2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Efficient Standard Co-Emulation Modeling Interface (SCE-MI) Usage to Accelerate TBA Performance

Ponnambalam Lakshmanan

Analog Devices, Bengaluru, India

ANALOG DEVICES

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

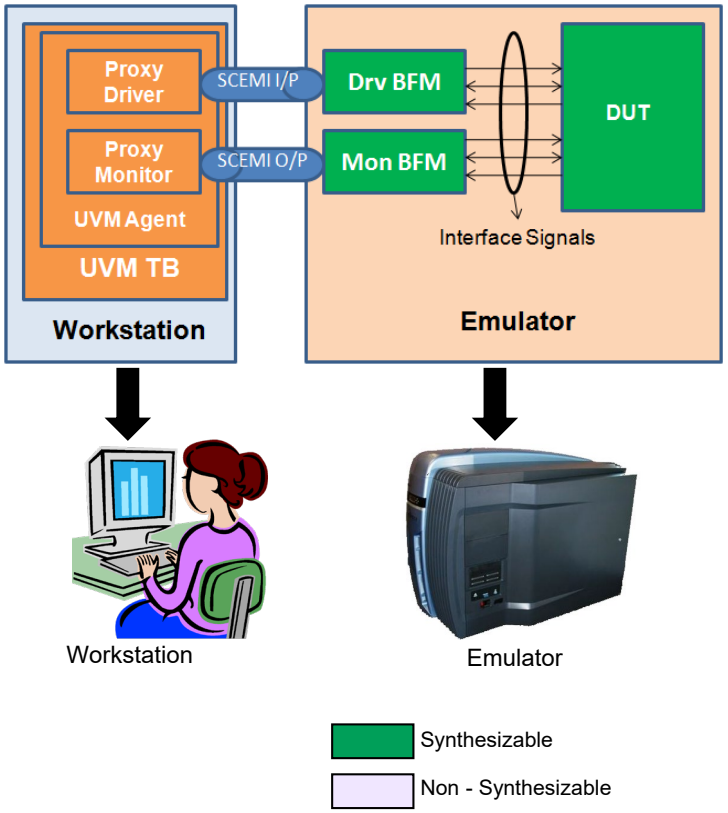
Acceleration Concept

- Hardware assisted acceleration offers better performance over software simulation.

- Hardware assisted acceleration techniques:
  - Signal Based Acceleration (SBA)
  - Transaction Based Acceleration (TBA)
  - Embedded testbench
  - Vector Based Acceleration (VBA)
  - In-circuit Emulation (ICE)

Transaction Based Acceleration

- Traditional testbench is split into two domains
  - Non-Synthesizable domain
    - Runs on simulator
  - Synthesizable domain
    - Runs on emulator



Factors Affecting Acceleration

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

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

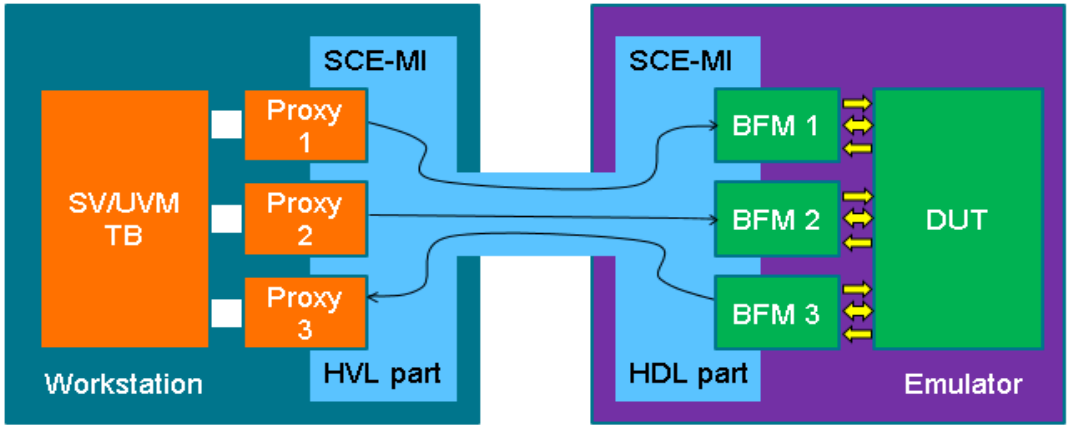
- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

