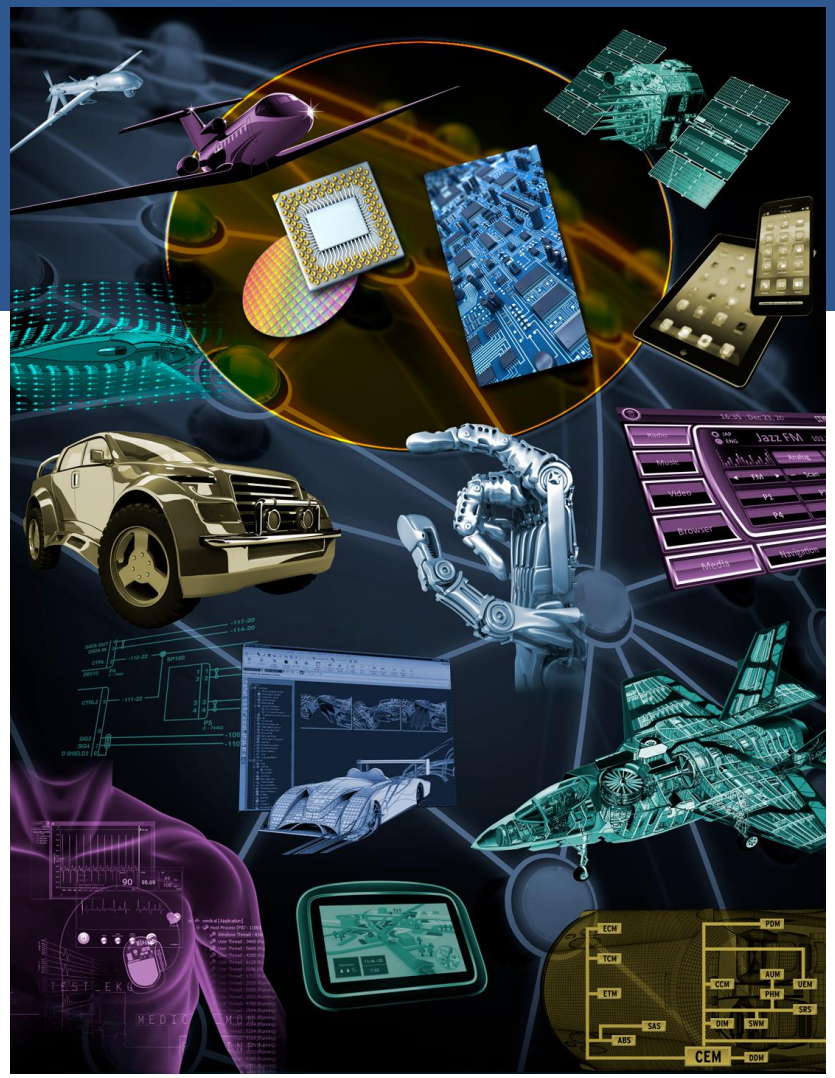


SystemC in the Real World - Moving Up in the World

Stuart Swan

HLS IP/Platform Architect

DAC: June 2019

Mentor®
A Siemens Business

Introduction

- My background
  - Mentor, Qualcomm, Cadence
  - Long involvement with SystemC standards
  - Direct involvement with many semiconductor companies
- Outline of Talk:
  1. Some general observations on moving up in model abstraction based on real-world experience across a number of companies
  2. Concrete example of using a single abstract model for both HW and SW for full chip

General Observations

- Moving up in model abstraction works, provides benefits.
- Companies are using SystemC in production for complex designs
  - HLS, virtual platforms, design verification, architectural analysis
  - You probably have a chip in your pocket that was designed with SystemC
- Current SystemC model adoption is fairly uneven
  - Frequently organizational issues will dictate the chosen technical approach.
- To successfully move up in model abstraction:
  - teams need catalyst to spur change
  - teams need good up-front understanding of where the risk/pain points are in a particular project
  - teams usually need some outside help in adopting new modeling approach

Benefits vs Costs…

- For your project, do the benefits of developing SystemC models outweigh the costs?
- Need to increase benefits and reduce costs! How?
  - Do more verification of larger part of system earlier, at higher level of abstraction.
  - "Integrate early and often" - enable continuous integration
  - Take advantage of high level synthesis (HLS)
  - Avoid writing duplicate models
  - Push back gently against natural tendency of different groups to go off and "do their own thing".

