2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Interface Centric UVM Acceleration for Rapid SOC Verification

Jiwoong Kim, Yoona Lhim, Hyungjin Park, Hyunsun Ahn, Seonil Brian Choi
Samsung Electronics Co., LTD.


accellera SYSTEMS INITIATIVE

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

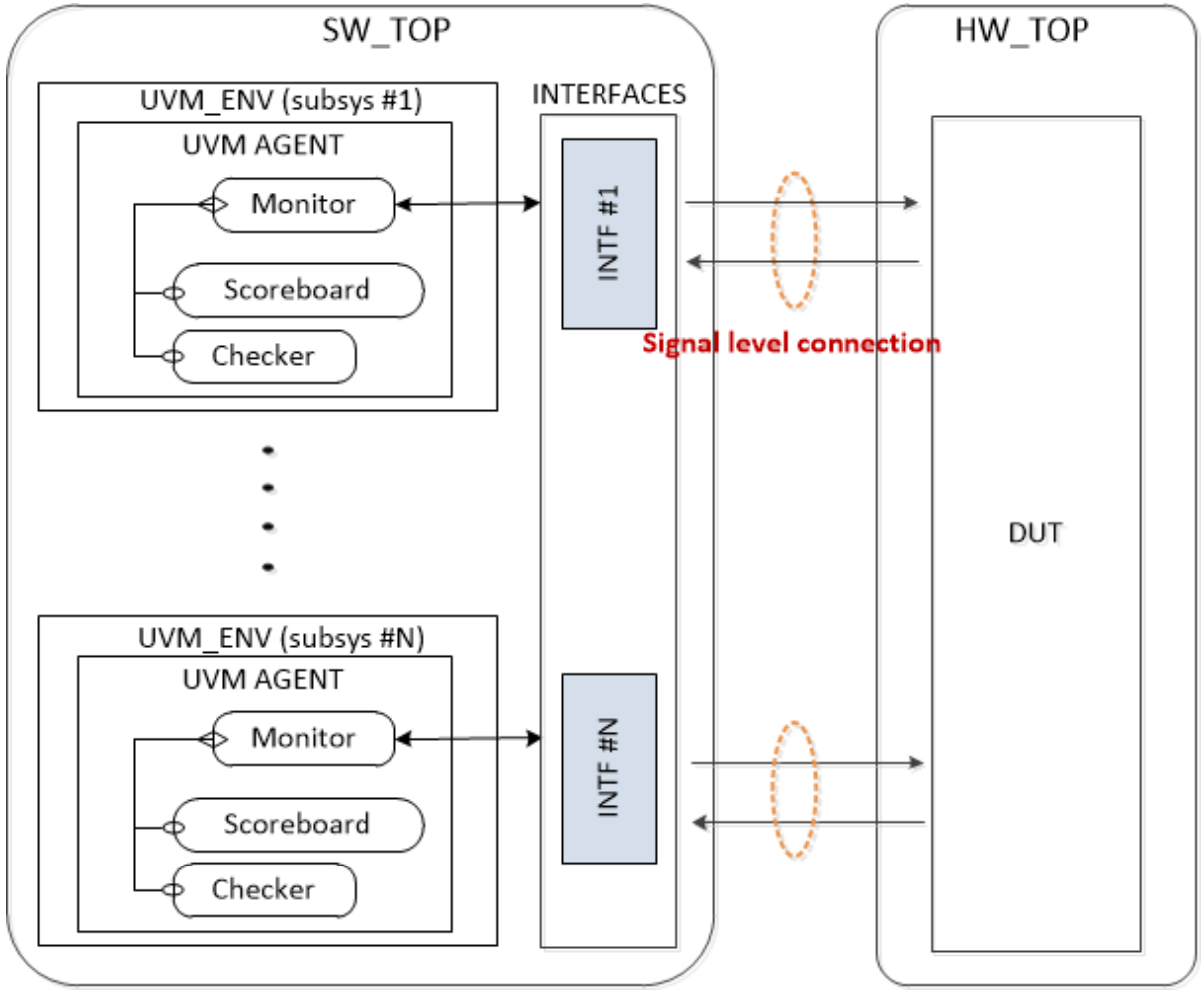
# Contents

- UVM environment for SOC verification
- HW/SW synchronization
- Interface centric UVM acceleration environment
- Interface implementation for UVMA
- Common Interface Library
- Speed-up for emulation runtime
- Examples
- Experimental results
- Conclusion

