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

Brandon Skaggs
Cypress Semiconductor, An Infineon Technologies Company
201. E. Main Street, Suite 500A
Lexington, KY 40508
brandon.skaggs@infineon.com

***Abstract*—The UPF LRM does not describe in detail the handling of bi-directional (inout) HDL port connections, and this leaves the handling of commonly-used connections open to the interpretation of EDA vendors. In this paper we present two common bi-directional supply port connections that are often the source of unexpected corruption before discussing three possible methods for working around them. We then examine how well each of the three main EDA simulators support the initial scenario and the proposed workarounds. Finally, possible future enhancements to the UPF LRM to ensure well-defined behavior across all tools is discussed.**

I. Introduction

The IEEE1801 Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems—commonly known as the Unified Power Format (UPF) standard—provides a framework for describing the power intent and supply distribution for both implementation and power-aware verification tools. Power domains, supply connectivity, and expected (or invalid) power states can be defined in a way to direct synthesis tools to select cell libraries (typically provided in *Liberty* format) appropriate for the design, LINTing tools to verify that implementation is achievable, and layout tools to properly implement the intent. Functional power-aware simulation (which often inserts or substitutes functional, power-aware behavioral models for *Liberty* cells/macros) can also use the UPF power intent specification to verify correct functional behavior along with power state transitions and correct isolation and retention during power-down.

However, the behavior of bi-directional HDL ports commonly used to model supplies within power-aware behavioral models are not well defined by even the most recent UPF LRM specifications. This leaves the behavior of these ports (and their interaction with connected supply ports) up to the implementation chosen by the EDA vendor.

This paper presents two common ‘problematic’ bi-directional port modeling issues found within power-aware simulations, describes possible methods for modeling these connections, and tests the support for each of these methods within the most widely-used power-aware simulators in the market. Finally, enhancements to future UPF LRM revisions are discussed—enhancements that would align expectations to a common, well-defined behavior.

A. *Motivation & Contributions of this Paper*

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. This paper presents a systematic survey of the behavior of contemporary EDA functional, digital simulators—in an area where the IEEE specification is not explicit—in order to inform these best practices and suggest improvements in future IEEE 1801 revisions.

B. *Organization of this Paper*

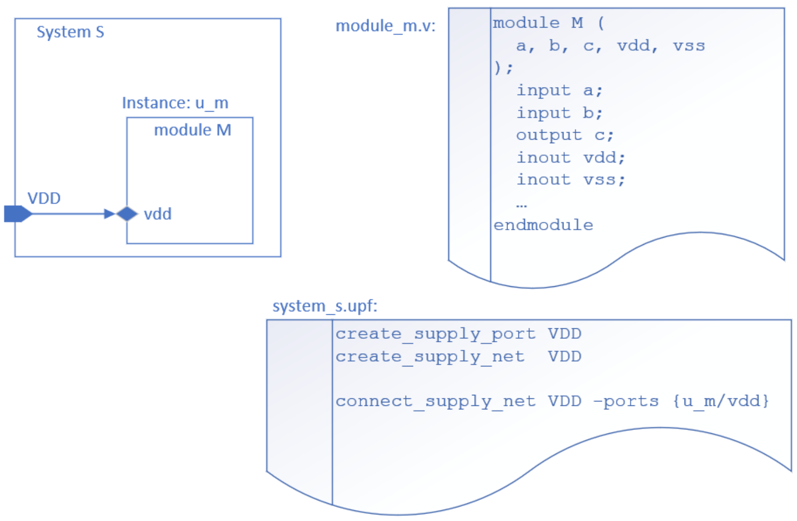
We begin our discussion by describing some common areas where bi-directional ports are used within power-aware simulations, followed by a presentation of several common methods for modeling bi-directional ports and their proper resolution. Data is then presented on the default behavior of the three major EDA simulators within these common scenarios and modeling approaches—discussing the results and any issues encountered. Finally, the paper concludes by discussing items for future study.

