A

DISSERTATION REPORT

ON

Power Management Validation using Accelerated Platform

Is submitted as a partial fulfilment of the degree of

MASTER OF TECHNOLOGY IN VLSI Design

BY

ULLAS

(2017PEV5111)

UNDER THE GUIDANCE OF

Dr. C. Periasamy

&

Neeraj Gupta

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY JAIPUR (JUNE 2019)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY
JAIPUR (RAJASTHAN) – 302017

Certificate

This is to certify that the dissertation report entitled **Power Management Validation using Accelerated Platform** submitted by **Ullas (2017PEV5111)**, in the partial fulfilment of the Degree Master of Technology in **VLSI Design** of Malaviya National Institute of Technology, is the work completed by him under my supervision, and approved for submission during academic session 2018-2019.

Date : 11 July 2019

Place : Jaipur

Dr. C. Periasamy
(Project Supervisor)
Professor
Dept. of ECE
MNIT Jaipur, India

CERTIFICATE

This is to certify that the thesis entitled "**Power Management Validation using Accelerated Platform**" is bonafide report of the work done by **Ullas** (WWID-11828090) in partial fulfilment of the requirement for degree of Master of Technology in VLSI Design, **Malaviya National Institute of Technology, Jaipur**. The project was carried out at **Intel Technology India Private Limited**, **Bengaluru** during the period June-2018 to June-2019 under my supervision and guidance.

External Project Supervisor

Date: 11th July 2019

Neeraj Gupta

Place: Bengaluru

(Pre-Si Verification Engineer)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING
MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY
JAIPUR (RAJASTHAN) – 302017

Declaration

I, Ullas, declare that this Dissertation titled as "**Power Management Validation using Accelerated Platform**" and the work presented in it is my own and that, to the best of my knowledge and belief.

I confirm that the major portion of the report except the refereed works, contains no material previously published nor present a material which to be substantial extent has been accepted or the award of any other degree by university or other institute of higher learning. Wherever I used data (Theories, results) from other sources, credit has been made to that source by citing them (to the best of my knowledge). Due care has been taken in writing this thesis, errors and omissions are regretted.

Date: 11/07/2019 **Ullas**
Place: Jaipur ID : 2017PEV5111

Acknowledgment

*I take immense pleasure in thanking of gratitude to my guide **Dr. C. Periasamy**, Professor, Malaviya National Institute of Technology (MNIT) Jaipur for being a source of inspiration and for timely guidance during the project. The supervision and support that he gave truly helped in the progression of my thesis. I am highly obliged to him for his valuable advices and moral support during research period.*

*I wish to express my deep sense of gratitude to my Internship co-supervisor **Neeraj Gupta**, Pre-si Verification engineer, Intel Technologies India Pvt. Ltd. Bangalore, and my manager **Mamta Garg**, Engineering Manager, Intel Technologies India Pvt. Ltd. Bangalore, for having permitted me to carry out this project work. Through-out the internship, they had given me much valuable advice on project work which I am very lucky to benefit from.*

*I would like to express my gratitude and sincere thanks to our Head of Department (HoD), **Dr. D.Boolchandani**, Malaviya National Institute of Technology (MNIT) Jaipur for allowing me to undertake this thesis work and for his guidelines during the review process.*

*I would also like to express my sincere thanks to **Santhosh kumar**, **Yoga Priya Vadivelu**, and **Sahil Madan** for helping me to carry out my final year thesis work at Intel, and all other team members for supporting and guiding me during the thesis.*

Ullas

Abstract

Every design verification technique requires coverage metrics to assess the quality of the design and determine when the design it is robust enough for tape-out. This project contains the functional coverage collection flow of full chip power management and approach to increase the Coverage Percentage. At end of the Project Coverage Percentage of 88% is achieved. This project also explains about the debugging of the full chip power management test cases through emulation.

First step towards the coverage collection of FCPM is get all the scenario to be tested. During this project test plan document is prepared consisting of all the scenario as Power states are the major functionality that need to be covered are identified and added in coverage module of the Power Checker.

After running the test case, Power Checker runs on post processing mode to generate the Virtual Database. This database has the coverage information. All the database of the individual test cases in the project are combined into one database to get consolidated coverage information and final database is used to analyze missed cover items.

There are many approach to analyze missed cover items. In this project few of the approach are discussed. Missed cover items are analyzed through simulation waveform. Uncovered items are to be covered through identifying proper test case. Sometime cover items are missed if required switches or fuses are disabled on SoC. These fuses are responsible for enablement of particular functionality on the SoC.

During the entire flow, many tools are used. Verdi- Coverage Point viewer is used to check which missed and hit covered items. Verdi is also used to see the simulation waveform. Urg is used to combine all the database of different test case into one.

Abbreviations

SoC	System on Chip
IP	Intellectual Property
FCPM	Full Chip Power Management
RTL	Register Transfer Logic
HVP	Hierarchical Verification Path
FSM	Finite State Machine
GPC	Global Power Checker
VDB	Virtual Database
TLM	Transaction Level Modelling
PCU	Power Control Unit
DTS	Digital Thermal Sensor
RPT	Report
DUT	Design Under Test

List of Figures

Contents

Certificate

Declaration

Acknowledgements

Abstract

Abbreviations

List of Figures

Chapter 1 Introduction

1.1 Pre - Silicon Validation

The current VLSI technologies develops the SoC in different phases. RTL design, pre- silicon verification, physical design and post-silicon validation are the major steps in the VLSI product development cycle. The RTLdesign phase encodes all the specification and functionality of the product. The verification phase verifies weather the RTL design is matching the specifications or not. Pre-silicon verification phase verifies the functionality of the design at RTL level whereas post-silicon validation phase verifies the functionality at silicon. The product should match with the specification at every phase of verification. Pre silicon validation is one of the most important validation phase. Pre silicon validation phase verify the correctness of design. Basically it's required modelling of the system.

1.2 Motivation

Pre-silicon debug and validation is most important phase in the product development cycle of SoCs. The system or product should work properly at silicon level as per the specification. Nowadays, verifying the system performance at silicon level is very difficult as well as very expensive task. If the SoC fails at silicon levels, debugging the failure becomes the major challenge. SoCs have very high complexity. So, before tape out, it is required to make sure that all the critical scenarios are covered to avoid any silicon failure. Here, the coverage plays very important role since verification engineer assess coverage results and make critical decisions on

what to do next. In fact, for the verification of large, complex system-on-chip (SoC) designs, coverage metrics and the responses to them guide the entire flow. Higher coverage percentage gives confidence whether product is ready for tape out.

1.3 Objective

The Objective of the project is:

- To explain about the flow of coverage collection
- To explain about coverage architecture
- To explain about approaches to increase the coverage percentage
- To explain about power management test debug

1.4 Organization

- In this thesis, Chapter 2 is the literature survey which will discuss about importance of low power SoC and
- Chapter 3 Type and importance of the coverage
- Chapter 4 Power States and Importance of each states.
- Chapter 5 Steps to collect the coverage.
- Chapter 6 Architecture of Coverage Collection.
- Chapter 7 Analysis of missed cover items.
- Chapter 8 Approach to debug power related test
- Chapter 9 Results and Discussion

Chapter 2

Literature Survey

2.1 Requirement of Low Power SoC Design

In the application, there is an ongoing quest for more functionality, performance and integration within SoC which in turn leads to power dissipation in the range of hundreds of watts. This has been specifically observed in the recent Intel processor variants like Itanium2 with the dissipation reaching 120 Watts. Devices in this class require interacted packing, heat sinks and a cooling system.
All these culminate in multiple problems that need to be sorted out to conserve the practicability of the upcoming applications. The enhanced integration of mobile applications required greater dependency on the battery lifetime of the system over preceding generations. Although the headway in CMOS Technology has caused a twofold increase in the transistor density every 18th months, the corresponding improvement in battery technology exceeds 5 years duration.