“But our group needs to write our own model because…”

- “We need our models to work in Matlab”
  - SystemC models can integrate into Matlab via mex
- “SW/FW guys need their own address map accurate model”
  - Make all models address map accurate
- “Our virtual platform needs to support RTOS / assembly code”
  - Create thin RTOS emulation API in SystemC to enable host code
- “DV requires everything in SV”
  - Use uvm_connect to enable SC/SV integration
- “Our architects require HW timing accuracy early in project”
  - Use open source NVIDIA Matchlib library
- “We need multiple models to support derivative designs”
  - Use modular SW techniques, C++ traits, #ifdef, so you can still have single model

A State of the Art SystemC Example

- NVIDIA Research has developed a new SystemC-based flow
  - They use HLS to synthesize a full-chip machine learning accelerator
  - Almost all design and verification done with single source SystemC model
  - HLS provides fully automated flow to placed gates
  - Chip has taped out and results are publicly available
- Flow is based on NVIDIA's Matchlib SystemC library
  - Matchlib library is open source on Github
  - NVIDIA Matchlib video seminar available on the web

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.

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.

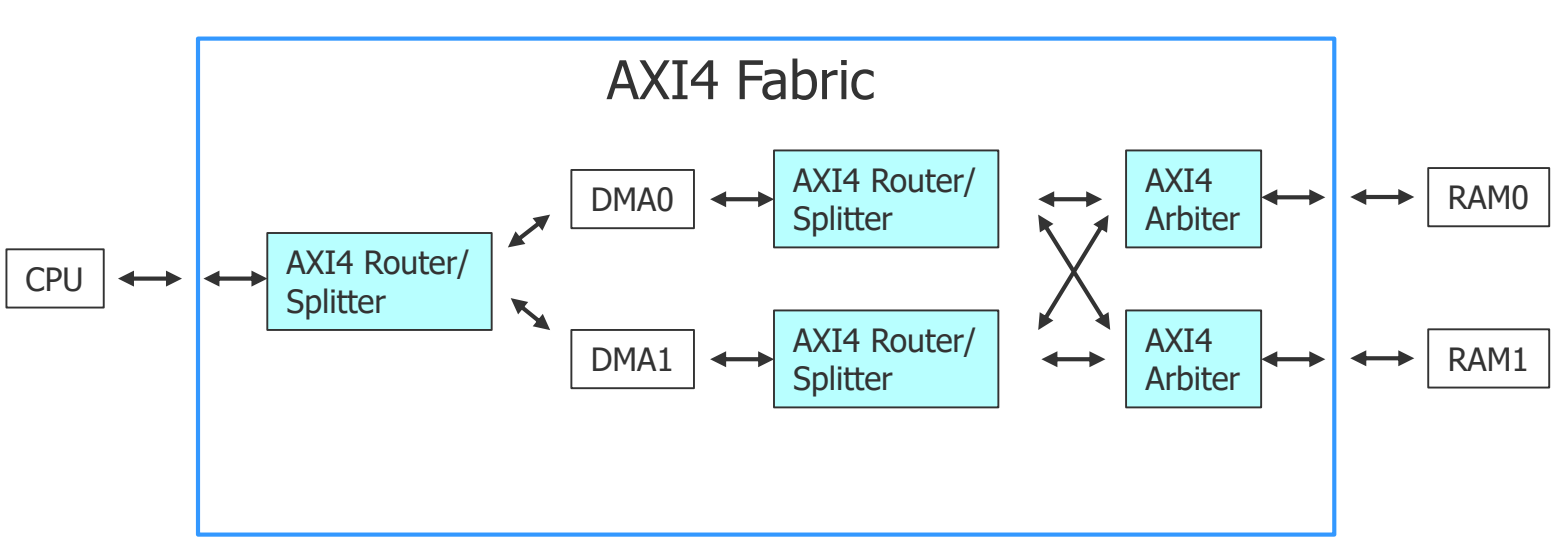
Key Parts of Matchlib

- "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

Matchlib Example: CPU + AXI4 Bus Fabric



Address Map

0x00000

0x7FFFF

0x80000

0x8FFFF

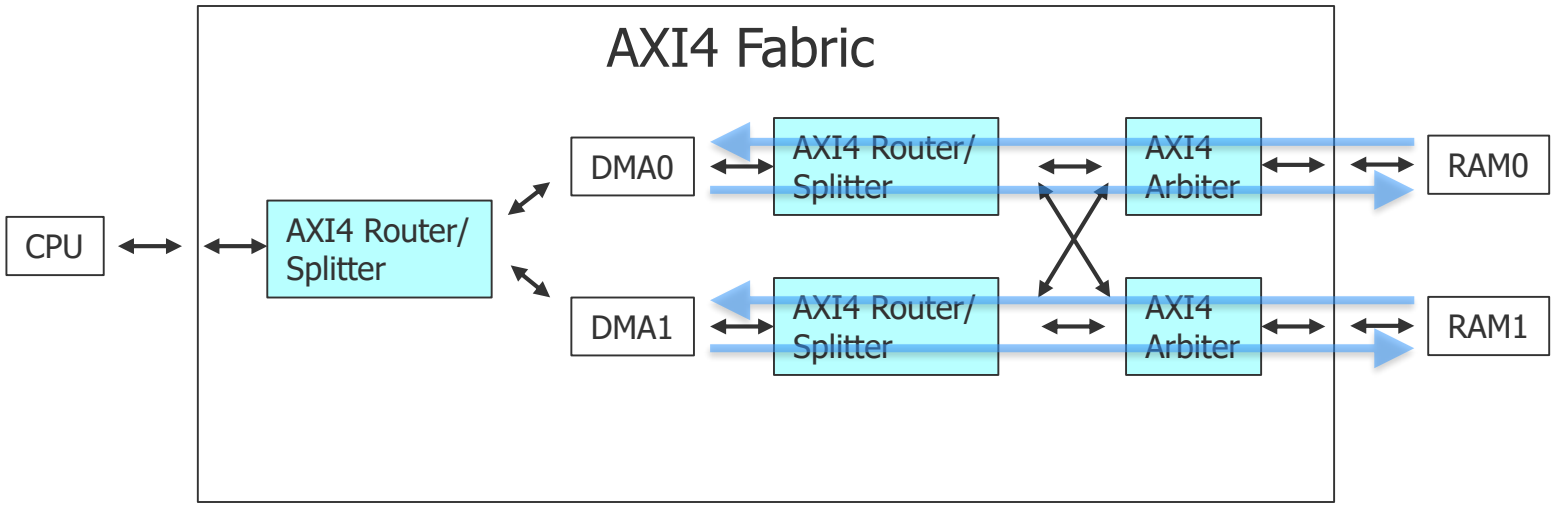
```
/**
 *  * \brief fabric module
*/
