2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

# Problematic Bi-Directional Port Connections: How Well is Your Simulator Filling the UPF LRM Void?

Brandon Skaggs

Cypress Semiconductor, An Infineon Technologies Company

## Agenda

- Motivation and Contributions of this Paper
- Common Bi-directional Port Issues
- Common Bi-directional Port Modeling Approaches
- Test Setup / Scenarios
- Results
- Analysis & Concluding Remarks

## Motivation

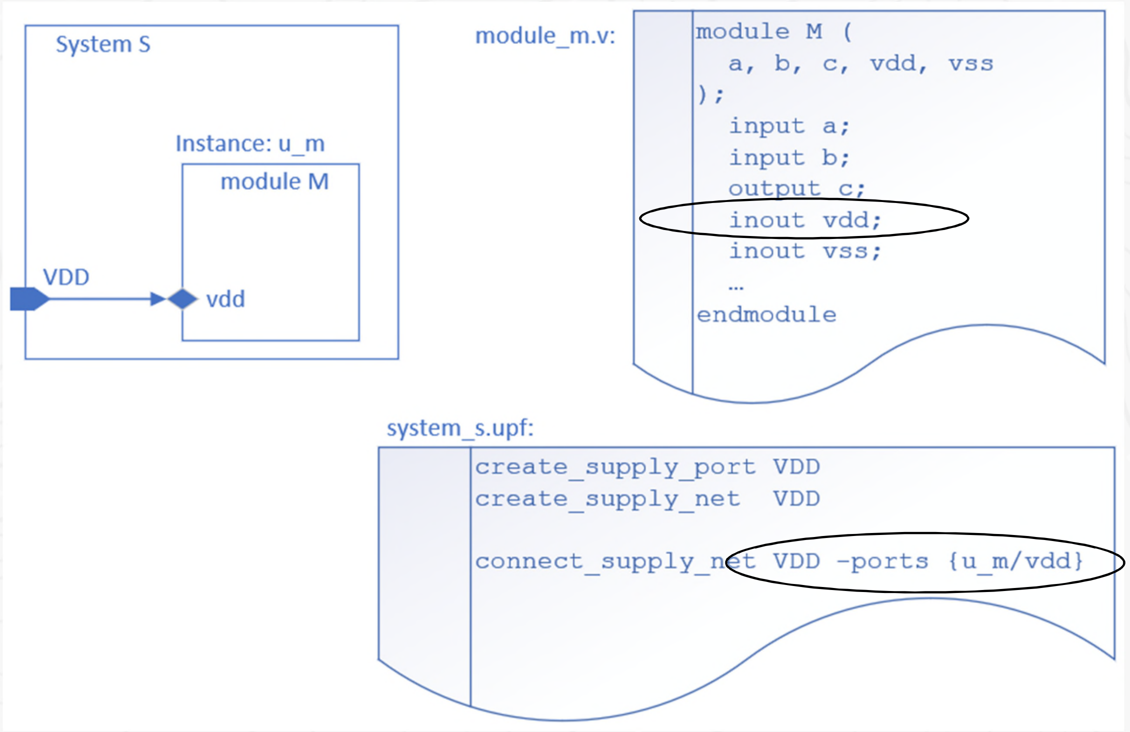
- The UPF LRM does not describe in detail the handling of bi-directional (inout) HDL port connections—leaving this open to interpretation of EDA tool vendors.
- How tools interpret proper behavior of bi-directional port semantics can have an impact on best practices for designing hierarchical power intent and writing effective models for simulation.
- An understanding of how contemporary commercial simulators handle these ports can inform best practices – and possibly suggest improvements in future IEEE 1801 revisions.

## Contribution

- This paper presents a systematic survey of the behavior of three contemporary power-aware digital simulators in handling bi-directional ports.
- This presentation will:
  - Discuss two common bi-directional supply port modeling scenarios.
  - Present three methods for modeling bi-directional connections—and review the support of each method by each simulator.
  - Present the problem statement and test scenarios under study
  - Review final results and concluding remarks—including possible enhancements to future UPF LRM revisions

## Common Bi-Directional Port Modeling Scenarios

## Scenario 1: Inout HDL to Supply Net

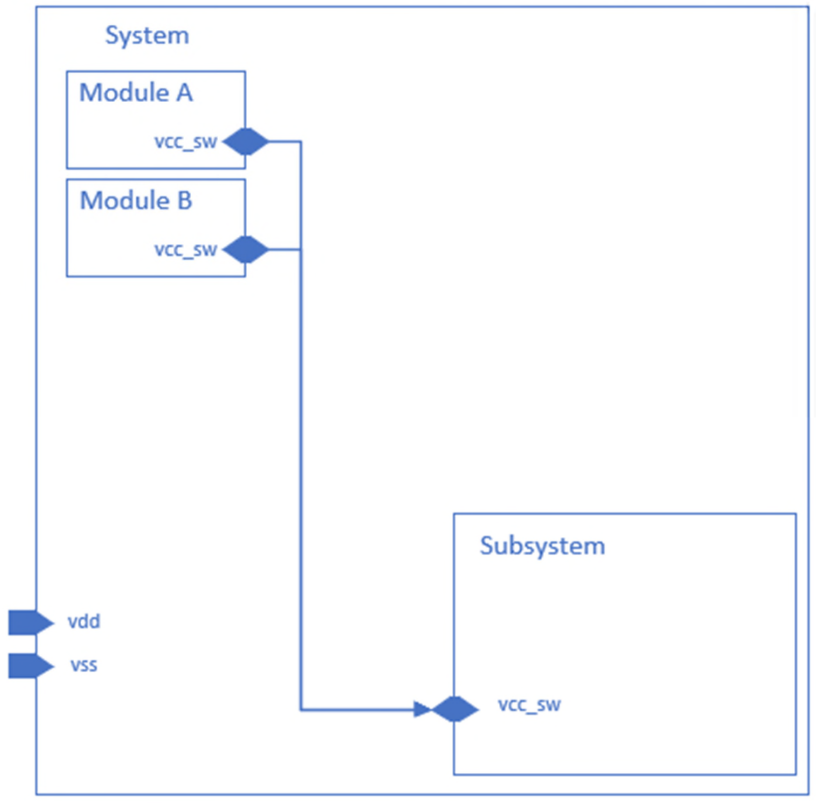


```
module M (
  a, b, c, vdd, vss
);
  input a;
  input b;
  output c;
  inout vdd;
  inout vss;
  …
endmodule
```

```
create_supply_port VDD
create_supply_net  VDD

connect_supply_net VDD -ports {u_m/vdd}
```

