Low Power Verification With LDO

Shang-Wei Tu
MediaTek Inc.
+886-3-5670766#23493
kuma.tu@mediatek.com

Amol Herlekar
Synopsys Inc.
+91-80-4018-8337
herlekar@synopsys.com

Yu-Juei Chen
MediaTek Inc.
+886-3-5670766#22794
maso.chen@mediatek.com

***Abstract*-Low-Dropout Regulators or LDOs are used to regulate an output voltage that is powered from a higher input voltage. Hence, they are commonly used in advanced low power designs, especially for Dynamic Voltage and Frequency Scaling (DVFS). For low power verification engineers, modeling such an LDO is a huge problem. Although Unified Power Format (UPF) or IEEE-1801 standard [1] is defined and released for both the low power implementation and verification, UPF still does not provide a mechanism to model LDOs. Hence, verification engineers are left with no choice but to continue modeling the behavior in SystemVerilog. As the number of LDOs grows in the design, this challenge becomes manifold and can result in a number of verification holes. Also since the LDO is modeled in SystemVerilog, it effectively loses the UPF messaging and debug infrastructure. Besides, another problem verification engineers facing is the resolution mechanism that determines the state and voltage of the supply net when the net has multiple supply sources driving it. Although UPF provides *parallel*, *one_hot*, and *parallel_one_hot* mechanism, these are not sufficient to exactly simulate all kinds of resolution.**

In this paper, we share our experience on how we were able to model the LDO in UPF by extending the UPF command "*create_power_switch*" with some more option. In addition, we also share how we extended the UPF command "*create_supply_net -resolve*" to simulate the required resolution semantics. With the new syntaxes, we are able to model the LDOs and the supply net resolution easily and more accurately.

I. INTRODUCTION

Optimizing power consumption has become equally important along with optimizing performance for mobile based SoCs. For applications like Internet of Things (IoT), minimizing power consumption is even more important than improving chip computing power, because changing battery could be very difficult and for some extreme cases, devices might require to be self-powered and operate with the very little energy. Hence, to aggressively minimize the power consumption, various power management schemes are used. IEEE-1801 or UPF is commonly adopted to capture the power intent of low power designs. However, adopting UPF into the IC design flow is not simple plug and play. Adopting UPF introduces extra design and verification efforts, since different power states could have different impact on design function and leakage power. Hence, we require a robust low power verification methodology to minimize the risk of failure of low power designs.

Low-Dropout Regulators or LDOs play an important role in the power management. LDOs are used to regulate an output voltage that is powered from a higher input voltage. However, for low power verification engineers, modeling such an LDO is a huge problem. Before the advent of Unified Power Format (UPF) or IEEE-1801 standard, engineers had different ways to model the power management logic within the design. UPF solved that problem to a large extent and now engineers had an industry blessed standard, which could be used to capture the power intent. Unfortunately, UPF still does not provide a mechanism to model LDOs. So verification engineers are left with no choice but to continue to model the same in SystemVerilog. Now this is a challenging task because most of the supply network is captured in the UPF. So engineers have to make sure that LDO inputs in SystemVerilog are connect to and being driven by correct UPF nets and LDO output is connected to corresponding UPF nets. As the number of LDOs grows in the design, this challenge becomes manifold and can result in a number of verification holes. Also since the LDO is modeled in SystemVerilog, it effectively loses the UPF messaging and debug infrastructure.

Another problem verification engineers facing is related to the resolution mechanism that determines the state and voltage of the supply net when the net has multiple supply sources driving it. Although UPF provides *parallel*, *one_hot* and *parallel_one_hot* mechanism, these are not sufficient to exactly simulate all kinds of resolution for real designs.

In this paper, we share our experience on how we were able to model the LDOs in UPF by extending the UPF command "*create_power_switch*" with some more option. Similarly, we also share how we extended the UPF command "*create_supply_net -resolve*" to simulate the required resolution semantics. We plan to submit these UPF

extensions to IEEE-1801 committee. With the new syntax, we are able to model the LDOs and the supply net resolution easily and more accurately and it helps to make our low power verification more robust removing various verification holes and instilling more confidence for us while signing off. With the new syntax, we are also able to specify all LDO output power states with "*create_power_switch*" command, and hence the simulator could automatically convert all these states into *covergroup* helping us to achieve comprehensive low power coverage [2].

This paper is organized as follows. Section II describes an example of an LDO and the UPF limitations in modeling LDOs. It also describes the supply net resolution challenges which cannot be resolved with *parallel*/*one_hot*/*parallel_one_hot* resolution techniques. Section III details our previous solution for modeling the LDO in SystemVerilog and also captures the issues encountered with that methodology. In Section IV, the new UPF syntax is introduced and its benefits have been illustrated. Finally, Section V concludes this paper.

II. Example LDO design and UPF Limitation for Modeling

A. LDO Design and the UPF Limitation

Given an LDO design, the I/O pins of the LDO are illustrated in **TABLE I** and the control sequence waveform is demonstrated in **Figure 1**. In short, the LDO will drive power when **LDO_EN** is active (i.e. 1'b1) and will be OFF when **LDO_EN** is inactive (i.e. 1'b0). The LDO output voltage will depend on the value of **LDO_VOSEL** when **LDO_EN** is active.

