A Simple Virtualized HFT Simulation

5/13/2026

Jayden Lee-Sin, Derrick Bassey, Carlos Espinoza

Data credit: Brandon Sahly

Project Overview

We aim to simulate 8 stocks being traded simultaneously using real historical trade data.

1.Algorithms

In order to identify trades we implement two binary heaps per stock, a min heap for asks and a max heap for bids.

2.Resources

Orders are stored within 2 private pages per stock, any overflow is redirected to public pages.

3.Communication

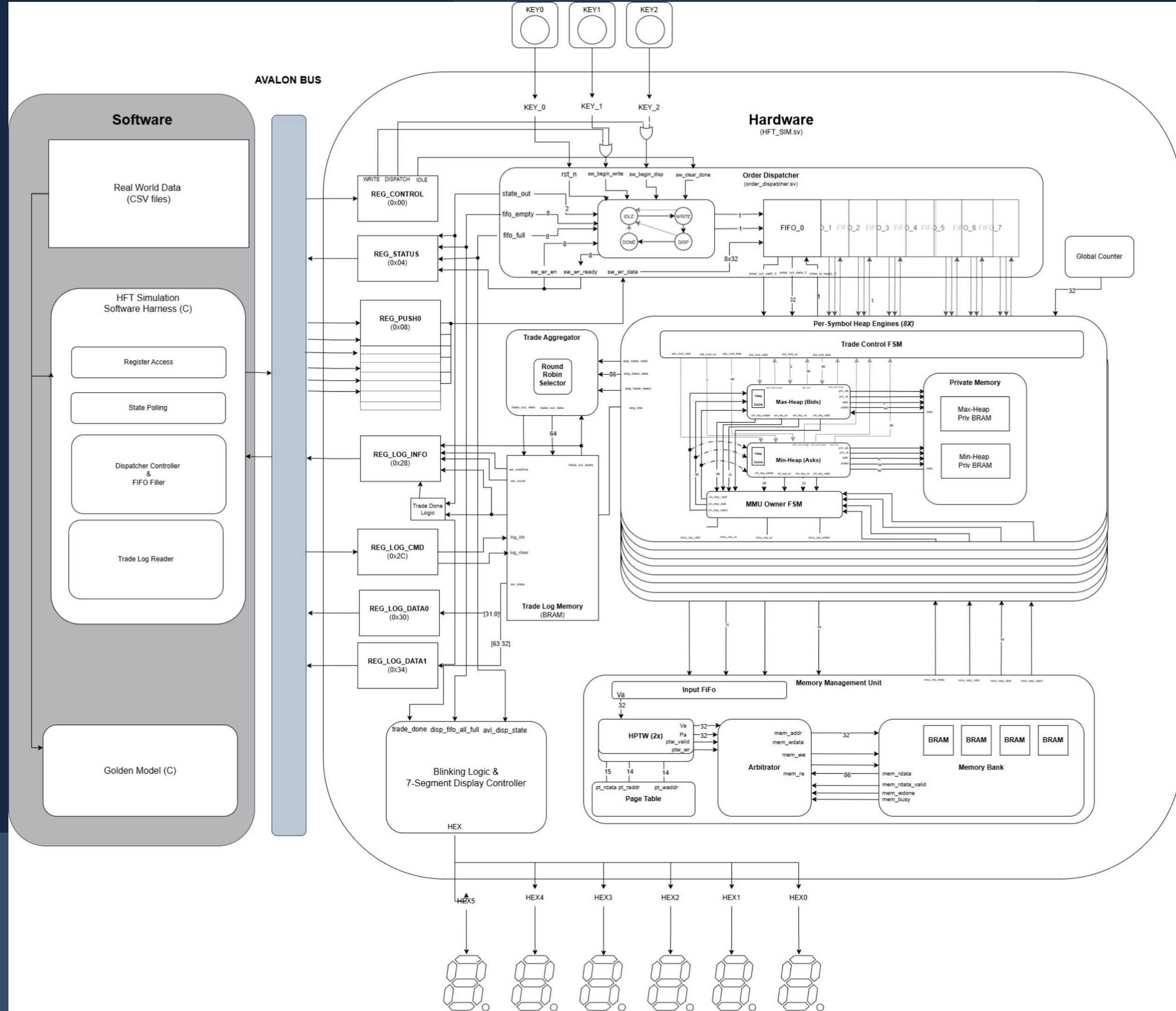
Software inputs the trade data by filling FiFos for each stock which are drained into the heap engines. After simulation, software reads the result from memory

4.Validation

A golden model written in C serves as a point of comparison for our hardware results.

