Enhancing TBA Performance Using Efficient SCE-MI Modelling

Ponnambalam Lakshmanan
Agenda

• Introduction
• Factors Impacting Acceleration
• SCE-MI
• Testbench Architecture and Existing Challenges
• SCE-MI Direct Memory Interface
• SCE-MI Function Based Interface
• SCE-MI Pipes – Challenges and Solutions
• Results
• Conclusion and Future Work
Introduction

• Acceleration Challenges – Addressed with Hardware assisted testbench acceleration
  – Hardware accelerators to our rescue
  – Various acceleration modes

• Transaction Based Acceleration (TBA)
  – Testbench (synthesizable part) + DUT runs on emulator
  – Testbench (non-synthesizable part) runs on simulator.
Factors Impacting Acceleration

• Various factors directly affect the emulator performance
  – Inefficient usage of SCE-MI
  – Usage of Behavioral constructs (Not purely-synthesizable)
  – Memory Handling
  – Amount of code running on simulator and emulator
  – Simulator – Emulator synchronizations
SCE-MI

• Standard Co-Emulation Modeling Interface (SCE-MI) is an Accellera standard

• Different flavors of SCE-MI
  – Pipe based Transaction
  – Function based Interface
  – Direct Memory Interface (DMI)
SCE-MI Pipes

- Salient Features:
  - Unidirectional
  - Batching
  - Buffering
  - Flushing
  - Data shaping
  - Blocking and Non-Blocking constructs
Testbench Architecture and Existing Challenges

Bottleneck 1: Multiple SCE-MI Pipes

Bottleneck 2: Sending small amount of data frequently

Bottleneck 3: Async loop to clear pipe

Simulator

UVM Master Agent

Proxy Driver

Proxy Monitor

Emulator

BFM Master Driver

BFM Slave Driver

BFM Master Monitor

BFM Slave Monitor

O/P SCE-MI Pipes

I/P SCE-MI Pipes

Proxy Driver

Proxy Monitor

UVM Slave Agent

© Accellera Systems Initiative
SCE-MI Direct Memory Interface

Backdoor Access

Proxy Agent

Simulator

Memory

BFM

DUT

Emulator

Memory

Backdoor Access
SCE-MI Direct Memory Interface

• Used to access DUT/BFM memory from the proxy domain

• Transfer Bulk data (Proxy <-> BFM)
  – Block interface
  – Word interface

• Performance Improvement:
  – HW-SW synchronizations reduced by ~50%
  – TBA run time decreased by >25%

• DUT/BFM registers cannot be accessed
SCE-MI Direct Memory Interface

Proxy invokes C-function to read mem

```c
// C-function to read memory
// This function sends the data stored in the memory back to the proxy side
void read_mem(svBitVecVal return_mem[8192]) {
    static void* vmem;
    int rAddr;
    static unsigned int width, depth;
    vmem = scei_mem_c_handle("hierarchical_path_memory");
    scei_mem_get_size(vmem, (unsigned int*)&width, (unsigned long long*)&depth);
    scei_mem_get_block(vmem, width, depth, return_mem);
}
```

Backdoor read of mem

C membrane

BFM indicating proxy to read memory

Proxy side

Proxy receive unblocks BFM

BFM side

Proxy receive acknowledges mem read

BFM waits for proxy to complete the read

© Accellera Systems Initiative
SCE-MI Function Based Interface

- Used to write into BFM registers
- Memory access through DMI has better performance
SCE-MI Function Based Interface

//C-function has the implementation of proxy's write function
//It also invokes the BFM's write function through it
const char* tpath = "path to the bfm where reg write function is defined";
static svScope tbscp = NULL;
extern void reg_bfm wr (svBitVecVal* reg data);
void write_bfm reg (svBitVecVal* reg data) {
    tbscp = svGetScopeFromName(tpath);
    svSetScope(tbscp);
    reg_bfm wr (reg data);
}

//Proxy code, here is where the write data is passed to the C-function which is to
//be written on to the BFM
import "DPI-C" context;
class proxy driver;
    bit [31:0] data wr;
    task reconfig;
    write_bfm req (data wr);
endtask
endclass

//BFM code, here is where the implementation of the write function resides
//It is invoked by the C-interface
module BFM;
    export "DPI-C" function reg_bfm wr;
    bit [31:0] ctl reg;
    function void reg_bfm wr (input bit[31:0] wr data);
        ctl reg = wr data;
    endfunction
endmodule

Invoke register write from proxy through C
C membrane
Proxy side
BFM register write from C interface
C-function imported and used in proxy
Verilog -function exported and used in the C - interface

© Accellera Systems Initiative
SCE-MI Pipes – Challenges and Solutions

- Multiple instances of SCE-MI pipe

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavioral evals with every SCE-MI access</td>
<td>• Send data via a single SCE-MI pipe.</td>
</tr>
<tr>
<td></td>
<td>• Static SCE-MI pipe - Bit vectors</td>
</tr>
<tr>
<td></td>
<td>• Dynamic SCE-MI pipe - Array of data</td>
</tr>
</tbody>
</table>

Implementation 1

- Proxy
  - Class 1
    - SCE-MI
    - Module 1
  - Class 2
    - SCE-MI
    - Module 2
  - Class 3
    - SCE-MI
    - Module 3

BFM

Implementation 2

- Proxy
  - Class 1
    - SCE-MI
  - Class 2
    - SCE-MI
  - Class 3
    - SCE-MI