TABLE I. Pin Specification of the example LDO.

Pin Name	Type	Direction	Description
VDD	**PG**	**Input**	**Power**
VSS	**PG**	**Input**	**Ground**
LDO_VOUT	**PG**	**Output**	**LDO output voltage**
LDO_VOSEL[2:0]	**Signal**	**Input**	**Output voltage level selection** **3'h0: 0.600V 3'h4: 1.000V** **3'h1: 0.700V 3'h5: 1.100V** **3'h2: 0.800V 3'h6: 1.150V** **3'h3: 0.900V 3'h7: 1.200V**
LDO_EN	**Signal**	**Input**	**Enable signal (Active high)**

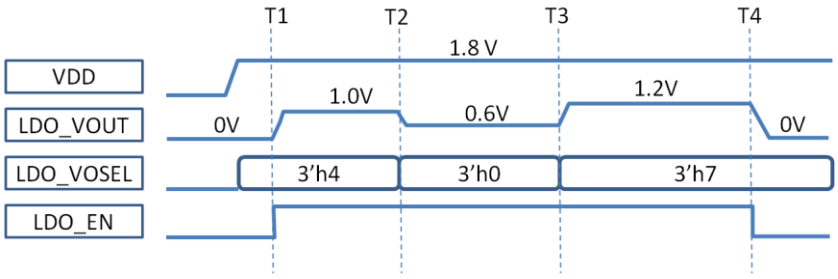

Figure 1. Control sequence waveform of the example LDO design.

To the best of the author's knowledge, there is no way such an LDO can be modeled using UPF commands for the power-aware simulation. The "*create_power_switch*" command cannot be used to model the voltage conversion from one voltage level to another. The voltage level of the input supply net and the output supply net used by "*create_power_switch*" command is the same when the power switch is ON and hence it cannot be used for modeling LDO scenarios where the output voltage is lesser that the input voltage.

B. Real World Design and the Limitation of UPF Supply Resolution

A power supply network as shown in **Figure 2** has been used in our real design. The output of the power switch and the LDO are shorted together and connected to the power domain's primary power. The driving strengths of the power switch and the LDO are similar, and either one can solely supply the domain logic operation. Ideally, either the power switch or the LDO will be enabled for normal operations. This can be handled by "*create_supply_net -reslove one_hot*" semantics. The control sequence waveform is demonstrated in **Figure 3**. Although conceptually, either the power switch or the LDO will be ON, there will be some overlap between them as shown in **Figure 3**. In real scenario, the switching OFF of one and ON of another should not happen at the same time to take care of voltage spikes. Such control sequence is too risky when considering the circuit delay of the supply network. To avoid voltage spikes, first LDO should be turned ON and then the PSW should be turned OFF or vice-versa. In this way, any voltage spike during the transition between the power switch and the LDO can be avoided.

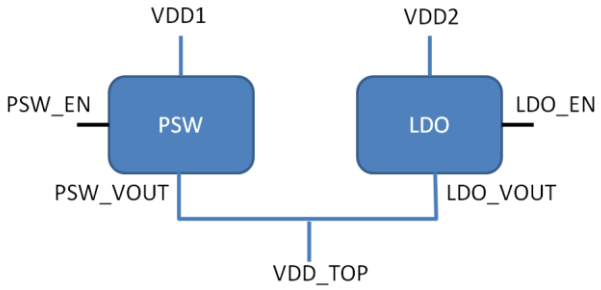

Figure 2. Power domain is supplied by either a power switch or a LDO.

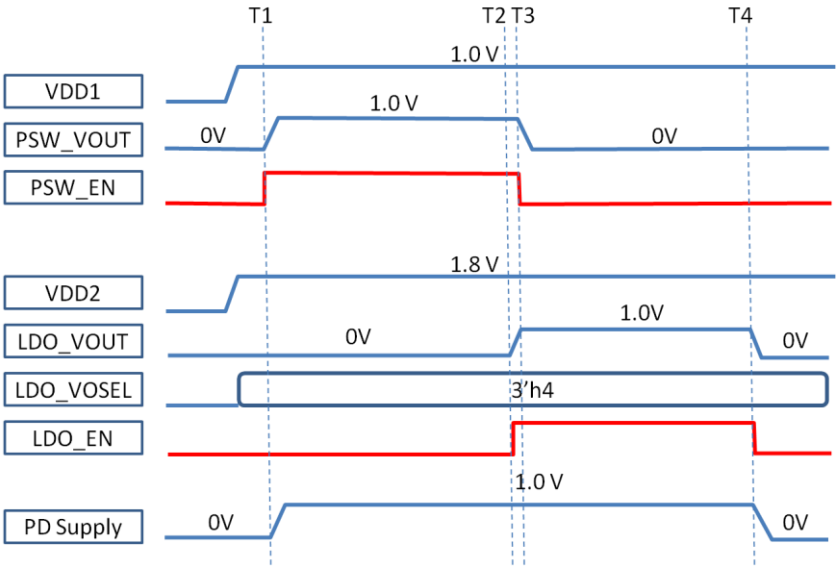

Figure 3. Control sequence waveform for the example design in **Figure 2**.

