

# Is Power State Table Golden?

Harsha Vardhan <sup>#1</sup>, Ankush Bagotra <sup>#2</sup>, Neha Bajaj <sup>#3</sup>

<sup>#</sup>Synopsys India Pvt. Ltd

Bangalore, India

<sup>1</sup>dhv@synopsys.com

<sup>2</sup>ankushb@synopsys.com

<sup>3</sup>nehab@synopsys.com

**Abstract:** Independent of the HDL, UPF [1] provides a consistent format to specify power-aware design intent and semantics for verification and implementation. Using the UPF, one can define power supply network, supply network behavior, and additional logical structures such as retention, isolation, and level shifter needed for a low power design. In the UPF, the Power State Table (PST) describes all the possible power states of a design and is used as a golden reference by implementation tools and static verification checkers. With chips becoming complex, hierarchical power domain distribution methodologies are becoming common. As a typical case in complex low power SoCs, hierarchical PSTs need to be merged resulting in several 1000s of states and depending on merging principles, the resultant PST can error prone. This makes verification and implementation a challenging task. It becomes complicated to ensure that all legal states for the design (architectural intent) are captured in the user intended PST before it is considered as ‘golden’. Only extensive and thorough simulation can ensure whether the PST coverage is complete or not before it can be considered golden. If the PST is over constrained, it will result in structural redundancy in implemented design while a under constrained PST will result in structural violations. UPF does not provide the capabilities to describe PST at abstract design level to describe the golden rules for merging, state completion and coverage semantics.

In this paper, we propose a methodology to qualify PST completeness and coherence with respect to High Level Voltage Relationship Constraints (HLVRC). HLVRC defines the relationship between voltage rails at architectural level. Using HLVRC, all possible low power Legal Design States (LDS) can be generated as per low power design rules. These states can become golden for verification and implementation flows. LDS can be used to validate the user specified PST or merged PST for missing or redundant states. In addition, LDS can also be used to check for structural,

functional and architectural integrity of an implemented low power design.

*Index Terms*—Low Power Verification, Static Verification, UPF,

## I. INTRODUCTION

The demand for low power design methodology led to the formalization of power formats such as UPF which allows the specification of power semantics, originally not possible to specify in HDL. UPF, with its ability to define power states and corruption semantics on them, has made low power flows powerful. At the same time, to manage the design complexity, hierarchical power domain distribution methodology, which exploits the concept of divide and conquer, is becoming common. Bottom-up and top-down implementation flows make power constraints handling a difficult job. In the design flows, as the constraints become complex, there is a need for a high level abstraction view with which all the constraints can be contrasted and cross checked for consistency.

PST is a collection of all possible power states for all input supply nets/ports of a design. Given a PST, implementation tools will implement the design in such a way that no structural (protection), functional and electrical violations could be found in the design. If there is a single PST for an entire design, it is clear that it represents the power states of the whole design. However, if there are multiple PSTs for a design – For example, if the design has multiple blocks, each having its own PST, and also has a top-level PST – it is not clear how these different PSTs relate to each other, and how they compose to represent the power states of the complete design. If

implementation and verification tools employ merging principles, the resultant merged PST may include "legal" states that are actually unreachable due to the design's supply network structure. If PST is specified for a block, it includes supply ports and supply nets defined at and below its current scope. In addition, there is supply network that is virtually created (supply ports, supply nets, connections, power switches) and that links different PSTs, there by posing certain physical constraints. In such a case, the following question arises:

- Do PSTs in their combined effect represent architectural power state intent correctly?

As power intent gets elaborated, transformed and modified in design flows, there is a need for an alternate way to specify abstracted concise constraints (voltage relationships) that can be used for:

- Automatic derivation of elaborated constraints (PST).
- Automatic comparison and consistency checks on user supplied constraints (PST) before they are golden for implementation and static verification flows.

