2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Challenges and Mitigations of Porting a UVM Testbench from Simulation to Transaction-Based Acceleration (Co-Emulation)

Vikas Billa, Sundar Haran
Microsemi India Pvt. Ltd, Hyderabad, India.
vikas.billa@microsemi.com, Sundararajan.Haran@microsemi.com

Microsemi
Power Matters.™

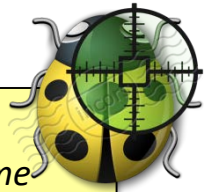
2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Agenda

- Problem Statement
- Simulation challenges & Proposed Solution
- Introduction to Emulation
- Two-Top TB Architecture
- A Case Study
  - Adopting Emulation
  - Behavioral Models
  - Verification IP’s
- Coding Guidelines
- Conclusion

Increasing Verification Pressures

Run time



Debug time

competition

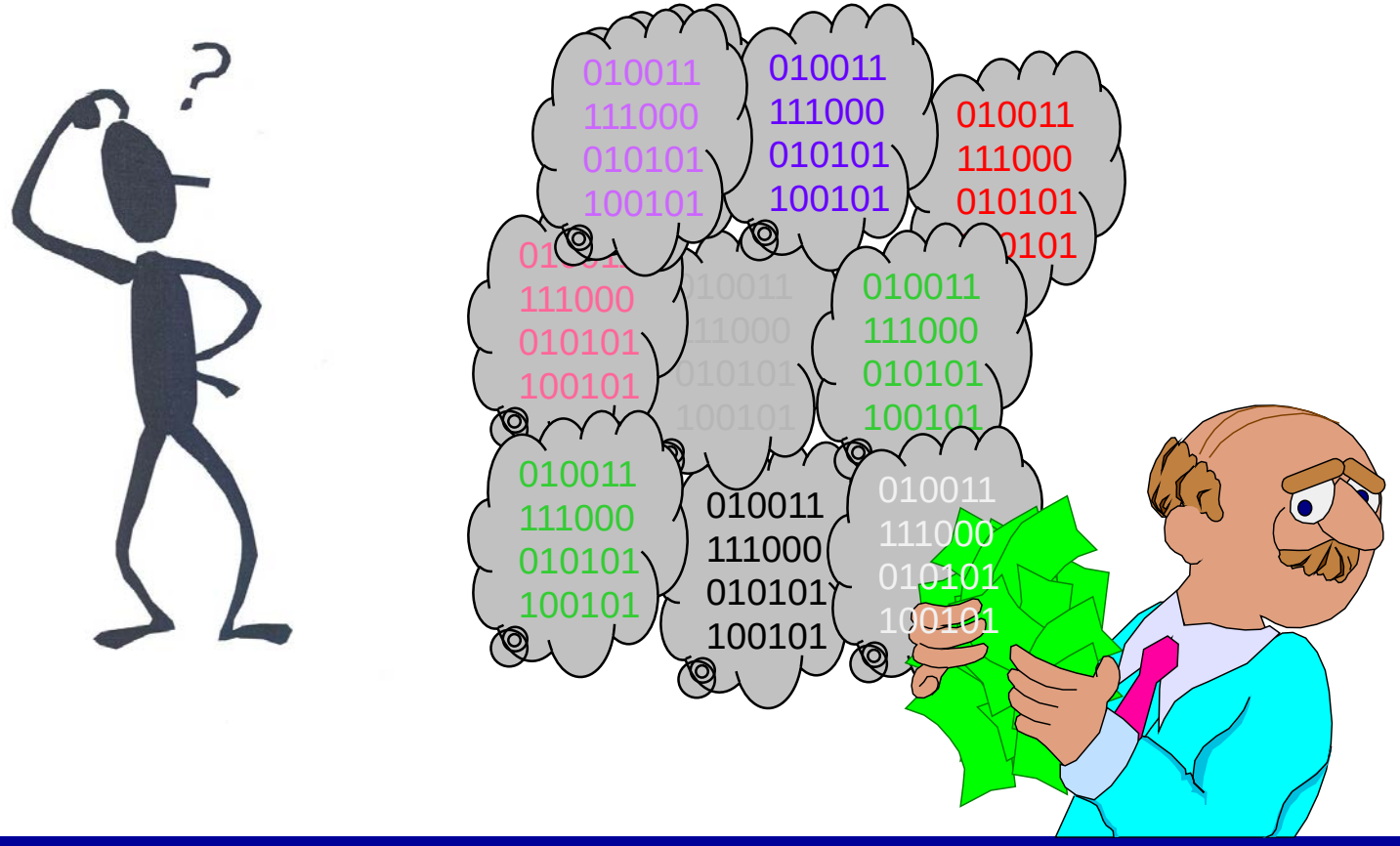
Time to Market

Complexity

Bugs

Problem Statement

- Problem: Old methods are no longer adequate
  - Growing SoC designs are pushing the limits of massive system level scenarios in simulation platform

