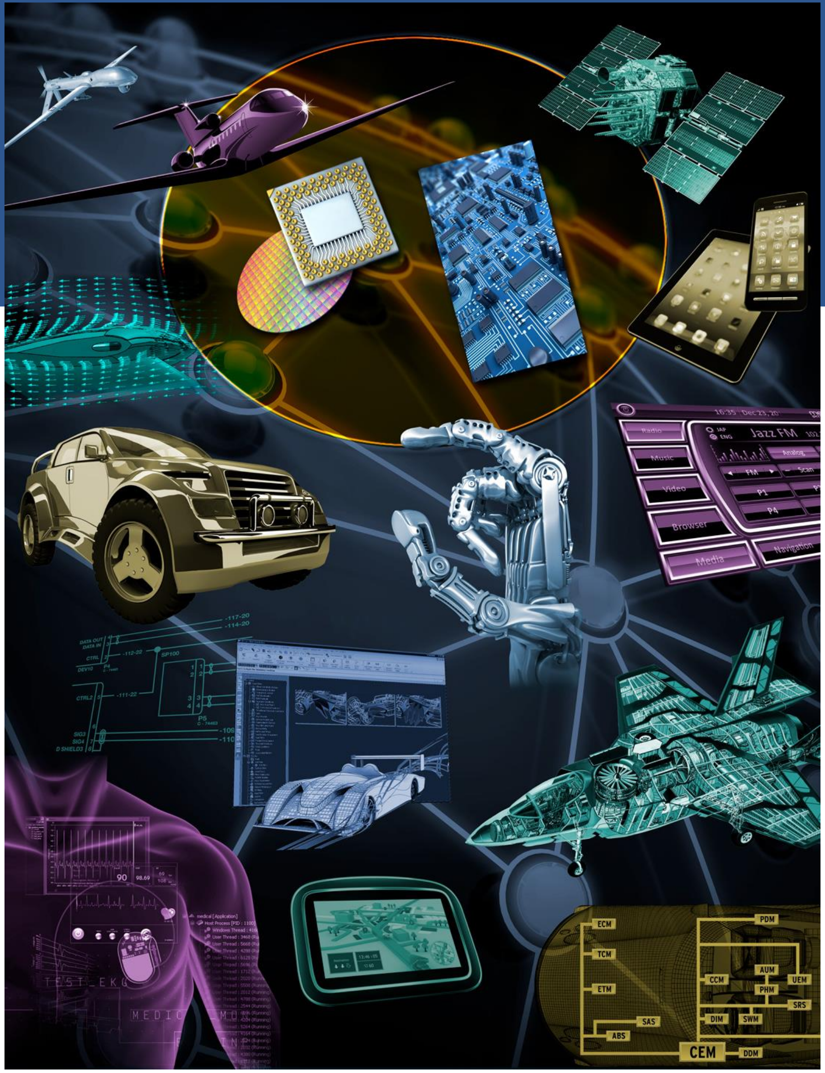


Early SOC Performance Verification Using SystemC with NVIDIA MatchLib and Catapult HLS

Stuart Swan

HLS IP/Platform Architect

DAC: June 2019

Mentor®
A Siemens Business

Introduction

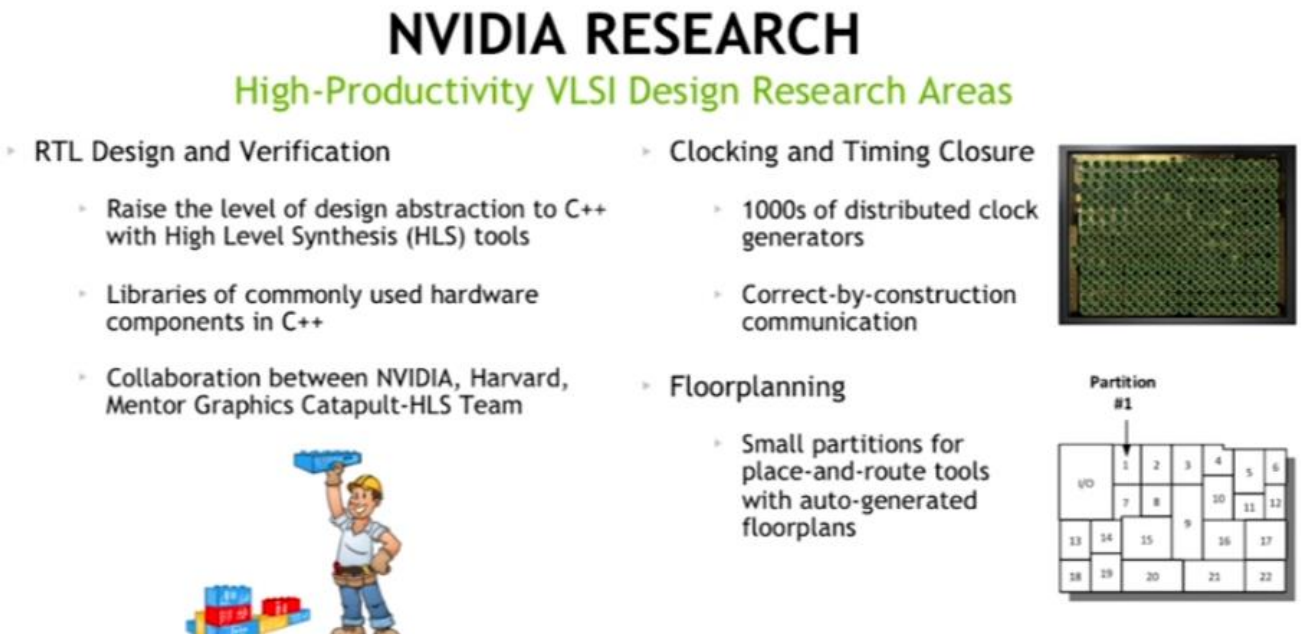
- My background
- NVIDIA Matchlib introduction
- Why use it?
- Walk through a simple example and look at HLS results
- Walk through a larger example
- Q & A

What is NVIDIA Matchlib?

- Good 30 minute intro video here:
  - Google: nvidia machine learning mentor events

DESIGN AND VERIFICATION OF A MACHINE LEARNING ACCELERATOR SOC USING AN OBJECT-ORIENTED HLS-BASED DESIGN FLOW

Brucek Khailany and Rangharajan Venkatesan
NVIDIA


NVIDIA RESEARCH

High-Productivity VLSI Design Research Areas

- RTL Design and Verification
  - Raise the level of design abstraction to C++ with High Level Synthesis (HLS) tools
  - Libraries of commonly used hardware components in C++
  - Collaboration between NVIDIA, Harvard, Mentor Graphics Catapult-HLS Team
- Clocking and Timing Closure
  - 1000s of distributed clock generators
  - Correct-by-construction communication
- Floorplanning
  - Small partitions for place-and-route tools with auto-generated floorplans

Mentor
A Siemens Business

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
- 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

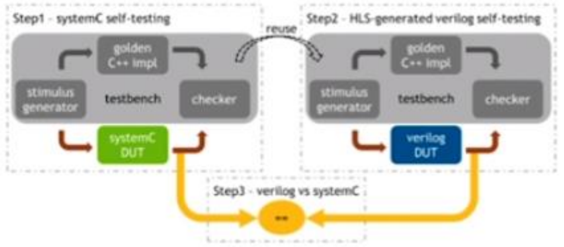
Matchlib SystemC Model Characteristics

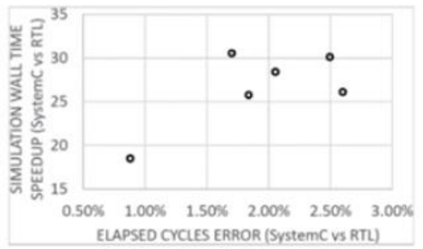
- Small
  - Typically 1/10 or less than the size of comparable RTL models
- Fast
  - Simulates ~30 times faster than RTL models in timing accurate mode
  - Simulates ~300 times faster than RTL models in blocking TLM mode
- Accurate
  - Not exactly RTL cycle accurate, but pretty close
  - Concurrent transactions 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

NVIDIA Matchlib and Catapult HLS Results

RC17 SYSTEMC-BASED VERIFICATION

Functional and Performance Verification on SystemC models


Step1 - systemC self-testing
reuse
Step2 - HLS-generated verilog self-testing
golden C++ impl
stimulus generator
testbench
checker
systemC DUT
verilog DUT
Step3 - verilog vs systemC
==