System Level Overview

- Orders are submitted from the Order Generator in software to the Order Dispatch Unit in hardware.
- Once all 8 FiFos in the Order Dispatch Unit are full, each FiFo is drained into the respective Heap Engine.
- Each Heap Engine manages the two binary heaps, caches the top three nodes of each heap, detects when a trade is should occur, and stores the nodes within its own private memory. Each stock has its own heap engine.
- When a Heap Engine is full it sends the overflow nodes into a shared memory pool controlled by the MMU, which is accessed via virtual address.
- Each trade is stored in the Trade Logger, which is a pool of memory. After all trades have been completed, the Output Logger reads the trade history back into software.



Resource Consumption

Memory

394 M10k RAM Blocks

- Public Memory: 240
- MMU: 6
- Order Dispatcher: 56
- Heap Engines: 24
- Trade Logger: 68

Logic

28,456.4 ALMs

- Public Memory: 3,390.4
- MMU: 3,854.3
- Order Dispatcher: 524.7
- Heap Engines: 20,364.2
- Trade Logger: 137.7
- SoC Overhead: 211

Timing

Slow 1100mV 85C Model:

53.85 MHz

Harness & User Interface - Software

- The harness loads data from .csv files and injects them over the avalon bus using round robin selection
- Monitors trading progress
- Users can push three buttons
  - A reset button
  - A transition to write in the Dispatcher's FSM
  - A transition to dispatch in the Dispatcher's FSM
- Uses ioread32 and iowrite32 to interact with hardware

Order Dispatcher - Hardware/Software

- An FSM consisting of four states
  - IDLE
  - WRITE
  - DISPATCH
  - DONE
- Write indicates that the FiFos can be written with data
- Dispatch pops entries from the FiFos into the heap
- Done informs software that FiFos are empty
- FiFos are written over the avalon bus

dispatch_fifo_bram.sv

Port Name	Direction	Width/Type
clk	input	1
rst_n	input	1
wr_en	input	1
wr_data	input	WIDTH
full	output	1
dispatch_en	input	1
out_ready	input	1
out_valid	output	1
out_data	output	WIDTH
empty	output	1
clear	input	1

order_dispatcher.sv

Port Name	Direction	Width/Type
clk	input	1
rst_n	input	1
sw_begin_write	input	1
sw_begin_dispatch	input	1
sw_clear_done	input	1
sw_wr_en	input	8
sw_wr_data	input	32
sw_wr_ready	output	8
state_out	output	2
fifo_empty	output	8
fifo_full	output	8
order_in_ready	input	8
order_out_valid	output	8
order_out	output	32

Heap Engine - Hardware

- Receives orders from the dispatcher, then extends them to contain the proper node data used internally
- Caches the top three nodes of each tree
- Stores the nodes in binary heaps, one for asks one for bids, and detects trades both full and partial
- Heaps can push, pop, peek, and modify the top node
- Each heap has 1 private memory page, giving each symbol a total of 2 private pages

heap_fsm.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
cmd_valid	input	1
cmd_op	input	2
cmd_data_in	input	86
cmd_ready	output	1
cmd_done	output	1
cmd_root_out	output	86
size_out	output	14
priv_we	output	1
priv_re	output	1
priv_addr	output	6
priv_wdata	output	86
priv_rdata	input	86
virt_req_valid	output	1
virt_req_va	output	32
virt_req_wr	output	1
virt_req_wdata	output	86
virt_req_ready	input	1
virt_resp_valid	input	1
virt_resp_data	input	86
virt_resp_reject	input	1

symbol_engine.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
now_ts	input	32
order_in_valid	input	1
order_in_data	input	32
order_in_ready	output	1
trade_out_valid	output	1
trade_out_data	output	86
trade_out_ready	input	1
mmu_req_valid	output	1
mmu_req_va	output	32
mmu_req_wr	output	1
mmu_req_wdata	output	86
mmu_req_ready	input	1
mmu_resp_data	input	86
mmu_resp_valid	input	1
mmu_resp_reject	input	1
bid_size_o	output	14
ask_size_o	output	14
engine_idle	output	1

priv_bram.sv

Port Name	Direction	Width (Bits)
clk	input	1
we	input	1
re	input	1
addr	input	log2(DEPTH)
wdata	input	WIDTH
rdata	output	WIDTH

