The Case for Low-Power Simulation-to-Implementation Equivalence Checking

Himanshu Bhatt	John Decker	Hiral Desai
(MCS-LP solution)	(Architect-LP solution)	(SMCS-LP validation)
Cadence Design Systems	Cadence Design Systems	Cadence Design Systems
hbhatt@cadence.com	jdecker@cadence.com	hiral@cadence.com

ABSTRACT

Power-aware design differs from conventional design in both well-understood, as well as, subtle ways. For a typical design today, the RTL describes the functional intent, drives the implementation process and relies on equivalence checking to assure the intent carries through to silicon. In power-aware design, the power format file – either CPF or IEEE 1801 (UPF) – is the specification for the power intent. The functional intent becomes the RTL and the power intent file. While many tools in the end-to-end flow can read the intent, how do we verify that each of these tools has interpreted the intent in the same way? Where does the equivalence checking need to take place for power-aware design?

Keywords

Low Power, Functional Verification, Equivalence Checking, CPF (Common Power Format), UPF Unified Power Format, IEEE 1801

1. INTRODUCTION

Since both the implementation flow and the simulation flow read the same power format files and RTL, most engineers would assume that information integrity is sustained. Therein lies the subtlety. Logic simulation is an abstraction of the design in which power rails and physical placement, among other details, are not necessary. The CPF and 1801 standards are designed to ensure that the tools have the same starting point, but that doesn't guarantee that they have interpreted that information correctly. There is no formal proof that the simulation model of the power intent is the same as what was implemented. The same problem exists for RTL, but the collected experience over the past generation has resolved the discontinuities. In today's low power context, the pace of design and the cost of silicon failure do not permit the electronics industry a similar "baking" time for power-aware design.

While the issues may be subtle, missing them can result in non-functioning silicon. One very important feature of power-aware designs which use the power shutoff functionality is isolation. The rules for isolating pins/ports are specified in the power format file. Typically, the simulator reads the power format file during RTL elaboration and calculates the isolation information in preparation for the simulation test. When the test runs, the isolation results can be validated using coverage and/or waveform checking. Similarly, the equivalence checker would typically check for the isolation placements throughout the design implementation using its formal analysis engine.

There are a number of rules on the precedence of multiple isolation rules, the ordering of isolation in back to back cases, and even the impact of the physical location on where isolation is inserted. But there is no formal proof of whether the tools in the verification chain made all of the same decisions as the tools in the implementation chain. While one might be tempted to think that this would only be caused by tool "bugs", the reality is actually more subtle. There are corner cases in both UPF and CPF where the specification is not detailed enough to define exactly what should happen in all possible cases. Each software design team (simulation and implementation) makes decisions on how to implement these cases, and it is possible that they can both have "valid" implementations that differ from one another. While every effort is taken to ensure consistency, existing methods fall

short in proving exhaustively that the two interpretations are correct. This problem exists in single vendor as well as mixed vendor flows. For single vendor cases, each vendor strives to be internally consistent, but the implementation and verifications teams are separate groups, running very different tasks on very different engines. In mixed vendor cases the problem is even worse, as the only communication is usually indirect finger pointing about whose problem is causing the difference. In both cases, there is a clear need for a formal proof to ensure that both tools are consistent.

This paper provides three examples where there were differences detected that were not tool interpretation related. In one case, the difference was related to a fundamental difference in RTL modeling between simulation and synthesis. The other case was caused by poor methodology decisions and the third one is a complex case of back to back isolation.

This paper will describe a method to successfully address this discontinuity. It discusses a technique which applies formal analysis and equivalency checking to prove that the isolation insertion implemented by simulation matches the insertion done in the implementation flow. While the focus of this paper is on isolation, the same approach could be used to validate almost all low-power intent between the implementation and simulation flows. This minimizes the risk of elusive bugs escaping into silicon.

2. Low Power isolation checking (SIM2LEC)

In a low power flow, the simulation and implementation tools uses the information in the *create_isolation_rule* CPF commands to identify the net segments that must be isolated, their isolation values, and the conditions under which isolation should happen. The simulator models isolation implicitly by adding virtual isolation cells to the RTL design. The implementation adds actual physical cells to the design to implement the power intent.