For modeling the LDO design in **Figure 2** and the control sequence in **Figure 3**, we need a suitable resolution function in UPF. The current UPF supply resolution only supports "*one_hot, parallel, parallel_one_hot*", but none can fulfill the requirements above. For "*one_hot*" resolution, a supply net can be resolved to ON only when there is single supply source ON at any time. However, for the case above, there are certain overlapping scenarios between two supply sources, so this resolution technique is not very useful. For "*parallel*" resolution, a supply net can be resolved to ON when all supply sources are ON. Obviously, this resolution cannot be used either, since we also require ON resolution for a supply net when "either one" supply source is ON. For "*parallel_one_hot*" resolution, at most one root supply driver shall be ON at any particular time with all sources sharing that driver resolved using parallel resolution. This resolution technique too is not useful. Hence, we have to find another way to model such resolution.

III. SystemVerilog Workaround Solution

A. SystemVerilog model for LDO Design

To simulate the LDO design illustrated in **TABLE I** and **Figure 1**, we use a SystemVerilog model in the absence of proper UPF support. The UPF snippet and the behavior model are illustrated in **Figure 4** and **5** respectively. In **Figure 4**, we only describe the input supply ports and nets of the LDO, and we use *connect_supply_net* command to connect **VDD** and **VSS** to the dummy supply ports **VDD_to_SV** and **VSS_to_SV** respectively. To get the information of the supply source on SystemVerilog model side, the dummy supply ports **VDD_to_SV** and **VSS_to_SV** are created in the behavior model. We need to have a UPF file for the LDO to make these connections. This UPF file may seem redundant, since all the power intent including supply ports **VDD** and **VSS** can be directly modeled in the SystemVerilog model. But this file is required to keep consistent macro/IP UPF integration style across the design. That is the UPF integrator can use *load_upf* and *connect_supply_net* command for loading all the macros' and IPs' UPFs and connecting their input/output supplies respectively. Hence, creating the redundant LDO UPF file can make the UPF integration flow consistent.

```
create_power_domain PD_LDO

create_supply_port VDD -direction in
create_supply_port VSS -direction in
create_supply_net VDD -domain PD_LDO
create_supply_net VSS -domain PD_LDO

connect_supply_net VDD -ports {VDD VDD_to_SV}
connect_supply_net VSS -ports {VSS VSS_to_SV}

set_domain_supply_net PD_LDO -primary_power_net VDD -primary_ground_net VSS
```

Figure 4. Example UPF code for the LDO design in Section 2.

Once the UPF supplies are available in the SystemVerilog model, the supply switching and voltage conversion functionality of the LDO can be coded in the behavior model as illustrated in **Figure 5**. The value of LDO output **LDO_VOUT** depends on **LDO_EN** and **LDO_VOSEL** signals which are already available in the behavior model. The dummy supply ports **VDD_to_SV** and **VSS_to_SV** should be declared as output ports with *supply_net_type* in the behavior model as recommended by IEEE. We should note that although the supply directions will have the same meaning if they are declared as *input* or *output* in the behavior model or defined with UPF "*create_supply_port –direction in*/*out*". But, if we declare the dummy supply ports as inputs, we will get multiple drivers (i.e., input supply port **VDD**/**VSS** defined in the UPF and input supply port **VDD_to_SV**/**VSS_to_SV** defined in the model) to the supply nets **VDD** and **VSS**. Hence, it is better to declare them as outputs. In addition, we also have *`ifdef POWER_AWARE_SIMULATION* for hiding all supplies for non-power-aware simulation, since the model will be used for both the power-aware simulation and non-power-aware simulation.

Although UPF has limitations for describing the supply switching and voltage conversion from a source supply net to a sink net, it is very easy and intuitional to model this behavior with SystemVerilog code. The example codes are demonstrated in **Figure 5** from Line 11 to 36. The ON or OFF state of **LDO_VOUT** depends on the state of both **LDO_EN** and **VDD_to_SV** which connects to **VDD**, and the voltage level of **LDO_VOUT** depends on the state of **LDO_EN** together with the value of **LDO_VOSEL**. Since the unit of *voltage* of *supply_net_type* is μV, the real variable **LDO_VOU_r** needs to be multiplied by 10^6 and then assigned to **LDO_VOUT.voltage** (Line 16 in **Figure 5**). The simulation waveform is demonstrated in **Figure 6**, and it shows the desired behavior. We can see that **LDO_VOUT.state** is *FULL_ON* only when **VDD.state** is *FULL_ON* and **LDO_EN** is 1'b1, and the value of **LDO_VOUT.voltage** depends on the value of **LDO_VOSEL** when **LDO_EN** is 1'b1.

