Static power-management verification of Cypress's PSoC® Programmable System-on-Chip for embedded systems

Johnie Au
Cypress Semiconductor
198 Champion Court,
San Jose, CA
+1 408 943 2149
jca@cypress.com

Prapanna Tiwari
Synopsys Inc
3075 W. Ray Road
Chandler, AZ, 85226
+1 650 215 8606
prapanna@synopsys.com

ABSTRACT
Power management is a necessity today to achieve the power-targets of a design. Verification is a key aspect in ensuring the design is functional on silicon while aiming to achieve its power targets. It includes both dynamic and static verification. This paper will focus on static verification and cover the challenges faced and solutions applied in static verification of Cypress's PSoC® Programmable System-on-Chip for embedded systems. It covers various stages in the design flow - from power-intent definition to sign-off verification/validation. It will cover how the programmable macros pose a challenge in low-power static verification but also in defining their power attributes. It will highlight the pitfalls a verification team must look out for when doing static verification and the desirable features of a low power static checker solution. The paper will give an overview of a solution deployed on the PSoC® chips and future challenges/enhancements being considered to meet the complex power architectures of advanced low-power SoCs.

Keywords
Power-gating, isolation, level-shifter, power-switch, power-state table (PST), UPF, Voltage Scaling, Dynamic Voltage Scaling, Adaptive Voltage Scaling, Dynamic Voltage and Frequency Scaling, Adaptive Voltage and Frequency Scaling, Power Management Unit (PMU), retention, DRC

1 INTRODUCTION

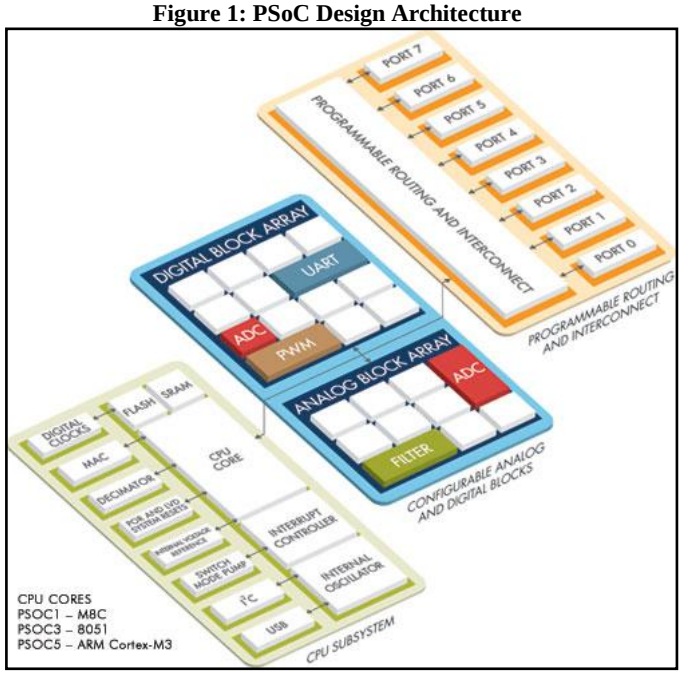
Advanced power-management techniques are a necessity for a product to remain competitive in the current consumer portable market, as designs pack more features on progressively smaller die sizes. Power management techniques primarily aim at keeping alive the regions of the chip which are functional and operating the functional regions at a voltage no higher than needed to achieve the performance targets. Based on the region alive in the chip, each power-mode allows only a subset of features to be used. As the device changes from one feature to another, it now requires changing the power-mode, based on functional, power and performance requirements of the chip. To add to the complexity, the power-management scheme of a design must be programmable (to save cost) as it's often aimed for multiple end products with different power architectures.

With this increased complexity, the problem space to be covered by verification also increases significantly. The design must be verified in all its functional and power modes and various configurations (possible use models/architectures). Verifying such a low-power SoC solely by dynamic verification techniques (simulation) will lead to significant increase in verification time/effort and impact design schedules. Static verification becomes very critical in power-managed designs, with its ability to catch bugs without vectors with much lesser setup/runtime cost as compared to dynamic verification. It can clean up the structural, semantic and architectural issues in the design/power-intent before simulations execute towards achieving the power-aware functional coverage goals. Used appropriately, it can even be improved to catch bugs which conventionally would require simulation to catch them. This paper aims to discuss various aspects of static power-management verification in detail and share relevant examples from the Cypress' PSoC experiences. This paper assumes basic knowledge of power techniques and discusses mainly the static verification flow.

2 DESIGN OVERVIEW

Cypress's PSoC family of designs chips are a low-power solution for embedded systems, combining an 8-bit microcontroller, flash memory, and SRAM with programmable analog and digital blocks. PSoC® devices contain up to 16 digital and 12 analog blocks. These digital and analog blocks are programmable enabling designers to create and quickly change advanced mixed-signal embedded applications. The same programmability that - helps a designer is a challenge for low-power verification as it significantly increases the number of power modes possible for the macro/chip. Some of these macros can have multiple power supplies, adding another level of complexity to the power-management verification, as discussed later in the paper. PSoC designs can be used in a variety of power-management techniques depending on the power goals of the end application. Power techniques discussed in this paper would include power-gating, retention, substrate biasing and dynamic voltage scaling with associated low-power structures required to enable these techniques –viz.-isolation, retention, level-shifter and power-switch cells. The power-management static verification for this design was done using a flow built around tool MVRC (Multi Voltage Rule Checker) from Synopsys Inc.