II. Key Terms

Unified Power Format (UPF)	IEEE standard format for describing intended design power intent abstractly
Power Management	Pertaining to the modulation of supply for the purpose of reducing circuit power
Power-aware Simulation	Digital simulation where power-related behavior is modelled.
Supply Net type	Object type representing a supply with a state and voltage component
Source	Object providing power to a sink
Sink	Object being driven by a source

III. Common Bi-Directional Port Modeling Scenarios

Consider the common scenario where a connection is made between a power-aware simulation model and a UPF supply net, as presented in Figure 1. Since supply voltages are often treated within Verilog as logic ports with ‘inout’ port direction, this creates a classic ‘supply net to HDL inout’ connection within the simulation.



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

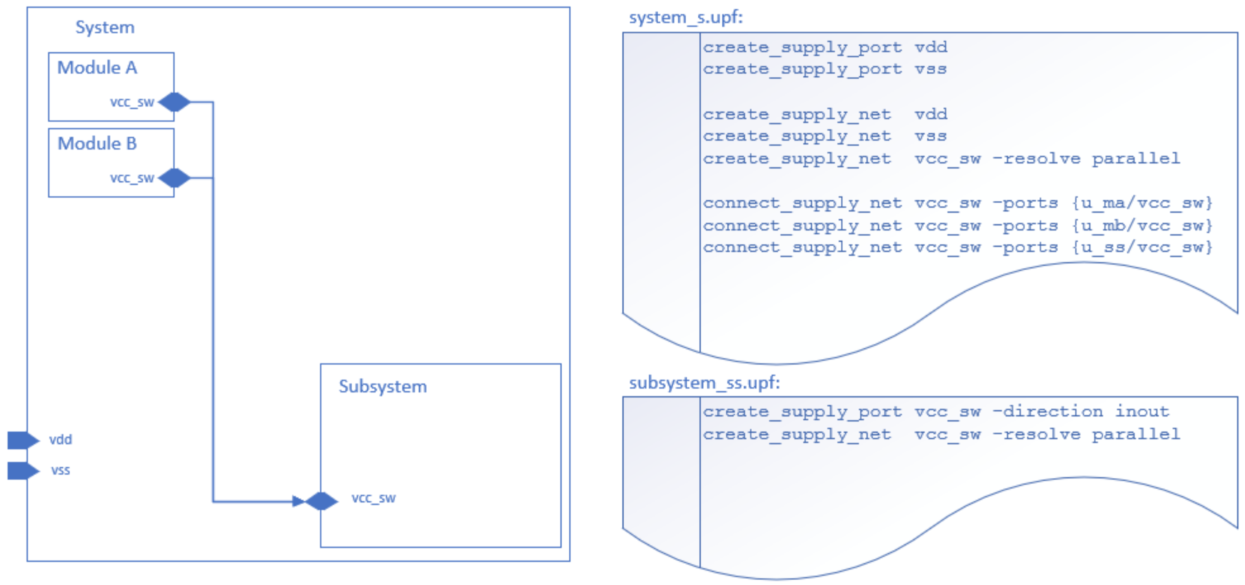
```
create_supply_port VDD
create_supply_net  VDD

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

Figure 1. Typical Bi-Directional HDL to Supply Net Connections (Scenario 1)

From the connection itself, it is not clear within the system S whether the behavioral model of module M is meant to be a source or sink for the connected supply net VDD. If the ‘vdd’ HDL port on module M is meant to model an input supply, it wouldn’t be expected to drive the connected supply net; however, if module M were modeling a power switch, and the HDL port was representing a switched output, then it would be a source and not a sink on the supply net. Per the UPF LRM, the existence of an ‘inout’ port implies the latter (a source driving the supply net), but the common scenario is the former (a simple sink being driven by the supply). This often creates a ‘multiple driver’ error and corruption on the supply net—when corruption on the supply net is not what would actually happen within the system.

This scenario can be extended to examine a similar resolution concern when ‘inout’ UPF supply ports are used to describe connections within a hierarchical power intent definition. Consider the scenario in Figure 2 where a supply net is driven by several power switches – distributed among submodules whose power intent is described independently and referenced hierarchically.



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 -resolve parallel
```