Introduction to SCE-MI

- Standard Co-Emulation Modeling Interface (SCE-MI) is an Accellera standard
- Communication interface between BFM and proxy
- Different types of SCE-MI use models:
  - SCE-MI Pipe based interface
  - SCE-MI Direct memory interface (DMI)
  - SCE-MI Function based interface

SCE-MI Pipes

- Salient Features:
  - Unidirectional
  - Batching
  - Buffering
  - Flushing
  - Data shaping
  - Blocking and Non-Blocking constructs

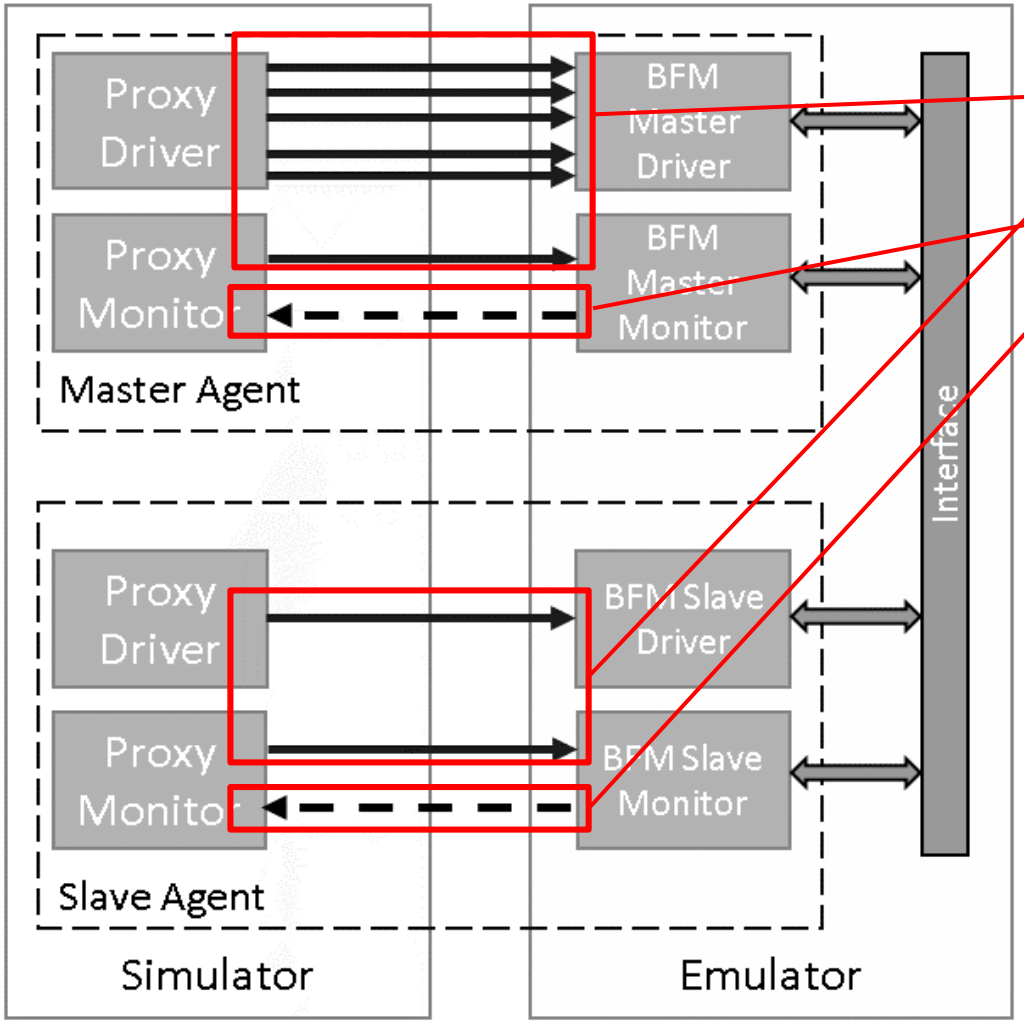


2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

Initial Architecture Challenges



Input SCEMI Pipes

Output SCEMI Pipes

Bottlenecks

- Multiple SCEMI pipes
- Transmission of small packets frequently
- Async loop to clear pipe contents