Figure 1: PSoC Design Architecture



3 STATIC VERIFICATION

Regardless of whether a power-management technique is used in a design, various static verification checks in any design flow can broadly be categorized in to following three buckets:

1. **Structural checks**, which include verifying the presence, placement, connectivity and context of different design structures. This could verify the cell instances in a post synthesis verilog netlist or LV/DRC rules on a post-layout design view (e.g. GDS).
2. **Functional checks**, which include semantic checks on the design elements based on the data available. This could verify clock domain partitioning rules at RTL stage, control signal polarities on a post-synthesis netlist, or any heuristic rules which are design to ensure correct functional behavior. Functional checks may not always be sign-off and often based on heuristics as they enforce static rules aimed at ensuring correct dynamic behavior. Certain assumptions have to be made about dynamic scenarios that a design would face and verify the design accordingly. These assumptions may run into an occasional exception. Finding functional bugs is traditionally the domain of dynamic verification but with increasingly complex designs, static verification is becoming instrumental in catching as many bugs as possible before going starting dynamic verification.
3. **Equivalence checks**, which include comparison of functional behavior of two separate design views. Unlike structural and functional checks which apply rules to a single design view and verify it to be a functional design, equivalence checks take two design views which are individually expected to be functional and compare them to check if they are identical in behavior.

Power management techniques impact all three aspects of static verification in a design flow.

New power structures in the design (such as isolation, level-shifter, retention, power-switch cells) require **structural checks** to ensure they are inserted and connected correctly and in the right context.

Power domain partitions now require **functional check** on power-control signals to ensure they are connected with right polarity and routed through appropriate regions to ensure functionality in all power-modes. Functional checks are also necessary on non-low power signals (e.g. clock-enable, clock-networks, reset etc) to ensure access to them is available in all active regions of the design in a given power mode.

Equivalence checks must also be power aware to ensure any difference in power-connectivity and associated functional behavior between the two design views being compared is caught. Without knowledge of power-partitions and connectivity an equivalence checker will not be able to comprehend the effect of power-modes on functional behavior of the active and inactive regions of the design.

Although all the static checks can be covered by dynamic verification with appropriate RTL, Netlist or Spice simulations, however, the completeness of the verification is dependent on the coverage achieved by the tests run. Given infinite time and resource, dynamic verification could theoretically catch all the bugs in the design. But in today's competitive markets, product development schedules are only getting shorter with time. In such a market, static verification is indispensable in catching design bugs early in the flow and with minimal effort. A good static verification methodology can be the difference between a successful and a failed product.

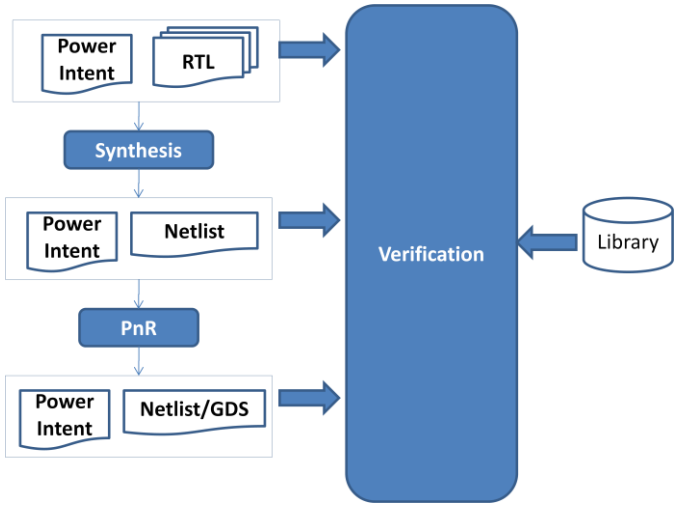
This is even more applicable to designs using power-management techniques. A unique aspect of power-management techniques is that majority of rules to be enforced is dependent on the power-technique being used, irrespective of the design type. For example, any design using power-gating will require isolation cells at the output of the OFF domains in the design. There can be numerous kinds of isolation cells used but the fact that isolation logic of some kind is necessary to protect the ON regions from the OFF logic is true, irrespective of the design category. There are a few exceptions to this rule but in general it applies to all aspects of power-management static verification. This in turn implies that all except the power-sequencing and certain functional completeness related issues could be theoretically caught using static verification. This can be boon to a design team trying to incorporate power-management techniques into an existing design on a tapeout schedule which is equal, if not shorter, than the last time.