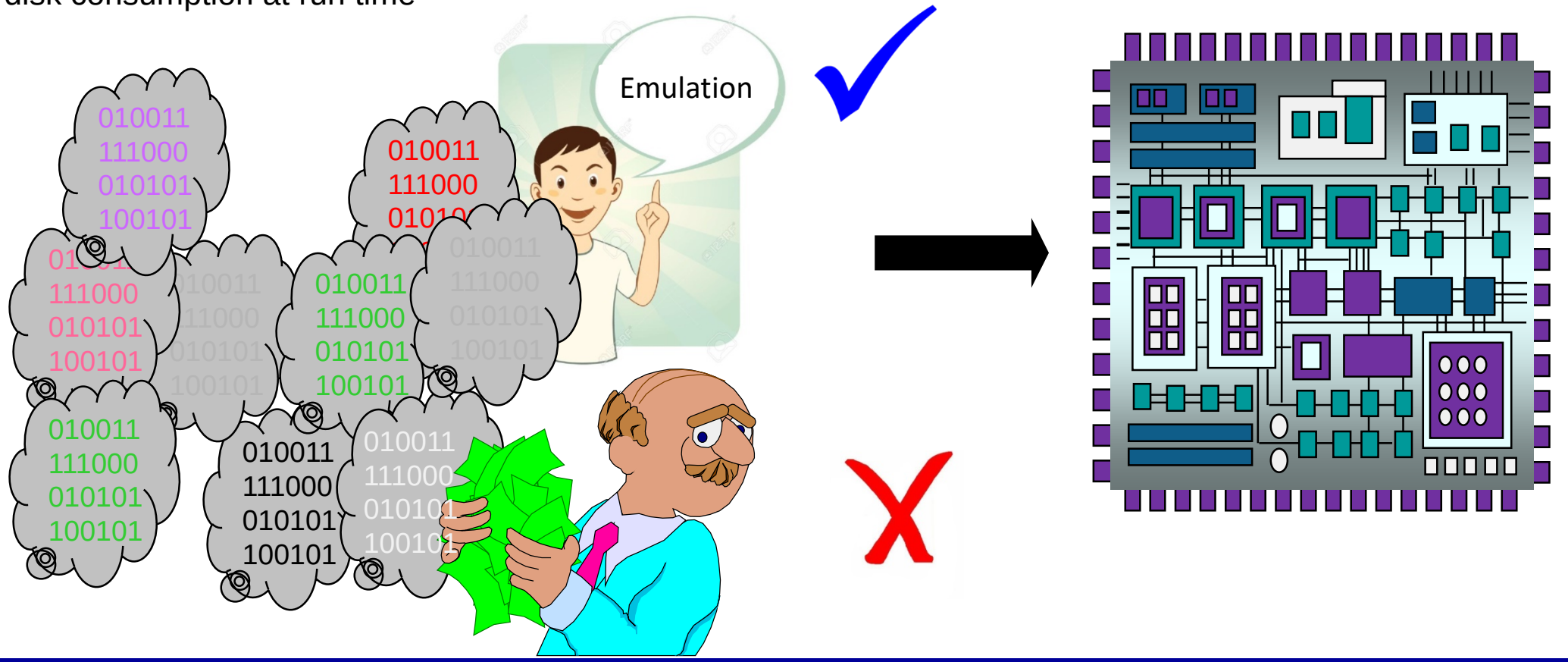


Simulation Challenges

- Adopting the UVM does not address the other verification needs
  - such as the ability to run, debug, and collect metrics for a large number of tests in a short amount of time
- The speed of the simulation is the primary bottleneck
- Limiting the number of simulation tests to meet requirements of tight schedules is alarming and raises doubt about the completeness of verification
- How much Disk usage can we afford for longer runs with bigger complex designs?

Proposed Solution

- Solution: The remedy for ever increasing simulation times is using emulation techniques
  - Its need of the hour, that verification experts port their complex testbench and DUT to Emulation platforms
  - This could be the way going forward in exercising system level scenarios that might require longer simulation runtime and huge disk consumption at run time



Introduction to Emulation

- Today's traditional verification flow involves verification at multiple abstraction levels
- Simulation offers a great springiness in debugging and the emulation offers mammoth performance gains
- An ideal solution is to make use of these offerings
  - develop a single, unified testbench which helps in enhancing the productivity and faster verification closure.
- In this paper, we will discuss a case study based on one of our native UVM testbench
- We partitioned the testbench into two top architecture that can be used not only for software simulation, but also used for hardware acceleration/emulation

Choice of Tool – Mentor's Veloce TBX

In this paper, we have used Mentor's Veloce TBX solution to develop emulation ready testbench. To create a unified testbench for both simulation and emulation we need to adhere to the below steps