Although this solution works, the power intent is only partially captured in UPF. Since the UPF in **Figure 4** does not have any information of **LDO_VOUT**, user can get confused while referring to UPF. Besides, the power state table cannot be captured completely due to the same limitation. Hence, although this solution can simulate the LDO perfectly, it is confusing to the UPF integrator and the verification engineer.

```
 1 import UPF::*;
 2 module LDO (
 3   `ifdef POWER_AWARE_SIMULATION
 4     output supply_net_type VDD_to_SV,
 5     output supply_net_type VSS_to_SV,
 6     output supply_net_type LDO_VOUT,
 7   `endif
 8     input LDO_EN,
 9     input [2:0] LDO_VOSEL
10 );
11     real LDO_VOUT_r;
12   `ifdef POWER_AWARE_SIMULATION
13     always @(*) begin
14         if(VDD_to_SV.state == UPF::FULL_ON && LDO_EN==1'b1) begin
15             LDO_VOUT.state = UPF::FULL_ON;
16             LDO_VOUT.voltage = LDO_VOUT_r*1000000;
17         end
18         else begin
19             LDO_VOUT.state = UPF::OFF;
20             LDO_VOUT.voltage = 0;
21         end
22     end
23   `endif
24
25     always @(*) begin
26         case (LDO_VOSEL)
27             3'h0: LDO_VOUT_r = 0.6;
28             3'h1: LDO_VOUT_r = 0.7;
29             3'h2: LDO_VOUT_r = 0.8;
30             3'h3: LDO_VOUT_r = 0.9;
31             3'h4: LDO_VOUT_r = 1.0;
32             3'h5: LDO_VOUT_r = 1.1;
33             3'h6: LDO_VOUT_r = 1.15;
34             3'h7: LDO_VOUT_r = 1.2;
35         endcase
36     end
37
38 endmodule
```

Figure 5. Example SystemVerilog behavior model for the LDO design in Section 2.

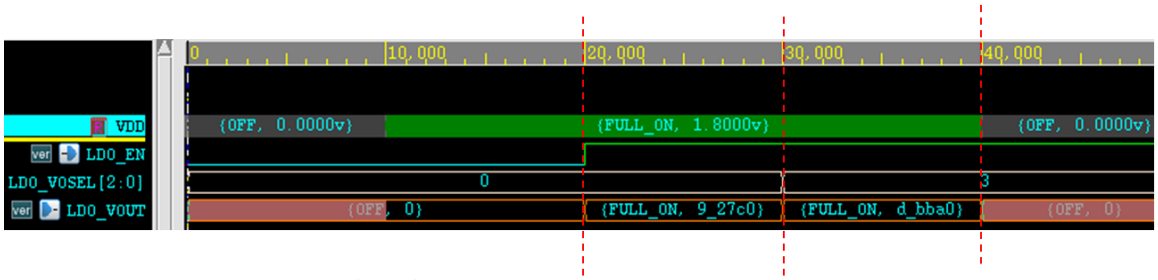

Figure 6. Simulation waveform of the LDO SystemVerilog model in **Figure 5** and the LDO UPF in **Figure 4**.

B. SystemVerilog model for Special Supply Resolution

To conquer the UPF limitation for the supply net resolution mentioned in Section II B, a resolution module as demonstrated in **Figure 7** was created. Conceptually, the output supply net will be resolved to ON when "either" one of the source input supply net is ON. The output supply net is resolved to OFF when both input supply nets are OFF (Line 9 to 12). If one input supply net is ON and the other one is OFF, the output supply net will be resolved to ON with the voltage level of the ON supply net (Line 28 to 35). However, if both input supply nets are ON or partial ON, we will resolve the output supply net to ON or partial ON and the output voltage level will be resolved to the lower one of the input supply nets (Line 14 to 16 and 23 to 26). For the case of either one input supply net partial ON, we will resolve the output supply net to the other input supply net (Line 17 to 22). This is reasonable. Since when we short two different voltage supplies together, there will be a large short circuit current from the higher

voltage source to the lower voltage source and the higher voltage source will drop the voltage level. However, in real world, there is always some resistance inside the voltage sources. Hence, if we short two different voltage sources, the real output voltage level will be less than the higher one and larger than the lower one. The result is governed by the Voltage divider rule as shown in **Figure 9**, and it is very easy to conclude that the output voltage level is between the higher one and the lower one if we analogues the higher voltage source and the lower one to *Vin* and ground in **Figure 9** respectively.

