University of Cincinnati

Date: 11/7/2013

I, Chandan Singh , hereby submit this original work as part of the requirements for the degree of Master of Science in Electrical Engineering.

It is entitled:

Fixing Power Bugs at RTL Stage using PSL Assertions

Student's name: **Chandan Singh**

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.

Committee member: Wen Ben Jone, Ph.D.

Committee member: Xuefu Zhou, Ph.D.

8002

Fixing Power Bugs at RTL Stage using PSL Assertions

A Thesis submitted to the

Graduate School at the

University of Cincinnati

In Partial Fulfillment of the

requirements for the Degree of

MASTER OF SCIENCE

in the Department of Electrical Engineering and Computing Systems

of the College of Engineering and Applied Sciences

By

Chandan Singh

Bachelor of Engineering (B. E.), 2009

Birla Institute of Technology & Science, Pilani, India

Committee Chair: Dr. Carla Purdy

ABSTRACT

Power dissipation has now become the most critical design constraint. Up till now, in the design flow of any SoC, power estimation and analysis came into the picture only after the completion of RTL synthesis. However, design optimization for low power is most suitable before synthesis. Each decrease in process geometry makes dynamic power targets harder to achieve. Also, changes made later in the design for power optimization lead to costly re-spin. It is better to pin-point power related problems in the design as early as possible when they can still be fixed. It also reduces risk by ensuring that the design meets power goals before embarking on its implementation. A novel approach is presented in this thesis which introduces power analysis at the RTL stage itself using PSL assertions. This will enable the SoC designer to optimize the design from a low power perspective at a very early stage (RTL) in the design flow where the scope of modification is maximized and the cost minimized.

ACKNOWLEDGMENTS

I wholeheartedly thank Dr. Carla Purdy, my thesis advisor for her timely feedback, suggestions & pro-activeness and will always cherish our association. I am also highly grateful to my manager, Rashna Seli at STMicroeletronics, under whose tutelage I learned the tricks of this "semiconductor" trade. The idea proposed in this thesis was conceptualized at STMicroelectronics and I thank ST for granting permission to pursue this idea as my thesis here at University of Cincinnati. I thank SECS for providing me with all the necessary EDA tools needed for the implementation of the proposed methodology.

I would like to thank Dr. Wen-Ben Jone and Dr. Frank Zhou for agreeing to be a part of the thesis committee.

Lastly, I take this opportunity to dedicate this thesis as well as my upcoming graduation to my mother and sister who have always been there for me.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

1.1 Motivation

The design of complex and high performance System-on-Chips has witnessed a series of ground breaking revolutions in the last three decades [1]. In the 1980's there was the introduction of language based design. The 1990's saw the adoption of design reuse and IP as a mainstream design practice. In the last few years, design for low power has started to change again how designers approach complex System-on-Chip designs. Until recently, designers were primarily concerned with improving the performance of their designs (throughput, latency, frequency), and reducing silicon area to lower manufacturing costs [2]. Now power is replacing performance as the key competitive metric for SoC design. As the technology has shrunk to 90 nm and below, the leakage current is increasing dramatically, to the point where, in some 65 nm technology designs, leakage current is as large as dynamic current. With the explosive growth of personal, wireless, and mobile communications, as well as home electronics, comes the demand for high-speed computation and complex functionality for competitive reasons. Today's portable products are expected not only to be small, cool, and lightweight, but also to provide extremely long battery life. And even wired communications systems must pay attention to heat, power density, and low-power requirements. Due to these ever growing challenges, designers need to think of all possible tricks to manage dynamic and leakage power and start as early as possible – at the Register Transfer Level (RTL) [3].

In any modern day chip, there are usually multiple modules which combine together to form sub-systems. These sub-systems are in turn integrated using some on chip networking and bus

architecture to form the top level System-on-Chip. These design blocks are complex in nature and hence this “divide and conquer” coupled with integration approach is the most preferred paradigm in today’s semiconductor world [4]. From the verification’s perspective at the RTL stage, the main motive is to ensure that the design blocks adhere to the predefined functional specifications. There is no systematic check existing at this stage which tells the designer integrating multiple blocks that while the sub-system or SoC is functionally correct it could be made more power efficient in terms of how blocks are interacting with each other.

In existing SoC design flow, after the functional verification of the RTL is complete, it is then synthesized using the technology specific target standard cell libraries using EDA tools like Design Compiler from Synopsys [3]. The synthesis process requires a set of timing, area and power related constraints. In other words, this means that the design has a pre-defined budget in terms of area, timing and power which should be respected at all costs. If any of these constraints are violated, the only possible solution (assuming the most optimal standard cells were picked up from the library) is to modify the RTL, redo the verification and then redo the synthesis. For complex chips with multiple blocks interacting with each other, this leads to a big cost. With numerous semiconductor companies fighting neck and neck for the valued market share, time to market is of paramount importance. Development of new design and verification methodologies which could potentially unearth timing and power related issues early in the design flow is gaining a lot of importance [4]. This presents a motivating challenge to VLSI researchers, R&D teams of semiconductor companies and EDA vendors all over the world to come up with innovative solutions which makes the RTL – Design & Verification not just functionally correct but ‘Aware

and Smart' in terms of anticipating power related issues which might occur later in the design flow thereby reducing the time-to-market cycle [5].

1.2 Thesis goal

The synthesis process converts the RTL of the design into the gate level net-list with cells picked up from the target library. The target library has timing and power attributes corresponding to each cell. This is reason why it makes more sense to do Power Estimation and Analysis after synthesis. However, design optimization for low power is most suitable before synthesis [6]. Eighty percent of Power Reduction opportunities are at RTL stage (Fig 1) [7].

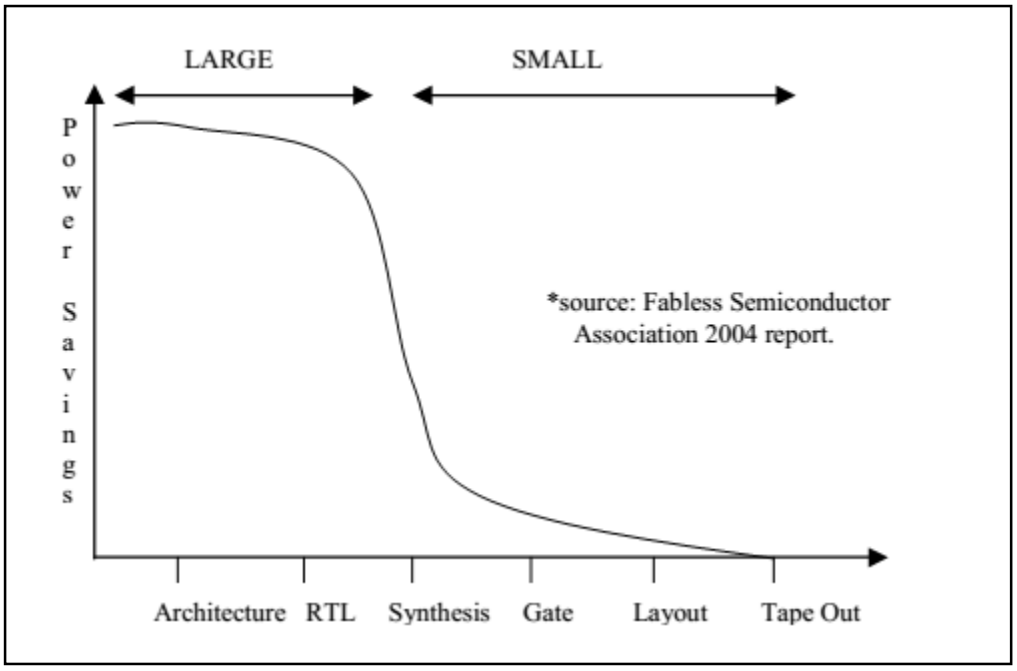

Fig. 1: Power saving opportunities at different stages in the design flow [7]

Each decrease in process geometry makes dynamic power targets harder to achieve. As discussed earlier, changes made later in the design for power optimization lead to costly re-spin. It is better to pin-point power related problems in the design as early as possible when they can

still be fixed [8]. It also reduces risk by ensuring that the design meets power goals before embarking on its implementation. This thesis presents a novel methodology to find out "power bugs" in the design at the RTL stage itself using the IEEE 1850 PSL (Property Specification Language) Assertions [9]. PSL Assertion Language has been chosen over others to implement this methodology due to reasons later explained in this document.

To have a clearer mental picture of this idea, imagine a hypothetical sub-system with two blocks – Block A and Block B in a master-slave configuration with the signal connections as made in the Fig. 2. Assume B1 is the chip select pin of the slave block. Block A sends the transactions to Block B through its ports A1, A2, A3 and A4.

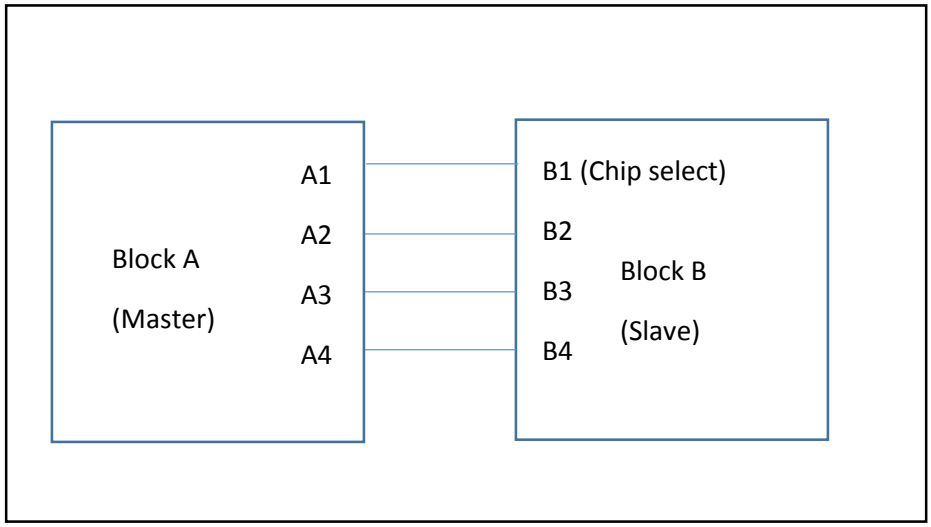

Fig. 2 Master-Slave configured sub-system

According to the design specifications of Block B, if the chip select (port B1) is disabled, then all the transactions sent by Block A will be ignored. The functional verification of this sub-system will ensure that block B drops the packets sent by Block A when the chip select is disabled. It should be noted that all the toggling of ports in such a scenario leads to wastage of dynamic power.

However, the functional verification process does not give any feedback to Block-A that it should stop toggling its own ports and consequently ports of Block-B when the chip select is disabled. Imagine a methodology which would work in conjunction with the functional verification process which gives this information to Block A, so that its design could be modified well in advance before synthesis to prevent such unnecessary toggling, thereby reducing dynamic power consumption. In this way, the functional verification process can catch "functional bugs" in the design while this methodology can catch "power bugs" in the design right at the RTL stage.

1.3 Document Organization

There are five chapters in this document. They are:

Chapter 1 is an introduction to the topic, setting up the platform for later chapters. The motivation behind this research, the goal of the thesis and the organization of this document is discussed.

1) Chapter 2 discusses proven dynamic power reduction techniques; explains the problem with these techniques and how the idea presented in this thesis is an attempt to solve that problem. It also talks about PSL Assertions, the language used for the implementation.
2) Chapter 3 talks about the implementation of the methodology presented.
3) Chapter 4 discusses the simulation results which substantiates the implementation.
4) Chapter 5 discusses the conclusion of this work and the area where this could be extended in future.

2. BACKGROUND

2.1 Mature dynamic power reduction techniques

A number of dynamic power reduction techniques have been developed in the past 10 years [1]. Some of the mature dynamic power reduction techniques are:

2.1.1 Clock gating

The distribution network of the clock on the chip constitutes a major fraction of the total dynamic power [10]. In most designs, up to 50% of the dynamic power can be consumed by the clock buffers. These buffers exhibit highest switching activity in the system, are large in numbers and often have to have high drive strength to minimize clock delay [1]. Due to these reasons they claim the lion's share of the total dynamic power. And the most common and intuitive way to reduce this power is to turn the clock off when it is not needed [11].

Currently all major EDA vendors support automatic clock gating. They infer the logic and insert the clock gating cell without changing the functionality of the system. Fig. 3 (next page) shows one such example.

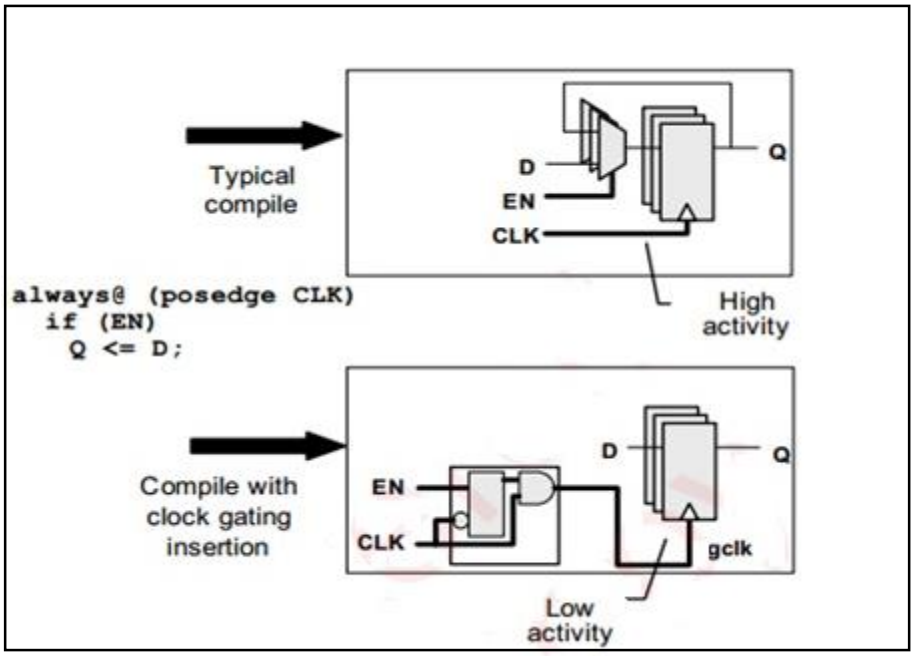

Fig. 3: Clock gating for a register inserted by the synthesis tool [12]

In a recent paper on clock gating techniques [12], Pokhrel discussed a nearly identical chip implemented both with and without clock gating. An existing 180nm chip without clock gating was re-implemented using clock gating in the same technology. Apart from this, minor changes in the logic were implemented (some blocks added, some removed with very little impact on the functionality).

According to Pokhrel, a reduction of around 34% to 43% of the total power was observed with clock gating. In his paper, he also concluded that clock-gating is meaningful only with registers with 3 or more bits. Using clock gating on 1-bit register was not all power and area efficient.

2.1.2 Gate level power optimization

Apart from clock-gating, synthesis tools can also perform other logic optimization techniques to minimize the switching activity of the design and hence reduce dynamic power consumption [1]. Fig. 4 shows one such optimization.

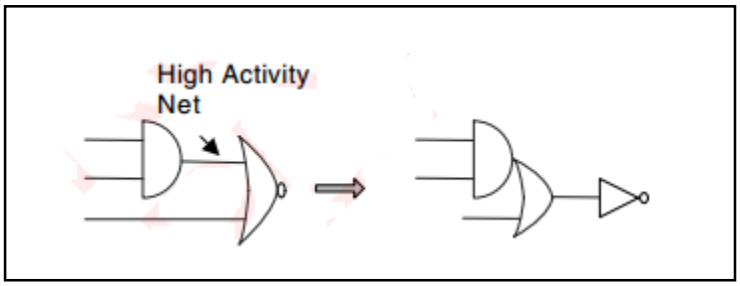

Fig. 4: Example of circuit remapping to save power

In this circuit, the output of the AND gate has a relatively high switching activity. It is possible to transform the above logic of AND-NOR into an AND-OR cell followed by the inverter so that the high switching net becomes an internal net of the AND-OR cell with much smaller load capacitance and hence low dynamic power [13].

Cell sizing and buffer insertions are some of the other common gate level power optimizations [1].

2.1.3 Multi-V_{DD}

There is a quadratic relation between voltage and dynamic power consumption and hence reduction in the voltage leads to considerable reduction in dynamic power. However, reducing the voltage increases the delay of the cells in the design.

One possible solution is to partition the design into different power domains. Consider the example of a chip shown in Fig. 5

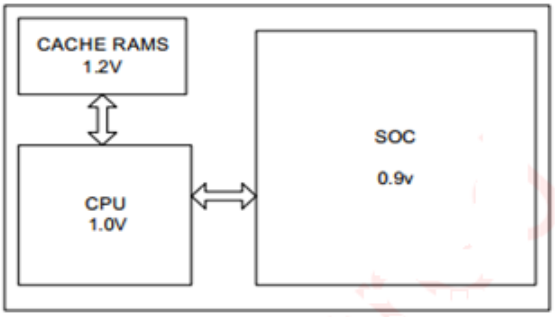

Fig. 5: Multi voltage design example

The caches are in the critical path of the design and hence they are working at the highest voltage supply. CPU is still a major performance intensive block so it still needs to work at a high voltage but lesser than the caches. Often, the rest of the SoC runs at a lower frequency as compared to the CPU and hence it can run at a still lower voltage. Each block in this system is working at a voltage so that it meets its minimum timing requirement. This technique provides significant dynamic power saving although it adds a lot of complexity to the design [1].