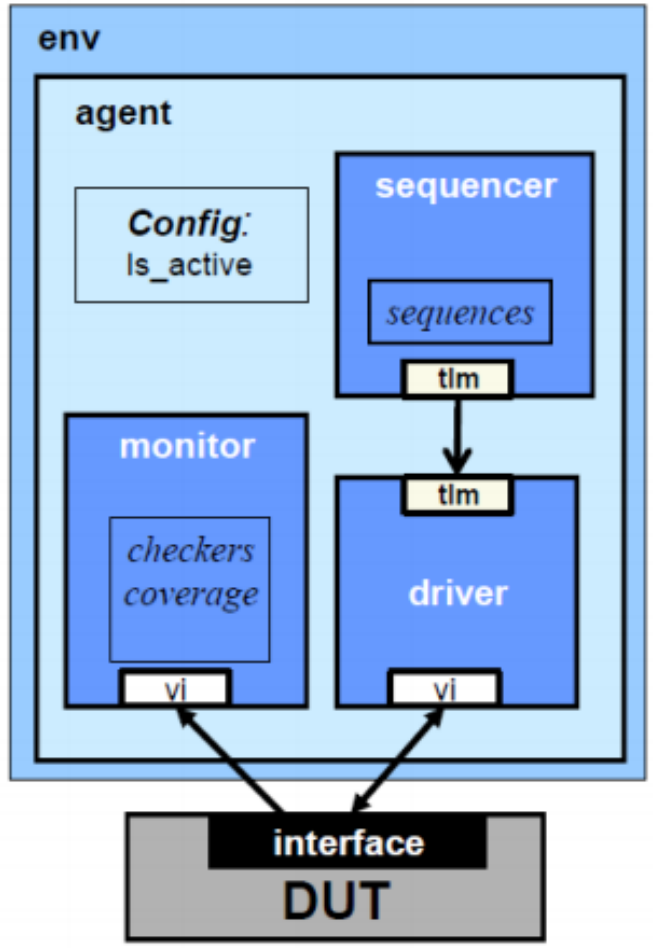
1. Employing two separate domains (two top architecture): an untimed hardware verification language (HVL-TOP) domain and a synthesizable hardware description language (HDL-TOP) domain

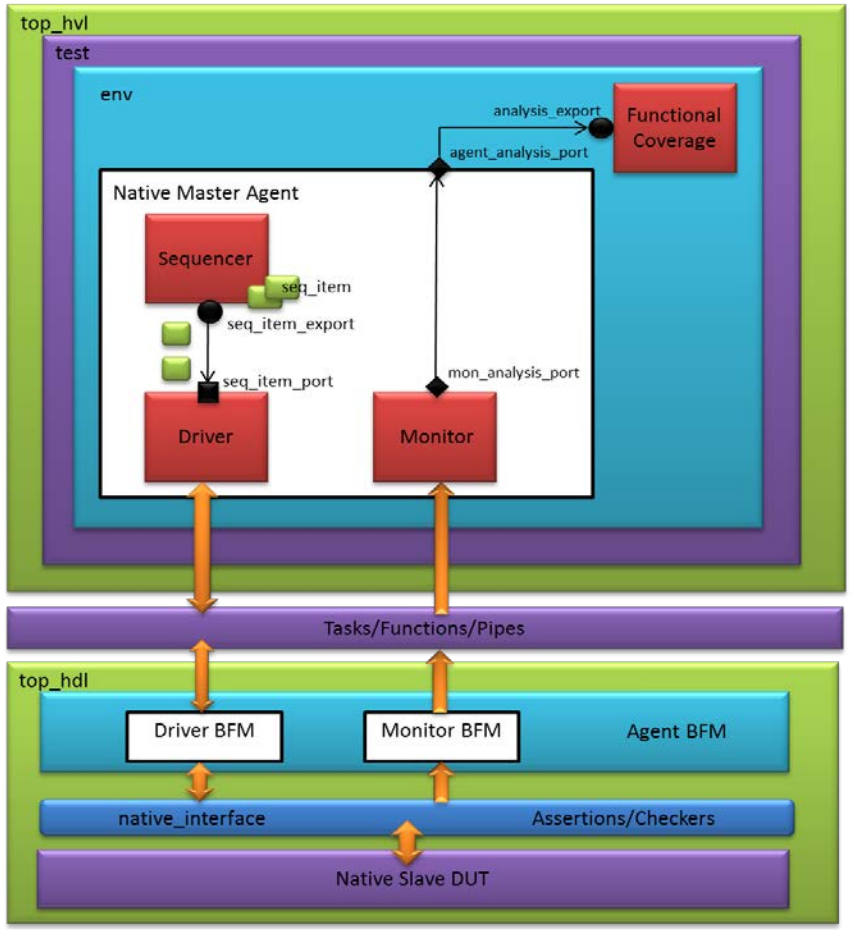
2. Modeling all the timed testbench code for emulator synthesis in the HDL domain (BFM), leaving the HVL domain untimed (proxy)

3. A transaction-level, probably an interface task/function or a pipe based approach to be used as a communication API between the HVL and HDL domains