Heap Engine: Trade Control FSM

T_IDLE: waiting for next order

T_PUSH: inserting the new order

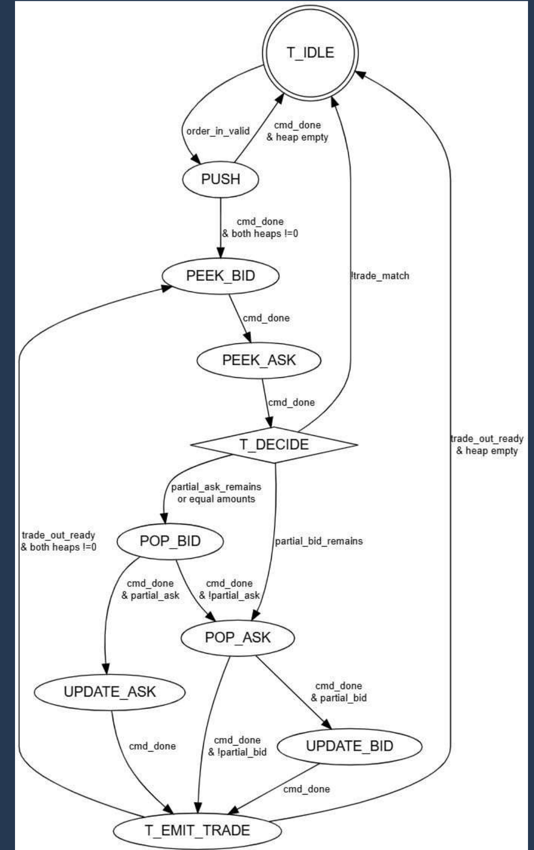
T_PEEK_BID / T_PEEK_ASK: read both roots

T_DECIDE: branches to match or no-match

T_POP_BID / T_POP_ASK: remove a heap root

T_UPDATE_BID / T_UPDATE_ASK: change quantity on a partial fill

T_EMIT_TRADE: push the trade out to the aggregator



Heap Engine: Heap FSM

S_IDLE: waiting for the next command from the trade controller

PUSH_LAUNCH: starting a push; decide between empty-heap fast path and sift-up

PUSH_SIFT: sift-up loop, read parent, compare, swap if needed, repeat

PUSH_FINAL: write the new node at its settled position

POP_LAUNCH: starting a pop; handle the 1-node special case

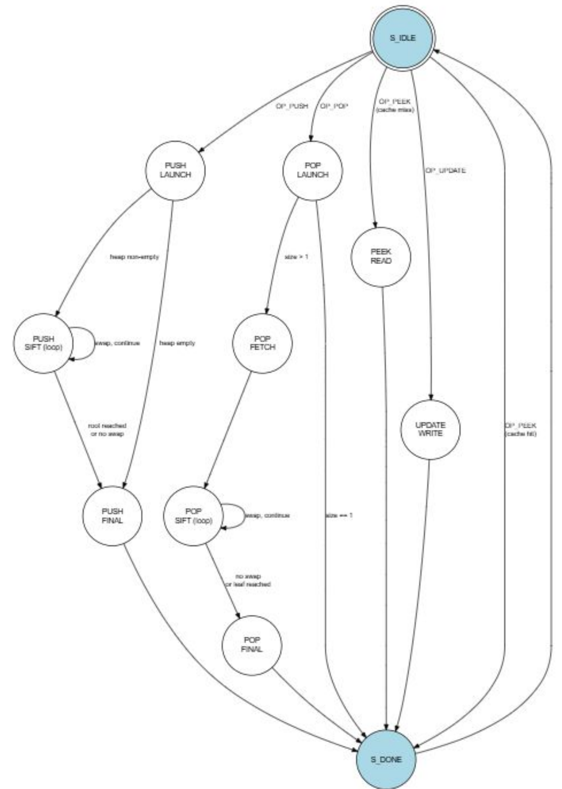
POP_FETCH: read the root (to return) and the last leaf (to replace it)

POP_SIFT: sift-down loop, read both children, compare, swap with winner, repeat

POP_FINAL: write the moved node at its settled position

PEEK_READ: read the root from BRAM/MMU (skipped entirely on top-3 cache hit)