accellera SYSTEMS INITIATIVE

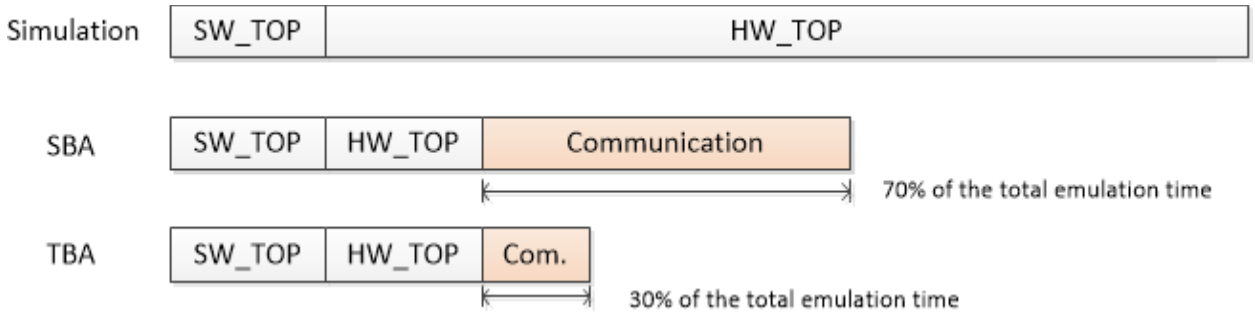
# UVM Environment for SOC Verification (SBA)

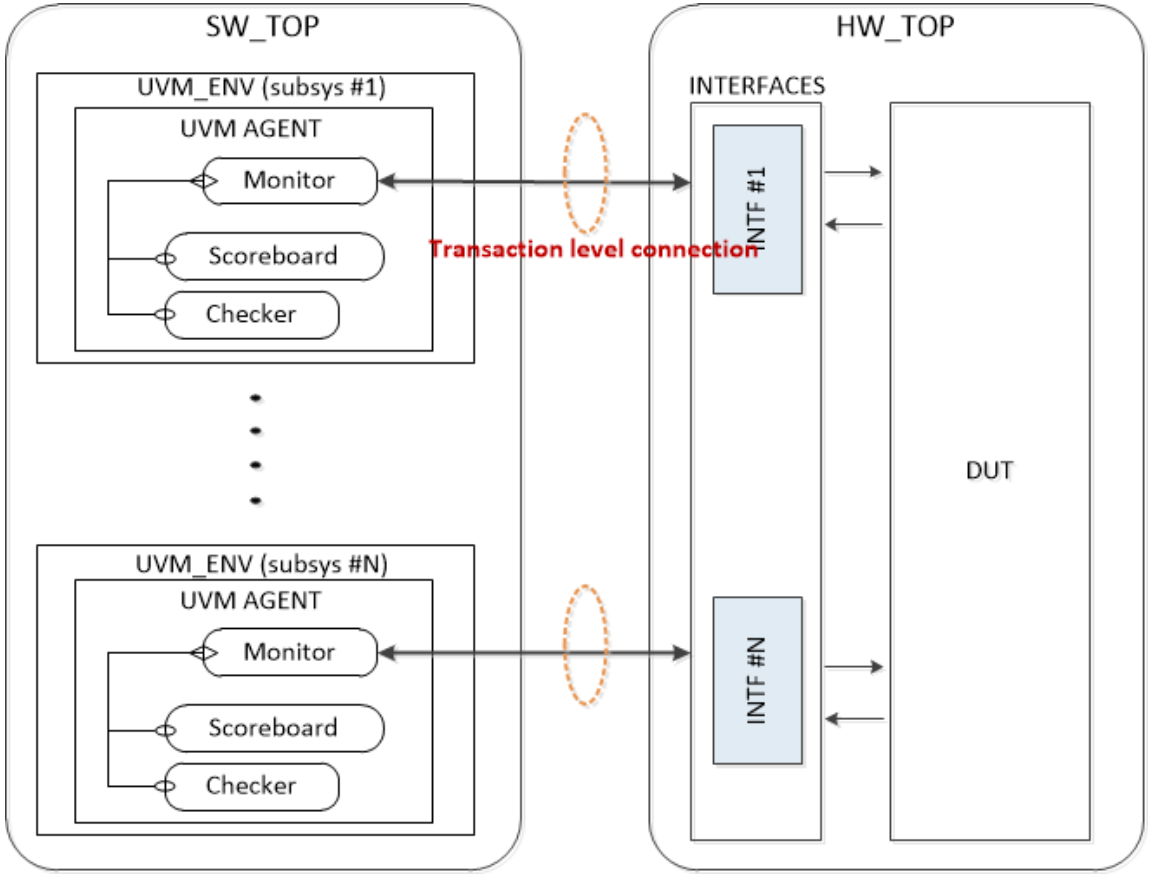
- Same as pure simulation environment
- SOC consists of various subsystems such as CPUs, peripherals and multimedia IPs.
- UVM_ENV of each subsystem has their own interface.
- Interfaces take small role.
  - UVM agents to DUT connection
- In emulation, HW_TOP and SW_TOP communicate at signal level.
  - Small acceleration gain