2.1.4 Dynamic voltage frequency scaling

Usually in any SoC, not all critical paths are exercised at once. Depending upon the system requirements, both the voltage and frequency of the design can be "scaled" down when the SoC is not doing performance intensive functions. This approach can lead to significant power savings [2]. However, like Multi-V_{DD} approach, this also adds significant complexity to the design.

2.2 Problems with existing techniques

2.2.1 Post-synthesis power analysis techniques

Apart from the power reduction approaches discussed above, numerous other approaches are being developed in an attempt to lower the power of the chip [3]. Observing these approaches closely we see a common trait in them. All of them can be employed only after or during the synthesis of the design. As discussed before, most of the power savings can be done at the RTL stage. These techniques, however intelligent they are, have a limitation that they can only optimize the design for power post the RTL stage.

2.2.2 Pre-synthesis power analysis techniques

The limitation of most of the existing post-synthesis power optimization techniques motivates researchers to think of pre-synthesis techniques. One such technique of finding out hotspots in the design based on the switching activity of different nodes using RTL simulation has been proposed by English, Man, Popovici and Schellekens [5]. This technique does a good job in finding zones in the circuit design which show high switching activity and hence higher dynamic power consumption. However, it does not relate the functionality of the design to its switching activity limiting the designer to make design changes based on the hotspots.

2.2.3 What's new?

The methodology presented in this thesis performs power analysis at the RTL stage by finding avoidable switching in the design based on its specifications. It aims at design optimization for power at the RTL stage itself, where there is maximum scope of power savings. It also overcomes the limitation of the above mentioned pre-synthesis technique [5].

2.3 PSL assertion

The idea in this thesis has been implemented using PSL Assertion. This section describes what PSL Assertion is, why PSL Assertion has been used for this implementation and finally explains one example of design behavior implemented using PSL Assertion.

2.3.1 What is PSL assertion?

Assertion is a language used for describing the behavior of the design under verification [14]. Many different variants of assertions are available and are extensively used by front-end design teams worldwide for functional verification purposes [15] [16]. PSL or Property Specification Language Assertion, an IEEE standard, is one of the most widely accepted assertion languages used in the industry. Simulators from all major EDA vendors such as VCS from Synopsys, QuestaSim from Mentor Graphics and Incisiv from Cadence support PSL Assertions.

The PSL Assertion language, due to its capabilities, is primarily used for functional verification. Using PSL Assertions, we can specify design behavior which should always hold true [9]. The assertion will monitor the traffic sent to the design as well the response of the design and will report a violation when the design property specified is not met. For example, consider the example of a master agent driving data to a slave agent. The handshaking protocol between the master and the slave specify that the slave should send an ‘ACK’ signal in three to five clock cycles after the master has sent a “READY” signal. The timing diagram in Fig. 6 explains this design requirement.

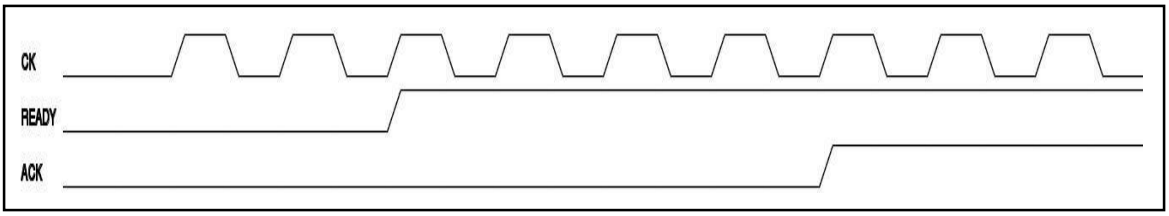

Fig. 6: Timing diagram of Master-Slave handshaking example

Using the constructs supported by PSL Assertion [9], the expected behavior of the signals in Fig. 6 can be tersely specified in a single line of code in the following manner:

// **psl assert** ((rose (READY)) -> ([*3:5]; **rose** (ACK))) @ **posedge** (CK);

This assertion will act as constant monitor to the system and whenever the above mentioned design behavior is not met, say the ACK signal is asserted 6 clock cycles after READY signal, then it will report a violation, thereby acting as a checker for this design property.

2.3.2 Why PSL assertions?

One can argue about the need to learn a new language for writing assertions while the same design behavior can also be expressed using Verilog or VHDL in which the RTL is written. But there are many benefits of using PSL Assertions for property checking over a design language like Verilog [9]. Some of them are:

1) **Terseness of the code**. Using assertions complex design behaviors spanning over multiple clock cycles involving multiple signals can easily be expressed. It would require many lines of code in Verilog to implement the same behavior. Hence, using assertions instead of Verilog will make the code less bug prone.

2) **Readability**. The constructs of the PSL Assertions make the code very easy to understand.

3) **Controllability**. PSL Assertions have great controllability in terms of their usage. They can be enabled or disabled directly from the command line while launching the compilation/simulation of the code. The severity of the action taken by the assertion in case of a failure can also be controlled. It can be configured as a "WARNING", "ERROR" or "FATAL", each having a different purpose, thus giving a lot a flexibility to the user.

4) **Zero simulation overhead for runs without Assertions**. The PSL Assertions are written as a "comment" (might sound confusing!) followed by the keyword psl. Revisiting the master-slave assertion described in section 2.3.1 we see that the assertion begins with "//" followed by the keyword "psl".

   // psl assert ((rose (READY)) -> ([*3:5]; rose (ACK))) @ posedge (CK);

   Since the line where the assertion is placed begins with a "//", by default it will be treated as a comment and will be ignored by the simulation tool. To enable the assertions we need to pass the switch "-assert" while compiling the design (different simulators have different switches for enabling the assertions, but all of them support PSL Assertions)

   This "smart" feature of PSL Assertion leads to zero performance overhead in simulations where the user wants the assertions to be disabled.

5) **External PSL Assertions**. This is probably the most powerful feature of PSL Assertions in terms of their usage. Many design and sub-blocks have their "legacy" or "golden" RTL which is expected not to be touched. Moreover, many design blocks are bought from external vendors

and the RTL of these blocks cannot be modified. We can still write assertions for these modules without even touching them.

PSL Assertions written external to the file are grouped into a "vunit" (Called Verification Unit). The "vunit" can be attached to the module externally and assertions act just like inline Assertions. The syntax is as follows:

```
vunit name [(<HDL_design_unit>)]
 {
   default clock = <clock_decl>;
   <assertions>;
   ...
 }
name
```

This feature has been used in the implementation of the methodology discussed in this thesis.

2.3.3 Design property and its PSL assertion: an example

As mentioned before, PSL is an IEEE standard. Its Language Reference Manual (LRM) describes all the constructs and types of assertions in detail [9]. One specific type of construct used for the implementation of the proposed idea is discussed below.

Design property: When the memory is in Power Down mode, then the input data port should not toggle.

Assertion:

//**psl** power_down_data_stable: **assert always** (PD) -> **stable** (D);

"power_down_data_stable" is the name of this assertion; "assert" and "always" are keywords which signify that this design behavior is true always when PD is set to logic high; "stable" is another keyword which checks for stability of the value at port D.

There are two parts of this Assertion – the part to the left of the implication operator "->" is called the enabling condition while the part to the right of the implication operator "->" is called the fulfilling condition. The enabling condition for this assertion is PD port being at logic high. The fulfilling condition is that D should not toggle and should remain stable. The assertion will remain idle when the enabling condition is false. Once the enabling condition becomes true, i.e., PD is set to logic high, the assertion starts checking the fulfilling condition and reports a violation when it is not met. So, this assertion will not do anything when PD is low and will check for stability at

the D port when PD is high which is exactly what the design behavior described for this assertion demands.

3 METHODOLOGY

The methodology presented in this thesis has been implemented on a memory sub-system using PSL Assertions. This subsystem consists of a Static Random Access Memory which is controlled by a generic memory controller. This is shown in Fig. 7. The reasons for choosing memory for the implementation of this idea are:

1) Around 50% to 70% of the chip area and power in modern days SoCs is attributed to memories [17]. Drawing a parallel from the well-established design strategy of "making the common case fast", we attempt to make the common case optimized for power [18].
2) The design specifications of the SRAMs are easy to understand, thereby keeping the main idea of this thesis its central theme rather than the design itself.

Having said that, this idea can be applied to any module in the design and is not limited just to SRAMs.

There are four main sections in this chapter. In the first section, the design specification of the SRAM has been explained. In the second section, based on the understanding of the memory design specifications, various functional scenarios which could be optimized for power have been identified. The third section explains the actual implementation of this technique on the scenarios identified using PSL Assertion. In the fourth section, the execution flow of this methodology has been explained.

3.1 Design Specifications of the Memory

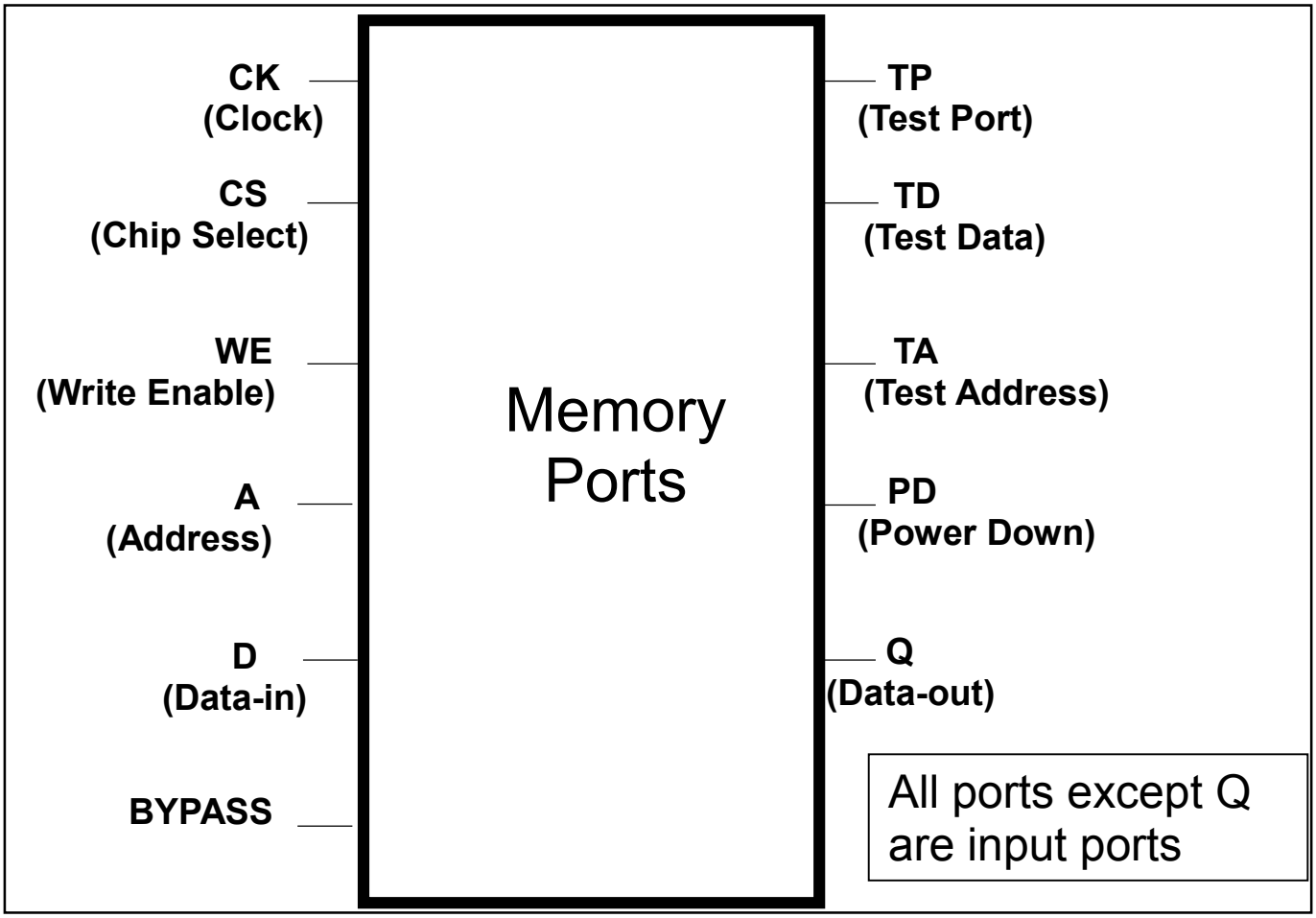

Fig. 7: Memory pin-out diagram

Table 1 lists the ports of the memory along with a short description of the each port.

Name of the Port	Description
CK	Clock of the memory. All inputs and outputs are synchronized with this clock.
CS	Chip select of the memory. It should be set to logic high when the memory is accessed. When it is at logic low, the memory is in No Operation or the NOP mode.
WE	Write Enable of the memory. When WE is at logic high, the memory is in write mode and when it is at logic low, the memory is in read mode.
A	Address of the memory location to be accessed for read or write
D	Data sent to the memory as input for comes in through this port.
Q	Data sent from the memory as output goes out through this port
BYPASS	When this port is set to logic high, D goes directly to Q in an asynchronous manner thereby bypassing the memory.
TP	It is the TEST PORT of the memory, when this port is set to logic high then the memory is in Test Mode.
TA	It is the Test Address port of the memory
TD	It is the Test Data port of the memory
PD	It is the Power Down port of the memory. When this port is in logic high state, then the memory is in “Sleep” mode. To perform read or write operations of the memory, this port should be set to logic low.

Table 1: SRAM ports with short descriptions

The memory with ports enlisted in Table 1 can work in multiple modes depending upon the states of its different ports.

3.1.1 Modes of operation of the memory

a) No Operation mode or the NOP mode: When the chip select port of the memory is set to logic low, then the memory is in "No-operation" or NOP mode. Read or Write accesses made to the memory in this mode is disabled. The memory samples the value at the "CS" port at the rising edge of the clock and if it is sampled as a "0", it knows that the NOP mode has been activated. It ignores all the requests until a rising edge of the clock comes where it is sampled as a "1". The timing diagram of the NOP mode has been shown in Fig. 8.

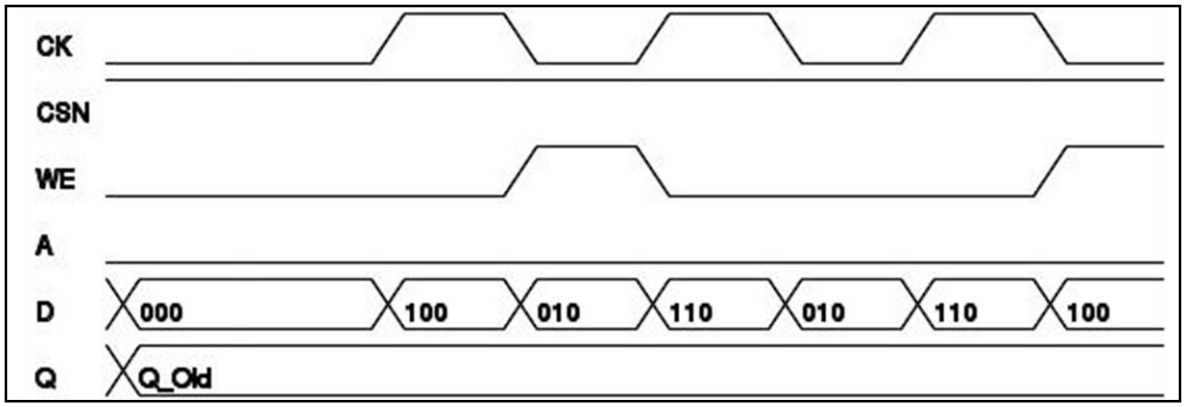

Fig. 8: Timing diagram – NOP mode

b) BYPASS mode: When the BYPASS port of the memory is set to logic high, then data sent in through the D port is sent directly and asynchronously to the Q port. No read/write access can be made to the memory in this mode. This mode is needed to bypass the memory in order to quickly test the logic which is connected to the output port of the memory. By supporting this feature in

the memory, the intended test data for fault testing of the “combo cloud” connected to the Q, can reach in considerably less time as compared to at least 2 clock cycles for writing and then reading the data through the memory. Fig. 9 is a pictorial representation of this mode.