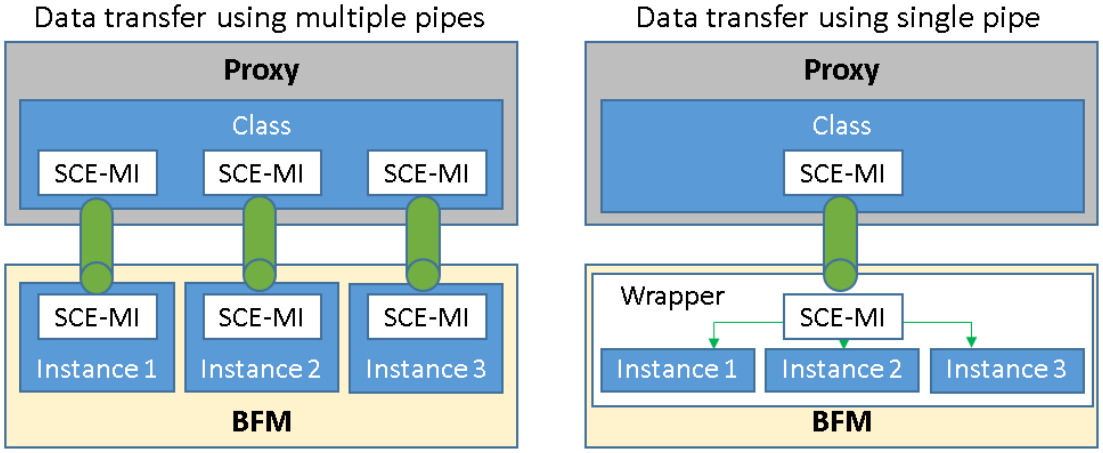
Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

Optimizing SCE-MI Pipe Usage

- Merging SCE-MI pipes

Challenge	Solution
Behavioral evals with every SCE-MI access	• Send data via a single SCE-MI pipe. • Static SCE-MI pipe - Bit vectors • Dynamic SCE-MI pipe - Array of data



Optimizing SCE-MI Pipe Usage

- Synchronous access

Challenge	Solution
• Asynchronous mode creates significant behavioral evals	Synchronous mode of SCE-MI pipe IS_CLOCKED_INTF = 1

Code Snippet

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

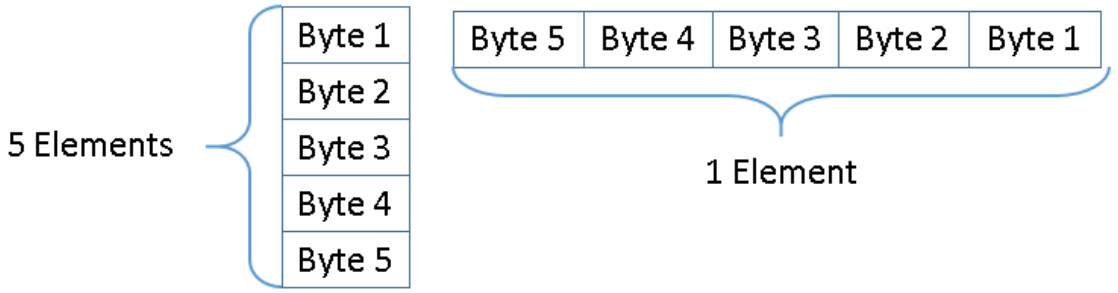
Code Snippet

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

Optimizing SCE-MI Pipe Usage

- Optimized data transfer

Challenge	Solution
Accessing more than one element per access results in behavioral evals • PAYLOAD_MAX_ELEMENTS = 5; • BYTES_PER_ELEMENT = 1;	If the intention is to access 5 Bytes per call then set • PAYLOAD_MAX_ELEMENTS = 1; • BYTES_PER_ELEMENT = 5;