Figure 2. Example of a Hierarchical Supply Connection Requiring ‘inout’ Supply Ports (Scenario 2)

In this example, the resolution of the *vcc_sw* supply net can only be obtained if the state of the submodule’s power switch can be reflected onto the net through the ‘inout’ supply port; however, some simulators do not fully support ‘inout’ supply ports, and instead model them as the default ‘input’ direction. This can create challenges when hierarchical abstraction and re-use are leveraged within the chosen verification methodology. Other methods for resolving the state of these parallel networks is required, and they are the topic of the next section.

IV. Methods for Modeling Bi-Directional Ports

There are several commonly-used methods for handing problematic bi-directional supply ports within simulation; this paper will discuss three methods, discuss the advantages and disadvantages of each, and test compatibility among the three most commonly-used simulators.

A. *Using Compiler Directives to Modify HDL Port Direction*

This method involves defining the behavioral model port directions based on compiler directives in order to select the most appropriate unidirectional port direction when the model is compiled for power-aware simulation. Since it uses well-supported pre-compiler syntax and directives to completely avoid the bi-directional situations discussed in Section III, this approach is often the most straightforward and widely-supported approach. However, it requires modification of IP-provided models—which may not be an option for cases where third-party IP are used. An example of a macro written with compiler directives to manage port direction is given in Figure 3.

B. *Defining Supply Net Custom Resolution Functions (UPF 3.x)*

IEEE1801-2015 (commonly referred to as UPF 3.0) revised the UPF LRM to add the ability to create custom supply net resolution function where the pre-defined resolution functions (*unresolved*, *one_hot*, *parallel*, and *parallel_one_hot*) were not sufficient to resolve more complex supply connections. By creating the capability to define resolution rules that could be unique for any given supply net, conflicts caused by bi-directionally-driven supply nets could potentially be resolved by examining the states of all drivers and assigning an appropriate value for the resolved net based on the context.

```
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
 ...
```

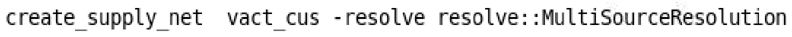
Figure 3. Example of Using Compiler Directives to Resolve Bi-Directional Ports

An example of a custom resolution function from the IEEE1801-2015 specification is presented in Figure 4. Figure 5 shows how this custom function could be referenced within the UPF supply net declaration.

```
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
```

Figure 4. Custom Resolution Function from IEEE 1801-2015, Section 6.24.3



```
create_supply_net  vact_cus -resolve resolve::MultiSourceResolution
```

Figure 5. Referencing a Custom Resolution Function from a User-Defined Package

C. Resolving Supply Net Using HDL Resolution Functions

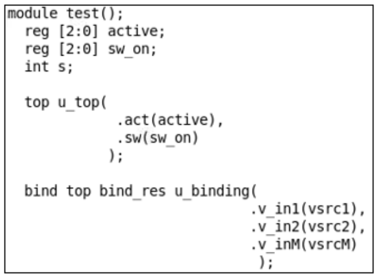
This approach involves intentionally removing UPF connections for supply nets that have problematic HDL bi-directional connections—replacing them with testbench code that monitors changes in the source supplies and then reflects the desired resolution on the HDL model nets and supply nets as appropriate. Since the testbench-provided code intercepts the possible drivers, the built-in HDL resolution function of the testbench language (Verilog, SystemVerilog, or VHDL) can be leveraged to resolve the supply net state before it is driven onto the supply net by the testbench – which is now its only source.

One key advantage to this approach (over using the UPF native supply net resolution functions or writing a custom resolution function) is the larger number of net values that can be used. Whereas UPF has only UNDETERMINED, OFF, PARTIAL_ON, and FULL_ON states for supply net resolution, HDL signals (in addition to ‘X’, ‘0’, and ‘1’ states) add signal values for high-impedance (‘Z’) and various signal strengths (‘W’, ‘L’, ‘H’, ‘strong’, ‘weak’, ‘pull’, etc.) – all with well-defined resolution functions that can be used to model priority or combinations of acceptable values. By giving different contributors to the supply net resolution different HDL strengths, a higher granularity of control is available.