BFM
SCE-MI Pipes – Challenges and Solutions

• Behavioral evals with SCE-MI pipe

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Asynchronous mode creates significant behavioral evals</td>
<td>Synchronous mode of SCE-MI pipe IS_CLOCKED_INTF = 1</td>
</tr>
</tbody>
</table>

**Code Snippet**

```verilog
scemi_input_pipe #( .BYTES_PER_ELEMENT(20),
                   .PAYLOAD_MAX_ELEMENTS(1),
                   .VISIBILITY_MODE(2),
                   .IS_CLOCKED_INTF(0) )
inbox (clk);
```

```verilog
scemi_input_pipe #( .BYTES_PER_ELEMENT(20),
                   .PAYLOAD_MAX_ELEMENTS(1),
                   .VISIBILITY_MODE(2),
                   .IS_CLOCKED_INTF(1) )
inbox (clk);
```
SCE-MI Pipes – Challenges and Solutions

- Amount of data accessed

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing more than one element per access results in behavioral evals</td>
<td>If the intention is to access 5 Bytes per call then set</td>
</tr>
<tr>
<td>• PAYLOAD_MAX_ELEMENTS = 5;</td>
<td>• PAYLOAD_MAX_ELEMENTS = 1;</td>
</tr>
<tr>
<td>• BYTES_PER_ELEMENT = 1;</td>
<td>• BYTES_PER_ELEMENT = 5;</td>
</tr>
</tbody>
</table>

5 Elements
  - Byte 1
  - Byte 2
  - Byte 3
  - Byte 4
  - Byte 5

1 Element
  - Byte 5
  - Byte 4
  - Byte 3
  - Byte 2
  - Byte 1
SCE-MI Pipes – Challenges and Solutions

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Frequent data transfer causes the emulator to halt frequently</td>
<td>• Accumulate multiple bytes of data and transfer at once</td>
</tr>
<tr>
<td>• Results in degraded acceleration</td>
<td>• Try to buffer the elements</td>
</tr>
<tr>
<td></td>
<td>• Avoid unnecessary flush() usage</td>
</tr>
</tbody>
</table>

**Code Snippet**

```vhdl
scemi_input_pipe #(  
  .BYTES_PER_ELEMENT(20),  
  .PAYLOAD_MAX_ELEMENTS(1),  
  .BUFFER_MAX_ELEMENTS(10),  
  .VISIBILITY_MODE(2),  
  .IS_CLOCKED_INTF(1) ) inbox (clk);
```
SCE-MI Pipes – Challenges and Solutions

- Clearing the contents of SCE-MI pipe

<table>
<thead>
<tr>
<th>Challenge</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discarding buffer contents on reset</td>
<td>Use the fastest clock available to synchronously fetch data from pipe</td>
</tr>
<tr>
<td>No inbuilt functions</td>
<td></td>
</tr>
<tr>
<td>Increase in step-count</td>
<td></td>
</tr>
</tbody>
</table>

**Code Snippet : Before**

```verbatim
code_snippet_before:
always @(posedge clk, posedge rst) begin
  if(rst) begin
    // Statements
    for(int i=0, i<20, i++)
      inbox.receive(1,ve,data,eom);
  end
  else begin
    // statements
  end
end
```

**Code Snippet : After**

```verbatim
code_snippet_after:
always @(posedge fst_clk, posedge rst) begin
  if(rst) begin
    rst_buff = 1;
    // statements
  end
  else begin
    if(rst_buff && !eom)
      inbox.receive(1,ve,data,eom);
    else
      rst_buff = 0;
  end
end
```

© Accellera Systems Initiative
Testbench Architecture: After Optimization

- SCE-MI Function to configure registers
- SCE-MI Pipe to establish Sync
- SCE-MI DMI to read data from BFM memory
- SCE-MI DMI to write into BFM Memory
- Replaced Data Pipe with SCE-MI DMI
- Replaced multiple data SCE-MI pipes with SCE-MI DMI

Diagram:
- Proxy Driver
- Proxy Monitor
- UVM Master Agent
- Emulator
- UVM Slave Agent
- Simulators
Results

• TBA performance improvement

<table>
<thead>
<tr>
<th></th>
<th>Gate Count</th>
<th>Bevals</th>
<th>HW-SW Sync</th>
<th>TBA Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Optimization</td>
<td>~2 M</td>
<td>18,299,173</td>
<td>4,728,998</td>
<td>~60 min</td>
</tr>
<tr>
<td>After Optimization</td>
<td>~2 M</td>
<td>218</td>
<td>18,439</td>
<td>5 min</td>
</tr>
</tbody>
</table>

• Simulation v/s TBA run-time comparison

<table>
<thead>
<tr>
<th></th>
<th>Simulation Time</th>
<th>TBA Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation v/s TBA</td>
<td>~360 min</td>
<td>~9 min</td>
</tr>
</tbody>
</table>
Conclusion

• SCE-MI has multiple interfaces for different use cases

• HW-SW sync and Bevals could greatly deteriorate the performance of emulator.

• Efficient usage of SCE-MI helps in leveraging maximum performance from emulator resulting in handsome **SPEED-UP**.
Future Work

- To integrate the agents with the actual DUT (IP) and test on emulator
- Porting to SOC environment and analyze the performance
Acknowledgements

• David Brownell, ADI
Questions