- Is module M being supplied by ‘vdd’?
- Or is module M supplying ‘vdd’ to the system?
- Per the UPF LRM, the existence of the ‘inout’ port implies a driver…

# Scenario 2: Hierarchical Supply Nets



system_s.upf:

```
create_supply_port vdd
create_supply_port vss

create_supply_net  vdd
create_supply_net  vss
create_supply_net  vcc_sw -resolve parallel

connect_supply_net vcc_sw -ports {u_ma/vcc_sw}
connect_supply_net vcc_sw -ports {u_mb/vcc_sw}
connect_supply_net vcc_sw -ports {u_ss/vcc_sw}
```

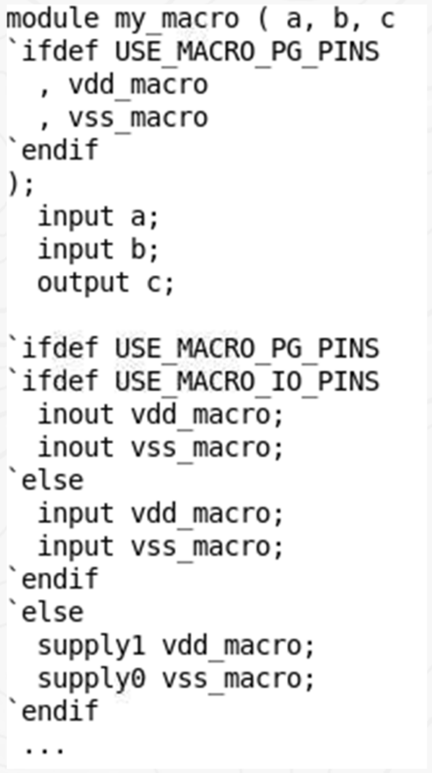
Not Supported?

subsystem_ss.upf:

```
create_supply_port vcc_sw -direction inout
create_supply_net  vcc_sw -resolve parallel
```

## Bi-Di Port Modeling Approaches

## Method 1: Compiler Directives



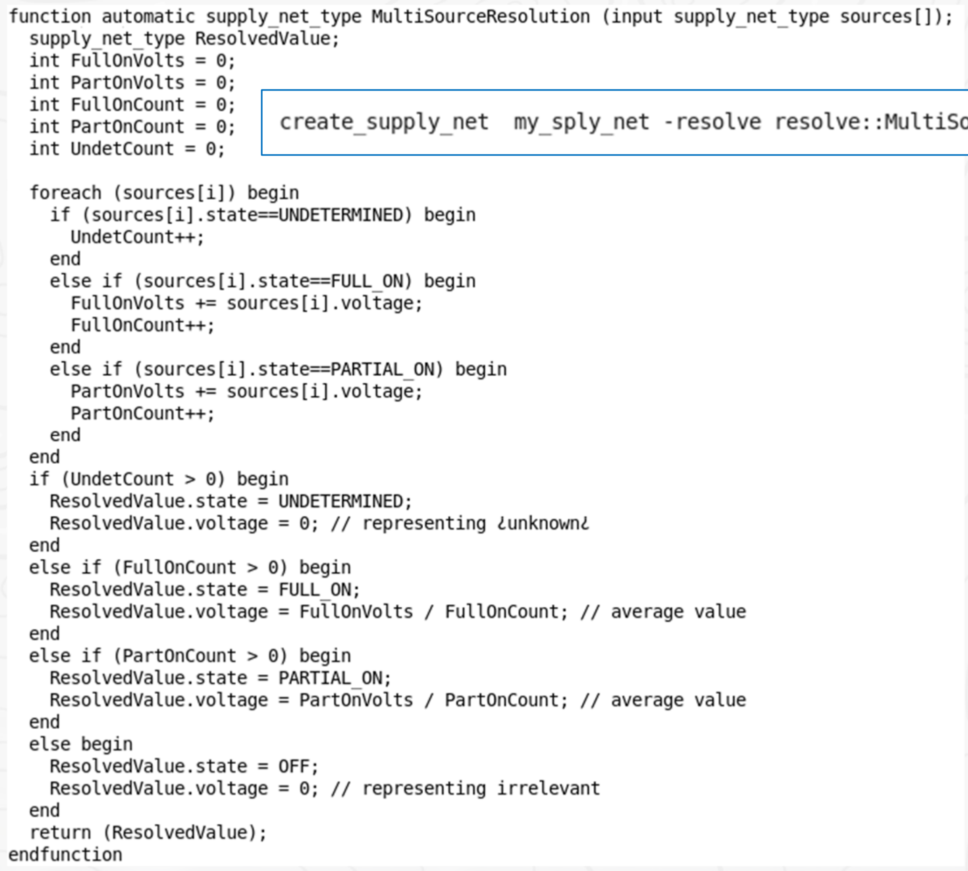
```
module my_macro ( a, b, c
`ifdef USE_MACRO_PG_PINS
  , vdd_macro
  , vss_macro
`endif
);
  input a;
  input b;
  output c;

`ifdef USE_MACRO_PG_PINS
`ifdef USE_MACRO_IO_PINS
  inout vdd_macro;
  inout vss_macro;
`else
  input vdd_macro;
  input vss_macro;
`endif
`else
  supply1 vdd_macro;
  supply0 vss_macro;
`endif
 ...
```

- Well-defined Verilog behavior
- Can require modification of IP-provided models.