Two-Top TB Architecture

- HVL and HDL top-level module hierarchies
- The HDL domain must be synthesizable
- The HVL domain contains non-synthesizable code
- The communication should be from either way i.e. from HDL to HVL or HVL to HDL

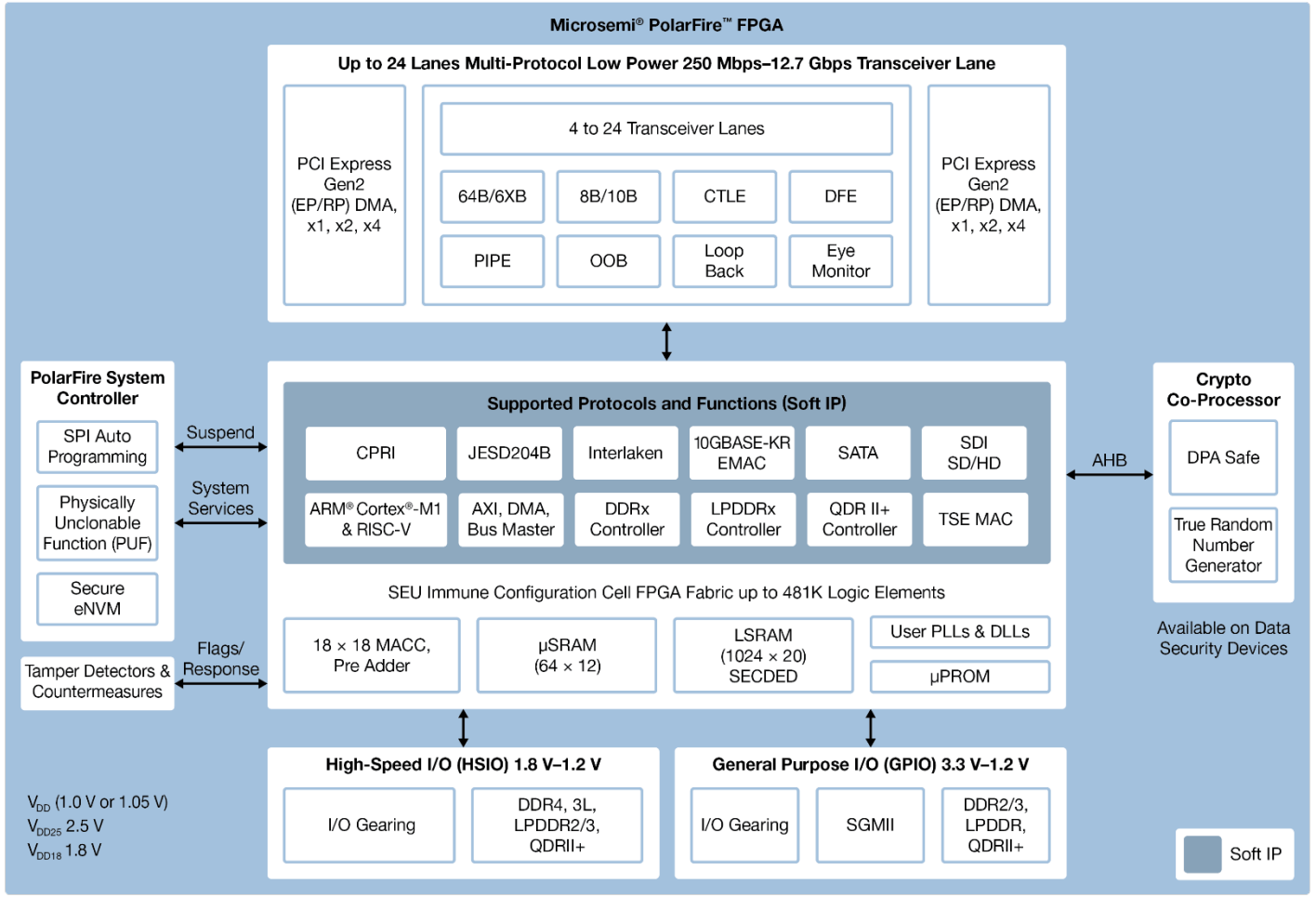




A Case Study

- This paper is a collective case study of PolarFire project
- PolarFire Overview
  - Microsemi's lowest power, cost-optimized mid-range PolarFire FPGA
  - The PolarFire FPGA family spans from 100K logic elements (LEs) to 500K LEs, and offers up to 50% lower power than competing mid-range FPGAs.
  - Applications within wireline access networks and cellular infrastructure, defense and commercial aviation markets, as well as industrial automation and IoT markets
  - An FPGA with an ARM Cortex M3 Processor and programmable analog, offering full customization, IP protection and ease-of-use
- Subsystems Identified for Porting highlighted in this case study