Optimizing SCE-MI Pipe Usage

- Minimal Synchronization

Challenge	Solution
• Frequent data transfer causes the emulator to halt frequently • Results in degraded acceleration	• Accumulate multiple bytes of data and transfer at once • Try to buffer the elements • Avoid unnecessary flush() usage

Code Snippet

```
scemi_input_pipe #(
.BYTES_PER_ELEMENT(20),
.PAYLOAD_MAX_ELEMENTS(1),
.BUFFER_MAX_ELEMENTS(10),
.VISIBILITY_MODE(2),
.IS_CLOCKED_INTF(1) ) inbox (clk);
```

Optimizing SCE-MI Pipe Usage

- Clearing the pipe

Challenge	Solution
• Discarding buffer contents on reset • No inbuilt functions • Increase in step-count	• Use the fastest clock available to synchronously fetch data from pipe

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

```
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
```

Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

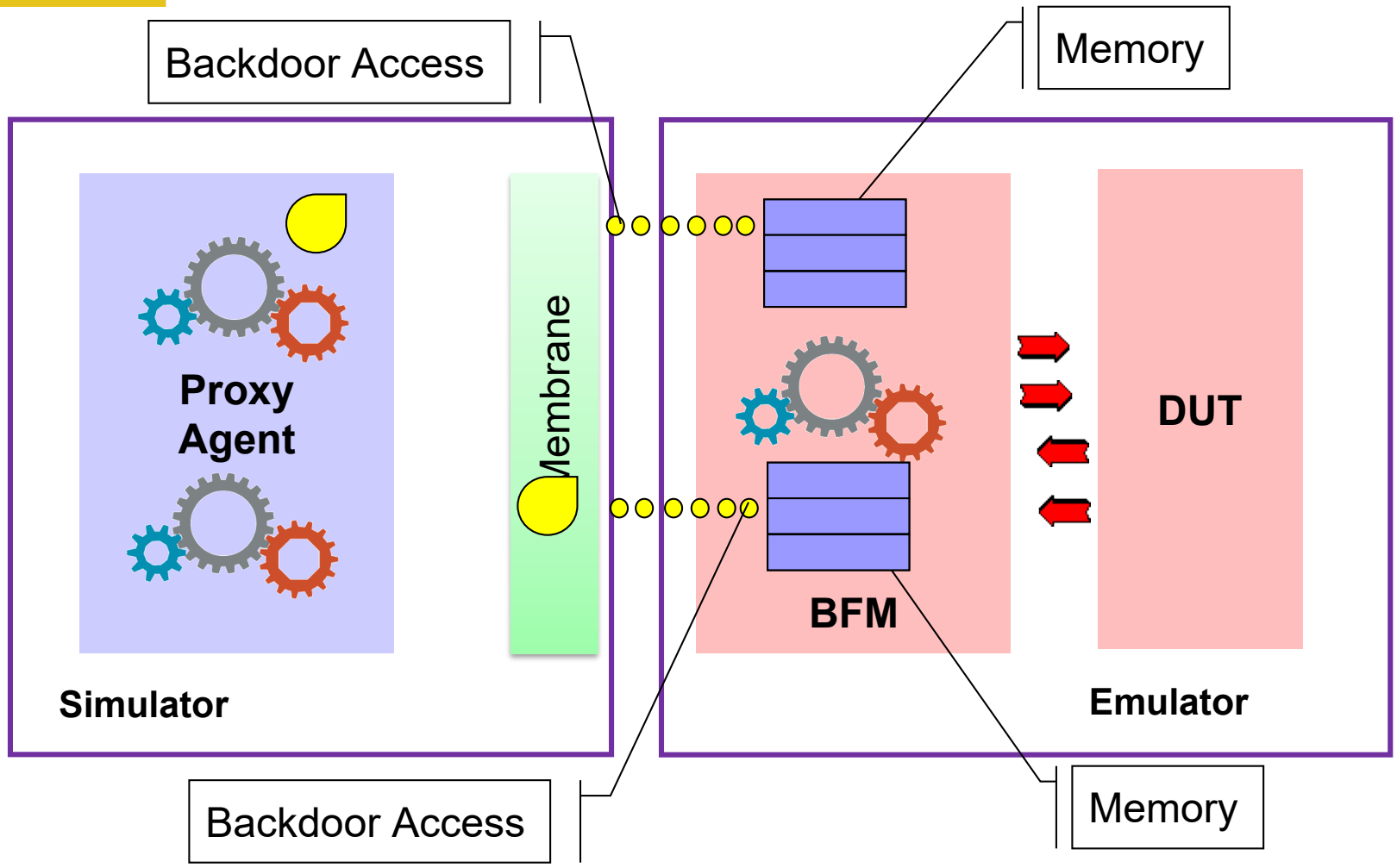
SCE-MI Direct Memory Interface

- Software side interface to perform backdoor read/write operations on hardware side memories

- Types of interfaces
  - Block interface
  - Word interface

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

SCE-MI Direct Memory Interface



SCE-MI Direct Memory Interface

C - Backdoor memory access through predefined SCE-MI API

C-membrane writes the memory contents in to the proxy

Invoke a memory read from BFM

C membrane

```
cont char* tbpath = "path to the proxy where mem_access is defined"
static svScope tbscp = NULL;

extern void mem_access (svBitVecVal* mem);
void read_mem(){
   static void* vmem;
   svBitVecVal mem;
   int rAddr;
   static unsigned int width, depth;
   tbscp = svGetScopeFromName(tbpath);