## Method 2: Custom Resolution Functions

- UPF provides several built-in resolution functions for resolving multiple-driver situations—which often occur when bi-directional ports are involved
  - ‘unresolved’ – resolves to UNDETERMINED if more than one source is connected/possible (default)
  - ‘one_hot’ – allows multiple drivers, but only one can be active at a time.
  - ‘parallel’ – expects multiple drivers, and is only FULL_ON if all sources are FULL_ON
  - ‘parallel_one_hot’ – combination of ‘one_hot’ and ‘parallel’: only one root supply can be active, but all derived supplies must be FULL_ON for FULL_ON resolution.
- For more complex interactions, a *SystemVerilog* function can be used instead to resolve the supply net in a custom manner.

# Method 2: Custom Resolution Functions



```
function automatic supply_net_type MultiSourceResolution (input supply_net_type sources[]);
  supply_net_type ResolvedValue;
  int FullOnVolts = 0;
  int PartOnVolts = 0;
  int FullOnCount = 0;
  int PartOnCount = 0;
  int UndetCount = 0;

  foreach (sources[i]) begin
    if (sources[i].state==UNDETERMINED) begin
      UndetCount++;
    end
    else if (sources[i].state==FULL_ON) begin
      FullOnVolts += sources[i].voltage;
      FullOnCount++;
    end
    else if (sources[i].state==PARTIAL_ON) begin
      PartOnVolts += sources[i].voltage;
      PartOnCount++;
    end
  end
  if (UndetCount > 0) begin
    ResolvedValue.state = UNDETERMINED;
    ResolvedValue.voltage = 0; // representing ¿unknown¿
  end
  else if (FullOnCount > 0) begin
    ResolvedValue.state = FULL_ON;
    ResolvedValue.voltage = FullOnVolts / FullOnCount; // average value
  end
  else if (PartOnCount > 0) begin
    ResolvedValue.state = PARTIAL_ON;
    ResolvedValue.voltage = PartOnVolts / PartOnCount; // average value
  end
  else begin
    ResolvedValue.state = OFF;
    ResolvedValue.voltage = 0; // representing irrelevant
  end
  return (ResolvedValue);
endfunction
```

```
create_supply_net  my_sply_net -resolve resolve::MultiSourceResolution
```

- Well-defined *SystemVerilog* behavior
- UPF 3.x only
- No inherently-defined way of distinguishing the multiple sources being resolved
  - No 'name' field to reference, so arguments must be kept track of positionally.

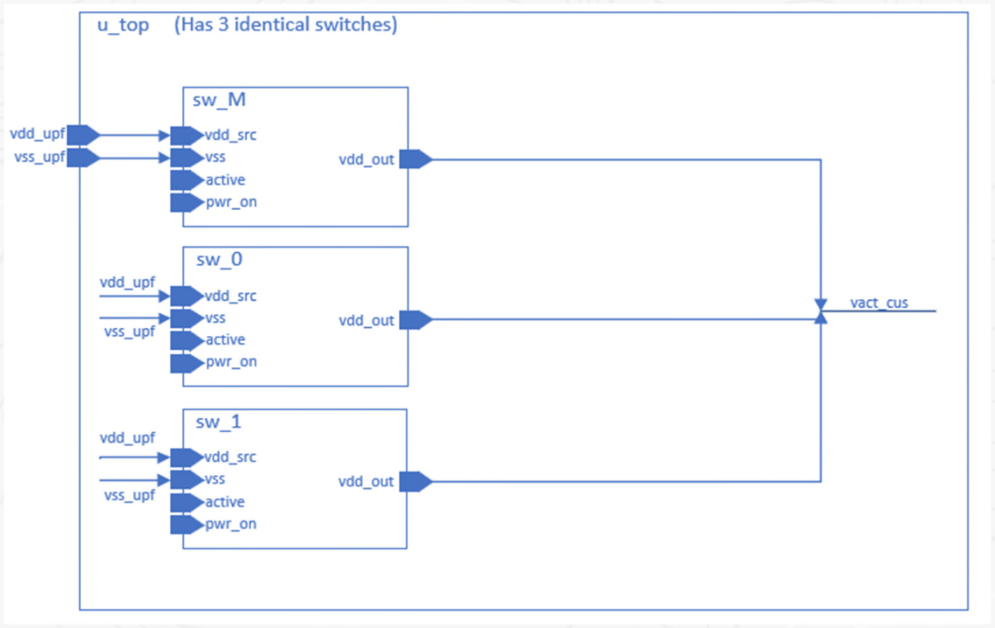
## Method 3: Using Built-in HDL Resolution

Rather than use the UPF-provided resolution functions, supply nets can be resolved in HDL directly by:

1. Intentionally removing/omitting ‘normally resolved’ UPF supply net connection
2. Including in testbench ‘bound in’ code that resolves the various HDL sources (using HDL values and signal strengths) to a single HDL resolution.
3. Connecting the resolved HDL signal to the UPF supply net within the UPF.

# Method 3: Using Built-in HDL Resolution

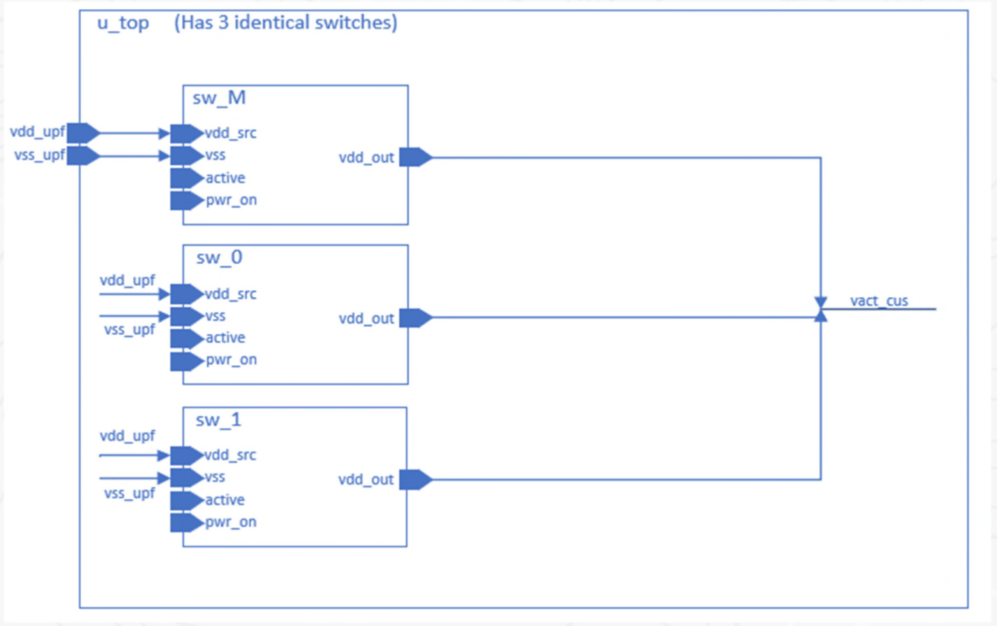
```
module top (
  act,
  sw
);
  input [2:0] act;
  input [2:0] sw;

  sw_model sw_0(
               .active(act[0]),
               .pwr_on(sw[0])
              );

  sw_model sw_1(
               .active(act[1]),
               .pwr_on(sw[1])
              );

  sw_model sw_M(
               .active(act[2]),
               .pwr_on(sw[2])
              );
endmodule
```