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

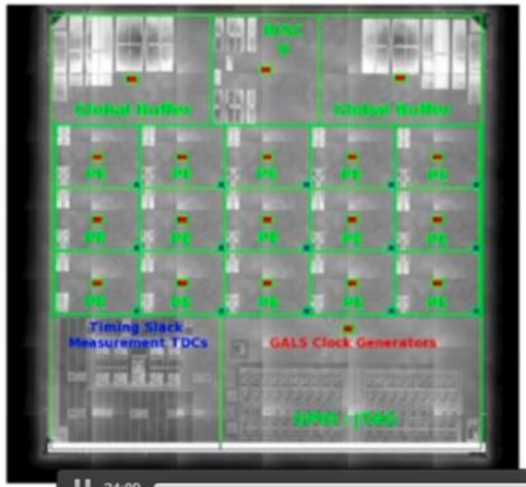
PERFORMANCE VERIFICATION

- 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

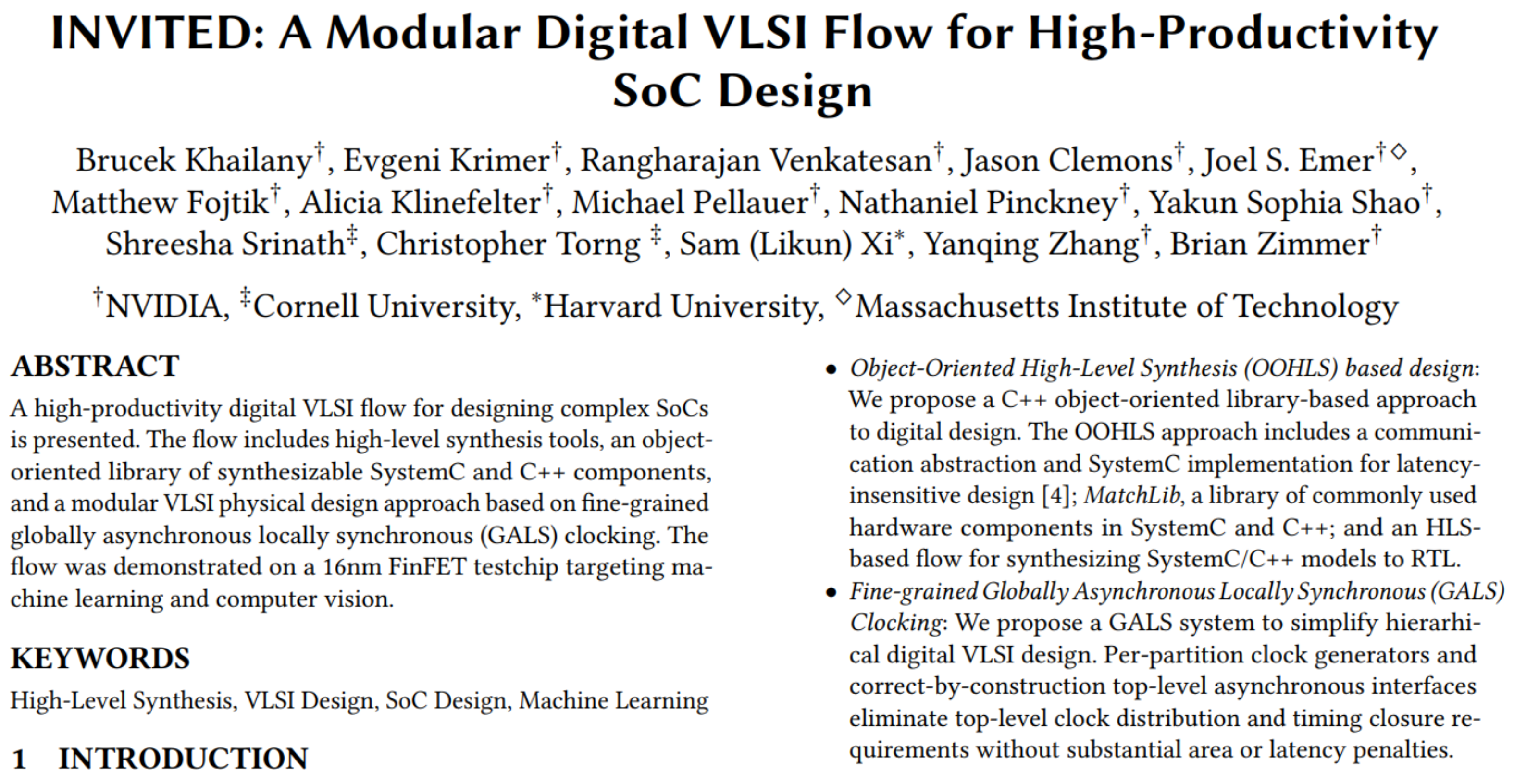


	RC17 Stats
Die Size	4 mm^2
Partitions	19 (5 unique)
Frequency range	510 MHz - 1.96 GHz
Voltage range	0.55-1.2 Volts
Performance (16b GMACS)	61.2-235.2
Max GMACS/W	192.1
Programmability	ML workloads (NN inference, K-means)

NVIDIA

NVIDIA Matchlib DAC 2018 Paper

- Google: dac 2018 nvidia modular digital



INVITED: A Modular Digital VLSI Flow for High-Productivity SoC Design

Brucek Khailany[†], Evgeni Krimer[†], Rangharajan Venkatesan[†], Jason Clemons[†], Joel S. Emer[†◇], Matthew Fojtik[†], Alicia Klinefelter[†], Michael Pellauer[†], Nathaniel Pinckney[†], Yakun Sophia Shao[†], Shreesha Srinath[‡], Christopher Torng [‡], Sam (Likun) Xi[*], Yanqing Zhang[†], Brian Zimmer[†]

[†]NVIDIA, [‡]Cornell University, [*]Harvard University, [◇]Massachusetts Institute of Technology

ABSTRACT

A high-productivity digital VLSI flow for designing complex SoCs is presented. The flow includes high-level synthesis tools, an object-oriented library of synthesizable SystemC and C++ components, and a modular VLSI physical design approach based on fine-grained globally asynchronous locally synchronous (GALS) clocking. The flow was demonstrated on a 16nm FinFET testchip targeting machine learning and computer vision.

KEYWORDS

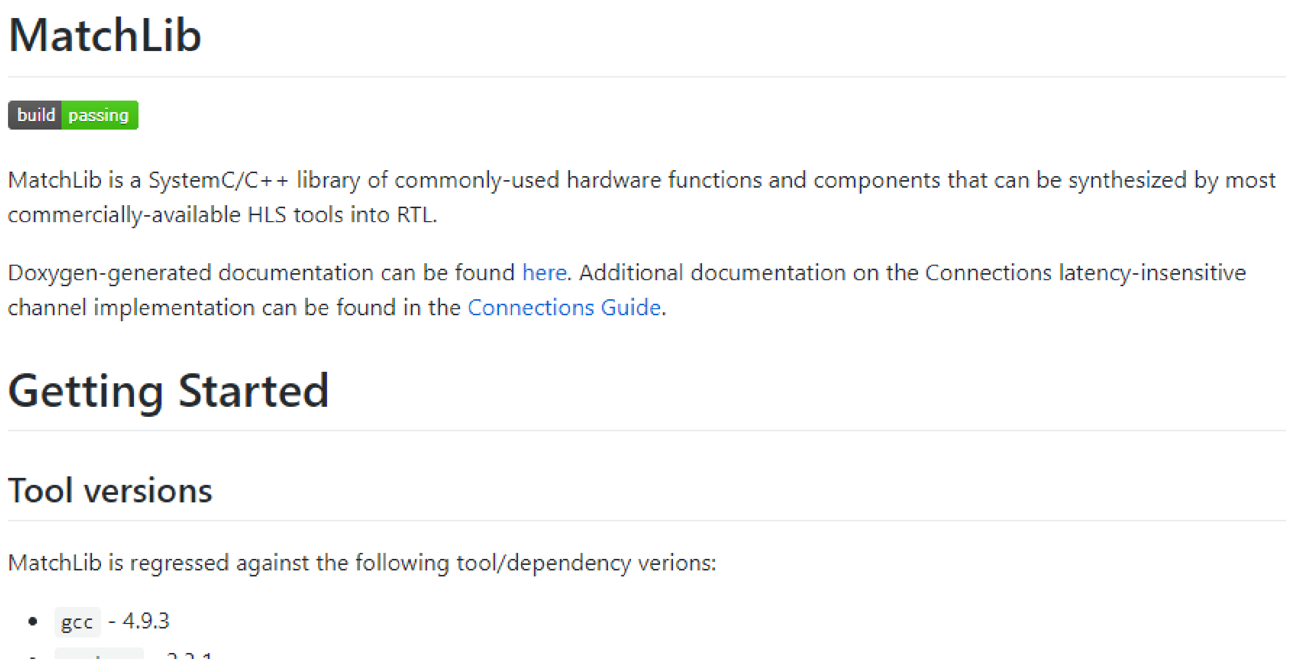
High-Level Synthesis, VLSI Design, SoC Design, Machine Learning

1 INTRODUCTION

- *Object-Oriented High-Level Synthesis (OOHLS) based design*: We propose a C++ object-oriented library-based approach to digital design. The OOHLS approach includes a communication abstraction and SystemC implementation for latency-insensitive design [4]; *MatchLib*, a library of commonly used hardware components in SystemC and C++; and an HLS-based flow for synthesizing SystemC/C++ models to RTL.
- *Fine-grained Globally Asynchronous Locally Synchronous (GALS) Clocking*: We propose a GALS system to simplify hierarhical digital VLSI design. Per-partition clock generators and correct-by-construction top-level asynchronous interfaces eliminate top-level clock distribution and timing closure requirements without substantial area or latency penalties.