Another key component of this approach is that it can be achieved by ‘binding’ resolution functions from within testbench code – without requiring changes within the design under test to accommodate this modeling.

Let’s look at an example of how HDL resolution could be performed. In this example, a system where the active supply net (vact_cus) can be driven by a single external supply (sw_M), or—if the supply is absent (driven ‘Z’ by sw_M)—it can be switched on internally by a series of internal switches (sw_1 and sw_2) and resolved as a parallel net. Note that in this scenario, there is no pre-defined resolution function for ‘vact_cus’ that is appropriate.

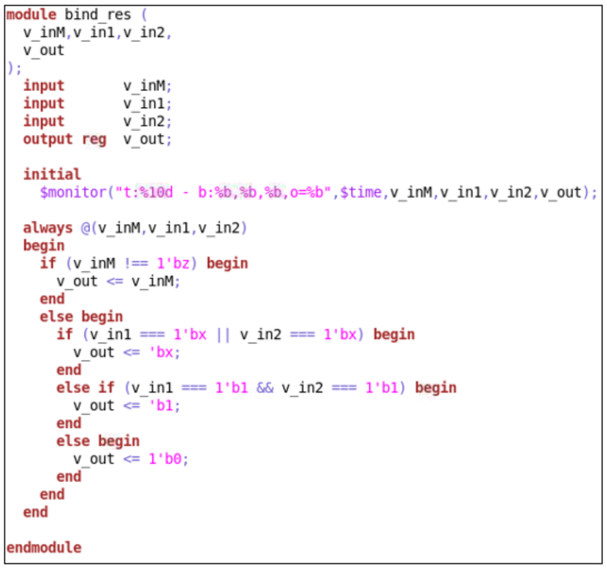
Rather than connect the outputs of all three switches to ‘vact_cus’ and provide a resolution function, the outputs of the switches are connected to a module that is added via a ‘bind’ statement from the testbench. This module contains the resolution function that allows the switch output to follow ‘sw_M’ if it is driven, but to follow the state of both ‘sw_1’ and ‘sw_2’ if it isn’t. A supply net connection within the UPF file makes the final translation and assignment from HDL to supply_net_type. Figure 6 shows the relevant HDL and UPF statements, and Figure 7 shows the resulting waveform.



```
module test();
  reg [2:0] active;
  reg [2:0] sw_on;
  int s;

  top u_top(
            .act(active),
            .sw(sw_on)
           );

  bind top bind_res u_binding(
                              .v_in1(vsrc1),
                              .v_in2(vsrc2),
                              .v_inM(vsrcM)
                              );
```