# Method 3: Using Built-in HDL Resolution

```
module top (
  act,
  sw
);
  input [2:0] act;
  input [2:0] sw;

  sw_model sw_0(
              .act
              .pwr
            );

  sw_model sw_1(
              .act
              .pwr
            );

  sw_model sw_M(
              .act
              .pwr
            );
endmodule
```

```
module bind_res (
  v_inM,v_in1,v_in2,
  v_out
);
  input      v_inM;
  input      v_in1;
  input      v_in2;
  output reg v_out;

  always @(v_inM,v_in1,v_in2)
  begin
    if (v_inM !== 1'bz) begin
      v_out <= v_inM;
    end
    else begin
      if (v_in1 === 1'bx || v_in2 === 1'bx) begin
        v_out <= 'bx;
      end
      else if (v_in1 === 1'b1 && v_in2 === 1'b1) begin
        v_out <= 'b1;
      end
      else begin
        v_out <= 1'b0;
      end
    end
  end
endmodule
```

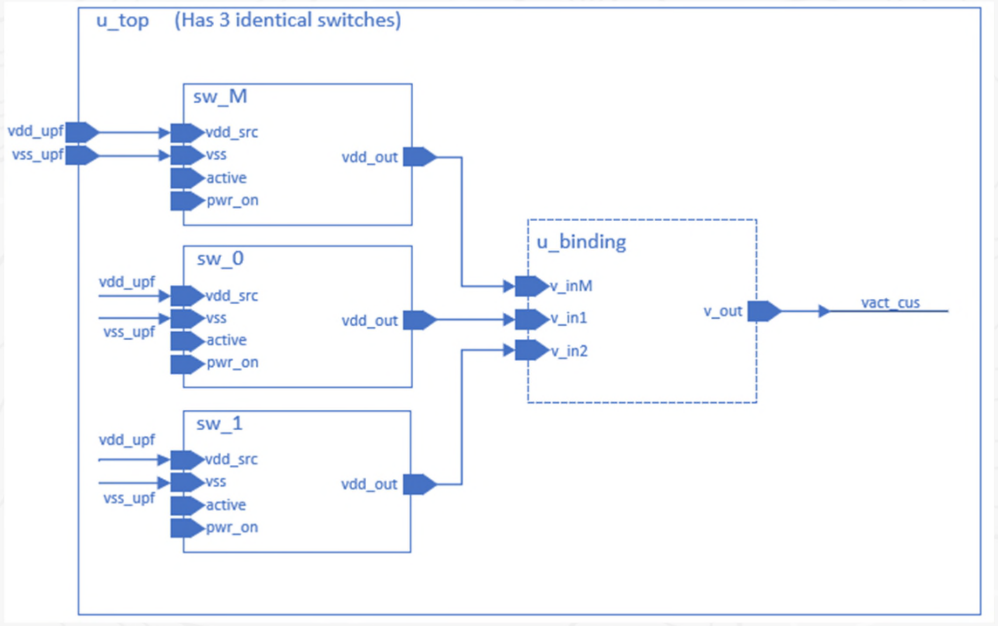


# Method 3: Using Built-in HDL Resolution

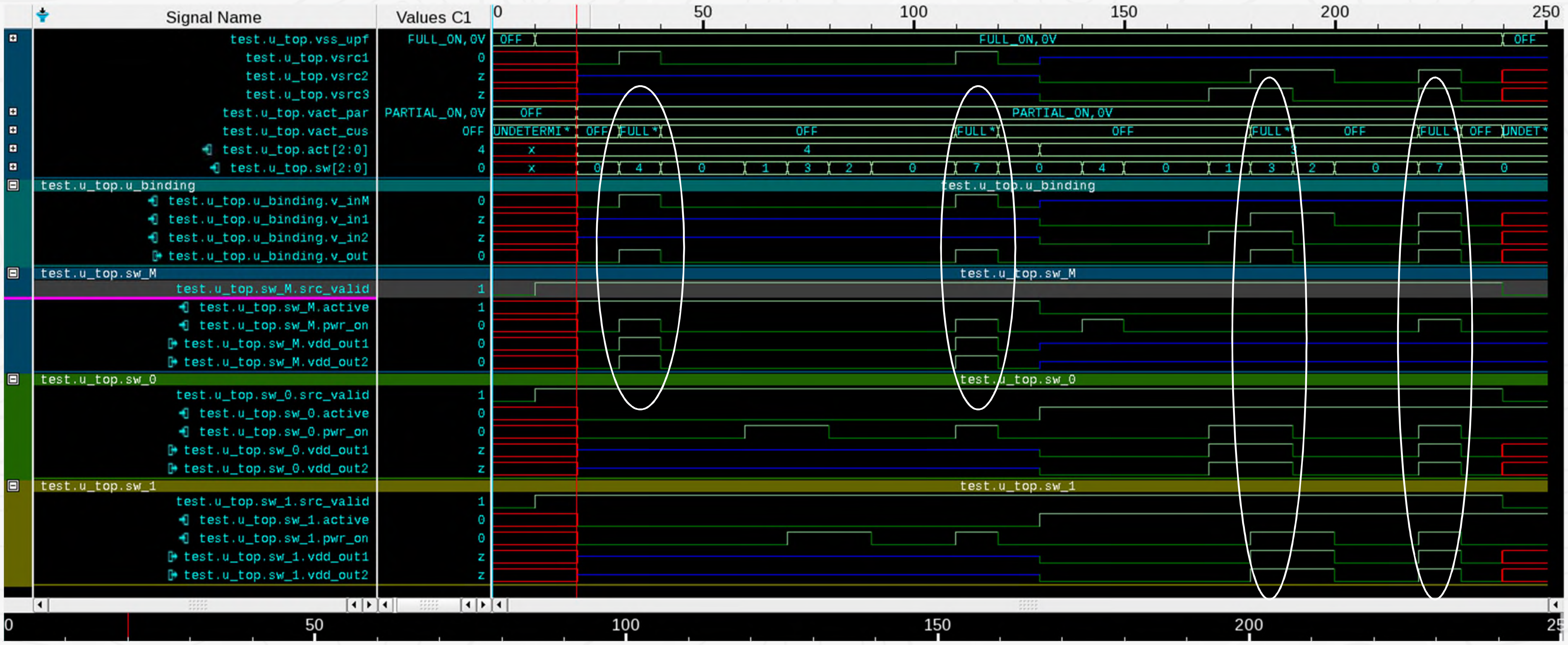
```
module top (
  act,
  sw
);
  input [2:0] act;
  input [2:0] sw;

  sw_model sw_0(
                 .act
                 .pwr
               );

  sw_model sw_1(
                 .act
                 .pwr
               );

  sw_model sw_M(
                 .act
                 .pwr
               );
endmodule
```

```
module bind_res (
  v_inM,v_in1,v_in2,
  v_out
);
  input      v_inM;
  input      v_in1;
  input      v_in2;
  output reg v_out;

  always @(v_inM,v_in1,v_in2)
  begin
    if (v_inM !== 1'bz) begin
      v_out <= v_inM;
    end
    else begin
      if (v_in1 === 1'bx || v_in2 === 1'bx) begin
        v_out <= 'bx;
      end
      else if (v_in1 === 1'b1 && v_in2 === 1'b1) begin
        v_out <= 'b1;
      end
      else begin
        v_out <= 1'b0;
      end
    end
  end
endmodule
```

```
# Connect input supplies to the bound module
connect_logic_net vsrc1  -ports { sw_M/vdd_out1 }
connect_logic_net vsrc2  -ports { sw_1/vdd_out1 }
connect_logic_net vsrc3  -ports { sw_0/vdd_out1 }

# Connect resolved supply net to bound module
connect_supply_net vact_cus  -ports {u_binding/v_out } -vct SV_LOGIC2UPF
```