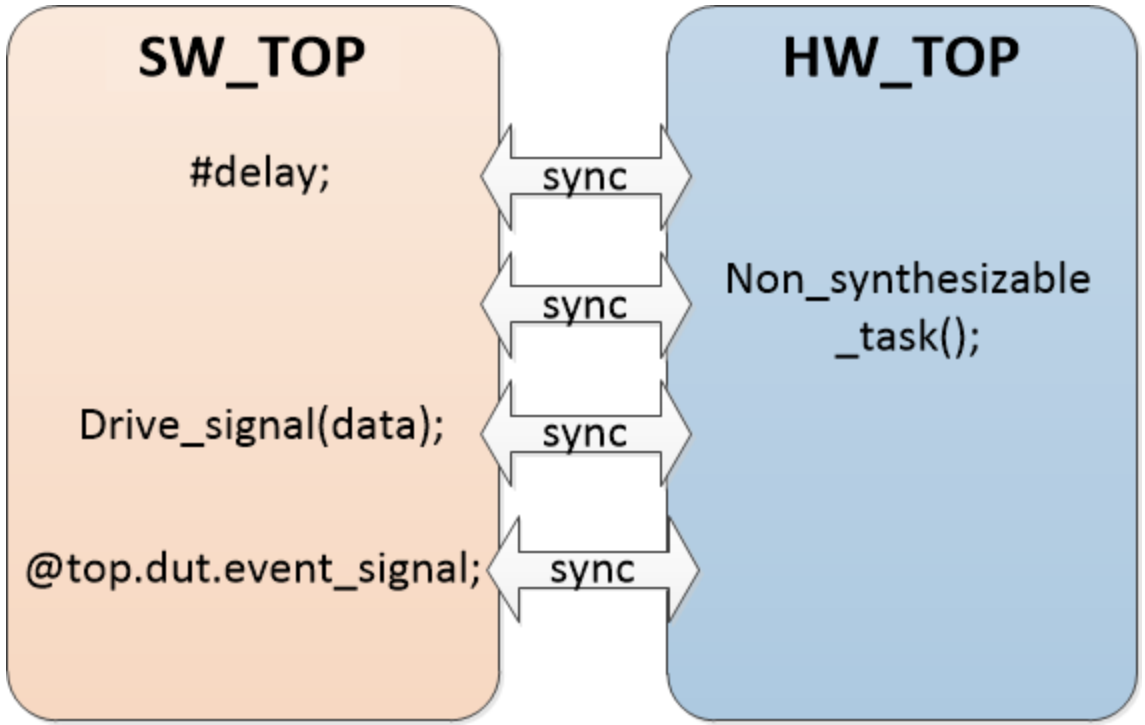
# UVM Environment for SOC Verification (TBA)

- Interface is moved to HW_TOP to remove signal level connections.
- Communications between HW_TOP and SW_TOP occur at transaction level.
  - Interfaces need additional role for transaction level communication.