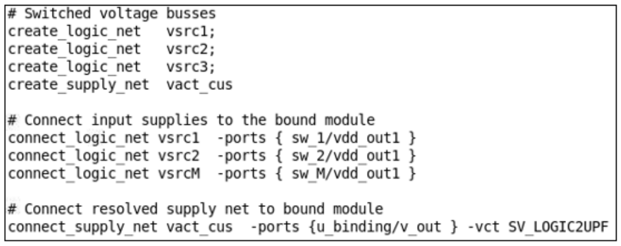


```
module bind_res (
  v_inM,v_in1,v_in2,
  v_out
);
  input       v_inM;
  input       v_in1;
  input       v_in2;
  output reg  v_out;

  initial
    $monitor("t:%10d - b:%b,%b,%b,o=%b",$time,v_inM,v_in1,v_in2,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
```



```
# Switched voltage busses
create_logic_net   vsrc1;
create_logic_net   vsrc2;
create_logic_net   vsrc3;
create_supply_net  vact_cus

# Connect input supplies to the bound module
connect_logic_net vsrc1  -ports { sw_1/vdd_out1 }
connect_logic_net vsrc2  -ports { sw_2/vdd_out1 }
connect_logic_net vsrcM  -ports { sw_M/vdd_out1 }

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

Figure 6. Resolving UPF Supply Net Drivers using HDL Resolution Functions

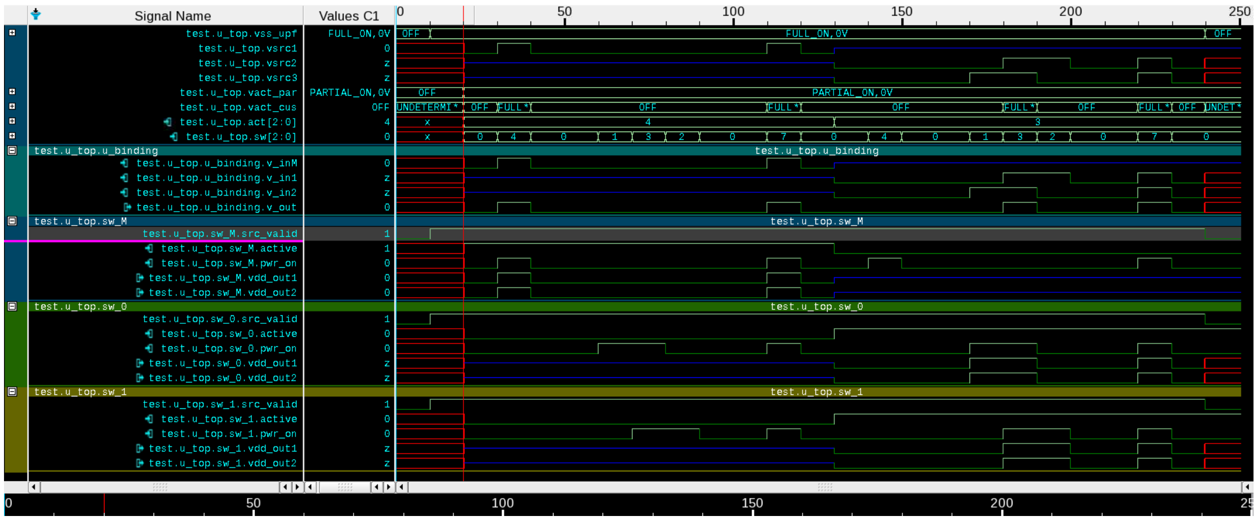


Figure 7. Resolved Supply Nets with Priority

From the waveform, you can see how 'vact_cus' is FULL_ON whenever sw_M is on; or, if sw_M is disconnected (Hi-Z), 'vact_cus' is FULL_ON when both sw_1 and sw_2 are on…

V. Test Scenarios

In order to provide a picture of EDA tool support for bi-directional ports, the behavior of the three most commonly-used power-aware simulators in the industry were tested for default behavior within the scenarios in Section III—for default behavior and for support of the as well as for the modeling methods described in Section IV.

Default Behavior:
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

Each simulator was randomly assigned a designator A, B, or C, and the results of testing with each are described in Section VI.

VI. Test Results

Scenario 1A: Test UPF to HDL bi-directional connections (used as inputs)

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. -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. -Bi-di supply ports resolved correctly to 'input'.

Scenario 1B: Test UPF to HDL bi-directional connections (used as outputs)

The reverse scenario – where the bi-directional HDL signal was coming from a switch model and intended to be a source – showed mixed results. Simulators B and C treated the connected supply nets the same as in Scenario 1A and the connected bi-directional HDL ports as sinks. Only Simulator A was able to determine that the bi-directional port was being driven by the HDL and modeled the appropriate behavior.

Simulator A	Simulator B	Simulator C
-Note that bi-di port is being treated as an output -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. -HDL bi-di ports were treated as inputs; UPF supply net connections were treated as sources.

Scenario 2A: Test hierarchical supply connections within UPF (HDL ports declared as bi-directional)

Because of the findings in Scenario 1B—that HDL bi-directional ports were always treated as inputs by two of the three simulators—it was not possible to model Scenario 2A with Simulators B and C and see any reasonable behavior for the system model where a submodule with an 'inout' UPF port drove onto a system supply net.

For Simulator A, the results were mixed; the behavior for 'inout' ports matched expectations; however, unusual results were seen when the submodule / hierarchical UPF port was declared as 'in' or 'out' mode.

Simulator A	Simulator B	Simulator C
-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 top-level supplies were on. -Unusual results were seen when hierarchical UPF port was declared as 'out': subsystem goes PARTIAL_ON when top-level supplies are enabled. -Unusual results were seen when hierarchical UPF port was declared as 'in': subsystem never goes FULL_ON.	Test scenario not supported (See Scenario 1B above)	Test scenario not supported (See Scenario 1B above)

Scenario 2B: Test hierarchical supply connections within UPF (HDL ports declared with functional direction)

This scenario was created to test the hierarchical UPF port connections after addressing the HDL port issues seen in Scenario 2A. The results showed that all three simulators modelled the proper behavior of the system supply net – 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**.