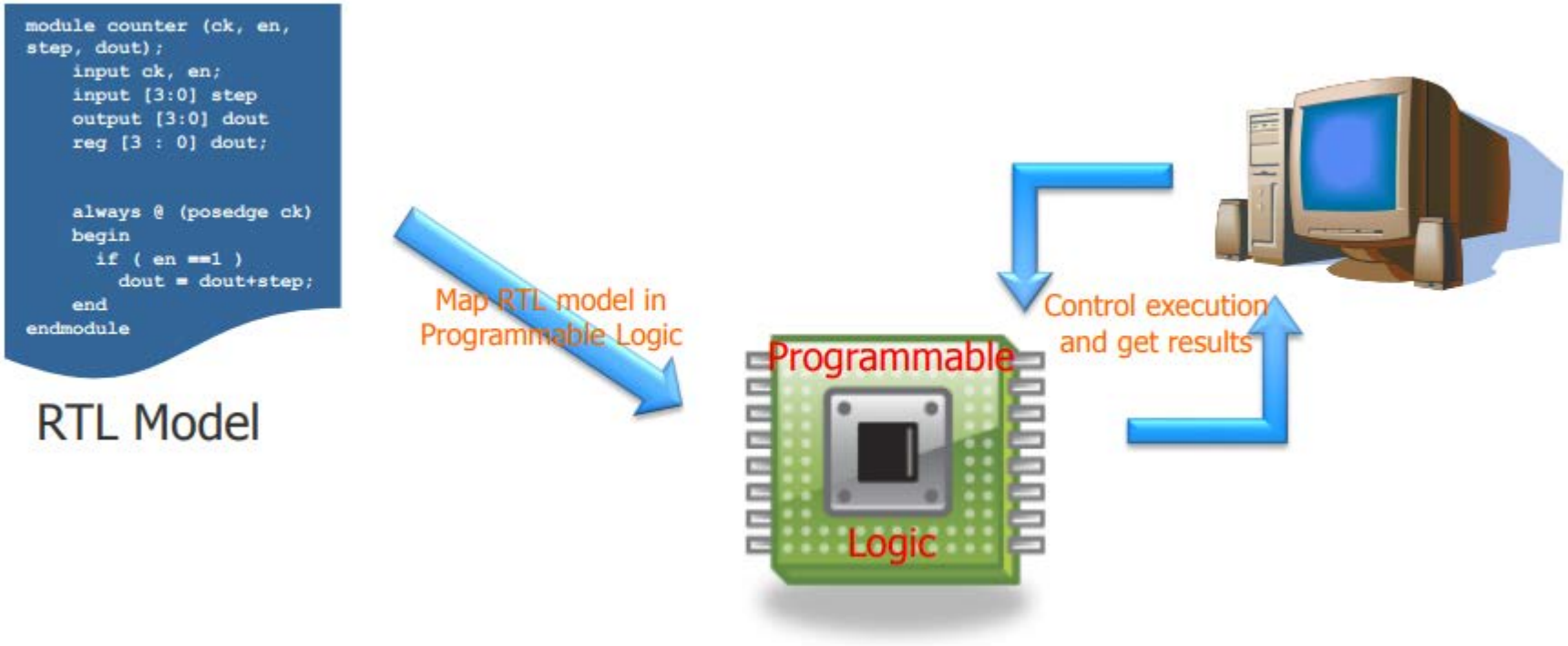
Microsemi PolarFire Architecture



Reference : https://www.microsemi.com/products/fpga-soc/fpga/polarfire-fpga

Adopting Emulation

- To overcome the simulation limits as mentioned in simulation challenges section, we have decided to have a emulation ready verification environment for our PolarFire project



Reference : https://indico.cern.ch/event/305730/contributions/703215/attachments/581425/800387/Veloce_Emulator.pdf

Behavioral Models - Porting Challenges

- We have ported around 78 models to emulation platform which was a huge effort
- Few changes in the HDL models are mentioned below which came across while converting it to synthesizable models
  - #delays are not supported in Veloce; replaced with @ posedge clk or @ negedge clk by using the internal clock generators
  - Converted real datatype to integer datatype
  - Removed tranif0 and tranif1 as is not supported in Veloce
  - .vams files are recoded to .v files

Behavioral Models Porting – Guideline 1

- As tranif0 and tranif1 is not supported in Veloce, the logic has been replicated using assign statement in the emulation model

Simulation Model

```
module fabric_model();

//code not shown

  for (i=0; i<80; i++) begin
  tranif0 t0 (x_blnl[i],x_gbl[i
x_bln_gbl_sell_b);
  end

endmodule
```

tranifx is not synthesizable

Emulation Model

```
module fabric_model();

//code not shown

  for (i=0; i<80; i++) begin
   assign x_blnl[i] =
(x_bln_gbl_sell_b  === 1'b0) ?
x_gbl[i]  : 'bx;

  end

endmodule
```

Behavioral Models Porting – Guideline 2

- In Veloce we need to pass seed as $random (my_seed), seed value can be assigned in the module declarations or else can be passed through command line using $value$plusargs as shown below