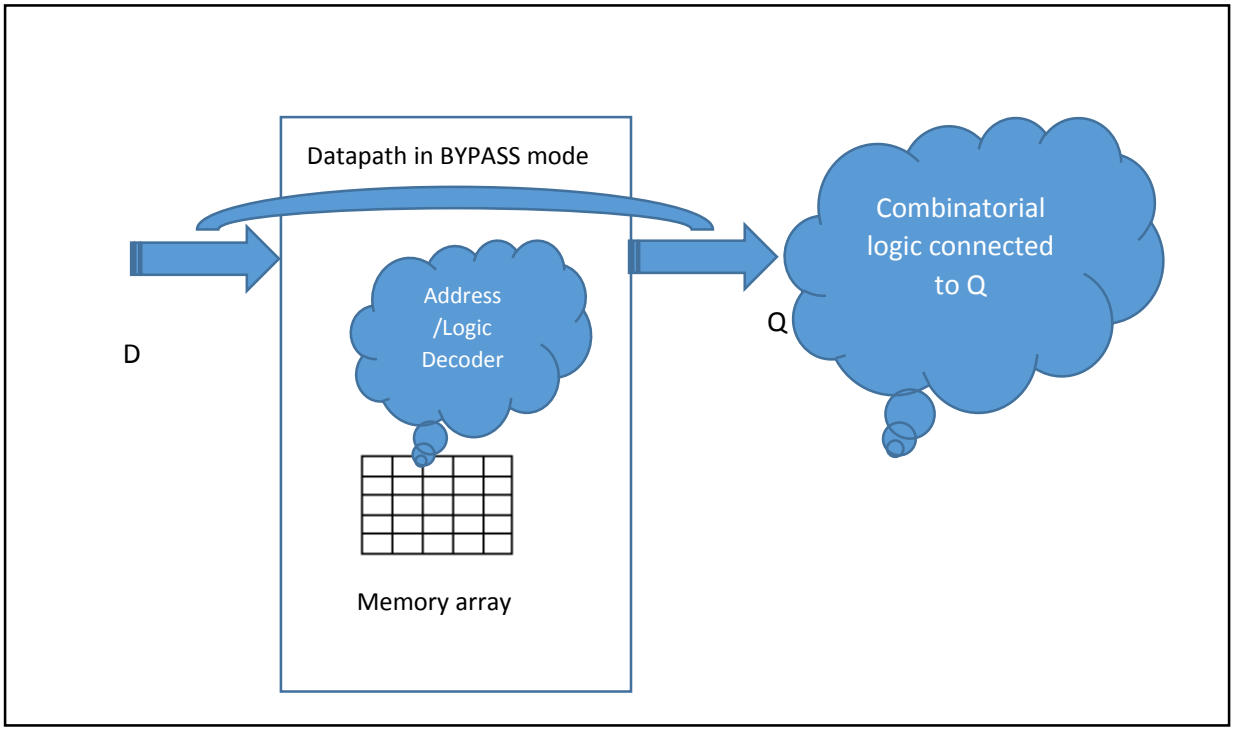

Fig. 9: Pictorial representation of Bypass mode

The timing diagram of the Bypass mode has been shown in Fig. 10

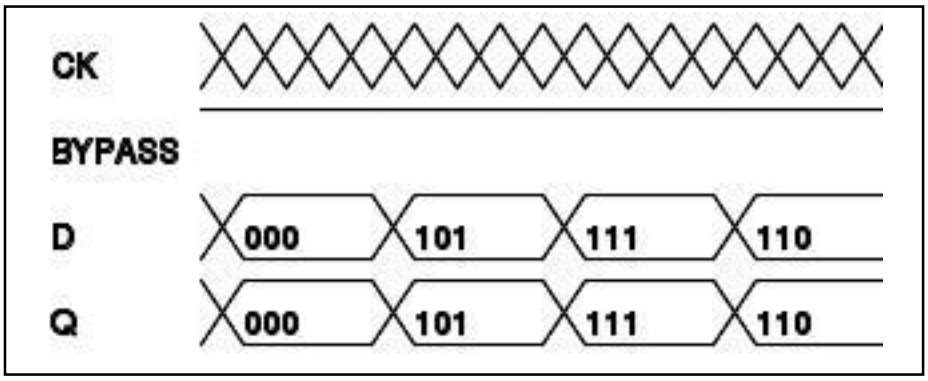

Fig. 10: Timing diagram: Bypass mode

c) WRITE Mode: When memory is not in NOP mode or BYPASS mode with WE port set to logic high, then the memory is in Write Mode. In this mode, the data is written to the memory array which is indexed by the address. There are two types of writes supported in the memory:-

i. Normal write mode: When the TP or the Test Port is set to logic low, then Write is done using the data on the D port and address on the A port. The timing diagram of the Normal Write mode is shown in Fig. 11

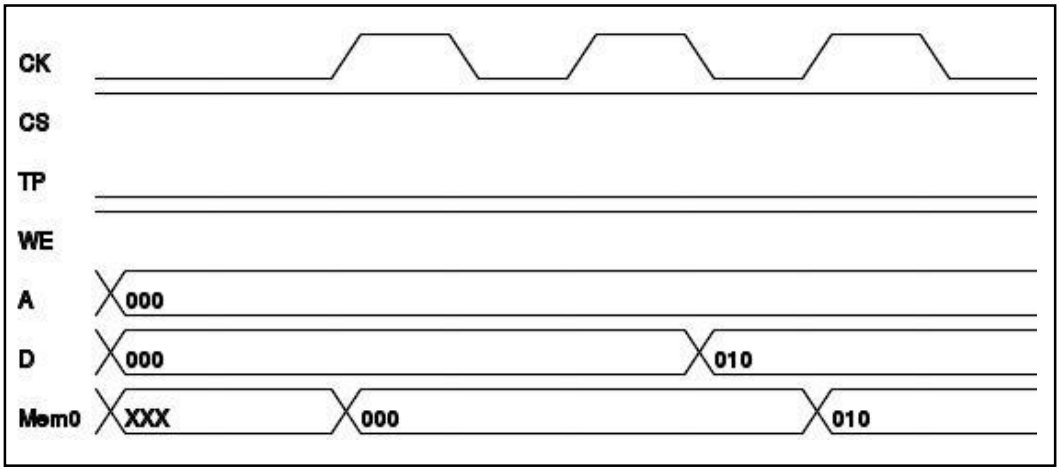

Fig. 11: Timing diagram – Normal write mode

ii. Test write mode: When the Test Port is set to logic high, then Write is done using data on the TD port and address on the TA port. The test ports are used for fault testing of the memory after manufacture using BIST or the Built in Self-Test. The timing diagram of the Test Write mode is shown in Fig. 12.

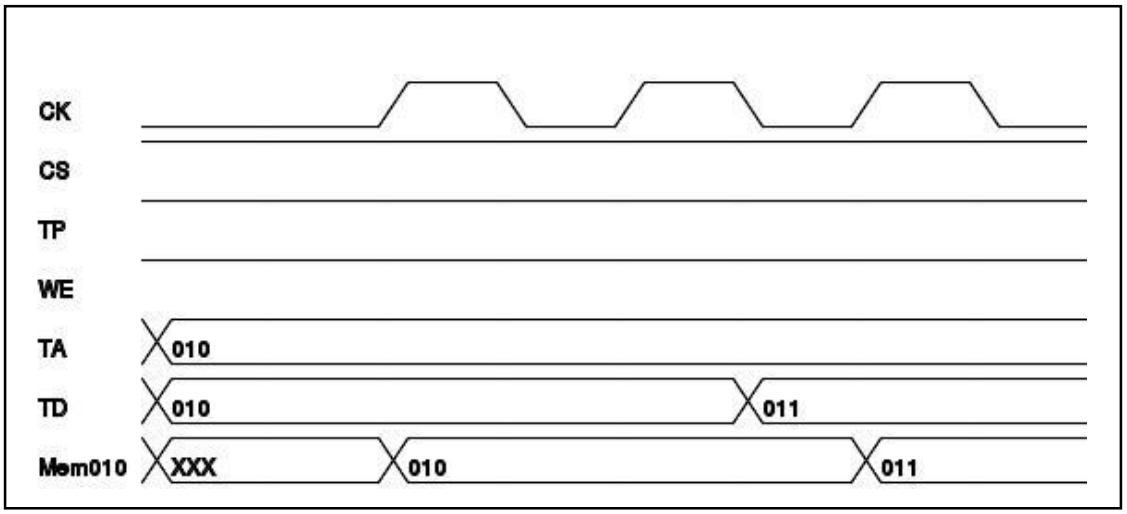

Fig. 12: Timing diagram – Test write mode

d) Read mode: When memory is not in NOP mode or Bypass mode with WE port set to logic low, then the memory is in Read Mode. In this mode, data from the memory at the given address location is sent the output. Just like the Write mode, there are two types of reads supported in the memory:-

i. Normal read mode: When the TP or the Test Port is set to logic low, then read is done using address on the A port. The timing diagram of the normal read mode is shown in Fig. 13.

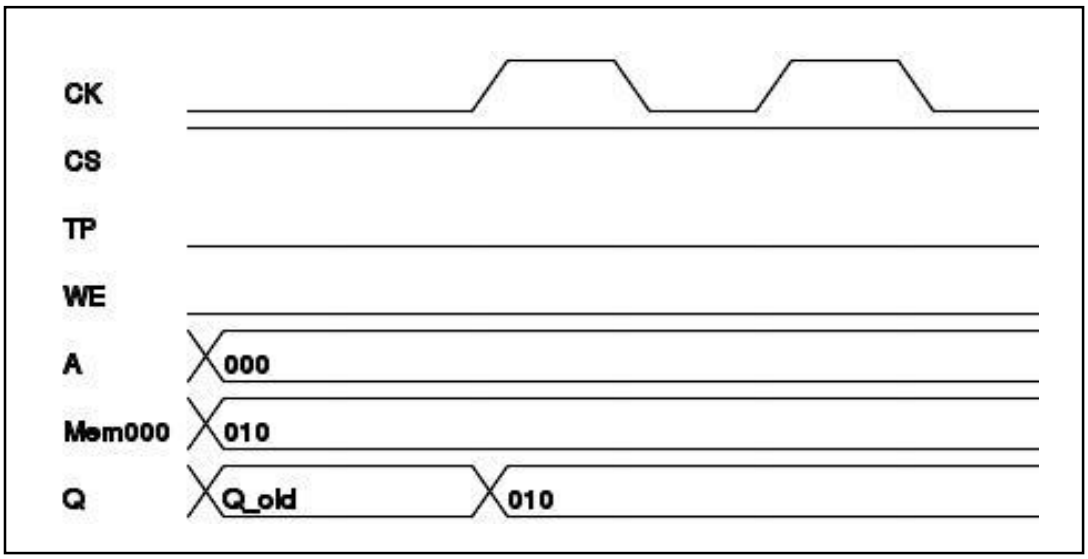

Fig. 13: Timing diagram – Normal read mode

ii. Test read mode: When the TP or the Test Port is set to logic high, then Read is done using address on the TA port. The timing diagram of the Test Read mode is shown in Fig. 14

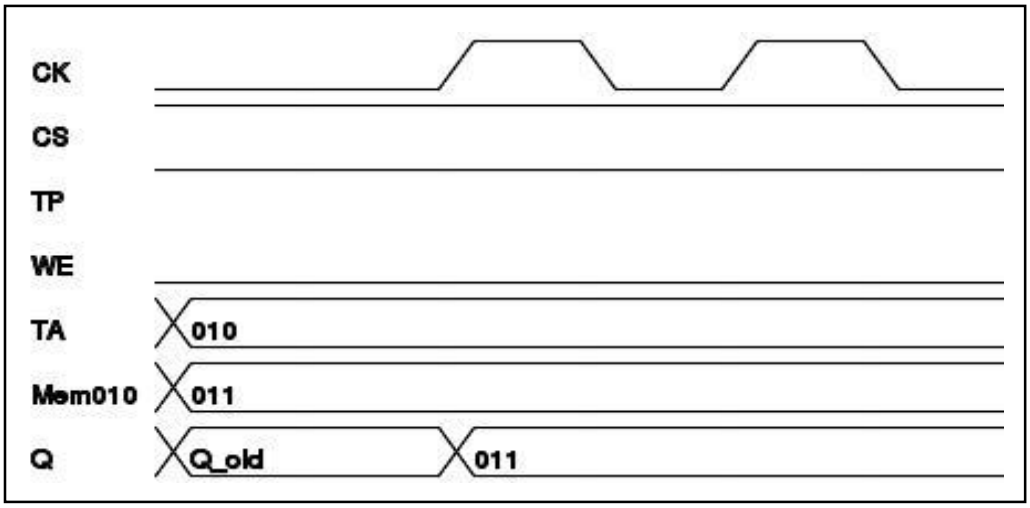

Fig 14: Timing diagram – Test r ead mode

e) POWER DOWN Mode: When the Power Down or PD port is sampled as a "1" at the rising edge of the clock, then the memory enters POWER DOWN mode. Functionally this mode is similar to the NOP mode as no accesses to memory can be made in PD mode as

well. This mode is used to safely power up/down the memory. Whenever we need to power up or down the memory, PD port should be set to logic high for the entire duration. The timing diagram of PD mode is shown in Fig. 15 ("vddm" is the memory power supply port)

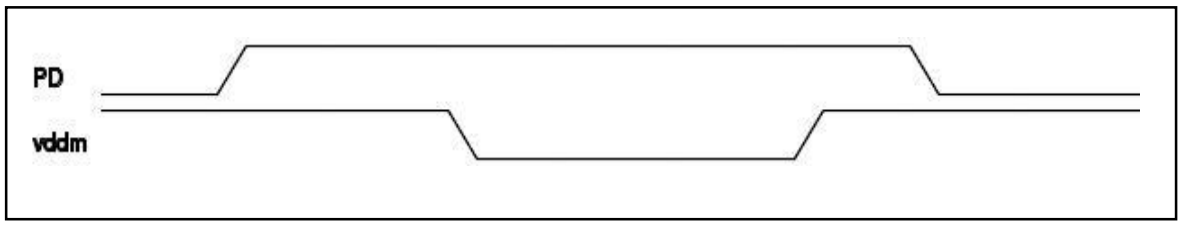

Fig. 15: Timing diagram – Power Down mode

3.1.2 Truth table of modes of operation

The truth table of the different modes of operations supported by the memory is shown in Table 2.

Mode of Operation	CK	CS	WE	BYPASS	TP	PD	Action on memory array	Action on output
NOP	0->1	1	X	0	X	0	No Action	No Action
BYPASS	X	X	X	1	X	0	No Action	Q = D
Normal Write	0->1	0	1	0	0	0	Memory[A] = D	No Action
Test Write	0->1	0	1	0	1	0	Memory[TA]= D	No Action
Normal Read	0->1	0	0	0	0	0	No Action	Q=Memory[A]
Test Read	0->1	0	0	0	1	0	No Action	Q=Memory[TA]
Power Down	X	X	X	X	X	1	Corrupt the memory contents	Corrupt the Output data

Table 2: SRAM modes of operation and its truth table

3.2 Power optimal memory design scenarios

A thorough understanding of the memory design specification enables us to come up with power optimization for its different modes. With clear knowledge of the design, one can think of many ways in which, depending upon the logic level of one port, toggling of some other port or ports is meaningless, thereby leading to unnecessary dynamic power consumption. A list of such possible scenarios is given below:

1) Power down mode: When "PD" is high and memory is in Power Down mode, all accesses to the memory are disabled. Hence, there is no point in toggling all the other ports when the memory is in this mode.
2) NOP mode: When the chip select (CS) port is low and memory in is NOP mode, then there is no point in toggling the WE, A, TA, TP, TD ports. It should be noted that memory can co-exist in Bypass mode with the NOP mode and hence toggling of D port makes sense in NOP mode with BYPASS mode enabled. However, if Bypass mode is also disabled, then toggling of D port also becomes useless and can lead to dynamic power wastage.
3) Read mode: When the memory is in Read mode, then toggling the data port, D is meaningless.
4) Bypass mode: When memory is in Bypass mode, then there is no use of toggling the A, CS, WE, TP, TA, TD ports.
5) Functional mode: When the memory is in functional mode, then there is no point in toggling the test ports. This means that when TP=0, TA and TD should not be toggled.
6) Test mode: When the memory is in test mode (TP=1), then there is no point in toggling the functional input ports of D and A.

The identification of such scenarios which could be optimized for power is a critical component of the methodology presented in this thesis. This above list can grow further as more of such "power saving" scenarios are identified.

3.3 PSL Assertions for the identified scenarios

The above mentioned scenarios can be implemented using many languages, Verilog being the most intuitive choice as the design is coded in Verilog. As mentioned before, the PSL Assertion language is extensively used in the industry mainly for functional verification purposes. However, we have used this language for the implementation of the proposed methodology thereby using it for "Power Verification" as well. The benefits of using PSL Assertion for this implementation have already been discussed in the previous chapter.

Here we list PSL assertions corresponding to each scenario explained in the previous section:

1) Scenario 1: Power down mode

//psl a_1: assert always (PD) -> stable(CK);

a_1 is the name of this assertion. This assertion will report a violation if the clock toggles in the Power Down mode.