In the following sections, we will highlight how the complex power methodologies open a new set of problems in terms of merging rules of different PSTs defined at different scopes. We will also question the coverage of PST states' defined in the UPF with respect to architectural intent. In the process, we will also recommend some simple rules for writing PSTs which will help in managing PST complexity. To address the completeness and coherence of large and multiple PSTs, we propose a methodology to qualify PSTs with respect to High Level Voltage Relationship Constraints (HLVRC). With HLVRC, we can guarantee that PST transformations are in line with architectural intent. While automatic derivation of PST from HLVRC will help in arriving at comprehensive PST, only extensive dynamic simulation can validate the reach ability of PST states and coverage to that effect. Dynamic low power simulation is outside the scope of this paper.

## II. PST-ITS SIGNIFICANCE

Any power management strategy or technique starts with a PST. It is a user input to describe the operating environment of a low power design. It defines the power mode in which the design works and relationship among various supply rails, supply nets that distribute power to various domains in the design. In effect, it defines all the possible power state combinations, which can exist at the same time during the operation of the design.

The UPF file describes the architectural power intent in two main forms, namely the power supply network and PSTs. The implementation tools use "power supply network" information to do "physical" power network implementation whose consistency can be verified by static checkers. To implement and check the protection requirements (Level Shifters, Isolation cells, Retention Registers and Always On Buffers), implementation and static checkers rely on the values that can be applied on each supply net, and valid combinations of supply nets. This information is referred to as the "Power State Table" (PST). The main focus of implementation and static verification tools is to ensure that the design is implemented and verified in such a way that no protection violations can be found in any power state in the user supplied PST.

## III. PST- COMPLEXITIES

### A. Exponential State Space

The total number of states in a PST can grow exponentially with respect to the total number of supply nets, and number of voltage states on each of them. For example, in a design with 20 power domain, each domain being independently ON or OFF (2 states) will theoretically lead to  $2^{20}$  states in the PST. If each domain can have different ON voltages (ex: 1.0, 1.2, 0.9), then the state space expands even more. Even if the 'practical' or 'legal' state space is much less than the theoretical space, it is still of the order of 100s of thousands in large SoCs. This typically leads problems such as:

- User errors in coding large PSTs
- Redundancy in PST states leading to huge run times and bad structural inference
- Under constrained PST states leading to bad structural inference

## B. State Reachability

An implementation tool synthesizes using PSTs as constraints, making low power verification comprehensive. This concept is similar to the idea of synthesis of functionally verified RTL where comprehensive verification is done using assertions and simulation techniques. The first objective of verification is to exercise the Power State Table. Assuming that static verification yields a clean result, we can assume that in a steady multi-voltage state, there are no further obvious electrically hazardous conditions. The corner cases may exist that need to be uncovered by dynamic verification. However, before we use dynamic verification, we have some basic functionality to verify [3].

Simulation and formal techniques are needed to prove that a given PST state is actually reachable or not. For example, given the PST in Figure 1, we might want to detect whether the State1 is reachable or not?

	neta	netb	netc
State1	ON	OFF	OFF
State2	ON	ON	ON

Figure 1: PST

In other words, it is required to explore the relationships of nets {neta, netb, netc} to see whether {neta:ON; netb:OFF} can force {netc:OFF} or not. Similarly all such permutations need to be exercised. In simple cases, it can be solved, but it is not always easy to track when PST state space grows. Furthermore, when any of these nets is controlled by switches with control functions driven by registers, it becomes a sequential use case which can be solved theoretically by a formal or a simulation tool [4].

## C. Hierarchical Flows

To support hierarchical flows, it is necessary to allow PSTs to be defined for top and block level scopes. If implementation and verification tools employ merging principles, the resultant merged PST may include "legal" states that actually are unreachable due to the design's supply network. Based on supply

network resolution and PST merging principles, the resultant PST may be:

- Over constrained – redundant states are generated
- Under constrained – valid states are lost.

In such cases, users might want to review the merged PST and correct power network or port states or individual PSTs to ensure that the resultant PST is in line with the architectural intent, else it will lead to incorrect implementation. For instance, bufferization of the signals that cross power domains must be performed with the knowledge of the "rail ordering" and the use of always on buffers which can remain ON while the power domain is shutdown. The rail ordering is essentially derived from resultant PSTs in cases of hierarchical flows. This demands strict rules in merging principles and strong syntax and semantic checks before merged PSTs can be considered golden for implementation and static verification.