One thing we have to mention is that if there are more than two supply sources required to be resolved, we have to use multiple resolution modules to resolve all of them and the structure will be similar to a binary tree.

```
 1 import UPF::*;
 2 module upf_sn_resolution(
 3    input supply_net_type in_sn1,
 4    input supply_net_type in_sn2,
 5    output supply_net_type out_sn);
 6
 7    (* vcs_always_on *)
 8    always @(in_sn1 or in_sn2) begin
 9       if ((in_sn1.state==UPF::OFF) && (in_sn2.state==UPF::OFF)) begin
10          out_sn.state = UPF::OFF;
11          out_sn.voltage = 0.0;
12       end
13       else if((in_sn1.state!=UPF::OFF) && (in_sn2.state!=UPF::OFF)) begin
14          if (in_sn1.state != UPF::FULL_ON && in_sn2.state != UPF::FULL_ON) begin
15             out_sn.state = in_sn1.state;
16             out_sn.voltage = (in_sn1.voltage > in_sn2.voltage)? in_sn2.voltage:in_sn1.voltage;
17          end else if (in_sn1.state != UPF::FULL_ON) begin
18             out_sn.state = in_sn2.state;
19             out_sn.voltage = in_sn2.voltage;
20          end else if (in_sn2.state != UPF::FULL_ON) begin
21             out_sn.state = in_sn1.state;
22             out_sn.voltage = in_sn1.voltage;
23          end else begin
24             out_sn.state = in_sn1.state;
25             out_sn.voltage = (in_sn1.voltage > in_sn2.voltage)? in_sn2.voltage:in_sn1.voltage;
26          end
27       end
28       else if((in_sn1.state==UPF::OFF) && (in_sn2.state!=UPF::OFF)) begin
29          out_sn.state = in_sn2.state;
30          out_sn.voltage = in_sn2.voltage;
31       end
32       else if((in_sn1.state!=UPF::OFF) && (in_sn2.state==UPF::OFF)) begin
33          out_sn.state = in_sn1.state;
34          out_sn.voltage = in_sn1.voltage;
35       end
36       else begin
37          $display("Error! undefined resolution: in_sn1 %s in_sn2 %s %t",
38             in_sn1.state.name, in_sn2.state.name, $realtime);
39       end
40    end
41 endmodule
```

Figure 7. Supply resolution module for the requirement depicted in **Figure 3**.

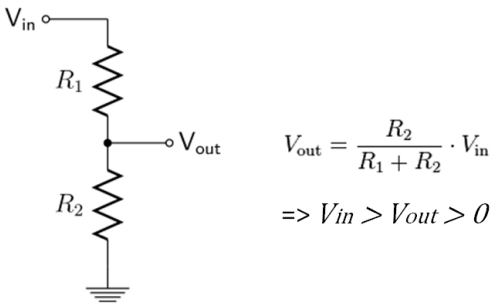

Figure 9. Voltage divider rule.

The next step is to instantiate this module in RTL, and then connects the corresponding UPF source supply nets to the inputs and the output to the target supply net. Comparing to the origin supply network illustrated in **Figure 2**, the new modified supply network with the resolution module is illustrated in **Figure 10** below. Since it is not a good practice to modify the RTL to add the resolution module, we use SystemVerilog *bind* statement to bind the resolution module to the target scope. The example code is demonstrated in **Figure 11**.

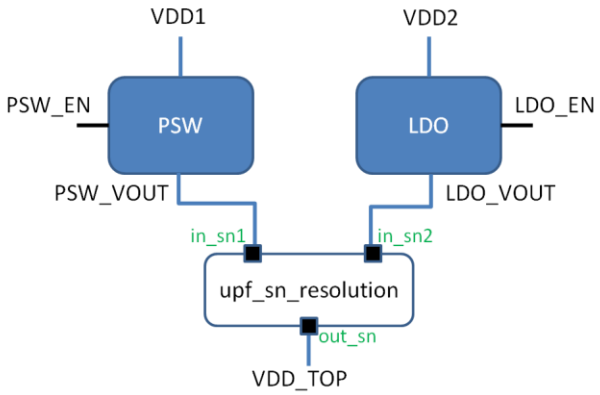

Figure 10. Block diagram for the supply network depicted in **Figure 2** with the supply resolution module.

```
bind top upf_sn_resolution sn_reso();
```

Figure 11. Example code for binding the resolution module to top module.

The corresponding UPF code for the supply network in **Figure 10** is demonstrated in **Figure 12**. As shown in **Figure 12**, a dummy supply net **LDO_VOUT_NET** (Line 9) is created to connect output port of LDO **LDO_VOUT** to the input of the resolution model (Line 17). The other input of the resolution model is connected to the output of the power switch (Line 17) through another dummy supply net **PSW_VOUT**. Finally, the output of the resolution model is connected to the target supply net **VDD_TOP** in UPF (Line 18). The power switch command (Line 24 to 29) is used to model the power switch functionality. The simulation waveform is demonstrated in **Figure 13**. From **Figure 13**, we can observe that the resolution model can resolve the supply correctly at time 30000 (only PSW ON), 40000 (both PSW and LDO ON), and 50000 (only LDO ON).