In summary, a well planned static verification methodology in a power-managed design can help catch bugs earlier in the flow with minimal effort and to some extent reduce the dependence on dynamic verification for identifying low-power bugs.

4 POWER MANAGEMENT STATIC VERIFICATION

From power-architecture definition to tapeout, all stages in the design flow require static verification. The subsequent sections will cover for each stage in the design the kind of static checks that must be done to ensure a smooth design cycle.

Figure 2: Typical Design Flow



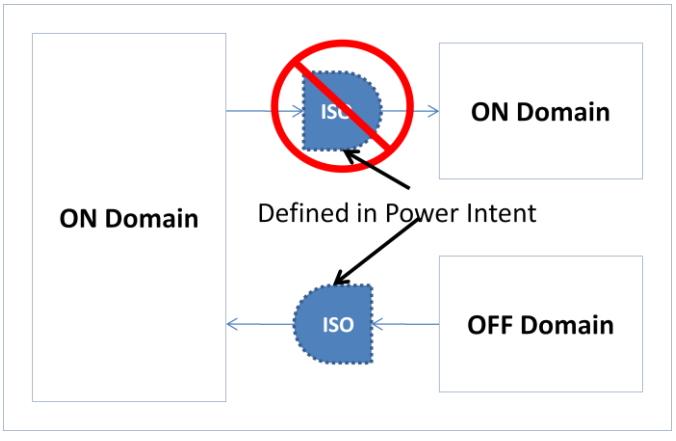
4.1 Power Intent Definition & Validation

The first step in any design flow is to define the power intent for the design. Power intent is captured as a separate text file that accompanies the design. It include information about power domains, chip low-power states, control signals, isolation/level-shifter/retention/power-switch strategy details etc. MVRC supports two mechanisms – UPF and an internal power intent format. There are also other industry formats available which can capture the power information. For sake of generality, we'll refer to it as the power-intent file. Power intent file captures both structure and semantics of the power architecture. To avoid legacy RTL changes, power structures (like isolation/retention/level-shifter/always-on-buffer/power-switch cells) are initially defined in the power intent file. Hence it's safe to say that power-intent file not only capture the power information but also part of the design structures which will be inserted into the design at a later stage in the design flow. The semantics part of power-intent identifies primarily the various legal power modes and voltage values for different parts of design and the supply network.

Once the power intent file has been defined, the first stage in verification is to validate this power intent to ensure that combined with the RTL, it provides a complete and coherent power description which can be expected to work on silicon. At this early stage in the design, following checks must be performed on the power intent of the design:

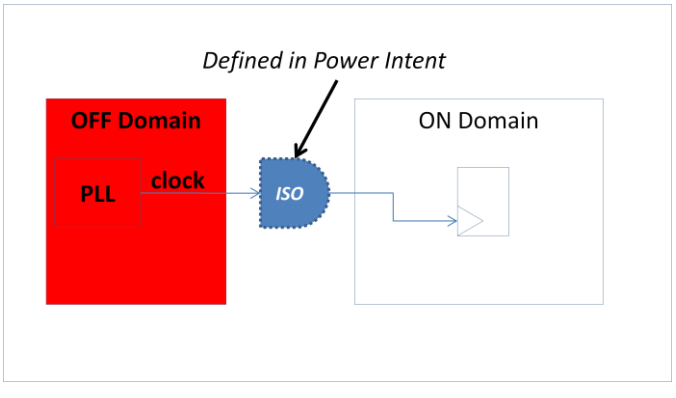
1. Structural checks should be performed on the power-intent itself to ensure it defines the necessary isolation and level-shifter requirements for the designed based on the power-modes of the chip.
    a. These checks should also ensure that the isolation and level-shifter requirements defined are no less AND no more than necessary. Having redundant (more than necessary) power structures in the design can also lead to functional bugs. E.g. isolation cell between two ON domains would tie down the signal when it needs to be functional and toggling.

Figure 3: Redundant Isolation in Power Intent



2. Functional checks should be performed to ensure power architecture itself is not causing bad silicon by design.
    a. For each power mode of the design, ensure all necessary signals required to keep the ON region functional is not being driven by any of the OFF domains in that mode. E.g. if a clock PLL's power domain is OFF in a chip level power-mode while the power-domain of the register that uses the clock is ON, it is an obvious functional issue even if that clock signal has an appropriate isolation cell defined on it.

Figure 4: Bad partitioning (Functional error)



    b. Power management state, transition and sequences for the design should be critiqued to ensure no unsafe rail transitions are likely to happen. This is not a sign-off check but more of a best-practices analysis to avoid any rush current related issues later in the flow.
    c. Library for the design must be validated to ensure it has the necessary cells and necessary attributes/info about them, required to successfully implement the power-intent on the current design. This check would help catch any library issues upfront and avoid late surprises. Fairly accurate analysis of the library can be done even before a design reaches the synthesis stage.