UPDATE_WRITE: overwrite the root with the new node (used after partial fills)
S_DONE: pulse cmd_done, return to S_IDLE



MMU - Hardware

- The MMU takes in requests from heap engines comprised of a virtual address, a valid bit, and data to write
- It uses round robin to select two winners
- Winners are pushed into FiFos
- Fifos feed the HPTWs
- The HPTWs use the page table to translate the virtual address into a physical one
- Physical requests are passed to the arbiter
- The arbiter sorts the request into one of four FiFos and track requests, relating it back to its virtual address
- The FiFos feed physical memory banks
- Physical memory outputs back to the arbiter which re-associates data and virtual address, allowing the MMU to forward data to the correct requester

mmu.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
req_valid_#	input	1
req_va_#	input	32
req_wr_#	input	1
req_wdata_#	input	86
req_ready_#	output	1
resp_data_#	output	86
resp_valid_#	output	1
resp_reject_#	output	1
mem_addr_#	output	32
mem_we_#	output	1
mem_re_#	output	1
mem_wdata_#	output	86
mem_rdata_#	input	86
mem_rdata_vali	input	1
mem_busy_#	input	1
mem_wdone_#	input	1

HPTW.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
va_valid	input	1
va	input	32
pa_valid	output	1
pa	output	32
bank_id	output	2
fault	output	1
busy	output	1
pt_raddr	output	14
pt_rdata	input	15
pt_we	output	1
pt_waddr	output	14
pt_wdata	output	15
alloc_avail	input	1
alloc_page_in	input	8
alloc_node_in	input	6
alloc	output	1
alloc_page_idx	output	8
alloc_node_idx	output	6

arbiter.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
ptw0_valid	input	1
ptw0_va	input	32
ptw0_pa	input	32
ptw0_bank_id	input	2
ptw0_wr	input	1
ptw0_wdata	input	86
ptw1_valid	input	1
ptw1_va	input	32
ptw1_pa	input	32
ptw1_bank_id	input	2
ptw1_wr	input	1
ptw1_wdata	input	86
ptw0_accept	output	1
ptw0_reject	output	1
ptw1_accept	output	1
ptw1_reject	output	1
rdata_#	output	86
rdata_va_#	output	32
rdata_valid_#	output	1
mem_addr_#	output	32
mem_we_#	output	1
mem_re_#	output	1
mem_wdata_#	output	86
mem_rdata_#	input	86
mem_rdata_vali	input	1
mem_wdone_#	input	1
mem_busy_#	input	1

mem_bank.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
mem_addr	input	32
mem_we	input	1
mem_re	input	1
mem_wdata	input	86
mem_rdata	output	86
mem_rdata_vali	output	1
mem_wdone	output	1
mem_busy	output	1

bram_dp_256x32.sv

Port Name	Direction	Width (Bits)
clk	input	1
we	input	1
re	input	1
waddr	input	8
raddr	input	8
wdata	input	32
rdata	output	32

tracking_fifo (arbiter.sv)

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
push	input	1
din	input	WIDTH
pop	input	1
dout	output	WIDTH
full	output	1
empty	output	1

dual_write_fifo (arbiter.sv)

