Abstract

Power management is mandatory in SoC design. It is a well known fact that any consumer product quality cannot be compromised despite the massive additional complexity that power efficient design imposes on them. Consumer products must only use the minimum power based on what they are connected to. This need, to reduce the power consumption has been recognized as a design issue and it is critical when ICs become more complex and faster and time to the market is shrinking.

Power management is currently performed bottom-up. This is achieved by adding low-level design features in the hardware and the control mechanism that can be used by the software applications. While tools exist to help the hardware engineers analyze the effects of design decisions on power consumption, the software developers has no way to view the effect on power, of their decisions taken on architecture, design or code. In addition, it is hard to understand the impact of their low power features for hardware engineers. Thus, given that most of the dynamic power choices are made in software, the biggest power gains are made when the application software layers are power-aware. Considering this, there is a strong need for a research to correlate power consumption data from the hardware domain in such a way that a software engineer can make informed choices about power management algorithms [1].

Considering an Ethernet System, we explore its power efficiency when used with its hardware and software address look-up table versions. This will let know the software engineer what to use, either the hardware or the software address look-up table according to his requirement. This has been achieved in the following major steps.

- Studied in detail the Cadence SoC Verification Kit which includes the necessary design IP to construct a realistic Ethernet Switch SoC.
- Explored the construction of software and hardware address look-up table.
- Created smart benchmark programs and integrated them with the Ethernet SoC using the hardware software co-verification methodology from Cadence.
- Collected power data after running the benchmark programs with respect to hardware and software perspective.
- Found the various modules consuming more power in each system and provided valuable suggestions on improving the performance of both hardware and software versions of the module.

Acknowledgements

It is really my pleasure to thank those who made this thesis possible.

First for all, I owe my deepest gratitude to my supervisor, Dr. Kerstin Eder, whose inspiration, support and encouragement, right from the beginning till the final level of the thesis helped me understand and proceed with confidence.

It is an honour for me to thank Nick Heaton from Cadence who has made available his support in a number of ways by providing all the necessary software and documents required and I am thankful for his timely help that formed the base for this project.

This thesis would not have been possible without Swaminathan Venkatesan and Ashwin Menon from Cadence who spend their time for me in clearing the questions that I had despite their busy schedule.

I would like to thank my parents for providing me full support in pursuing this degree at the University of Bristol. Even though I have not seen them for one year, their encouragement and moral support helped me through the tough times.

Last but not the least, I would like to thank Lord Almighty for being with me always.

Table of Contents

Chapter 1

Aims and Objectives

The aim of this project is to undertake and investigate a system power analysis in order to attain maximum power efficiency and to let the software engineers know how the power management can be done efficiently ahead of completing the software development. It is to show how a system level power analysis can be performed and how this analysis reflects back to the hardware view. This project will analyze and critique system power consumption data from a software engineer's perspective.

We have taken a System-on-Chip (SoC) for this project which consists of four port Ethernet Media Access Controls (MACs) and one Address Look-up Table which is of key interest. The other modules in the SoC include a processor, memory and few assorted peripherals. This system has been given two options to choose from, an embedded software and a hardware IP, for performing the Address Look-up Table functionality. The power consumption of the overall system may differ while using the hardware and software look-up table blocks. The hardware look-up table block (written in Verilog) and the software look-up table block (written in C) also has their advantages and disadvantages in other perspectives. It is true that there is a trade-off between hardware and software versions when it comes to performance, maintenance and size.

However, which version is better to use when it comes to saving power? Given both hardware and software versions of look-up table algorithms in a system, how will a software engineer know which one to choose depending on his/her requirements?

This project analyses the power consumption of this Ethernet system in two main perspectives, the hardware Address Look-up Table perspective and the software Address Look-up Table perspective. This involves the understanding of the overall system, detailed study on the construction of the hardware and the software versions of the look-up table modules, the switching between hardware and software versions, study on the Cadence verification kit, and study on Universal Verification Components (UVCs) etc.

The understanding of Cadence SoC Verification Kit includes the study of all the necessary design IP to construct a realistic Ethernet Switch SoC, and a complete related tool chain for power analysis. This verification kit gives us the ability to automate and reuse the advanced verification techniques. This includes the full range of EDA tools that we can use. This kit comes with various Cadence simulation engines which include Incisive Enterprise Simulator, Incisive Formal Verifier, Incisive Enterprise Verifier, Incisive Enterprise Manager, and Incisive Specman Elite. Most of these tools are used for this research.

Benchmark programs are the key elements for this project. They represent the system scenario. Understanding, creating, executing and running the benchmark programs with the use of different seeds for this system gives us the power information. The output is

presented in a table format containing all the information required for understanding the power consumption. This is further analysed in way so as to find which version is the most power efficient.

The important step of this project is to correlate the data obtained to the software level, to help the software developers understand the data and to know how power management can be done efficiently. This is achieved by analysing the results obtained after running the benchmark programs and by providing valuable suggestion on improving both the versions of the look-up table modules by performing critical analysis.

Chapter 2

Background, Context and Previous Work

2.1. Introduction:

A vast number of low-level hardware techniques have been developed in order to assist the hardware designers these days. To name a few, clock-gating, power-shut off, and dynamic voltage/frequency scaling are the most widely used techniques. For example, the RAM which is partitioned into various segments can be individually put in shut-off or in stand-by condition which can be done in the hardware layer, use of multi-voltage islands, dynamic scaling of threshold voltage and clock frequency. These techniques will help conserve the battery power while delivering high performance [6].

Other examples include the sleep mode in which the firmware requests the CPU to sleep after a particular timeout value, and hibernation mode in which the application layer requests for the hibernation, after the user presses the hibernate button. Techniques for power management should also assure that every part of the product should receive power in a proper and efficient way. When there is high power dissipation, it generally increases the junction temperature of the transistors and the resistances of interconnects. Thus there will be shortfall in the performance if the power dissipation is not considered.

New strategies and design techniques are required to address the power management issues. These strategies need to address every level of abstraction within the design-cycle. The gains, thus procured are of a much higher magnitude when considered in the earlier phases of the design, say from system design phase [5].

The software developers will not have any idea on how their software will have an impact on the hardware power dissipation. If they could know this information way before they start their phase in the project, there is a chance that the power saved will have a big number.

This is the main intention of this project, a research through which the power consumption data from the hardware layer can be correlated to the software layer and software engineers can make decisions way ahead of the coding phase. The following section analysis the various steps that have been used already in order to achieve low power. We will go from the very basics of power dissipation concepts and then into a wider context. Further down, we will look into how low power intent is verified which forms the beginning step of this project and finally how the latest trend in using common power format can help in saving power.

2.2 Power Dissipation in CMOS:

The total power dissipated in a CMOS is a function of its switching activity, voltage, capacitance and its transistor structure. Thus the total power is the sum of dynamic and the leakage power. The switching power and the short circuit power represent the dynamic power.

2.2.1 Switching Power:

Switching power is the power dissipated during charging and discharging of the capacitance. This is given by the following equation [1].

$$Power_{switching} = f.\alpha.C.V_{dd}^2$$

Where,
f = The switching frequency
α = The switching activity
C = The effective capacitance
V_{dd} = The supply voltage

Short-Circuit Power:

During the time when the gate switches its state, there is a power dissipated by an instantaneous short circuit between the ground and the supply voltage. This power dissipation depends on the short circuit current during the switching, supply voltage and the switching frequency [1].

$$Power_{Shortcircuit} = f.V_{dd}.I$$

I = The short circuit current
V_{dd} = The supply voltage
f = The switching frequency

From the above equations it is clear that the switching activity and the clock frequency reduction will reduce the dynamic power dissipation. The dynamic power will be dissipated only while switching.