Power structures in RTL

For various reasons outside the scope of this paper, some design teams may choose to insertion isolation cells and level-shifters in RTL itself. In such cases, the power-intent should still capture the definition for those cells as they would have been if the cells didn't exist in RTL. The static verification tool should then ensure that the behavior defined in power-intent matches with that inserted in the RTL. This is crucial to ensure a correct dynamic verification and avoid late surprises in the flow. Catching a missing or incorrect isolation/level-shifter cell in RTL is dependent on the tests being run and assertions being checked. It is possible for dynamic verification to miss such bugs and establish a golden reference design with this bug. In such cases all subsequent stages of the design flow will continue on the same incorrect assumption and a bug might go undetected until very late in the design flow. Static checks are the best way to catch any power structure related bugs early in the design cycle without any ambiguity.

4.2 Post –Synthesis Static Verification

As design progresses through various stages, the power intent, captured as a separate file originally, progressively gets implemented and becomes part of the design itself. A synthesis tool can take the RTL and the power-intent file and synthesize the power structures necessary for isolation, level-shifter and retention requirements of the design. It may need to insert always-on buffers while connecting signals depending on the power partitions and any placement planning info fed to the tool. Always-on buffers are primarily dual rail buffer cells which have a back-up supply that helps them remain functional even when their primary supply turns OFF. They are mainly used in connecting feed through signals through OFF domains. Power switches are typically not synthesized at this stage as power-routing information is unavailable. With these changes done to the design (in addition to the regular logic synthesis), static verification must now be done on the output netlist to ensure it is electrically correct and is consistent with the power intent.

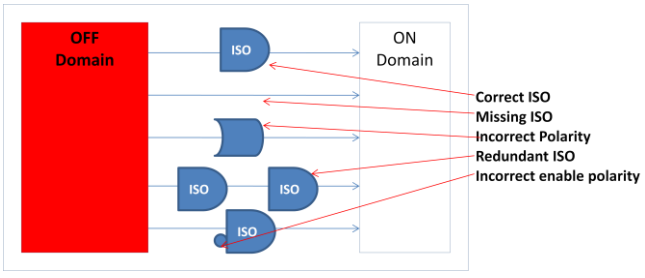
Structural checks

Power structures in the design

Regardless of what has been captured in the power-intent file, the first check that should be done on a netlist is to verify if the power structures inserted in the netlist are necessary and sufficient for functional silicon.

Power modes for the design should be the guideline on which these checks should be based on. Each power mode indicates the voltage value of each power domain in the design. Combining this with the design connectivity information, static verification should infer the isolation and level-shifting requirements for each power domain in the design. These requirements should then be compared against the cells instantiated in the synthesized netlist. All instances of isolation (ISO) and level-shifters (LS) cells must be verified to ensure there are no occurrences of missing, redundant, incorrect-type, incorrect-output polarity (ISO only), incorrect voltage range (LS only) and incorrect enable connectivity (ISO only).

Figure 5: Examples of errors on isolation cells



Power Intent V/s Design

In addition to verifying the design, a comparison must be done between the power-intent and the design to ensure they are consistent in the isolation and level-shifter requirements for the design. Cells inserted in the design must be consistent with the isolation and level-shifting directives power-intent. Any difference must be reviewed in detail.

Occasionally a situation may arise where the power-intent may NOT be consistent with the isolation/level-shifting requirements inferred purely based on power-modes of the chip. This happens when there are macros which have inbuilt isolation or level-shifting logic or have some purely analog pins on their interface. Such information must be captured in the library as cell attributes and also in power-intent file as overriding directives from the user. Without this information, MVRC would end up giving false violations basing its analysis purely on power-modes of the chip.

Another common exception possible in this check is when the following three items are not all in sync:

1. Isolation/level-shifting directives based on power-modes
2. Explicit isolation/level-shifting directives from user
3. Isolation/level-shifter instances in the netlist

If (1) and (2) are same but different from (3), it is likely to be a synthesis issue and appropriate logs should be checked.

If (1) and (3) are same but different from (2), careful review must be made to identify if (2) has been defined incorrectly or the synthesized netlist has incorrect structures and (2) should be an overriding directive for netlist

If (2) and (3) are same but different from (1), careful review must be done to identify the root cause of these differences. Common reason for such differences is cells in the design with exception conditions such as inbuilt isolation/level-shifting or pure analog pins which don't require isolation/level-shifter cells on them.

Multi-rail macros and challenges in PSoC designs

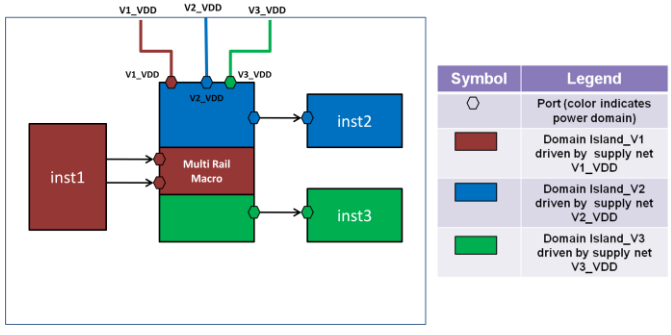
Cypress' PSoC design had a unique challenge were the several multi-rail analog and digital blocks instantiations. Each of these macros has multiple power pins including both digital and analog voltages. Consequently, interface of these macros consists of various ports associated to its different power-pins of the macro. For static verification, the following aspects had to be accounted for:

1. Information about supply rails connected to all the power pins of these multi-rail macros must be defined. This helps MVRC correctly associate logic pins of a cell to its power pins. Multiple instances of the same macro may be

instantiated in different power contexts (i.e. with different power pin connectivity).