// psl a_2: assert always (PD) -> stable(CS);

a_2 is the name of this assertion. This assertion will report a violation if the chip select toggles in the Power Down mode.

//psl a_3 : assert always (PD) -> stable (A);

a_3 is the name of this assertion. This assertion will report a violation if the address toggles in the Power down mode.

//psl a_4 : assert always (PD) -> stable (D);

a_4 is the name of this assertion. This assertion will report a violation if the data toggles in the Power Down mode.

//psl a_5 : assert always (PD) -> stable (WE);

a_5 is the name of this assertion. This assertion will report a violation if the write enable toggles in the Power down mode.

//psl a_6 : assert always (PD) -> stable (BYPASS);

a_6 is the name of this assertion. This assertion will report a violation if the BYPASS port toggles in the Power down mode.

//psl a_7 : assert always (PD) -> stable (TP);

a_7 is the name of this assertion. This assertion will report a violation if the Test Port toggles in the Power down mode.

//psl a_8 : assert always (PD) -> stable (TA);

a_8 is the name of this assertion. This assertion will report a violation if the Test Address port toggles in the Power down mode.

//psl a_9 : assert always (PD) -> stable (TD);

a_9 is the name of this assertion. This assertion will report a violation if the Test Data port toggles in the Power down mode.

2) Scenario 2: NOP mode

//psl a_10: assert always (!CS) -> stable(CK);

a_10 is the name of this assertion. This assertion will report a violation if the clock port toggles in the NOP mode

//psl a_11: assert always (!CS) -> stable(WE);

a_11 is the name of this assertion. This assertion will report a violation if the write enable port toggles in the NOP mode

//psl a_12: assert always (!CS) -> stable(A);

a_12 is the name of this assertion. This assertion will report a violation if the Address port toggles in the NOP mode

//psl a_13: assert always (!CS) -> stable(TP);

a_13 is the name of this assertion. This assertion will report a violation if the Test Port toggles in the NOP mode

//psl a_14: assert always (!CS) -> stable(TA);

a_14 is the name of this assertion. This assertion will report a violation if the Test Address port toggles in the NOP mode

//psl a_15: assert always (!CS) -> stable(TD);

a_15 is the name of this assertion. This assertion will report a violation if the Test Data port toggles in the NOP mode

//psl a_16: assert always (!CS & !BYPASS) -> stable(D);

a_16 is the name of this assertion. This assertion will report a violation if the Data port toggles in the NOP mode with BYPASS also disabled.

3) Scenario 3: Read mode

//psl a_17: assert always (CS && !WE && !BYPASS) -> stable(D);

a_17 is the name of this assertion. This assertion will report a violation if the Data port toggles in the Read mode.

4) Scenario 4: Bypass mode

//psl a_18: assert always (BYPASS) -> stable (WE);

a_18 is the name of this assertion. This assertion will report a violation if the write enable port toggles in the Bypass mode

//psl a_19: assert always (BYPASS) -> stable (CS);

a_19 is the name of this assertion. This assertion will report a violation if the chip select port toggles in the Bypass mode

//psl a_20: assert always (BYPASS) -> stable (A);

a_20 is the name of this assertion. This assertion will report a violation if the Address port toggles in the Bypass mode

//psl a_21: assert always (BYPASS) -> stable (TA);

a_21 is the name of this assertion. This assertion will report a violation if the Test Address port toggles in the Bypass mode

5) Scenario 5: Functional mode

//psl a_22: assert always (!TP) -> stable(TA);

a_22 is the name of this assertion. This assertion will report a violation if the Test Address port toggles in the Functional mode

//psl a_23: assert always (!TP) -> stable(TD);

a_23 is the name of this assertion. This assertion will report a violation if the Test Data port toggles in the Functional mode

6) Scenario 6: Test mode

//psl a_24: assert always (TP) -> stable(A);

a_24 is the name of this assertion. This assertion will report a violation if the Address port toggles in the Test mode

//psl a_25: assert always (TP) -> stable(D);

a_25 is the name of this assertion. This assertion will report a violation if the Data port toggles in the Test mode

So, in total 25 PSL Assertions have been written to cover all the scenarios which could be optimized for power.

3.4 Execution flow of proposed methodology

This section describes all the steps needed to implement the proposed idea. Fig. 16 shows the flowchart of the methodology.

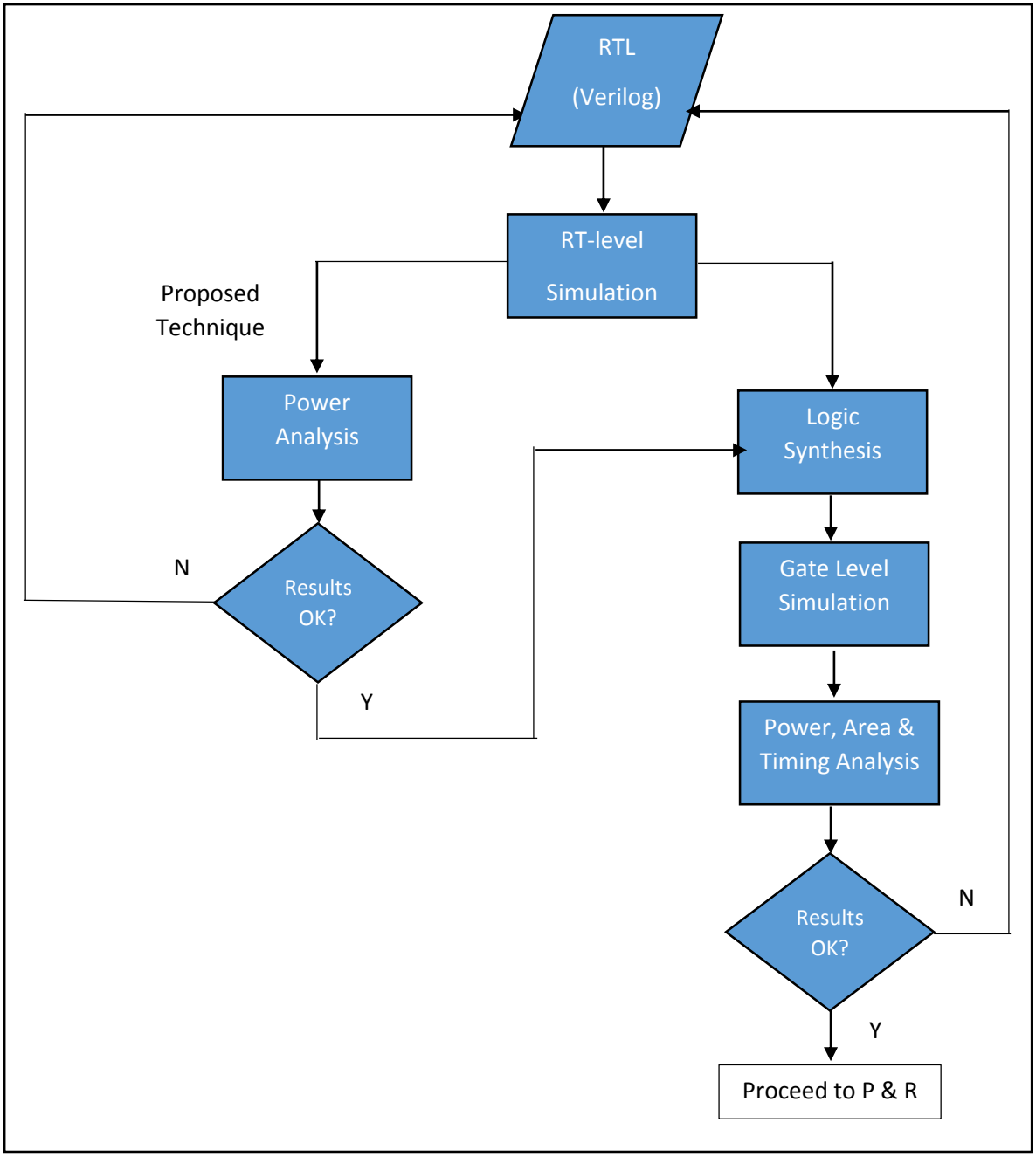

Fig. 16: Flowchart – proposed methodology

STEP-1) Code the Assertions

Once the assertions for the different power saving scenarios have been developed, they need to be attached to the memory module. There are two ways to do this:

1) Inline Assertions: In this method, the assertions are placed as it is inside the memory module definition. As discussed before, under normal simulations these assertions will be ignored by the simulator as they are written in "commented" format.

   In order to enable these assertions during simulation, "-assert" switch will be passed to the simulator when launching the run.

   Command: (Using VCS)

   Vcs memory.v –psl –l sim.log –gui –timescale=1ns/1ns

   ./simv

   The benefit of this method is the ease of its execution. The assertions are written inside the memory module definition and hence can be enabled or disabled very easily from the command line itself.

   However, we need to touch the memory module while using the inline assertions. This might not be preferred if the memory module definition is a "golden" or "legacy" code or bought from a third party vendor. In that case, the second method of "out of body" assertions should be used.

2) Out-of-body Assertions: In this method, a separate file called the "Vunit" or the Verification unit is created and all the assertions are written inside the "vunit" [9]. Using this method enables us to write the power optimal assertions without even touching the memory module definition.

The vunit file for this implementation is shown on the next page; "power_optimal_memory" is the name of this vunit and "memory_module_instance" is the instance name of the memory module with which this vunit is connected.

Command: (Using VCS)

Vcs memory.v –assert *vunit_file_name* –l sim.log –gui –timescale=1ns/1ns

./simv

```
vunit power_optimal_memory(memory_module_instance) {
a_1: assert always (PD) -> stable(CK);
a_2: assert always (PD) -> stable (CS);
a_3: assert always (PD) -> stable (A);
a_4: assert always (PD) -> stable (D);
a_5: assert always (PD) -> stable (WE);
a_6: assert always (PD) -> stable (BYPASS);
a_7: assert always (PD) -> stable (TP);
a_8: assert always (PD) -> stable (TA);
a_9: assert always (PD) -> stable (TD);
a_10: assert always (!CS) -> stable (CK);
a_11: assert always (!CS) -> stable(WE);
a_12: assert always (!CS) -> stable(A);
a_13: assert always (!CS) -> stable(TP);
a_14: assert always (!CS) -> stable(TA);
a_15: assert always (!CS) -> stable(TD);
a_16: assert always (!CS & !BYPASS) -> stable(D);
a_17: assert always (CS && !WE && !BYPASS) -> stable(D);
a_18: assert always (BYPASS) -> stable (WE);
a_19: assert always (BYPASS) -> stable (CS);
a_20: assert always (BYPASS) -> stable (A);
a_21: assert always (BYPASS) -> stable (TA);
a_22: assert always (!TP) -> stable(TA);
a_23: assert always (!TP) -> stable(TD);
a_24: assert always (TP) -> stable(A);
a_25: assert always (TP) -> stable(D);
}
```

It should be noted that the keyword "//psl" is removed in this implementation method.

STEP-2) Run the simulation

The above vunit file is attached to the memory module. The memory module is instantiated inside the memory controller module. Six different test-benches were created targeted at the 6 design scenarios discussed. The test-benches were made exhaustive enough to fully cover all the modes of the operations. The assertions failures reported during the simulations are captured in the log files of each individual run. Assertions failures are using reported like this:

```
"memory.v", 39: pd.I0.a_1: started at 130ns failed at 130ns
        Offending 'stable(CK)
```

STEP-3) Debug the Assertion failures

Re-launch the simulation in which assertion failures have been reported in GUI mode. Find out the signal from the memory controller which is causing that assertion to fail. Analyze the logic which controls that port of the memory controller and check for the possibility of redesigning that part of logic so that the assertion would not fail.

In some cases, changing the RTL in order to make the assertion pass would be too difficult. It might lead to a loop of assertion related fixes where changing the design for one assertion failure might fail an earlier passing assertion. In such cases, we will have no choice but to ignore the power optimization reported by that particular assertion. This means that there should be a clean mechanism in place by which individual assertions could be ignored. One such mechanism by which individual assertions can be disabled from the command line itself while launching the simulation has been explained below.

Mechanism to disable Assertions individually: Consider the assertion "a_1"

a_1: assert always (PD) -> stable(CK);

A "parameter" named a_1_en (Enable for assertion a_1) is defined and set to 1 by default. The assertion is now slightly modified to:

a_1: assert always (PD **&& a_1_en**) -> stable(CK);

By adding this parameter (with the default value of 1) in the enabling condition for this assertion, we ensure that the assertion remains enabled by default; when it has to be disabled due to reasons discussed before, the parameter a_1_en is redefined to 0 externally from the command line using "defparam" while passing arguments for the simulation.

defparam *Instance_name_of_memory*.a_a_en = 1'b0;

Similar parameters have been defined for the other assertions along with adding it in their respective enabling conditions.

STEP-4) Fix the design logic and finalize

Check all the simulations thoroughly and make sure all assertions either pass or are disabled because the logic could not be altered to accommodate the suggested change (repeat STEP-3 until no failures are reported).

After running the test regressions for functional checks too, the “power optimal” RTL is now good and ready for synthesis.

4 RESULTS

There are three sections in this chapter. The first section discusses the debugging of the assertion by taking one example. The second section shows the simulation snapshots of all the six test-benches with explanation. The third section shows the results in terms of switching activity with respect to ports in the design which can potentially be avoided.

4.1 How to debug and fix an Assertion failure: an example

The following assertion failure has been reported in one of the simulation runs.

"memory.v", 43: pd.I0.a_5: started at 60ns failed at 60ns

Offending 'stable(WE)'

We re-run the simulation in the GUI mode. As shown in Fig. 17 below, we clearly see the assertion "a_5" reporting failure at times 520ns, 540ns and 560ns.

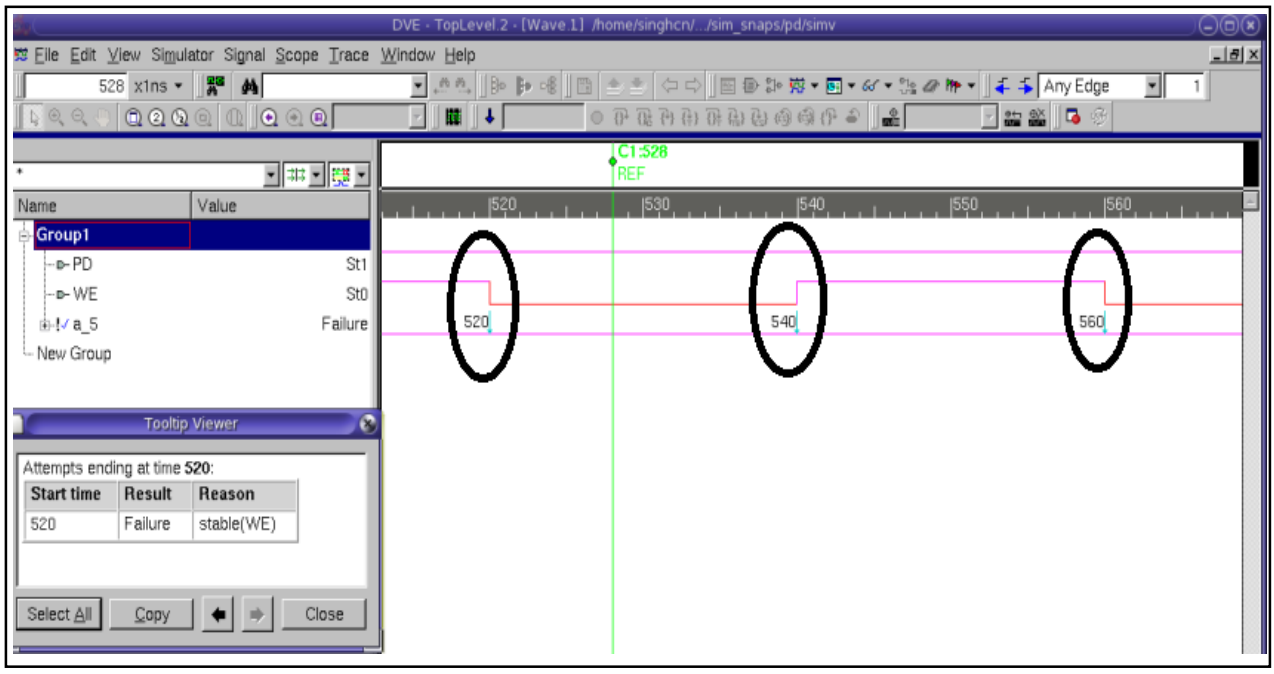

Fig. 17: Simulation snapshot showing failures of the assertion "a_5"

Details of one such failure at 520ns is shown in the simulator's window. It also shows the reason for failure of the assertions as "stable(WE)". This was the fulfilling condition of the assertion "a_5" which has not been met while it's enabling condition (PD = 1) was evaluated as true.

We can infer from this assertion failure that the memory controller which controls the ports of the memory is toggling the "WE" port of the memory in Power Down mode. This is a scenario which can potentially be fixed thereby saving dynamic power.