The concern is that this isolation inserted by the simulation tools could be different from synthesis or equivalency checking. Providers of low power tools spend a great deal of effort to obey the same CPF semantics for all aspects of low power design. However, up until now, there has been no formal technique to do this verification.

To ensure that the isolation inferred by both the simulation tools and the other downstream tools in synthesis, equivalency checking, and place and route match, there needs to be a technique for doing this closed loop verification. This technique is named for the tools that were used in the original development and from hereon would be referred as SIM2LEC isolation checking.

2.1 SIM2LEC methodology

The SIM2LEC methodology is to use formal equivalency checking to compare the design with the isolation insertion defined by a simulation output file to the same design with isolation natively inserted in the equivalency checking tool per the CPF rules. If there is any difference in what the simulator infers as isolation and the equivalency checkers insertion based on CPF isolation rules, non-equivalence will result.

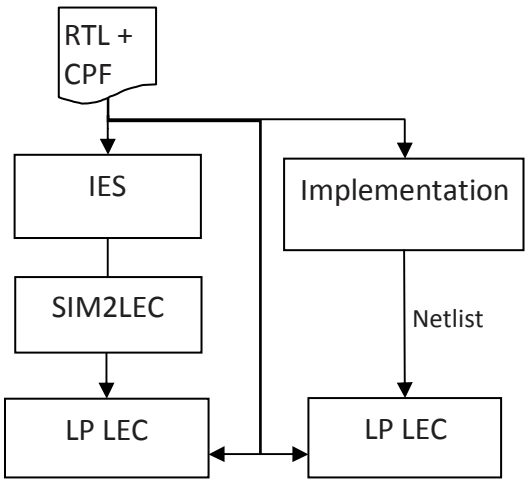


Figure 1: Same golden Intent and LP LEC used to verify implementation and simulation

The flow above describes the process as implemented and used in the Cadence low power solution, but the general concept applies to any low power flow.

2.1.1 Alternative solutions considered

The SIM2LEC methodology provides a clean formal solution to this complex problem. It was not, however, the original solution as proposed by our customers. Originally, the customer asked for assertions to be generated automatically based on the CPF file, which could run during simulation to verify the low level details of the isolation insertion. We have also had requests for the simulator itself to generate this type of assertion.

This approach had a number of methodology issues that lead to it being rejected. Using the simulator to generate this type of assertion was not considered viable, since the solution requires an independent verification of the power intent. The primary reason for rejection was that the assertions were not a formal proof and relied on the quality of verification environment; if a specific isolation case was not covered by a test, it could lead to issues not being detected. Since the low power environment is typically software controlled, the amount of simulation time required to provide the needed coverage was prohibitively large. The process also added a large number of low level assertions that could impact the overall performance.

The SIM2LEC flow using equivalency checking enables a closed loop methodology where the simulation model of the design is verified using the same tool that verifies the implementation. The formal proof of the intent is the key aspect that ensures what was simulated, matches with what was implemented for the design.

2.2 What can the flow detect?

2.2.1 Case1 – Methodology issues

In theory, the same CPF should be used for simulation as for implementation. But in practice, designers may take short cuts. When the change is limited to the physical constraints an implementation designer may sometimes feel a simulation run is not required. But isolation is complex; a change in physical location of the isolation cell can result in a change of behavior due to the power supply for the cell as well as the order of isolation cells when placed in series. In those situations, if the physical related change in the CPF is not re-simulated, a false-positive result in simulation will occur. Without the SIM2LEC methodology, this subtle discontinuity, among others, would only be detected by gate-level simulations. Since gate-level simulations are very limited, it is unlikely that this issue would be detected.

Following the recommended methodology would ensure that all changes to the power intent are simulated. The SIM2LEC flow provides a good method to verify this.