NVIDIA Matchlib is Open Source on Github

- Google: nvidia matchlib github



MatchLib

build passing

MatchLib is a SystemC/C++ library of commonly-used hardware functions and components that can be synthesized by most commercially-available HLS tools into RTL.

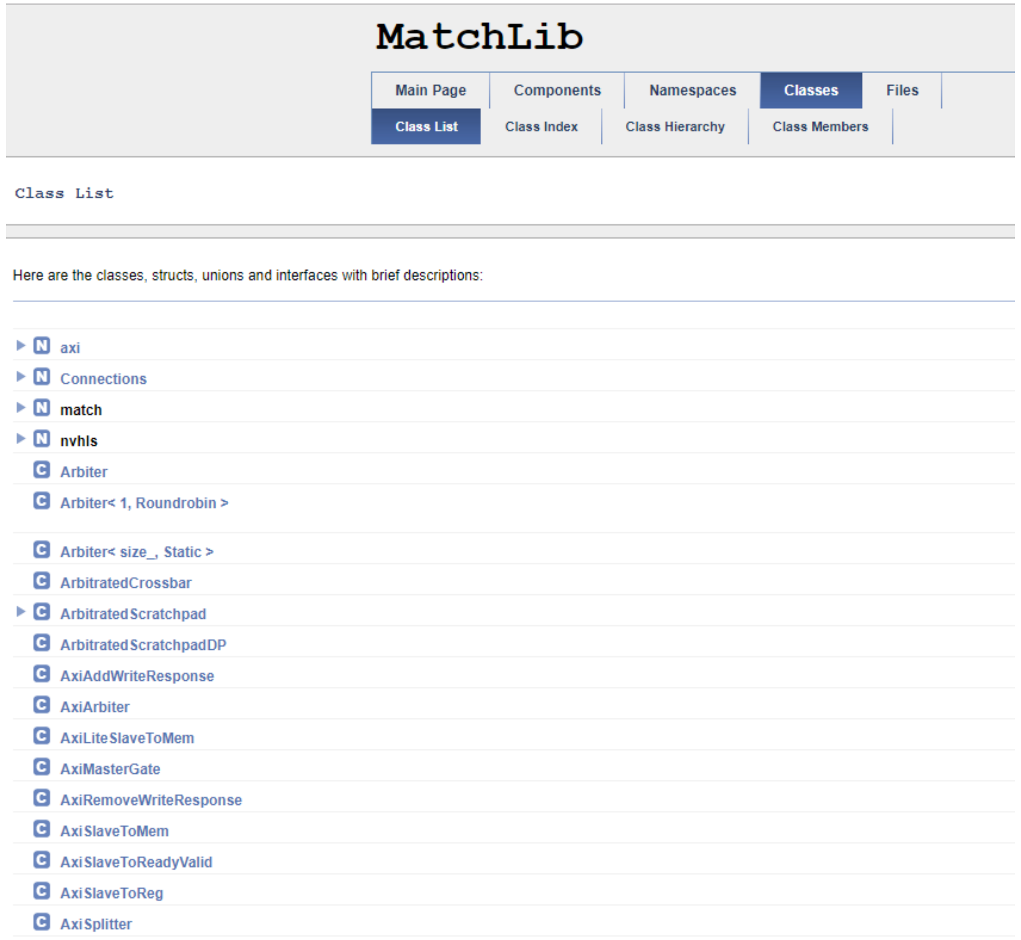
Doxygen-generated documentation can be found here. Additional documentation on the Connections latency-insensitive channel implementation can be found in the Connections Guide.

Getting Started

Tool versions

MatchLib is regressed against the following tool/dependency verions:

- gcc - 4.9.3



Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

- axi
- Connections
- match
- nvhls
- Arbiter
- Arbiter< 1, Roundrobin >
- Arbiter< size_, Static >
- ArbitratedCrossbar
- ArbitratedScratchpad
- ArbitratedScratchpadDP
- AxiAddWriteResponse
- AxiArbiter
- AxiLiteSlaveToMem
- AxiMasterGate
- AxiRemoveWriteResponse
- AxiSlaveToMem
- AxiSlaveToReadyValid
- AxiSlaveToReg
- AxiSplitter

30 years ago what enabled change from gate-level to RTL?

1. Verilog/VHDL standardized approach to RTL capture
2. Designers could focus on cycle level timing
   - RTL synthesis, timing, and P&R tools automatically handled gate level timing concerns.
3. RTL IP (Designware, ARM, ...) enabled more reuse
4. RTL flow supported any type of digital design.
5. RTL flow supported synthesis of full chip.
6. Focus of verification effort moved to RTL level, enabling much greater efficiency.

Goal of Matchlib & HLS is to enable switch from RTL to C++

1. Matchlib and C++/SystemC provide a standardized approach to design capture
2. Designer focuses on chip architecture, functionality, and throughput analysis/verification.
   - HLS adds pipelining, optimizes microarchitecture, provides fully automated flow to placed gates.
3. HLS IP enables more reuse
   - Fixed point types, AXI interconnect, NOC, banked memories, algorithm IP…
4. Matchlib + HLS flow supports any type of digital design
5. Matchlib + HLS flow supports HLS synthesis of full chip.
6. Focus of verification effort moves to C++/SystemC level, enabling much greater efficiency.

(But, teams can also adopt Matchlib + HLS flow in steps,
they do not need to adopt the entire flow all at once.)

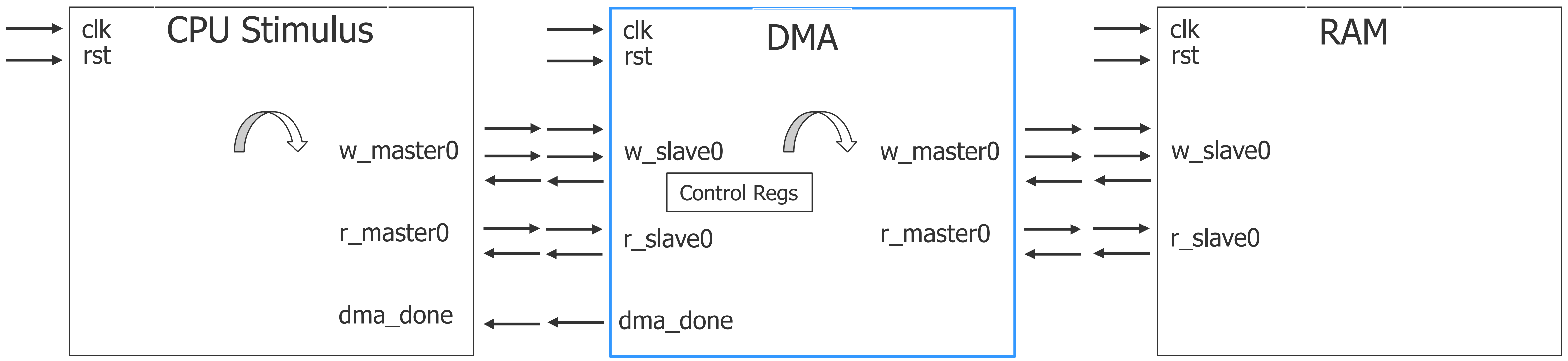
Complexity / Risk in Modern Designs has Shifted…

- 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



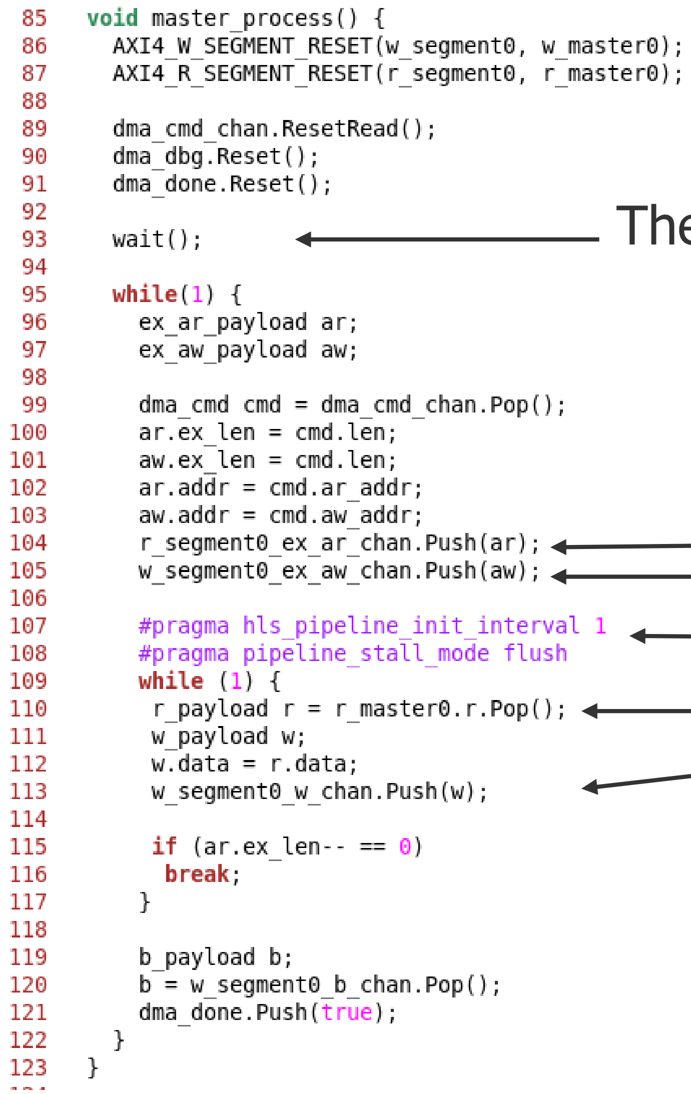
```
/**
 *  * \brief dma module
*/
#pragma hls_design top
class dma : public sc_module, public local_axi {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  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);
```