2.2.2 Leakage Power:

The leakage power is continuous and it is dependent on the supply voltage, threshold voltage and the size of the transistor [1].

$$P_{Leak} = f.(V_{th}.V_{dd}.W/L)$$

V_{th} = The threshold voltage
V_{dd} = The supply voltage
W = The transistor width
L = The length of transistor

2.3 Power Management in a wider Context:

Power management for SOC's is done by many ways with respect to, clock, memory considerations, voltage, and frequency. The following are the most widely employed power management techniques. Before we start with the correlation of the power consumption data, we will have to understand the basic ways in which the industry has been moving in order to save power. After this, we will look into the new concepts on the implementation [6].

2.3.1. Clock Gating:

The specification of the system clock requirements, controlling clock frequency and gating are critical in reducing the predefined factors of power consumption. For saving power, the clock gating is supported by additional logics to the circuit. This disables certain portions of the circuit so that the values do not change during the normal operation. This in turn will make the switching power to nil and only the basic leakage power will be there. In simple words, turning the clocks off when they are not required is called clock gating. This is generally done by any of the two ways, soft control and dynamic control [6] [1] [14].

Soft Control:

In soft control method, the software can be extended and it can be made to enable or disable the clock. It is also now possible to switch the clock frequency to various parts of the design.

Dynamic Control:

In dynamic control method, enabling and disabling of clock frequency is done by the hardware control. If the hardware detects request from functional units for any specific function it sets the clock frequency accordingly. In both the cases, the control is done through a central clock distribution circuit.

2.3.2 Using multiple V_{th}:

For power optimization, gates can have different threshold voltages. Most libraries these days come with different cell switching thresholds. They can be used in a circuit to meet the demands for speed, area and lower power dissipation [12] [14].

2.3.3 Operand Isolation:

Certain parts of the circuit need not respond to their inputs all the time. During this time their data paths can be made inactive and it will have a constant value. This inactivation of the data path results in saving the dynamic power [13].

2.3.4. Memory Considerations:

Power consumption also depends on the memory usage. A good memory configuration means a lot of power can be saved. There have always been trade-offs between the usage of RAM and ROM, hardware functionality and software functionality [2].

Trade-off between RAM and ROM:

The code can either be placed in ROM or RAM and there have always been tradeoffs between them. Usage of RAM will provide a good performance however the power usage by RAM is high compared to the ROM. Most of the dynamic power requirements in RAM are in pre-charging and discharging the bit lines. The power consumption in RAM can be reduced to an extent by partitioning. For e.g. 1024X32 RAM can be partitioned into four 256X32 RAMs. Sub-partitioning within a RAM is also possible. ROM on the other hand uses less power. However, there is a limit for its usage which depends on the application and the performance.

2.3.5. Dynamic Voltage Scaling:

It is possible to vary the voltage based on the demands of the upcoming task. This is generally done by using a Power Control Module (PCM). Voltage can either be increased or it can be decreased. Overvolting is a dynamic voltage scaling technique through which the voltage is increased which results in improved performance. Undervolting is a technique in which the voltage is decreased which results in power conservation. The MOSFET operates according to the input voltages which switch between high voltage and low voltage during the normal operation. The toggling of voltage depends on the charging and discharging of the capacitance. When we apply higher voltages, it results in faster charging and faster discharging which in turn results in a faster operation of the circuit. Because of this there is a trade off in choosing the voltages since the system will become slower for undervolting and consumes power for overvolting [14] [2].

2.3.6. Dynamic Frequency Scaling:

When a section of the design has to wait for other tasks to get completed, the frequency can be lowered. This technique is generally done in conjunction with dynamic voltage scaling since the power is depended much on the voltage then the frequency. These day is has become common to operate a system briefly in its peak speed and then stay in a deep idle state than running it in a lower frequency [14] [2].

2.3.7. Power Islands:

A system generally consists of different sections with different performance requirement. For e.g. an I/O controller can be made to turn-off its interfaces which are not needed when no devices are currently connected to them or a mouse movement for keyboard entry for a sleeping PC. In traditional systems, it is possible to have different power

supply for different chips. However it does not allow us to control or give different voltage levels within a chip. Here, a new concept called power island or voltage island comes in picture where different areas of a single chip is made to operate at different voltage levels and different frequency levels independent of each other. High voltages will be used in the most performance critical areas and nominal voltages will be used in the performance critical but not performance limiting areas. Low voltages will be used in the non-critical areas of the design. Different voltages can be given to different areas like cache memory, CPU, Logic, memory and analog sections of the SoC [4] [14].

2.4. Verifying the Low Power Intent of a Design:

Various verification methodologies have been used in this project to explore the power intent. It is obvious that we will have to look into a variety of techniques used for performing the verification for power intent. The low power design architectures add more complexity to the verification of the system. There can be different modes of operation in a low power design and all the modes should be verified. Thus the verification space increases along with the increase in the number of modes. Certain features may not be available in all the modes. There should be a proper understanding of what features is valid in what mode. All the stake holders should agree on this, including the Architect, System designers, RTL designers and Verification engineers. The verification should be done in such a way that all the modes are entered and exited without any errors and all the legal transitions are checked. This is then finally documented with the intent of power architecture and communicated with all the stake holders. The following are various stages of verification. It should be noted that the following steps are only for verifying a system with respect to its low power behaviour [5].

2.4.1. Verification Plan:

The verification plan in terms of low power design defines testing plan for all the power modes and scenarios and all the features in a mode. The various modes of operation and all the valid transitions with its priority are defined. The various conditions in which transitions can occur are defined. The various methods that cause a mode change like writing a register, response to external signals, effects of other transitions are defined. The coverage items should be planned. Tests for assertions to ensure power events happen only at legitimate times should also be included [2].

2.4.2. Test benches:

This part involves the creation of the advanced power aware test benches. This includes the important coverage and assertions which are to be covered. The various power modes and their sequences, the behaviour of the system during each mode, the verification of power control logic, the interfaces between the various domains, power management decisions, execution of the power management, system operation in each mode, and the interactions between domains and units [2].

2.4.3. Execution of Test Benches:

The test benches are run formally and also involve simulation and emulation. During the execution care must be taken so as to cover all the possible states. To name a few, the interaction between the analog components, the transitions and the standby mode, the power OFF and power ON sequence, the interactions between the various domains, the power control module for the detailed domain level control etc.

2.4.4. Coverage Report and Analysis:

A report has to be produced which consists of the coverage and the assertion coverage. This will include the reports from software, hardware and CPU point of view. The metrics should include the coverage of each domain state and transition, the assertion ensuring the proper state transitions, assertions and coverage of individual control signals. Generally the various states would be High, Low, Stand-by and OFF. The coverage should also include the unit level coverage where each unit can have multiple domains and also can have independent control of its power.

2.4.5. Verification Closure:

When all the planned features and the goals are met which are given in the plan, the verification can be closed.

2.5. Common Power Format and EDA tools:

The Ethernet SoC uses this format files for its low power intentions. Also, the usages of these techniques have increased in the industry. Hence it is necessary to have a good background study on this topic. Using the conventional design flow, the designers had to manually model the additional considerations to verify the impact of low power during their simulation stage. This involves multiple definitions for the same information again. These definitions for the low power model will involve another set of definitions for synthesis, placement, verification and equivalence checking. Ever after this there was no guarantee that what was verified really matched the design. This in turn affected the productivity, added more silicon failure, added more time to market the product etc [15].