In the current technology trend, smaller devices are gaining importance. To achieve smaller devices, aggressive scaling is done. Scaling of the devices are done to accommodate more number of transistor in lesser area but due to this many problem arises. Few are mentioned below:

- Increase in leakage power.
- Reduction in yield due to process parameter variations.

- Reliability is reduced
- Testing is becoming difficult

Packaging and cooling cost is dependent on the amount of power dissipation by the chip. Generated heat on the chip should be taken out; otherwise it will start malfunctioning.

Demand of the portable systems and hand held devices such as phone, palmtop etc. is increasing at high rate. Not only this, limitless functionality with limited power supply is the requirement. Limitless functionality will increase power consumption and battery life is becoming primary concern but unfortunately the battery technology has not kept off with the energy requirement.

Reliability is the major issue as we move into deep sub-micron. Increase in the power dissipation leads to increase in temperature and it has been found that for every 10 rise in temperature, doubles the failure rate.

In offices, 80% of the power consumption is due to computer, printer and other related devices and many of the time, these system are not in use but still they are on. So the power management unit will automatically turn off the screen and the other power hungry components are shut down. Power dissipated by these devices are mostly in the form of heat and normally some cooling technique such as air conditioner is used to transfer the heat to the environment.

So, the only way out is to go for low power devices.

2.2 Coverage, the Heart of Verification

Importance of Coverage

Every design verification technique requires coverage metrics to gauge

progress, assess effectiveness, and help determine when the design is robust enough for tape-out. At every step of the way and with every bug-finding technology and tool, verification engineers assess coverage results and make critical decisions on what to do next.

In fact, for the verification of large, complex system-on-chip (SoC) designs, coverage metrics and the responses to them guide the entire flow. The term ”coverage-driven verification” describes a methodology built around coverage metrics as the primary way to manage verification.

Coverage is used to measure tested and untested portions of the design. Coverage is defined as the percentage of verification objectives that have been met.

Code Coverage

Code Coverage measures how much of the design code is exercised. This include execution of design blocks, no of lines etc.

Coverage-driven verification is made possible by the wide range of structural coverage information available in modern verification tools. The most traditional form, RTL code coverage, has migrated from specialized add-on tools directly into the more advanced simulators, providing much better performance and ease of use.

Once limited to line coverage, today’s code coverage metrics may also include toggle, condition, path and finite-state-machine (FSM) coverage. These metrics can be gathered automatically in simulation, under user control to select or exclude specific metrics or portions of the RTL.

Code coverage is very helpful at identifying” holes” in verification: if a section of code has not been exercised then it has not been verified. However, high code coverage metrics do not necessarily mean that a

design is bug-free or that the verification effort is complete and thorough. Although code coverage is valuable, it should be supplemented by the specification of functional coverage points that must be exercised for thorough verification.

Functional Coverage

It is user defined metric that measures how much of the design specification has been exercised in verification. These allow designers to specify corner cases based on their knowledge of the implementation.

For example, functional coverage might track whether the verification process has filled and emptied every FIFO in the design.

Assertion Coverage

It checks whether sequence of behavior have occurred. Assertions, an essential part of modern SoC verification, can also provide valuable coverage feedback. Knowing which assertions pass in simulation and which ones fail is one form of coverage. Any assertions failing indicate that functional bugs have been found. However, successful assertions do not provide any run-time feedback information: they may have succeeded or they may not have had the opportunity to execute at all.

Conclusion

Figure8.6summarizes the different sources of coverage metrics used in a modern verification process. Thiswiderange ofcoverage information helps verification teamsassess progress and determine what to do next.

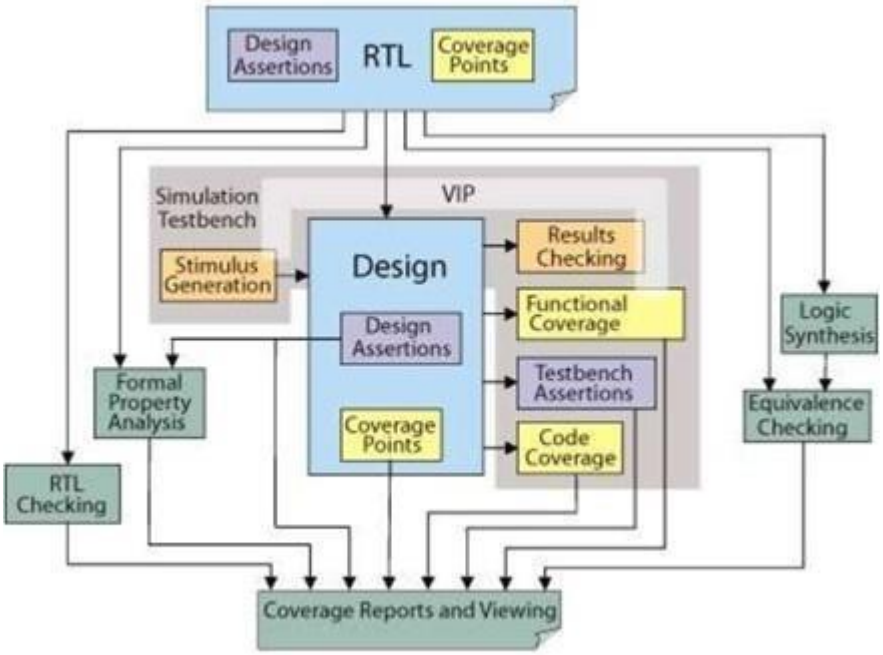


Figure 2.1: Verification management requires a broad range of coverage metrics

Chapter 3 Power States

This Chapter explains about the power states and its type.

Power States are the Intel's Technology to optimize the power consumption.

3.1 P-states

P-states are the power optimization technique by CPU and operating system during code execution. In this, based on the requirement, CPU will operate at different frequency and hence voltage. P0 is the highest frequency state. It reduces the power consumption without impacting the performance. The power management software periodically mon- itors the processors utilization. If the utilization is less than certain threshold then it enters in next higher P-state.

3.2 C-states

In this technique, power optimization is done during idle mode (i.e. when no code is executed). There are two types of C-state:

- Core C-state
- Package C-state

The processor has up to 60 plus cores in a package. Core C-state is for each core and Package C-state.

