# SISG-CFG

# Low Power Verification Plan

## John Pratheep Kulanthai Samy

## Rev 0.1

Contents

[1 Scope 5](#_Toc35797879)

[2 TB Architecture 6](#_Toc35797880)

[2.1 TB architecture block diagram: 6](#_Toc35797881)

[2.2 PMXTOR: 7](#_Toc35797882)

[2.3 NSPS: 9](#_Toc35797883)

[2.4 Q Channel & PMU interface Signals: 10](#_Toc35797884)

[2.5 Q Channel Domain (QCD): 11](#_Toc35797885)

[2.6 CXT 12](#_Toc35797886)

[2.7 AHB\_XTOR 12](#_Toc35797887)

[2.8 REG\_XTOR 12](#_Toc35797888)

[2.9 SLV 13](#_Toc35797889)

[2.10 AHB\_SLV 13](#_Toc35797890)

[2.11 APB\_SLV 13](#_Toc35797891)

[2.12 NocStudio 13](#_Toc35797892)

[2.13 Nemesis 13](#_Toc35797893)

[2.14 Gen\_streamX 13](#_Toc35797894)

[2.15 NocWeaver 13](#_Toc35797895)

[2.16 Simulation Generation 14](#_Toc35797896)

[2.16.1 Example 1: Full synchronization of all masters 14](#_Toc35797897)

[2.16.2 Example 2: No synchronization for unaffected masters 14](#_Toc35797898)

[2.16.3 Example 3: Master with reduced synchronization 16](#_Toc35797899)

[2.16.4 Example 4: Explicitly induce traffic during power transition sequence 17](#_Toc35797900)

[2.16.5 Example 5: Power notification instead of synchronization 17](#_Toc35797901)

[2.16.6 Example 6: Simply commands if masters powered down 18](#_Toc35797902)

[3 Unified Power Factor (UPF): 19](#_Toc35797903)

[3.1 Power Management Techniques 21](#_Toc35797904)

[3.1.1 Power Gating 21](#_Toc35797905)

[3.1.2 Multi Voltage 21](#_Toc35797906)

[3.1.3 Dynamic Voltage and Frequency Scaling 22](#_Toc35797907)

[3.2 Power Management Cells 23](#_Toc35797908)

[3.2.1 Power Switch 23](#_Toc35797909)

[3.2.2 Isolation Cell 23](#_Toc35797910)

[3.2.3 Level Shifter 23](#_Toc35797911)

[3.2.4 Retention Register 24](#_Toc35797912)

[3.2.5 Multi Voltage with cell requirements 24](#_Toc35797913)

[4 Power Down/Up Sequences 25](#_Toc35797914)

[4.1 Power Down Accept Sequence 25](#_Toc35797915)

[4.2 Power Up Accept Sequence 25](#_Toc35797916)

[4.3 Power Down Deny Sequence 26](#_Toc35797917)

[4.4 Power Up Deny Sequence (valid ?) 26](#_Toc35797918)

[4.5 Master Bridge Power Down and Up Sequence 26](#_Toc35797919)

[4.6 Slave Bridge / Router Power Down and Up Sequence 27](#_Toc35797920)

[5 Software components 28](#_Toc35797921)

[6 Power simulation 29](#_Toc35797922)

[6.1 Static 29](#_Toc35797923)

[6.1.1 Conformal LP (CLP) 29](#_Toc35797924)

[6.1.2 VCLP 29](#_Toc35797925)

[6.2 Dynamic (Power Aware) 30](#_Toc35797926)

[6.2.1 Xcelium 30](#_Toc35797927)

[6.2.2 VCS NLP 30](#_Toc35797928)

[7 Test Flow 31](#_Toc35797929)

[8 Test Plan: 32](#_Toc35797930)

[9 Checkers 33](#_Toc35797931)

[9.1 NSPS LP Checker 33](#_Toc35797932)

[9.2 NS\_COMMON\_PROPS\_LP\_CHECKERS 34](#_Toc35797933)

[10 Coverage 36](#_Toc35797934)

[10.1 Steps For Coverage Collection 36](#_Toc35797935)

[11 Open Issues 37](#_Toc35797936)

[12 References 38](#_Toc35797937)

# Scope

Scope of this document is to verify code correctness, LP correctness and transport correctness under different traffic and power-domain scenarios

# TB Architecture

## TB architecture block diagram:

**Top**

NoC

NSPS(0..n)

PMXTOR (PMU)

CXT0

CXTn

AHB

XTOR

AXISLV

APBSLV

AHBSLV

S

Y

N

C

U

N

I

T

req

ack

req

ack

req

ack

ack

req

Q ChannelI/f

rst\_<pd>

global reset and clocks

pwr & iso controls <pd>

clocks

ClkGen

REG

XTOR

req

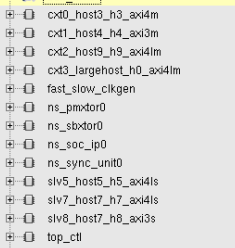
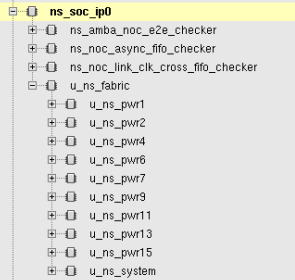
ack

pwr0

pwrn

system

Figure : Low Power Functional Block Diagram

## PMXTOR:

Pm\_xtor is a new transactor that needs to be built. Its interface is rather simple in that it has to support pairs of req/ack signals, up to the number of agents supported.

Pm\_xtor stimulus should be of the following vector format, similar to cxt and ahb\_xtor. Nemesis will generate stimulus for all of these transactors.

Table : PMGR\_XTOR instruction Format

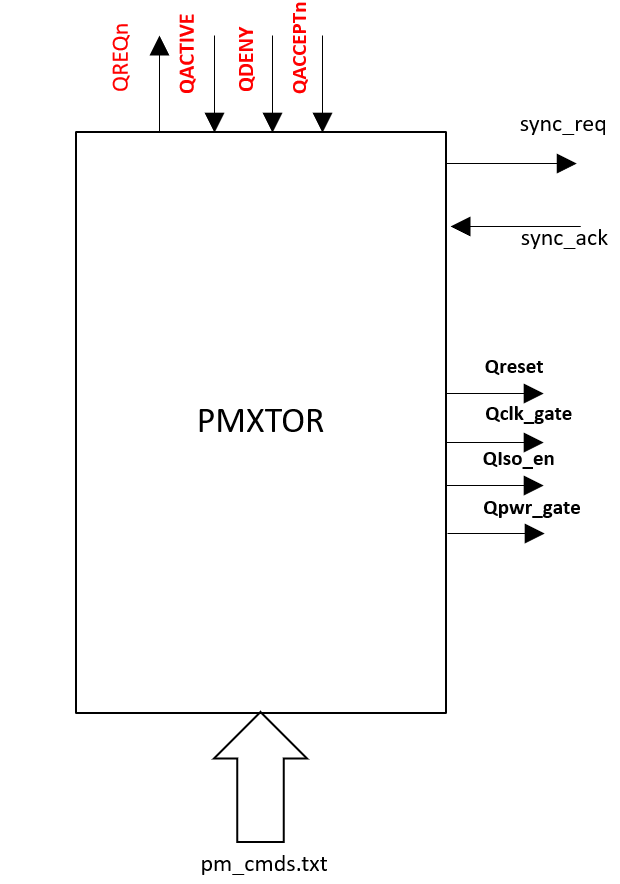
|  |  |  |
| --- | --- | --- |
| Bits | Field | Description |
| Instruction\_vec[259:0] | Agent\_vector | Each bit represents an agent that requires synchronization. Up to 256 agents plus regxtor. Regxtor is bit 256. |
| Instuction\_vec[263:260] | Opcode | Indicate which instruction this is. (See table below for encoding). |

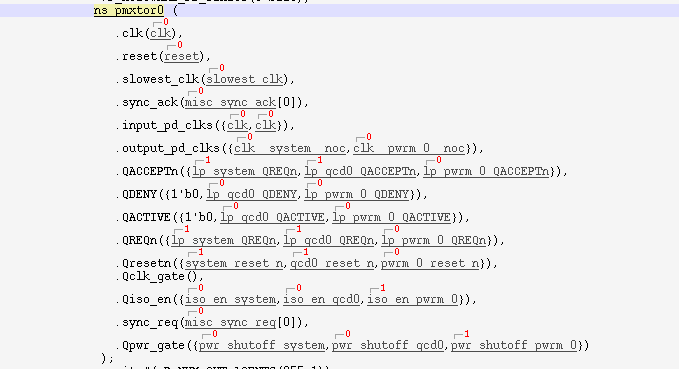
Table : PMGR\_XTOR Opcodes

|  |  |  |
| --- | --- | --- |
| Opcode | Instruction Name | Description |
| 0 | Sync | Synchronizes with agents whose bit position in Instruction\_vec[259:0] are 1 with four-way handshake. |
| 1 | NOP | Nop command. Use instruction\_vec[63:0] as number of cycles. |
| F | EOF | End of file. This must be the last line in the command vector file. |