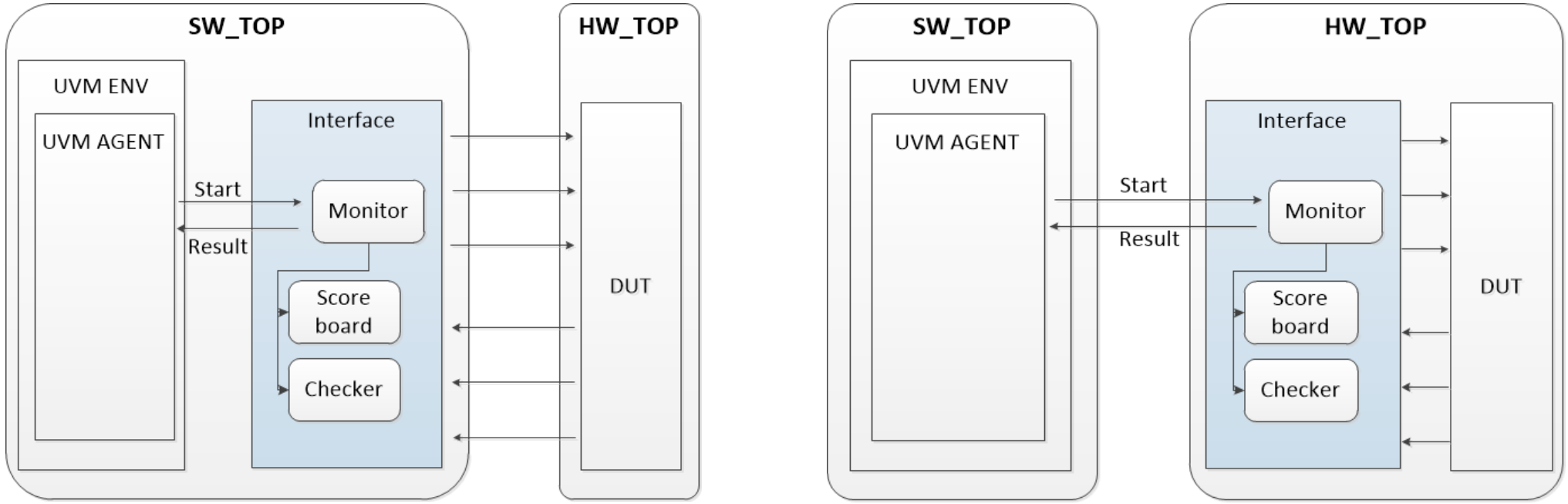



# HW/SW Synchronization




- HW/SW syncs is required to synchronize for correct functionality when HW_TOP and SW_TOP communicate.
- Emulator is stopped to synchronize.
- Frequent HW/SW syncs and large data transfer cause low performance.
- The worst thing that cause HW/SW sync is exported clock signal.

# Interface centric UVM Acceleration Evironment




- Verification components are implemented in the interface.
  - Interfaces play key role in the proposed environment.
- Interfaces can be moved to HW_TOP for emulation and SW_TOP for simulation.

# Interface Implementation for UVMA

**intf_inst.sv**

```
`include "subsystem1_intf_inst.sv
`include "subsystem2_intf_inst.sv
`include "subsystem3_intf_inst.sv
```

**subsystem1_intf_inst.sv**

```
Subsystem1_intf subsystem1_intf(
    .RESETn(HW_TOP.RESETn),
    .CLK(HW_TOP.CLOCK)
);
```

**HW_TOP**

```
`ifdef EMULATION
    `include intf_inst.sv
`endif
```

**SW_TOP**

```
`ifndef EMULATION
    `include intf_inst.sv
`endif

Initial begin
    // uvm_config_db::set for
subsystem's interfaces
end
```

**Subsystem_env can get virtual interface using uvm_config_db:: get method**

# Common Interface Library - 1

- Frequently used parts of the testbench that cause unnecessary HW/SW sync are implemented in a common interface library.