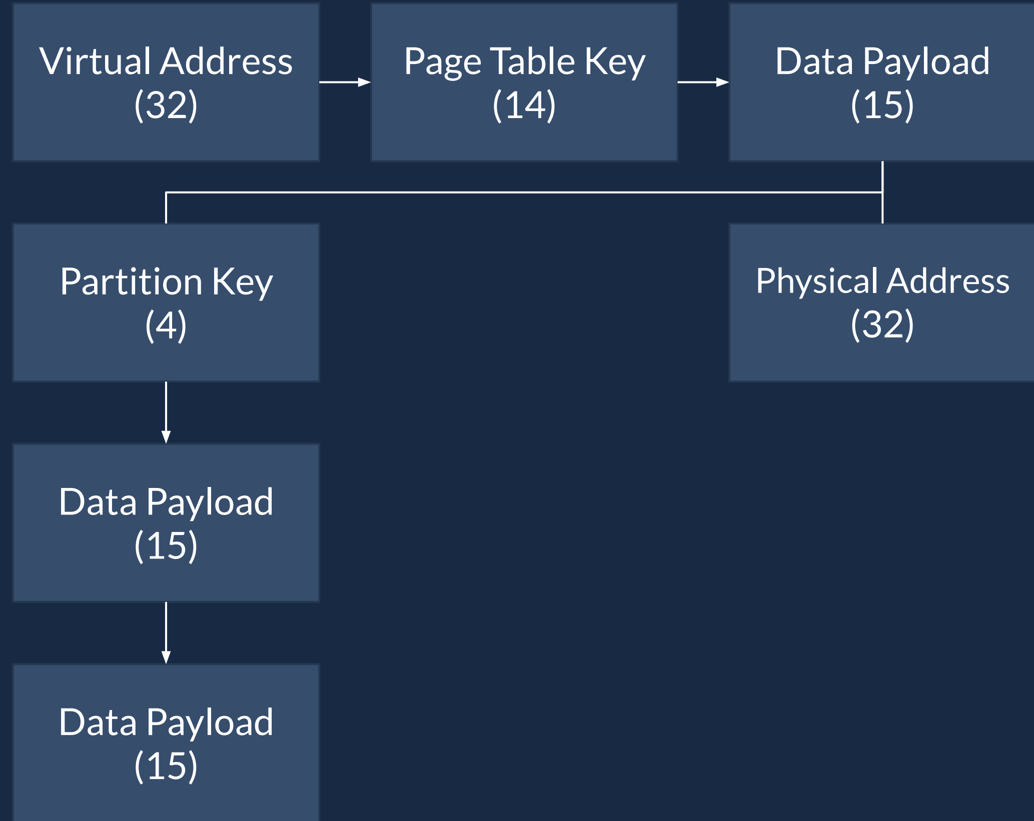
Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
wr0_en	input	1
wr0_data	input	WIDTH
wr1_en	input	1
wr1_data	input	WIDTH
rd_en	input	1
rd_data	output	WIDTH
empty	output	1
full	output	1
count	output	$clog2(DEPTH+1)

Fifo (mmu.sv)

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
wr_en	input	1
wr_data	input	WIDTH
rd_en	input	1
rd_data	output	WIDTH
empty	output	1
count	output	4

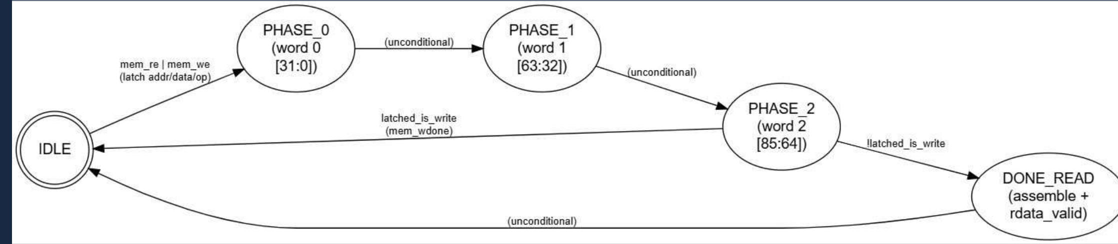
MMU - Hardware

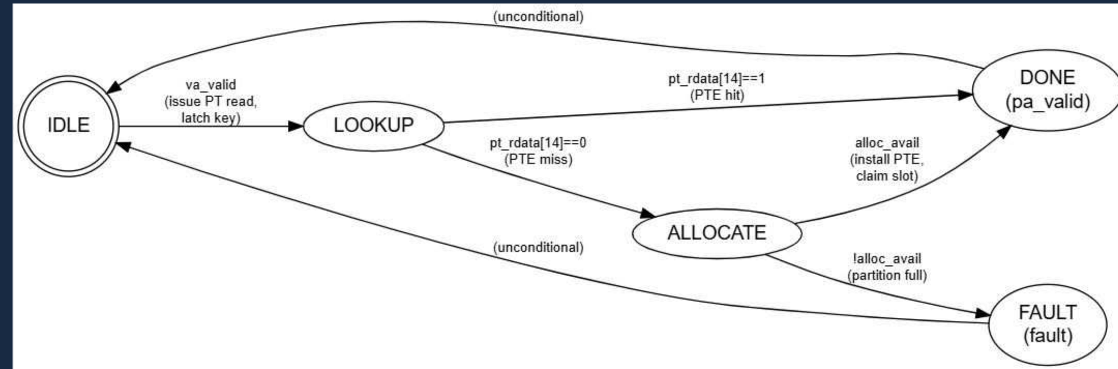
Aside on the page table: The page table works on a per-node basis, the HPTW extracts a 14-bit key from the 32-bit virtual address comprised of the engine ID and bottom 11 bits. Then reads the page table at that key, it returns a 15-bit [valid[1], physical page index[8], and node index [6]]. If the valid bit is 1 then this node has already been allocated, and a physical address is computed as: 11'd0 + physical page index [8] + 3'd0 + node index [6] + 4'd0. On a miss, the mmu selects a new node and page. This is done using a partition system, there are 16 partitions, each with access to 15 physical pages. When requester has access to 2 partitions based on their engine ID and virtual address bit 10, upon request the partition provides the current node tracked by a counter, when the counter rolls over the partition increases the page.



