Matchlib: A New Open-source Library to Enable Efficient Use of High Level Synthesis

Stuart Swan, Platform Architect,

Mentor, A Siemens Business

accellera SYSTEMS INITIATIVE™

Mentor® A Siemens Business

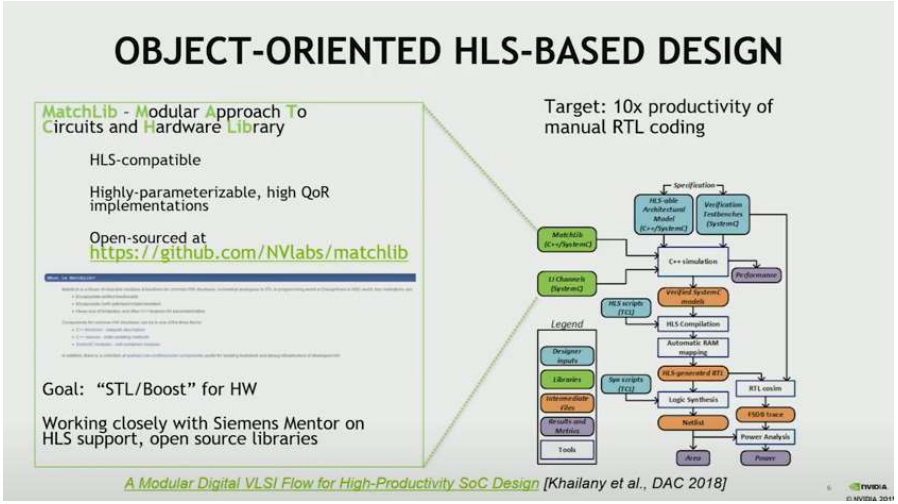
SYSTEMC™ EVOLUTION DAY
OCT 29, 2020 | VIRTUAL WORKSHOP

Copyright Permission

- A non-exclusive, irrevocable, royalty-free copyright permission is granted by **Mentor, A Siemens Business** to use this material in developing all future revisions and editions of the resulting draft and approved Accellera Systems Initiative **SystemC** standard, and in derivative works based on the standard.

What is NVIDIA MatchLib?

- Good 20 minute intro video here:
  - https://www.youtube.com/watch?v=n8_G-CaSSPU



Key Parts of MatchLib

- "Connections"
  - Synthesizeable Message Passing Framework
  - SystemC/C++ used to accurately model concurrent IO that synthesized HW will have
  - Automatic stall injection enables interconnect to be stress tested in SystemC
  - Supports message latency and capacity back-annotation into pre-HLS model
- Parameterized AXI4 Fabric Components
  - Router/Splitter
  - Arbiter
  - AXI4 <-> AXI4Lite
  - Automatic burst segmentation and last bit generation
- Parameterized Banked Memories, Crossbar, Reorder Buffer, Cache
- Parameterized NOC components

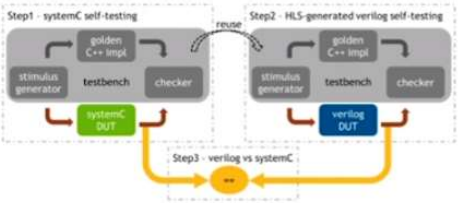
MatchLib SystemC Model Characteristics

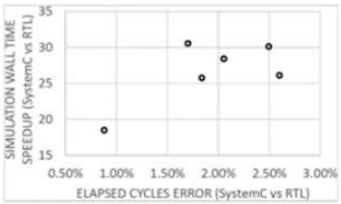
- Small
  - Typically 1/10 or less than the size of comparable RTL models
- Fast
  - Simulates ~30 times faster than RTL models in timing accurate mode
  - Simulates ~300 times faster than RTL models in blocking TLM mode (via compile time flag)
- Accurate
  - Not exactly RTL cycle accurate, but pretty close
  - Concurrent transactions in HW are modeled very accurately
- Fully automated path to placed gates via SystemC HLS
- Enables SW/FW models to be integrated via C++ host-code or CPU models
- Enables single-source model for HW and FW for full flow

MatchLib Results using HLS

RC17 SYSTEMC-BASED VERIFICATION

Functional and Performance Verification on SystemC models





FUNCTIONAL VERIFICATION

- Most verification run on SystemC/C++, signed off using C++ coverage tools
- Reuse of SystemC testbenches on HLS-generated RTL DUTs
- Automated stall injection and in-design assertions for improved coverage

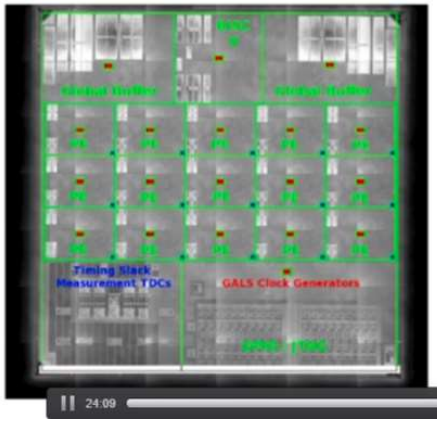
PERFORMANCE VERIFICATION

- Sim-accurate SystemC models for Latency-Insensitive Channels
- Up to 30x speedup vs. RTL
- Less than 2.6% error in cycle count

17 NVIDIA

RC17 SOC PHYSICAL DESIGN

87M Transistor SoC in TSMC 16nm FinFET



	RC17 Stats
Die Size	4 mm²
Partitions	19 (5 unique)
Frequency range	510 MHz - 1.96 GHz
Voltage range	0.55-1.2 Volts
Performance (16b GMACS)	61.2-235.2
Max GMACS/W	192.1
Programmability	ML workloads (NN inference, K-means)