Although this solution works correctly, it is cumbersome when we adopt this solution into real project. Besides, it also makes the power intent partial and inadequate in UPF when user is not aware of the resolution function, and the UPF file could be difficult to maintain if there are multiple resolution modules instantiated.

```
create_power_domain PD_TOP

create_supply_port VDD1
create_supply_port VDD2
create_supply_port VSS
create_supply_net VDD1 -domain PD_TOP
create_supply_net VDD2 -domain PD_TOP
create_supply_net PSW_VOUT -domain PD_TOP
create_supply_net LDO_VOUT_NET -domain PD_TOP
create_supply_net VDD_TOP -domain PD_TOP
create_supply_net VSS -domain PD_TOP

connect_supply_net VDD1 -ports VDD1
connect_supply_net PSW_VOUT -ports sn_reso/in_sn1
load_upf LDO.upf -scope u_LDO
connect_supply_net VDD2 -ports { VDD2 u_LDO/VDD }
connect_supply_net LDO_VOUT_NET -ports { u_LDO/LDO_VOUT sn_reso/in_sn2 }
connect_supply_net VDD_TOP -ports sn_reso/out_sn
connect_supply_net VSS -ports VSS

set_domain_supply_net PD_TOP -primary_power_net VDD_TOP \
    -primary_ground_net VSS

create_power_switch PSW -domain PD_TOP \
    -input_supply_port { vin VDD1 } \
    -output_supply_port { vout PSW_VOUT } \
    -control_port { psw_en PSW_EN } \
    -on_state { PSW_ON vin {psw_en} } \
    -off_state { PSW_OFF {!psw_en} }
```

Figure 12. Example UPF code for the supply network in **Figure 10**.

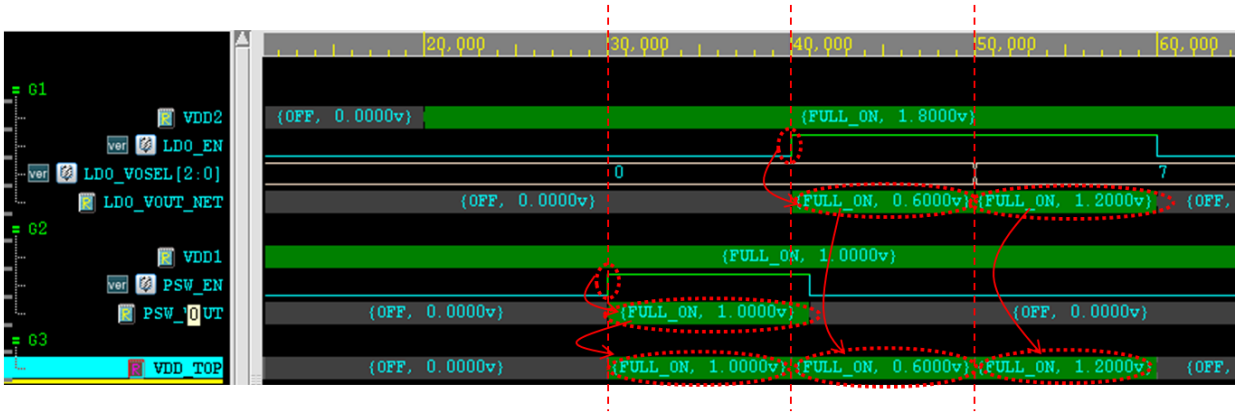

Figure 13. Simulation waveform of the example UPF code in **Figure 12** with the resolution module.

IV. Proposed UPF Solution with Revised Syntax

A. *Proposed UPF Syntax for Modeling LDO Design*

LDO can actually be considered as a special power switch which can not only switch the voltage but also regulate its value based on the control signals. Is it possible to extend the UPF to model such a power switch? *create_power_switch* command already supports multiple ON states today and hence if we can associate every ON state with a voltage value, it will serve the purpose. Our proposed solution is extending *create_power_switch* command to add one argument *–output_voltage* as follows:

create_power_switch *switch_name*

…

[-output_voltage {*state_name voltage_value*}]

With the proposed argument *–output_voltage* for *create_power_switch* command, we can rewrite the LDO UPF demonstrated in **Figure 4** to the code demonstrated in **Figure 14**. The major benefit is that all the LDO power intent including the output supply **LDO_VOUT** (Line 5, 8, and 12), the switching and voltage conversion functionality (Line 16 to 39), and the power state table (not demonstrated in **Figure 14**) can be described completely in the UPF file, so the behavior model can be simplified to an empty module with only input control signals. This solution is much better than the one mentioned in Section III A, and we can get exactly the same behavior as shown in **Figure 6** with the latest version of Synopsys power-aware simulator MVSIM NLP which supports the proposed new argument for *create_power_switch* command. In addition, the simulator can automatically convert all the control signals and switch states of the LDO into *covergroup* helping us to achieve comprehensive low power coverage [2].

```
 1 create_power_domain PD_LDO
 2
 3 create_supply_port VDD -direction in
 4 create_supply_port VSS -direction in
 5 create_supply_port LDO_VOUT -direction out
 6 create_supply_net VDD -domain PD_LDO
 7 create_supply_net VSS -domain PD_LDO
 8 create_supply_net LDO_VOUT -domain PD_LDO
 9
10 connect_supply_net VDD -ports VDD
11 connect_supply_net VSS -ports VSS
12 connect_supply_net LDO_VOUT -ports LDO_VOUT
13
14 set_domain_supply_net PD_LDO -primary_power_net VDD -primary_ground_net VSS
15
16 create_power_switch PSW_LDO -domain PD_LDO \
17     -input_supply_port { vin VDD } \
18     -output_supply_port { vout LDO_VOUT } \
19     -control_port { en LDO_EN } \
20     -control_port { sl0 LDO_VOSEL[0] } \
21     -control_port { sl1 LDO_VOSEL[1] } \
22     -control_port { sl2 LDO_VOSEL[2] } \
23     -off_state { LDO_OFF {!en}} \
24     -on_state { LDO_ST0 vin {en && !sl0 && !sl1 && !sl2}} \
25     -on_state { LDO_ST1 vin {en && sl0  && !sl1 && !sl2}} \
26     -on_state { LDO_ST2 vin {en && !sl0 && sl1  && !sl2}} \
27     -on_state { LDO_ST3 vin {en && sl0  && sl1  && !sl2}} \
28     -on_state { LDO_ST4 vin {en && !sl0 && !sl1 &&  sl2}} \
29     -on_state { LDO_ST5 vin {en && sl0  && !sl1 &&  sl2}} \
30     -on_state { LDO_ST6 vin {en && !sl0 && sl1  &&  sl2}} \
31     -on_state { LDO_ST7 vin {en && sl0  && sl1  &&  sl2}} \
32     -output_voltage { LDO_ST0 0.6 } \
33     -output_voltage { LDO_ST1 0.7 } \
34     -output_voltage { LDO_ST2 0.8 } \
35     -output_voltage { LDO_ST3 0.9 } \
36     -output_voltage { LDO_ST4 1.0 } \
37     -output_voltage { LDO_ST5 1.1 } \
38     -output_voltage { LDO_ST6 1.15 } \
39     -output_voltage { LDO_ST7 1.2 }
40
41 ###port state and PST###
```