2. Information about any inbuilt isolation and level-shifter logic must be captured. This will help avoid any false errors when verifying the design. This information cannot be inferred based on the verilog netlist and must be defined as an explicit attribute of the cell in library or in power-intent file.

Figure 6: Multi-rail Macro



Without this information, MVRC could make incorrect assumptions about the voltage level for the macro ports and give spurious errors. However, with this information provided, all the exceptions can be avoided and a clean analysis can be done.

Retention cell checks

If retention scheme is being used in a design, following checks must be done:

1. Each retention region marked in the power-intent file must be checked in the design to ensure that all registers in that region are of the retention cell type.
    a. Each retention cell instance should be verified to ensure its primary and back-up power pins are connected as directed in the power intent file.
    b. Each of retention cell instances should be verified to ensure save/restore connectivity as per the power-intent file.
    c. Voltage values for supply rails connected to the retention cells should be analyzed for all power modes to ensure the cell will be able to perform save/restore functions appropriately.
2. Each non-retention region in the design should be checked to ensure no retention cells have been instantiated. A redundant retention cell instance could lead to functional errors.

Power Switches

Power switches are typically not inserted at this stage and hence no associated checks need to be done yet. These checks should be done post-layout netlist, once the cells have been inserted and connected in the design.

Functional checks

Functional checks are necessary at this stage to ensure the critical signals in the design have not been connected up in a manner that causes them to be corrupted or isolated unexpectedly or be connected with a wrong polarity.

Keep Alive Signals

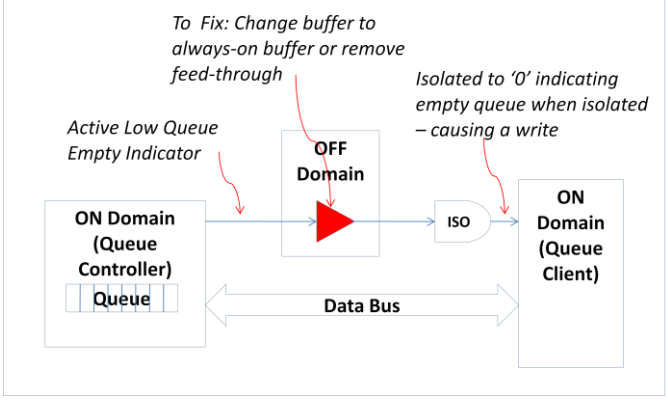
For each power domain in the design, there will be a certain list of critical control signals which are necessary for its functionality both in power-ON as well as power-OFF modes. In power-ON mode, the keep alive signals are the ones required to ensure functional behavior (e.g. clock, reset, clock-gating signal etc) while in power-OFF mode, the signals help ensure the block is appropriately isolated from the rest of the design (e.g. power-enable signal, isolation control signal, clock-gating signal etc). It is a generic description that has been given here and the list of these keep-alive signals will be dependent on the design's architecture and end application. The keep-alive signals can broadly be bucketed into following categories:

1. All power control signals (e.g. isolation/power/save/restore enable signals)
2. All clock/reset control signals (e.g. clock, clock-enable, clock multiplexer selector, reset signals etc)
3. Boot strap signals for any all power domains
4. Any external power related interrupts indicating wake-up or shutdown events
5. Scan control signals (depending on test methodology)
6. Any form of enable/disable signals or indicator signals should be reviewed as candidates for these checks. (E.g. an empty indicator signal for a queue if partitioned/corrupted/isolated incorrectly could incorrectly flag an empty queue to destination logic initiating a queue write when infact the queue is full)

Although many heuristic rules can be applied, the process of identifying these signals is predominantly manual beyond the common categories mentioned above. However, this is a onetime effort per project and is definitely well spent. Identifying power domain boundaries at RTL stage itself can give an idea of the control signal networks which need to be tested to ensure they are alive in all power modes.

Having identified these signals, various checks must be done to ensure they are accessible and alive in all the necessary regions of the design, in all power modes of the chip. The simplest check is to trace the network of these signals from source to each leaf level cell-pin in the netlist and ensure that access to that signal is available at each cell-pin in all power-modes defined in the power-intent. Even a simple ill-connected buffer is sufficient to cause functional issues. If missed in static checking, these bugs could be caught in simulations if the specific test vector is run where the scenario gets simulated.