NVIDIA

Complexity/Risk in Modern Designs has Shifted…

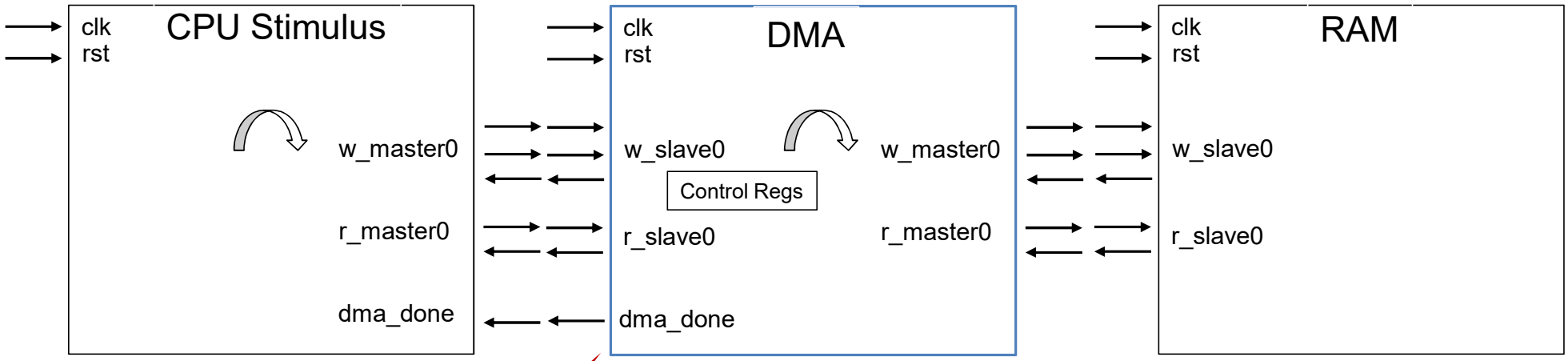
- As an example, performance of ML / Vision chips is often in terms of trillions of MACs per second
- But, design and verification of MACs is not the hard part
- Hard part is often managing the movement of data in the chip across all scenarios
- Today's HW designs often process huge sets of data, with large intermediate results.
  - Machine Learning, Computer Vision, 5G Wireless
- The design of the memory/interconnect architecture and the management of data movement in the system often has more impact on power/performance than the design of the computation units themselves.

MatchLib + SystemC HLS Addresses Complexity / Risk in Modern Designs

- Evaluating and verifying memory/interconnect architecture at RTL level is often not feasible:
  - Too late in design cycle.
  - Too much work to evaluate multiple candidate architectures.

- The most difficult/costly HW (& HW/SW) problems are found during system integration.
  - If integration first occurs in RTL, it is very late and problems are very costly.
  - MatchLib + SystemC HLS lets integration occur early when fixing problems is much cheaper.

Simple Example: AXI4 DMA using MatchLib



The CPU Stimulus programs DMA control registers

The design to be synthesized with HLS is a DMA

```
/**
 *  * \brief dma module
*/
#pragma hls_design top
class dma : public sc_module, public local_axi {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  r_master INIT_S1(r_master0);
  w_master INIT_S1(w_master0);
  r_slave  INIT_S1(r_slave0);
  w_slave  INIT_S1(w_slave0);
  Connections::Out<bool> INIT_S1(dma_done);
```

The DMA reads and writes to the RAM

= top level of design

The DMA performs a memory copy using AXI4 bursts

Entire AXI4 DMA C++ is 170 lines
RTL after HLS is 6000 lines

```
85   void master_process() {
86     AXI4_W_SEGMENT_RESET(w_segment0, w_master0);
87     AXI4_R_SEGMENT_RESET(r_segment0, r_master0);
88
89     dma_cmd_chan.ResetRead();
90     dma_dbg.Reset();
91     dma_done.Reset();
92
93     wait();
94
95     while(1) {
96       ex_ar_payload ar;
97       ex_aw_payload aw;
98
99       dma_cmd cmd = dma_cmd_chan.Pop();
100      ar.ex_len = cmd.len;
101      aw.ex_len = cmd.len;
102      ar.addr = cmd.ar_addr;
103      aw.addr = cmd.aw_addr;
104      r_segment0_ex_ar_chan.Push(ar);
105      w_segment0_ex_aw_chan.Push(aw);
106
107      #pragma hls_pipeline_init_interval 1
108      #pragma pipeline_stall_mode flush
109      while (1) {
110       r_payload r = r_master0.r.Pop();
111       w_payload w;
112       w.data = r.data;
113       w_segment0_w_chan.Push(w);
114
115       if (ar.ex_len-- == 0)
116        break;
117      }
118
119      b_payload b;
120      b = w_segment0_b_chan.Pop();
121      dma_done.Push(true);
122    }
123  }
```