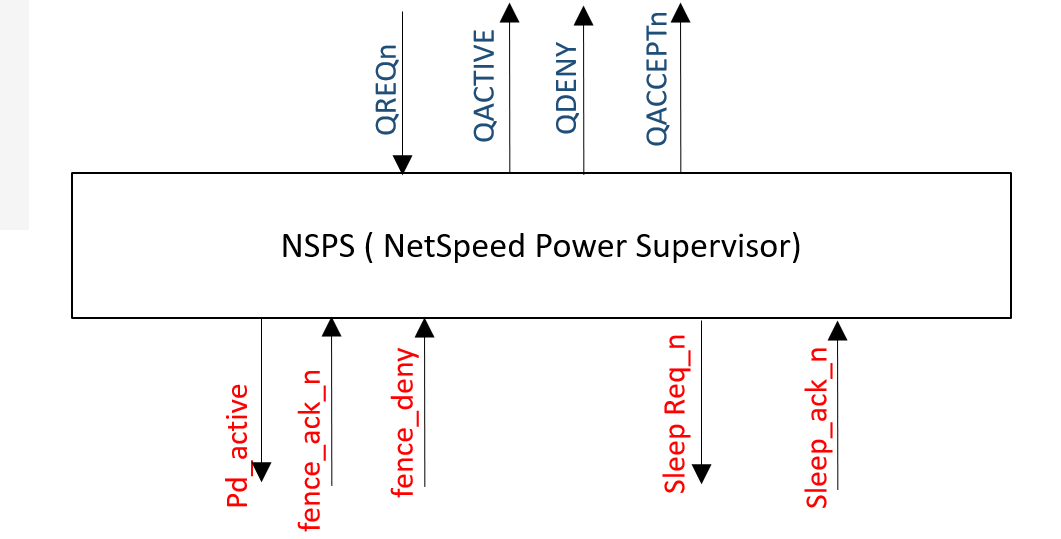
Table : PMGR\_XTOR parameters

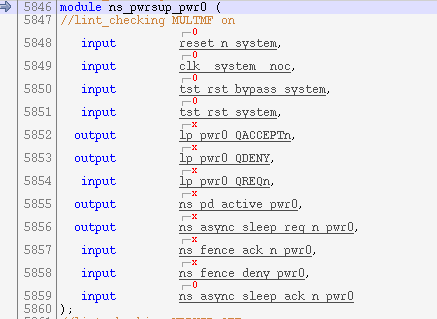
|  |  |
| --- | --- |
| Parameter | Description |
| P\_NUM\_OF\_AGENTS | Number of master agents that require four-way handshake. |



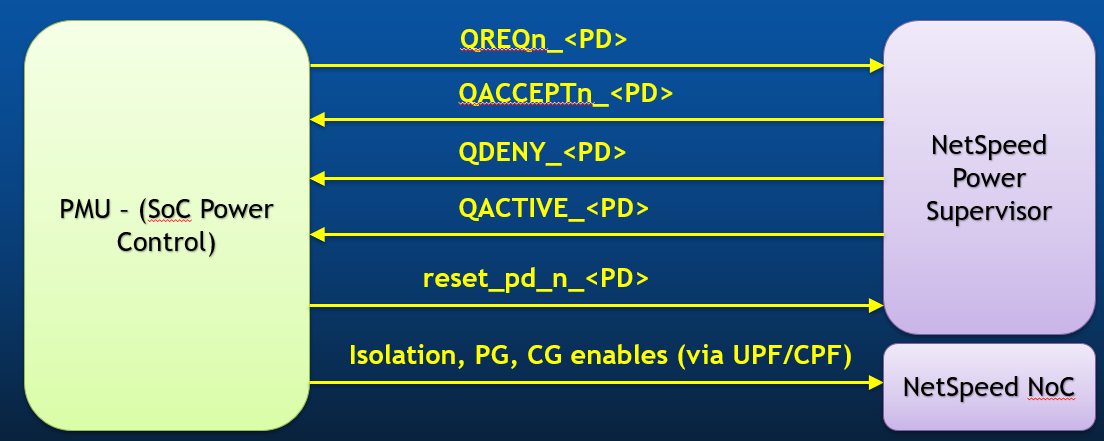


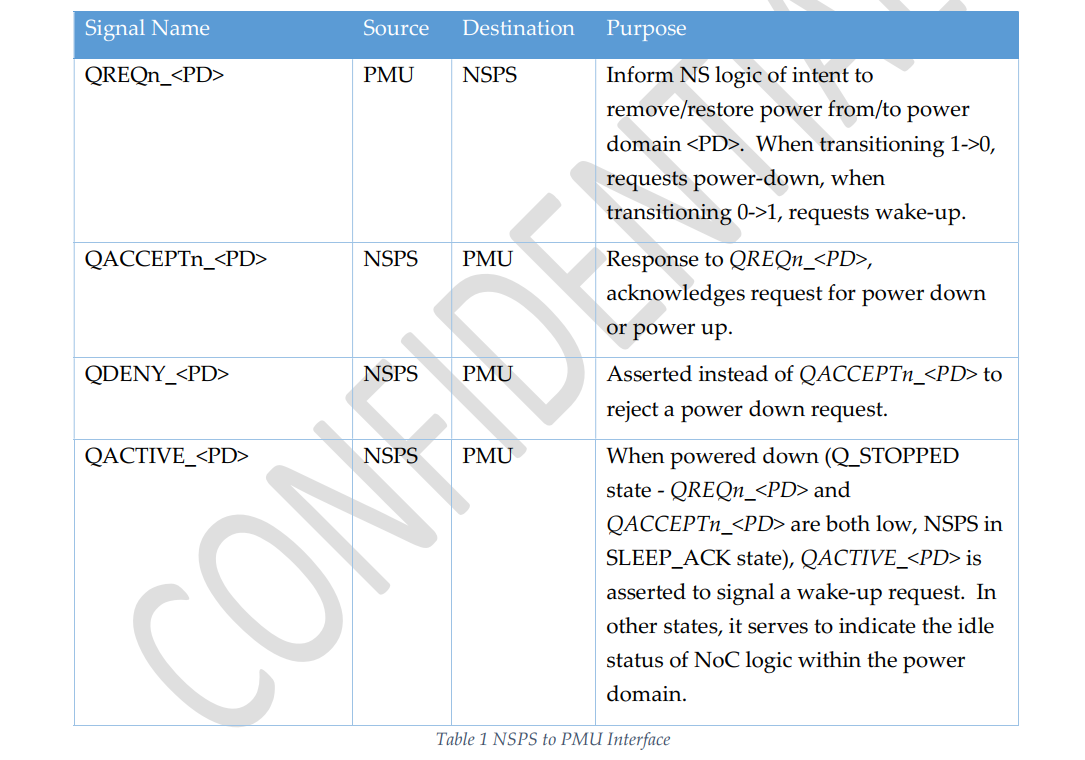
## NSPS:

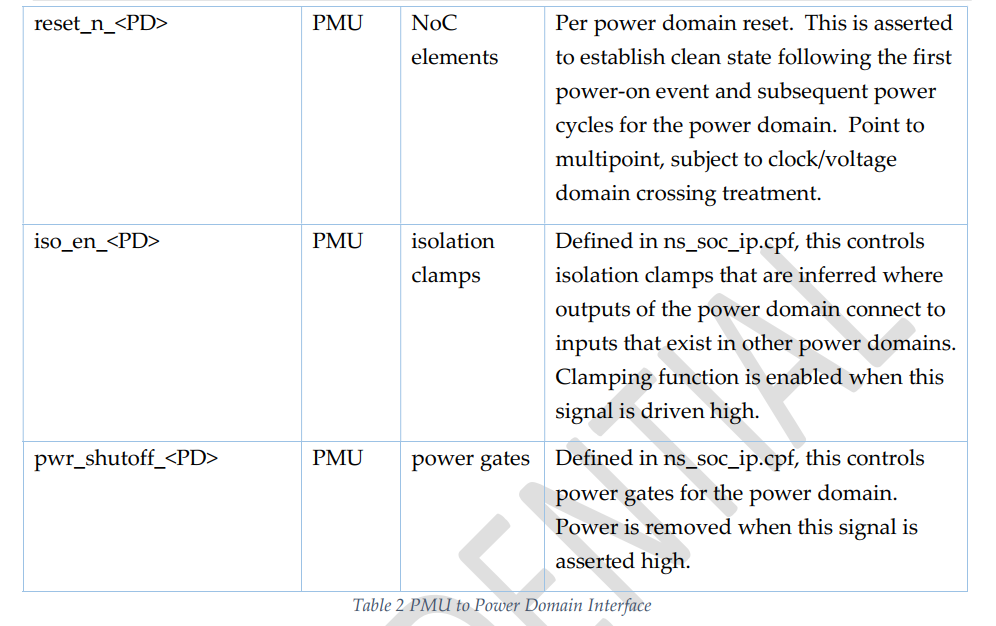




## Q Channel & PMU interface Signals:









## Q Channel Domain (QCD):

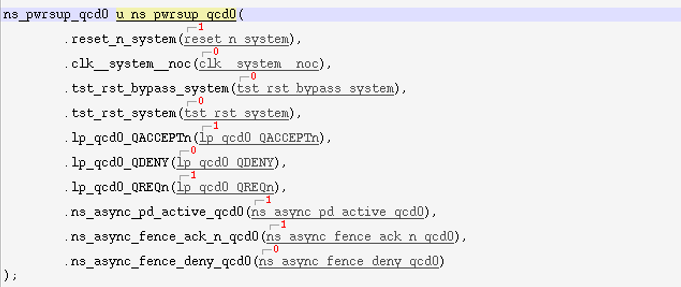
Has no power intent. Intended to capture the non-power dependency of a slave host such as clock-gate

There is a need to manage dependencies on the independent availability status of multiple slave host resources that may exist beyond a given slave bridge's host interface. For example, the slave agent may have multiple different clock domains that can be independently shut off, all of which are required to successfully complete transactions initiated over the NoC.

In order to satisfy these requirements, we now have a concept of general purpose "Q-channel domains". NocStudio will instantiate one Q-channel interface and NSPS per Q-channel domain in addition to creating one Q-channel interface and NSPS for each power domain.

Q-channel domains cannot be assigned to NoC elements like bridges, routers or IP like CCC, LLC, IOCB etc. Q-channel domains can only be assigned to the host side of slave bridges in the NoC (in addition to the power\_domain\_host property on slave bridges).

From the NoC's perspective, Q-channel domains assigned to the host side of the slave bridge are treated for fencing and draining purposes as additional power domains of the host. This means that if the slave's host side has a power domain "pd0" and additional Q-channel domains "qcd0" and "qcd1", the NoC will require pd0, qcd0 and qcd1 to all be powered up (QACCEPTn signal asserted on all 3 interfaces) before requests can be sent to the slave host.  Note that no specific voltage, power or clock resources are associated with Q-channel domains - they simply exist to manage a resource dependency that has no other relevance (electrical or otherwise) to the NoC.  No power intent (CPF or UPF) is inferred for Q-channel domains, whereas the power\_domain\_host specified power domain is used for power intent, e.g., inferring isolation clamps and level shifters as needed).



## CXT

cxt needs to add support for a new command “power\_sync” and the req/ack sidebands to handshake with pmgr\_xtor.

In addition, cxt needs to add a thin layer of logic at the interface to model isolation logic for power-aware simulation to model masters being powered down. This means we need to add cxt isolation information to the CPF file for power-aware simulation.

## AHB\_XTOR

Everything for cxt applies to ahb\_xtor.

## REG\_XTOR

Reg\_xtor needs to add support for a new command “power\_sync” and the req/ack sidebands to handshake with pmgr\_xtor.

## SLV

slv model needs to have an input signal that will reset all data to be 0xdeadbeef. This signal will assert whenever a slave is supposed to power down. Nemesis data checking will expect 0xdeadbeef after a slave powers back up. The goal of having this feature is to ensure that a slave that should power down does power down. In most of our verification, we verify that domains that should be powered up are up. This is one mechanism that checks domains that should be down are down. The actual signal could come from one of the power registers for the power domain the slave is in.

## AHB\_SLV

Everything slv model has applies to ahb\_slv.

## APB\_SLV

Everything slv model has applies to apb\_slv.

## NocStudio

NocStudio needs to generate the following for every low-power config:

For every power profile, a register access sequence that allows entry into that power profile. Ideally this would output in regxtor syntax already, in terms of register reads, writes and polls.

For every power profile, a list of legal traffic that lists masters with ID, base/mask, in valid\_traffic\_flows.txt. The format can match the existing section at the end of valid\_traffic\_flows.txt(ID: <id> axi4m..).

NocStudio needs to generate CPF file for power-aware simulation. It needs to generate both a CPF file for the Noc itself, and another CPF file that includes cxt isolation information in addition.

## Nemesis

Nemesis needs to use all the information NocStudio generates to produce random power transitions with proper synchronization. Dynamically produce legal traffic for each power profile for all master transactors.

Call the reg accesses sequences for reg\_xtor. Adjust data checking and traffic generation to dynamically shift to new profiles. Understand power-down of slaves and adjust data checking to expect 0xdeadbeef.

Programmable knobs for additional user control.

Regular Nemesis data checking applies. In addition, an option can be added to expect 0xdeadbeef for every slv/apb\_slv/ahb\_slv that powers back up. This option can be turned off for data to accumulate for data checking.

Nemesis will have an option to allow programmable number of power profiles to transition through within each simulation.

## Gen\_streamX

Gen\_streamX needs to stitch in pmgr\_xtor, all the side bands of all the master xtors, all the new inputs to all the slave models.

## NocWeaver

NocWeaver needs to be able to generate random low-power NoCs.

## Simulation Generation

### Example 1: Full synchronization of all masters



Figure : Full synchronization of all masters

In the above example, three traffic masters are shown: cxt0, cxt1, and cxt2. Each cxt runs through its own set of commands until it hits two power\_sync commands. The first synchronization command of each transactor indicates synchronization between pmgr\_xtor and all the other xtors. At this point, all masters have quiesced their traffic and are ready to transition into the next power profile. After the first synchronization, cxt’s would move on to the second power\_sync which would not be satisfied until regxtor has completed the NocStudio sequence of writing to all the registers to complete the power down sequence. Once the second power\_sync is complete, the NoC has completed transition into the new power profile. Cxt’s will start traffic for the new profile after this point. If a cxt is powered down in the new power profile, this could mean no traffic until the next power profile transition.

This scheme works for all power transitions. Any combinations of masters and slaves can be turned on and/or off. However, this scheme is heavily serializing with all masters fully stopping traffic for every power transition. The following examples will show improvements that can be made on this scheme.

### Example 2: No synchronization for unaffected masters

In example 1, all masters participate in all power transition synchronization. But what if the new power profile does not impact a particular master at all? That master should not have to stop its traffic. The following is an example where cxt2 does not participate in the power transition synchronization. By keeping cxt2 traffic going, we cover the case where there is traffic in the NoC while regxtor is performing a power profile transition. This is an important case that would not be exercised in example 1.



Figure : Example of master not affected by power events

### Example 3: Master with reduced synchronization

What if a slave is being powered down in the new power profile? A master that used to issue traffic to that slave only needs to stop issuing to the slave before the power transition happens. As long as that’s the case, it does not need to wait for the actual power transition to complete.

In the example below, cxt2 only has one synchronization event to get to the starting point of power transition. Cxt2 then starts issuing traffic while regxtor is performing power transition reg accesses at the same time.



Figure : Example of cxt2 sending traffic without waiting for power transition to finish

### Example 4: Explicitly induce traffic during power transition sequence

In example 3, cxt2 may or may not reach its power\_sync before cxt0 and cxt1. To ensure that cxt2 can issue traffic while regxtor is executing the power transition sequence, we can change the stimulus as follows.



Figure : Explicit synchronization to ensure cxt2 has ongoing traffic while regxtor sequence in progress

In the above example, pmgr\_xtor explicitly synchronizes with cxt2 first to allow it to start traffic for the new power profile right away, before synchronizing with the other transactors to start the power transition sequence. This significantly increases the likelihood that cxt2 has ongoing traffic while regxtor is going through its power transitioning sequence.

### Example 5: Power notification instead of synchronization

In example 3, cxt2 still has to synchronize with the other agents to stop traffic until the beginning of the power transition point. Another possible optimization is to add a power\_notify command, such that cxt2 will notify pmgr\_xtor with a req, and continue to issue the traffic for the new power profile without waiting for an ack. This scheme, however, involves more complexity for the cxt hardware to have the four-way handshaking continue in the background. When cxt2 hits the next power\_sync or power\_notify command, it will have to check that the outstanding four-way handshake is done. This scheme also requires cxt to support a new “power\_notify” command. Note that pmgr\_xtor is not aware of the difference between power\_notify and power\_sync. The four-way handshake signaling is the same for both.