Figure 14. Example UPF code with proposed *create_power_switch –output_voltage* command for the LDO design in Section 2.

B. Proposed UPF Syntax for Special Supply Resolution

To conquer the real supply resolution issue mentioned in Section II B (**Figure 2** and **3**), we propose new resolution mechanisms as shown below by extending UPF *create_supply_net* command:

create_supply_net *net_name*
…
[**-resolve** <**unresolved** | **one_hot** | **parallel** | **parallel_one_hot** | **either** | **strong** | **weak**>]

The details of the resolution mechanisms for -*resolve either*, *strong*, and *weak* are listed in **TABLE II**, **III**, and **IV** respectively. The difference between –*resolve strong* and *weak* is whether the output voltage is resolved to the higher one of the supply sources or the lower one when multiple supply sources are ON. The reason for providing these two resolution mechanisms is as mentioned in Section III B, the output voltage level will be between two supply sources when we short two supply sources, since the output voltage is governed by the Voltage divider rule. Hence, if someone prefers to simulate this design as the best case scenario, he can use –*resolve strong*. If some user prefers to simulate this design as the worst case scenario, he can use –*resolve weak*. The difference between –*resolve either* and *weak* is whether the output supply is resolved to *UNDETERMINED* or ON when one source supply is *UNDETERMINED* and the other one is ON. This is another best-case and worst-case scenario consideration.

TABLE II. Resolution mechanism for –*resolve either*.

Supply Source1 (sw1)	Supply Voltage (v1)	Supply Source2 (sw2)	Supply Voltage (v2)	Resolved Supply Net State	Resolved Supply Net Voltage
FULL_ON	1.5	FULL_ON	2.5	FULL_ON	1.5
FULL_ON	1.5	OFF	0	FULL_ON	1.5
OFF	0	FULL_ON	2.5	FULL_ON	2.5
OFF	0	OFF	0	OFF	0
PARTIAL_ON	0	FULL_ON	2.5	FULL_ON	0
FULL_ON	1.5	PARTIAL_ON	0	FULL_ON	0
PARTIAL_ON	0	OFF	0	PARTIAL_ON	0
OFF	0	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	PARTIAL_ON	0	PARTIAL_ON	0
UNDETERMINED	-	FULL_ON	2.5	UNDETERMINED	0
FULL_ON	1.5	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	OFF	0	UNDETERMINED	0
OFF	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	PARTIAL_ON	0	UNDETERMINED	0
PARTIAL_ON	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	UNDETERMINED	-	UNDETERMINED	0

TABLE III. Resolution mechanism for *–resolve strong*.

Supply Source1 (SW1)	Supply Voltage (V1)	Supply Source2 (SW2)	Supply Voltage (V2)	Resolved Supply Net State	Resolved Supply Net Voltage
FULL_ON	1.5	FULL_ON	2.5	FULL_ON	2.5 (maximum of SW1 and SW2 voltage values)
FULL_ON	1.5	OFF	0	FULL_ON	1.5 (voltage value of the root supply driver)
OFF	0	FULL_ON	2.5	FULL_ON	2.5
OFF	0	OFF	0	OFF	0
PARTIAL_ON	0	FULL_ON	2.5	FULL_ON	2.5
FULL_ON	1.5	PARTIAL_ON	0	FULL_ON	1.5
PARTIAL_ON	0	OFF	0	PARTIAL_ON	0
OFF	0	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	PARTIAL_ON	0	PARTIAL_ON	0
UNDETERMINED	-	FULL_ON	2.5	UNDETERMINED	0
FULL_ON	1.5	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	OFF	0	UNDETERMINED	0
OFF	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	PARTIAL_ON	0	UNDETERMINED	0
PARTIAL_ON	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	UNDETERMINED	-	UNDETERMINED	0

TABLE IV. Resolution mechanism for *–resolve weak*.