**Previous Test sequence**

```
repeat(cycle_delay) @(posedge aclk);
```

**This code cause error when the task is called concurrently.**

**Modified Test sequence**

```
vintf.aclk_ctrl.wait_posedge(cycle_delay);
```

**Interface of subsystem**

```
//Instantiation of WaitClock for aclk
waitClock aclk_ctrl (aclk);
```

**Common interface library**

```
interface waitClock(input clk);
    task wait_posedge (bit[31:0] n = 1);
        bit [31:0] i;

        for(i=0; i<n; i++) @(posedge clk);
    endtask
endinterface
```

# Common Interface Library - 2

- Tasks which need concurrent access should be treated carefully when using HW resources.

1. **When concurrent call of wait_posedge occurs, push the value of target counter (the end time of the task) into SW queue.**
2. **Called tasks compare target and current counter value when an event occurs. If current value matches target value, the task is terminated.**

1. **In HW part, the minimum value of target counter is set as the current target.**
2. **When counter value matches target value, event is triggered**

```
task automatic wait_posedge(input int n);
    int target_count, curr_cnt;
    set_current_target(n, curr_cnt);
    target_count = curr_cnt + n;
    sw_queue_push(target_count);
    do begin
        @(counter_event);
    end while(target_count != curr_event_cnt);
endtask
```

# Speed-up for Emulation Runtime

- Synthesizable verification components
  - Scoreboards, checkers and monitors operated in the SW_TOP are converted synthesizable and moved to the interface.
  - Verified RTL and Extended HW synthesizable syntax are used.

- HW/SW communication through FIFO
  - Non-synthesizable components communicate through FIFO to send and receive information in bulk.

# Examples (simple scoreboard) - 1

- TBA Implementation

UVM_MONITOR

```
fork
    forever begin
        vintf.get_input(input_data);
        input_port.write(input_data);
    end
    forever begin
        vintf.get_output(output_data);
        output_port.write(output_data);
    end
    begin
        vintf.wait_test_done();
    end
join_any
```

INTERFACE

```
task get_output(output out_item_t
item);
    bit done;
    done = 0;

    while(!done) begin
        @(posedge clk);
        if (out_hand_shake)
        begin
            item.data = out_data;
            item.info = out_info;
            done = 1;
        end
    end
endtask
```

# Examples (simple scoreboard) - 2

- Proposed Implementation

UVM_MONITOR

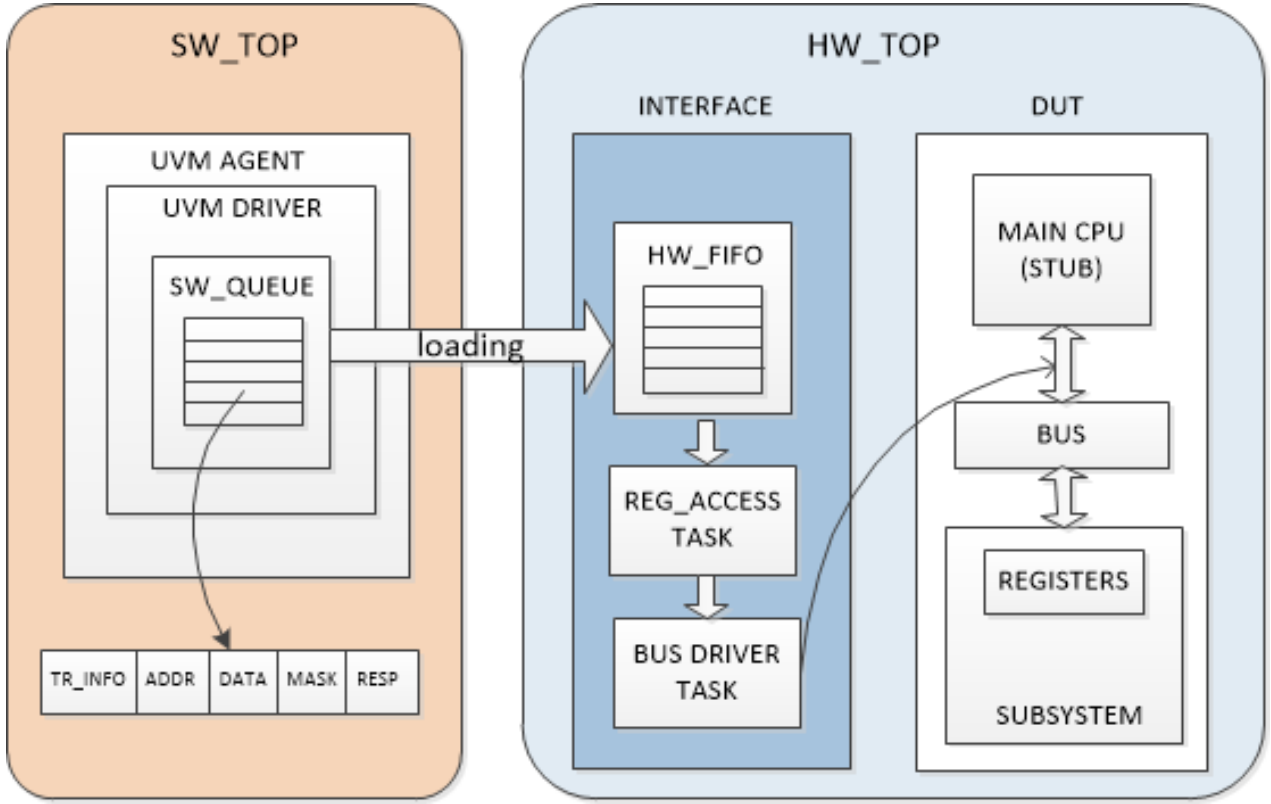
```
fork
    begin
        vintf.sb_output_data(result);
    end
join
```