Figure : Cxt2 does power\_notify without waiting for ack.

### Example 6: Simply commands if masters powered down

Further simplification of commands are possible if a master is powered down.

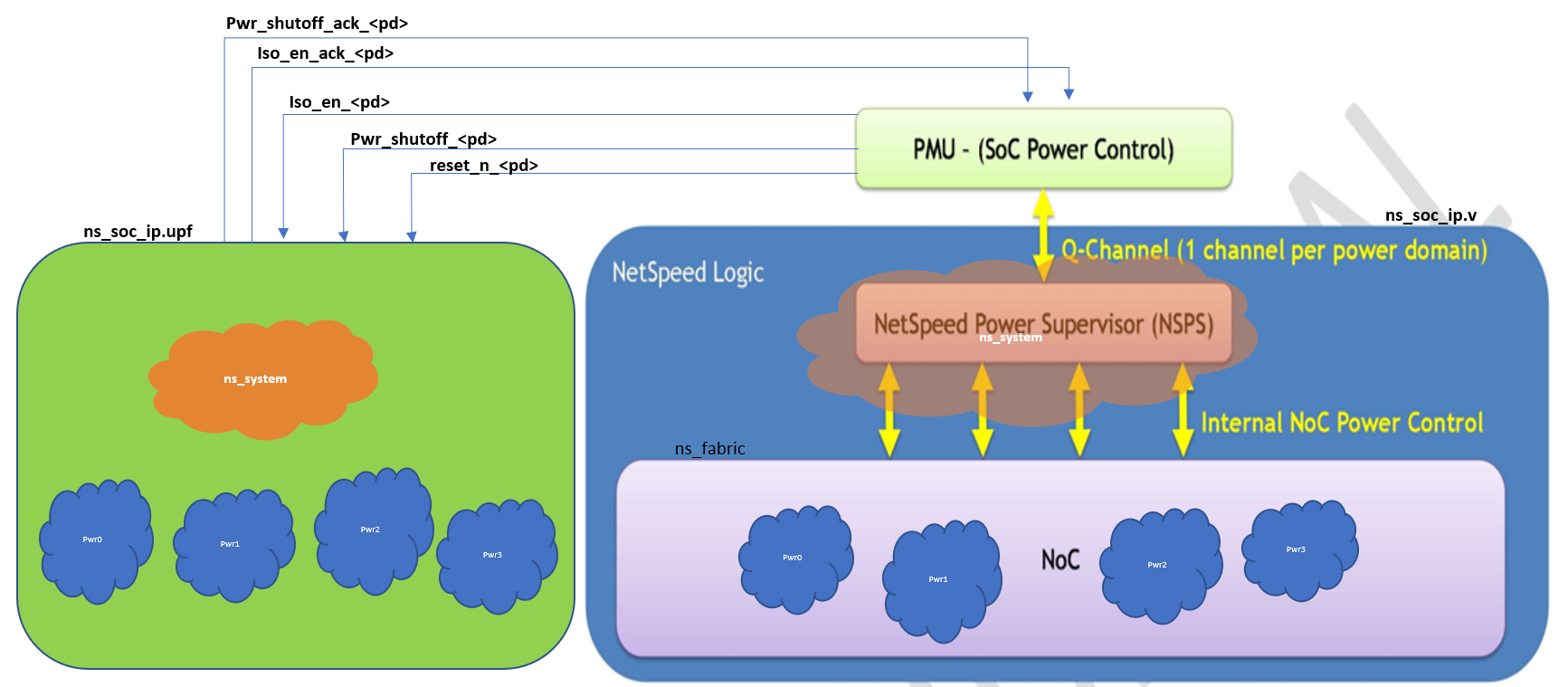


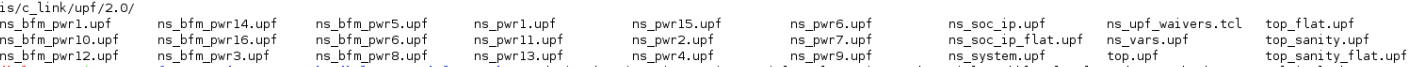
Figure : Reduce power\_sync commands when cxt0 and cxt2 power down

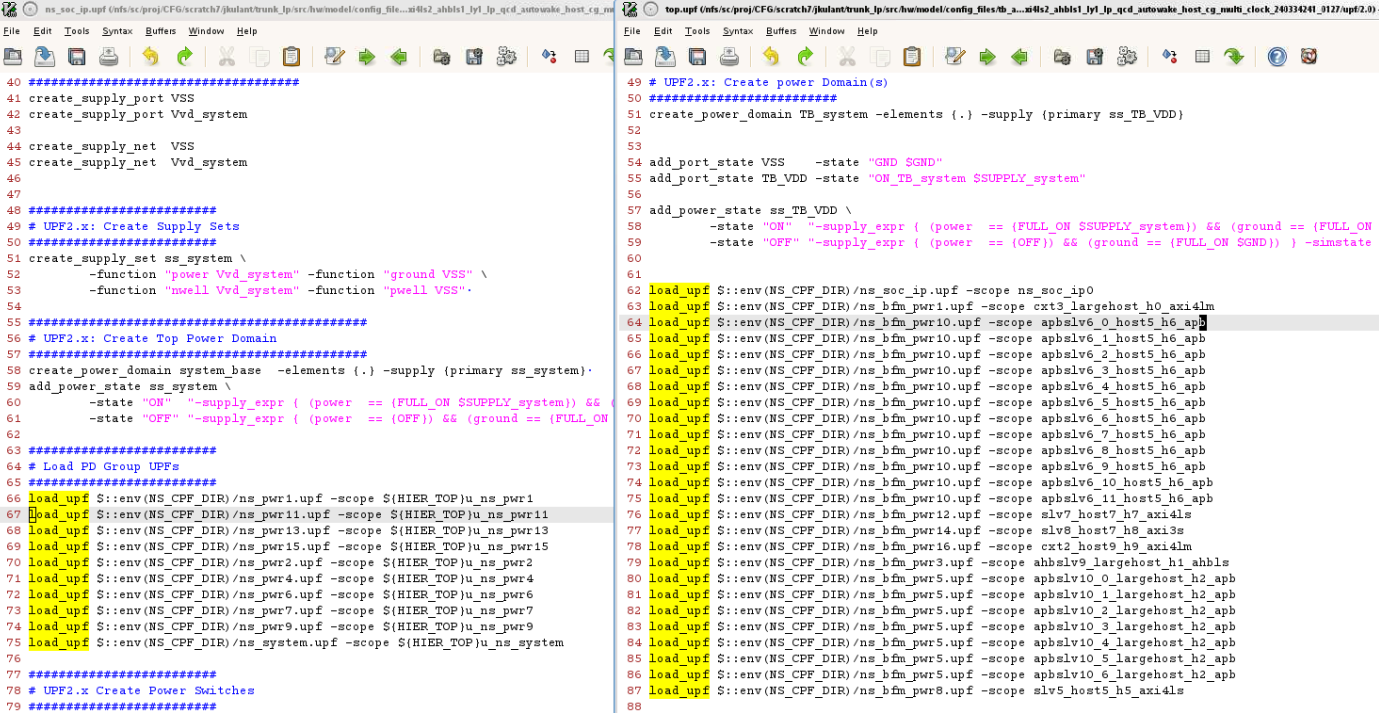
In the above example, two power transitions happen, with the first transition powering down cxt0 and cxt1 completely. But since there is no traffic in the new power profile, the next set of commands for these cxt’s should be two more power\_sync commands. In this case, we can remove the two extra power\_sync as shown above for cxt0 and cxt1, and remove cxt0 and cxt1 from pmgr\_xtor synchronization command arguments. This reduces the number of instructions.

# Unified Power Factor (UPF):

Unified Power Format (UPF) is a language used to describe the power intent, or power strategy, of a logic design. UPF is IEEE Standard 1801.



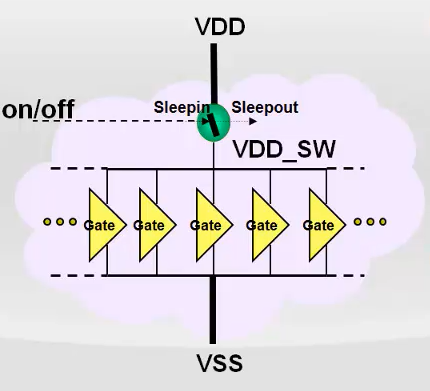




## Power Management Techniques

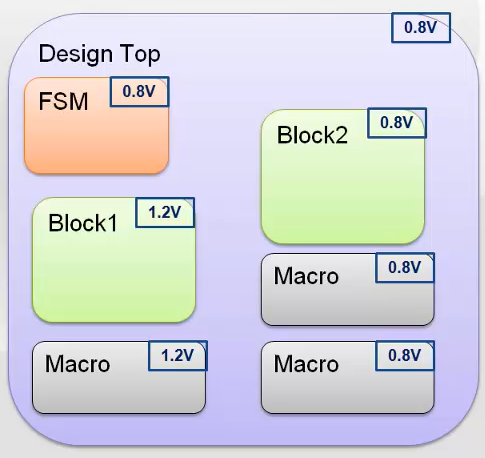
### Power Gating

Power reduction technique to save leakage power by shutting off / powering down the unnecessary logic.