□ = 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



```
void master_process() {
  AXI4_W_SEGMENT_RESET(w_segment0, w_master0);
  AXI4_R_SEGMENT_RESET(r_segment0, r_master0);

  dma_cmd_chan.ResetRead();
  dma_dbg.Reset();
  dma_done.Reset();

  wait();

  while(1) {
    ex_ar_payload ar;
    ex_aw_payload aw;

    dma_cmd cmd = dma_cmd_chan.Pop();
    ar.ex_len = cmd.len;
    aw.ex_len = cmd.len;
    ar.addr = cmd.ar_addr;
    aw.addr = cmd.aw_addr;
    r_segment0_ex_ar_chan.Push(ar);
    w_segment0_ex_aw_chan.Push(aw);

    #pragma hls_pipeline_init_interval 1
    #pragma pipeline_stall_mode flush
    while (1) {
     r_payload r = r_master0.r.Pop();
     w_payload w;
     w.data = r.data;
     w_segment0_w_chan.Push(w);

     if (ar.ex_len-- == 0)
      break;
    }

    b_payload b;
    b = w_segment0_b_chan.Pop();
    dma_done.Push(true);
  }
}
```

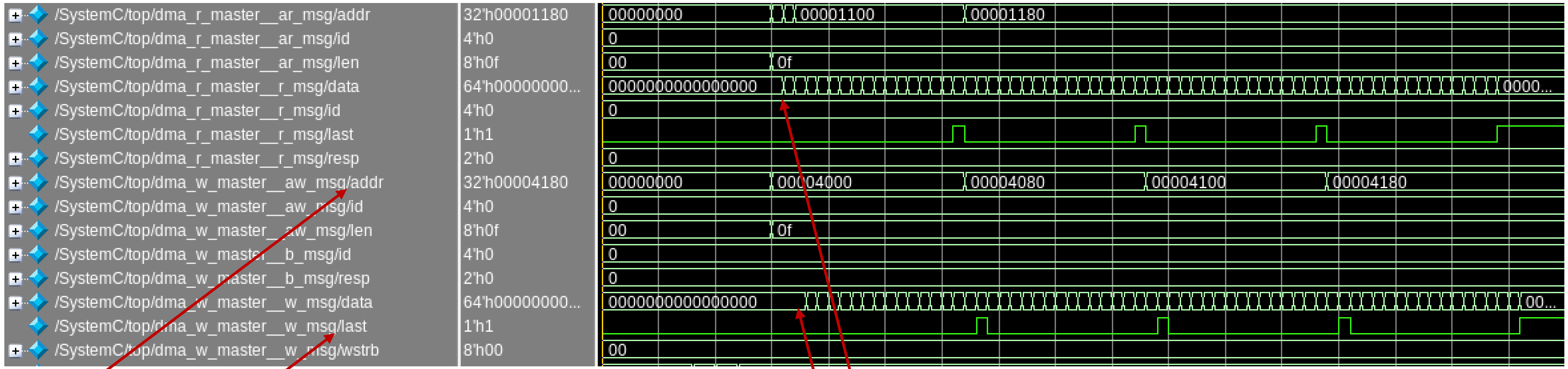
The only clock/wait is for reset state

This IO is in parallel

Main compute loop gets pipelined in HLS

This IO is in parallel

AXI4 DMA Waveforms Before HLS (SystemC simulation)



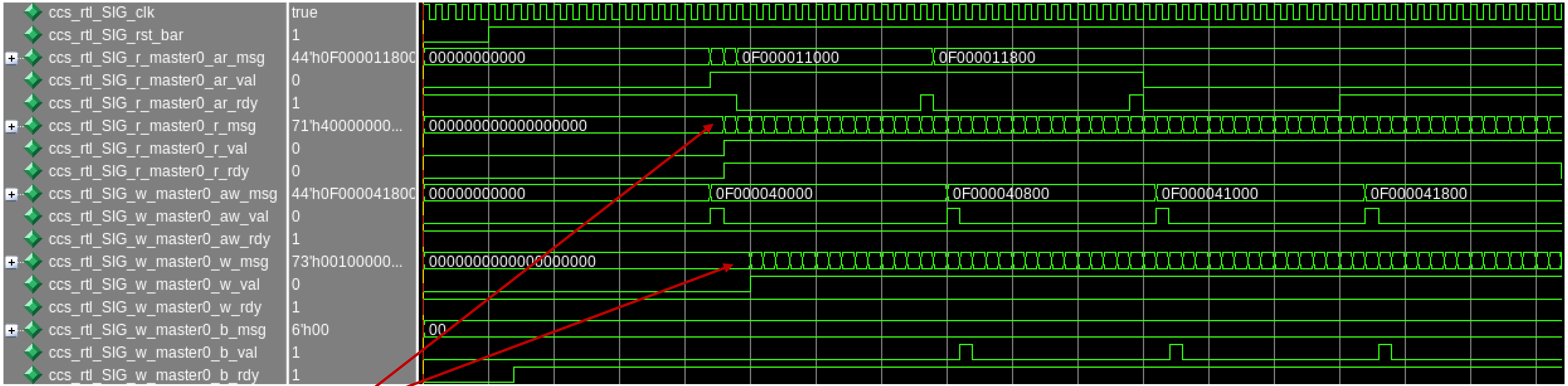
/SystemC/top/dma_r_master__ar_msg/addr	32'h00001180
/SystemC/top/dma_r_master__ar_msg/id	4'h0
/SystemC/top/dma_r_master__ar_msg/len	8'h0f
/SystemC/top/dma_r_master__r_msg/data	64'h00000000...
/SystemC/top/dma_r_master__r_msg/id	4'h0
/SystemC/top/dma_r_master__r_msg/last	1'h1
/SystemC/top/dma_r_master__r_msg/resp	2'h0
/SystemC/top/dma_w_master__aw_msg/addr	32'h00004180
/SystemC/top/dma_w_master__aw_msg/id	4'h0
/SystemC/top/dma_w_master__aw_msg/len	8'h0f
/SystemC/top/dma_w_master__b_msg/id	4'h0
/SystemC/top/dma_w_master__b_msg/resp	2'h0
/SystemC/top/dma_w_master__w_msg/data	64'h00000000...
/SystemC/top/dma_w_master__w_msg/last	1'h1
/SystemC/top/dma_w_master__w_msg/wstrb	8'h00

Automatic AXI4 last bit generation

Automatic AXI4 burst address segmentation

Read and write burst streams are concurrent.
R/W bus utilization is 100%
(1 read and 1 write beat per clock cycle)