The part of the logic in the memory controller which drives the "WE" port is now analyzed to find out a possible fix of the solution. One possible solution is to "AND" the inverted "PD" signal to the logic generating the signal for driving the "WE" port. This will ensure that whenever PD is high, i.e., memory is in Power Down mode, "WE" port will always stay at logic "low" level.

This is one of the power optimizations that has been performed on the design right at the RTL stage using this methodology. After making this fix, we re-run the simulation and find out that no more "a_5" failures are being reported. The pictorial representation of the problem and its fix has been shown in Fig. 18 and 19.

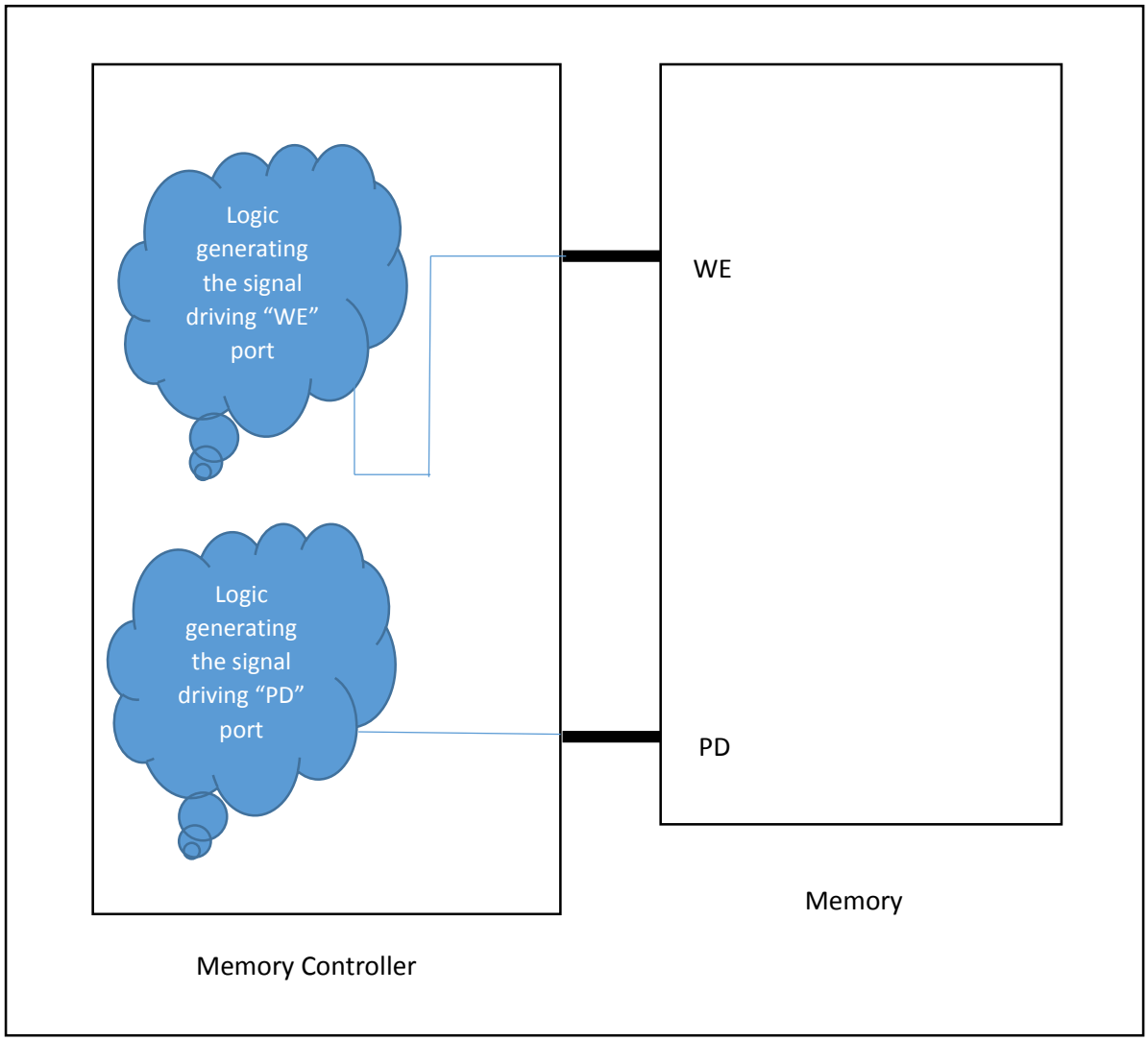

Fig. 18: Memory sub-system block diagram **before the fix**

(No relation between the logic generating the WE signal and the PD signal).

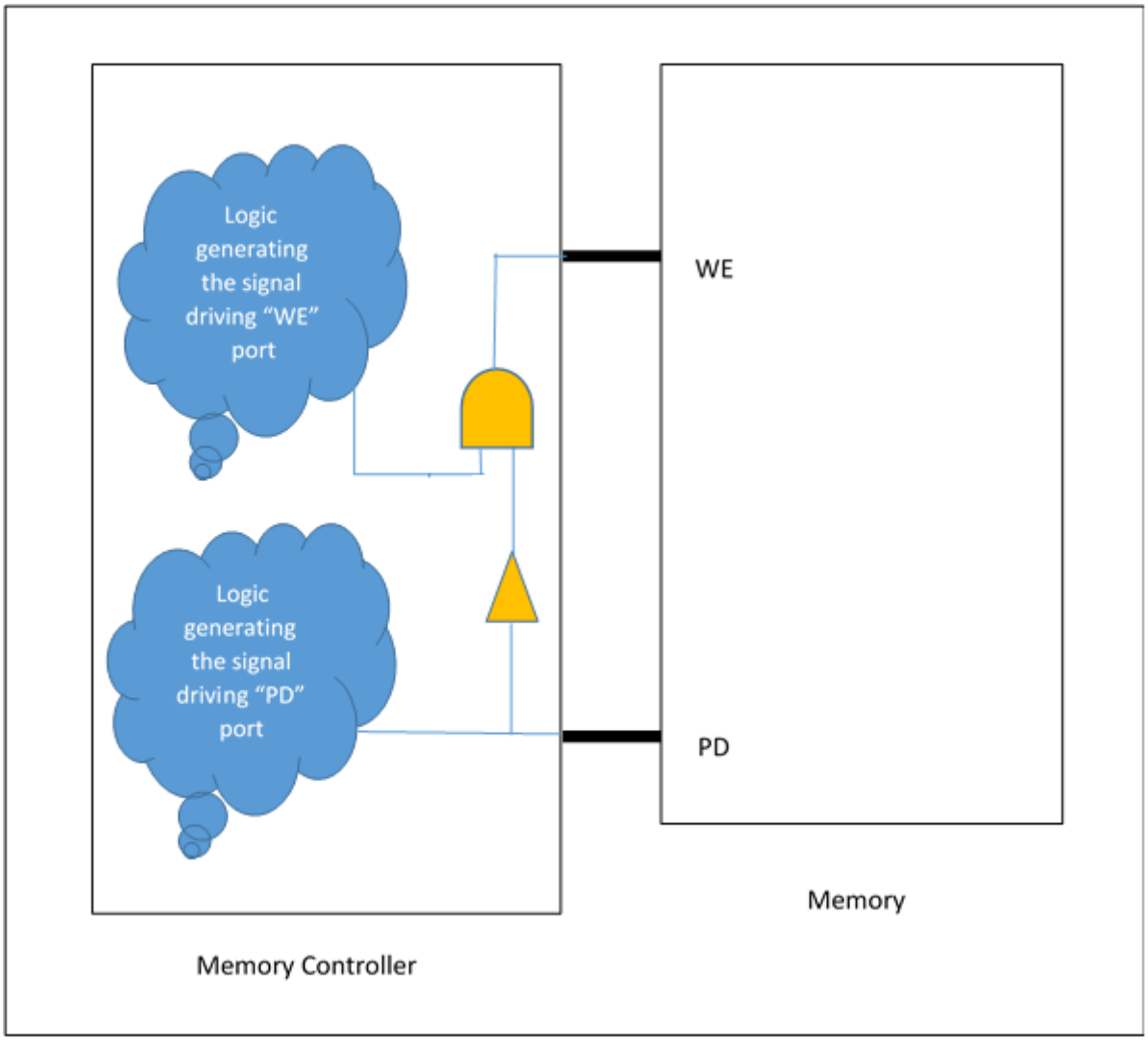

Fig. 19: Memory sub-system block diagram **after the fix**

(Logic generating "WE" signal is now "ANDed" with inverted PD signal).

4.2 Simulation snapshots

Assertions failures reported by the six test-benches created for the six identified design scenarios have been shown in this section.

4.2.1 Power down mode

a) Failures of the assertion “a_1” reported as shown in Fig. 20 at times 170ns, 180ns, 190ns and 200ns. “CK” toggling when “PD” is at logic high state.

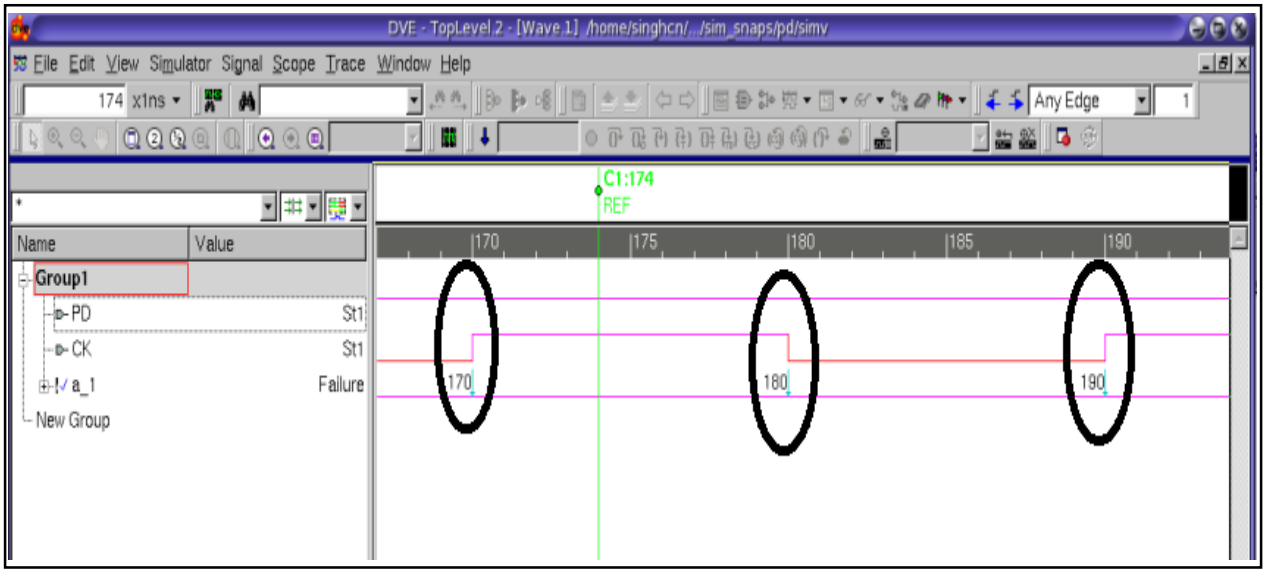

Fig. 20: Simulation snapshot showing failures of the assertion “a_1”

b) Failures of the assertion “a_2” reported as shown in Fig. 21 at times 60ns, 80ns and100ns. “CS” toggling when “PD” is at logic high state.

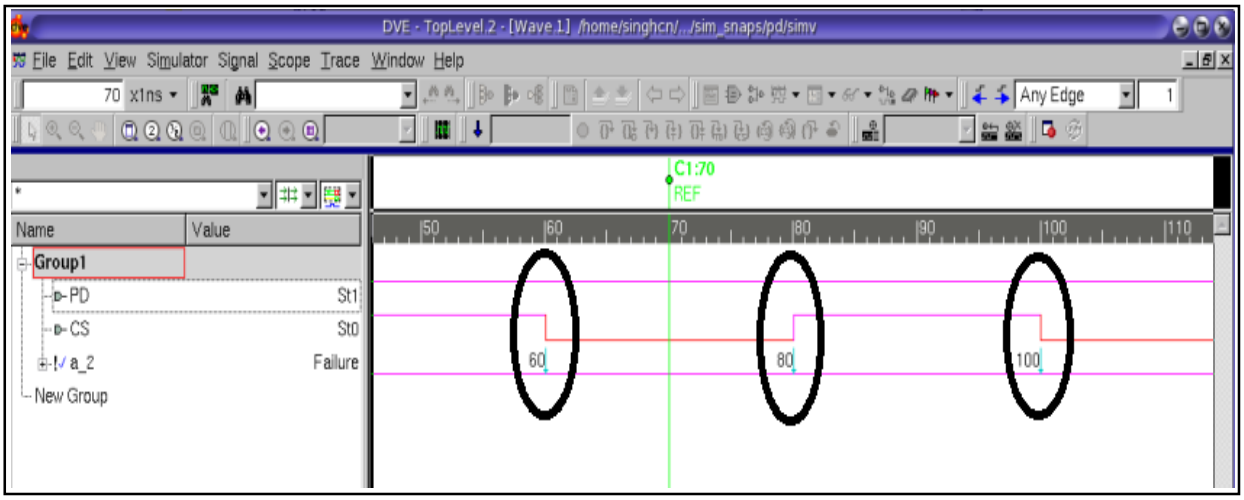

Fig. 21: Simulation snapshot showing failures of the assertion “a_2”

c) Failures of the assertion “a_3” reported as shown in Fig. 22 at times 180ns, 200ns and 220ns. “A” toggling when “PD” is at logic high state.

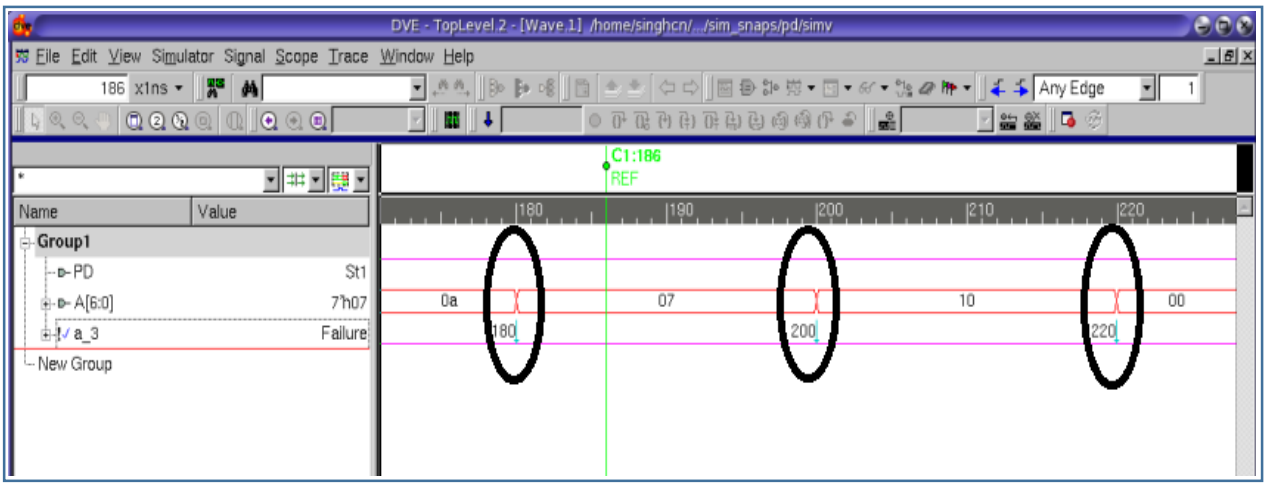

Fig. 22: Simulation snapshot showing failures of the assertion “a_3”

d) Failures of the assertion “a_4” reported as shown in Fig. 23 at times 300ns, 320ns and 340ns. “D” toggling when “PD” is at logic high state.

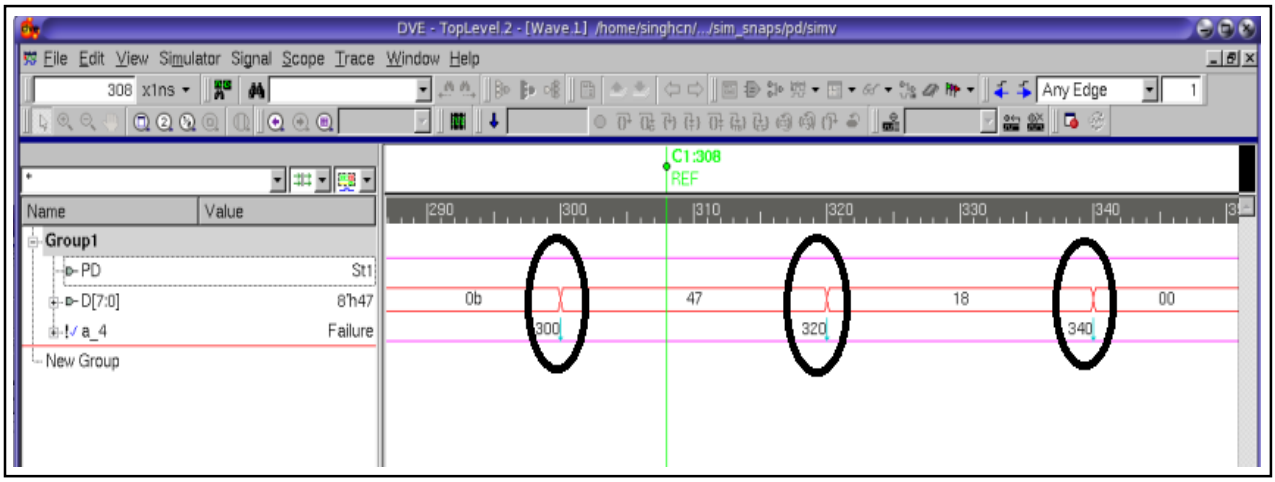

Fig. 23: Simulation snapshot showing failures of the assertion “a_4”

e) Simulation snapshot showing failures of a_5 already shown in the previous section of this chapter in Fig. 17.

f) Failures of the assertion “a_6” reported as shown in Fig. 24 at times 540ns, 560ns and 580ns. “BYPASS” toggling when “PD” is at logic high state.