Also, in hierarchical flows, if some blocks are not allowed to change, structural violations are impossible to resolve if they happen within fixed blocks. Those errors can only be prevented by enforcing certain relationship among PSTs specified at different hierarchy.

## IV. HIGH LEVEL VOLTAGE RELATIONSHIP CONSTRAINTS

In order to address the complexities highlighted in the above section, we introduce HLVRC to capture high level low power architectural intent of design. HLVRC is an abstract representation of the following:

- (1) hierarchical rail order relationships
- (2) power network dependencies

This serves the need for specifying abstract and concise constraints (voltage relationships) that can be used for (a) Automatic derivation of elaborated constraints (PST) for implementation and verification (b) automatic comparison and consistency checks on user supplied constraints (PST) before they are golden constraints for implementation and static verification

As power intent gets transformed in the design flows, HLVR can be used to ensure that the transformations are in-line with the architectural intent.

The following code example introduces various components that defines the format for HLVR

```
define_rail_name <rail_name> -value <voltage_value>

set_rail_order -order <number> -rail <rail_name> -rail <rail_name> ....

set_rail_constraint -main_rail <rail_name> -dependent_rail <rail_name> ....
```

A. Definition: *define\_rail\_name*

**define\_rail\_name** defines the rails present in the design and their respective voltage values as per high level design intent.

B. Definition: *set\_rail\_order*

**set\_rail\_order** is used to indicate the order of the rails. '0' order number indicates the rail is more 'on' than all other rails. The increasing order number indicates the rails are more relative off. A particular order number can have multiple rails, but a rail should only be present in one order. Multiple rails can be added to same order in case they are equivalent and independent

C. Definition: *set\_rail\_constraint*

**set\_rail\_constraint** is used to define the dependency among rails of different order. There can be multiple dependent rails on a signal main rail.

Let us take an example of a topology as mentioned in Figure 2[2]. The design has 20 voltage domains that have a hierarchical relationship. The domain C5 is mostly ON while the relative-OFF domains are mentioned in lower rows. The arrows capture the dependencies or independence of various voltage rails.

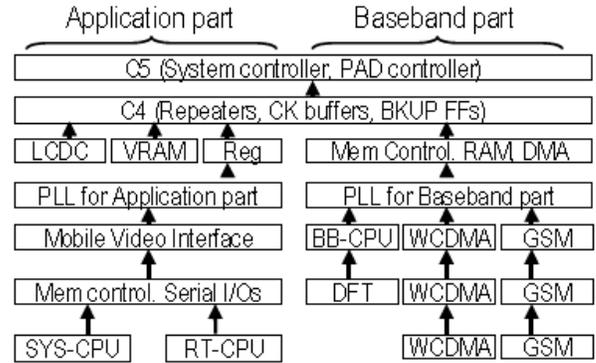


Figure 2: Domain Voltage Hierarchical Relation [2]