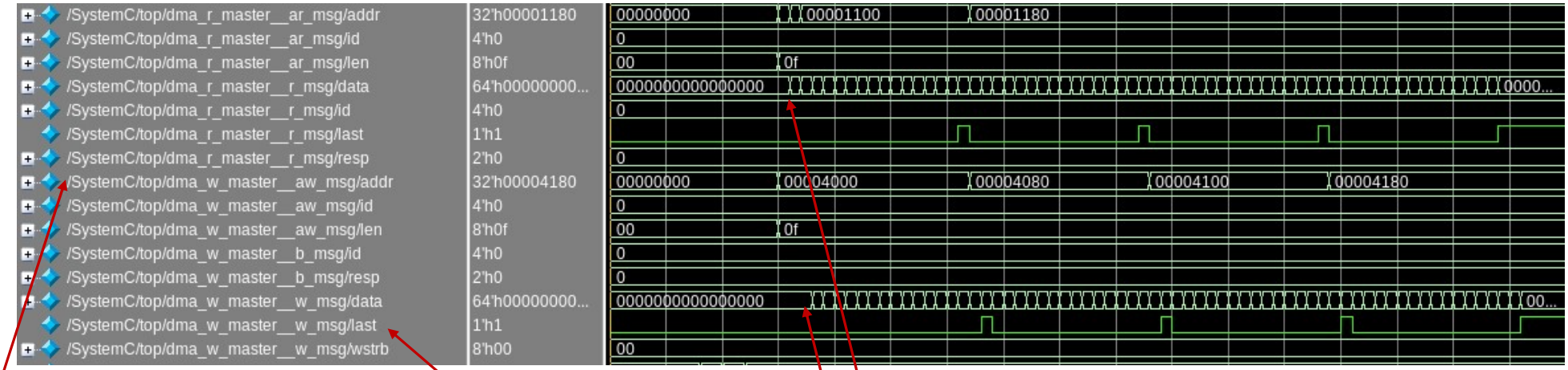
The only clock/wait is for reset state

This IO is in parallel

Main compute loop gets pipelined in HLS

This IO is in parallel

AXI4 DMA Waveforms Before HLS (SystemC simulation)



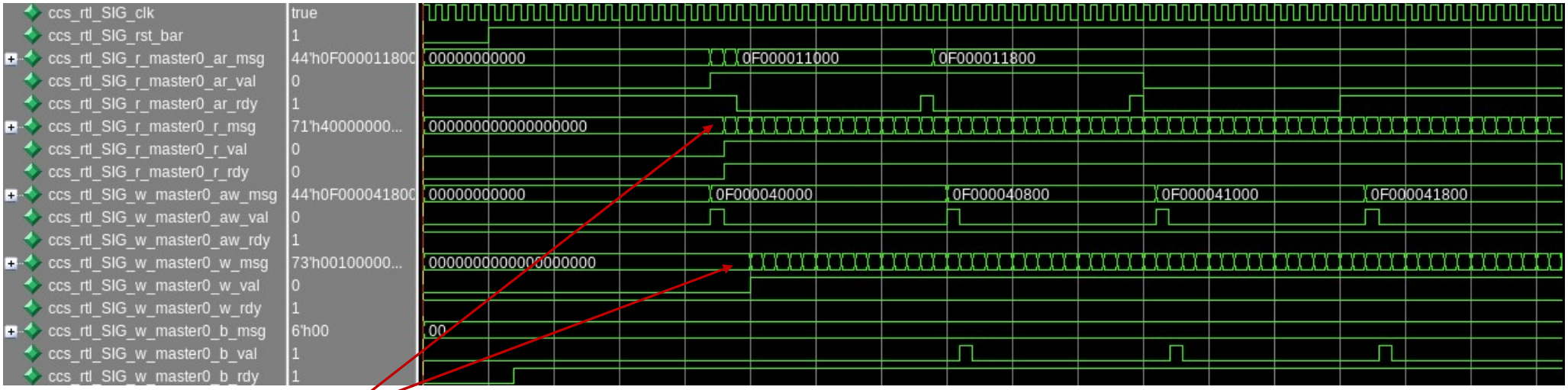
Automatic AXI4 last bit generation

Read and write burst streams are concurrent.

Read/write bus utilization is 100%

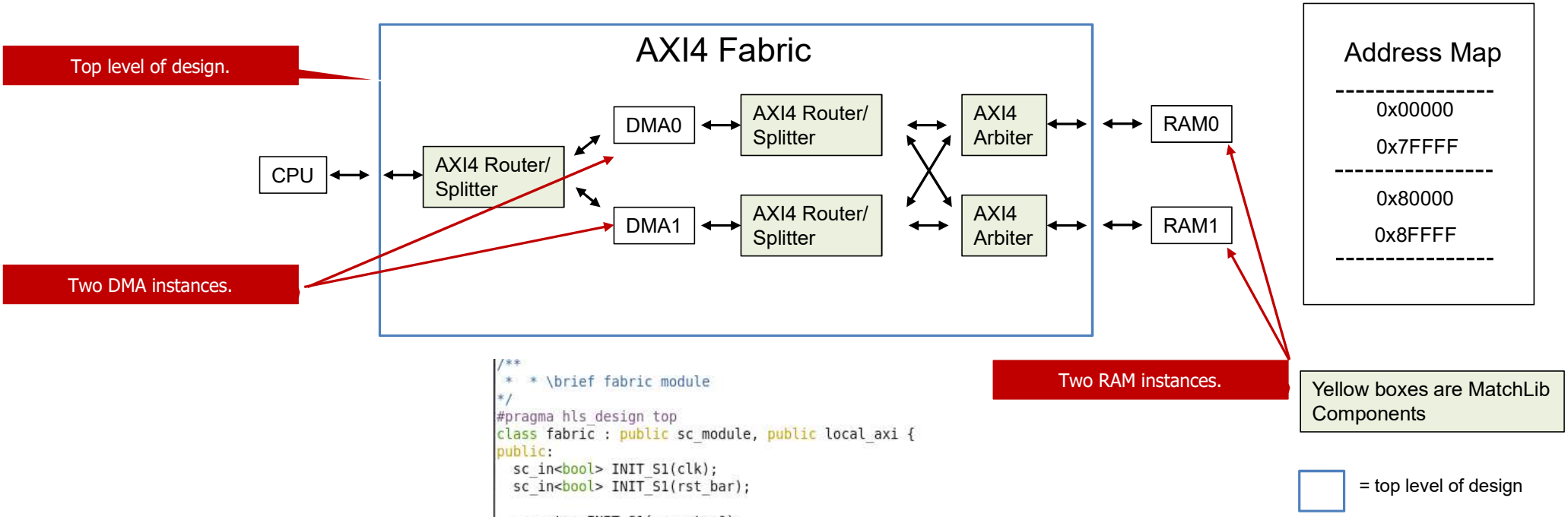
Automatic AXI4 burst address segmentation

AXI4 DMA Waveforms After HLS (Verilog Sim)



RTL waveforms are almost same as SystemC waveforms. Throughput is the same.