The power information is available only on papers most of the time and is generally not available in the functional description. The information on what the power modes are, when a block can be turned off, when block requires retention, where the level shifting is done, where should the isolation be inserted, what hierarchy should have what power supply are often not given in the functional description. This power intent should be separated from the implementation [3] [11].

This resulted in a need for a standard which can be reused for power saving. It should be available and introduced early in the design phase. This made way for the Common Power Format. The art and the techniques of the low power design styles will be

enhanced by the Common Power Format [CPF]. By using the CPF, the power intent has to be given only once as it can be reused and understood by most of the tools. The Common Power Format is a special format file for specifying all the power saving techniques in the design phase [7] [9] [10].

The following are benefits of the CPF:

- CPF helps the designers to find the optimal power specifications to achieve the desired specification. This lets the designers to find the way get the best trade-off when it comes to timing, area and power.
- Unlike the conventional design flow, using CPF helps the teams to increase the productivity by avoiding repetitive tasks.
- By using CPF the manual work done is reduced and it paves the way for automation. This joins hands with robust verification, thus eliminating the failure risks.
- The RTL files when used in the conventional way needs to be modified each and every time for different power intent. The usage of CPF files will avoid this measure for RTL files. This paves way for design reuse.
- The CPF is very easy to adopt and it also helps to overcome cost, time and deployment issues.
- Using CPF also enables RTL functional verification which helps in validating the power related operation [15].

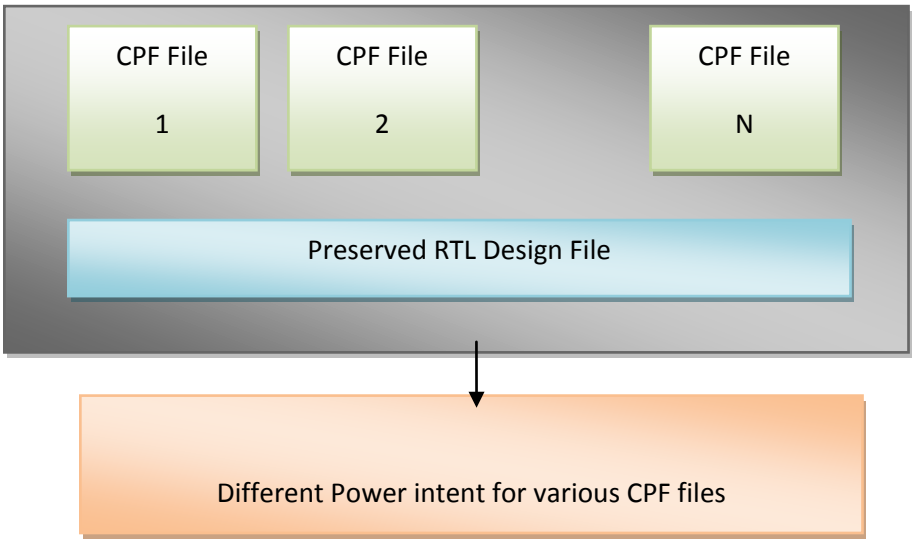


Figure 2.1 Using Multiple CPF Files for the same RTL

2.5.1. Using CPF for Power Intent:

The usage of CPF separates the power intent from the RTL intent. This allows us to capture the power intent separately. Thus the full design requirement can be captured by capturing the power intent, design intent and the timing intent separately. The RTL file

can be instantiated different number of times and each may have a different CPF file with different power intent [7].

All the stages use the power intent specified in the CPF. The stages would be Design, Quality Check, Low power verification, Synthesis, Net list Checks, Equivalence checks, Test Design and Physical Implementation.

Low power verification using CPF, basic stages:

- The initial step would be the creation of the RTL file for the design intent and the creation of the CPF file for the power intent. Here the RTL file can be a reusable one since the CPF will take care of the power intent.
- This step involves the verification of the contents of the CPF and the RTL code. They must be check so that they are syntactically correct. Check are also done so as to make sure that the design intent and the power intents are in alignment.
- Now the system is check for its low power behaviour over its normal behaviour. Thus the functionality of the system is checked through simulation by super imposing the CPF with RTL. There will be control signals in the CPF for the isolation, retention and the power down. These control signals will be then generated in the power controller. Thus the low power behaviours can be generated by the assertions of the corresponding control signals. Now, different cases can be examined by just changes the CPF file for various power intents [2] [9] [25].

2.5.2. Low Power Synthesis:

We will have to perform the synthesis of the low power feature. Here the low power structures along with the same CPF file used in the simulation are synthesized. During this stage the compiler adds the entire low power cells. The compiler does the following during synthesis,

- The output of the power domain will be added with isolation cells
- The specified input cells will be added with the isolation cells
- The level shifters will be added to signals crossing the voltage domains
- All flops will be replaced by the Retention flops in the specified locations

During the simulation stage the virtual connections will be created. However during the synthesis phase these connections will be replaced by the RTL connections. Thus all the low power cells are connected as simulated which is also given in the CPF file [1].

2.5.3. Structural Checking:

In the low power design the formal verification involves logical check as well as the verification based on low power intent.

The logical checking involves the checking with respect to the logical correctness of the design where as the low power verification is about making sure that the circuit is electrically correct from a low power point of view. This will check if the retention and the isolation are with respect to the one given in the CPF file and they are complete.

The logical checking should be capable of supporting the low power intent since the low power intent has added complexity due to the addition of isolation and the state retention cells. As we know that the isolation or retention cells are not present in the RTL and it is from the CPF, the tools should check if they have been inserted correctly and that if the generated net list is logically correct with respect to the RTL [15].

2.5.4. CPF Basic Definitions:

Power Domains:

Creation of power domains is the basic part in defining a CPF file. A power domain can fall in any of the following three types naming, unswitched domain, internal switchable domain, externally switchable domain [25].

Unswitched Domain:

When a domain created is not required to be powered down by any controls in the scope in which the domain is created then it is said to be unswitched domain. The shutoff condition will not be specified for this type of domains [25].

Internal Switchable Domain:

This power domain can be powered down. A power switch network will be used in this case to get the power supply from another domain. The shutoff condition will be mentioned in the CPF so as to power down. The power domain from which it gets the power will also have to be mentioned [25].

External Switchable Domain:

This type of domains can be powered down by external switches which lie outside the chip. However, the external switch will have to be controlled by the signals on the chip itself. Here the condition which will cause the shut off will have to be mentioned together with a command saying that it will be external. The above mentioned power domains can be either a base/derived domain, primary/secondary domain [25].

Base and Derived Power Domains:

A particular domain which can be any of the above three is said to be a base power domain of another power domain if the primary power and ground nets of this power domain provide the power supply through some switch network [25].

A standard cell instance can be created from a base power domain from which its primary power supply and ground nets can be derived to provide the power supply to the power pin and the ground pin. This is said to be a derived domain [25].

Primary and Secondary Power Domains:

A power domain is said to be a primary domain of a standard cell instance if the primary power as well as the primary ground nets provide the same to the cell.
A power domain is said to be secondary domain of a low power instance if the primary power as well as the primary ground nets provide the power supply to the secondary power or secondary ground instance [25].

Power Modes:

During a steady state of a design certain power domains are to be switched off and certain power domains are to be switched on which is called as power mode. A nominal condition is the one which specifies the voltages of all the power supplies given to the power domains. The power supply voltage and the ground voltage and even some times the PMOS, NMOS bias voltages will be specified in a nominal condition. Generally there can be three states, ON state, Standby state and OFF state. A power mode will be defined by specifying the above mentioned nominal condition for each mode [25].