From Figure3.1, we can see different parameter related to C-state. For e.g. in C0 state

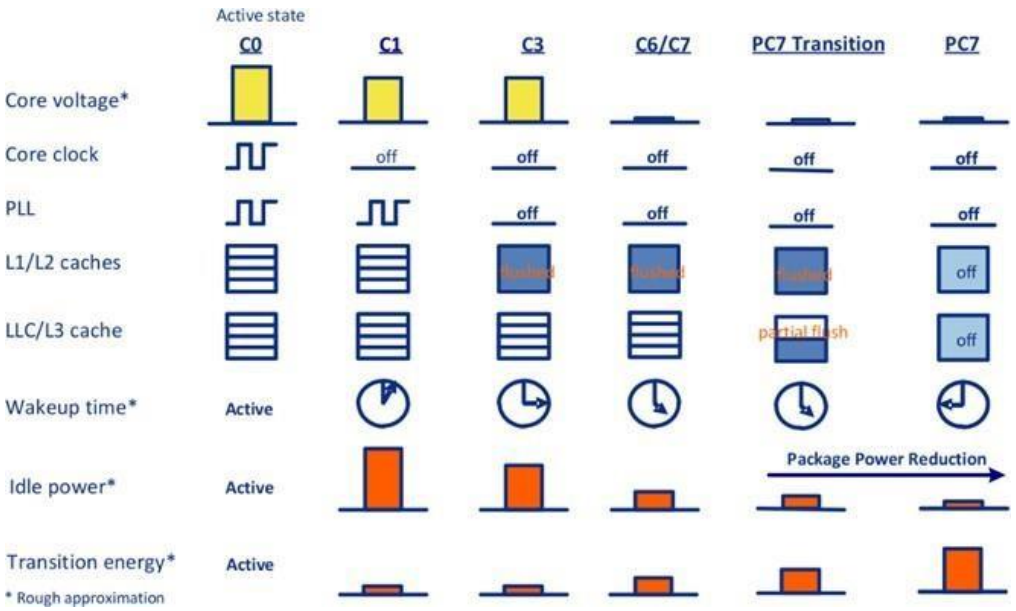


Figure 3.1: Overview of the C-state and PC-state

Core voltage is more as compared to C7 state. Core clock is on in C0 but off in C1, C2, C3, C7. Idle power is highest in C0 and minimum in PC7.

3.3 Dropping in deeper C-state

From Figure3.1, it is evident that as processor goes in deeper C-state, wake-up time is increasing. So the price of dropping in deeper C-state is added latency while core is trying to wake-up.

From the figure3.2Case 1, next interrupt is far when core when core is in C1, so in this case core will go to C6-state. But in Case 2, two interrupt are close enough. If core will drop in C6-state then response time or wakeup time will more.

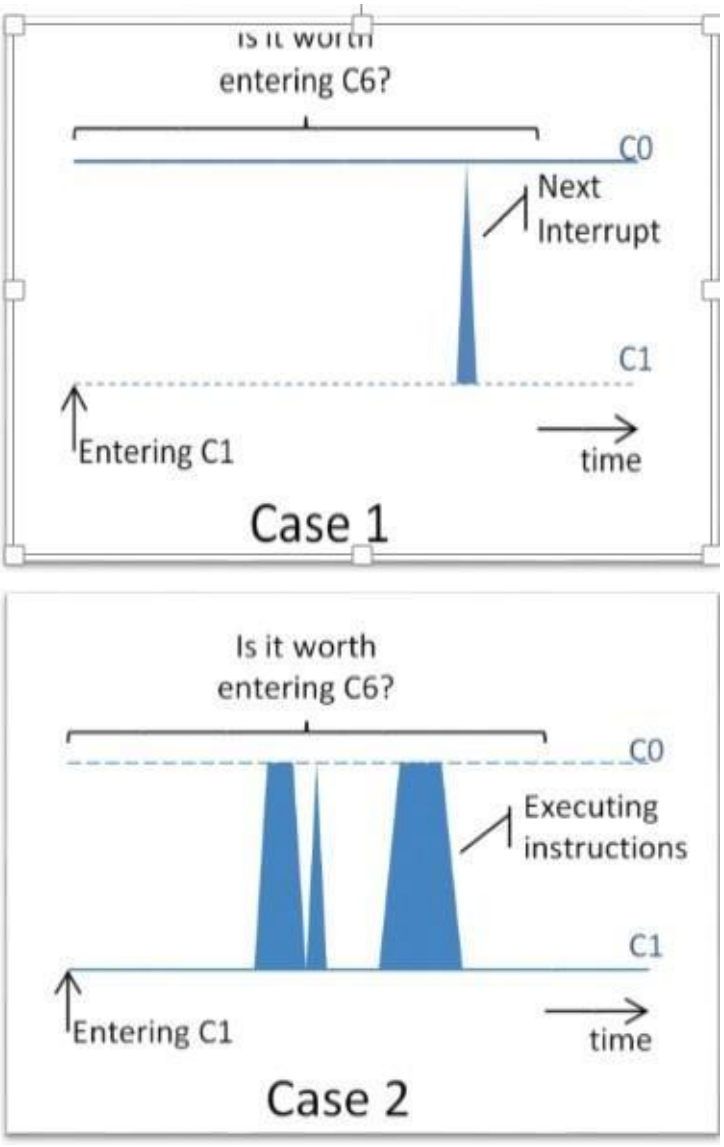


Figure 3.2: Is it worth entering in C6-state

3.4 T-States

T-state is known as Thermal Throttling state in which, it saves the processor from over- heating and hence burning itself. It also has different T-state which depends on the junction temperature. When the sensor register that junction temperature is reaching certain level, Hardware power manger places the processor in different T-states depending upon the temperature using clock gating technique. Different T-state is achieved by the percentage of clock gating. For e.g. when the percentage of clock gating is 10%, it is T1 state or when it is 25% it can be T2 and so on.

Chapter 4

Steps to Collect Coverage

This chapter discuss about the way to get the overall functional coverage percentage.

4.1 Coverage Flow in SoC

To understand the coverage, it is required to know the complete verification flow of coverage technology. Below figure4.1depict the complete verification flow diagram:

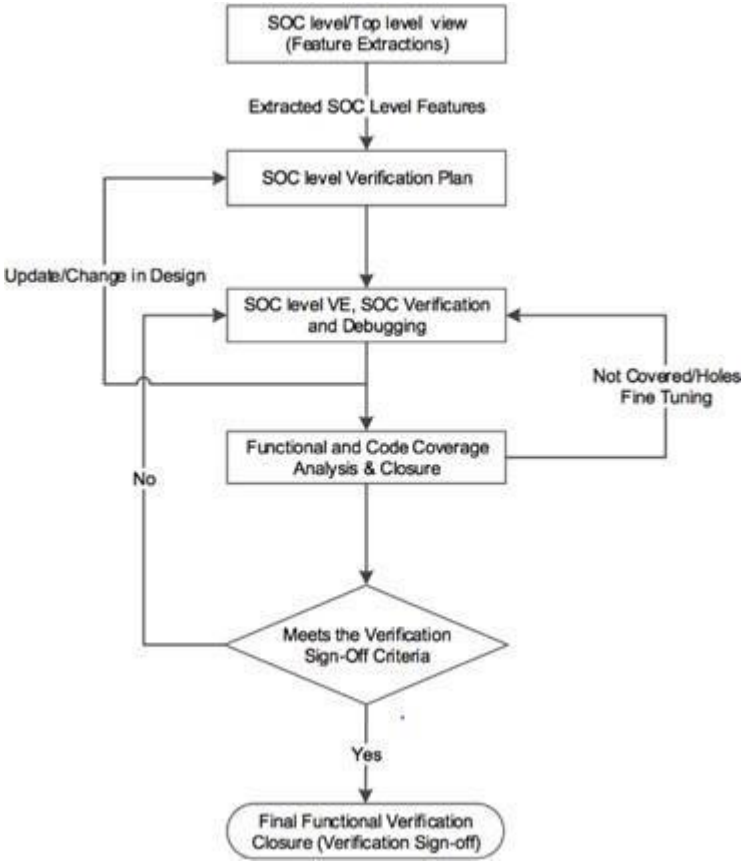


Figure 4.1: Coverage Flow in SoC

4.2 Coverage Collection Flow

Below figure4.2shows the steps involved for coverage collection:

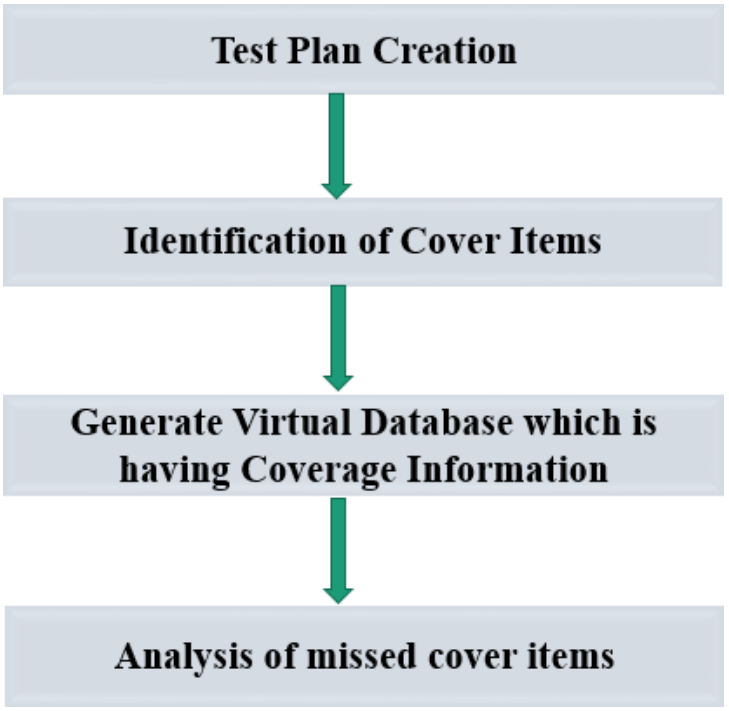


Figure 4.2: Coverage Collection Flow

4.3 Test Plan Creation

First step toward coverage collection is creation of the Test Plan. Planning is the most important phase of SoC verification. It contains all the scenarios to be tested at a com- mon place which makes tracking easier because manual collection of the data may cause in missing scenarios and hence reduction in coverage percentage.

In this project Hierarchical Verification Plan also known as hvp is used. In this all the cover groups and coveritems which need to be tested is defined in predefined format.

Below figure4.3shows the way to define the hvp file. After the creation

of the hvp file, it is uploaded in a tool called Verdi which shows all the cover items with respective hit and missed. From there missed cover items are analyzed.

Figure 4.3: Hierarchical Verification Plan

Figure 4.4: Verdi as a Coverage Point Viewer

4.4 Identification of Cover Items

This section explains identification of cover groups and items for functional coverage metrics.

Functional Coverage Metrics measure the verification program based on functional requirement. There are many way to do functional coverage measures. One among them is Cover Group which is explained in this report. It consist of the state values observed on the buses, grouping of control signals.

Cover Groups are identified based on the FSM defined in reference model and also on control signal.

Below are the few example of the important functionality for which Cover Group and items has to be identified:

Example 1: Below is the example for one of the P-state FSM

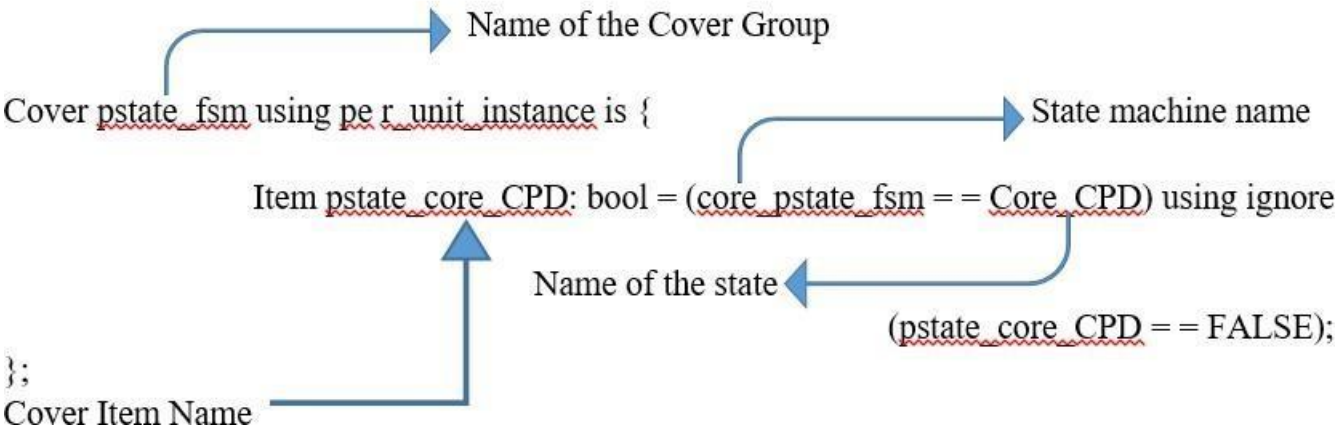


Figure 4.5: Example of P-state Cover Item

Example 2: When P-state cross C-state

```
Cover pstate_cross_cstate using per_unit_instance is {

        Item pstate_CC1: bool = (core_pstate_fsm = = Core_PLL_off_&& core_cstate_fsm = =
                                                                    CC1 ) using ignore

                                                                    (pstate_CC1 = = FALSE);
};
```

Figure 4.6: Example of P-state Cross C-state Cover Item

Prochot: It means processor hot. When the CPU reaches its maximum temperature (100 to 105C) then prochot signal initiates thermal throttling so that CPU can reduce its speed and save itself from over heating.

Example:

Example:

```
Cover prochot_fsm using per_unit_instance is {
                                    Item prochot: bool = (prochot_pin_&& core_enable) usi
                                  (prochot = = FALSE);
};
```

Figure 4.7: Example of Prochot Cover Item

After adding the cover items, now we have to check whether the addition of the cover items have introduced any error in the model. To check for the error, run a script known as run pc.pl in the test result folder of the test case of the same model. run pc.pl is the Perl script is which runs the power checker and generate desired logs andresults.

