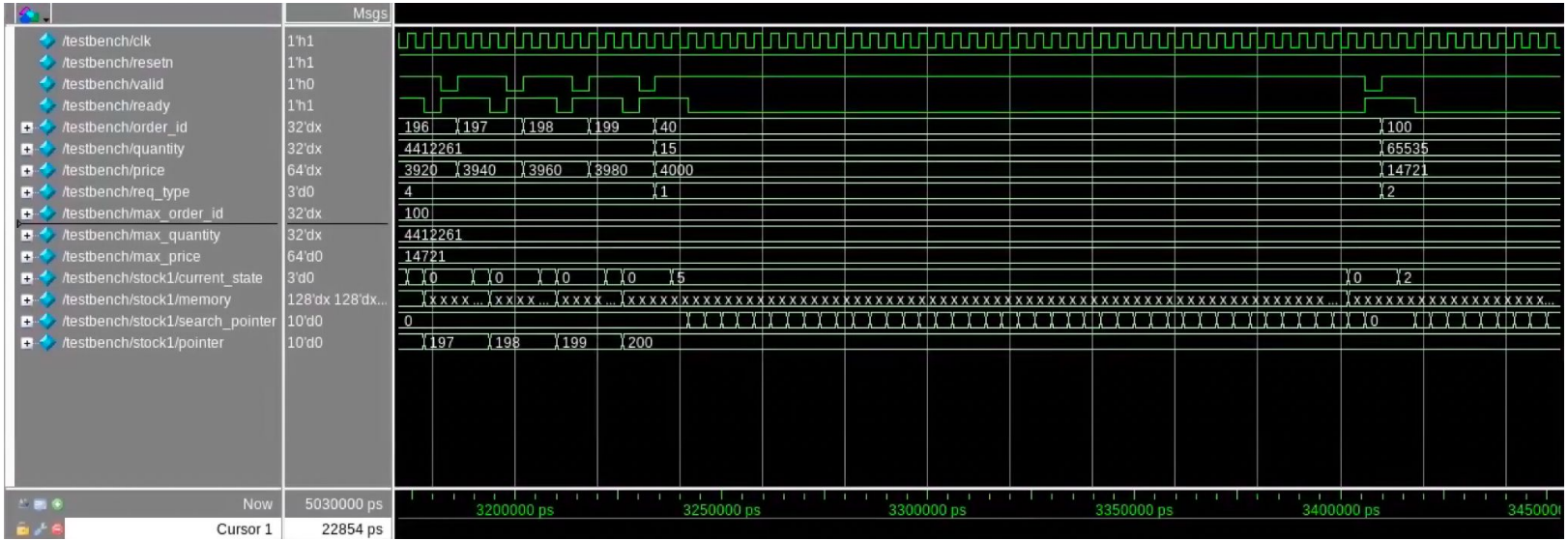
HW Design - Order Book - Delete Order



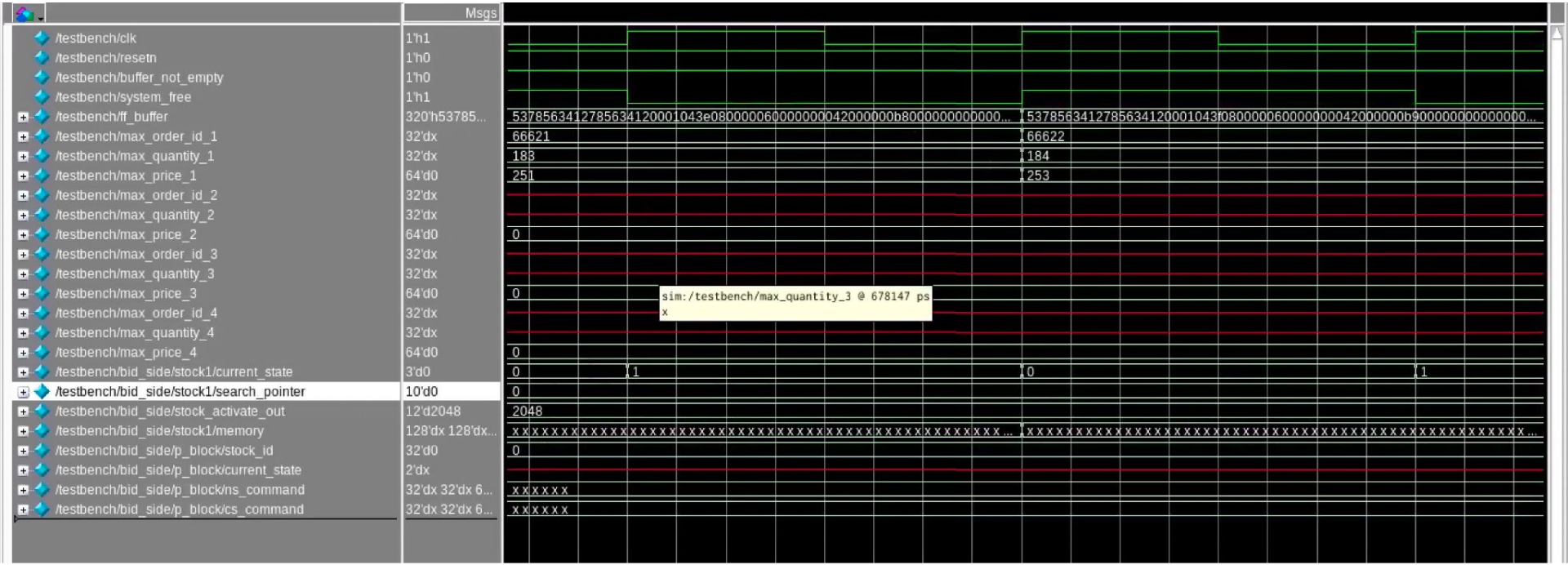


Memory	
Address	Data
1	XXXXXXXXXXXXX
2	XXXXXXXXXXXXX
3	XXXXXXXXXXXXX
4	XXXXXXXXXXXXX
5	XXXXXXXXXXXXX
6	XXXXXXXXXXXXX
7	XXXXXXXXXXXXX
8	XXXXXXXXXXXXX
9	XXXXXXXXXXXXX
10	XXXXXXXXXXXXX
11	XXXXXXXXXXXXX
12	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
992	XXXXXXXXXXXXX
993	
994	
995	
996	
997	
998	
999	
1000	

HW Design - Order Book - Delete Order - Wave



HW Design - Order Book - Overall Window



HW Design - Performance

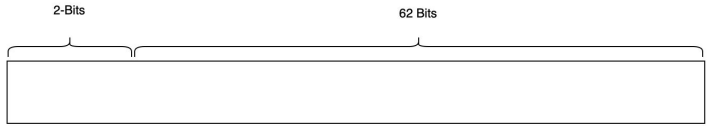
Request Type	Number of Cycle
ADD ORDER	1 cycle
DELETE ORDER	N cycles
DELETE ORDER is MAX	N + N - 1
DECREASE ORDER	~Depends on the position

Here N is the number of valid entries in the Book

HW Design - Optimizations

Memory	
Address	Data
1	XXXXXXXXXXXXX
2	XXXXXXXXXXXXX
3	XXXXXXXXXXXXX
4	XXXXXXXXXXXXX
5	XXXXXXXXXXXXX
6	XXXXXXXXXXXXX
7	XXXXXXXXXXXXX
8	XXXXXXXXXXXXX
9	XXXXXXXXXXXXX
10	XXXXXXXXXXXXX
11	XXXXXXXXXXXXX
12	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
992	XXXXXXXXXXXXX
993	
994	
995	
996	
997	
998	
999	
1000	