#pragma hls_design top
class fabric : public sc_module, public local_axi {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  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

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)

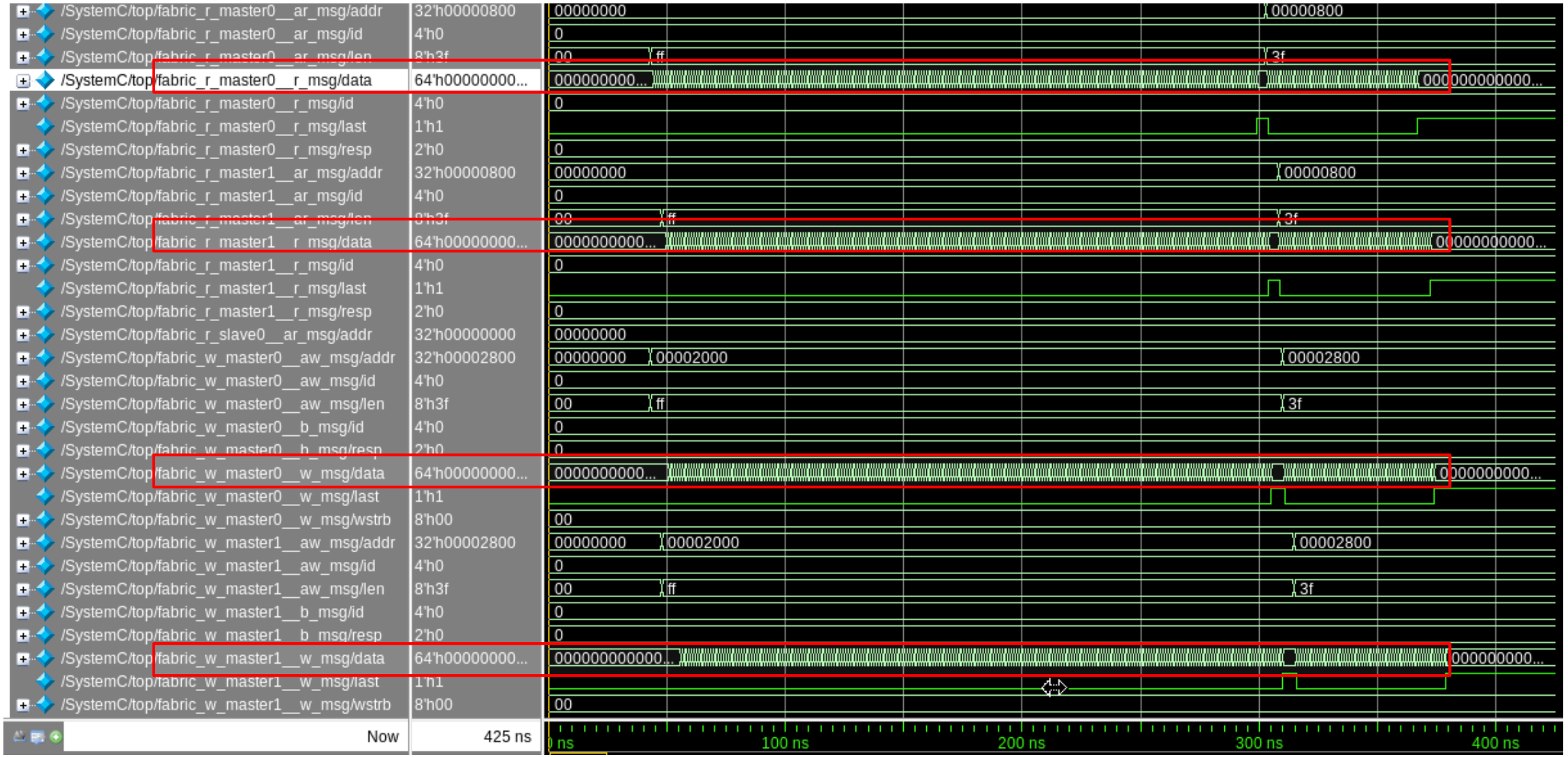
```
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 0
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 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
```

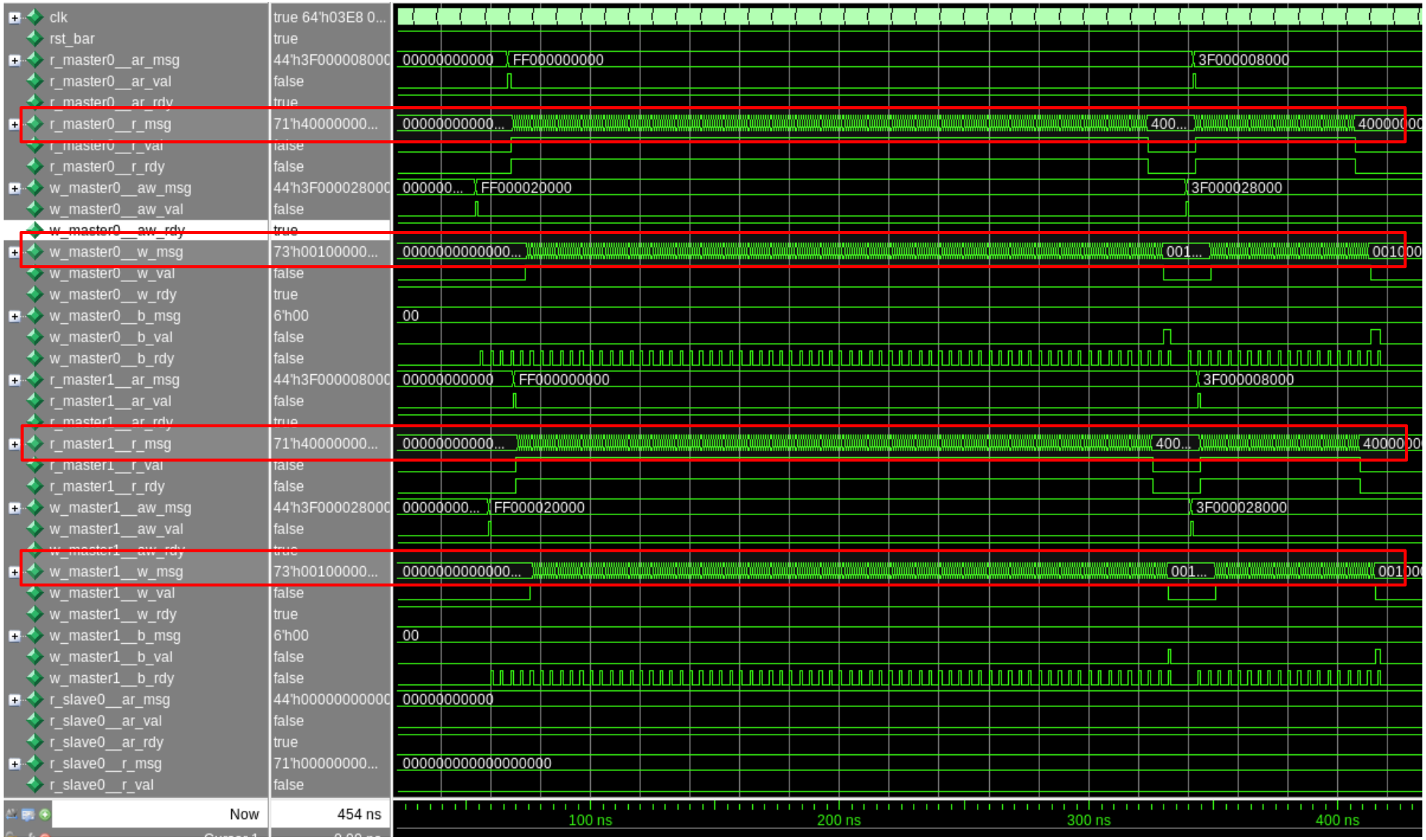
Before and after HLS we get nearly one beat per clock cycle