Below figure shows results after the GPC run. The highlighted part shows that there is no syntax error and warning. In case of any error, it

```
TLM report: Total number of Transactions =32838    Total trans frequency=864
TLM report: Last Period trans number     =22       Last period trans frq=22
TLM report: Cycles per second (frequency)=1759454  Total Test time(secs)=38
Last specman tick - stop_run() was called
Normal stop - stop_run() is completed
Checking the test ...
Checking is complete - 0 DUT errors, 0 DUT warnings.
[66859272] in bitvec_u-@2 at line 1936 in @common_bitvecdkdc_type: INFORM: Ending bitvec cache size:
Reusing existing coverage model  ./cov_work/scope/sn_00000000_3ea77802.ucm
Wrote 1 cover_struct to ./cov_work/scope/common_top_sn1/sn_00000000_3ea77802.ucd

sh: quit: command not found
GPC PASS
sccj004534>
```

Figure 4.8: Error check of power checker

4.5 Generation of Virtual Database

Once all the cover items that needs to be tested are added in the Power Checker then all the test cases are run. Along with test run, Power Checker runs to generate the Virtual

Database (vdb) for each individual test cases. This vdb has coverage information.

Now when each test case has vdb, next step is to combine all the vdb into one con- solidated vdb to get overall coverage information. URG is a Synopsys tool which is used to combine all the individual vdb(s) into one final merged virtual database.

This merged vdb is opened along with the test plan in Verdi to find hit and missed cover items.

4.6 Analysis of Missed Cover Items

Missed cover items are analyzed through different tracker and simulation waveforms. This is discussed in detail in Chapter 6.

Chapter 5 Power Checker

This chapter describes about Power Checker and its uses.

5.1 Introduction

Power Checker is a tool that contains high level power management checker, tracker and coverage for FC power validation. It runs on both simulation and emulation. It can also run on lower platform by disconnecting some of the features.

5.2 Structure of Power Checker

Power Checker structure is divided into two layer. One is the TLM which is present in RTL code and another layer is power checker component that is in Specman.

The figure5.1SHOWS the component of power checker:

TLM: TLM are scattered in several places in the core and uncore. Each TLM in RTL code, collects the signals information from the RTLs unit signal into a data structure and sets a valid bit, if there is any update to signal. Valid bit triggers a DPI call that samples the data structure to traces that will be used when power checker is running in post-processing mode. The TLM reader of the power checker converts the traces of the TLM into a structure and send it to the monitor using method port.

Monitor: It collects all the data from the TLM reader and combines all into one data structure which will be further be used by reference model, checkerand tracker. Monitor has input and output method port. Through input port it receive signal values. And

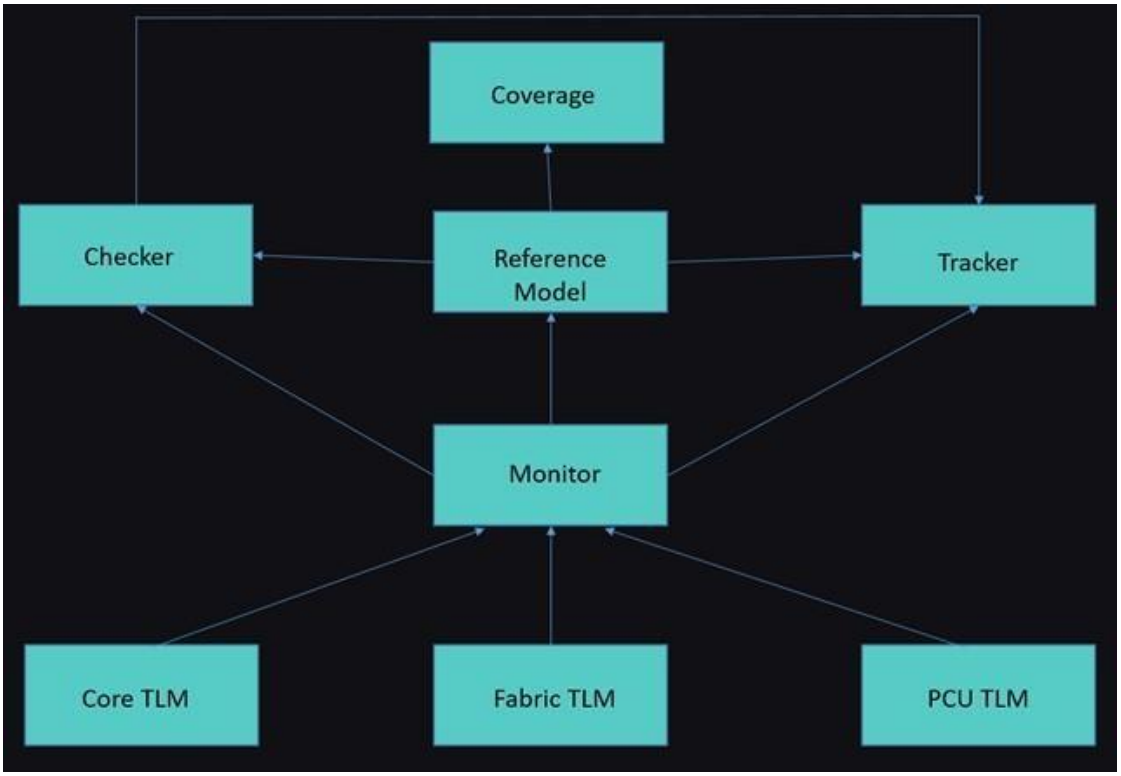


Figure 5.1: Component of the Power Checker

through output port it transfer signal values to reference model, checker and tracker. For FC validation all the method port are connected. To use power checker for lower platform, only required method port are connected.

Reference Model: It contains FSM for the all the features. Features are Thread C-state, Core C-state, Core P-state, TT1, Core S1, ICCP, Ring P-state, Package C-state and Package S-state. Core C-state and Core P-state reference model are shown in figure
5.2. Each FSM describes the features flow and the states represents the RTL current or

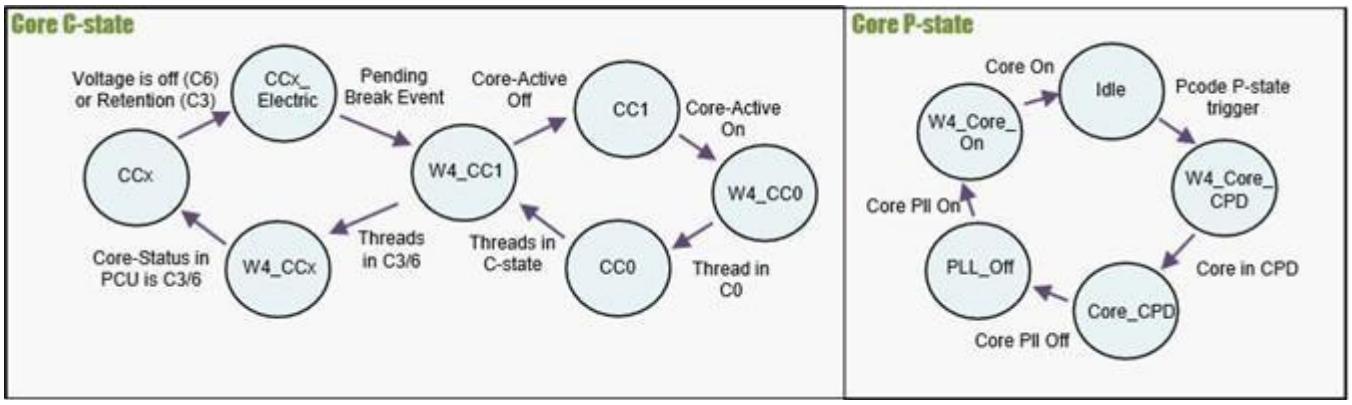


Figure 5.2: Core C-state and Core P-state reference model

expected state. For e.g. when MWAIT command is executed by thread then the FSM state will move to W4 TCx state from the running state (CCo).