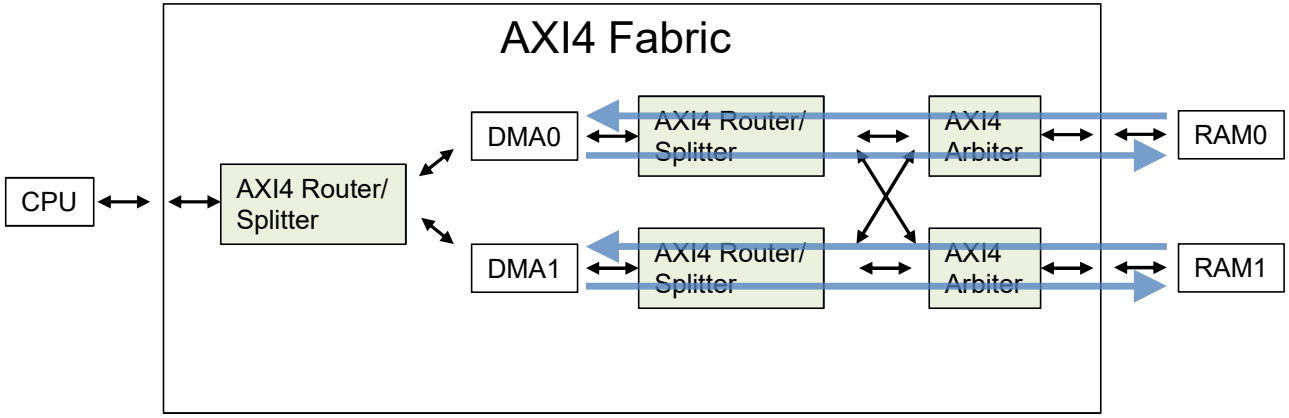
Larger Example: AXI4 Bus Fabric using MatchLib




```
/**
 *  * \brief fabric module
*/
#pragma hls_design top
class fabric : public sc_module, public local_axi {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  r_master INIT_S1(r_master0);
  w_master INIT_S1(w_master0);
  r_master INIT_S1(r_master1);
  w_master INIT_S1(w_master1);
  r_slave  INIT_S1(r_slave0);
  w_slave  INIT_S1(w_slave0);
  Connections::Out<bool> INIT_S1(dma0_done);
  Connections::Out<bool> INIT_S1(dma1_done);
```

AXI4 Bus Fabric using MatchLib – Test #0



RAM0 and RAM1 each have one read and one write port

Test #0: Concurrently,
DMA0 reads/writes 320 beats to RAM0
DMA1 reads/writes 320 beats to RAM1

AXI4 Bus Fabric Test #0 simulation logs

BEFORE HLS (SystemC simulation)

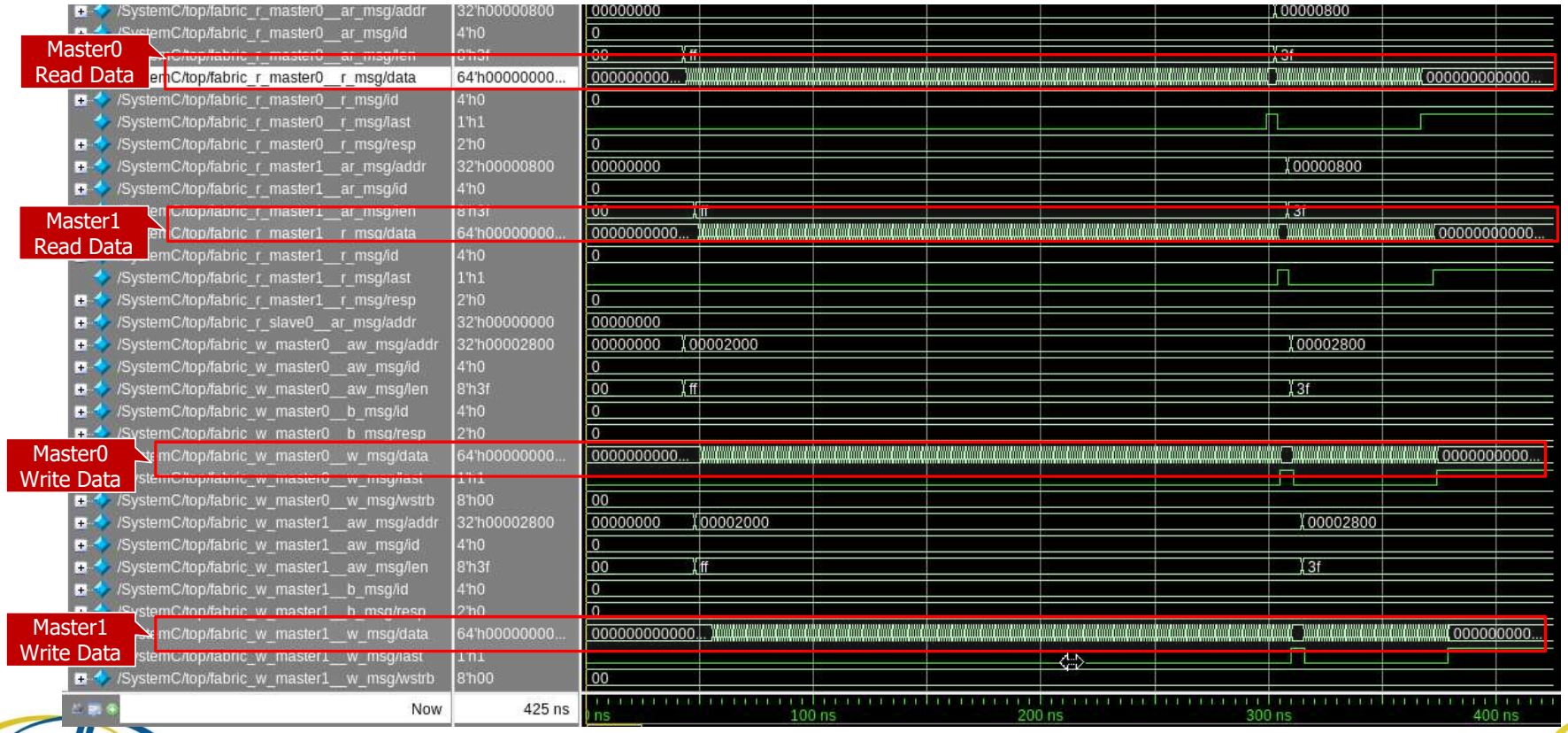
```
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 0
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 len: 03f
309 ns top.ram1 ram read  addr: 000000800 len: 03f
311 ns top.ram0 ram write addr: 000002800 len: 03f
316 ns top.ram1 ram write addr: 000002800 len: 03f
385 ns top dma_done detected. 1 1
385 ns top start_time: 46 ns end_time: 385 ns
385 ns top axi beats (dec): 320
385 ns top elapsed time: 339 ns
385 ns top beat rate: 1059 ps
385 ns top clock period: 1 ns
425 ns top finished checking memory contents
```