- **Get data from input data FIFO and compare with output data**
- **If checking part is non-synthesizable, use FIFO to minimize HW/SW sync**

INTERFACE

```
always @ (posedge clk) begin
    if(in_hand_shake)
        in_fifo.push(in_data);
end
task sb_outdata(output result);
    out_item_t item;
    bit result_temp;
    result = 0;
    while(!test_done) begin
        @(posedge clk);
        if (out_hand_shake) begin
            item.data = out_data;
            item.info = info;
            check_data(item, result_temp);
            result = result | result_temp;
        end
    end
endtask
```

# Examples (register access) - 1




- Register access transactions generate significant amount of HW/SW sync in emulation.
- Register access commands are collected in the software queue.
- Commands are loaded in the hardware FIFO when loading conditions of the software queue are met.

Simulation code example of register access (polling)

```
do begin
    regA.read(status, read_data);
end while(read_data != expected_data);
```

# Examples (register access) - 2

- Example of proposed polling method

SW_TOP

Test sequence

```
regA.polling(expected_data, mask);
```

UVM DRIVER

```
if(trans.tr_type == REG_POLLING)
begin
    //make register polling command
    //push to sw_queue
    //move sw_queue data to HW FIFO
    //call vintf.reg_access()
end
```

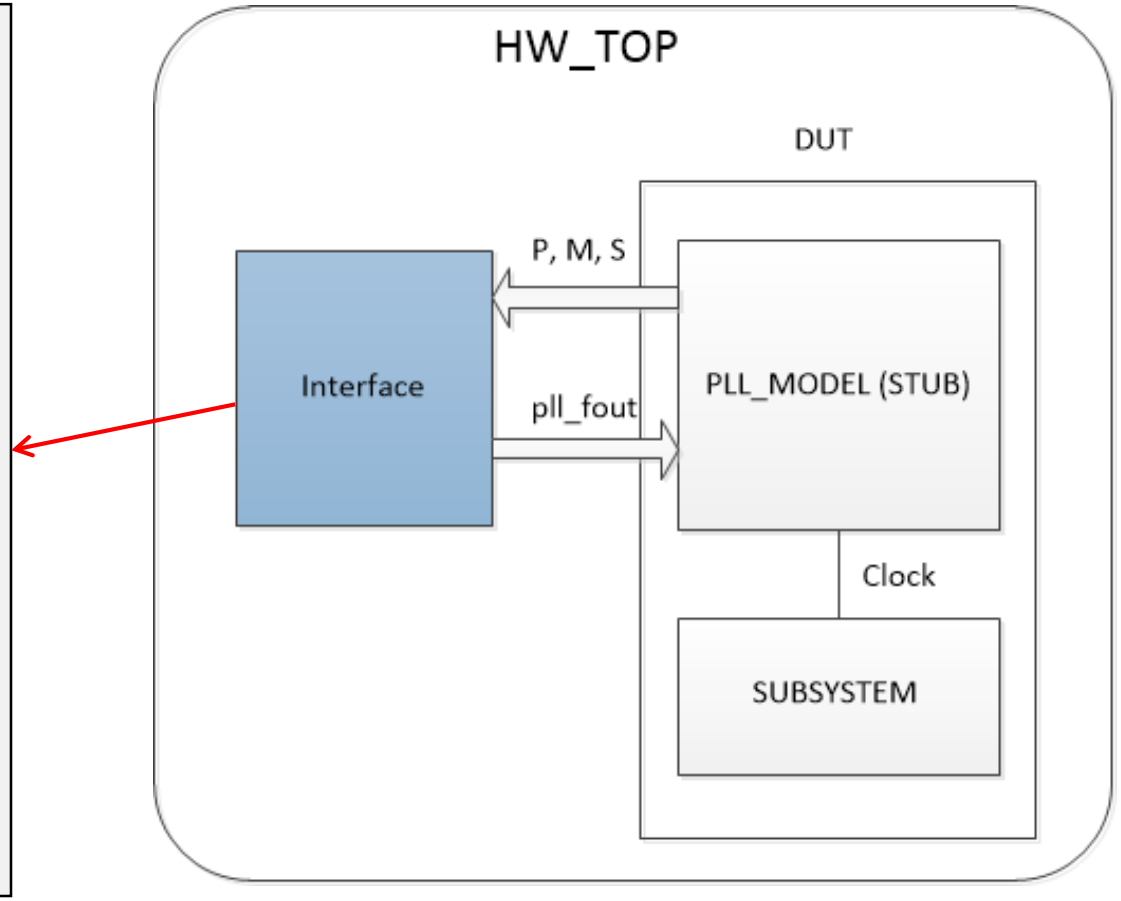
HW_TOP (interface)

```
task reg_access();
    while(hw_fifo.size != 0) begin
        hw_fifo.pop(cmd); //Read command
        if(cmd.tr_type == REG_POLLING)
            reg_polling(cmd);
        // The rest of code is not shown.
    end
endtask

task reg_polling(input tr_cmd_t cmd);
    do begin