Checker: Each feature has a separate checker. It contains two types of checking.

- Timeout Checking of Temporal states
- Checking Rule for each state of the FSM

The states which are clock depend and voltage change is limited by time is known as temporal state. For each state of the FSM, there is checking rule that verify that relevant indications have the expected value during the state.

For example, in figure5.3, core clock(MCLK) is on in Idle state and off in Core CPD and Core Pll off state(Core is asleep). So these are the checking rule for each state of the FSM.

State	Timeout	MLC
Idle	No	IDI On CPD uninhibited APIC timers run MCLK run MNSCLK run
W4_Core_CPD	Yes	IDI On MNSCLK run
Core_CPD	Yes	CPD inhibited APIC timers stop MCLK stop
Core_PLL_Off	Yes	IDI Shutdown CPD inhibited APIC timers stop MCLK stop MNSCLK stop
W4_Core_On	Yes	

Figure 5.3: Core P-state checker

Tracker: It prints the following under respective column with the time stamp.

- Current state of each features of all the FSM
- Signals value

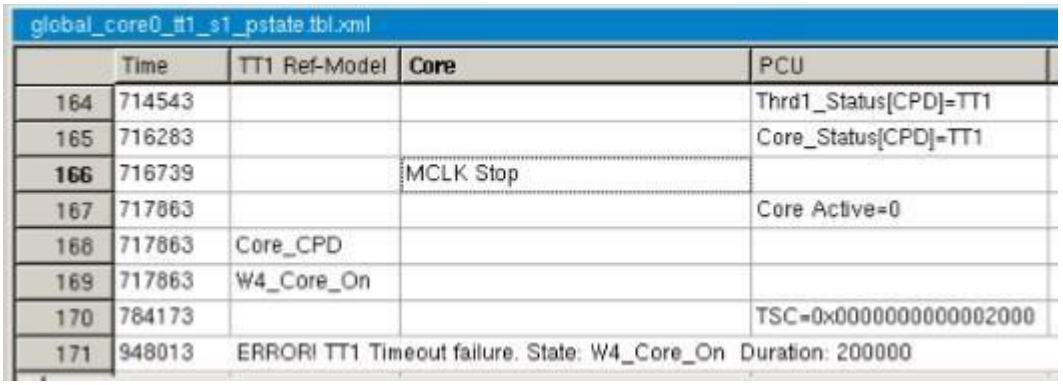
Figure5.4is package C-state tracker. So, in the first column current C-state is printed. The signals which are coming from PCU, Cores etc. are displayed with the current value under respective columns.

global_pkg_cstate.tbl.xml

	Time	Cst Ref-Model	PCU	Cores
75	771483		PM Req: EA=0 CF=1	
76	786443			
77	787553		PM Rsp	
78	791843			
79	798263		MPH Block	
80	805528		TSC=0x0000000000004800	
81	847258		Trigger Ring C7	
82	847258	W4_PC2R		
83	849048			
84	849608			
85	849608	PC2R		
86	854483			
87	854563			
88	854643			
89	854723			
90	861923			
91	887448		TSC=0x0000000000005000	
92	887603			
93	890313		Trigger MC Shutdown	
94	890313	W4_PC6		
95	895448			
96	895883			
97	898123		Pcode/Vcode Hit	
98	899323			
99	900003			
100	900003	PC6		

Figure 5.4: Package C-state tracker

When Checker fails, then the error message will be printed in the tracker as shown in below figure:



global_core0_tt1_s1_pstate.tbl.xml

	Time	TT1 Ref-Model	Core	PCU
164	714543			Thrd1_Status[CPD]=TT1
165	716283			Core_Status[CPD]=TT1
166	716739		MCLK Stop	
167	717863			Core Active=0
168	717863	Core_CPD		
169	717863	W4_Core_On		
170	784173			TSC=0x0000000000002000
171	948013	ERROR! TT1 Timeout failure. State: W4_Core_On Duration: 200000		

Figure 5.5: Tracker with error message

Coverage: This consist of Coverage Groups and items based on the functionality. It generate vdb for all the test cases which can be used as for analysis of coverage metrics. Coverage metrics is indication of the quality of regression or single test.

5.3 Advantage of Power Checker

Robust FC level checking: Most of the FC validation depends on self-checking test and memory dump which is not sufficient for power management validation because this method will not detect bug like wrong clock ratio after a GV- transition.

Fast development and maintenance: FC checker and tracker are difficult to develop because compilation is time taking process. But this power checker is highly modular. It can be developed in part and then later can be integrated to FC. Fixing of checker and recompiling will take very time as it provide post-process rerun of the power checker.

Chapter 6

Analysis of Missed Cover Items

In this chapter different approaches are discussed for the analyses of missed cover items.

The First steps towards analyzing the missed cover item is to find out which signal is responsible for not hitting the cover items. To know this signal value should be known. In Power checker, in tracker module add the required signal. Since tracker function is to print the signal with value, the signal responsible for not hitting the cover items value can beknown.

Now there are many reasons for missing the cover items.

- Sometime the case is that the test case is unable to meet the required conditions for the particular cover items. In such situation different test case which can hit the cover items are run or modification is made to the test case to satisfy the conditions of the cover items. For e.g. Thermal Throttling scenarios (T-state) can be hit when the processor reaches to certain temperature. To achieve this test needs to run for more number of cycle but the originally test were running for less number of cycle than it actually need to create the Thermal Throttling.

- Another situation is, wrong signal is being referred by Power Checker. When the power checker runs it takes the signal value

from RTL and send it to coverage module through TLM. And when the power checker assign wrong signal value from design to coverage module, then cover item wont get hit. For e.g. suppose a cover item is written to test that Core 1 should reach to C6 state but in actual design

Core 1 is defined as Core[2], so on this kind of situations cover items won't get hit. To verify mapping of the signal from design to coverage module, waveform in generated. And in waveform, both design and power checker signal value can be seen and required modification is made to hit the cover items.

- Next example is for the missed scenarios related to temperature like DTS. DTS stands for Digital Thermal Sensor. It shows the difference between current tem-perature and maximum junction temperature. After analyzing the waveform, it is noticed that RTL design has correct values of the signals but Power Checker has all 0s.So finally, it is found that some fuses are not enabled in the SoC which are required to enable temperature related scenarios.

Chapter 7

Full Chip Power Management Validation Strategy

This Chapter discuss about few

- Tools for FCPM Validation
- Tool for writing Test Cases
- Approaches to debug the failure of power related test cases.

7.1 Tools for FCPMValidation

- Simulation
- Emulation

As long as the DUTs size is manageable and the simulation time is in a day or less, HDL software simulators are the best choice for hardware debug. They are easy to use, quick to setup, extremely fast to compile the DUT, and superbly flexible with regard to debugging a hardware design. Furthermore, they are also reasonably priced. However, they become challenging at the system level when the DUT reaches into several tens of million gates. At this level it takes simulation times in days so it increase

time to market. When the DUT reaches into several tens of million gates that time for reduce time to market Emulation is good choice. So at FC level emulation is good choice to validate power management.