# Method 3: Using Built-in HDL Resolution



```
Signal Name                          Values C1
test.u_top.vss_upf                   FULL_ON,0V
test.u_top.vsrc1                     0
test.u_top.vsrc2                     z
test.u_top.vsrc3                     z
test.u_top.vact_par                  PARTIAL_ON,0V
test.u_top.vact_cus                  OFF
test.u_top.act[2:0]                  4
test.u_top.sw[2:0]                   0
test.u_top.u_binding
  test.u_top.u_binding.v_inM         0
  test.u_top.u_binding.v_in1         z
  test.u_top.u_binding.v_in2         z
  test.u_top.u_binding.v_out         0
test.u_top.sw_M
  test.u_top.sw_M.src_valid          1
  test.u_top.sw_M.active             1
  test.u_top.sw_M.pwr_on             0
  test.u_top.sw_M.vdd_out1           0
  test.u_top.sw_M.vdd_out2           0
test.u_top.sw_0
  test.u_top.sw_0.src_valid          1
  test.u_top.sw_0.active             0
  test.u_top.sw_0.pwr_on             0
  test.u_top.sw_0.vdd_out1           z
  test.u_top.sw_0.vdd_out2           z
test.u_top.sw_1
  test.u_top.sw_1.src_valid          1
  test.u_top.sw_1.active             0
  test.u_top.sw_1.pwr_on             0
  test.u_top.sw_1.vdd_out1           z
  test.u_top.sw_1.vdd_out2           z
```

## Test Setup & Results

## Test Setup

- Default Behavior Tests:
  - Scenario 1A: Test UPF to HDL bi-directional connections (used as inputs)
  - Scenario 1B: Test UPF to HDL bi-directional connections (used as outputs)
  - Scenario 2A: Test hierarchical supply connections within UPF (HDL ports declared as bi-directional)
  - Scenario 2B: Test hierarchical supply connections within UPF (HDL ports declared with functional direction)
- Support for Modeling Approaches
  - Modeling 1: Test support for using compiler directives
  - Modeling 2: Test support for UPF custom resolution functions
  - Modeling 3: Test support for HDL resolution functions

## Results: Scenario 1A

- Test data showed that for the common scenario – where a bi-directional HDL port on a macro model was intended to be used as an input (sink) to the model, all three simulators resolved the port behavior as expected