2.2.2 Case2 - Feedthrough paths

Simulation and synthesis operate on the design in very different ways; even what is considered an operation differs. Take the following example:

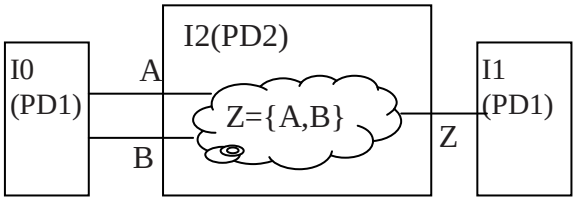


Figure2: Feedthrough example

In this example, PD1 is a power shutoff domain. In synthesis, the concatenation of Z is just a wire, so there is a feedthrough path from I0 to I1, since both I0 and I1 are in PD1 there is no need for isolation.

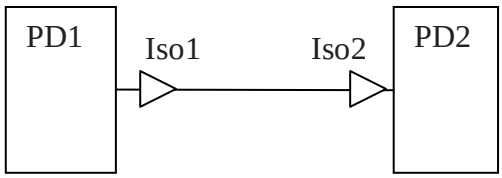
Originally, the simulator treated the concatenation as an operator. To the simulator this path was not seen as a feedthrough path, and so isolation was inserted between I0 and I2. This difference in how the same RTL construct was treated, lead to functional differences in the behavior of the design.

Was this a bug in the simulator? No, most simulators take the RTL code very literally. The concat is an operator in the verilog specification, so the simulation view was not “incorrect” in the

general case. But from a full LP flow perspective this caused issues.

2.2.3 Back to back isolation

One more complex case of isolation is what we term back to back isolation. In general, in a CPF flow there is a single isolation cell placed on a domain crossing. But in the back2back isolation case there are two isolation cells on the path. The diagram below describes the situation.



CPF

```
Create_isolation_rule –name Iso1 –from PD1
  –isolation_target from
  –isolation_condition X
Create_isolation_rule –name Iso2 –to PD2
  –isolaiton_target to
  –isolation_condition Y
```

UPF

```
Set_isolation iso1 –domain PD1 –applies_to
outputs  -source_clamp 0 –isolation_signal X
Set_isolation iso2 –domain PD1 –applies_to
inputs   -sink_clamp 1 –isolation_signal Y
```

Figure3: Back to Back isolation example

In this situation, the order of the isolation depends on the isolation location specified in the power intent. In most cases the simulator does not need to worry about location other than for assigning the correct secondary domain, but in this case it makes a difference in the logic function.

Iso1 – Location	Iso2 – Location	Isolation order
To (fanout)	From (fanin)	Iso2->iso1
From(fanin)	From(fanin)	Iso2->Iso1
To(fanout)	To(fanout)	Iso1->iso2
Parent	Parent	Iso1->Iso2

Table1: Back to Back isolation order

In this situation, the isolation value seen at the input of PD2 can differ based on the location specified. For example, the cases of both locations being set in the "to" domain or the "from" domain have a different order of isolation. Neither power intent specification (UPF/CPF) is detailed enough to specify the exact order. It is entirely conceivable that a verification engineer with little physical design experience could come to a different conclusion on the order than an implementation engineer. Since the spec is not detailed enough, both engineers would feel they have valid isolation and yet the results would be functionally different.

2.2.4 Use Cases

There are many more of these complex situations that can also cause differences of simulation versus implementation. It is difficult for R&D in the same company to be sure that all possible cases have been covered. When the flow involves multiple tool vendors, the task is even less likely for them to be in sync.

The SIM2LEC flow provides a formal method to prove the consistency between these very different tools in the flow. Each tool is optimized for the task that it performs. There will be differences because of this. Low power is lucky; the type of modeling done for low power can be checked formally. Frankly, it would be ideal if we could formally check the simulation and implementation view for all the RTL. But the general solution is much harder (probably impossible) to solve.

In the case above, the solution was to enhance the simulator (IES) to provide feedthrough analysis that would be more consistent with the implementation flow.

2.3 Use Model

The user needs to add the following internal option to *irun* or *ncelab* in IES:

-lps_genclp_iso [filename]

This option generates an output file after the simulation run, which is then consumed by Conformal Logical Equivalence Checking (Conformal LEC) to check the simulator's isolation placement.