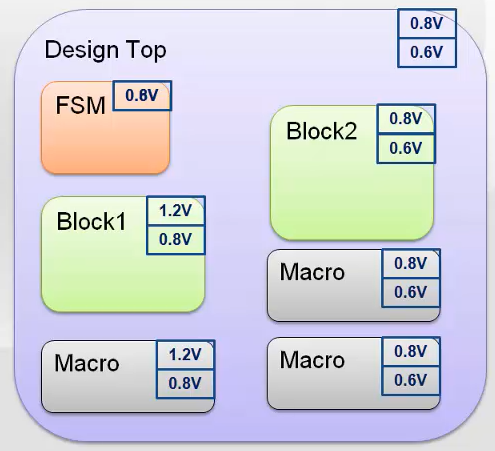
### Multi Voltage

Power savings technique to operate different blocks of logic at different voltages



### Dynamic Voltage and Frequency Scaling

Power saving technique to change the voltage and/or clock frequency while the chip is running to save power



## Power Management Cells

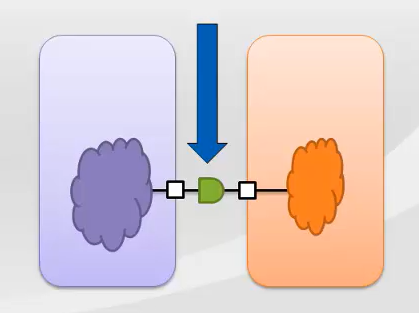
### Power Switch

Software controlled switch can be implemented on VDD or VSS.



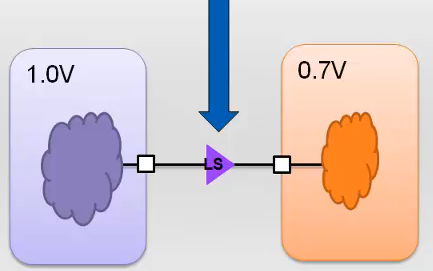
### Isolation Cell

Passel logic values during normal mode operation and clamps its output to some specific logic value when a control signal is asserted



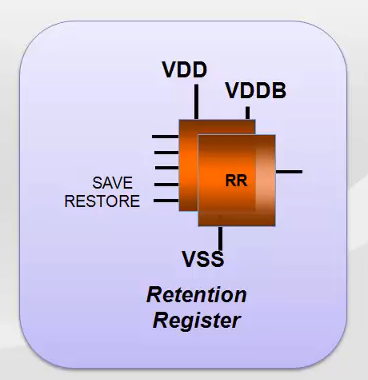
### Level Shifter

Translates Signal values from an input voltage swing to different output voltage swing

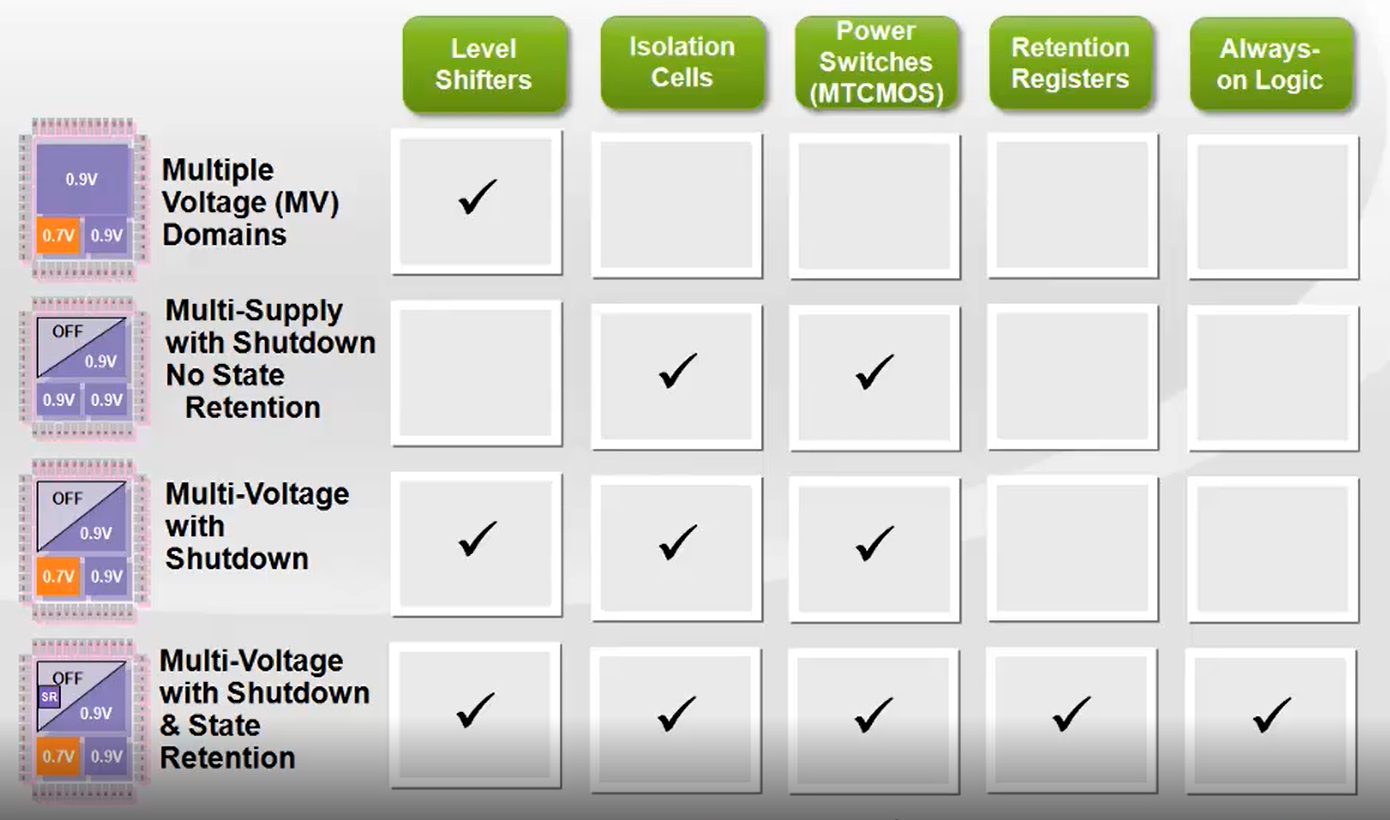


### Retention Register

Extends the functionality of a sequential element with the ability to retain its memory value during the power down state