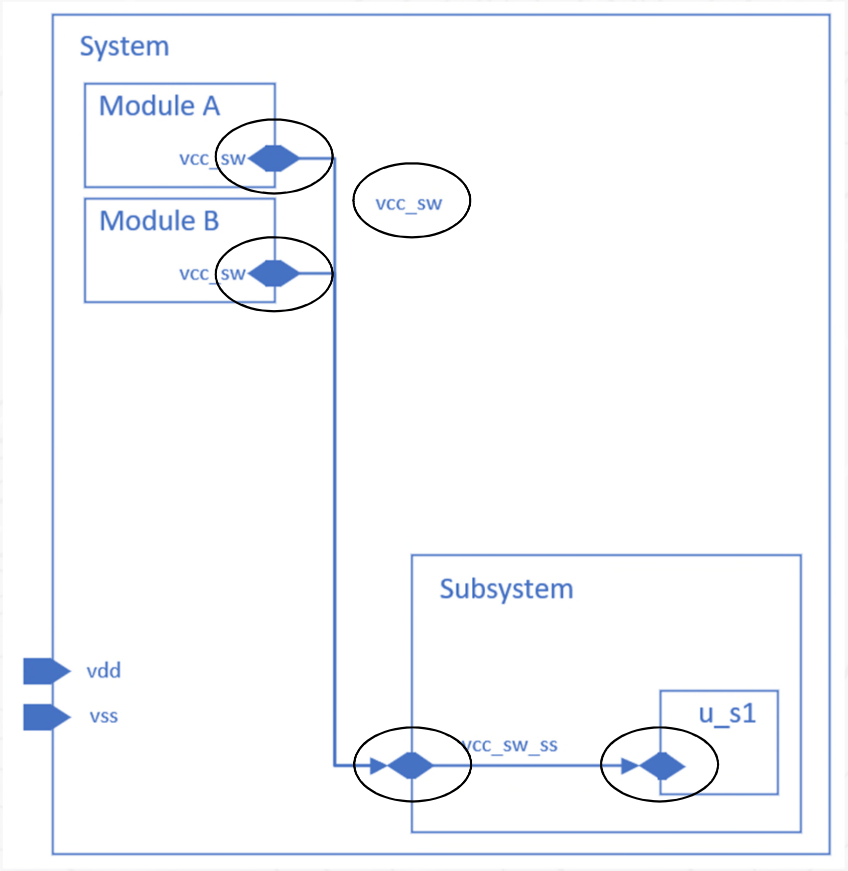
| Simulator A | Simulator B | Simulator C |
|---|---|---|
| -Warning that resolution (vct) functions could not be applied to HDL bi-di ports.<br>-Bi-di supply ports resolved correctly to 'input'. | -Bi-di supply ports resolved correctly to 'input'. | -Warning that the bi-directional HDL ports would be treated as inputs.<br>-Bi-di supply ports resolved correctly to 'input'. |

## Results: Scenario 1B

- The reverse scenario – where the bi-directional HDL signal was coming from a switch model and intended to be a source – showed mixed results.

| Simulator A | Simulator B | Simulator C |
|---|---|---|
| -Note that bi-di port is being treated as an output<br>-Bi-di supply ports resolved correctly to 'output'. | -HDL bi-di ports were treated as inputs; UPF supply net connections were treated as sources. | -Warning that bi-di HDL port was being treated as an input.<br>-HDL bi-di ports were treated as inputs; UPF supply net connections were treated as sources. |

# Scenario 2A: Hierarchical Supply Net



system_s.upf:

```
create_supply_port vdd
create_supply_port vss

create_supply_net  vdd
create_supply_net  vss
create_supply_net  vcc_sw -resolve parallel

connect_supply_net vcc_sw -ports {u_ma/vcc_sw}
connect_supply_net vcc_sw -ports {u_mb/vcc_sw}
connect_supply_net vcc_sw -ports {u_ss/vcc_sw}
```

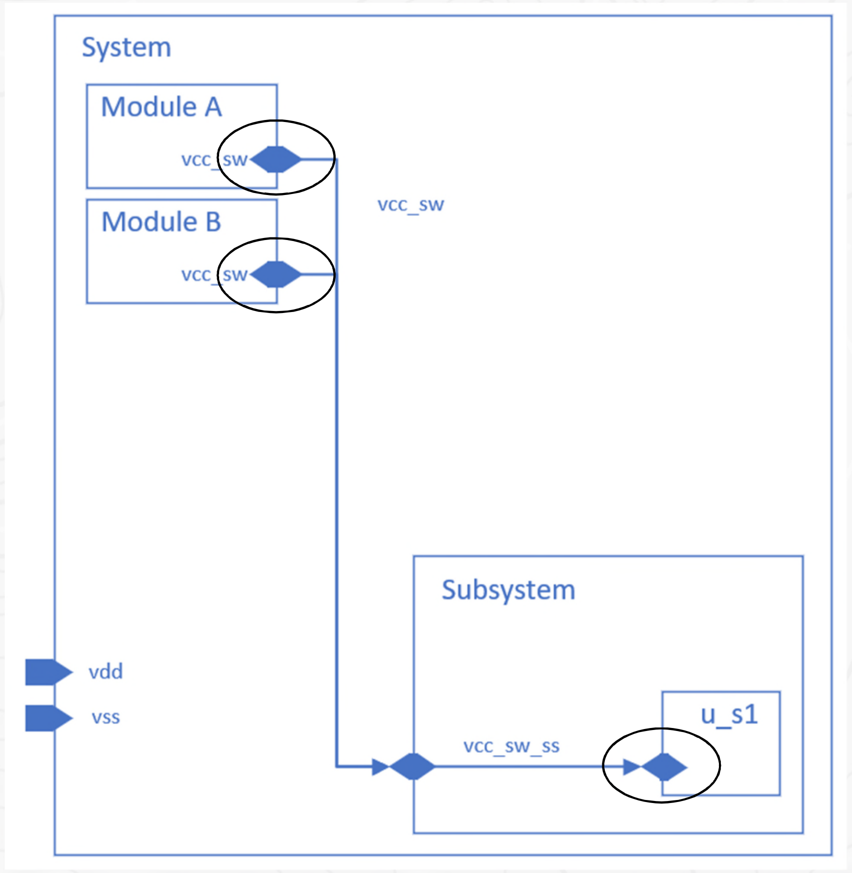
subsystem_ss.upf:

```
create_supply_port vcc_sw -direction inout
create_supply_net  vcc_sw_ss -resolve parallel
```