Figure 7: Functional bug on queue status indicator



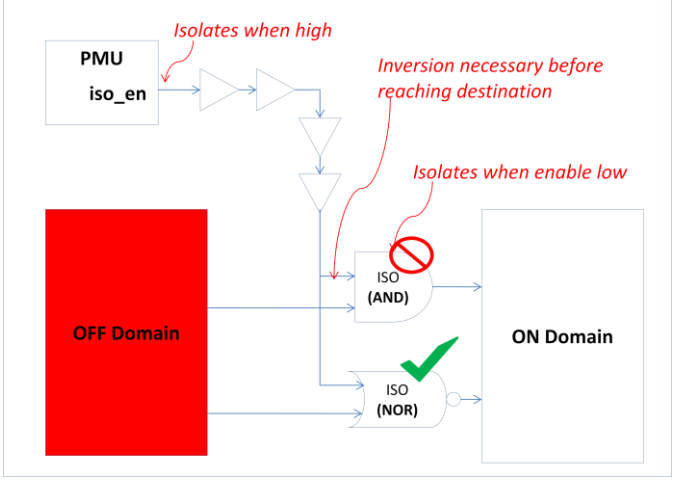
Signal Polarity

In addition to the keep-alive checking, all power control signal networks must be traced to ensure they reach destination cell pins with expected polarity. Two factors should be checked:

1. Active polarity of the control signal as specified in power-intent file (e.g. isolation enable signal isolates when high)
2. Polarity expected at the cell pin (e.g. AND gate isolates when isolation enable pin is low).

Taking an example of isolation enable signal for a domain, simply ensuring that the signal reaches the isolation cell with the same polarity as at the source is not sufficient. An active high isolation enable signal (isolates when high) connected to an AND based isolation cell (isolates to low when enable low) would cause isolation error. The polarity of the enable signal arriving at the cell-pin should be validated against the functionality of the cell. The same signal connected to an NOR based isolation cell (isolates to low when enable high) would give correct functionality. As evident, functionality of the isolation cell must also be considered when doing the polarity checks.

Figure 8: Polarity validation



Equivalence checks

At this stage, it is strongly advisable to run the equivalence checks between the RTL+power-intent and Netlist+power-intent design views. Structural and functional checks can ensure each individual design view has no low-power bugs, however, only equivalence check can guarantee that indeed the original power-intent and functional behavior with which RTL verification was done and a golden reference was established, has been implemented into the post-synthesis netlist. Tools like Formality from Synopsys can do a power-aware equivalence and ensure original power intent has not been modified in any way by the synthesis step.

4.3 Post –Layout Static Verification

Post layout design database would contain the power-routing/connectivity for the design. Power-switches, if specified in power-intent file, would be instantiated into the netlist. This is the final stage of static verification in the design flow and also the most accurate. All power intent information has been implemented in the design and is available in the design database. MVRC need not rely entirely on the power-intent specified by the user.

Input formats

It is essential to capture the input formats available at this stage of the design. In addition to the netlist formats (verilog/vhdl netlists) commonly used by static verification tools in pre-layout design flow, various physical databases are also available – such as – Milkyway library, LEF/DEF, GDS. These databases can help provide more information about the power-connectivity of the design as compared to a verilog netlist. Static checkers can use that information to add to the data extracted from the verilog netlists. Typically static verification is done using a PG-netlist, which is a verilog netlist with power-ground pin and associated connectivity present in the netlist itself.

Power Connectivity checks

First check to be done at post-layout stage is to verify the power/ground (PG) pin connectivity against the connectivity specified in the power-intent file. All power and ground pins for each cell in the design should conform to the information captured in the power-intent file. Any discrepancy, even if it does not cause a functional error, should be highlighted as an error. Entire design flow, from RTL to layout phase, has relied on the power-intent file for their checks and analyses. This includes any manual checks/modifications/reviews done by engineering teams and checks/analyses done by any software tools in the flow. A difference, even though not necessarily an error as per the rules applied by a static verification tool, could break the inputs/assumptions on which the earlier steps were based on. Differences in PG pin connectivity v/s power-intent file should be reviewed carefully to identify the root cause.

Multi-rail checks offer unique challenge as their power/ground pin connectivity can't all be inferred based on instance level power-domain partitioning. Each power pin can have a unique supply-rail connection directive for it in the power-intent file. The power-connectivity checks must take this into account to avoid incorrect analysis. In PSoC designs, with their multiple programmable digital and analog blocks instantiated across the design, this was a key check to be done to ensure all macros are connected correctly to the appropriate supply rails in the design. Different instances of the same macro could be connected to completely different supply rail networks on the same design, based on the context in which they are being used.