2.5.5. Low Power Implementation using CPF and EDA:

Low power implementation done using CPF and Electronic Design Automation tools consists of various steps in which the Gate level optimization, Multiple V_{th} optimization, Multiple V_{dd} optimization, power Shut off optimization and, dynamic voltage and frequency scaling optimization play an important role. Having seen the basic concepts, now let us explore how these can be implemented also using the CPF [7].
Here a brief explanation on how the CPF files can be used to implement the low power intent will be explained. There are certain features which will be taken care by the EDA and there are certain features which can be added through a CPF file. The CPF commands are explained wherever applicable [15].

Gate Level Optimization:

The gate level optimization is considered to be the most basic in the implementation stage however the others are given importance in the synthesis stage. Even though clock gating is introduced in the synthesis stage, the designer will know the accurate placement of the cells, their routing distances and the physical distance only in the implementation stage. It is because of this, resizing, restructuring and pin swapping can be done more accurately in order to save power. Clock gating violations are due to the distance factor, i.e. when the clock gating cell is far away from the leaf cell. In the physical implementation stage the clock gating cells are just moved physically closer to the leaf cells. The EDA tools used these days are smart enough to clone the clock gating

elements properly so as to avoid the timing violations since they know the distance between both the cells. Thus the best way to deal with the clock gating is to de-clone during the synthesis and then clone selectively during the implementation stage. Again, this can be automated [2].

Multiple V_{th} values:

As mentioned, the usage of multiple V_{th} has become common when it comes to power intent. This needs a proper EDA tool which supports different V_{th} voltage libraries with the same cell functionality. When the value of V_{th} is high, it means that the cell is of low power and low performance and on the other hand if the V_{th} is low the cell power is high and so is the performance. Generally when the V_{th} is high the designer can save the power by up to eighty percent with a small increase in time delay. The EDA tool generally takes care of calculating the timing analysis and the cells can be easily interchanged during the layout stage [1] [12] [14].

Multiple Power Domains:

This step is carried out in the initial stage of floor planning. There can be different power domains created which will have their own set of libraries according to the voltage domain [8] [14].

Implementation using CPF:
The following CFP File description is used for the creation of multiple power domains [1].

The CPF Representation:

```
# Set the top domain
set_design TOP

# Set the default domain
create_power_domain \
–name Top_Domain –default

# Set Domain_A
create_power_domain \
–name Domain_A \
–instances {uX uZ}

# Define Domain_B – PSO when pso is low
create_power_domain
–name Domain_B\
–instances {uY} \
```

The lines which start with '#' are the comments. The first command *set_design* is used to specify the top module. The command *create_power_domain* creates the power domain with a name given in the name command and it will also have the information about the pins in the command *instances*. The shut off condition can also be provided in the CPF. So here, the top module name is Top_Domain and the other two domains are Domain_A

and Domain_B. the instances X and Z belongs to Domain_A and instance Y belongs to Domain_B. This structure can be represented as shown below.

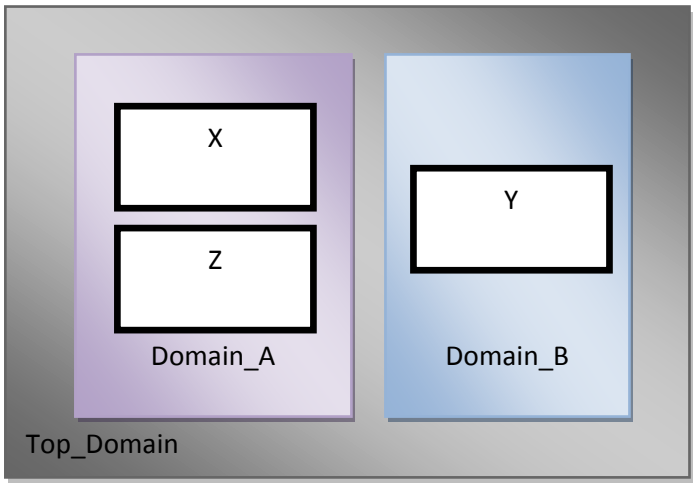


Figure 2.2 Multiple Power Domains obtained using the CPF File

Placement and optimization:

After creating the power domains, they have to be placed properly. Here the tools which can be used with the CPF will know the boundaries of each power domain defined. This is because of the reason that none of the logic should be moved from one domain to the other. Most of the EDA tools will know to use the appropriate timing libraries.

Level shifting:

All the signals crossing the power domains will have to be attached to a Level shifter. The Level shifting from higher voltage to lower voltage is generally optional. However the vice-versa is mandatory. This has become an automated task by using the CPF [7].

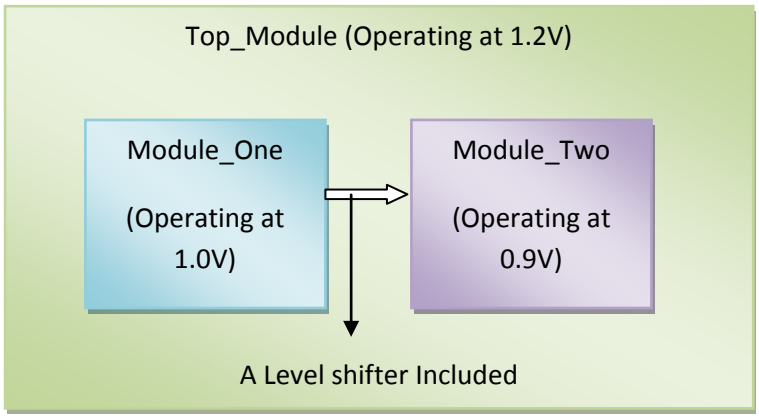


Figure 2.3 A Multiple supply Voltage with a Level Shifter

Implementation of Level Shifters using CPF:

The Level Shifter is implemented using the create_level_shifter_rule command. The name of the Level shifter the start and destinations must be provided [1].

The CPF Representation:

```
# Define Level-Shifters

create_level_shifter_rule
–name LS1 \
–to {Domain_B}
–from {Domain_A}

create_level_shifter_rule
–name LS2 \
–to {Domain_A}
–from {Domain_B}
```

Power Shut Off (PSO):

The Power Shut Off, also called as power gating is a low power design technique in which a certain part of the chip or a particular chip is shut off for saving power. This is one of most highly used and complex techniques used in the low power design. The implementation of the PSO can be done either as on-chip power shutoff or off-chip power shutoff. The implementation in which the power switch for the PSO is made within the SoC is called as on-chip PSO. The implementation in which the power switches are made external to the SoC is called as off-chip PSO. Power Shut Off can also be classified as,

- Course Grained PSO
- Fine Grained PSO

When an entire chip is shutdown by using a single signal, it is called as course grained PSO. This can save power leakage during stand by or sleep mode. When power is shut down to a block or certain portions of a chip which allows the operation of other blocks, it is called fine grained PSO.

Power domain will be defined in the CPF file and it is then implemented in such a way that the macros in the PSO reside in the correct physical area of the power domain and the macros which are not there in the CPF file reside outside the physical area. This is needed since the physical area is the one which decides if a macro is powered by a PSO or not. The power switch thus created can be inserted as a column or in a ring fashion.

The CPF enables tools that will automatically insert the power domain. More power switches are required if the power domain is having more logic and area. The number of power switches should be optimal so that not too much of area is wasted or not too much of rush current is produced during wakeup [10].

Handling Rush Current:

If the number of power switches is less it creates a rush current which happens during wake up. This can be reduced by the following ways [1].