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

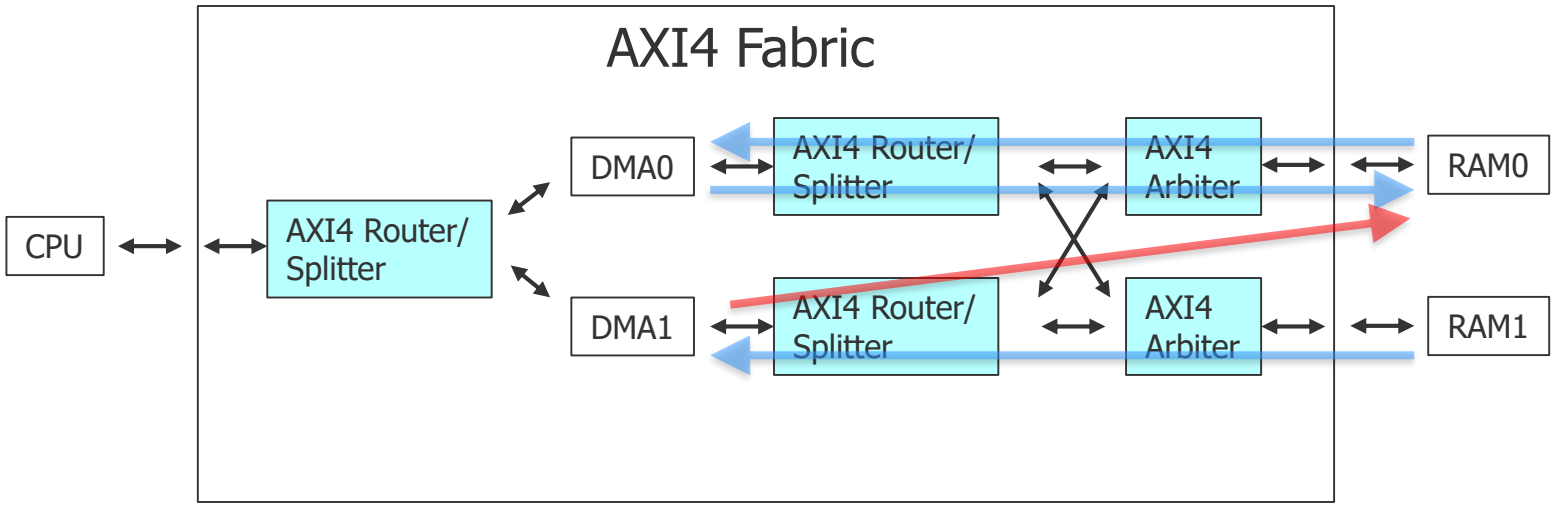


AXI4 Fabric Waveforms After HLS – Test #0 (Verilog)

Throughput In RTL Matches SystemC



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)

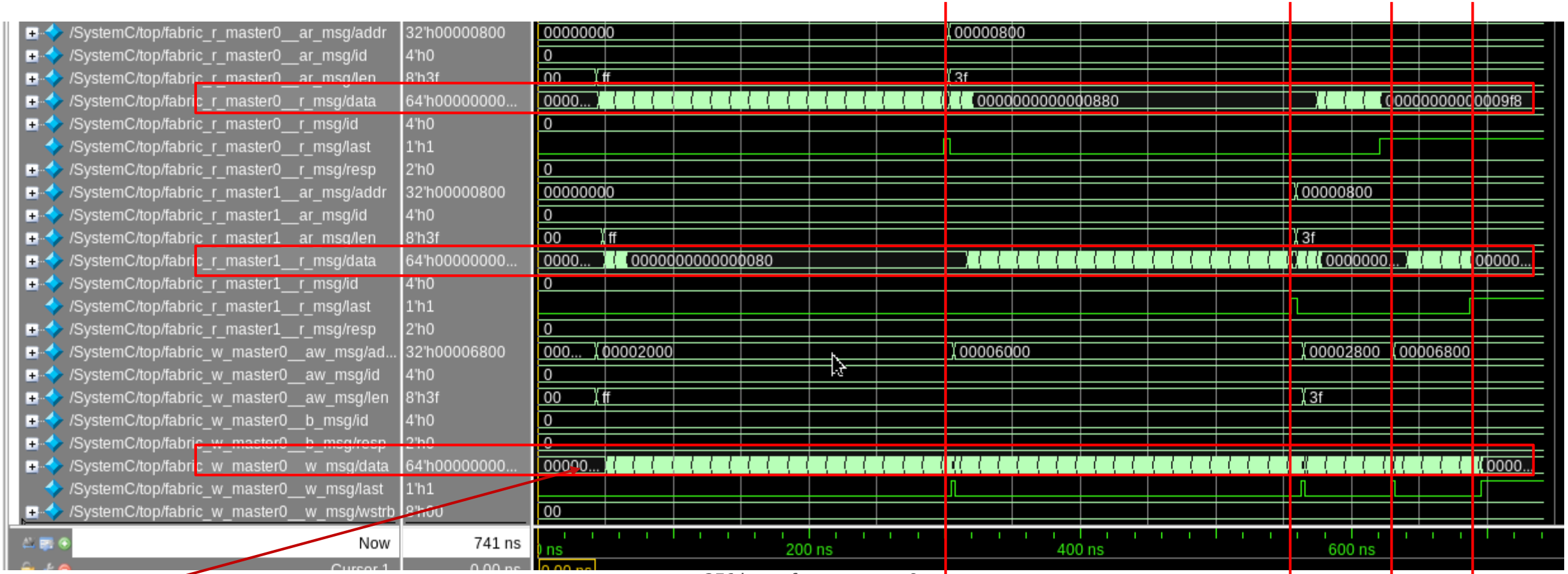
```
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 1
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 len: 03f
308 ns top.ram0 ram write addr: 000006000 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 HLS –Test#1 (SystemC)