AFTER HLS (Verilog RTL simulation)

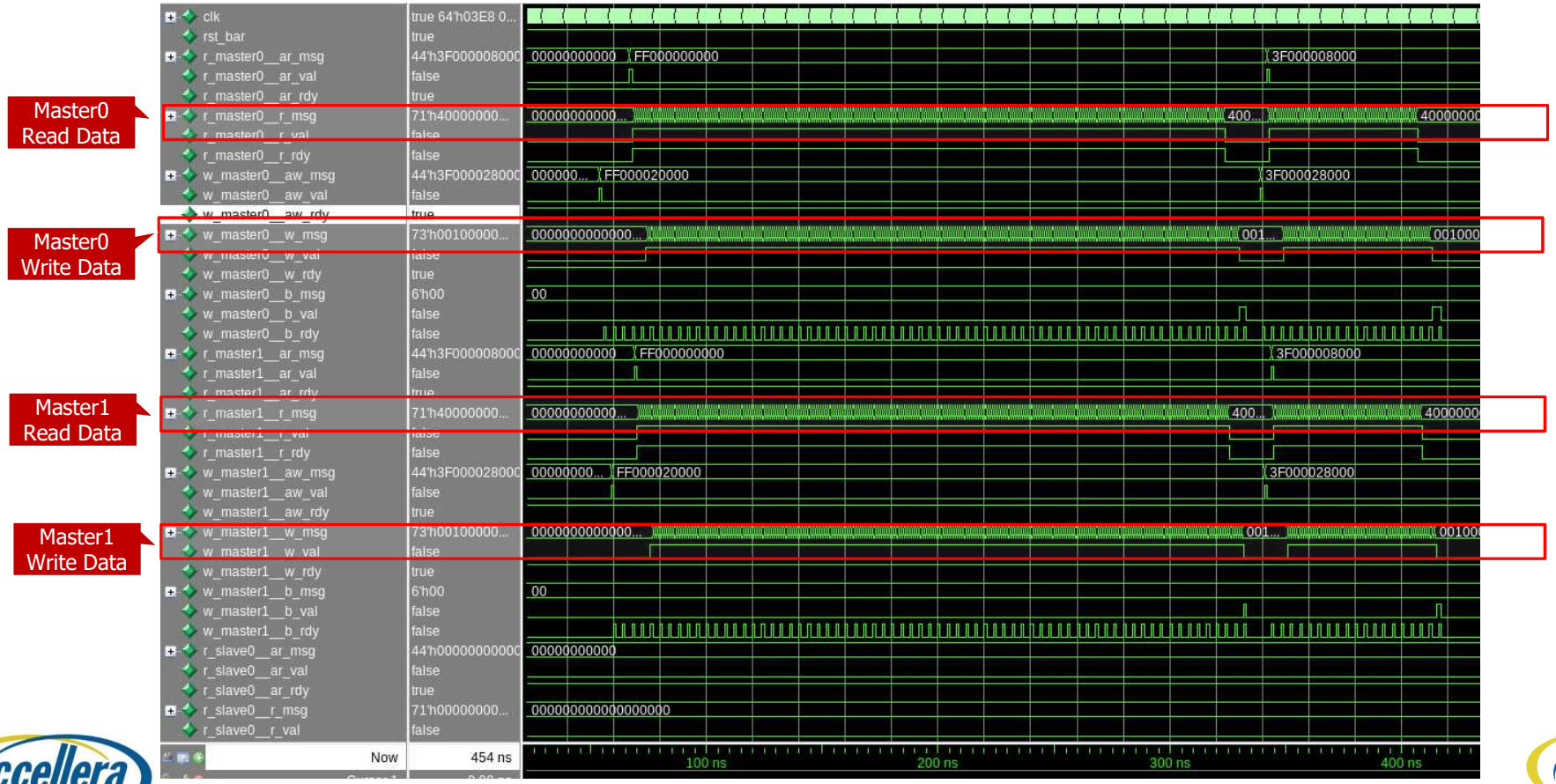
```
# 0 s top Stimulus started
# 6 ns top Running FABRIC_TEST # : 0
# 55 ns top/ram0 ram write addr: 000002000 len: 0ff
# 60 ns top/ram1 ram write addr: 000002000 len: 0ff
# 68 ns top/ram0 ram read  addr: 000000000 len: 0ff
# 70 ns top/ram1 ram read  addr: 000000000 len: 0ff
# 340 ns top/ram0 ram write addr: 000002800 len: 03f
# 342 ns top/ram1 ram write addr: 000002800 len: 03f
# 343 ns top/ram0 ram read  addr: 000000800 len: 03f
# 345 ns top/ram1 ram read  addr: 000000800 len: 03f
# 414 ns top dma_done detected. 1 1
# 414 ns top start_time: 55 ns end_time: 414 ns
# 414 ns top axi beats (dec): 320
# 414 ns top elapsed time: 359 ns
# 414 ns top beat rate: 1122 ps
# 414 ns top clock period: 1 ns
# 454 ns top finished checking memory contents
```