        drive_read_req(cmd);
        wait_read_resp(r_data);
    end while(r_data&mask != cmd.data&mask);
endtask
```

# Examples (PLL modeling)

- Example of PLL modeling

```
initial begin
    pll_fout   = 1'b0;
    forever begin
        #(h_period) pll_fout = ~pll_fout;
    end
end

always @(posedge CLK or negedge RESETN) begin
    if (~RESETN) begin
        h_period <= 16'hFFFF;
    end
    else if(PMS_prev != PMS_curr) begin
        get_h_period(P, M, S, h_period);
    end
end
```



# Experimental Results

| Subsystem | #of scenarios | Runtime (Avg. min.) | | | Hardware runtime ratio in emulation (%) | |
|---|---|---|---|---|---|---|
| | | Simulation | Emulation | | | |
| | | | Previous | Proposed | Previous | Proposed |
| Image Codec | 170 | 450 | 23 | 14 | 68 | 90 |
| Digital Signal Processor | 161 | 570 | 28 | 18 | 69 | 93 |
| Neural Processor Unit | 188 | 620 | 31 | 19 | 65 | 91 |

- Experiments are conducted on our mobile SOC environment.
  - More than 500 verification scenarios including digital signal processor, image codec, neural processor unit which takes relatively long simulation time
- 20% improvements in speedup compared to previous work and more than 30x speedup in the runtime compared to our pure simulation.

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Conclusion

- To increase emulation speedup while reusing the existing UVM simulation environment, interface centric verification solution is proposed.
- 30x speedup and enables to verify complex scenarios that could not be done in pure simulation.
- Future work
  - Mixing and scheduling verification scenarios for given emulation and simulation resources

accellera SYSTEMS INITIATIVE