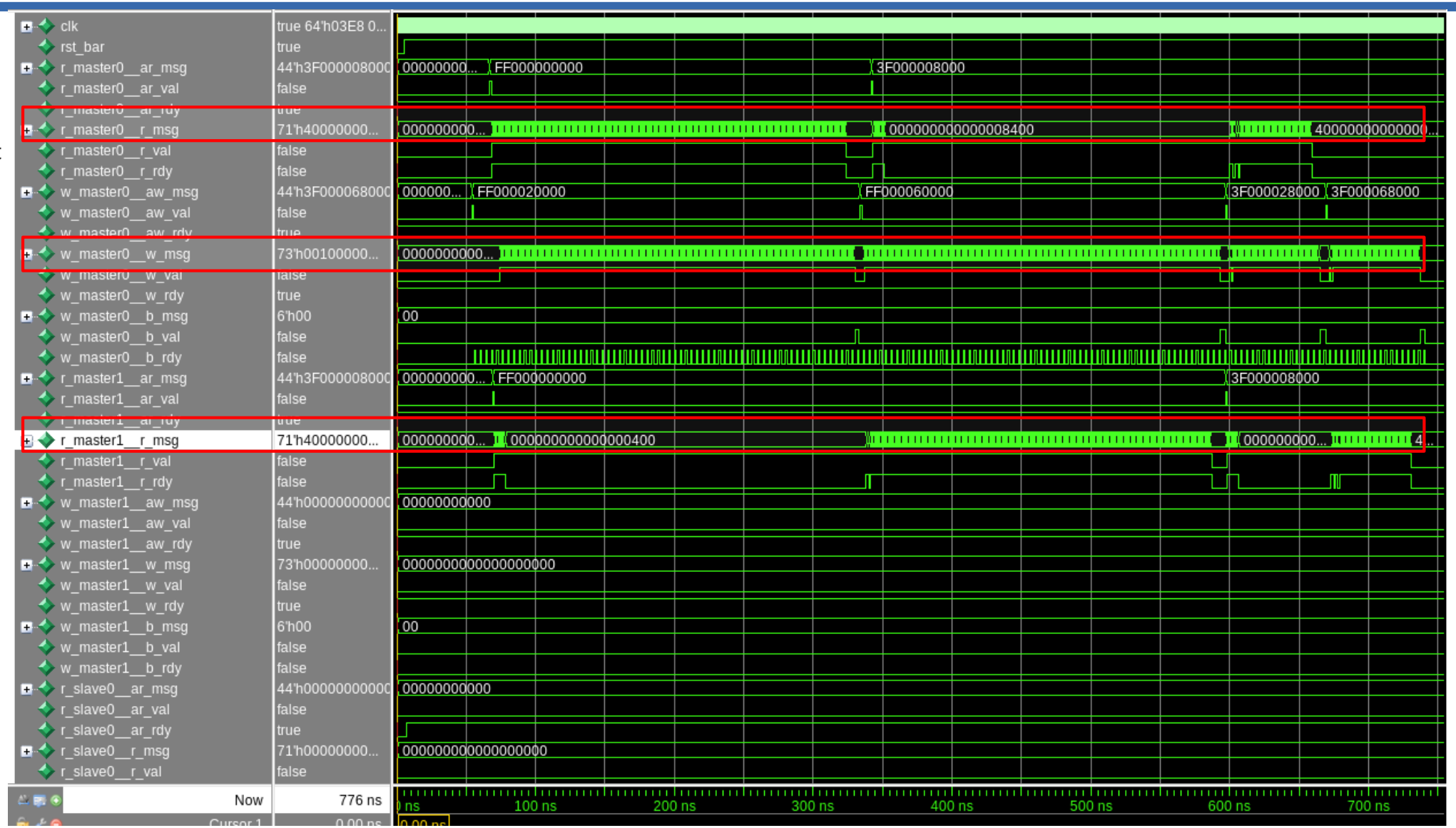


w_master0 fully utilized over 700 ns due to write contention

r_master0 and r_master1 underutilized due to write contention

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

Throughput In RTL Matches SystemC



Conclusion

- SystemC is in use in the real world, and provides real benefits
  - But, need to be clear-eyed about benefits vs costs
- Matchlib and HLS are good example of a modern SystemC-based 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.