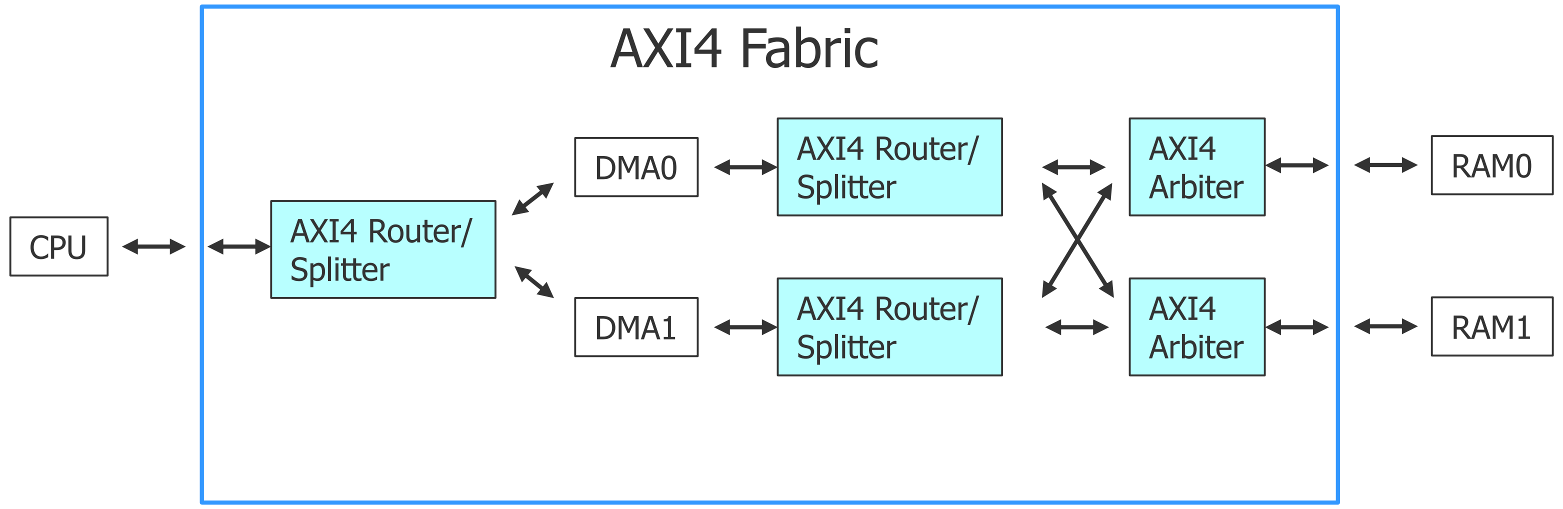
AXI4 DMA Waveforms After Catapult HLS (Verilog Sim)



ccs_rtl_SIG_clk	true
ccs_rtl_SIG_rst_bar	1
ccs_rtl_SIG_r_master0_ar_msg	44'h0F000011800
ccs_rtl_SIG_r_master0_ar_val	0
ccs_rtl_SIG_r_master0_ar_rdy	1
ccs_rtl_SIG_r_master0_r_msg	71'h40000000...
ccs_rtl_SIG_r_master0_r_val	0
ccs_rtl_SIG_r_master0_r_rdy	0
ccs_rtl_SIG_w_master0_aw_msg	44'h0F000041800
ccs_rtl_SIG_w_master0_aw_val	0
ccs_rtl_SIG_w_master0_aw_rdy	1
ccs_rtl_SIG_w_master0_w_msg	73'h00100000...
ccs_rtl_SIG_w_master0_w_val	0
ccs_rtl_SIG_w_master0_w_rdy	1
ccs_rtl_SIG_w_master0_b_msg	6'h00
ccs_rtl_SIG_w_master0_b_val	1
ccs_rtl_SIG_w_master0_b_rdy	1

RTL waveforms are almost the same as SystemC waveforms:
- Throughput is same
- Bus utilization is the same
- HLS may have added pipeline stages (under user control)
- HLS may have increased latency (under user control)

Larger Example: AXI4 Bus Fabric using Matchlib



Address Map

0x00000
0x7FFFF
0x80000
0x8FFFF

```
/**
 *  * \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);
```