   svSetScope(tbscp);
   vmem = scemi_mem_c_handle("hierarchical_path.mem");
   scemi_mem_get_size(vmem, (unsigned int *)&width, (unsigned long long *)&depth);
   scemi_mem_get_block(vmem, rAddr, depth, mem);
   mem_access(mem);
}
```

Proxy side

```
export "DPI-C" function mem_access;
class proxy_monitor;
   function mem_access(input bit [7:0] return_mem [8192]);
      //logic code to process the read BFM memory
   endfunction
endclass
```

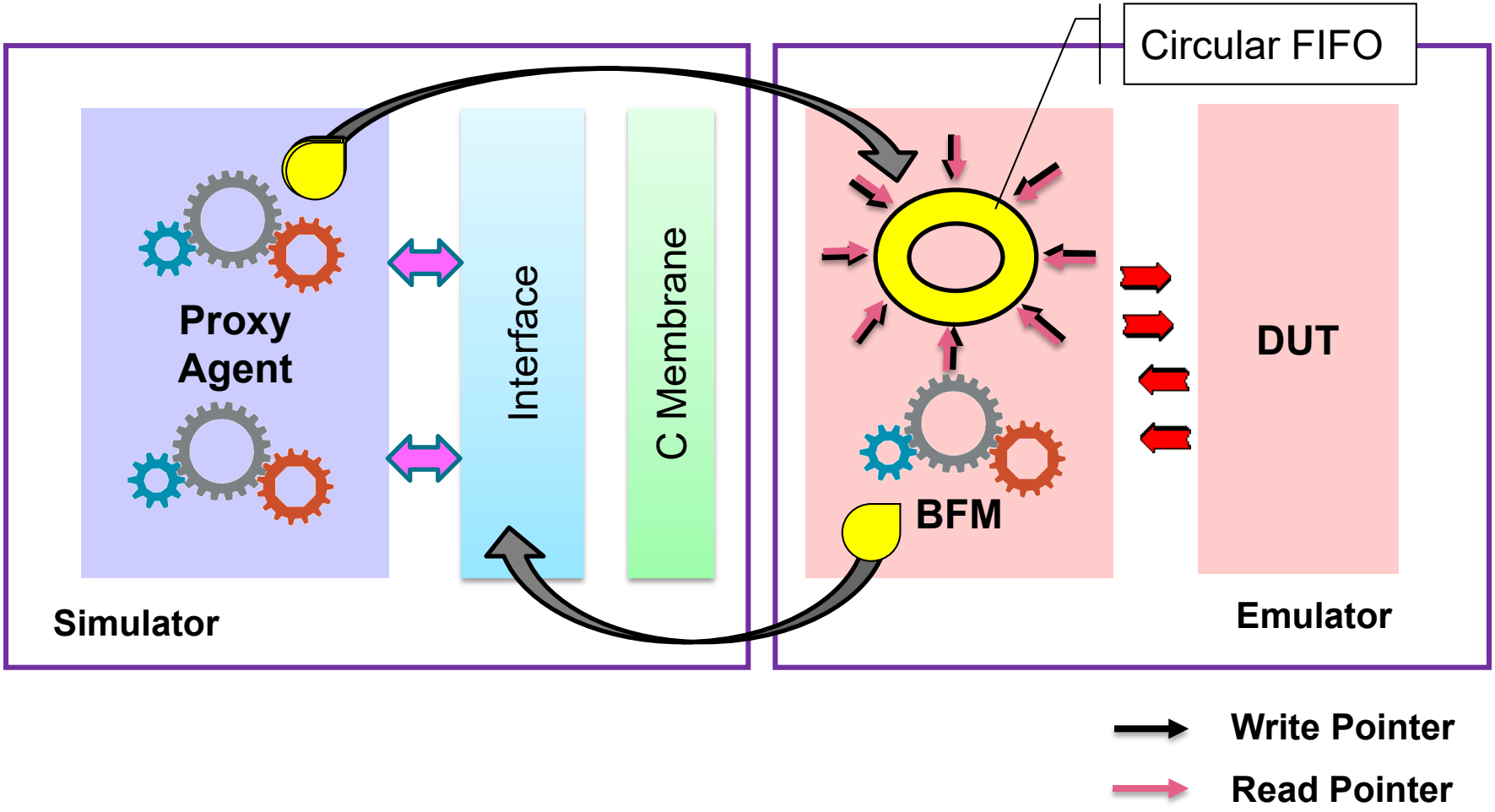
BFM side

```
import "DPI-C" task read_mem();
module monitor_bfm;
   bit [31:0] mem [8192];
   bit [31:0] wr_ptr;
   bit        tbs; //Synchronizer between proxy and BFM
   always @(posedge clk) begin
      mem[wr_ptr] = data;
      if(wr_ptr == FULL) begin
         read_mem;
         tbs = ~tbs;
      end
      else wr_ptr = wr_ptr ++;
   end
endmodule
```