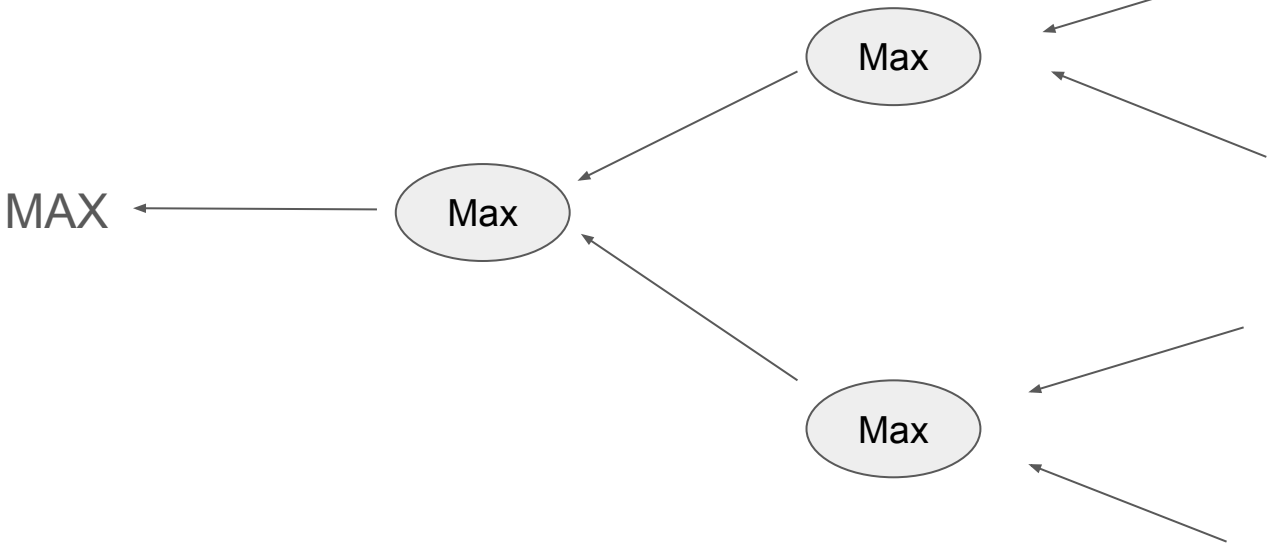
1-way memory



4-way memory

Memory	
Address	Data
1	XXXXXXXXXXXXX
2	XXXXXXXXXXXXX
3	XXXXXXXXXXXXX
4	
5	
6	
7	XXXXXXXXXXXXX
8	XXXXXXXXXXXXX
9	XXXXXXXXXXXXX
10	XXXXXXXXXXXXX
11	
12	
.	
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
992	XXXXXXXXXXXXX
993	
994	
995	XXXXXXXXXXXXX
996	XXXXXXXXXXXXX
997	XXXXXXXXXXXXX
998	XXXXXXXXXXXXX
999	
1000	

HW Design - Optimizations



Memory	
Address	Data
1	XXXXXXXXXXXXX
2	XXXXXXXXXXXXX
3	XXXXXXXXXXXXX
4	
5	
6	
7	XXXXXXXXXXXXX
8	XXXXXXXXXXXXX
9	XXXXXXXXXXXXX
10	XXXXXXXXXXXXX
11	
12	
.	
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
.	XXXXXXXXXXXXX
992	XXXXXXXXXXXXX
993	
994	
995	XXXXXXXXXXXXX
996	XXXXXXXXXXXXX
997	XXXXXXXXXXXXX
998	XXXXXXXXXXXXX
999	
1000	

HW Design - Performance

Extra Combinational Logic and States - did 10 ways

Request Type	Number of Cycle
ADD ORDER	1 cycle
DELETE ORDER	~N/4 cycles
DELETE ORDER is MAX	N/4 + 3 Cycles
DECREASE ORDER	~Depends on the position (/4 the previous time)

Here N is the number of valid entries in the Book

Demo

Thank You!