For such a design, the low power architecture is so complex and yet the architectural power intent is very concisely and crisply captured in a diagram. If power intent were to be detailed, the formats may run into 10s of thousands of lines. Assuming that each domain can be ON or OFF, the theoretical state space is  $2^{20}$ . Using the voltage relationships and power network information should bring down the state space, but still for such complex designs, the PST state space is still a huge number which cannot be hand written or will be prone to errors.

```
define_rail -name C5 -value {1.0} -value {OFF}
define_rail -name C4 -value {1.0} -value {OFF}
define_rail -name LCDC -value {1.0} -value {OFF}
define_rail -name VRAM -value {1.0} -value {OFF}
define_rail -name REG -value {1.0} -value {OFF}
define_rail -name PLL_app -value {1.0} -value {OFF}
.....
.....
.....
set_rail_order -order 0 -rail C5
set_rail_order -order 1 -rail C4
set_rail_order -order 2 -rail LCDC -rail VRAM -rail REG -rail MEM_ctrl
set_rail_order -order 3 -rail PLL_app -rail PLL_base
set_rail_order -order 4 -rail Mobile_V -rail BB_CPU -rail WCDMA_1 - rail
GSM_1
set_rail_order -order 5 -rail MEM_serial -rail DFT -rail WCDMA_2 -rail
GSM_2
set_rail_order -order 6 -rail SYS_CPU -rail RT_CPU -rail WCDMA_3 -rail
GSM_3

set_rail_constraint -main_rail C5 -dependent_rail C4
set_rail_constraint -main_rail C4 -dependent_rail MEM_ctrl -
dependent_rail VRAM -dependent_rail REG -dependent_rail LCDC
set_rail_constraint -main_rail MEM_ctrl -dependent_rail PLL_base
set_rail_constraint -main_rail PLL_base -dependent_rail BB_CPU -
dependent_rail WCDMA_1 -dependent_rail GSM_1
set_rail_constraint -main_rail BB_CPU -dependent_rail DFT
set_rail_constraint -main_rail WCDMA_1 -dependent_rail WCDMA_2
set_rail_constraint -main_rail GSM_1 -dependent_rail GSM_2
set_rail_constraint -main_rail WCDMA_2 -dependent_rail WCDMA_3
set_rail_constraint -main_rail GSM_2 -dependent_rail GSM_3
.....
.....
.....
```

The above HLVR snippets show how HLVR can be written for such complex architectures in a few lines. Having captured the intent using HLVR, golden PST can be deduced automatically. If hierarchical flows are used then block PSTs can be inferred automatically.

**V. PST-MANAGEMENT (SOME BEST PRACTICES)**

In this section, simple guidelines for writing PSTs in UPF are highlighted. These guidelines will help to address PST's complexity. The assumption is that the implementation and verification tools will have consistent merging principle which is outlined below.

Merging Principle: A "block" PST cannot make a legal state which is illegal according to a "top" PST. Neither can a "top" PST make a legal state that is illegal according to a "block" PST. Any state that is illegal according to any PST must be illegal. The final set of legal states is those that are not ruled out by any other PST.

In addition, the strong syntax and semantic checks should be as per the guidelines mentioned below.

(1) Multiple PSTs per scope

When multiple PSTs are defined in one scope, the intersection of the entries of all the PST becomes PST of that scope. This practice makes it much easier to construct a PST instead of user writing a large PST

For example, assume that supply port SP1, SP2, SP3, SP4, SP5 and SP6 are all defined in scope "top/mid", where SP1 SP2 and SP3 are related, and SP4, SP5 and SP6 are related. Specifically, the user intended PST table is described in Figure 3.

	SP1	SP2	SP3	SP4	SP5	SP6
State1	S1	S1	S1	S4	S4	S4
State2	S2	S2	S2	S4	S4	S4
State3	S3	S3	S3	S4	S4	S4
State4	S1	S1	S1	S5	S5	S5
State5	S2	S2	S2	S5	S5	S5
State6	S3	S3	S3	S5	S5	S5
State7	S1	S1	S1	S6	S6	S6
State8	S2	S2	S2	S6	S6	S6
State9	S3	S3	S3	S6	S6	S6

Figure 3: PST

The same PST could be defined by the intersection of the following two tables as specified in Figure 4.

	SP1	SP2	SP3
State1	S1	S1	S1
State2	S2	S2	S2
State3	S3	S3	S3

	SP4	SP5	SP6
State1	S4	S4	S4
State2	S5	S5	S5
State3	S6	S6	S6

Figure 4: Multiple PSTs

- (2) Use of don't cares or wild cards for similar rails in a PST state will make PST more concise and more readable

For the above example, the following PSTs can also represent the same PST but in a more precise manner with the use of wild card.