- This can be reduced by the introduction of a delay. This delay can be introduced between when the power switch turns on. This will increase the turn on time of the PSO domain and the rush current can be reduced.
- Another way of reducing the rush current is by arranging the power switches as a group and turning them on and off group wise. In this case, instead of a single power switch, only the first or last group will get the rush current.
- There is also another way in which the switches have multiple enable pins. The smaller switch will be turned on first so that it gets almost 95% of the voltage.

Adding Isolation cells:

The isolation cells can generally be inserted by the tool during the synthesis provided the tool is capable of handling CPF. The isolation cells must be placed near the PSO. While inserting the isolation cells, proper care should be taken so that it is not placed in a wrong power domain. Care should be taken that they are connected to always ON power supply. The following is the CPF for the creation of Isolation Cells [1].

The CPF Representation:

 # It is for the output of Domain_B
 # isolated high on rising edge

```
create_isolation_rule \
–name namex \
–from Domain_B \
–isolation_condition {“Condition are given here”} \
–isolation_output high \
–pins “pin names”
```

The command create_isolation_rule is used to create the rules for the isolation cells. Here the name, the start location, the condition and the output are given in the CPF file.

Adding State Retention:

For the creation of state retention register, the regular registers will be transformed into state retention registers. This will take place during synthesis. The EDA or the designer should take care so that the extra space is allocated to place the additional power routing since both the switchable power supply and always-ON power supply is used for adding a state retention register. The state retention can be added using the CPF file and it is given as shown below with the *create_state_retention_rule* command [1].

The CPF Representation:

```
# It is for storing for falling edge
# and restoring on raising edge

create_state_retention_rule \
–name namex \
–restore_edge {“signal_name”} \
-instances “list”
```

Adding Always ON Buffer:

The addition of PSO domain requires a buffer so that it can store few nets in this domain stays ON all the times. The voltage supplies will be through row and it gives continuous power for the regular buffers and a secondary power pin is used to keep the values in the Always ON buffers. This can also be handled during the physical implementation [1].

Adding Dynamic Voltage and Frequency Scaling (DVFS):

The basic difference between the DVFS and the multiple V_{dd} is that in VDFS, any particular power domain will be able to operate at various modes. These modes will have their own supply voltages and frequencies. By using an advanced EDA tool the combinations of operating voltage and operating frequency can be optimized in parallel by totally automating the process. The power benefits achieved here are way far better than the traditional optimization. For e.g. a system can be designed in such a way that in active mode all the blocks operate in 125MHz and 1.08V and in the slow mode they operate at 66Mhz and 0.9V.

The power modes representing the exclusive voltages can be represented using a CPF file and this will be used in supporting Dynamic Voltage Scaling and Dynamic Frequency Scaling. The CPF file will be as shown below [14] [1].

The CPF Representation:

```
# Top_Domain is always high
# Domain_A is medium or low
# Domain_B is medium or low

create_nominal_condition –name High_Vg \
–voltage 1.2
create_nominal_condition –name Medium_Vg \
–voltage 1.0
create_nominal_condition –name Low_Vg \
–voltage 0.8
create_nominal_condition –name Zero_Vg \
–voltage 0

# Domain_A is Medium and Domain_B is Medium

create_power_mode –name Mode_One \
```

–domain_conditions {Top_Domain@ High_Vg \
Domain_A@Medium_Vg *Domain_B*@Medium_Vg}

 # Domain_A is Low and Domain_B is Low

create_power_mode –name Mode_Two \
–domain_conditions {Top_Domain@ High_Vg \
Domain_A@Low_Vg *Domain_B*@Low_Vg}

 # Domain_A is off and Domain_B is Low

create_power_mode –name Mode_Three \
–domain_conditions {Top_Domain@ High_Vg \
Domain_A@Zero_Vg *Domain_B*@Low_Vg}

The command *create_power_mode* is used to create the power modes and create_nominal_condition is used to *create_nominal_condition* is used to create different voltage levels. There can be different mode created which depends on the various nominal conditions selected.

The Ethernet SoC in this project uses CPF files for every module and it is then finally integrated. The study of CPF files present in the SoC helps to understand its power architecture better. So a proper understanding of CPF is required. Simvision, which is used in this project, helps us to view the power domain information and the power mode information as shown below. We find the CPU is in nominal condition and the voltage given according to the CPF file is 1.2V.

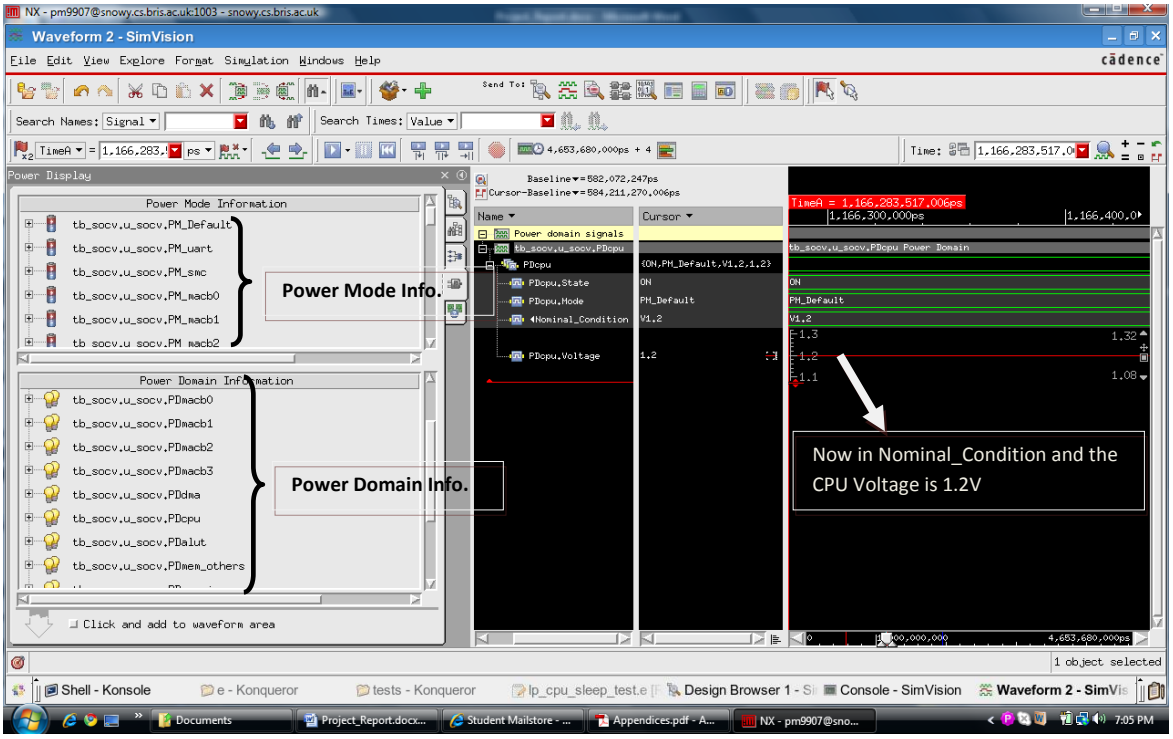


Figure 2.4 Simvision showing the CPF information after the test run.

Chapter 3

Project Design and Concepts

3.1. Introduction:

In this project, the *e* verification language is used for writing all the test cases and Specman is used as a tool for executing all our test cases. It is necessary that we go through how *e* verification language works and how it can be used with Specman. We have also used the latest concept in the verification industry, Open Verification Methodology. This chapter will take us through how this concept can be used in our project, implementing the hardware software co-verification. We will finally look into the hardware Device Under Test and an example program showing how the required data can be obtained for analysis.