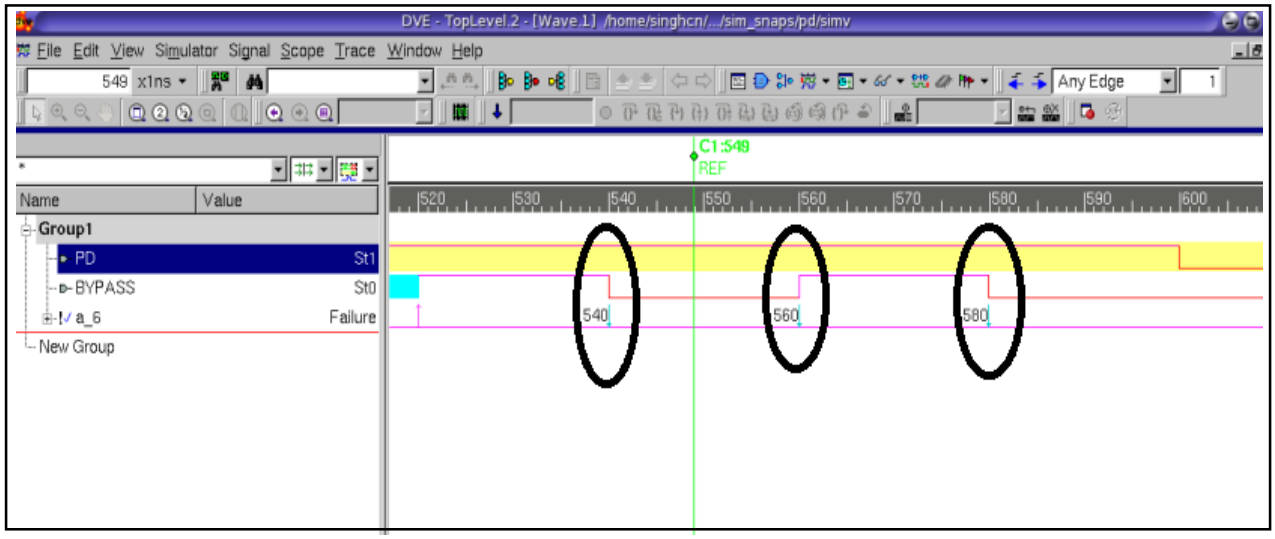

Fig. 24: Simulation snapshot showing failures of the assertion “a_6”

g) Failures of the assertion “a_7” reported as shown in Fig. 25 at times 660ns, 680ns and 700ns. “TP” toggling when “PD” is at logic high state.

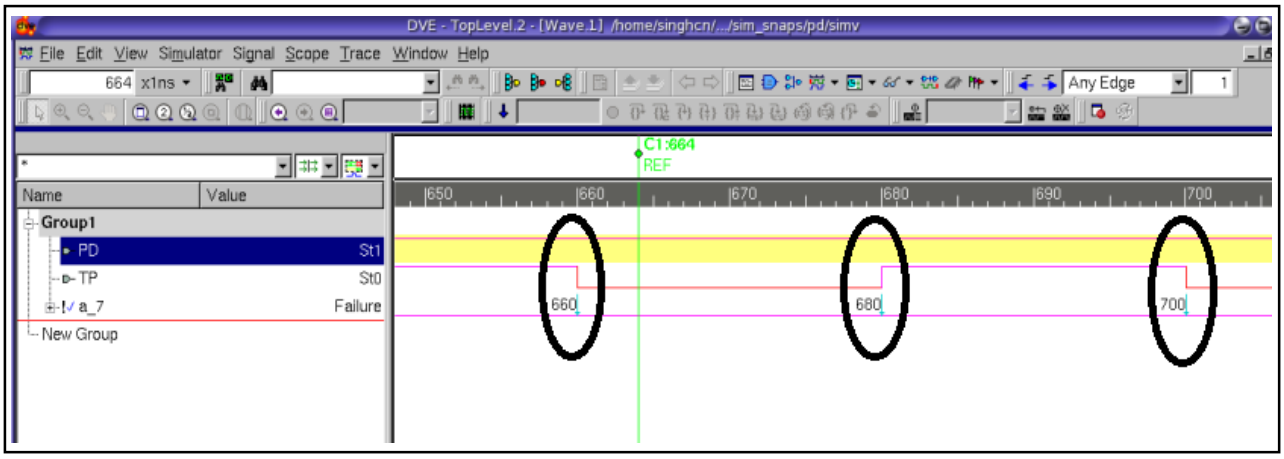

Fig. 25: Simulation snapshot showing failures of the assertion “a_7”

h) Failures of the assertion “a_8” reported as shown in Fig. 26 at times 780ns, 800ns and 820ns. “TA” toggling when “PD” is at logic high state.

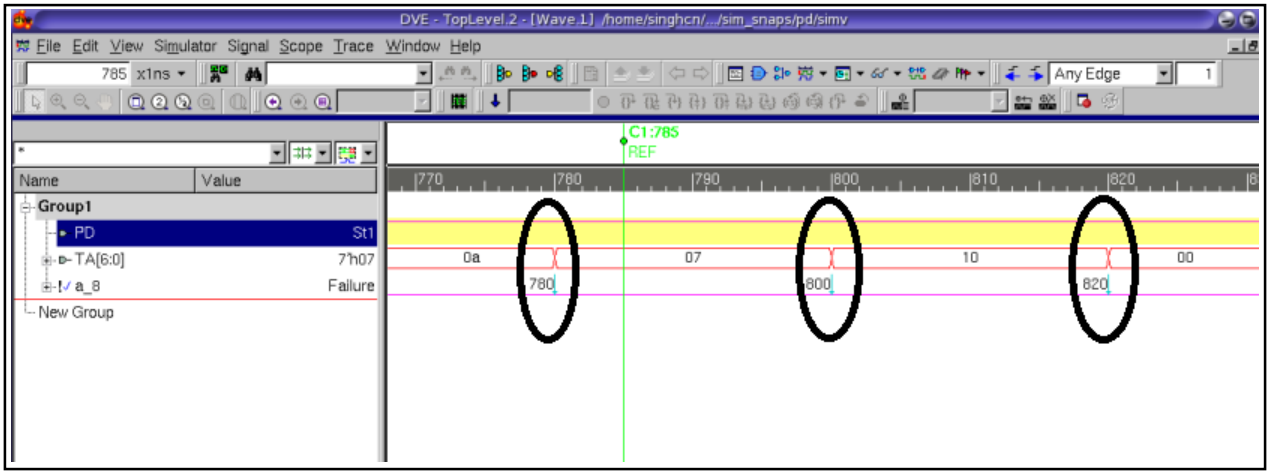

Fig. 26: Simulation snapshot showing failures of the assertion “a_8”

i) Failures of the assertion “a_9” reported as shown in Fig. 27 at times 900ns, 920ns and 940ns. “TD” toggling when “PD” is at logic high state.

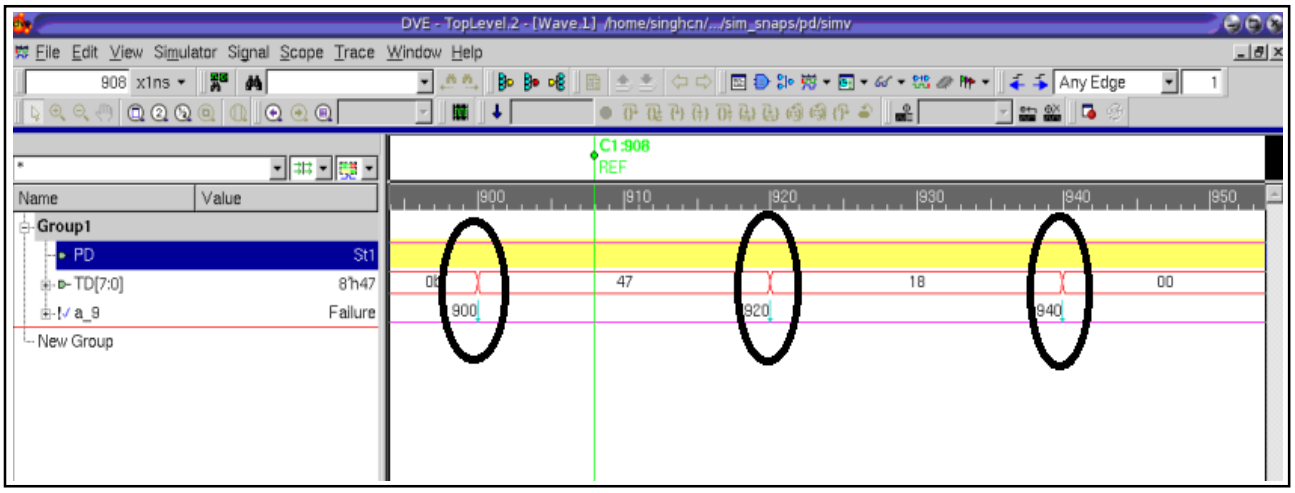

Fig. 27: Simulation snapshot showing failures of the assertion “a_9”

4.2.2 NOP Mode

a) Failures of the assertion “a_10” reported as shown in Fig. 28 at times 210ns, 220ns and 230ns. “CK” toggling when “CS” is at logic low state.

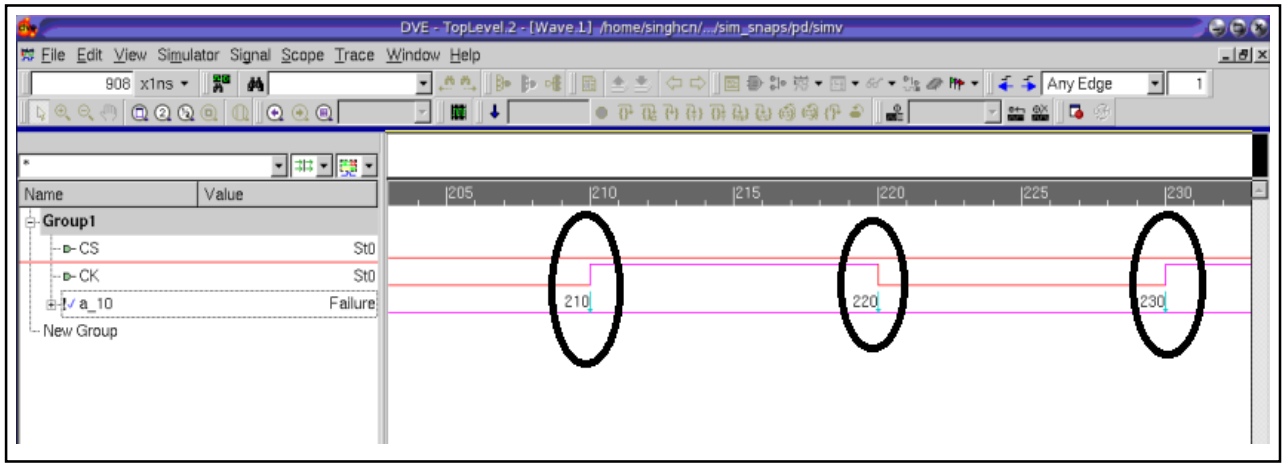

Fig. 28: Simulation snapshot showing failures of the assertion “a_10”

b) Failures of the assertion “a_11” reported as shown in Fig. 29 at times 420ns, 440ns and 460ns. “WE” toggling when “CS” is at logic low state.

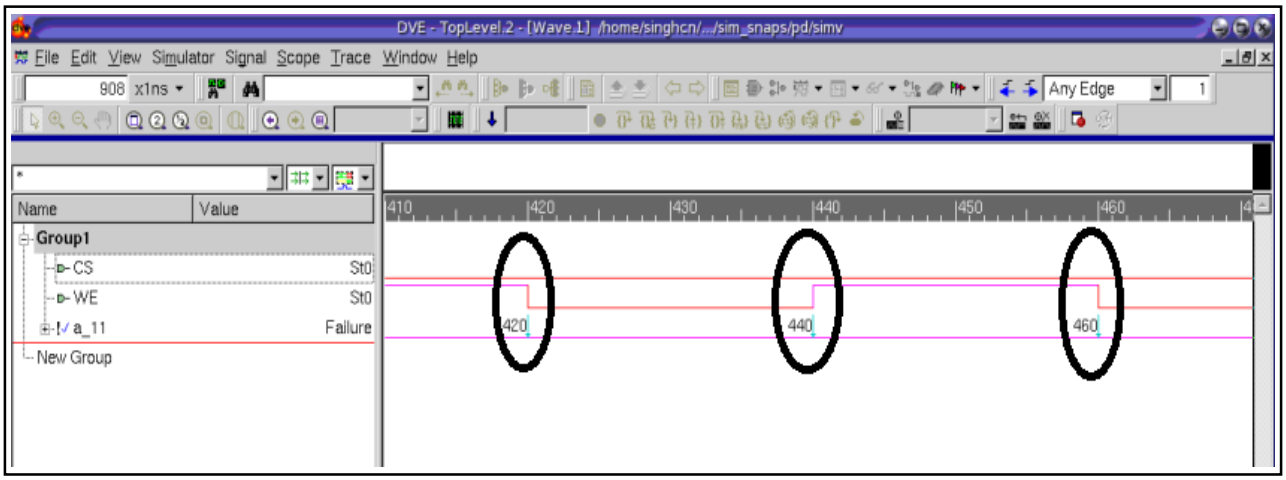

Fig. 29: Simulation snapshot showing failures of the assertion “a_11”

c) Failures of the assertion “a_12” reported as shown in Fig. 30 at times 180ns, 200ns and 220ns. “A” toggling when “CS” is at logic low state.

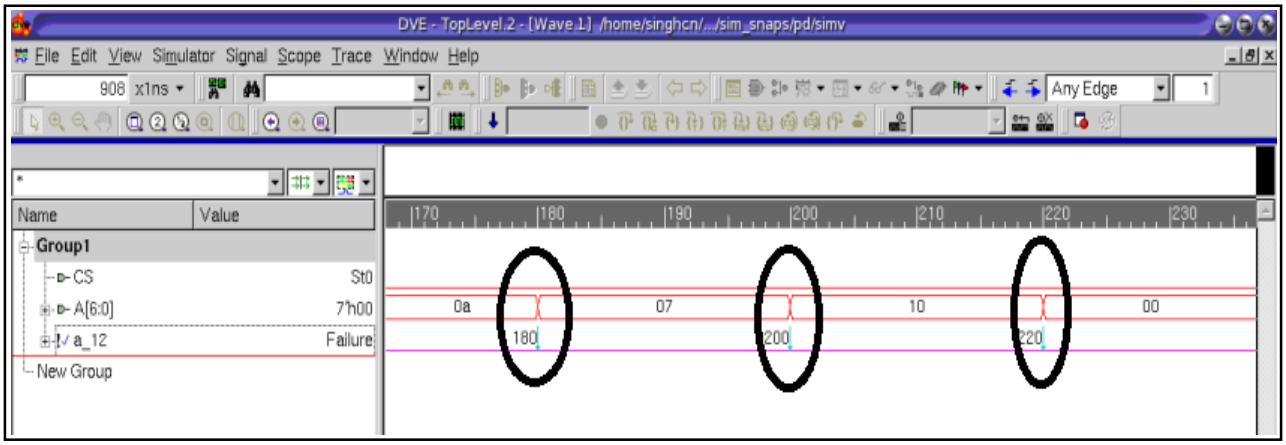

Fig. 30: Simulation snapshot showing failures of the assertion “a_12”

d) Failures of the assertion “a_13” reported as shown in Fig. 31 at times 660ns, 680ns and 700ns. “TP” toggling when “CS” is at logic low state.