MMU - Hardware

- State machines for memory bank and page table walker





Trade Log - Hardware

- The trade aggregator uses round robin selection to forward confirmed trades to the trade log
- The trade log is a single port memory that has a depth supporting 8704 compressed trades (64 bits)
    - Created using the bottom 7 bits of amount, the 3 bit engine ID, the 16 bit price, and the 32 bit
- Uses an overflow flag to indicate when the log is full

trade_aggregator.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
eng_trade_valid	input	8
eng_trade_data	input	86
eng_trade_ready	output	8
trade_out_valid	output	1
trade_out_data	output	64
trade_out_ready	input	1

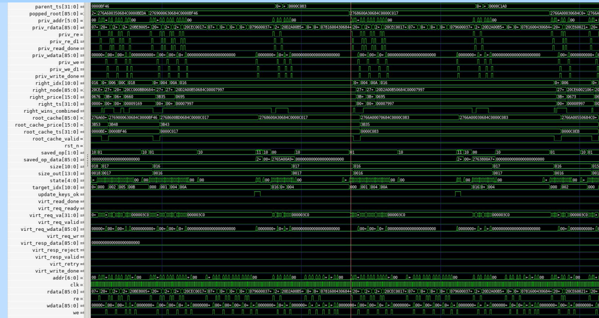
trade_log.sv

Port Name	Direction	Width (Bits)
clk	input	1
rst_n	input	1
trade_in_valid	input	1
trade_in_data	input	64
trade_in_ready	output	1
sw_re	input	1
sw_addr	input	5
sw_rdata	output	64
sw_count	output	14
sw_overflow	output	1
sw_clear	input	1

Trade Log Reader - Software

- Once the harness is in the done state, it reads the 64 bit entries in trade log
- The first word is the timestamp and the second contains the trade info
- Stores the trade results in a text file to be compared to the golden model