3.2. The *e* Verification Language and Specman:

This section takes us through the working of *e* language with Specman Elite.

3.2.1 The e Language Introduction:

Initially, both Device Under Test (DUT) as well as the test environment were designed in VHDL or Verilog. In a typical RTL test environment, the test benches were written in RTL which then writes data into the DUT or read the data from the DUT. This approach has the following drawbacks,

- Writing test cases became time consuming and it became very tedious since it was difficult to read and maintain.
- It was hard to test the corner cases and testing the robustness was difficult.
- Creating and maintaining the test environment became difficult since they had little shared code.

This paved way for an object oriented verification language with random generation, and a test bench tool which could reduce the effort for creating a test environment. This complete verification automation system will help increase the overall productivity of the verification [16] [17] [18].

A Complete Verification Automation System:

A complete verification automation system will help us with stimuli generation, driving stimulus, collecting the output, data checking and coverage measurement. Here we concentrate on the *e* verification language and the Specman Elite tool which forms a

complete verification automation system since it will be used throughout this project [16] [17] [18].

Data Object in *e*:

A *struct* represents a data object in *e* verification language. This represents one stimulus. The Specman Elite contains a random stimulus generator which is capable of generating a random value for a stimulus which will depend on the constraints given by the verification engineer [16].

Stimuli Generation:

The generation of stimuli is automated in *e* Language. The verification engineers will be able to provide the constraints which are derived from the specifications. The constraints will also depend on the test cases to define the test case. When a DUT is presented with an illegally constructed data it will discard it by not involving the design [16].

Driving the Stimulus:

The *e* Language allows us to create drivers in an object oriented way. These drivers help us to drive the test vectors created during the stimuli generation into the DUT. The driver should be able to provide an interface with the simulator, obey the rules to the provided input protocol for the DUT, convert the abstract data structure to bits and drive them to the DUT. A *unit* represents a driver object in *e*. However, it does not move through the verification system unlike the *struct* does [17] [18].

Collecting output:

The output produced from the DUT after the injection of the stimulus should be collected and checked. Receivers can be created in *e* language for this purpose and this involves in creation of an interface with the simulator which should be able to recognize the output protocol for DUT and receive the bits. It should also covert the received bits into an abstract data structure for further checking. *unit* is used to collect the output [16] [17].

Data Checking:

The data received from the DUT should be checked which can be of two types namely data value check and the data timing check. In value check, the bits produced from the DUT are compared against the expected data. In timing check, the system is checked for any protocol violation [16] [17].

Measuring Coverage:

The coverage measurement is done so as to find if the test plan goals have been met or not. It also enables us to, check if all the legal values of variables have been covered,

check for all the legal transitions of a state machine, and finally check the combinations of basic coverage under different states [16].

3.2.2. Interaction between the Specman and Simulator:

Simulator and Specman are two different processes which talk to each other by means of an interface and run concurrently and synchronize with each other during the simulation stage. There is a special file called the *stub* file through which both simulator and Specman communicate to each other. The verification environment contains the components required for verification in the form of *e* language. These files will be compiled and simulated. The simulator on the other hand compiles and simulates all the files in VHDL or Verilog. In this project, all the files are in Verilog.

During invocation, the Specman and Simulator are invoked at the same time. The Specman then gains control and the Simulator will stop the simulation at the time zero. The simulation will be then started by explicitly giving control to the Simulator. The control will be passed back to the Specman according to *e* code set by the verification engineer. After doing the necessary computations the Specman then passes the control to Simulator. This happens till a procedure in the Specman is called which completes the simulation [16] [17] [18].

3.3 Open Verification Methodology:

This section gives us an introduction to the Open Verification Methodology and the various parts and construction of Universal Verification Components which are extensively used in this project.

3.3.1. Introduction:

Cadence Design Systems and Mentor Graphics jointly introduced the concept of Open Verification Methodology, popularly known as OVM. The OVM is an open source verification methodology which has been architected for multiple languages. The OVM libraries provide methodologies to match the test benches which are regardless of the implementation language. The concept of OVM provides a standard transaction level communication channel which can be used to exchange information between different languages [19].

The Open Verification Methodology is used for creating a verification environment which maximizes the reusability of the code so that it can be used for different projects which can involve mixed languages. The methodology of OVM shows us how to achieve code reusability by using well known methods for designing, coding and packaging [19]. We have used many UVCs in this project and the important ones will be discussed in the section 3.6. Also, we will discuss an important UVC that we use in this project, The Ethernet UVC on section 3.3.3. Since the focus of this project is on achieving low power, we will be concentrating only on the basics of OVM and UVCs in the sections below.

The OVM Test Benches:

An OVM Verification environment is made up of reusable Universal Verification components called UVCs (Universal Verification Components). OVM supports UVCs to be written in *e* language, System Verilog or the combination of both. OVM also supports SystemC. OVM provide us a standard for the planning, configuration and standard debug message formats [19].

3.3.2. Universal Verification Components for OVM:

UVC forms the basis for an OVM verification environment. It is a ready-to-use, completely encapsulated verification environment which is configurable. It is generally used for an interface protocol or sometimes for a full system. UVCs can be used with *e*, System Verilog, VHDL or SystemC. An UVC will contain a complete set of basic elements for the purpose of simulation, coverage collection and checking for a specific protocol or a specific architecture [19].

Compared to the normal verification environment, an UVC can be used in more than one setting. Many UVCs can be combined to produce a new UVC provided there are no name clashes between them. Following the best practices (according to the OVM) will avoid any kind of conflicting situations. The concept allows us to use the UVCs as a plug and play component. Building a new verification environment should be easy and this new combined environment can also become a new UVC which in future can be plugged so as to create a bigger verification environment. Generally commercial UVCs come in an encrypted form so as to protect the company's intellectual property. The UVCs can also be extended to add more functionality which is generally done in a separate file [19].

Components and Types of UVCs:

The UVCs can be categorized into three types according to their architecture and the OVM guidelines. The basic building block of an OVM environment is the interface UVC.
The Module UVCs are the ones used for integrating the interface UVCs with other verification components which are required for verification [19].

Interface UVCs:

Interface UVCs are the ones which interact directly with the DUT. They are made up of several components and they focus on a specific protocol such as TCP/IP, Ethernet etc. Generally an interface consists of the following components.

Agents: An UVC will have at least one Agent. An agent consists of a driver, sequencer and a bus monitor. UVCs may also have more than one agent which may have various operating modes. In active mode, the agent drives and monitors the interface. In passive mode, the agent only monitors the interface where as the driving aspects are turned off. There can also be protocol specific agents.

Drivers: A driver is the one which interacts with the DUT by driving, sampling the DUT signal and also it converts all the transactions into the format required by the DUT.

Sequencer: A sequencer is used to generate the stimulus data which will be then passed to the driver for further execution. A sequencer will be able to generate a single transaction or a sequence of transactions which will be used to form a more properly structured stimulus.

Bus Monitors: The tester and the other components have to understand the signal information. The bus monitor does this job. It extracts the signal information from the bus and then it translates them into events and/or status information. The bus monitor has a component called the checker which checks the protocol and the data and thus verifies that the DUT produces the right output. The bus monitor also does the coverage collection [19].