Simulation Model

```
module c_model();
 //code not shown
 always@(posedge
clk)begin
 addr=$random();
end
 endmodule
```

Emulation Model

```
module c_model();
 //code not shown
int myseed = 10;
 always@(posedge
clk)begin
 //addr=$random();
  addr=$random(my_seed);
end
```

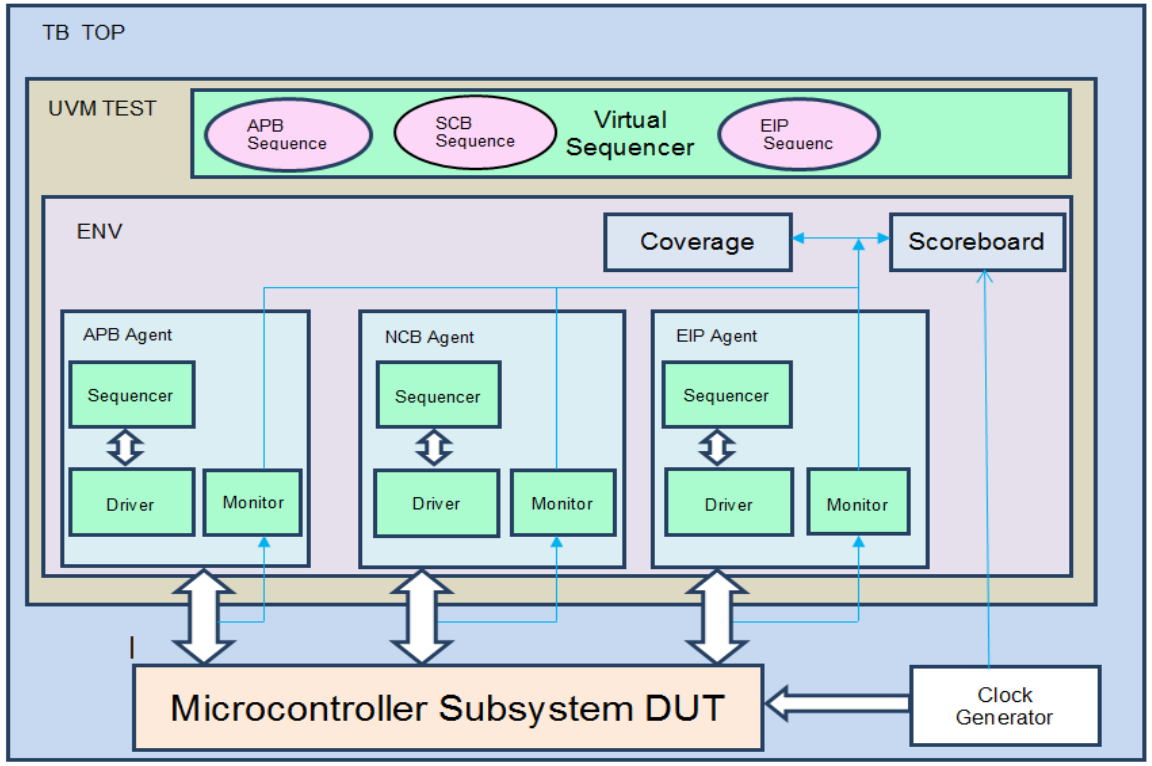
Pass seed value

```
if($value$plusargs ("RANDOM_SEED=%d ", my_seed))
begin
                my_seed=seed;
end
```

make all +RANDOM_SEED=200

Verification IP's

- The Microcontroller subsystem shown in figure 5 has 12 IP's out of which 10 are native protocol IP's and 2 are general protocol
- For Generic Protocol VIP's Mentor Graphics has provided Veloce Transactor Library (VTL)



Microcontroller Subsystem Simulation Verification Environment

Verification IP - Porting Challenges

- The main challenge here is to port all the existing Simulation VIP's to emulation ready VIP's in a specified time
- Before actual porting the authors have few followed the below steps
  1. Initially the authors went through the Veloce user guide
  2. Ported a native protocol simulation VIP to emulation Platform
     - ✓ Went through numerous phases in understanding the two-top architecture in practical
     - ✓ It took us several debug cycles to bring up the initial version of emulation ready VIP
     - ✓ All the best practices which we found during the porting are mentioned in coding guidelines section