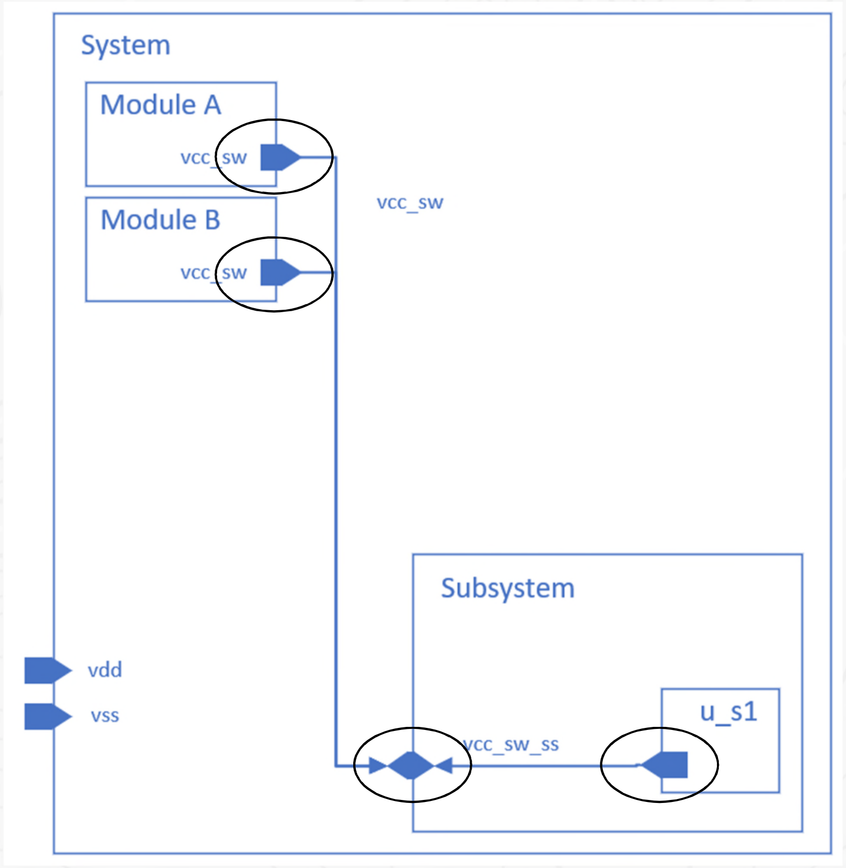
## Results: Scenario 2A – Hier UPF, bi-di HDL ports

- Could only test Simulator A (B & C treated as inputs)
- For Simulator A, the results were as expected; the behavior for HDL 'inout' ports matched expectations
  - Warning that 'inout' for UPF port direction is not supported…?
  - Top-level supply net resolved as expected—only FULL_ON when all hierarchical supplies were on.
  - Submodule supply net showed PARTIAL_ON as expected when only top-level supplies were on.

# Scenario 2A: Hierarchical Supply Net

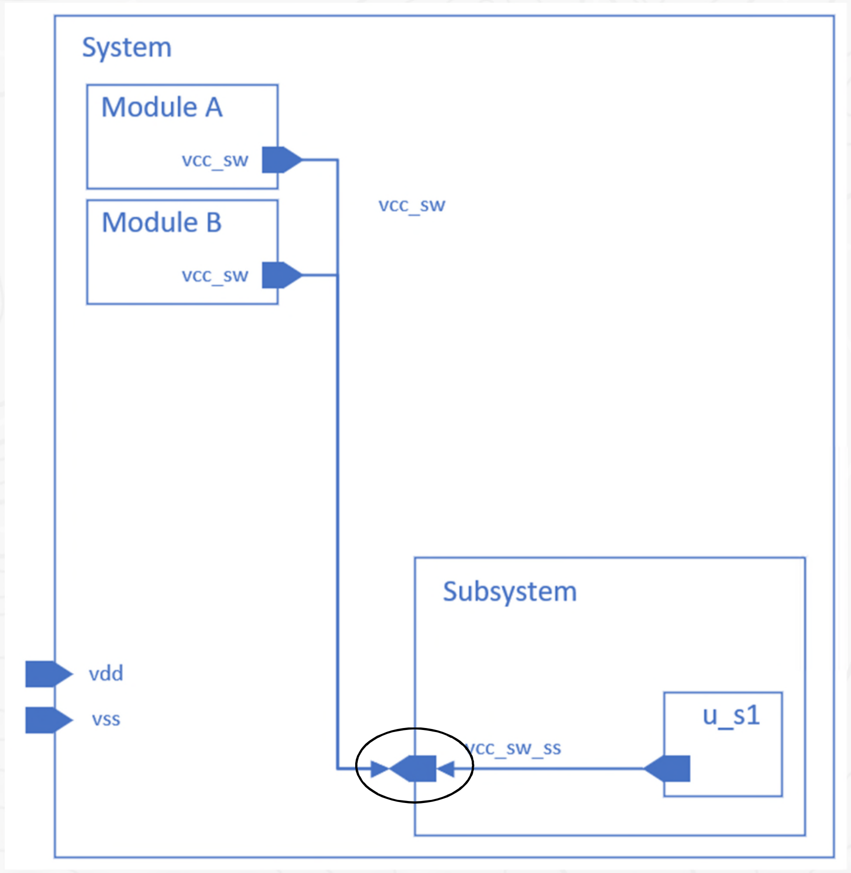


## Scenario 2B: Hierarchical Supply Net



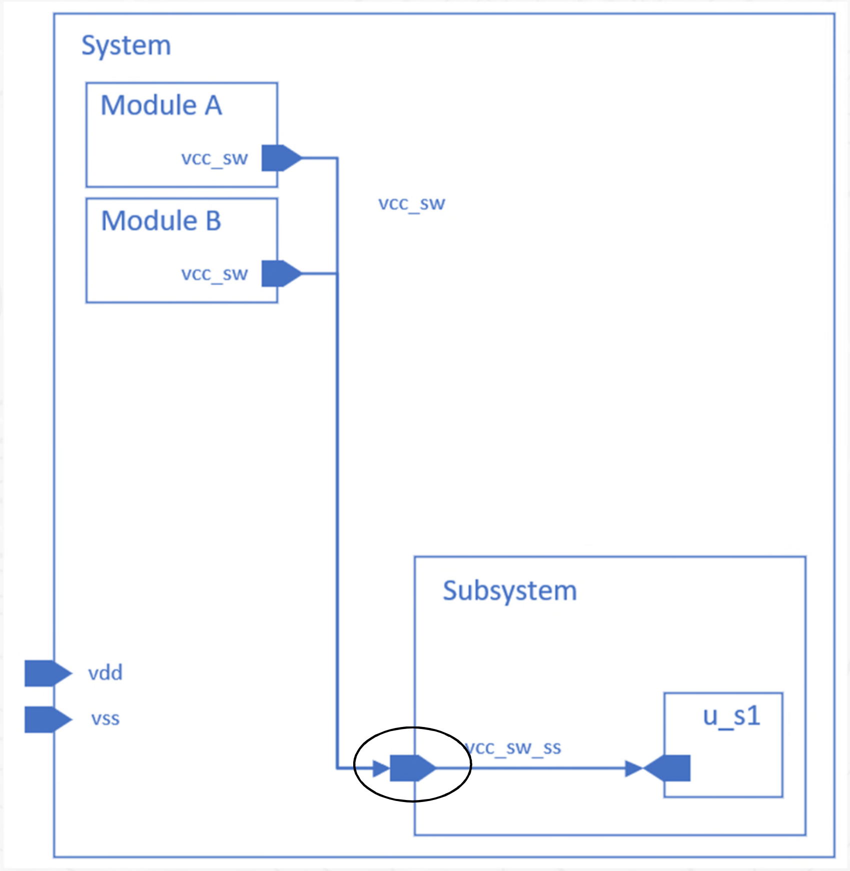
- System-level supply net (vcc_sw) has three drivers; it should be PARTIAL_ON when any one is on and only FULL_ON when all three are FULL_ON.
- Subsystem-level supply net (vcc_sw_ss) should exactly mimic this behavior.