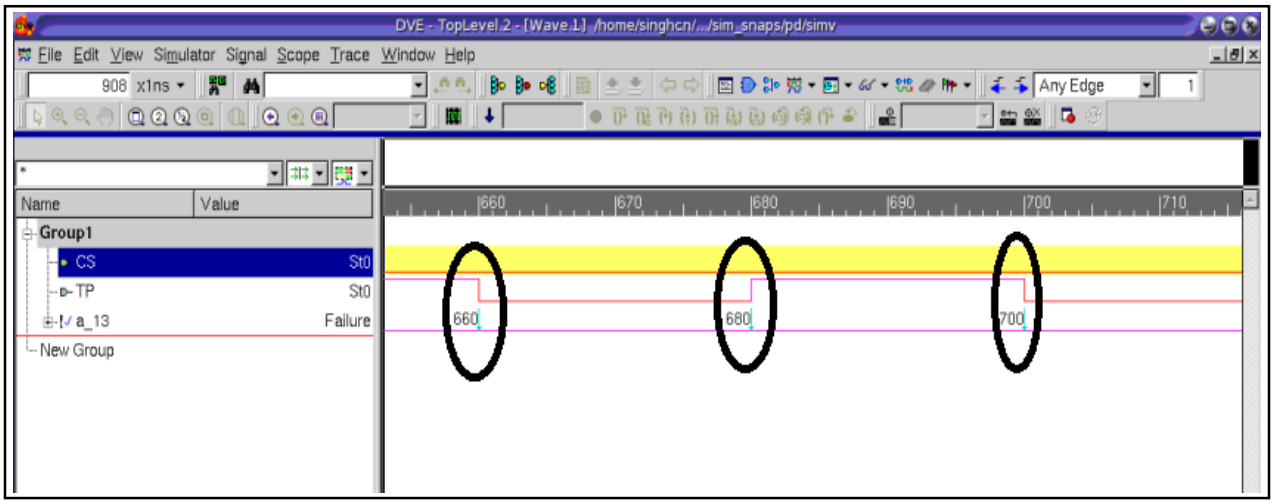

Fig. 31: Simulation snapshot showing failures of the assertion “a_13”

e) Failures of the assertion “a_14” reported as shown in Fig. 32 at times 780ns, 800ns and 820ns. “TA” toggling when “CS” is at logic low state.

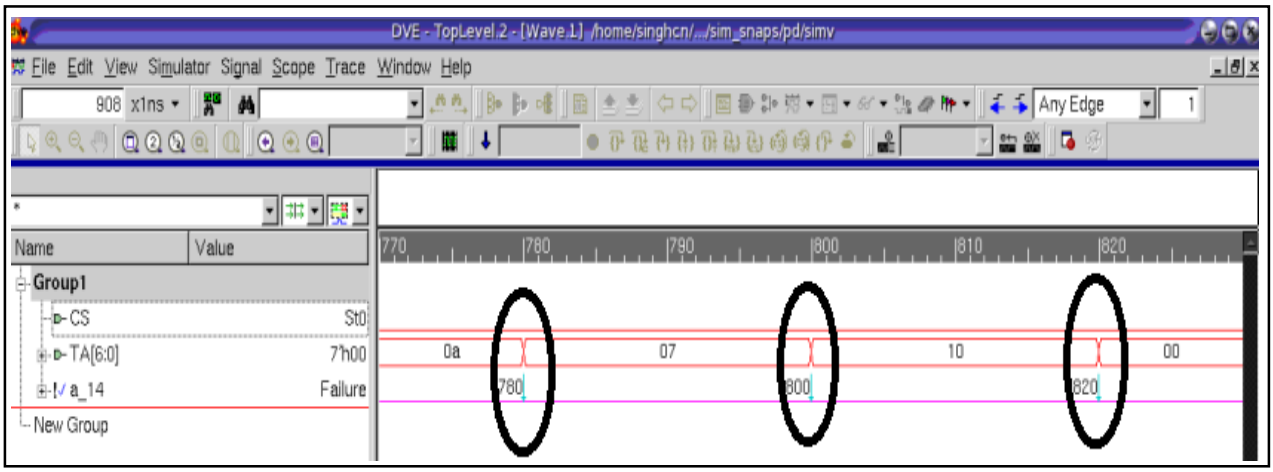

Fig. 32: Simulation snapshot showing failures of the assertion “a_14”

f) Failures of the assertion “a_15” reported as shown in Fig. 33 at times 900ns, 920ns and 940ns. “TD” toggling when “CS” is at logic low state.

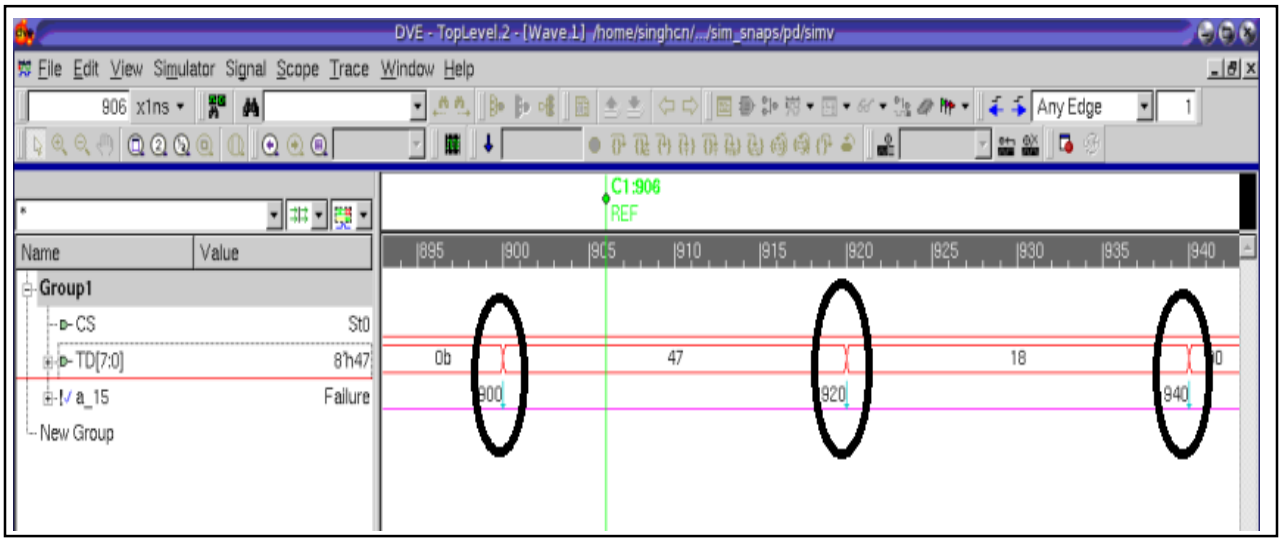

Fig. 33: Simulation snapshot showing failures of the assertion “a_15”

g) Failures of the assertion “a_16” reported as shown in Fig. 34 at times 640ns, 660ns and 680ns. “D” toggling when both “CS” and “BYPASS” are at logic low state (NOP with No BYPASS).

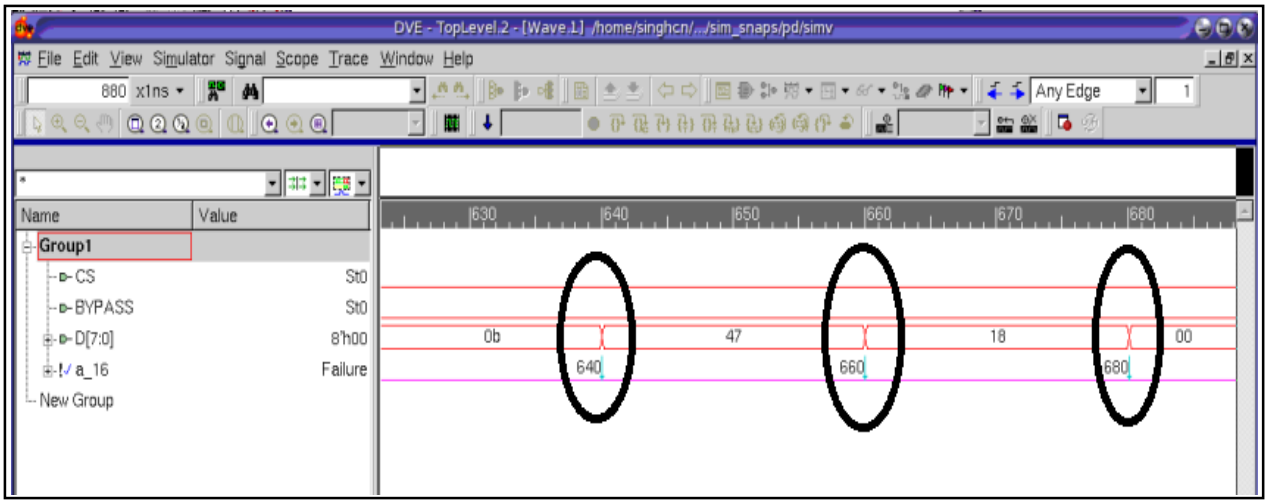

Fig. 34: Simulation snapshot showing failures of the assertion “a_16”

4.2.3 Read mode

a) Failures of the assertion “a_17” reported as shown in Fig. 35 at times 740ns, 760ns and 780ns. “D” toggling in Read mode.

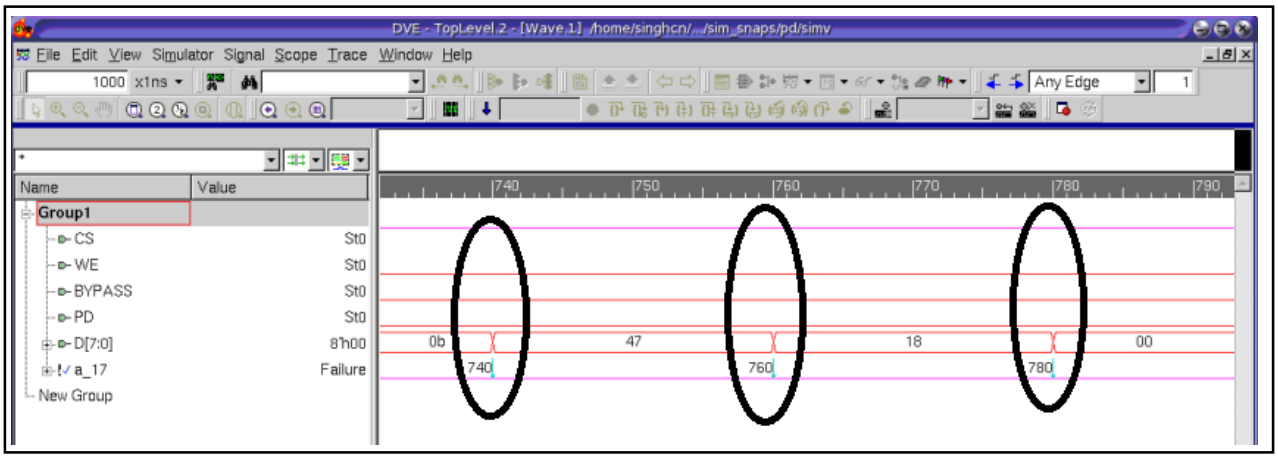

Fig. 35: Simulation snapshot showing failures of the assertion “a_17”

4.2.4 Bypass mode

a) Failures of the assertion “a_18” reported as shown in Fig. 36 at times 860ns, 880ns and 900ns. “WE” toggling when “BYPASS” is at logic high.

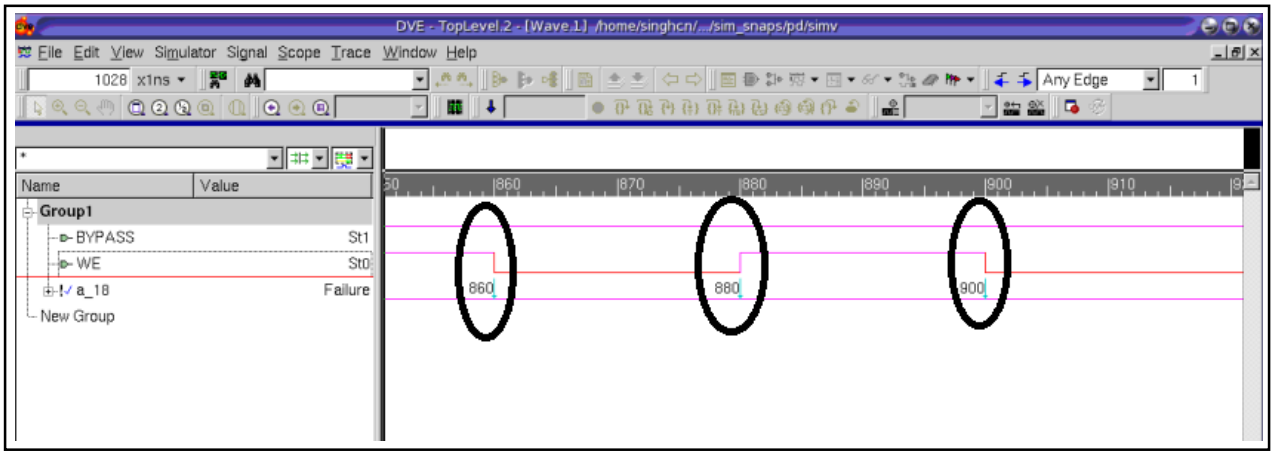

Fig. 36: Simulation snapshot showing failures of the assertion “a_18”

b) Failures of the assertion “a_19” reported as shown in Fig. 37 at times 1020ns, 1040ns and 1060ns. “CS” toggling when “BYPASS” is at logic high.

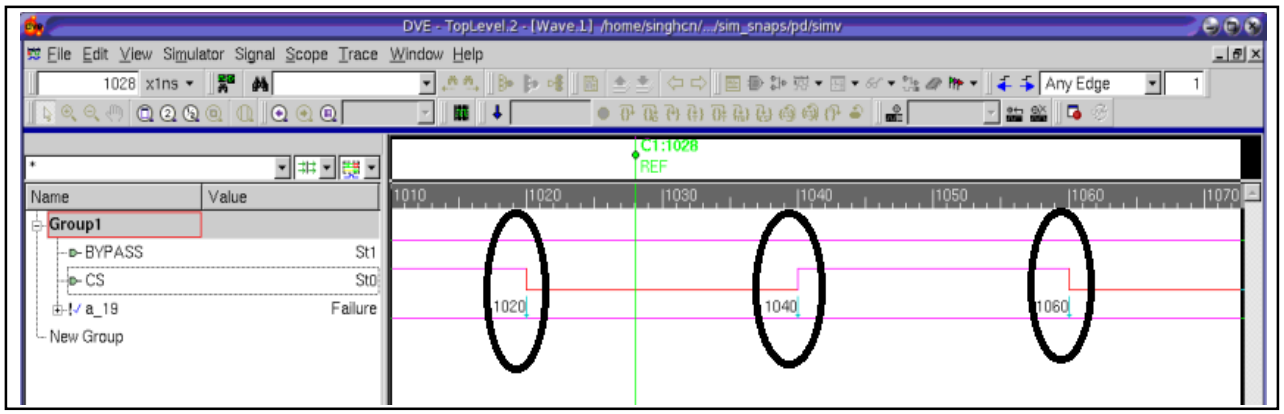

Fig. 37: Simulation snapshot showing failures of the assertion “a_19”

c) Failures of the assertion “a_20” reported as shown in Fig. 38 at times 940ns, 960ns and 980ns. “A” toggling when “BYPASS” is at logic high.

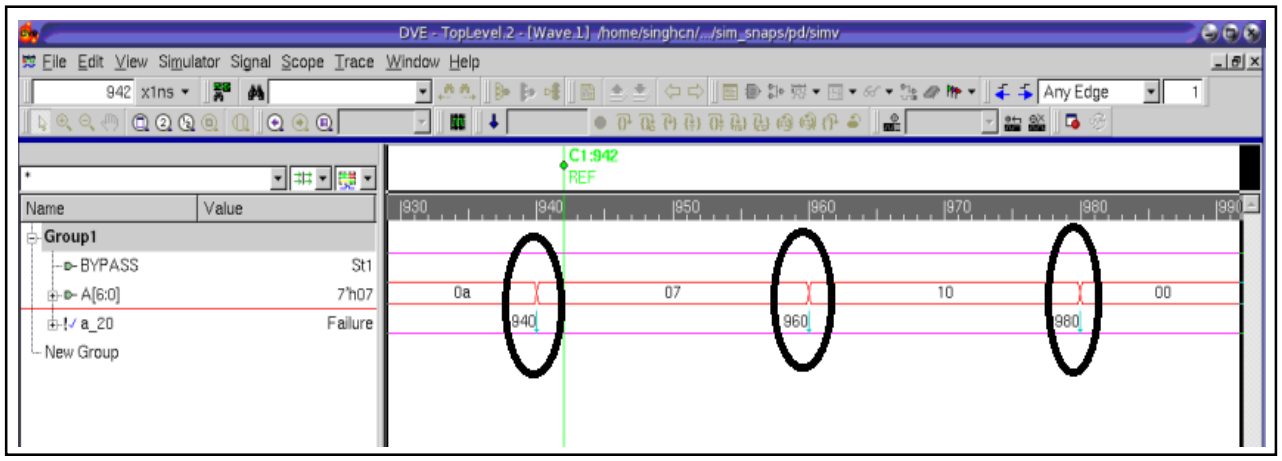

Fig. 38: Simulation snapshot showing failures of the assertion “a_20”

d) Failures of the assertion “a_21” reported as shown in Fig. 39 at times 1100ns, 11200ns and 1140ns. “TA” toggling when “BYPASS” is at logic high.