	SP1	SP2	SP3	SP4	SP5	SP6
State1	S1	S1	S1	*	*	*
State2	S2	S2	S2	*	*	*
State3	S3	S3	S3	*	*	*

	SP1	SP2	SP3	SP4	SP5	SP6
State1	*	*	*	S4	S4	S4
State2	*	*	*	S5	S5	S5
State3	*	*	*	S6	S6	S6

Figure 5: PSTs with Wild Card

There are totally 6 PST entries in above two tables, but it takes 9 entries to define the PST in one single table. It is easy to see that use of multiple tables could significantly reduce the total number of entries in the PST.

- (3) Establish PST relationships using direct references.

There is no need to re-define the intermediate states and values in PST implying a relationship with other PST's when these states can be referenced directly in the PST. This practice will also help tools, and otherwise the manual merging of PSTs

- (4) Restrict supply-net availability to have optimal number of supply nets in PST.

Supply nets should be declared optimally and reused as much as possible. There is no need to re-declare a net in every scope when it can be re-used in another scope.

## VI. CASE STUDY

Through our case study, we demonstrate the various issues that can be addressed by HLVRC. We will analyze a topology specified in Figure 6.

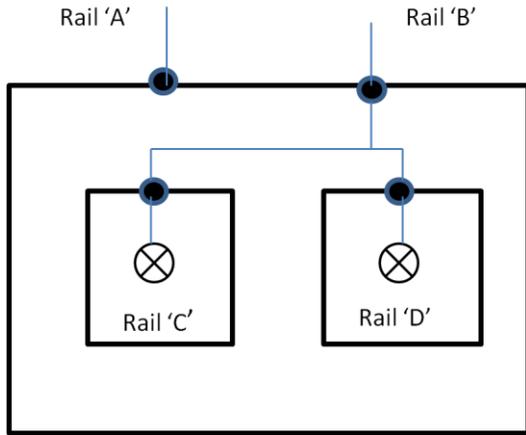


Figure 6: Case Study Topology

The HLVRC to capture the architectural intent for this topology is specified below:

```
define_rail -name A -value {1,2} -value {OFF}
define_rail -name B -value {1,2} -value {OFF}
define_rail -name C -value {1,2} -value {OFF}
define_rail -name D -value {1,2} -value {OFF}
set_rail_order -order 0 -rail A -rail B
set_rail_order -order 1 -rail C -rail D
set_rail_constraint -main_rail A -dependent_rail C -
dependent_rail D
```

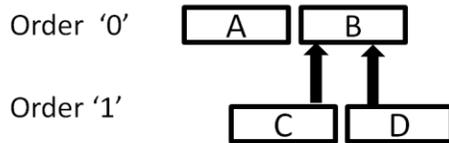


Figure 7: Case Study Ordering

The individual state tables generated for case study topology are specified in Figure 8.

	A	B
PS1	ON	OFF
PS2	ON	ON

PS3	OFF	ON
PS4	OFF	OFF

	B	C
PS1	ON	ON
PS2	ON	OFF

	B	D
PS1	ON	ON
PS2	ON	OFF

Figure 8: Block Level PSTs

The golden PST is derived from HLVRC in Figure 9. In the PST the '\*' indicates don't care as per the best practices from previous section. The maximum possible number of states for this topology is 16 but with the HLVRC inference, the states were reduced to 10.

	A	B	C	D
State1	ON	ON	*	*
State2	ON	OFF	OFF	OFF
State3	OFF	ON	*	*
State4	OFF	OFF	OFF	OFF

Figure 9: Golden PST Inferred

## VII. APPLICATION OF HLVRC

### A. Syntax Checks For Rails

Ideally all the rails present in HLVRC should be present in the PST defined in the UPF. If a rail is not specified in the PST defined in UPF and is present in HLVRC, an inconsistency is observed and an error is flagged for incompleteness.

For example, if the PST for case study topology is as specified below, the analysis for rail D as specified in HLVRC is missing and it should be flagged as an error.

	A	B	C
State1	ON	ON	*
State2	ON	OFF	OFF
State3	OFF	ON	*
State4	OFF	OFF	OFF

Figure 10: PST

### B. Over Constraint/ Under Constraint PST

A PST defined in UPF is over constraint when the number of states in the PST is more than the states which can be inferred from the design. This might lead to testing the design in the states which never occurs, and results in redundant protection devices, and other low power constructs.