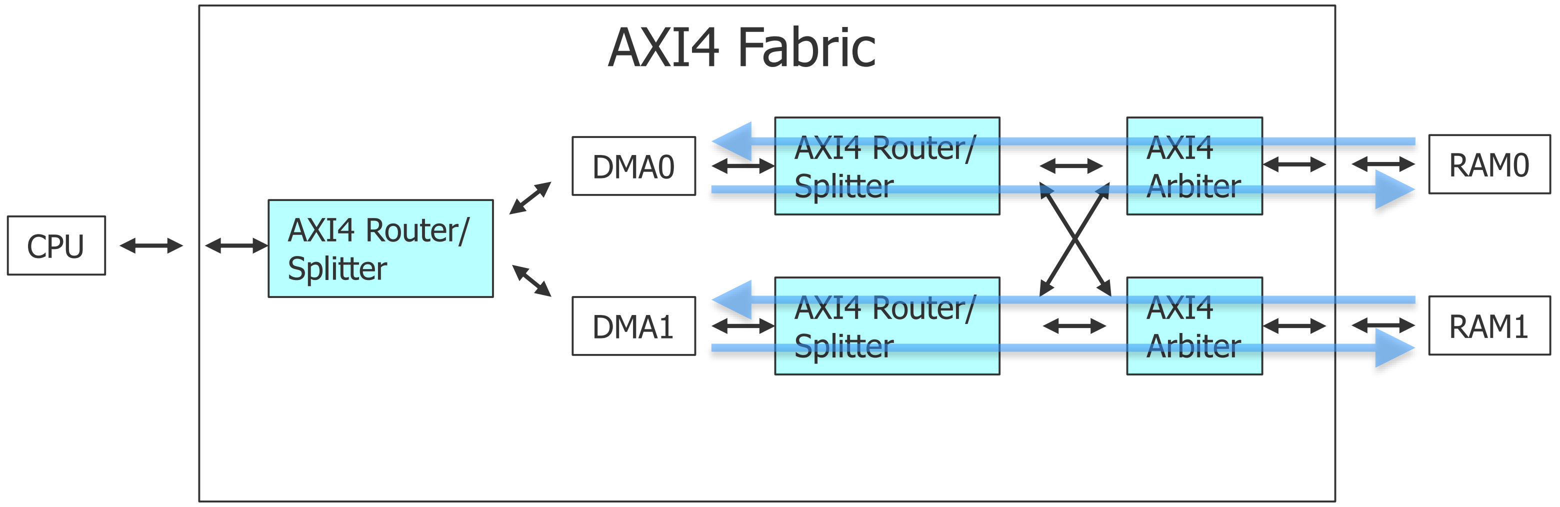
Blue boxes are Matchlib Components

☐ = top level of design

DMA0 and DMA1 are two instances of DMA shown earlier

Mentor® A Siemens Business

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

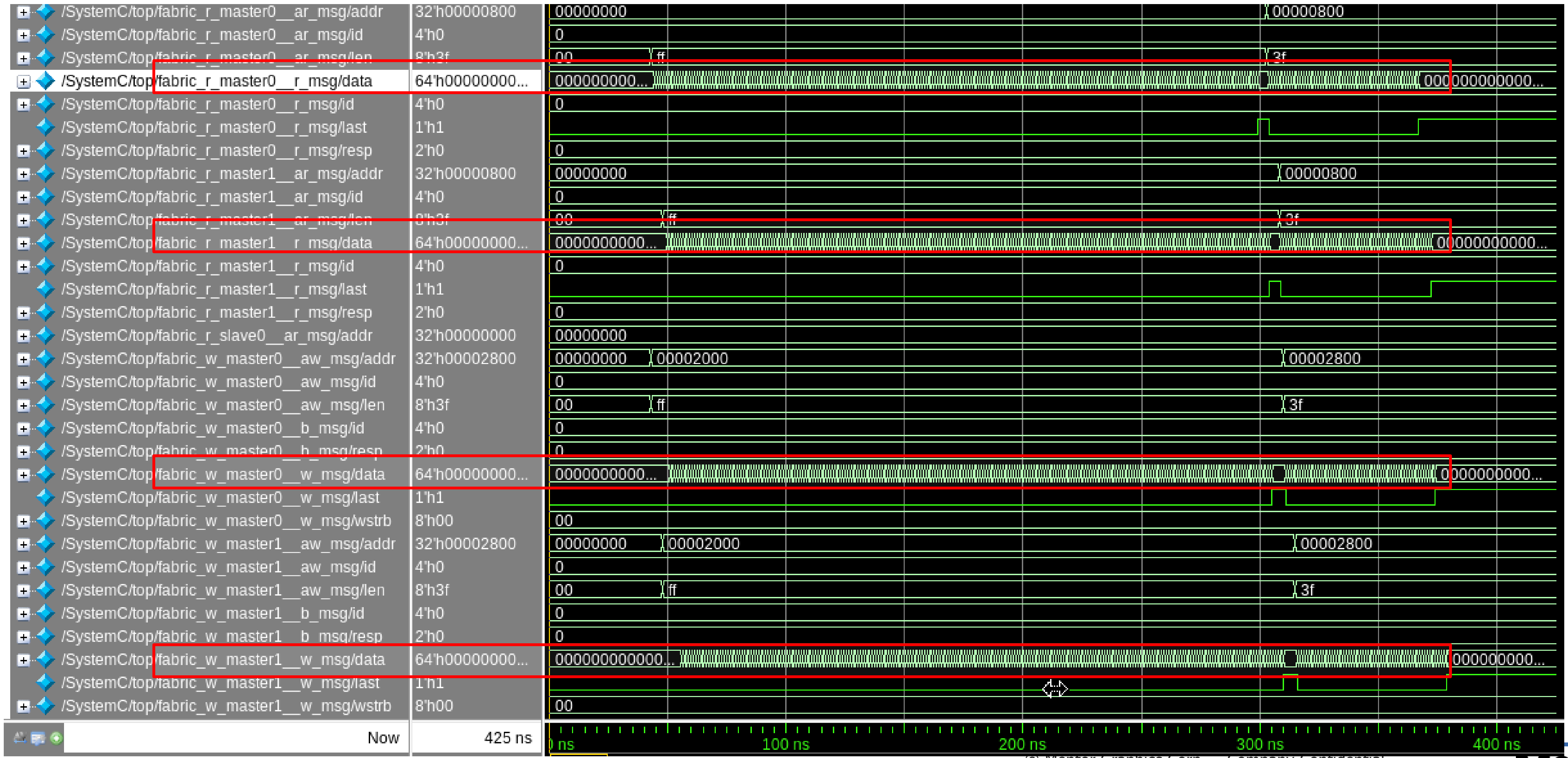
```
BEFORE HLS (SystemC simulation)
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 0
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns 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
(this converges closer to one as length of burst increases)

Mentor
A Siemens Business

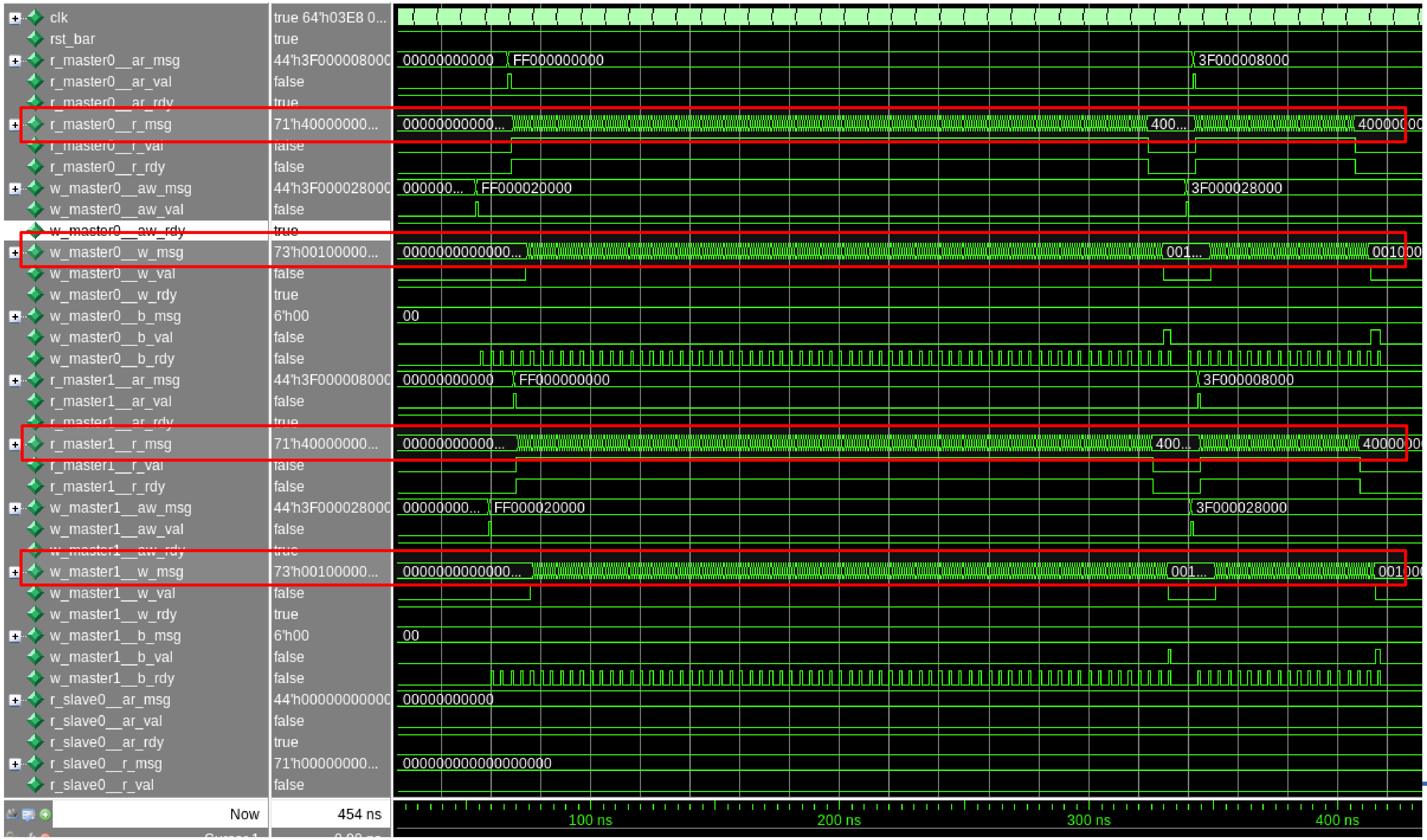
AXI4 Fabric Waveforms Before Catapult HLS–Test #0 (SystemC)



/SystemC/top/fabric_r_master0__ar_msg/addr	32'h00000800
/SystemC/top/fabric_r_master0__ar_msg/id	4'h0
/SystemC/top/fabric_r_master0__ar_msg/len	8'h3f
/SystemC/top/fabric_r_master0__r_msg/data	64'h00000000...
/SystemC/top/fabric_r_master0__r_msg/id	4'h0
/SystemC/top/fabric_r_master0__r_msg/last	1'h1
/SystemC/top/fabric_r_master0__r_msg/resp	2'h0
/SystemC/top/fabric_r_master1__ar_msg/addr	32'h00000800
/SystemC/top/fabric_r_master1__ar_msg/id	4'h0
/SystemC/top/fabric_r_master1__ar_msg/len	8'h3f
/SystemC/top/fabric_r_master1__r_msg/data	64'h00000000...
/SystemC/top/fabric_r_master1__r_msg/id	4'h0
/SystemC/top/fabric_r_master1__r_msg/last	1'h1
/SystemC/top/fabric_r_master1__r_msg/resp	2'h0
/SystemC/top/fabric_r_slave0__ar_msg/addr	32'h00000000
/SystemC/top/fabric_w_master0__aw_msg/addr	32'h00002800
/SystemC/top/fabric_w_master0__aw_msg/id	4'h0
/SystemC/top/fabric_w_master0__aw_msg/len	8'h3f
/SystemC/top/fabric_w_master0__b_msg/id	4'h0
/SystemC/top/fabric_w_master0__b_msg/resp	2'h0
/SystemC/top/fabric_w_master0__w_msg/data	64'h00000000...
/SystemC/top/fabric_w_master0__w_msg/last	1'h1
/SystemC/top/fabric_w_master0__w_msg/wstrb	8'h00
/SystemC/top/fabric_w_master1__aw_msg/addr	32'h00002800
/SystemC/top/fabric_w_master1__aw_msg/id	4'h0
/SystemC/top/fabric_w_master1__aw_msg/len	8'h3f
/SystemC/top/fabric_w_master1__b_msg/id	4'h0
/SystemC/top/fabric_w_master1__b_msg/resp	2'h0
/SystemC/top/fabric_w_master1__w_msg/data	64'h00000000...
/SystemC/top/fabric_w_master1__w_msg/last	1'h1
/SystemC/top/fabric_w_master1__w_msg/wstrb	8'h00
Now	425 ns

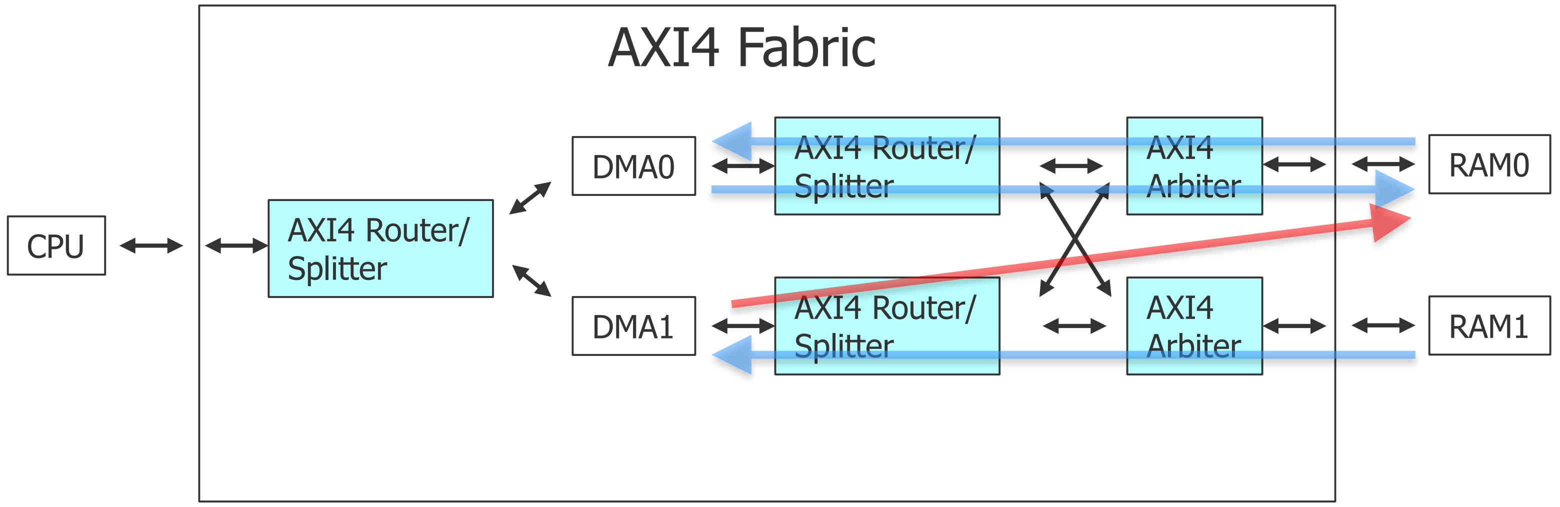
AXI4 Fabric Waveforms After Catapult HLS – Test #0 (Verilog)

Throughput
In RTL
Matches
SystemC



clk	true 64'h03E8 0...
rst_bar	true
r_master0__ar_msg	44'h3F000008000
r_master0__ar_val	false
r_master0__ar_rdy	true
r_master0__r_msg	71'h40000000...
r_master0__r_val	false
r_master0__r_rdy	false
w_master0__aw_msg	44'h3F000028000
w_master0__aw_val	false
w_master0__aw_rdy	true
w_master0__w_msg	73'h00100000...
w_master0__w_val	false
w_master0__w_rdy	true
w_master0__b_msg	6'h00
w_master0__b_val	false
w_master0__b_rdy	false
r_master1__ar_msg	44'h3F000008000
r_master1__ar_val	false
r_master1__ar_rdy	true
r_master1__r_msg	71'h40000000...
r_master1__r_val	false
r_master1__r_rdy	false
w_master1__aw_msg	44'h3F000028000
w_master1__aw_val	false
w_master1__aw_rdy	true
w_master1__w_msg	73'h00100000...
w_master1__w_val	false
w_master1__w_rdy	true
w_master1__b_msg	6'h00
w_master1__b_val	false
w_master1__b_rdy	false
r_slave0__ar_msg	44'h00000000000
r_slave0__ar_val	false
r_slave0__ar_rdy	true
r_slave0__r_msg	71'h00000000...
r_slave0__r_val	false
Now	454 ns

AXI4 Bus Fabric using Matchlib – Test #1



RAM0 and RAM1 each have one read and one write port

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)

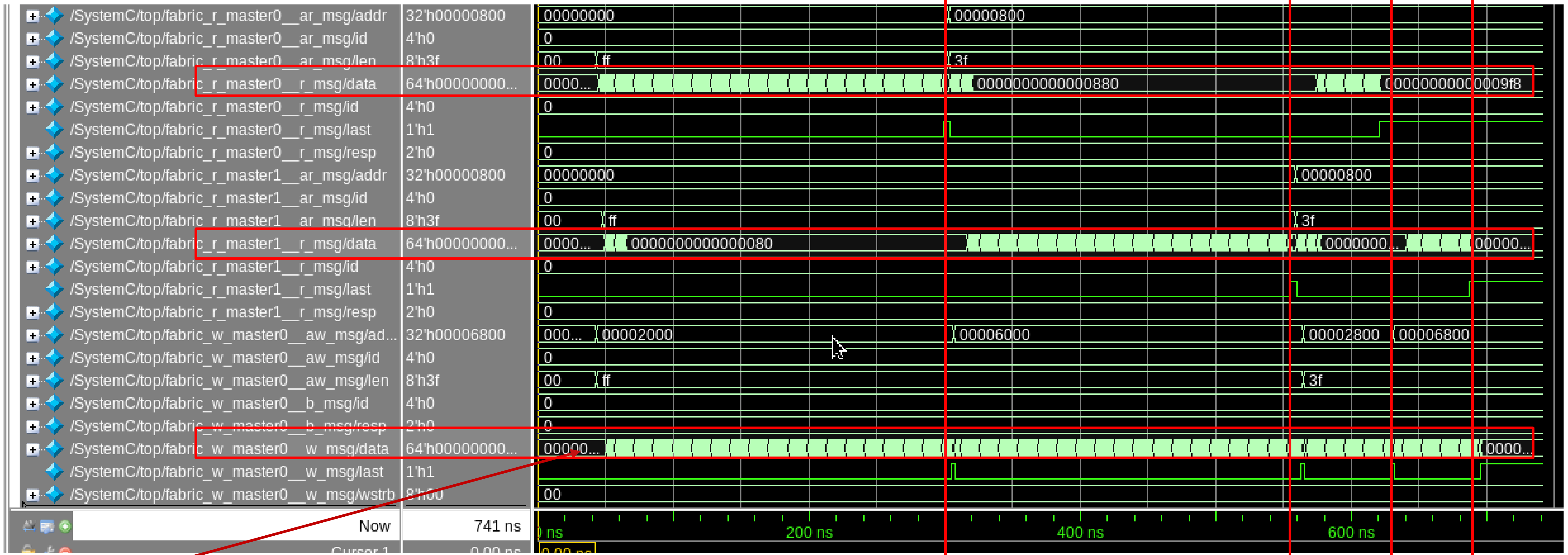
```
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 1
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 len: 03f
308 ns top.ram0 ram write addr: 000006000 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 burst writes to RAM0 cause per-DMA burst beat rate to be above two clock cycles