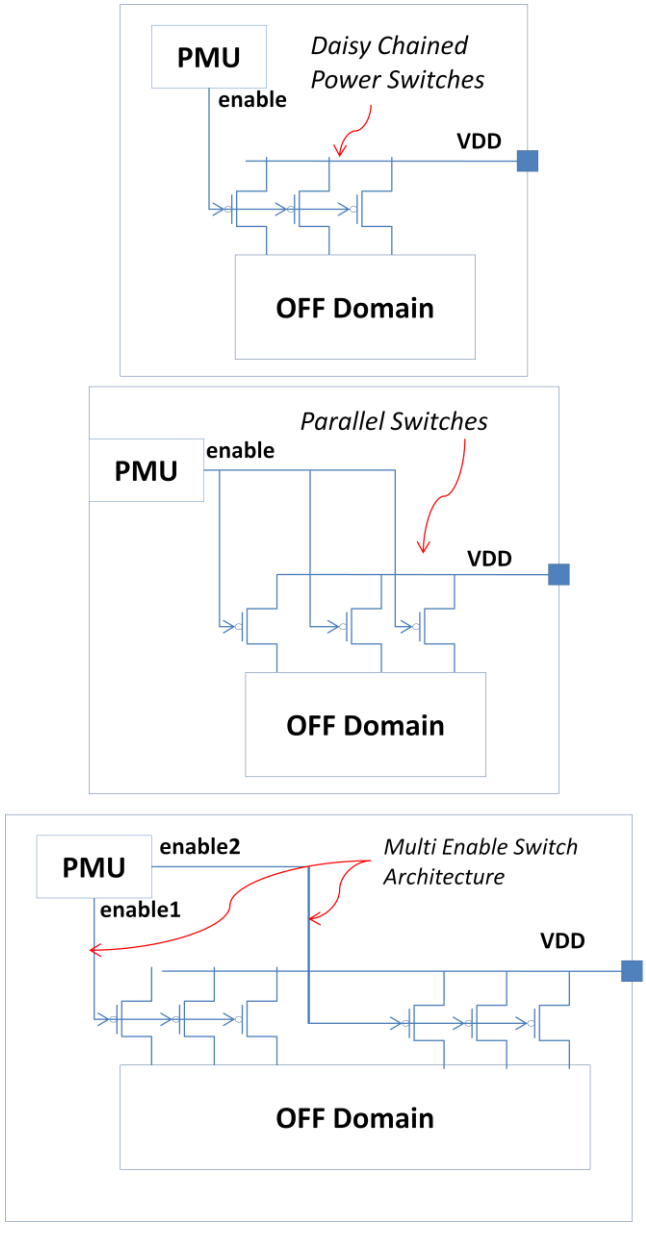
Power Switch Networks

In a post layout netlist, power-switches will have synthesized and connected based on the power-intent information and needs driven by the physical layout.

Power Switch Network v/s power intent

A key challenge in PG netlist verification is to map the power-switch requirement defined in the power-intent to the power-switch architecture implemented in the netlist. Typically power-intent would simply indicate a power-switched domain. This requirement will almost always be converted into sever power-switches connected in a specific pattern (daisy-chain, grid based etc) to achieve the expected functional behavior (signal controlled ON/OFF) with correct electrical constraints (E.g. IR-drop constraints). Accuracy of power-switch network verification is tied to how accurately a static verification tool can do this power-intent to PG-netlist mapping of power-switch topologies.

Figure 9: Different power-switch implementations



Power Switch Connectivity Check

Once mapped, all the power-switches for a given power-domain must be verified to ensure the appropriate connectivity for at least the following ports:

1. Input and output supply rails (also called primary and virtual rails)
2. Enable signal connectivity and polarity
    a. Power down polarity/functionality of power-switch cell should be considered here.
3. Acknowledge port connectivity, if applicable

Depending on the library being used, there may be other ports on the power-switch and associated semantics that may need to be verified.

Structural, functional and equivalence checks

Structural and functional checks to be done on post-layout netlist are same as those done for the post-synthesis netlist. There are no new checks that come up at the PG-netlist stage, except that the analysis should be based on PG-pin connectivity in the design and not on the power-intent file based partitioning. This is especially important if power-intent v/s PG-pin connectivity comparison discussed earlier in the paper could not be done for any reason. Only the power-modes information should be taken from the power-intent file as this is not available in the PG-netlist.

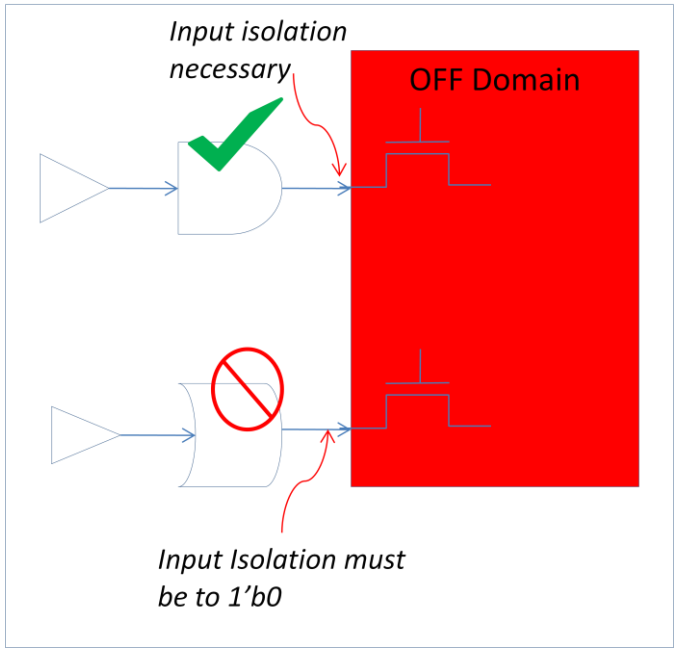
Equivalence checks should be done to ensure post-synthesis Netlist and power-intent are functionally equivalent to the PG-netlist.