For example, if the PST defined in UPF for case study topology has state specified below

	A	B	C	D
State7	ON	OFF	ON	OFF

Figure 11: PST State

This above state is not a valid state as rail C and D are dependent on rail B and are also of lower order than B. So the design will never reach this state. On co-relation with golden PST specified in Figure 9, this redundancy can be detected before implementation and verification flows.

A PST defined in UPF can also be under constraint when there are some states which are possible in a design but are not present in PST. This leads to wrong implementation results as tools will never implement the entire intent and verification will not be done on the design for missed out state. Understanding the relation between the rails, it is feasible to comment on all the possible permutations of different rails, which a design can possess.

For example, if the PST defined in UPF for case study topology does not have a valid state specified below.

	A	B	C	D
State6	ON	ON	ON	OFF

Figure 12: PST State

If this state is not present in PST, then design will never be tested for this state.

### C. Merged PST

With complex low power topologies emerging, merged PST as a concept has become quite common in the hierarchal flows, where the blocks are individually verified with their respective PST

defined for block level UPF's. This approach is helpful in managing small PST at the top level. But when these block level PSTs are merged at top level, the merged PST is error prone as various blocks of PST states gets missed in the merging process because they do not find the overlap in the other PST's.

For example, if the PSTs specified on left side are from various blocks of UPF's and are merged to define the top level UPF as specified below.

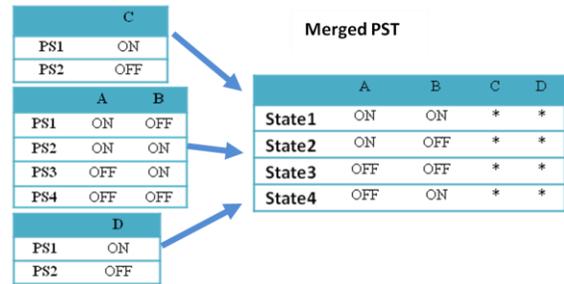


Figure 13: Merged PST

The merged PST is over constraint with reference to golden PST specified in Figure 9 and has the unnecessary states present, which are not possible in the design. If this merged PST is used by the verification tool, it might end up giving wrong results with respect to the requirement of the isolations cells in the design. If this PST is used by the implementation tool, it will insert the isolation cells in the design which will never be required as per the functionality of the design.

## VIII. CONCLUSION

In this paper, we have acknowledged the problem of considering the PST defined in UPF as golden in view of complex low power SoCs with hierarchical PST with each PST having a large no of states. To address the problem, we have presenting the HLVRC framework. It defines the guidelines for a reference check for PST. The framework also provides the ability to define the overall architectural intent of the design that can be used as metrics in terms of quality, coverage, and signoff. In doing so, it brings down the turnaround time for fixing a design issue and effectively portraying the accurate status design.

## **IX. Limitations**

At present, the framework does not honor the multiple voltage states for a supply net.

## **X. Future Work**

Our future goal is to incorporate the HLVRC framework in static verification tools and multiple voltage state support for a supply net. We also intend to define more appropriate semantics to capture architectural design constraints.

## **XI. Reference**

1. *Unified Power Format (UPF 2.0) Standard [Draft Version]*; IEEE Draft Standard for Design and Verification of Low Power Integrated Circuits, IEEE P1801/D18; 23rd October, 2008.
2. Hierarchical Power Distribution and Power Management Scheme for a Single Chip Mobile Processor. DAC, 2006.
3. Low Power Methodology Manual,  
<http://www.synopsys.com/community/partners/arm/pages/lpmm.aspx>
4. SNUG 2011- "UPF power state table verification methodology using MVSIM", Christophe Chavel, ST Ericson,  
[https://www.synopsys.com/news/pubs/snug/france2011/b2\\_clavel\\_paper.pdf](https://www.synopsys.com/news/pubs/snug/france2011/b2_clavel_paper.pdf)