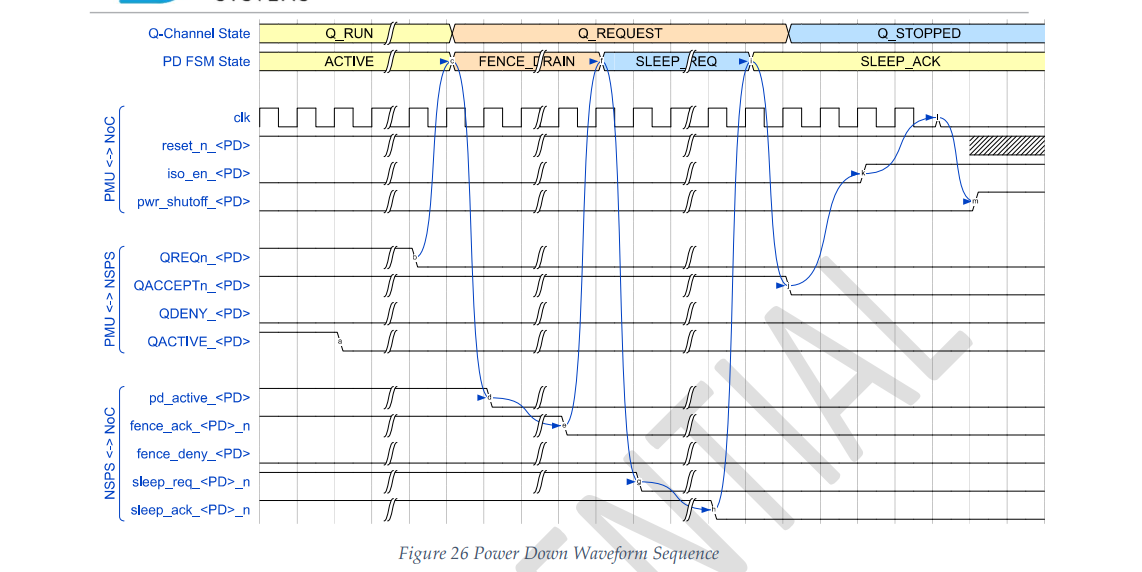


### Multi Voltage with cell requirements

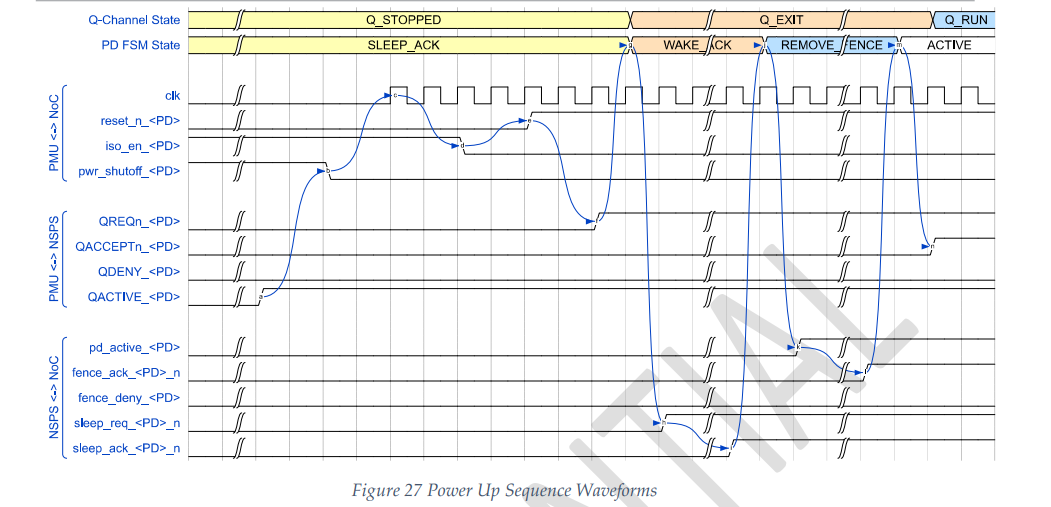


# Power Down/Up Sequences

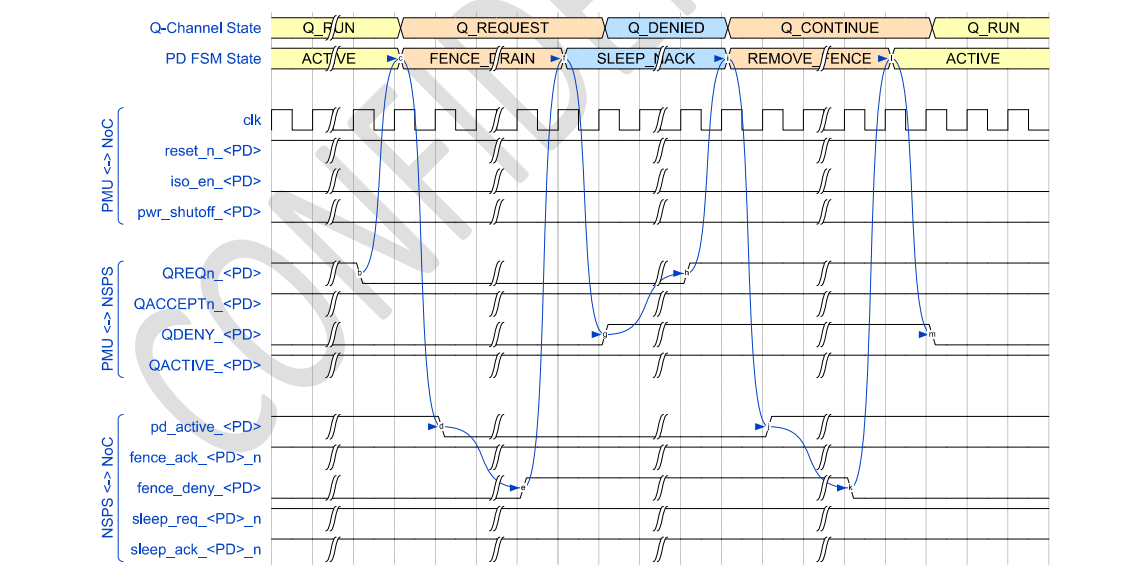
## Power Down Accept Sequence



## Power Up Accept Sequence

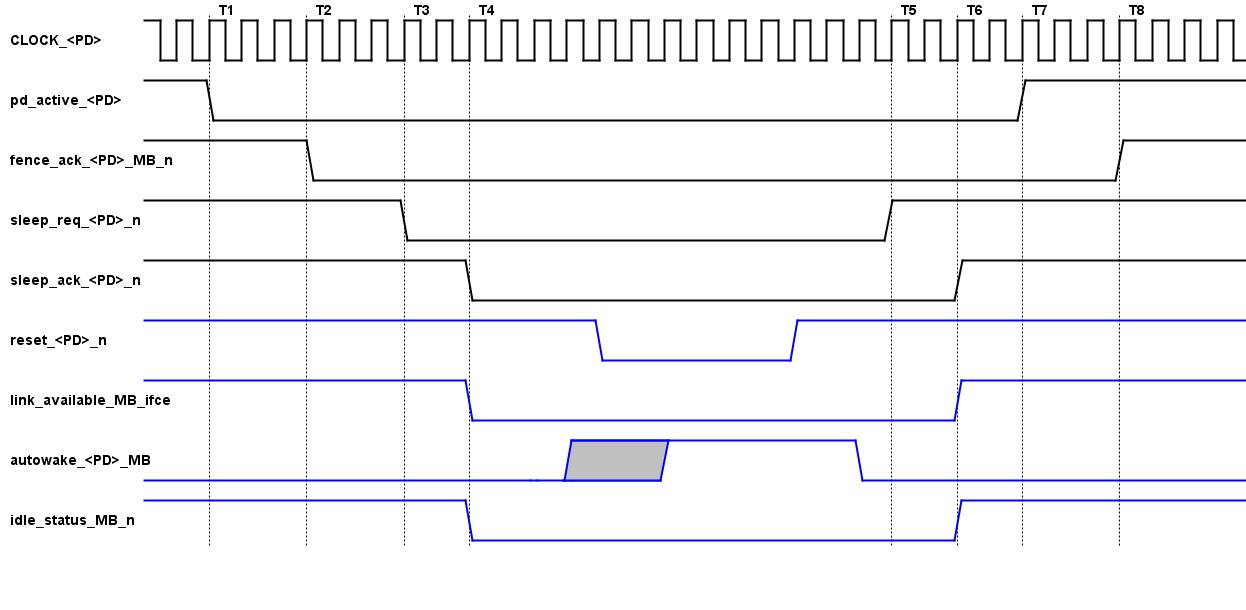


## Power Down Deny Sequence

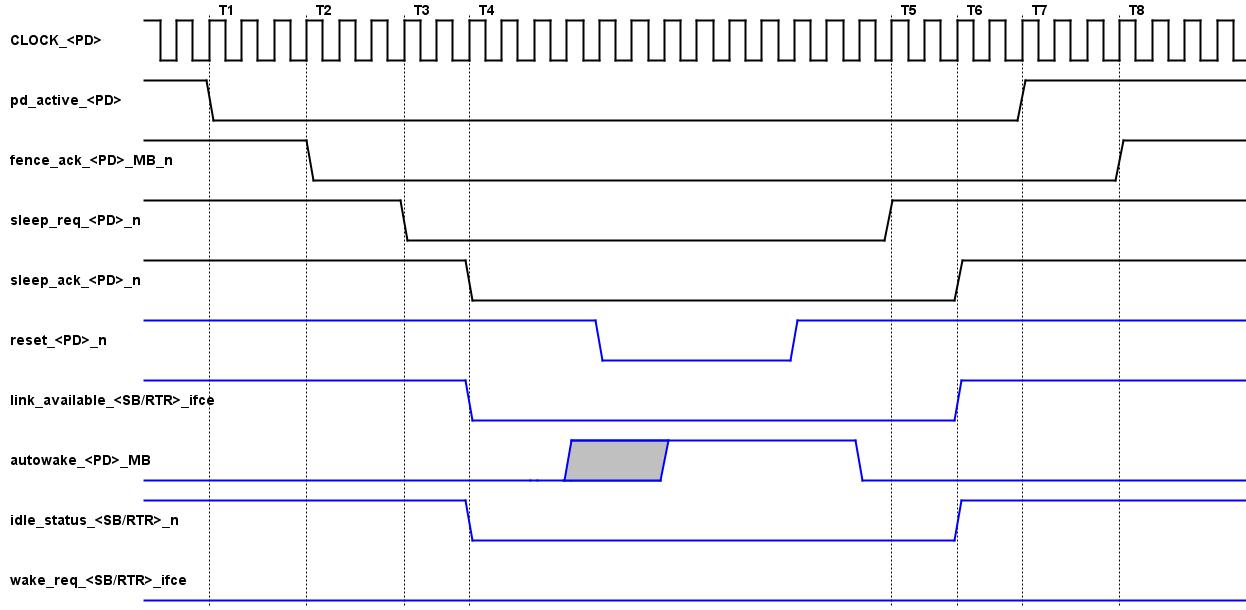


## Power Up Deny Sequence (valid ?)

## Master Bridge Power Down and Up Sequence



## Slave Bridge / Router Power Down and Up Sequence



# Software components

In addition to the hardware components in the testbench, NocStudio and Nemesis also play key roles in this environment.