Before and after HLS we get nearly one beat per clock cycle

AXI4 Fabric Waveforms Before HLS–Test #0 (SystemC)



AXI4 Fabric Waveforms After HLS – Test #0 (Verilog)

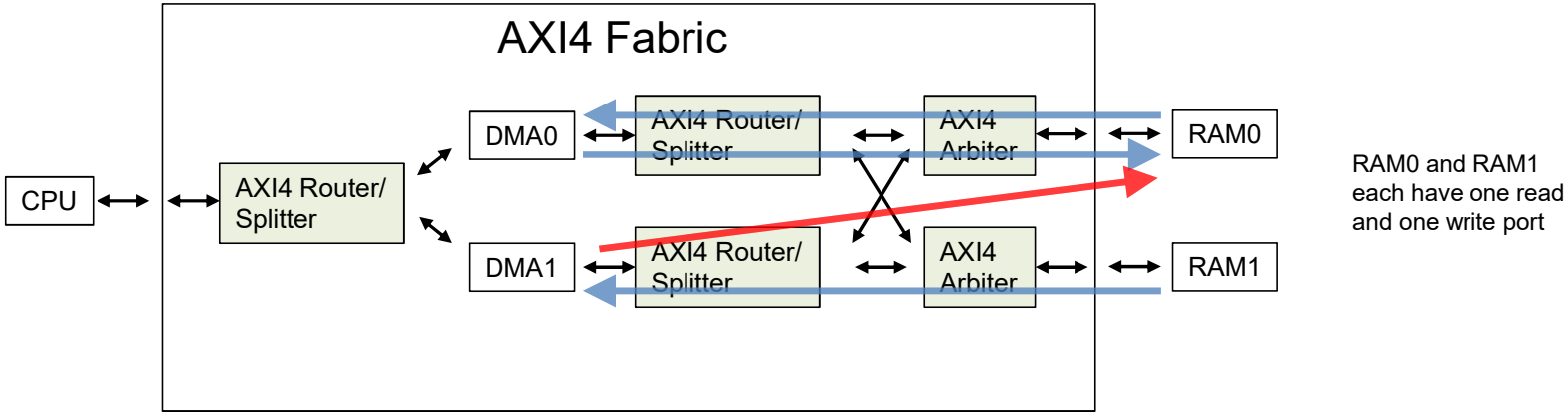


Thoughput in RTL Matches SystemC

accellera SYSTEMS INITIATIVE™

SYSTEMC™ EVOLUTION DAY
OCT 29, 2020 | VIRTUAL WORKSHOP

AXI4 Bus Fabric using MatchLib – Test #1



Test #1: Concurrently,
DMA0 reads/writes 320 beats to RAM0
DMA1 reads 320 beats from RAM1 and writes to RAM0
Note contention on RAM0 writes

AXI4 Bus Fabric Test #1 simulation logs

BEFORE HLS (SystemC simulation)

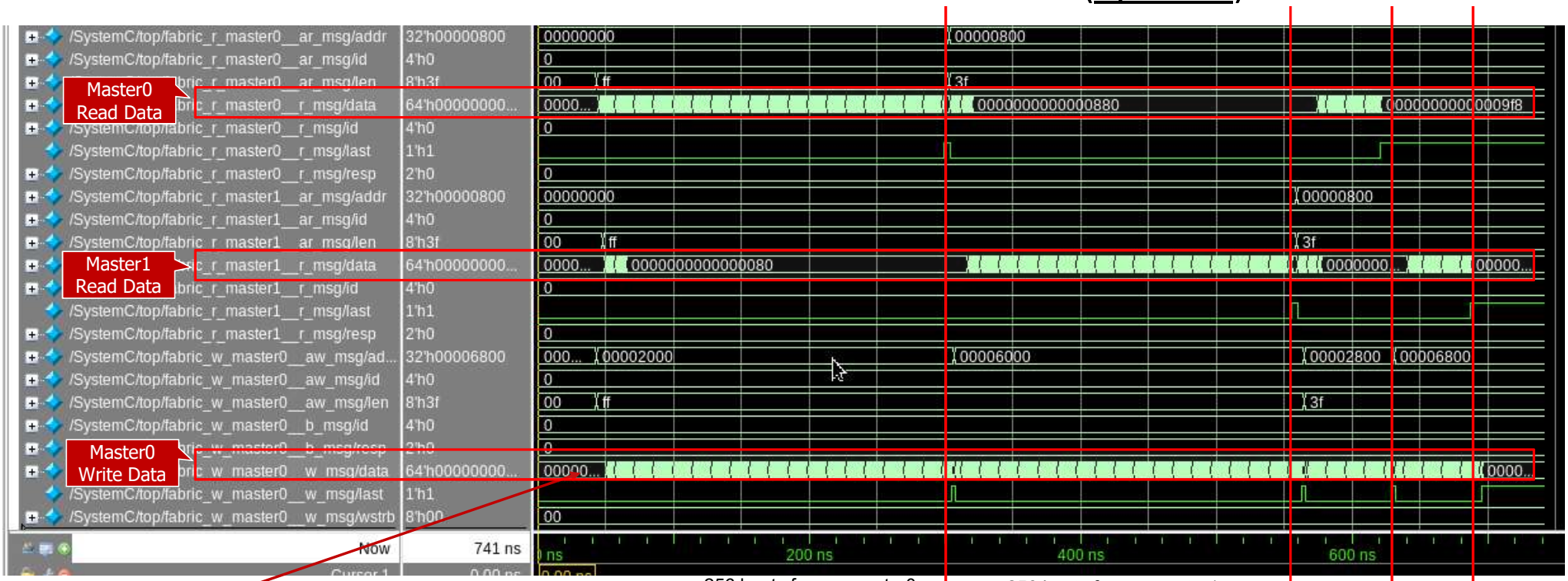
```
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 1
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 len: 03f
308 ns top.ram0 ram write addr: 000006000 len: 0ff
560 ns top.ram1 ram read  addr: 000000800 len: 03f
566 ns top.ram0 ram write addr: 000002800 len: 03f
632 ns top.ram0 ram write addr: 000006800 len: 03f
701 ns top dma_done detected. 1 1
701 ns top start_time: 46 ns end_time: 701 ns
701 ns top axi beats (dec): 320
701 ns top elapsed time: 655 ns
701 ns top beat rate: 2047 ps
701 ns top clock period: 1 ns
741 ns top finished checking memory contents
```