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



```
/SystemC/top/fabric_r_master0__ar_msg/addr     32'h00000800
/SystemC/top/fabric_r_master0__ar_msg/id       4'h0
/SystemC/top/fabric_r_master0__ar_msg/len      8'h3f
/SystemC/top/fabric_r_master0__r_msg/data      64'h00000000...
/SystemC/top/fabric_r_master0__r_msg/id        4'h0
/SystemC/top/fabric_r_master0__r_msg/last      1'h1
/SystemC/top/fabric_r_master0__r_msg/resp      2'h0
/SystemC/top/fabric_r_master1__ar_msg/addr     32'h00000800
/SystemC/top/fabric_r_master1__ar_msg/id       4'h0
/SystemC/top/fabric_r_master1__ar_msg/len      8'h3f
/SystemC/top/fabric_r_master1__r_msg/data      64'h00000000...
/SystemC/top/fabric_r_master1__r_msg/id        4'h0
/SystemC/top/fabric_r_master1__r_msg/last      1'h1
/SystemC/top/fabric_r_master1__r_msg/resp      2'h0
/SystemC/top/fabric_w_master0__aw_msg/ad...    32'h00006800
/SystemC/top/fabric_w_master0__aw_msg/id       4'h0
/SystemC/top/fabric_w_master0__aw_msg/len      8'h3f
/SystemC/top/fabric_w_master0__b_msg/id        4'h0
/SystemC/top/fabric_w_master0__b_msg/resp      2'h0
/SystemC/top/fabric_w_master0__w_msg/data      64'h00000000...
/SystemC/top/fabric_w_master0__w_msg/last      1'h1
/SystemC/top/fabric_w_master0__w_msg/wstrb     8'h00
Now                                            741 ns
Cursor 1                                       0.00 ns
```