SCE-MI Function Based Interface

- Adopts SystemVerilog Direct Programming Interface (DPI) concept.
- End user is required to implement all the functions
  - No built in functions
- Used to configure BFM registers

SCE-MI Function Based Interface



SCE-MI Function Based Interface

```
//C-function has the implementation of proxy's write function
//It also invokes the BFM's write function throuh it
const char* tbpath = "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_c(svBitVecVal* reg_data){
    tbscp = svGetScopeFromeName(tbpath);
    svSetScope(tbscp);
    reg_bfm_wr(reg_data);
}
```

C membrane

BFM register write from C interface

```
//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 write_bfm_write_c = function void write_bfm_reg(bit [31:0] data);
class proxy_driver;
   bit [31:0] data_wr;
   task reconfig;
      write_bfm_reg(data_wr);
   endtask
endclass
```

C-function imported and used in proxy

Proxy side

Invoke register write from proxy through C

```
//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
```

BFM side

Verilog -function exported and used in the C - interface

Trimming Development Time

- Factors
  - Increase in effort and time to implement complex functionality in a synthesizable format.
  - Increase in area occupied by the BFM on the emulator.

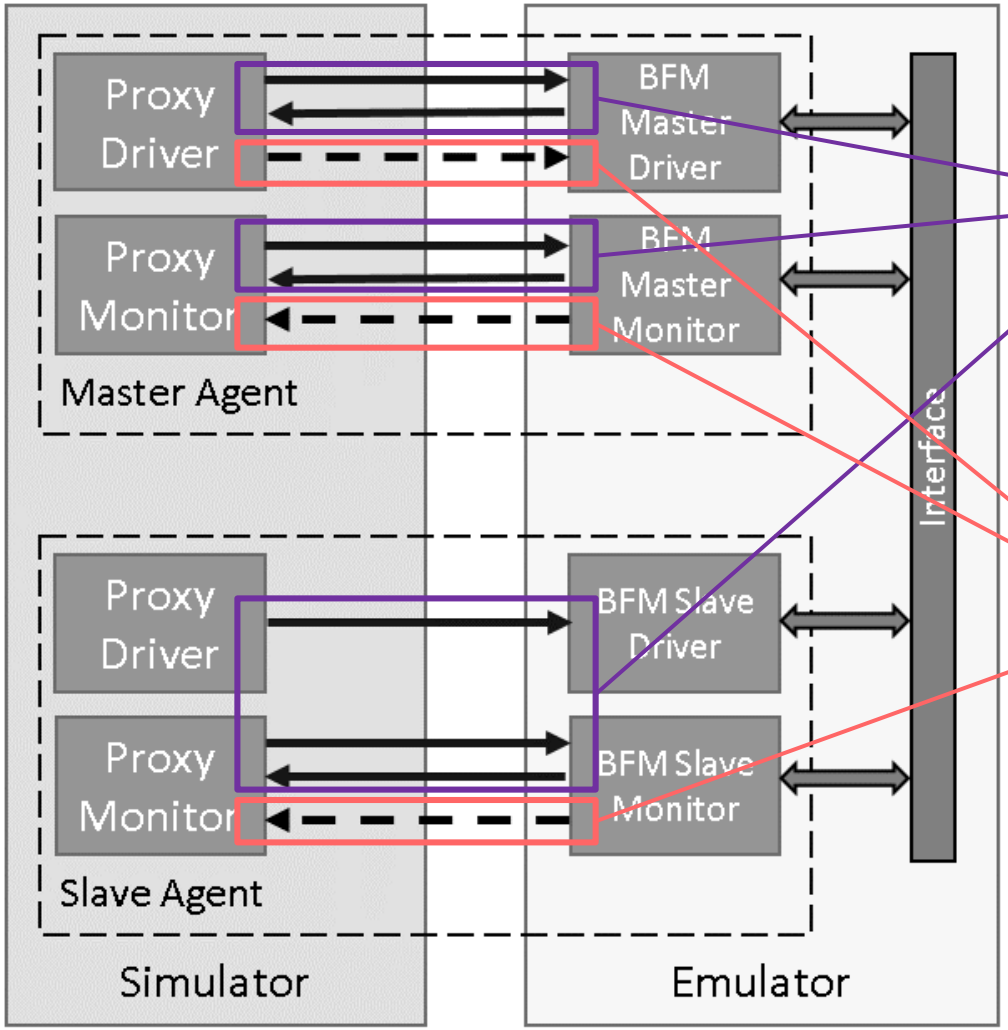
- Implement complex functionality that is utilized occasionally as a method in the C membrane and import/invoke from the BFM.

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

Optimized Architecture



SCEMI Function Based Interface

- To configure BFM registers
- To update proxy about events in BFM

SCEMI Direct Memory Interface

- To write/read data from BFM memory

Pipes are removed

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Acceleration Concept
- Introduction to SCE-MI and SCE-MI Pipes
- Testbench Architecture and Existing Challenges
- Optimizing SCE-MI pipe usage
- SCE-MI Direct Memory and Function Based Interface
- Optimized Architecture
- Results and Conclusion

Results

- TBA Performance improvement

Implementation	TBA properties			
	Gate count	Bevals	HW-SW Sync	TBA Time
Pipe only	~2 M	18,299,173	4,728,998	~60 min
Pipes only (Optimized)	~2 M	2,859,320	1,284,765	~35 min
DMI + Function based Interface	~1.5 M	218	9,871	~4 min

- Simulation v/s TBA run-time comparison for usecase

Simulation Time	TBA Time
~360min	~8min

Conclusion

- SCE-MI has multiple interfaces for different use cases
  - SCE-MI DMI used for backdoor read/write operation
  - SCE-MI Function Based Interface used to write into registers
  - SCE-MI Pipes are intended for streaming, variable length messaging, etc.

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

Conclusion

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

- HW-SW syncs generated using vendor proprietary implementation
  - Number of syncs are kept to minimum.

Future Work

- Porting to SOC environment and analyze the performance
- Explore UVM registers support decribed in SCEMI v2.3

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Acknowledgements

- David Brownell, ADI
- Anilkumar T. S, Cadence
- Dr. Hans van der Schoot, Mentor Graphics

Questions