# Scenario 2B: Hier Supply Net, ‘out’ UPF port



- System-level net (vcc_sw) still retains three drivers and ‘parallel’ resolution expectations…
- Subsystem-level net (vcc_sw_ss) now has only one driver (u_s1), so it should not be influenced by system-level drivers…

# Scenario 2B: Hier Supply Net, ‘in’ UPF port



- System-level supply net (vcc_sw) should not depend on subsystem net state for resolution.
- Subsystem-level supply net (vcc_sw_ss) should see two drivers (same as Scenario 2A)

## Results: Scenario 2B – Hier UPF, uni-di HDL ports

- The results showed that all three simulators modelled the proper behavior of the system supply net when it was declared ‘inout’ – showing FULL_ON only when all contributing supplies were on.
- However, all three simulators exhibited slightly different behavior when dealing with UPF supply port connections that are hierarchically connected, and **all three exhibited slightly errant behavior based on an expectation from the LRM**.

# Results: Scenario 2B

| Simulator A | Simulator B | Simulator C |
|---|---|---|
| Same results as Scenario 2A:<br>-Top-level supply net resolved as expected—only FULL_ON when all hierarchical supplies were on.<br>-Submodule supply net showed PARTIAL_ON when only top-level supplies were on. | -Top-level supply net resolved as expected—only FULL_ON when all hierarchical supplies were on.<br>-Submodule supply net showed PARTIAL_ON when only top-level supplies were on. | -Top-level supply net resolved as expected—only FULL_ON when all hierarchical supplies were on.<br>-Submodule supply net showed PARTIAL_ON when only top-level supplies were on. |
| -Unusual results were seen when hierarchical UPF port was declared as 'out': subsystem goes PARTIAL_ON when system supplies are active. | -Unusual results were seen when hierarchical UPF port was declared as 'out': subsystem goes PARTIAL_ON when system supplies are active. | -Unusual results were seen when hierarchical UPF port was declared as 'out' or 'in': results match the 'inout' port declaration case. |
| -Unusual results were seen when hierarchical port was declared 'in': subsystem supply never goes FULL_ON | -Unusual results were seen when hierarchical UPF port was declared as 'in': subsystem and system nets go PARTIAL_ON as soon as parent domain is valid. | |

## Results: Modeling 1 – Compiler directives

- Compiler directives are well supported by all three simulators under consideration, and there were no issues using these to get the expected behavior.

## Results: Modeling 2 – UPF resolution

- Support for user-defined resolution functions was somewhat uneven, with all three simulators exhibiting slightly different behavior.
- Simulator A was the only one to accept the example from the IEEE-1801 specification as written.

| Simulator A | Simulator B | Simulator C |
|---|---|---|
| Supported IEEE-1801 example resolution function. | Only allows resolution between two supply nets – task (not function). | Allows function definition only via vendor-provided IEEE-1801 custom extension, limited resolution options. |

## Results: Modeling 3 – HDL Resolution

- When using HDL resolution functions, the results were mostly positive; all three simulators supported the basic functions of binding a resolution module via a testbench and resolved HDL conflicts as expected.
- However, only Simulator A supported doing this on HDL nets defined and connected within the UPF with ‘create_logic_net’ and ‘connect_logic_net’ commands.
  - Simulators B and C required the nets to be defined within the HDL.
  - **This limitation in Simulators B and C seems arbitrary and out of sync with the expectations of the UPF LRM to allow the creation of new logic nets when needed**.

## Analysis & Conclusions

## Analysis

- The UPF/HDL resolution vectors ‘vct’ do not seem to be used for resolving supply net drive direction—in spite of the fact that they are written and applied with a source/destination direction in mind.
- Simulator A was the only simulator to allow bi-directional HDL ports to be connected and used as outputs; the other two defaulted the ports to ‘inputs’ in spite of the context of what they were being connected to.  It was also the only simulator to fully support the example IEEE1801 custom resolution function as written, and it was the only simulator to allow binding connections with UPF-provided logic nets.

## Analysis

- UPF custom resolution functions are difficult to apply because supply nets do not contain a ‘name’ or ‘id’ that would allow a user to distinguish between a set of supplies to resolve.
- The resulting supply sources of ‘supply_net_type’ appear only with state and voltage information, and the UPF LRM does not provide an explicit definition about the order that supply net connections are maintained in when multiple ‘connect_supply_net’ commands could each make multiple supply net connections each.

## Conclusions

- The default treatment of inout HDL ports and UPF hierarchical port connections vary across EDA vendors…
- Basic support of common bi-directional port modeling function is uneven across the three major EDA simulators…
- Future UPF revisions should seek to take advantage of ‘vct’ drive direction information when provided to determine which side of a supply net is the source.
- Future UPF revisions should make this ordering of resolution function supplies explicit, or future supply_net_type definitions should include a name field that is populated when nets are declared.

## Questions?

Thank you for your attention!