For any given low-power config, NocStudio must provide power transition sequences for every legal power profile. The power transition sequences would contain information such as turning Noc isolation logic on and off, turning bridges and routers on and off, turning cxt isolation on and off, etc. In addition, NocStudio must provide all legal traffic flows for each power profile.

Nemesis generates the power transitioning events and the input stimulus for all transactors in Figure 1 in the testbench. It uses all the information NocStudio provides, randomly selects among legal power profiles to enter and produce legal traffic flows for power profile that it enters to check functional correctness. During each Nemesis simulation, the Noc can transition through multiple traffic profiles, with the exactly number programmable and limited by simulation time.

# Power simulation

Low power designs use various power optimization techniques to achieve their power consumption goals. For RTL, this means the designs can have multiple power (voltage) domains, with signals crossing domain boundaries. UPF (Unified Power Format), an industry standard language, is typically used to define the power intent of the design. This includes the specification of what happens to signals at power domain boundaries, how a power switch is constructed, what the legal power states of the design are etc. While some of these intents can be checked through dynamic simulation, e.g., using VCS-NLP, static checks provide a much more efficient way to catch issues early. Low Power Lint (RTL) is such a static check technique where the design (RTL+UPF) is checked for certain pre-determined rule violations. This only requires elaborating the design (RTL) along with it's UPF, and no (dynamic) simulation needs to be run.

## Static

### Conformal LP (CLP)

Power aware equivalency checking is an advanced method for RTL through place-and-route functional verification. Power aware equivalency checking replaces boundary checks, which can be used only to verify two netlists that have isolation already inserted.

Power aware equivalency checking offers the following features:

■ Low power design equivalency checks

■ State element domain consistency checks

■ Can handle cell modeling, switch modeling, and retention instances

■ Power intent comparison

■ State retention strategy comparison

Version used: confrml/191/19.10.300

### VCLP

• Reporting, Violation Analysis, Waivers

• PST Debug • VC UPF (DIUC) – Design Independent UPF Checker

• ML RCA – Machine Learning, Root Cause Analysis

• LPSS (ISOSPEC) – lp\_signal\_supply check

Version used: vcsmx/P-2019.06-SP2

Training Materials at

<https://dtspedia.intel.com/Low_power_lint_rtl#Training_Material_from_March_2020>

## Dynamic (Power Aware)

### Xcelium

Comprehensive solution for low power including architecture optimization, power estimation and analysis, functional verification, implementation and signoff, and IP for digital and mixed-signal designs at both chip and system level

Support for both industry-standard power intent formats (CPF and IEEE 1801), enabling customers to adopt the design flow of their choice

Production proven on thousands of designs mitigating risk of re-spins, reducing product development time and costs

Version Used: xcelium/19.03-s003

### VCS NLP

VCSMX is the POR RTL simulator for power simulations. It is sometimes referred to as 'NLP' or 'VCS-NLP' (Native Low Power). VCS reads in your UPF file and your HDL (Verilog and VHDL) and instantiates 'virtual' power elements based in the design, using their UPF description: power switches, isolation cells, supply network, etc.

The NLP user guide can be found in your VCS install area: $VCS\_HOME/doc/UserGuide/pdf/mvsim\_native.pdf.

Also, see training docs below under [See Also](https://intelpedia.intel.com/SoC_Low_Power_Design#See_Also). SoC-DA provide NLP and UPF training.

#### Power Aware Debug (Verdi)

Verdi power‐aware debug tool provides more advanced automation capabilities that can help engineers track down the root cause of a bug across both the RTL and power intent files. For example, if there is an ‘X’ value, can the tool help determine whether it is caused by incorrect RTL code, or a power off, or perhaps there is something wrong in the power control sequence? Additional features for power‐aware debug automation include:

■ Equivalent treatment of RTL and UPF for automatic location of the drivers/loads.

■ Provide visualization for signals driven by UPF, so that debugging can be directed to trace into the UPF, as the cause may reside in the power control defined in the UPF.

■ Un‐roll trace paths throughout different power domains to easily identify retention, isolation or level shifter signals. By doing this, paths can be presented with adequate power domain and power‐related logic information for further debugging.

■ Annotation of power modes on RTL, waveform, schematic and UPF power views so that engineers can examine the dynamic power modes and states easily in any view, which helps to correctly determine the next debug step.

■ Automatic location of the root cause of a bad value in either the RTL or UPF code; enabling engineers to seamlessly trace between the RTL and UPF code.

more details on verdi power debug features can be found at: [Power\_Aware\_Debug\_Verdi3.pdf](https://intelpedia.intel.com/images/3/3e/Power_Aware_Debug_Verdi3.pdf)

Version Used: verdi/P-2019.06-SP2

# Test Flow

Test flow has following sequences

ns\_pd\_to\_pdnum\_map.txt file is generated specifc to LP sims which associates ids to the power domains

ns\_power\_map\_tables.sv : Lists the data structures needed for nemesis and the checkers

Lists the pd between masters and slaves.

Lists the pds based on the traffic ie rd/wr,qos etc

Whether autowake is enabled for power domain

Commented lines are significant here

Pm\_cmds\_all.txt : indicates the weightage of the various FSM states through which the power domains traverse

Valid\_traffic\_flows.txt : contains the power domains which are relevant for the traffic that flows from the master to the slave ids mentioned

Top.v :

- Q channel domain (QCD) instantiated

- Pmxtor instance connections

Pm\_cmds.txt : contains the list of commands to turn on turn off power domains in a random fashion. Also contains the sync commands