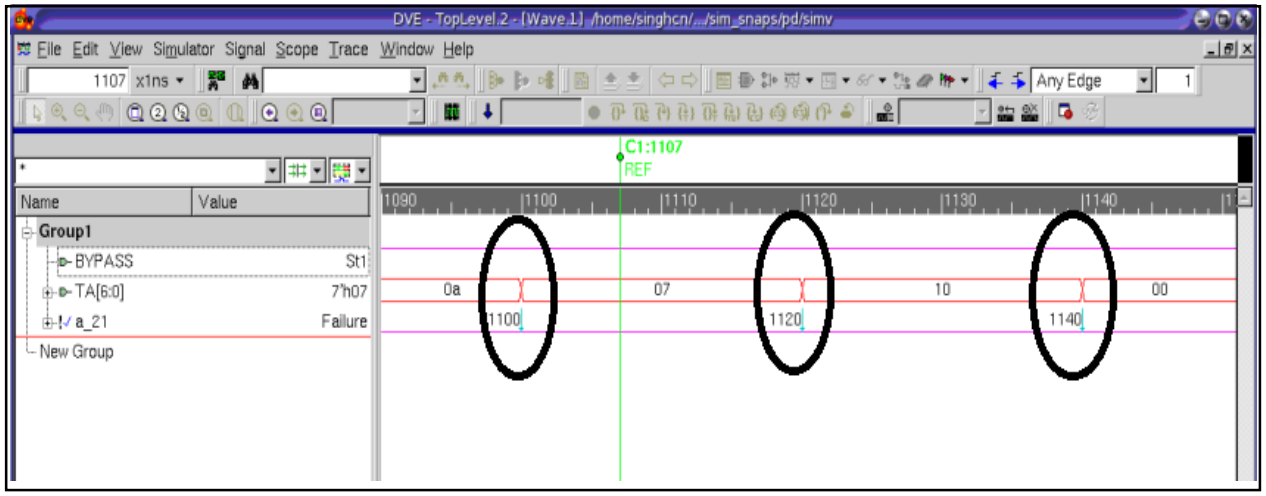

Fig. 39: Simulation snapshot showing failures of the assertion “a_21”

4.2.5 Functional mode

a) Failures of the assertion “a_22” reported as shown in Fig. 40 at times 1100ns, 1120ns and 1140ns. “TA” toggling when “TP” is at logic low.

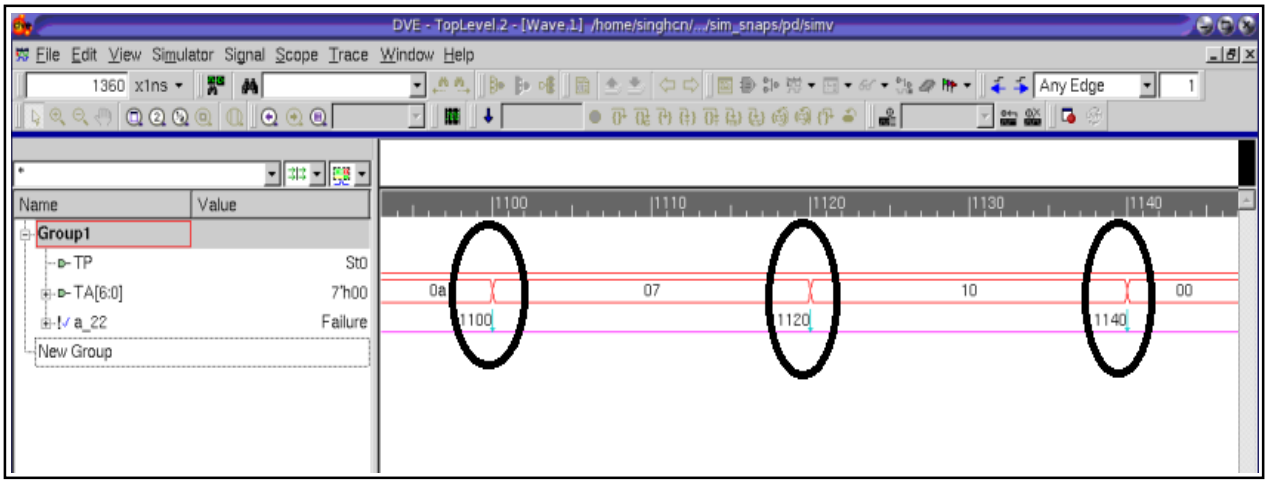

Fig. 40: Simulation snapshot showing failures of the assertion “a_22”

b) Failures of the assertion “a_23” reported as shown in Fig. 41 at times 380ns, 400ns and 420ns. “TD” toggling when “TP” is at logic low.

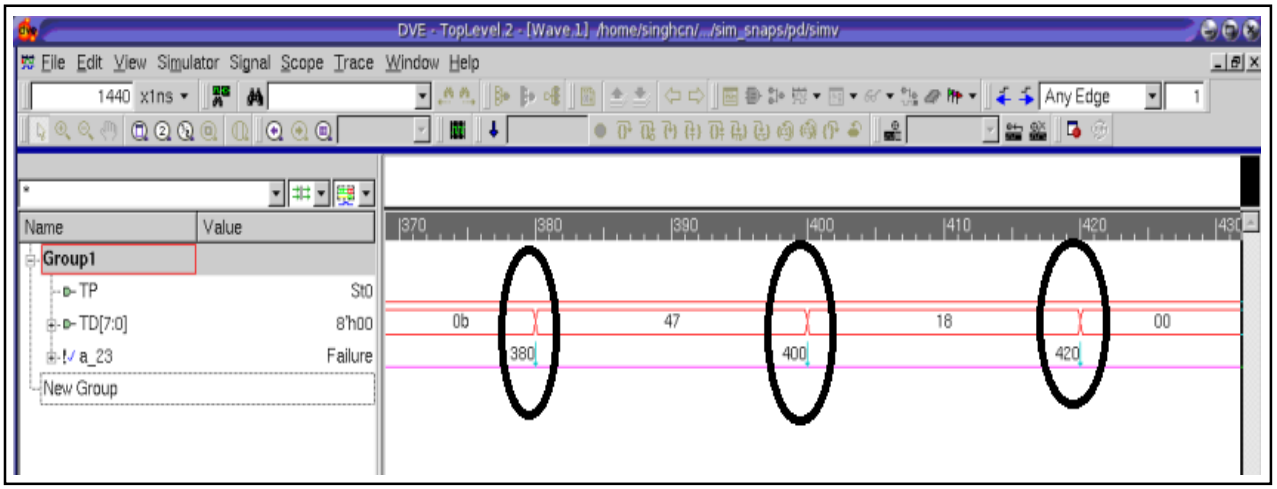

Fig. 41: Simulation snapshot showing failures of the assertion “a_23”

4.2.6 Test mode

a) Failures of the assertion “a_24” reported as shown in Fig. 42 at times 180ns, 200ns and 220ns. “A” toggling when “TP” is at logic high.

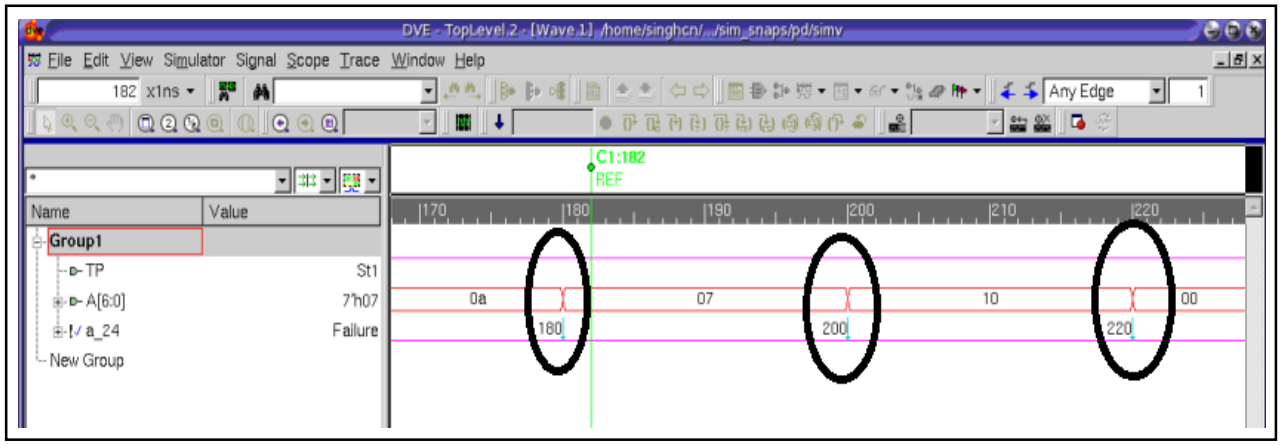

Fig. 42: Simulation snapshot showing failures of the assertion “a_24”

b) Failures of the assertion “a_25” reported as shown in Fig. 43 at times 300ns, 320ns and 340ns. “D” toggling when “TP” is at logic high.

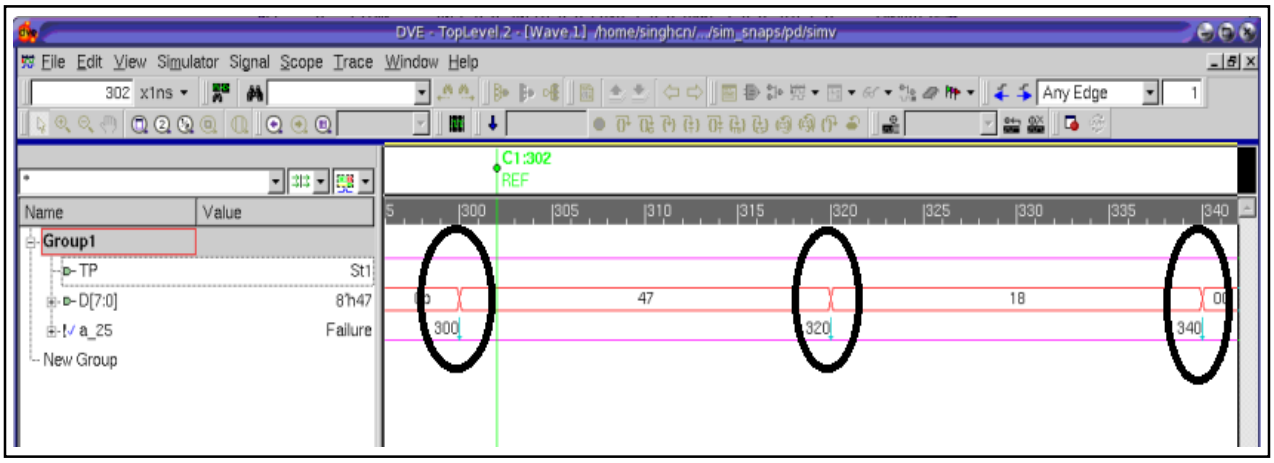

Fig. 43: Simulation snapshot showing failures of the assertion “a_25”

4.3 Avoidable switching activity results

Memory Port	Total number of avoidable switching events	*Percentage of total switching activity
CK	773	64
CS	78	23
WE	93	27
A	106	19
D	323	56
BYPASS	24	29
TP	72	46
TA	116	61
TD	116	61

Table 3: Avoidable switching activity results

*Results based on the stimuli applied to fully exercise the memory-subsystem in all possible modes.

From the above results, we observe that a considerable amount of switching in our memory sub-system can be avoided if we optimize the design at the RTL stage itself. However, these figures depend a lot on the type of design, its complexity and its “use-cases”.

Also, we can intuitively infer from the results that this approach will be more suited for a memory intensive design as compared to a CPU intensive design.

5 CONCLUSION AND FUTURE SCOPE OF WORK

The methodology of finding "Power Bugs" in the design right at the RTL stage has been presented in this thesis. It overcomes the drawbacks of most power analysis and optimization techniques which come into the picture only after the synthesis of the design is complete. The implementation of this methodology using PSL Assertions enables backward compatibility with existing module definitions, tersely specifying the power saving scenarios as compared to Verilog or VHDL. It also enables writing assertions for third party design modules and legacy modules using the "vunit" file. The standard format in which assertion violations are reported in both batch and GUI mode by most modern day simulators further enhances the ease of debug.

The proposed methodology, although being a powerful RTL stage power analysis technique, suffers from a couple of drawbacks:

1) **Accuracy:** In the post-synthesis stage, power analysis is very accurate as we know the power consumption of the constituting physical cells of the design. This information is not available at the RTL stage and hence this analysis will not be as accurate as the post-synthesis one. However, this drawback is not a "show-stopper" for the proposed methodology. Design can still be optimized for low power using this approach. The benefits of the optimizations made can later be confirmed using the post-synthesis analysis results. Also, this approach does not aim at replacing the existing power analysis and optimization techniques, it is meant to go hand-in-hand with the existing ones. The

"Power Compiler" tool provided by Synopsys can also be used along with this approach to cross-check the results and do further analysis once the RTL is functionally verified.

2) **Conflicting Assertions:** Sometimes, while fixing the design based on the assertion failures, we observe that the fix of assertion failure "A" might lead to other assertion failure "B" and vice-versa thereby causing a loop in our approach. In such a scenario, we need to take a call based on which fix is more beneficial, modify the RTL based on that assertion and disable the other conflicting assertion altogether. The mechanism to disable individual assertions has already been discussed before.

The idea discussed and implemented in this thesis can be extended in multiple directions. EDA vendors can standardize this methodology and include it as an add-on feature in their simulators. Design teams can adopt this methodology in which they would develop the "vunit" file along with each associated RTL modules.

Using the dynamic power characterized for each port in the design, we can extend this idea to do power estimation as well. As an example, say the assertions related to "CS" port report violations at 10 times and the characterized dynamic power for "CS" is 10 microwatts. Optimizing the design to prevent needless toggling of "CS" port will save, 10x10 or 100 microwatts. The same calculation can be done for all the other ports with failing assertions. By adding this value for all the ports, we can report the total dynamic power that could potentially be saved. In this way, the power verification at the RTL stage can be made even smarter, thereby enabling faster time-to-market.

REFERENCES

[1] Michael Keating, David Flynn, Rob Aitken, Alan Gibbons and Kaijian Shi, *Low Power Methodology Manual: For System-On-Chip Design*, Springer, 2007.

[2] Preeti Ranjan Panda, Subrangshu Das, Sukumar Jairam, Abhishek Ranjan, Nikhil Tripathi and Sanjiv Narayan, "Tutorial T2: System and RTL Low Power Design", *VLSI and Embedded System Conference*, 2013.

[3] Christian Piguet, *Low-power Electronics Design*, CRC Press, 2008.

[4] IEEE *Standard for Design and Verification of LowPower Integrated Circuits*, 2009.

[5] T. English, K. L. Man, E. Popovici and M. P. Schellekens, "HotSpot:Visualizing dynamic power consumption in RTL designs", *Design & Test Symposium (EWDTS) East-West*, 2008.

[6] C. Karfa, C. Mandal and D. Sarkar, "Verification of Register Transfer Level Low Power Transformations", *VLSI (ISVLSI) IEEE Computer Society Annual Symposium*, 2011.

[7] *Fabless Semiconductor Report*, ARM Ltd., 2004.

[8] S. Ahuja, D. A. Mathaikutty, G. Singh, J. Stetzer, S. K. Shukla and A. Dingankar, *Power estimation methodology for a high-level synthesis framework*, Quality of Electronic Design, 2009.

[9] IEEE Standard (1850-2010): *Standard for Property Specification Language (PSL)*, - IEC 62531:2012(E), 2012.

[10] D. Baghel, B. Pandey, M. Pattanaik, A. Shukla and M.P. Dev, "Clock gated low power sequential circuit design", *Information & Communication Technologies (ICT) IEEE Conference*, 2013.

[11] J. Shinde and S. S. Salankar, "Clock gating - A power optimizing technique for VLSI circuits", *India Conference (INDICON)*, 2011.

[12] K. C. Pokhrel, "Physical and Silicon Measures of Low Power Clock Gating Success: An Apple to Apple Case Study" *Synopsys Users Group (SNUG)*, 2007.

[13] Sasan Iman and Massoud Pedram, *Logic Synthesis for Low Power VLSI Designs*, Kluwer Academic Publishers, 1998.

[14] Cindy Eisner and Dana Fisman, *A Practical Introduction to PSL*, Springer, 2006.

[15] H. Foster, A. Krolnik and D. Lacey, *Assertion-Based Design*, Second Edition, Kluwer Academic Press, 2004.

[16] B. N. Uchevler and K. Svarstad, "Assertion based verification using PSL-like properties in Haskell", *International Symposium, Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, 2013.

[17] Wai-Kai Chen, *Memory, Microprocessor, and ASIC*, CRC Press, 2003.

[18] Jerry Whitaker, *The Electronics Handbook*, CRC Press and IEEE press, 1996.