The -lps_genclp_iso option extracts the isolation information in the log file and generates a command file that can be consumed by Conformal LEC. This file provides information on the simulator's implicit isolation results. Conformal then uses the RTL design and the CPF file to determine its own isolation placement and compares this with the simulator's implicit isolation to verify that they are consistent.

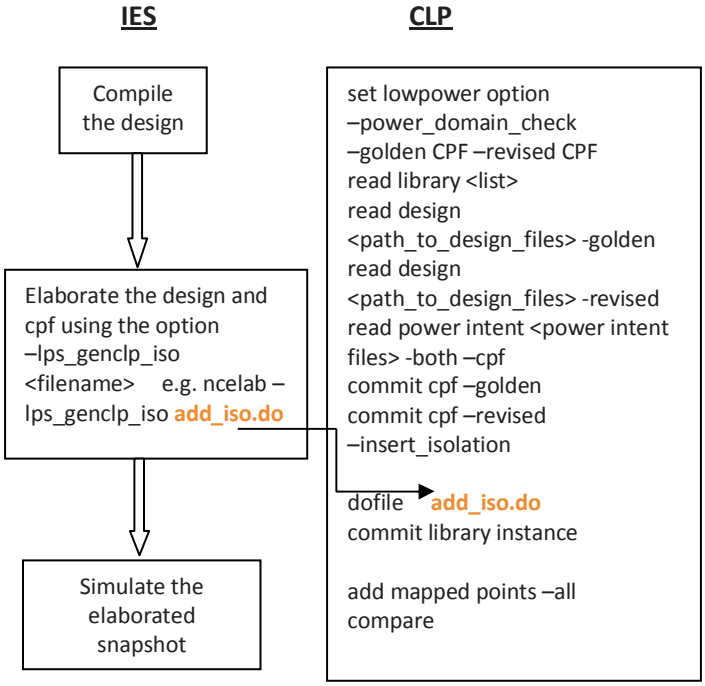


Figure4: SIM2LEC isolation flow

2.4 SIM2LEC Flow Details

Example

The following example illustrates the SIM2LEC isolation flow:

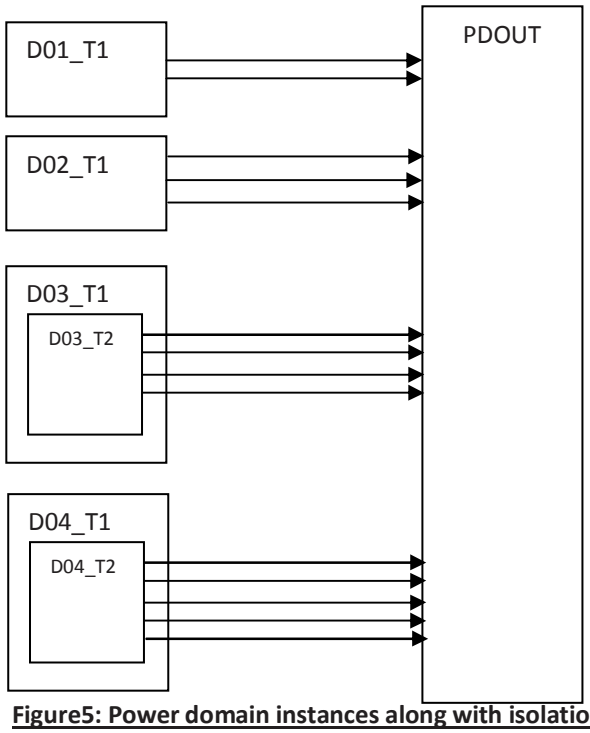


Figure5: Power domain instances along with isolation constraints

Example CPF file

```
## Power Domains ##
create_power_domain -name PDAON -default
create_power_domain -name PDON \
-instances {PDOUT}
create_power_domain -name PD01 \
-instances {D01_T1} \
-shutoff_condition {PCU/O_PSW\[A\]} \
-base_domains PDAON
create_power_domain -name PD02 \
-instances {D02_T1}  \
-shutoff_condition {PCU/O_PSW\[A\]} \
-base_domains PDAON
……..
……..
## Isolation Rules ##
create_isolation_rule -name iso01   \
-from PD01 -to PDON \
-isolation_condition "PCU/O_ISO_A_ " \
-isolation_output high
create_isolation_rule -name iso02 \
 -from PD02 -to PDON \
 -isolation_condition "PCU/O_ISO_A_" \
 -isolation_output high
```

```
create_isolation_rule -name iso03 \
-from PD03 -to PDON \
-isolation_condition "PCU/O_ISO_A_ " \
-isolation_output high \
 -pins {D03_T1/D03_T2/D03_A_C20\[A\] \
D03_T1/D03_T2/D03_A_C21\[A\] }
……..
……..
```

IES options for generating "do" file for CLP

```
irun <design files> \
      <tb files> \
       -lps_cpf <cpf file> \
       …….
       …….
       -lps_genclp_iso add_iso.do
```

IES output file "add_iso.do"

```
//DATE: 11/07/11
//TIME: 15:26:07
//FILE generated with TOOL:     ncelab  11.10-b020

//CPF source: ./NET01_1/CPF/CPF01_COMMON.cpf:47
add library instance -type OR
-target D01_T1/D01_A_A[A]
-connection {i0 D01_T1/D01_A_A[A]}
-connection {i1 "PCU/O_ISO_A_" }
-prefix NET01_1_CPF_CPF01_COMMON.cpf:47
//CPF source: ./NET01_1/CPF/CPF01_COMMON.cpf:49
add library instance -type OR
-target D02_T1/D02_P_B0
-connection {i0 D02_T1/D02_P_B0}
-connection {i1 "PCU/O_ISO_A_" }
-prefix NET01_1_CPF_CPF01_COMMON.cpf:49
add library instance -type OR
-target D02_T1/D02_A_B1
-connection {i0 D02_T1/D02_A_B1}
-connection  {i1 "PCU/O_ISO_A_" }
-prefix NET01_1_CPF_CPF01_COMMON.cpf:49
add library instance -type OR
-target  D02_T1/D02_A_B0
-connection {i0 D02_T1/D02_A_B0}
 -connection {i1 "PCU/O_ISO_A_" }
-prefix NET01_1_CPF_CPF01_COMMON.cpf:49
……..
……..
```

Script for running CLP using this "add_iso.do"

```
set log file ./clp.log -replace
set lowpower option -power_domain_check -golden CPF
-revised CPF
read library -cpf <file.cpf> -statetable -liberty
read design  -sv <design> -root <design top> -gol
read design  -sv <design> -root <design top> -rev
read power intent <file.cpf> -both -cpf
commit cpf –gol
commit cpf -rev -insert_isolation
dofile add_iso.do
commit library instance
go
compare
exit –f
```

CLP comparison output

```
==========================================
// 68 compared points added to compare list
==========================================
Compared points     PO     DFF      Total
------------------------------------------------------------------------------
Equivalent          40     28       68
==========================================
// Command: compare
==========================================
Compared points     PO     DFF      Total
------------------------------------------------------------------------------
Equivalent          40     28       68
==========================================
```

3. CONCLUSIONS AND FURTHER DEVELOPMENTS

The IE2CLP flow for isolation comparison provides closed-loop verification between the simulation and implementation tools. It ensures that no bugs in a low power design escape silicon.

The following can thus be concluded based on this flow:

- Power formats, such as CPF, unify intent across the flow
- Implementation and verification both read the same isolation data, but have different abstractions in which to apply the data
- Simulation to implementation adds formal rules to find differences in interpretation when the power-format data is applied in each separate flow

As a part of future development, another aspect which is being worked upon is the state retention consistency check between IES and CLP. The existing SIM2LEC flow is being enhanced to ensure

that the state retention registers between IES and CLP are consistent.

4. ACKNOWLEDGMENTS

The authors would like to thank the entire Low Power simulation and implementation team without whose support this flow would not have been a success.

5. REFERENCES

[1] Low-Power Simulation Guide, ver. 11.1.
[2] Si2 Common Power Format Specification, ver. 2.0
[3] www.powerforward.org