# Test Plan:

Test scenarios, checker and coverage details find in the test plan located at

<>/doc/internal\_docs/Low\_Power\_Docs



# Checkers

## NSPS LP Checker

|  |  |  |
| --- | --- | --- |
| Sl No | Assertions | Description |
| **Category - NSPS LP Checker** | | |
| 1 | assert\_timeout\_check | When Req & Ack staying high, req de-asserted, ack has to be de-asserted before fixed number of cycles & 1 clk cycle later req again has to be asserted or req should go high again. |
| 2 | Deassert Timeout\_check | When Req & Ack staying Low, req asserted, ack has to be asserted within fixed number of clk cycles & minimum 1 clk cycle delay. |
| 3 | sleep\_assert\_timeout\_check | When sleep req & ack are high, sleep req getting de-asserted, sleep ack should go low within fixed number of clk cycles & minimum 1 clk cycle delay. |
| 4 | sleep\_deassert\_timeout\_check | When sleep req, ack are low & sleep req being asserted then sleep ack has to be asserted within fixed number of clk cycles & minimum 1clk cycle delay. |
| **Category – NSPS Power sequence LP Checker** | | |
| 1 | ns\_handshake\_checker | There are 3 Phases for immediate assertion checkers. P\_4\_PHASE,P\_6+PHASE & P\_8\_PHASE.  State Definition-{req\_n,ack\_n,deny\_n}  { ASSERT\_ACCEPTED = 3'b001,  CMD\_DEASSERT = 3'b101,  DEASSERT\_ACCEPTED = 3'b111,  DEASSERT\_DENIED = 3'b100,  RETRACT\_DEASSERT = 3'b000,  CMD\_ASSERT = 3'b011,  ASSERT\_DENIED = 3'b010,  RETRACT\_ASSERT = 3'b11 0} |
| 2 | P\_6\_PHASE- NSPS\_QCHANNEL\_CHECK | ASSERT\_ACCEPTED->CMD\_DEASSERT  CMD\_DEASSERT->DEASSERT\_ACCEPTED  DEASSERT\_ACCEPTED->CMD\_ASSERT  CMD\_ASSERT->ASSERT\_ACCEPTED or ASSERT\_DENIED  ASSERT\_DENIED-> RETRACT\_ASSERT  RETRACT\_ASSERT-> DEASSERT\_ACCEPTED |
| 3 | P\_4\_PHASE-NSPS\_SLEEP\_CHECK | ASSERT\_ACCEPTED->CMD\_DEASSERT  CMD\_DEASSERT->DEASSERT\_ACCEPTED  DEASSERT\_ACCEPTED->CMD\_ASSERT  CMD\_ASSERT->ASSERT\_ACCEPTED |
| 4 | P\_8\_PHASE-NSPS\_FENCE\_CHECK | ASSERT\_ACCEPTED->CMD\_DEASSERT  CMD\_DEASSERT ->(DEASSERT\_ACCEPTED or DEASSERT\_DENIED)  DEASSERT\_ACCEPTED->CMD\_ASSERT  DEASSERT\_DENIED-> RETRACT\_DEASSERT  RETRACT\_DEASSERT-> ASSERT\_ACCEPTED  CMD\_ASSERT ->(ASSERT\_ACCEPTED or ASSERT\_DENIED)  ASSERT\_DENIED-> RETRACT\_ASSERT  RETRACT\_ASSERT-> DEASSERT\_ACCEPTED |
| 5 | negedge QREQn & cmb\_reset\_n =1 | expected fence\_state=DEASSERT\_ACCEPTED (P\_FENCE\_EN ==1)  expected sleep\_state=DEASSERT\_ACCEPTED (P\_SLEEP\_EN==1) |
| 6 | negedge fence\_ack\_n &  cmb\_reset\_n =1 | expected QREQn state=CMD\_ASSERT  expected SLEEP state=DEASSERT\_ACCEPTED(P\_SLEEP\_EN ==1) |
| 7 | negedge sleep\_ack\_n &  cmb\_reset\_n =1 | expected QREQn state=CMD\_ASSERT  expected FENCE state=ASSERT\_ACCEPTED(P\_FENCE\_EN ==1) |
| 8 | negedge QACCEPTn &  cmb\_reset\_n =1 | Expected FENCE state=ASSERT\_ACCEPTED(P\_FENCE\_EN ==1)  Expected SLEEP state=ASSERT\_ACCEPTED(P\_SLEEP\_EN ==1) |
| 9 | posedge fence\_deny &  cmb\_reset\_n =1 | Expected QCHANNEL state=CMD\_ASSERT  Expected SLEEP state=DEASSERT\_ACCEPTED(P\_SLEEP\_EN ==1) |

## NS\_COMMON\_PROPS\_LP\_CHECKERS

|  |  |  |
| --- | --- | --- |
| Sl No | Assertions | Description |
| **Category - NS\_COMMON\_PROPS\_LP\_CHECKERS** | | |
| 1 | Idle\_Check | When ack\_sig is high, check for actual\_signal to be equal to expected signal |
| 2 | Reset\_Check | When pd\_reset is high, check for actual\_signal to be equal to expected signal |
| 3 | Reset\_Check\_Param | When pd\_reset is high, check for actual\_signal to be equal to expected signal |
| 4 | Reset\_Check\_apb | When pd\_reset & psel is high, check for actual\_signal to be equal to expected signal |
| 5 | Post\_Idle\_Check | When ack signal is de-asserted, check for actual\_signal to be equal to expected signal |
| 6 | Input\_Flit\_Valid\_Check\_During\_Ifceblocked | During element ifceblocked (SleepAck assertion), no active input flit valid  When sleep\_ack is high, i\_flit\_valid shouldnot be high |
| 7 | Input\_Flit\_Valid\_Check\_During\_Reset | During element Reset, no active input flit valid  When pd\_reset is high, i\_flit\_valid shouldnot be high |
| 8 | Output\_Flit\_Valid\_Check\_During\_Ifceblocked | During element ifceblocked (SleepAck assertion), no active output flit valid  When sleep\_ack is high, o\_flit\_valid shouldnot be high |
| 9 | Output\_Flit\_Valid\_Check\_During\_Reset | During element Reset, no active output flit valid  When pd\_reset is high, o\_flit\_valid shouldnot be high |
| **Category - NS\_AXI4LS\_ISO\_LP\_CHECKER** | | |
| 1 | ns\_brdg\_rtr\_intf\_iso\_lp\_checker |  |
| 2 | ns\_axisb\_intf\_iso\_lp\_checker |  |
| 3 | ns\_ring\_intf\_iso\_lp\_checker |  |
| 4 | ns\_power\_intf\_iso\_lp\_checker |  |
| **Category - NS\_AHBLS\_ISO\_LP\_CHECKER** | | |
| 1 | ns\_brdg\_rtr\_intf\_iso\_lp\_checker |  |
| 2 | ns\_ahbls\_intf\_iso\_lp\_checker |  |
| 3 | ns\_ring\_intf\_iso\_lp\_checker |  |
| 4 | ns\_power\_intf\_iso\_lp\_checker |  |
| **Category - NS\_ACEMSTRBRDG\_CORE \_LP\_CHECKER** | | |
| 1 | ns\_rssb\_rtr\_intf\_iso\_lp\_checker |  |
| 2 | ns\_acemb\_intf\_iso\_lp\_checker |  |
| 3 | ns\_ring\_intf\_iso\_lp\_checker |  |
| 4 | ns\_power\_intf\_iso\_lp\_checker |  |
| **Category - NS\_ACESLVBRDG\_CORE \_LP\_CHECKER** | | |
| 1 | ns\_power\_sequence\_element\_lp\_checker |  |
| 2 | ns\_power\_intf\_iso\_lp\_checker |  |
| 3 | LP\_AR\_Q\_EMPTY\_CHECK |  |
| 4 | LP\_R\_Q\_EMPTY\_CHECK |  |
| 5 | LP\_AW\_Q\_EMPTY\_CHECK |  |
| 6 | LP\_W\_Q\_EMPTY\_CHECK |  |
| 7 | LP\_B\_Q\_EMPTY\_CHECK |  |

# Coverage

## Steps For Coverage Collection

**One config and single run:**

      1. buildSim -m tb\_axi1\_axi4m\_axi4s\_lp\_v2 -J 3 -g 4 -c 3 -N noc\_dev

               -c 2 functional coverage

               -c 3 functional + code coverage

      2. runSim -m tb\_axi1\_axi4m\_axi4s\_lp\_v2 -d nemesis -O auto\_slave\_prop\_file\_9.txt -A auto\_amap\_axi\_generic\_4.txt -Q 10 -w 1 -X 5 -T 1000

      3. imc -64bit -load sim/ tb\_axi1\_axi4m\_axi4s\_lp\_v2/nemesis/cov\_work/scope/test\_sv1/

      4. OR enter command “imc -64bit” and load the test\_sv1 by selecting the path in the GUI

**One config and multiple runs:**

      1. runRegress -f axi\_nocs/regress/regress\_list\_1m1s\_cov -q -5 -j 20 -N noc\_dev (Example)

                -c 2/3  is specified in the file “regress\_list\_1m1s\_cov”

      2. To merge coverage from multiple runs

                a. imc –batch (enters imc command/batch mode)

                b. merge ./sim/\*/\*/cov\_work/scope/test\_sv1 –out all  (all can be any name)

                c. merged directory will be in ./cov\_work/scope/all

      3. imc -64bit –load cov\_work/scope/all/

**Note : Two different config and single run for each config merging will give lot of warnings/errors.**

      1. tb\_axi1\_axi4m\_axi4s\_lp\_v2\_clockskew

      2. tb\_axi1\_axi4m\_axi4s\_lp\_v2

While merging the coverage from above configs, I see lot of below mentioned warnings.

\*W,WEMIN3:         Instance 'top.ns\_soc\_ip0.u\_ns\_fabric.u\_ns\_system.u\_ns\_system\_ns\_hfnets.u\_ns\_hf\_sleep\_ack\_deasserted\_pwrm\_0\_11' and its hierarchy not merged because it is not present in target instance 'top.ns\_soc\_ip0.u\_ns\_fabric.u\_ns\_system.u\_ns\_system\_ns\_hfnets'. To merge this instance, use the 'merge\_config' command (prior to the 'merge' command) to map the source instance/type, from the secondary run, to its corresponding location in primary run.

# Open Issues

Techniques

Technologies

Standards

Stages

JIRAs

Protocols

Power Aware

Tools

# References

<https://wiki.ith.intel.com/display/EN/Low+Power+Functional+Verification>

<https://intelpedia.intel.com/SoC_Low_Power_Design#Quick_Links_on_UPF_Tools.2FFlow.2FMethodology>

<>/trunk/Docs/internal\_docs