7.2 Tool for writing Test Case

Require one compiler which covert any higher level languages such like system Verilog, C, C++, Perl, Python to machine level language which we use in simulation or emulation method. Intel have its own tool for writing test. It's targeted to create tests from fully random to much directed flows using a powerful constraints solving technology. **Key features of tool:**

- An expressive test specification language for modular, abstract, maintainable test writing.
- Ability to direct tests towards interesting cases using constraints, biasing, and heuristics.
- A highly-controllable, constraint-based test generator capable of generating a full spectrum of tests, from highly random to highly directed tests.
- A declarative, maintainable, expandable model of IA32 architecture.

7.3 Forcing Signals

There are some functionality which can be verified only by forcing signal values for e.g. DTS, prochot etc. In simulation, we can force signal with any value at any time stamp through test case. But in emulation we cannot force signal through test case. Here, we can force signal using injector. For that we have to write inject script. Here the time at which values to be injected are defined in the script.

7.4 Approach to Debug Power Management Test

In emulation, there is no waveform through which debug can be done but different trackers and log files are generated once the test run gets over. These logs gives the picture of what processor is doing at different time and what are the values of the signal. Next step is to analyze the log files and to find out which signal or register values are not set properly. After finding this, waveform is generated for the particular time interval. Next is

to tracker the drivers of the signal and find the signal which is not set with the correct value.

Chapter 8

Results and Discussion

8.1 Coverage Percentage

Below chart shows that the coverage percentage of 88% is achieved at the end of the project.

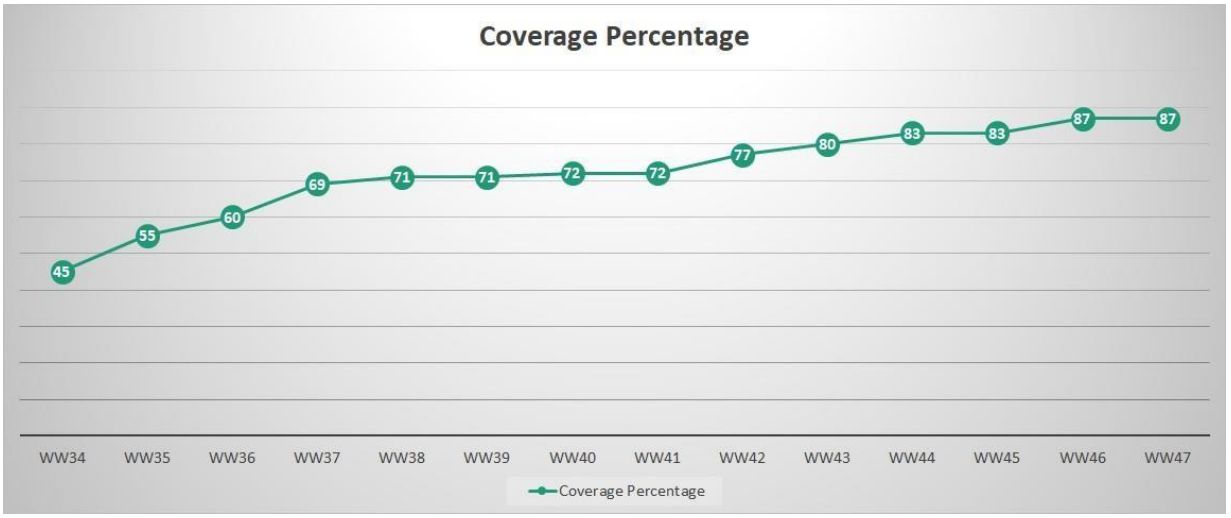


Figure 8.1: Weekly Coverage Percentage

8.2 FCPM Test Debug

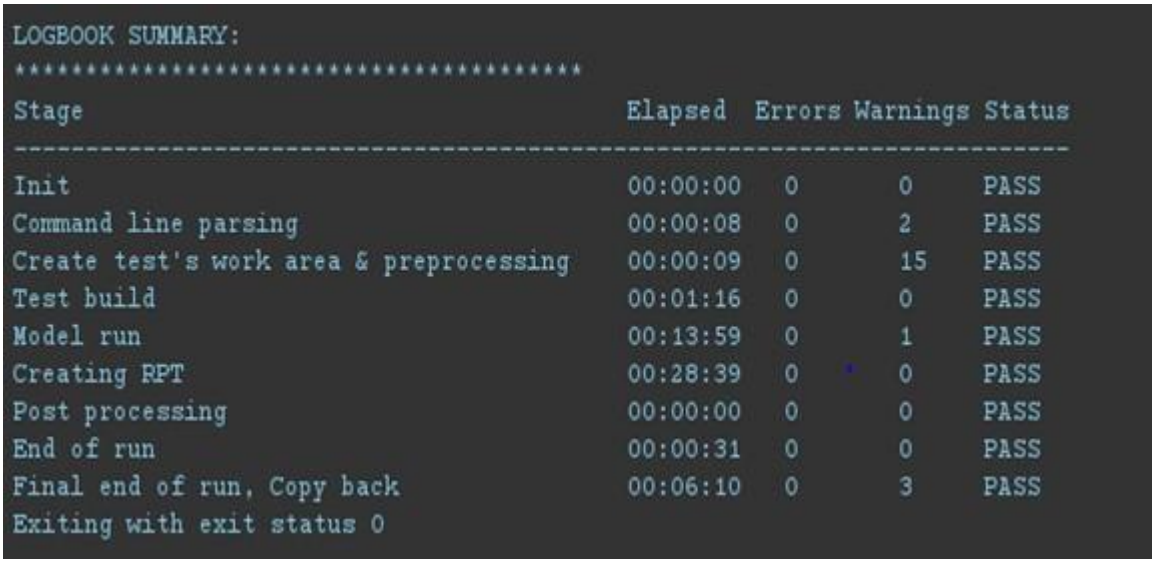
When the test runs, it goes through below stages:

- **Command line Parsing: -** In this stage, fuses are set using switches. Doing this some features are disabled or enabled and depending upon the test plan, flow of the test can be controlled.

- **Create tests work area: -** In this stage, environment is set depending upon project.
- **Test build: -** In this stage compiler compile test case and convert high level language to machine level language.
- **Model run: -** At this stage, converted machine level language loads on to the emulation board and runs test case on emulation board.
- **Creating RPT: -** In this stage, it create all transaction data trackers file which are more useful for debug purpose, coverage and checker. Post processing: - At this stage all post process script (like coverage and checker script) runs.

8.3 Result after Test Case Run

Figure8.2SHows Result after Test Case Run:



```
LOGBOOK SUMMARY:
****************************************
Stage                                    Elapsed  Errors Warnings Status
--------------------------------------------------------------------------------
Init                                     00:00:00   0       0     PASS
Command line parsing                     00:00:08   0       2     PASS
Create test's work area & preprocessing  00:00:09   0       15    PASS
Test build                               00:01:16   0       0     PASS
Model run                                00:13:59   0       1     PASS
Creating RPT                             00:28:39   0       0     PASS
Post processing                          00:00:00   0       0     PASS
End of run                               00:00:31   0       0     PASS
Final end of run, Copy back              00:06:10   0       3     PASS
Exiting with exit status 0
```

Figure 8.2: Logbook Summary

8.4 Debugging Power Management Flow

Log-1 This log is the first step towards debug. It shows the reason of the failure. Below are two main reason:

- No Halt Encountered: It means test case timed out after running for the full

cycle. This thing is common when we use to give large wake time and the cycle run is not sufficient to cover that wake time. Solution is to either decrease the wake time or increase the run cycle

- Processor dead: When the processor is not aced, test end checker run and write in EBx as dead and dead is printed for that particular processor.