Simulator A	Simulator B	Simulator C
Same results as Scenario 2A.	-Top-level supply net resolved as expected—only FULL_ON when all hierarchical supplies were on. -Submodule supply net showed PARTIAL_ON when top-level supplies were on. -Unusual results were seen when hierarchical UPF port was declared as ‘out’: net behaves as if it were ‘inout’ and goes PARTIAL_ON when system supplies are enabled. -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.	-Top-level supply net resolved as expected—only FULL_ON when all hierarchical supplies were on. -Submodule supply net showed PARTIAL_ON when top-level supplies were on. -Unusual results were seen when hierarchical UPF port was declared as ‘out’ or ‘in’: results match the ‘inout’ port declaration case.

Modeling 1: Test support for using compiler directives

As described in Section 4A above, compiler directives are well supported by all three simulators under consideration, and there were no issues using these to get the expected behavior.

Simulator A	Simulator B	Simulator C
Full support for compiler directives	Full support for compiler directives	Full support for compiler directives

Modeling 2: Test support for UPF custom resolution functions

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 B would only allow custom resolution sequences written as tasks—and documentation confirmed that **resolution was limited to only two input supply nets**. This restriction seems arbitrary and likely to make this vendor’s implementation unreasonable for real applications.

Simulator C supported some modifications to the standard supply net resolution functions via tool-specific extensions to the UPF commands, and only a very limited scope of changes. **There was no capability to define a generic resolution function within an HDL package**.

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.

Modeling 3: Test support for HDL resolution functions

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.

Since this approach attempts to replace supply net connections normally made via UPF with logic net connections, the preferred location for creating these connections would be within UPF – rather than forcing a change to the HDL under test. **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**.

Simulator A	Simulator B	Simulator C
Binding worked as expected; logic nets created and connected within UPF.	Binding won't work on UPF-created logic nets. If HDL nets are created in original RTL, binding works as expected.	Binding won't work on UPF-created logic nets. If HDL nets are created in original RTL, binding works as expected.

VII. Analysis & Conclusions

Some other general observations can be made from the above results:

- The UPF/HDL resolution vectors 'vct' do not seem to be used for resolving drive direction—in spite of the fact that they are written and applied with a source/destination direction in mind. **Future UPF revisions should seek to take advantage of this information when provided to determine which side of a supply net is the source**.

- 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.

- 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. **Future UPF revisions should make this ordering explicit**, or future supply_net_type definitions should include a name field that is populated when nets are declared.

VIII. Future Study

This survey was performed with the most contemporary versions of tools available to the author; however, in some cases the results could be different in newer releases. The results should be reviewed with each tool vendor to confirm that the behavior seen is consistent with newer releases or if newer versions have a different behavior – either by default or with available options.

References

[1] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801-2013, 29 May 2013.

[2] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801-2015, 5 December 2015.

[3] Design Automation Standards Committee of the IEEE Computer Society, "IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems", IEEE Std. 1801-2018, 27 September 2018.