Major changes in Porting - Driver

- The time consuming tasks are placed in the HDL driver and can be called by the HVL Driver as shown below

```
class apb4_master_driver extends uvm_driver
#(apb4_transaction_c);
    virtual apb4_master_driver_bfm BFM;
task run_phase(uvm_phase phase);
        pkt_t req,rsp;
        forever begin
            apb4_master_seq_item_s req_s, rsp_s;
            seq_item_port.get_next_item(req);
apb4_master_seq_item_converter::from_class(req,
req_s);
            BFM.drive_data(req_s, rsp_s);
           apb4_master_seq_item_converter::to_cla
ss(rsp, rsp_s);
            $cast(rsp, r    clone());
            rsp.set_id_info(req);
            seq_item_port.item    ne(rsp);
        end// !forever begin
    endtask : get_and_drive
endclass : apb4_master_driver
```

```
interface apb4_master_driver_bfm
(apb4_interface APB);
//pragma attribute apb4_master_driver_bfm
partition_interface_xif
    string tID;

    import
apb4_master_shared_pkg::apb4_master_seq_item_s;

task drive_data(apb4_master_seq_item_s req,
output apb4_master_seq_item_s rsp ); // pragma
tbx xtf

    @(posedge APB.PCLK);
// code not shown here
endtask: drive_data

endinterface: apb4_master_driver_bfm
```

HVL Driver

HDL Driver

Calling drive_data task from BFM

Major changes in Porting - Monitor

- completeness of the transaction the response need to be send back to HVL from HDL
  - happening through the proxy.write(item) method called in HDL and it is implemented in the HVL

```
class apb4_master_monitor extends uvm_monitor;

    virtual apb4_master_monitor_bfm BFM;
    uvm_analysis_port #(item_t) sb_post;

    task run_phase(uvm_phase phase);
        forever begin
          BFM.collect_data();
        end
    endtask : run_phase

     function void write(apb4_master_seq_item_s
item_s);
        item_t item;
apb4_master_seq_item_converter:: to_class(item,
item_s);
        this.item.copy(item);
        sb_post.write(this.item);
    endfunction: write
endclass : apb4_master_monitor
```

Proxy

HVL Monitor

```
interface apb4_master_monitor_bfm
(apb4_interface APB);
// pragma attribute apb4_master_monitor_bfm
partition_interface_xif
     import
apb4_master_shared_pkg::apb4_master_seq_item
    import apb4_agent_pkg::apb4_master_monitor;
        apb4_master_monitor proxy; // pragma
tbx oneway proxy.write
 task collect_data(); // pragma tbx xtf
         apb4_master_seq_item_s item;
         @(posedge APB.PCLK);

        //code not shown here

        proxy.write(item);

endtask : collect_data
 endinterface : apb4_master_monitor_bfm
```

HDL Monitor

Back Pointer

Major changes in Porting - Top

- In the hvl_top we have used only the run_test() and in the hdl_top the interface, monitor_bfm, driver_bfm are instantiated and are set using uvm_config_db

HVL top

```
module top_tb();

import uvm_pkg::*;
`include "uvm_macros.svh"

import apb4_agent_pkg::*;

`include "apb4_master_demo_tb.sv"
`include "apb4_master_test_lib.sv"

initial
begin
    $timeformat( -9, 3, " ns", 12);
    run_test();
end

endmodule : top_tb
```

HDL top

```
module top_hdl();
logic PCLK;
logic PRESETn;
 apb4_interface APB(PCLK, PRESETn);  // APB
interface
 // tbx vif_binding_block
initial begin
  import uvm_pkg::uvm_config_db;
  uvm_config_db #(virtual
apb4_interface)::set(null, "uvm_test_top",
$psprintf("%m.APB") , APB);
end
 apb4_master_monitor_bfm
APB_MONITOR(APB.apb4_mon_mp);
apb4_master_driver_bfm  APB_DRIVER
(APB.apb4_mp);

endmodule: top_hdl
```

VIP Porting – Guideline 1: fork join

- To achieve parallel process in SystemVerilog we use fork join construct, but this fork join is not synthesizable in HDL.

```
task run_phase(uvm_phase phase);
  box_transaction_c req;
 box_transaction_c rsp;
forever begin
    box_master_seq_item_s req_s, rsp_s;
    seq_item_port.get_next_item(req);
    if(req.fab_box_read == 1) begin
                fork
                    BFM.read_data(req_s);
BFM.read_capture_data(req_s, rsp_s);
                join
            end
     else begin

`uvm_info(tID,$sformatf("Inval
Combination"),UVM_MEDIUM)
                      end
       // code not shown here
        end// !forever begin
   endtask : run_phase
```

```
interface box_master_driver_bfm (box_interface
BOX);
  import
box_master_shared_pkg::box_master_seq_item_s;

 task read_data(box_master_seq_item_s req); //
pragma tbx xtf
  // code not shown
endtask:: read_data



task read_capture_data(box_master_seq_item_s
req, output box_master_seq_item_s rsp);
    // code not shown
endtask: read_capture


nterface: box_master_driver_bfm
```

HVL Driver

HDL Driver

read_data will send the control signals and read_capture_data will wait for read_Status signal to assert.