4.4 Transistor Checks

Details of transistor level implementation of a cell can provide valuable information to MVRC, which is not available in a verilog netlist or a library model. A spice netlist is the preferred way of getting this information to the tool.

Transistor checks can mainly improve low-power checks in only post-synthesis stages of the design. In doing structural checks at post-synthesis and post-layout netlists, a spice netlist can provide information on how a cell has been implemented and the kind of structures seen at a specific port of the cell. For example, un-buffered input of an OFF power-domain would require an isolation cell at that input. Moreover, that isolation cell should isolate the signal to a low value and not to a high value. Without such an isolation cell, the design will have higher inactive state current for the OFF power domain and could miss its power targets.

Figure 10: Input isolation requirement identified only based on transistor checks



This kind of a check is not possible by using only verilog models of the cells instantiated in the design – a spice level netlist for all cells in the design is a must and can help avoid such bugs and catch them as early as post-synthesis stage in the design flow.

Transistor level checks on a design can also help identify any sneak leakage paths. These are paths existing between the power and ground rails in the design which may be enabled continuously or be controlled based on the logic state of the data pins of the cells. The leakage paths can be entirely within a cell (intra cell) or can span across multiple cells in the design (inter cell).

Again, it is impossible to identify such errors without incorporating spice netlist information into the static verification flow.

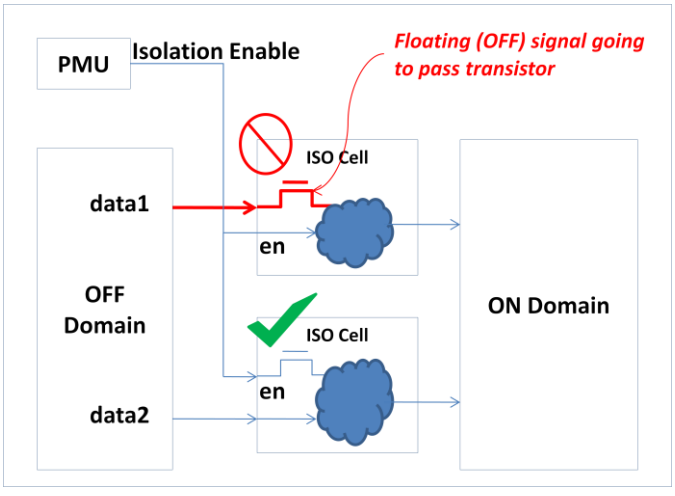
There are similarly several other checks possible on the design once the transistor level spice netlists are available to the static verification tools. These checks can help identify design issues early in the design flow and in the context of the power-intent of the design. These checks are not a replacement to the existing LV/DRC or any other spice netlist based verification/validation flows on a design. They are only meant to improve the accuracy of existing verilog netlist based static verification.

Needless to say, these issues will not be caught by the RTL/Netlist simulations of the design and a full spice netlist simulation of a chip will prove too costly to the product schedule. A static verification methodology which accounts for transistor level information for the cells in the design, can significantly improve static verification accuracy.

Library checks at RTL

Although spice netlists mainly help in post-synthesis design flow, they can also help with the library checks at RTL stage itself. At RTL stage, based on the power-intent, the spice netlist of the low-power cells (isolation, level-shifter, retention, power-switch, always-on buffer) should be validated to ensure they can perform their appropriate functions. E.g. an isolation cell spice netlist can be validated to ensure that the data-pin of the cell is not an un-buffered input (i.e. does not connect to a pass-transistor).

Figure 11: Pass transistor at data pin of isolation cell creates leakage path – Connection rules must be enforced at chip level



The list of these cells can be identified based on the attributes populated in the library and spice netlist from the library can be used to validate if these cells are indeed the right choice for the design.

5 Summary

In summary, static verification is instrumental at every stage in the design flow. At library preparation and power-intent definition stages it can help ensure correct and complete power intent and a library consistent with that power intent. As parts of the power intent get implemented into the design, static checks help validate those changes against the original power intent and a goal of functional silicon. Equivalence check ensures various versions of the design are consistent with the original power-architecture intended and verified.

Static verification helps catch bugs early in the design cycle without a need of any test vectors or simulation environment. It acts as a persistent check at every stage of the design flow, ensuring no design transformation (synthesis, clock-synthesis, layout, manual changes, ECO's etc) modifies the design in a manner which violates the original power intent and/or causes an electrical or functional error on the design.