- Processor is not going to the deep sleep state. This could be because of some fuse which is not allowing it to happen. We need to enable the fuse so that the processor could go to deep sleep.

Figure 8.3: LOG1 - Emulation Run

Log-2 This logs shows the series of the liner instructions pointer run by processor. From this log it can be find out at which instruction processor is stuck.

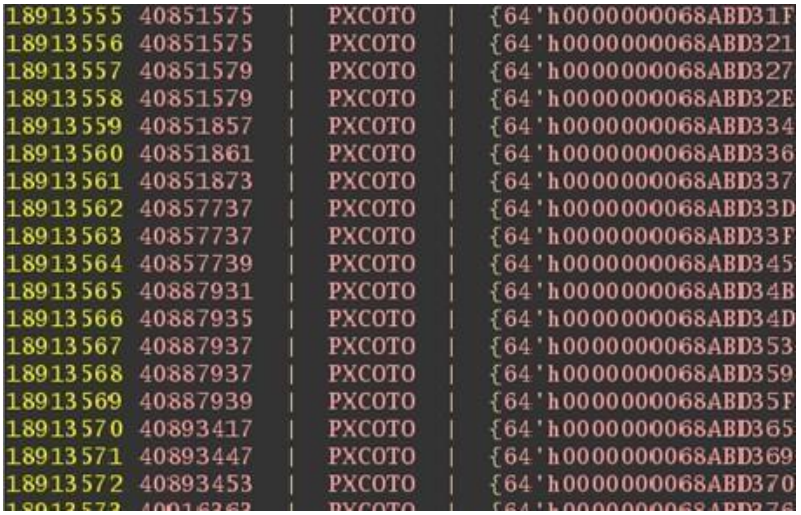


Figure 8.4: LOG2 - Linear Instruction Pointer

Log-3 After finding out at which instruction the processor is stuck, for furtherdebug,it is important to know that what that instruction is trying to perform. It information can be found in Log - 3 by searching for that instruction.

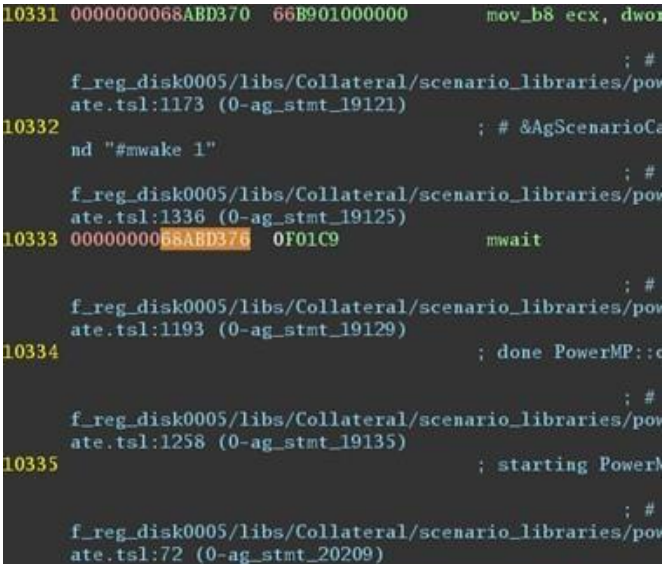


Figure 8.5: Log3 - Instructions Description

Log-4 This log shows all the operation performed by PCU and all the value set by it in registers which is helpful in further debug.

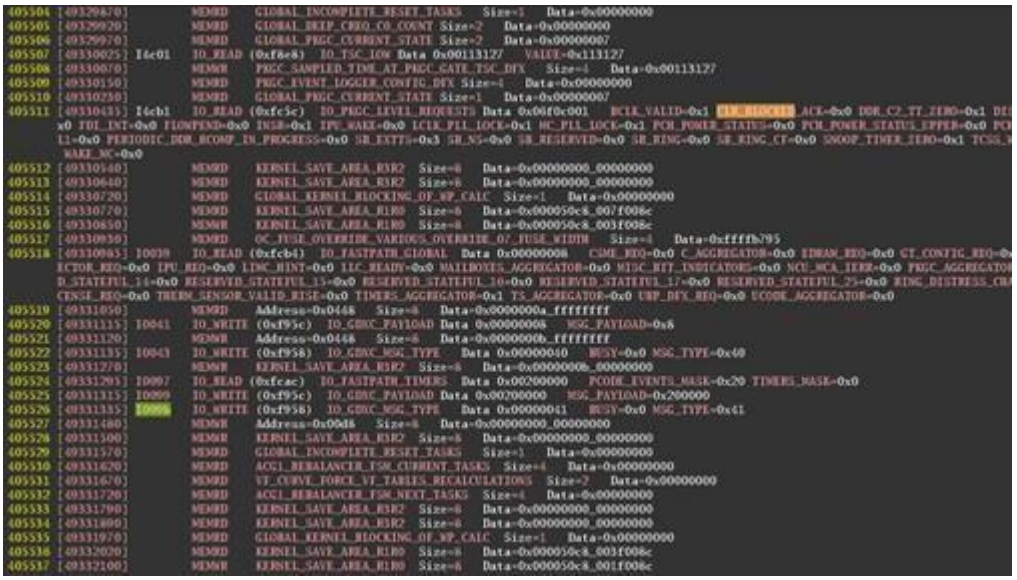


Figure 8.6: Log4 - PCU Tracker

Chapter 9 Conclusion

Power Management contains some of the more complex features in the CPU. Many components in the CPU are involved in Power Management flows. Since additional components are added in each CPU generation, and Power goals get more aggressive, the Power Management features become more challenging for design and validation. This enables shifting of Power Management validation to emulation. In emulation, waveform are not available and debug is performed from logs and trackers which has information about the signals value and operations performed by the processors.

Coverage is also another important aspect of Validation. It guide the engineer throughout the entire flow and also determine when the design is robust enough for tape out. In this project, process is discussed for coverage collection through a novel agent global power checker. Different approaches are discussed to analysis the missed cover items and ways to increase the coverage percentage.

Bibliography

[1]A-32 Intel Architecture Software Developers Manual Volume 3 Systems Programming Guide

[2] Yossef Lampe,2011 DTTC, The GLobal Power Checker(GPC): A Noval Checker, Tracker and Coverage Package for Power Valdation on Simulation and Emulation

[3] Sasi Pavan Majety and Rammohan Koteshwar, 2019 DTTC, Enabling Hetero Vali- dation First TIme in Intel History: Journey of LKF Pre-Si SoC Hetero Validation

[4] Taylor Kidd, 2019,List of Useful Power and Power Management Arti- cles, Blogs and References, http://software.intel.com/en-us/articles/ list-of-useful-power-and-power-management-articles-blogsand-references

[5] https://www.Thomaskrenn.com/en/wiki/Processor_P-states_and_C-states

[6] https://software.intel.com/en-us/blogs/2008/05/29/ what-exactly-is-a-p-state-pt-1