VIP Porting – Guideline 2: Configuration

- Care should be taken in HDL while we are configuring the agent as UVM_PASSIVE
- The key element is to enable the HDL drive_data task logic only if the is_active configuration is UVM_ACTIVE

```
 interface box_master_driver_bfm (box_interface BOX);
  import box_master_shared_pkg::box_master_seq_item_s;
  import box_master_shared_pkg::box_master_config_item_s;

 task  drive_data(box_master_seq_item_s req,
box_master_config_item_s req_config,output
box_master_seq_item_s rsp); // pragma tbx xtf
        @(posedge BOX.fab_box_clk)
       if(req_config.is_active == 1'b1) begin // UVM_ACTIVE
        // code not shown here
      end
endtask: write_data

endinterface: box_master_driver_bfm
```

Checking if is_active is UVM_ACTIVE or UVM_PASSIVE

VIP Porting – Guideline 3: $urandom_range

- $urandom_range(MIN,MAX) construct is not synthesizable in HDL domain

```
module random();
logic CLK;
bit cnt=1;
int unsigned seed, my_seed;
bit [2:0] addr;
int unsigned MAX=6, MIN=2;

initial begin
 if($value$plusargs ("RANDOM_SEED=%d ", my_seed))
begin
            my_seed=seed;
  end
end
always@(posedge CLK) begin
    addr=$random(my_seed);
     addr= MIN+(addr %(MAX-MIN));
     $display( "Random Range addr=%0d", addr   );
end
// clock generator
end
endmodule
```

make all +RANDOM_SEED=200

addr is randomized and then MIN and MAX selections are used

VIP Porting – Guideline 4: $display

- For runtime controllability, use test_plusargs/value_plusargs as shown below

```
`define HDL_UVM_NONE      0
`define HDL_UVM_LOW       1
`define HDL_UVM_MEDIUM    2
`define HDL_UVM_HIGH      3
`define HDL_UVM_DEBUG     4
int glbl_verbose;
`define VEL_INFO(strID="", msg="",verbosity) \
if($test$plusargs("HDL_UVM_DEBUG")) \
   glbl_verbose = 4; \
if($test$plusargs("HDL_UVM_HIGH")) \
   glbl_verbose = 3; \
if($test$plusargs("HDL_UVM_MEDIUM")) \
   glbl_verbose = 2; \
if($test$plusargs("HDL_UVM_LOW")) \
   glbl_verbose = 1; \
if($test$plusargs("HDL_UVM_NONE")) \
   glbl_verbose = 0; \
if(glbl_verbose >= verbosity )  \
$display("UVM_INFO @ %0t : %m[%s] %s", $time, strID, msg);
```

Declared a VEL_INFO define

```
module counterud (CLK, CLR, UP_DOWN, Q);
input CLK, CLR, UP_DOWN; output [3:0] Q; reg    [3:0] tmp = 0;
  always @(posedge CLK)
   begin
    if (CLR) begin
      tmp = 4'b0000;
    end
    else
      if (UP_DOWN) begin
        tmp = tmp + 1'b1;
        `VEL_INFO("counter", "Incrementing", `HDL_UVM_LOW);
      end
      else begin
           tmp = tmp - 1'b1;
           `VEL_INFO("counter", "Decrementing", `HDL_UVM_HIGH);
         end
       end
 assign Q = tmp;
endmodule
```

Shown usage of VEL_INFO define

make all +HDL_UVM_MEDIUM

Conclusion

- In this paper, we have discussed on how to port a simulation environment to a emulation ready UVM framework by using Mentor Veloce TBX Flow
- The key highlights in porting are:
  - Two-Top TB architecture
  - Communication API between HDL and HVL and vice versa
- To summarize we have discussed on
  - Simulation Challenges
  - Porting from Simulation VIP to Emulation VIP
  - Coding Guidelines
  - Finally developed a unified testbench without conceding any of the UVM capabilities

Future Work

- The ported behavioral models and VIP's are tested at block level
- Currently working on integrating these models and VIP's
  - in to the top level verification environment
  - once this is done we are planning to do a performance analysis between pure simulation and emulation environments

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

References

1. UVM User Manual, uvmworld.org
2. UVM Cookbook - Emulation, Mentor Verification Academy
3. Standard Co-emulation Modeling Interface (SCEMI) Version 2.2, Accellera, January 2014
4. Master.pdf – veloce user guide
5. How to Boost Verification Productivity with SystemVerilog/UVM and Emulation, Hans van der Schoot, DVCon Europe 2015
6. STMicroelectronics: Simulation + Emulation = Verification Success
7. https://indico.cern.ch/event/305730/contributions/703215/attachments/581425/800387/Veloce_Emulator.pdf
8. https://www.microsemi.com/products/fpga-soc/fpga/polarfire-fpga
9. Microsemi Internal Specification Documents
10. Microsemi Internal Testbench Documents

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Acknowledgement

- We profoundly thank colleagues and management team at Microsemi India Pvt. Ltd, Hyderabad for their valuable support
- Special thanks to
  - ***Rohit Malyan***, Emulation Consultant, Mentor Graphics

Thanks to **DVCON** USA for giving us this opportunity

Microsemi
Power Matters.™

Questions