Hardware Validation
Golden Model
Dataset



Testbench

We created a hardware testbench that ensures our hardware simulator matches our software pre-synthesis

```
A partial fill has been executed at 180507767! Sold 3 shares of TUS at $1434. A ask of 80 shares remains.
A partial fill has been executed at 180525067! Sold 80 shares of TUS at $1432. A bid for 8 shares remains.
Bid Submitted for TUS; Price: $1414, Amount 56, Timestamp: 180528367
Ask Submitted for TUS; Price: $1411, Amount 34, Timestamp: 180530467
A partial fill has been executed at 180533167! Sold 8 shares of TUS at $1432. A ask of 26 shares remains.
A partial fill has been executed at 180535267! Sold 26 shares of TUS at $1427. A bid for 52 shares remains.
Bid Submitted for TUS; Price: $141, Amount 81, Timestamp: 180553467
Ask Submitted for TUS; Price: $1411, Amount 8, Timestamp: 180556267
A partial fill has been executed at 180558067! Sold 8 shares of TUS at $1427. A bid for 44 shares remains.
Bid Submitted for TUS; Price: $141, Amount 71, Timestamp: 180560067
Ask Submitted for TUS; Price: $1411, Amount 68, Timestamp: 180562167
A partial fill has been executed at 180563767! Sold 44 shares of TUS at $1427. A ask of 24 shares remains.
A partial fill has been executed at 180565567! Sold 24 shares of TUS at $1426. A bid for 51 shares remains.
Bid Submitted for TUS; Price: $141, Amount 45, Timestamp: 180567567
Ask Submitted for TUS; Price: $1409, Amount 28, Timestamp: 180569367
A partial fill has been executed at 180570867! Sold 28 shares of TUS at $1426. A bid for 23 shares remains.
Bid Submitted for TUS; Price: $1408, Amount 57, Timestamp: 180573067
Ask Submitted for TUS; Price: $1407, Amount 45, Timestamp: 180574767
A partial fill has been executed at 180576367! Sold 23 shares of TUS at $1426. A ask of 22 shares remains.
A partial fill has been executed at 180578067! Sold 22 shares of TUS at $1426. A bid for 19 shares remains.
Bid Submitted for TUS; Price: $1406, Amount 37, Timestamp: 180579967
Ask Submitted for TUS; Price: $1402, Amount 1, Timestamp: 180581867
A partial fill has been executed at 180583567! Sold 1 shares of TUS at $1426. A bid for 18 shares remains.
Bid Submitted for TUS; Price: $1401, Amount 66, Timestamp: 180585467
Ask Submitted for TUS; Price: $1401, Amount 85, Timestamp: 180587267
A partial fill has been executed at 180589467! Sold 18 shares of TUS at $1426. A ask of 67 shares remains.
A partial fill has been executed at 180591367! Sold 67 shares of TUS at $1421. A bid for 4 shares remains.
Bid Submitted for TUS; Price: $140, Amount 18, Timestamp: 180593367
Ask Submitted for TUS; Price: $1401, Amount 34, Timestamp: 180601067
A partial fill has been executed at 180618867! Sold 4 shares of TUS at $1421. A ask of 30 shares remains.
A partial fill has been executed at 180621467! Sold 13 shares of TUS at $1418. A ask of 17 shares remains.
A partial fill has been executed at 180623267! Sold 17 shares of TUS at $1417. A bid for 65 shares remains.
Bid Submitted for TUS; Price: $140, Amount 78, Timestamp: 186124167
Ask Submitted for TUS; Price: $1353, Amount 24, Timestamp: 186145767
A partial fill has been executed at 186148467! Sold 24 shares of TUS at $1417. A bid for 41 shares remains.
Bid Submitted for TUS; Price: $1352, Amount 90, Timestamp: 186152567
Ask Submitted for TUS; Price: $1345, Amount 34, Timestamp: 186169767
A partial fill has been executed at 186208467! Sold 34 shares of TUS at $1417. A bid for 7 shares remains.
Bid Submitted for TUS; Price: $1344, Amount 13, Timestamp: 186212967

Per-Symbol Breakdown:
MSF: 1473 trades, 96 open asks, 81 open bids, 24 pages used
BUS: 1411 trades, 137 open asks, 100 open bids, 32 pages used
AAP: 1448 trades, 104 open asks, 95 open bids, 27 pages used
MMM: 1512 trades, 52 open asks, 75 open bids, 19 pages used
SBU: 1410 trades, 151 open asks, 83 open bids, 32 pages used
WMT: 1510 trades, 64 open asks, 71 open bids, 19 pages used
BSX: 1456 trades, 110 open asks, 73 open bids, 26 pages used
TUS: 1443 trades, 89 open asks, 102 open bids, 28 pages used

Overflow:
Accesses:       0
Current usage:  0/8
Maximum usage: 0/8
Penalty cycles: 0

Totals:
Hard rejects: 0
Trades:       11663
Reads:        49804
Writes:       24809
Cycles:       228720
```

Golden Model

Our software simulation produces a text file that can be directly compared to our hardware output

	BID	ASK
MSFT	542.31	543
MSFT	542	542.17
MSFT	535.46	535.55
MSFT	533.5	533.63
MSFT	531.4	531.5
MSFT	529.27	529.5
MSFT	528.52	528.53
MSFT	527.67	527.71
MSFT	525.78	525.82

Dataset

Our dataset was collected via a bloomberg terminal. It's comprised of the daily price of each selected stock since 1993, amounts have been randomized

Outlooks & Results

- Synthesis takes long
  - Like really long
- Fitting everything in memory requires being clever - Fit is the worst
- Make things as small as possible
- FiFos and Round Robin are very useful
- Hardware Runtime:
  - 0.0015 seconds
- Software Runtime:
  - 0.471695 seconds
- ~2.1% discrepancy
- 5,694,242 trades per second
- 13,117% increase in speed
- 99.68% reduction in runtime