256 beats from r_master0

256 beats from r_master1

64 beats from r_master0

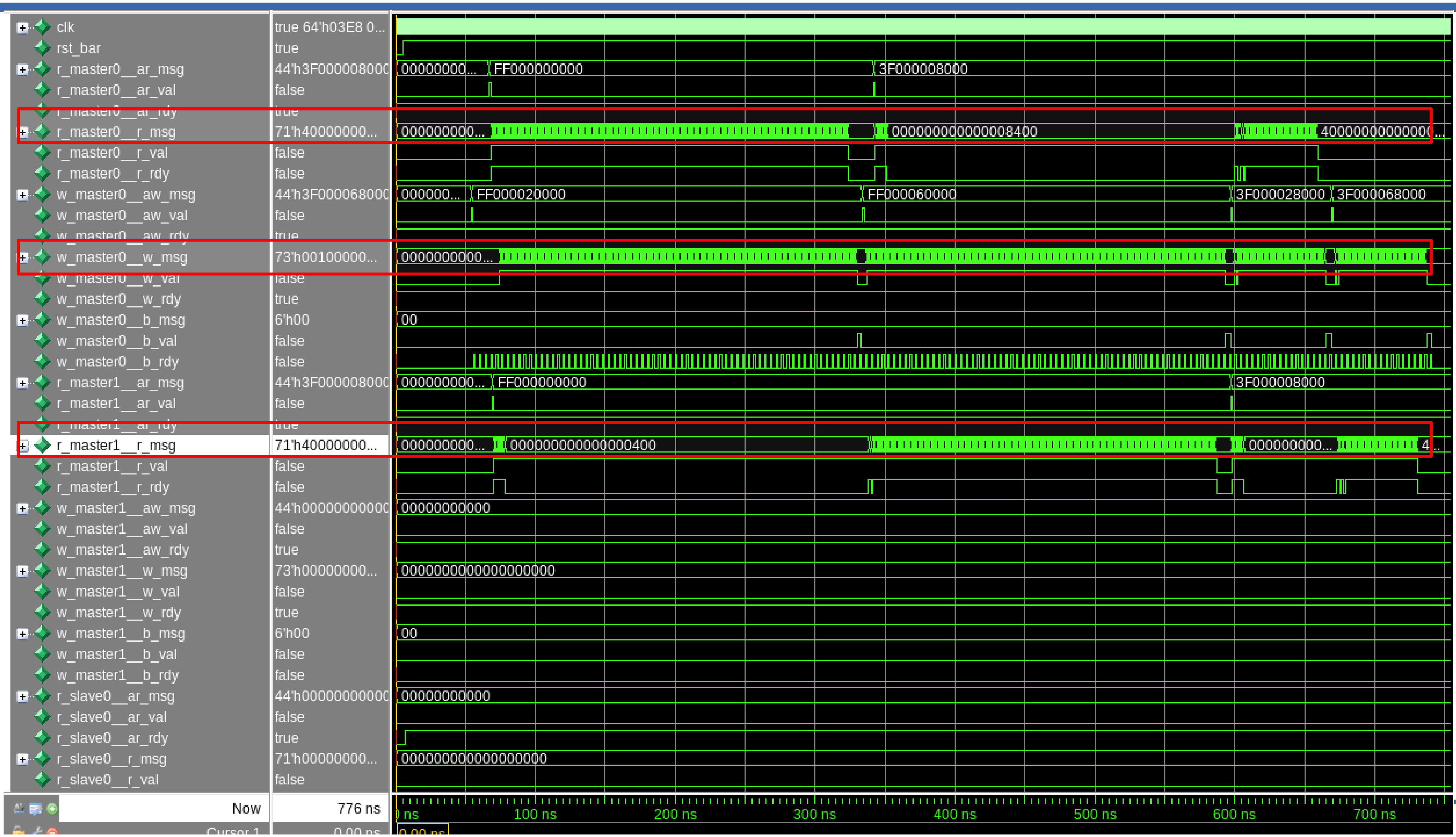
64 beats from r_master1

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 Catapult HLS – Test #1 (Verilog)

Throughput In RTL Matches SystemC



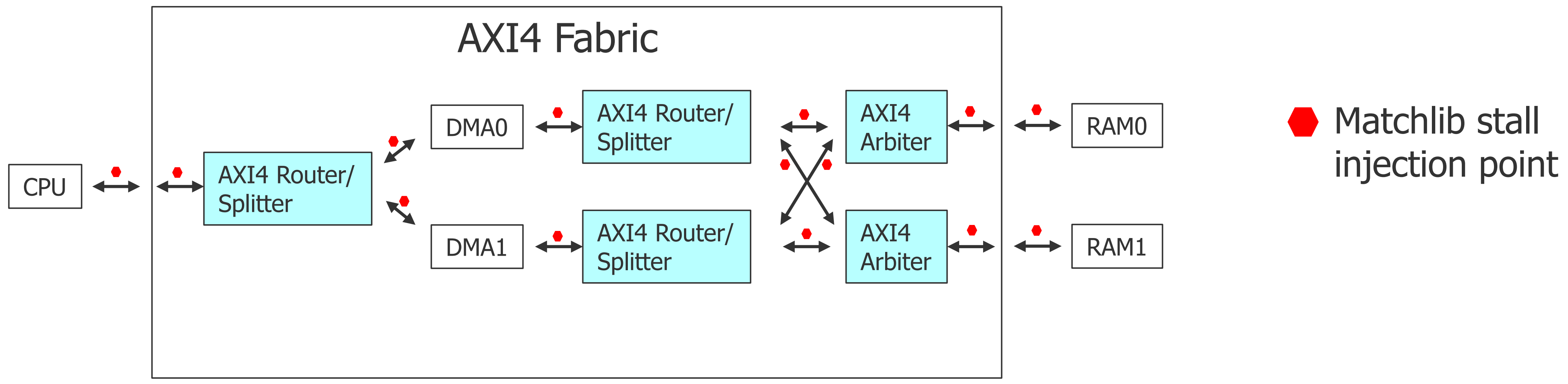
```
clk                  true 64'h03E8 0...
rst_bar              true
r_master0__ar_msg    44'h3F000008000
r_master0__ar_val    false
r_master0__ar_rdy    true
r_master0__r_msg     71'h40000000...
r_master0__r_val     false
r_master0__r_rdy     false
w_master0__aw_msg    44'h3F000068000
w_master0__aw_val    false
w_master0__aw_rdy    true
w_master0__w_msg     73'h00100000...
w_master0__w_val     false
w_master0__w_rdy     true
w_master0__b_msg     6'h00
w_master0__b_val     false
w_master0__b_rdy     false
r_master1__ar_msg    44'h3F000008000
r_master1__ar_val    false
r_master1__ar_rdy    true
r_master1__r_msg     71'h40000000...
r_master1__r_val     false
r_master1__r_rdy     false
w_master1__aw_msg    44'h00000000000
w_master1__aw_val    false
w_master1__aw_rdy    true
w_master1__w_msg     73'h00000000...
w_master1__w_val     false
w_master1__w_rdy     true
w_master1__b_msg     6'h00
w_master1__b_val     false
w_master1__b_rdy     false
r_slave0__ar_msg     44'h00000000000
r_slave0__ar_val     false
r_slave0__ar_rdy     true
r_slave0__r_msg      71'h00000000...
r_slave0__r_val      false
Now                  776 ns
Cursor 1             0.00 ns
```

Mentor
A Siemens Business

What about QOR when using Matchlib + HLS ?

- Matchlib has been carefully designed to give good power/perf/area QOR thru HLS.
- For example:
  - Loops are coded so that HLS can unroll them and they generate efficient HW
  - Variables are sized to minimum necessary bitwidths
  - Unneeded functionality can be eliminated via compile-time parameter settings
- NVIDIA has taped out 10M+ gate chips using this flow, and is quite happy with QOR
  - See the NVIDIA paper and video referenced earlier in this presentation for discussion
  - NVIDIA DAC paper: "comparable QoR (±10%) can be achieved" compared to hand-written RTL
- Matchlib makes it possible evaluate multiple candidate architectures and select the best
  - This may have a larger impact on QOR than more limited-scale optimizations

Matchlib enables automatic stall injection in C++



- Every Matchlib channel can have automatic randomized stall injection enabled.
  - Stall injection stress-tests all communication in system.
- <u>Very quick and effective way to uncover bugs early.</u>

Matchlib + HLS enables efficient verification flow

- Smaller, faster models
  - C++/SystemC model is typically about 1/10 or less than the size of the RTL model
  - Simulates ~30 times faster than RTL models in timing accurate mode
  - Simulates ~300 times faster than RTL models in blocking TLM mode
- Easier integration with C/C++ reference models and C/C++ software/firmware code
  - Easy to integrate C stimulus generators, C reference models for checking
  - Using a single C++ debugger for all C++ code (TB, DUT) has advantages over separate language debuggers
  - SW/HW interactions can be tested early by running SW in host-code mode against SystemC HW model
- C++/SystemC testbench reuse
  - Catapult makes it possible to automatically use same testbench for SystemC and Verilog models
- You can put thin Verilog wrapper around SystemC DUT
  - If using SV UVM DV flow, enables SV DV effort to start much earlier (even before any HLS)

Recommended Verification Flow

- Make C++/SystemC testbench self-checking
  - Check functional correctness
  - Check performance correctness
  - Apply same TB to SystemC model as well as HLS-generated Verilog RTL
- Use Catapult verification tools
  - Use Catapult C Design Checker to find model coding issues without simulation
  - Use Catapult Coverage (CCOV) to evaluate test completeness over C model source code
  - Use Catapult functional coverage (CCOV) to evaluate test completeness over verification goals
- Set up automated regression tests on C++/SystemC models to enable continuous integration
- Set up automated tests to run Catapult HLS and run tests on generated RTL
  - Helps catch downstream issues quickly
- Goal: find and fix almost all bugs early, in C++/SystemC

Recap: NVIDIA Matchlib and Catapult 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.
- Mentor Catapult HLS team is ready to help design teams adopt and succeed with Matchlib
  - Mentor-provided fixes and enhancements to Matchlib source code
  - Mentor-provided examples, assistance
  - Mentor can help you identify best step-wise approach for adoption of Matchlib + HLS
  - Contact: stuart_swan@mentor.com
- 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://uploads-ssl.webflow.com/5a749b2fa5fde0000189ffc0/5d3b1bef8474c4537c1d494b_Khailany_Brucek_CRAFT_Final.pdf
  - http://crva.io/documents/agile-and-open-hardware/khailany-sigarch-visioning-oahw2019.pdf
  - https://www.youtube.com/watch?v=n8_G-CaSSPU

Mentor®

A Siemens Business

www.mentor.com