AFTER HLS (Verilog RTL simulation)

```
# 0 s top Stimulus started
# 6 ns top Running FABRIC_TEST # : 1
# 55 ns top/ram0 ram write addr: 000002000 len: 0ff
# 68 ns top/ram0 ram read  addr: 000000000 len: 0ff
# 70 ns top/ram1 ram read  addr: 000000000 len: 0ff
# 335 ns top/ram0 ram write addr: 000006000 len: 0ff
# 343 ns top/ram0 ram read  addr: 000000800 len: 03f
# 598 ns top/ram1 ram read  addr: 000000800 len: 03f
# 598 ns top/ram0 ram write addr: 000002800 len: 03f
# 670 ns top/ram0 ram write addr: 000006800 len: 03f
# 736 ns top dma_done detected. 1 1
# 736 ns top start_time: 55 ns end_time: 736 ns
# 736 ns top axi beats (dec): 320
# 736 ns top elapsed time: 681 ns
# 736 ns top beat rate: 2128 ps
# 736 ns top clock period: 1 ns
# 776 ns top finished checking memory contents
```

Two concurrent writes to RAM0 cause beat rate to be above two clock cycles.

AXI4 Fabric Waveforms Before HLS –Test#1 (SystemC)



256 beats from r_master0

256 beats from r_master1

64 beats from r_master0

64 beats from r_master1

w_master0 fully utilized over 700 ns due to write contention

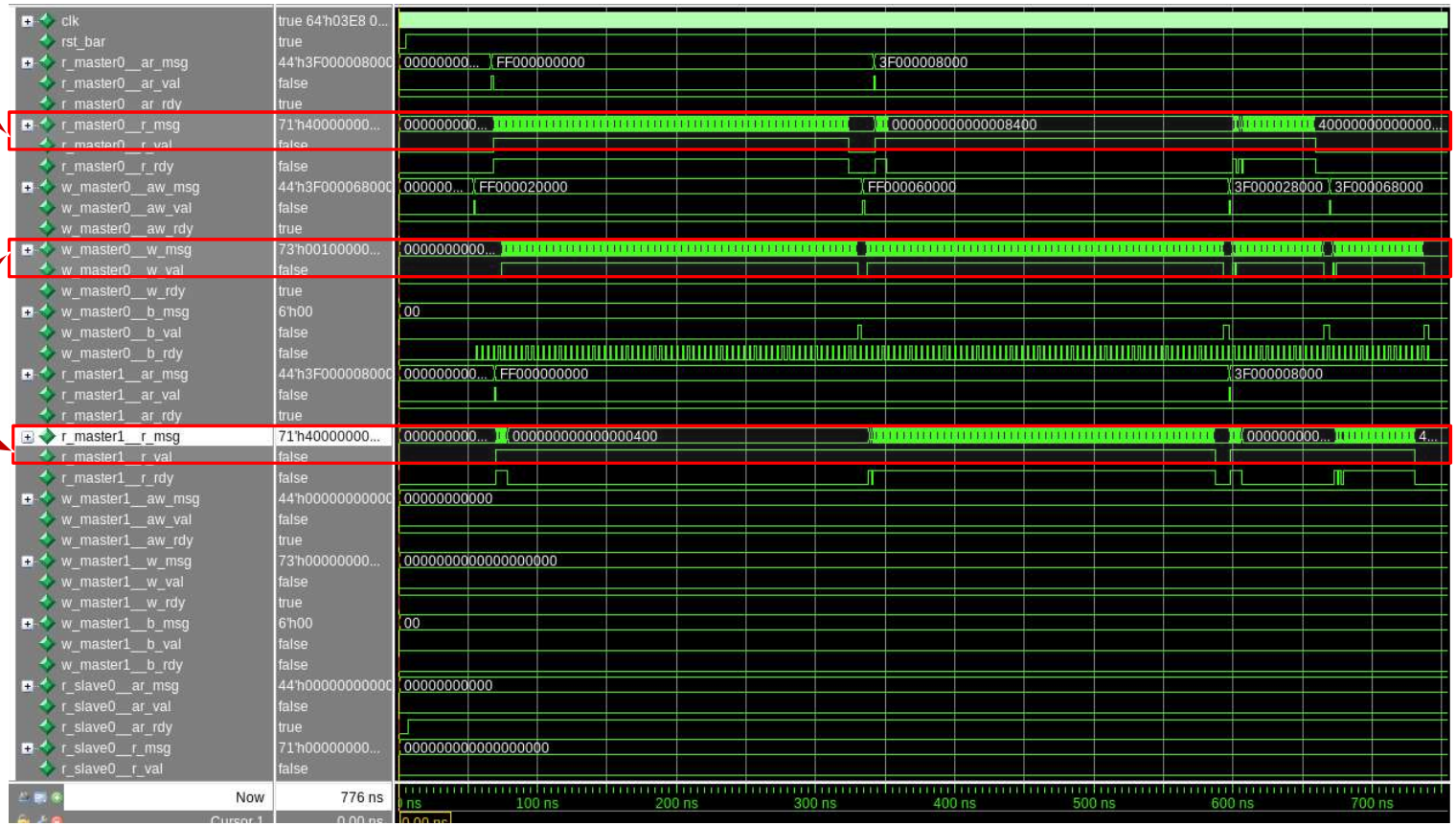
r_master0 and r_master1 underutilized due to write contention