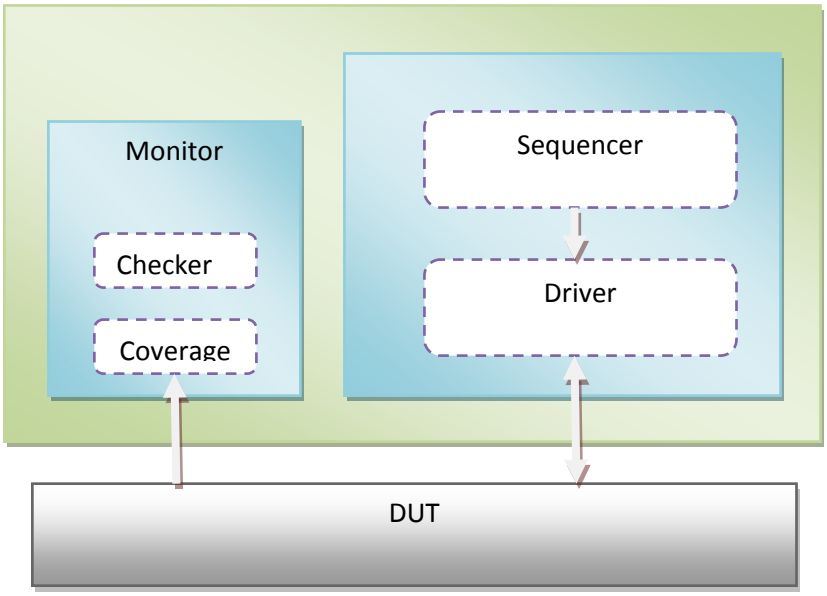


Figure 3.1 UVC Agent

The diagram above shows all the components of an UVC and shows how they interact with a device under test. The other types of UVC generally called the system UVC and monitor UVC are discussed below.

Module and System UVCs:

A system UVC will be used to do a system level verification. They will be using interface UVCs for connecting to the system interface and module UVCs which can be a sub-system of currently verified system. A system UVC generally does not have its own agents whereas it instantiates other UVCs and connects them so as to create a verification environment. A smaller system UVC is generally called as Module UVC.

A system UVC contains the following which is similar to interface UVCs.

Monitor: This collects all the information and most of them will be collected from the interface UVC's monitor.

Virtual Sequencers: They are connected to the sequencers of the interface UVCs and the module UVCs.

Scoreboard: Scoreboard is the one which receives the inputs from the interface UVCs. It is always a good practice to have as scoreboard in a system UVC [19].

Further exploration into UVCs and its creation rules will be out of scope of this project. In this project we also use the *e* UVCs, called as *e*VCs, ready to use and configurable. We now proceed with the Ethernet UVC in the next section.

3.3.3. The Ethernet UVC:

The Ethernet UVC that we use in this project is a ready-made and highly configurable *e* Verification Component (*e*VC). This UCV is capable of generating Ethernet packets and random data packets. It is also able to drive the generated packets according to the protocol which comes with a protocol checker for the DUT to make sure that the DUT follows the Ethernet protocol.

The monitor present here is completely passive and the Bus Functional Model (BFM) generates the data packets. Most of the passive activities of the UVC are done by the monitor and the active activities like the interactions with the DUT are done by the BFM. The monitor also does the protocol checks and is used in the coverage collection which is predefined in this UVC.

The Ethernet UVC comes with a scoreboard as explained in the previous section. This is used to check if the number and the order of the data that are collected from the DUT are matching with the expected result. This scoreboard functionality is optional. However, in this project we have enabled the scoreboard option so that we verify and collect the necessary information for a working System. We will not go deep into the construction of this Ethernet UVC as it is a commercially available component which can be used to verify any Ethernet Systems [23].

3.4. Hardware and Software Co-Verification:

This project involves the power analysis of the hardware as well as the software which brings us the need to study more on hardware and software co-verification. We will be using metric driven, constrained-random stimulus to verify the overall power [20]. In this section we will look into the various ports used for the interaction between hardware and the software. In general, this verification involves three major components,

The Device Under Test:

While performing the hardware software analysis the following sections of the DUT should be taken into consideration. The device hardware (developed in Verilog), the lower level system software (developed in C) and hardware software interface (developed in C).

The Software UVC:

Software UVC created will be a self contained verification component for a certain purpose which simulates the system software. It should allow higher level coordination with the hardware UVC to stress the hardware, by providing automatic checking to ensure the functionality of the DUT. The software UVC controls the flow of the system software by checking its mailbox. This mailbox is used to check for the function to call next when the system software starts running. When the function gets completed, the return value is stored in the mailbox for the verification environment to retrieve.

Connecting DUT and Software UVC:

The software UVC uses various ports to connect to the DUT. The following are the various ports used when it comes to co-verification [20].

Method Ports:

These ports are used to create connections between the UVCs and the system software. There are two types of method ports, outgoing method ports and incoming method ports. The outgoing ports are used by the UVCs so as to call the functions in the embedded software. The parameters of the outgoing method ports can be randomized so as to allow maximum investigation of the software. On the other hand, the incoming method ports are called by the system software like normal functions [20].

Monitor Ports:

Software functions need instrumentations so as perform checking and coverage. This functionality can be performed by using the monitor ports. Monitor ports are used to notify the verification environment when an internal software function is called; this in turn helps us to check the function's parameter or its return value. It is also used to monitor the internal local variables. Cadence software helps us to create the very first method port by creating wrapper functions. These wrapper functions can be simply used in the user source code. This will call the verification environment with the function's argument or the function's return value [20].

Simple Ports:

The simple ports are used to create a connection between the system software variables and the UVC. They are defined as bi-directional, so that they will be able to read and write software variables. The outgoing simple port allows the UCV to write to the software variables. An incoming simple port helps the UVC to read the software variable. They are generally used to check the functionality of the DUT and to report the status back to the UVC [20].

Event Ports:

When the value of a software environment is changed, the event port helps us to call the UVC. This triggering of the software UVC by the event port will be useful when the UVC sequence is to be maintained [20].

3.5. The Device Under Test:

In this section we will look into the detailed block diagram of the SoC and the Address Look-up Table.

3.5.1. The System Block Diagram:

The following is the device which has been taken for our hardware software verification for the maximum power utilization. It has been developed in Verilog [2].