Supply Source1 (SW1)	Supply Voltage (V1)	Supply Source2 (SW2)	Supply Voltage (V2)	Resolved Supply Net State	Resolved Supply Net Voltage
FULL_ON	1.5	FULL_ON	2.5	FULL_ON	1.5 (minimum of SW1 and SW2 voltage values)
FULL_ON	1.5	OFF	0	FULL_ON	1.5
OFF	0	FULL_ON	2.5	FULL_ON	2.5
OFF	0	OFF	0	OFF	0
PARTIAL_ON	0	FULL_ON	2.5	FULL_ON	0
FULL_ON	1.5	PARTIAL_ON	0	FULL_ON	0
PARTIAL_ON	0	OFF	0	PARTIAL_ON	0
OFF	0	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	PARTIAL_ON	0	PARTIAL_ON	0
UNDETERMINED	-	FULL_ON	2.5	FULL_ON	2.5
FULL_ON	1.5	UNDETERMINED	-	FULL_ON	1.5
UNDETERMINED	-	OFF	0	UNDETERMINED	0
OFF	0	UNDETERMINED	-	UNDETERMINED	0
UNDETERMINED	-	PARTIAL_ON	0	PARTIAL_ON	0
PARTIAL_ON	0	UNDETERMINED	-	PARTIAL_ON	0
UNDETERMINED	-	UNDETERMINED	-	UNDETERMINED	0

With the proposed new resolution mechanisms which has already been implemented in MVSIN NLP, we can create a UPF code as demonstrated in **Figure 15**. To fulfill the requirement depicted in **Figure 3**, we can use –*resolve weak* (Line 8) for **VDD_TOP** and then connect it directly to the output of the power switch rule (Line 22) and the output of the LDO (Line 14). The benefits are obvious comparing the solution used in Section III B (i.e., we don't need to create a resolution module or a bind file to instantiate the module. We also don't need to create dummy supply nets in UPF and connect them). Hence, the UPF supply network connection will be very simple with the proposed resolution mechanisms and designers can even choose the best-case or the worst-case scenario for the resolution result to be simulated in the power-aware simulation.

```
 1 create_power_domain PD_TOP
 2
 3 create_supply_port VDD1
 4 create_supply_port VDD2
 5 create_supply_port VSS
 6 create_supply_net VDD1 -domain PD_TOP
 7 create_supply_net VDD2 -domain PD_TOP
 8 create_supply_net VDD_TOP -domain PD_TOP -resolve weak
 9 create_supply_net VSS -domain PD_TOP
10
11 connect_supply_net VDD1 -ports VDD1
12 load_upf LDO.upf -scope u_LDO
13 connect_supply_net VDD2 -ports { VDD2 u_LDO/VDD }
14 connect_supply_net VDD_TOP -ports u_LDO/LDO_VOUT
15 connect_supply_net VSS -ports { VSS u_LDO/VSS }
16
17 set_domain_supply_net PD_TOP -primary_power_net VDD_TOP \
18     -primary_ground_net VSS
19
20 create_power_switch PSW -domain PD_TOP \
21     -input_supply_port { vin VDD1 } \
22     -output_supply_port { vout VDD_TOP } \
23     -control_port { psw_en PSW_EN } \
24     -on_state { PSW_ON vin {psw_en} } \
25     -off_state { PSW_OFF {!psw_en} }
```

Figure 15. Example UPF code for the supply network in **Figure 2** with the proposed resolution mechanism.

V. CONCLUSIONS

Minimizing power consumption is very crucial for the IC design especially for the mobile applications. Hence, the low power verification is "must" to guarantee correct circuit functionality after adding low power techniques to a design.

Although UPF helps in capturing the power intent of such designs to a large extent, there are some low power techniques which cannot be modeled using UPF like LDOs and special supply net resolution, which have been described in this paper. Initially, we presented our existing solution using SystemVerilog modules connected to the UPF supply network. The existing solution can meet the desired functionality but it leaves some verification holes related to coverage. The existing solution is a bit cumbersome to understand and maintain as the UPF code is unable to have a view of the complete supply network. Therefore, we propose some extensions to the UPF commands "*create_power_switch –output_voltage*" and "*create_supply_net –resolve weak*/*strong*/*either*" to easily model LDOs and special supply net resolution. This will help in making our low power verification more robust. We have collaborated with Synopsys to implement these extensions into the power-aware simulator MVSIN NLP and verified these solutions, and the extensions are only for power-aware simulation. We hope to donate these to the IEEE-1801 committee and hope that these will be accepted and find their place in the next version of the standard.

REFERENCES

[1] IEEE Standard for Design and Verification of Low-Power Integrated Circuits, *IEEE Std 1801, May 2013*.

[2] Shang-Wei Tu, Tom Lin, Archie Feng, and Chen Ya Ping, "UPF Code Coverage and Corresponding Power Domain Hierarchical Tree for Debugging," *Design and Verification Conference (DVCon)* 2015.