AXI4 Fabric Waveforms After HLS – Test #1 (Verilog)

Master0 Read Data

Master0 Write Data

Master1 Read Data



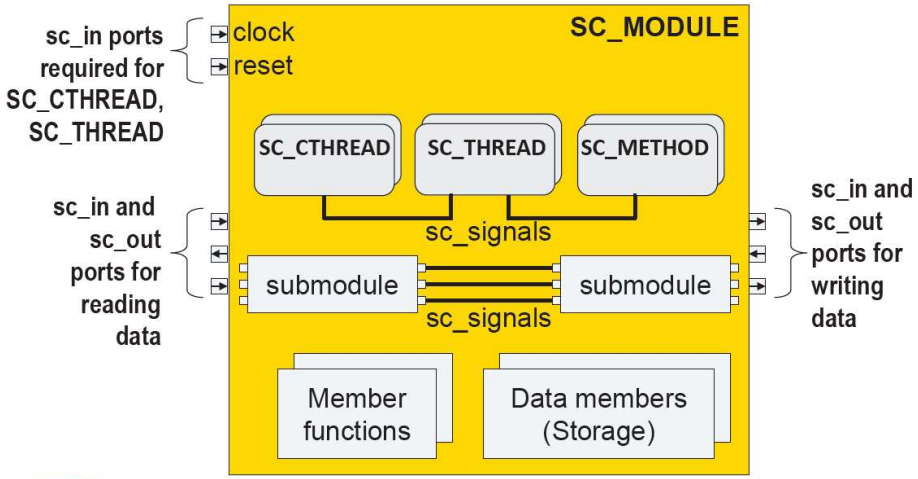
Throughput in RTL Matches SystemC

Recap: MatchLib and HLS Enable Modern D/V Flow

- Designer focuses on chip architecture, functionality, and throughput analysis/verification.
  - HLS adds pipelining, optimizes microarchitecture, provides fully automated flow to placed gates.
- Focus of verification effort moves to C++/SystemC level, enabling much greater efficiency.
- Additional introductory material on MatchLib is publicly available on web:
  - https://www.mentor.com/hls-lp/events/nvidia-design-and-verification-of-a-machine-learning-accelerator-soc-using-an-object-oriented-hls-based-design-flow
  - https://www.mentor.com/hls-lp/multimedia/early-axi4-soc-performance-verification-using-nvidia-matchlib-and-catapult-systemc-hls
  - https://uploads-ssl.webflow.com/5a749b2fa5fde0000189ffc0/5d3b1bef8474c4537c1d494b_Khailany_Brucek_CRAFT_Final.pdf
  - https://www.youtube.com/watch?v=n8_G-CaSSPU

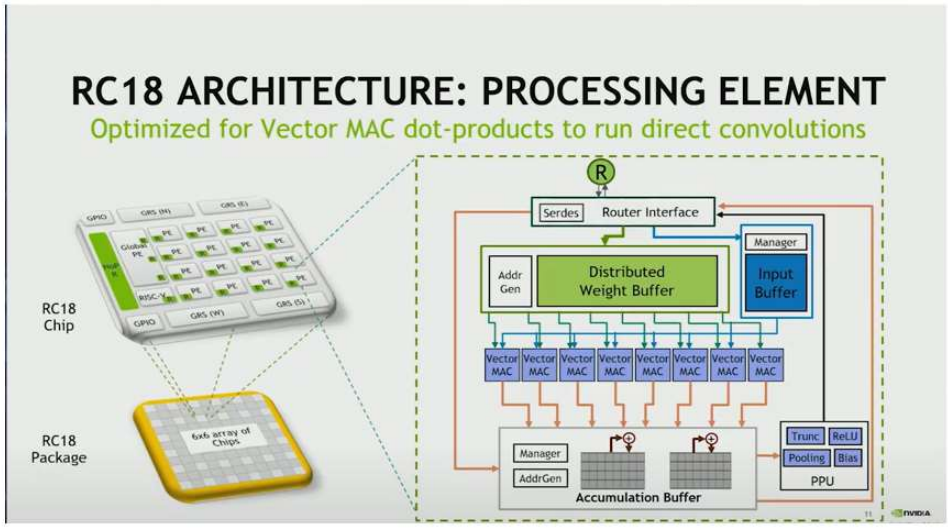
MatchLib Relationship to SystemC Standards

- SystemC Synthesizeable Subset Standard focuses on what's in diagram below
  - Modules, Ports, Processes, clocks, resets, signal IO, datatypes



Real World Design Example

- Consider a real world example:
  - May have 100s or 1000s of SystemC processes
  - May generate millions of gates
  - May have very complex interconnect
  - Biggest risks may be in interconnect



In Real World - Interconnect Modeling is Key

- In pre-HLS model, need:
  - Throughput accuracy
  - Message latency and capacity back annotation
  - Random stall injection
  - Waveform generation
  - Transaction logging and debugging
  - Accurate and also fast TLM modes
  - Integration with SV UVM

Proposed SystemC HLS Standards Layers

SystemC MatchLib IP Blocks Standard

- Parameterized AXI4 Fabric Components
- Banked Memories
- Crossbar, Reorder Buffer, Cache
- Parameterized NOC components

SystemC Synthesizable Connections Standard

- Throughput accurate modeling
- Message latency and capacity back annotation
- Random stall injection
- Waveform generation
- Transaction logging and debugging
- Accurate and also fast TLM modes

SystemC Synthesizable Subset Standard

SystemC Language Standard