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

Stuart Swan, Platform Architect, Mentor, A Siemens Business
stuart_swan@mentor.com
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
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

The DMA reads and writes to the RAM

/** *
* dma module
*/

#include <systemc>

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_master);
    w_master INIT_S1(w_master);
    r_slave INIT_S1(r_slave);
    w_slave INIT_S1(w_slave);

    Connections::Out<bool> INIT_S1(dma_done);

    // DMA signals
    w_master0 w_master
    r_master0 r_master
    w_slave0 w_slave
    r_slave0 r_slave
    dma_done dma_done

    // Control Registers
    Control Regs

    // clk
    clk clk
    rst rst

    // CPU Stimulus
    clk clk
    rst rst
    w_master0 w_master
    r_master0 r_master
    dma_done dma_done

    // RAM
    clk clk
    rst rst
    w_slave0 w_slave
    r_slave0 r_slave
The DMA performs a memory copy using AXI4 bursts

```c
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 hls 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);
  } ...
```

Entire AXI4 DMA C++ is 170 lines
RTL after HLS is 6000 lines
AXI4 DMA Waveforms Before HLS (SystemC simulation)

Automatic AXI4 last bit generation

Automatic AXI4 burst address segmentation

Read and write burst streams are concurrent.

Read/write bus utilization is 100%
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);
```

Address Map

- 0x00000
- 0x7FFFF
- 0x80000
- 0x8FFFF

Top level of design.

CPU

Two DMA instances.

AXI4 Router/Splitter

DMA0

AXI4 Router/Splitter

DMA1

AXI4 Router/Splitter

AXI4 Arbiter

RAM0

AXI4 Arbiter

RAM1

Two RAM instances.

Yellow boxes are MatchLib Components

= top level of design
AXI4 Bus Fabric using MatchLib – Test #0

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

RAM0 and RAM1 each have one read and one write port
Before and after HLS we get nearly one beat per clock cycle.
AXI4 Fabric Waveforms Before HLS–Test #0 (SystemC)

© Accellera Systems Initiative
AXI4 Fabric Waveforms After HLS – Test #0 (Verilog)

Master0 Read Data

Master0 Write Data

Master1 Read Data

Master1 Write Data

Throughput in RTL Matches SystemC

© Accellera Systems Initiative
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

<table>
<thead>
<tr>
<th>BEFORE HLS (SystemC simulation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 s top Stimulus started</td>
</tr>
<tr>
<td>6 ns top Running FABRIC_TEST # : 1</td>
</tr>
<tr>
<td>44 ns top.ram0 ram read addr: 0000000000 len: 0ff</td>
</tr>
<tr>
<td>44 ns top.ram0 ram write addr: 0000020000 len: 0ff</td>
</tr>
<tr>
<td>49 ns top.ram1 ram read addr: 0000000000 len: 0ff</td>
</tr>
<tr>
<td>304 ns top.ram0 ram read addr: 0000008000 len: 03f</td>
</tr>
<tr>
<td>308 ns top.ram0 ram write addr: 0000060000 len: 0ff</td>
</tr>
<tr>
<td>560 ns top.ram1 ram read addr: 0000008000 len: 03f</td>
</tr>
<tr>
<td>566 ns top.ram0 ram write addr: 0000028000 len: 03f</td>
</tr>
<tr>
<td>632 ns top.ram0 ram write addr: 0000068000 len: 03f</td>
</tr>
<tr>
<td>701 ns top dma_done detected. 1 1</td>
</tr>
<tr>
<td>701 ns top start_time: 46 ns end_time: 701 ns</td>
</tr>
<tr>
<td>701 ns top axi beats (dec): 320</td>
</tr>
<tr>
<td>701 ns top elapsed time: 655 ns</td>
</tr>
<tr>
<td>701 ns top beat rate: 2047 ps</td>
</tr>
<tr>
<td>701 ns top clock period: 1 ns</td>
</tr>
<tr>
<td>741 ns top finished checking memory contents</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AFTER HLS (Verilog RTL simulation)</th>
</tr>
</thead>
<tbody>
<tr>
<td># 0 s top Stimulus started</td>
</tr>
<tr>
<td># 55 ns top.ram0 ram write addr: 0000020000 len: 0ff</td>
</tr>
<tr>
<td># 68 ns top.ram0 ram read addr: 0000000000 len: 0ff</td>
</tr>
<tr>
<td># 70 ns top.ram1 ram read addr: 0000000000 len: 0ff</td>
</tr>
<tr>
<td># 335 ns top.ram0 ram write addr: 0000060000 len: 0ff</td>
</tr>
<tr>
<td># 343 ns top.ram0 ram read addr: 0000008000 len: 03f</td>
</tr>
<tr>
<td># 598 ns top.ram1 ram read addr: 0000008000 len: 03f</td>
</tr>
<tr>
<td># 598 ns top.ram0 ram write addr: 0000028000 len: 03f</td>
</tr>
<tr>
<td># 670 ns top.ram0 ram write addr: 0000068000 len: 03f</td>
</tr>
<tr>
<td># 736 ns top dma_done detected. 1 1</td>
</tr>
<tr>
<td># 736 ns top start_time: 55 ns end_time: 736 ns</td>
</tr>
<tr>
<td># 736 ns top axi beats (dec): 320</td>
</tr>
<tr>
<td># 736 ns top elapsed time: 681 ns</td>
</tr>
<tr>
<td># 736 ns top beat rate: 2128 ps</td>
</tr>
<tr>
<td># 736 ns top clock period: 1 ns</td>
</tr>
<tr>
<td># 776 ns top finished checking memory contents</td>
</tr>
</tbody>
</table>

Two concurrent writes to RAM0 cause beat rate to be above two clock cycles.
AXI4 Fabric Waveforms Before HLS – Test#1 (SystemC)

- Master0 Read Data
- Master1 Read Data
- Master0 Write Data

- 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
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://uploads-ssl.webflow.com/5a749b2fa5fde0000189ffc0/5d3b1bef8474c4537c1d494b_Khailany_Brucek_CRAFT_Final.pdf
  - https://www.youtube.com/watch?v=n8_G-CaSSPU
  - https://forums.accellera.org/files/category/2-systemc/
MatchLib Open Source Example Kit

- [https://forums.accellera.org/files/category/2-systemc/](https://forums.accellera.org/files/category/2-systemc/)
- Or Google: MatchLib Accellera SystemC Evolution Day
- This kit contains a representative set of MatchLib examples presented at Accellera SystemC Evolution Day 2020 and fully self-contained source files and scripts so that the examples can be built and run on any Linux compatible system with no other required software. All contents of the kit are open source.
- Trouble downloading? Contact stuart_swan@mentor.com
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**
- **SystemC Synthesizable Connections Standard**
- **SystemC Synthesizable Subset Standard**
- **SystemC Language Standard**

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

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