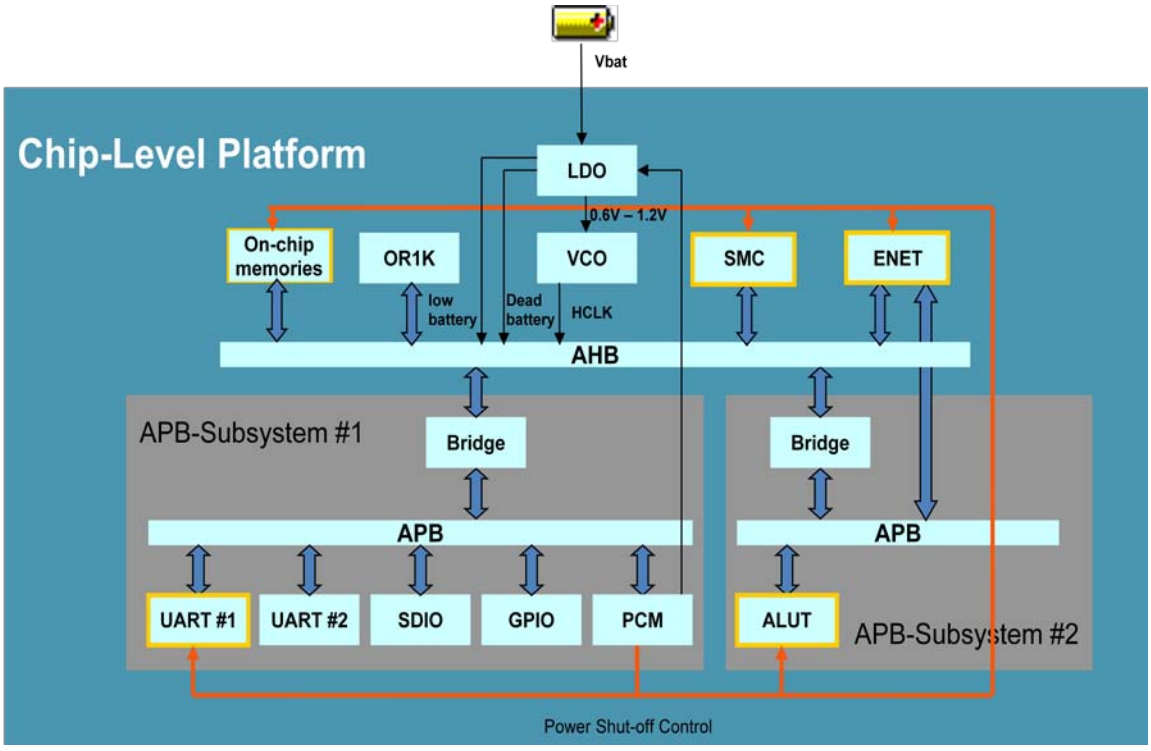


Figure 3.2 The System Block Diagram

As shown in the block diagram below, the test hardware platform uses the Advanced Microcontroller Architecture Buses (AMBA). The protocols used here are the Advanced High Performance Bus (AHP) and the Advanced Peripheral Bus (APB). The AHB is used connect the inbuilt Static Random Access Memory (SRAM), Static Memory Controller (SMC), OR1K processor, Ethernet and the VCO.

The other two major blocks are the APB connected Subsystem 1 and Subsystem 2. The Subsystem 1 has the Universal Asynchronous Synchronous Receiver Transmitter, General Purpose Input Output (GPIO) and Power Control Module (PCM). They are connected together by the APB. The Subsystem 2 has the Advanced Look-up Table (ALUT) which is of our interest. The ALUT will be storing the addresses of the ports and it will be used as a look-up table so as to increase the data transfer rate. The ALUT operation will be explained in detail in a different section.

Firmware Block Diagram:

The firmware of the verification block has the components which include device driver for power manager, device driver for the MACs, a system controller and the applications programs.

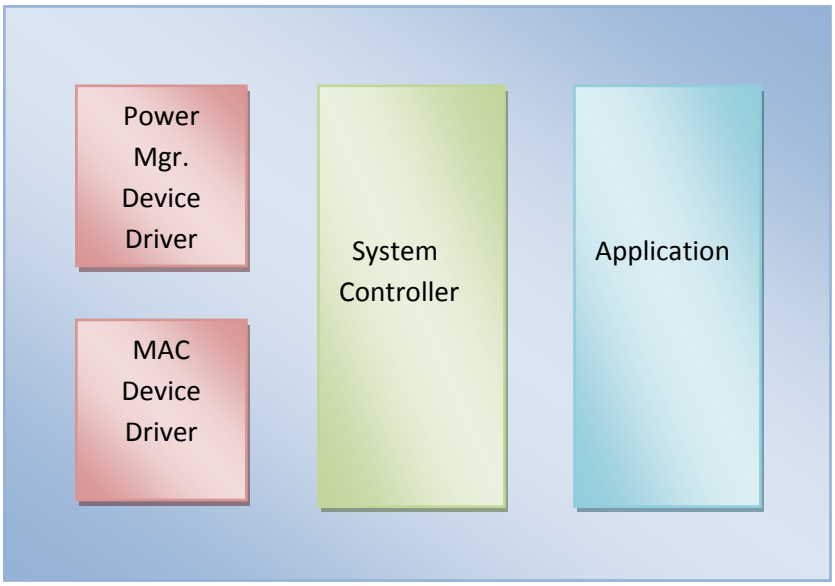


Figure 3.3 Block diagram of the Firmware, Cadence Verification Kit [2]

The system has got three major firmware blocks. (i)The device driver layer consists of the drivers for MAC and the power manager. (ii) The system controller provides the power management autonomously to the hardware. (iii) The final block is the Ethernet switch application layer which comes with high level power management state machines. While using the software version of the look-up table, the C programming language version of the look-up table will be loaded from the firmware [2].

3.5.2. The ALUT Block:

The ALUT block is in the second APB subsystem and it is connected to all the four Ethernet MACs. The look-up table uses the RAM and has the following data entries which include the address bits, port bits, last accessed bits, and a valid bit. The ALUT performs address check as well as the age check in which address checking is of our interest [21].

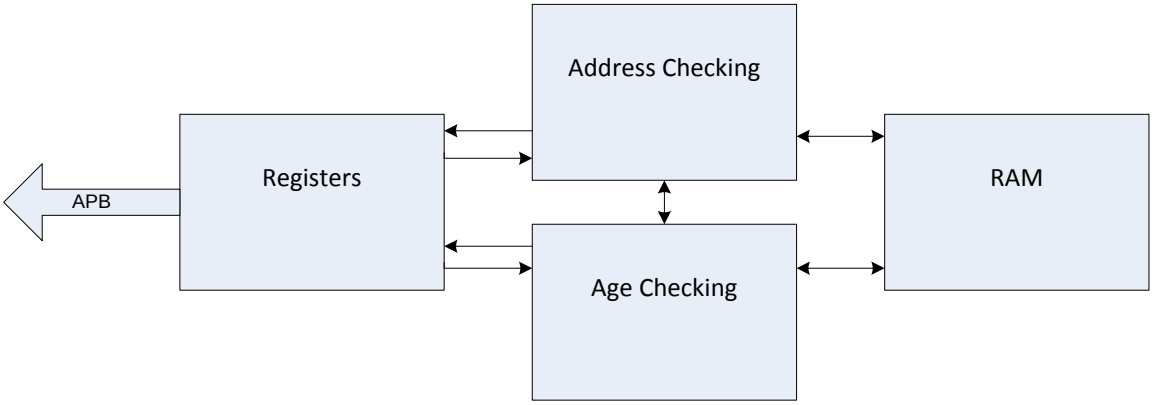


Figure 3.4 ALUT Block Diagram

ALUT Address checking functionality:

There is a register in which the current operation to be performed is mentioned whether it is address check or the age check, called the *command* register. This register should be issued with a check address command for using the ALUT.

There are 256 entries in the SRAM for the ALUT. The different fields in one 83 bit entry is as shown below,

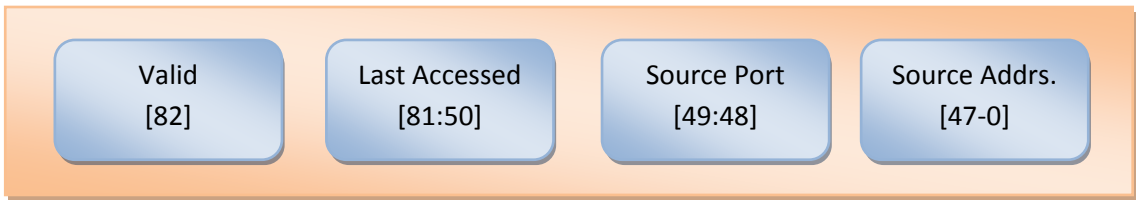


Figure 3.5 Look-up Table entries bitwise

The checking operation is initially done by comparing the input destination address and the stored MAC switch address. If there is a mismatch then the checking will be done for the input destination address against the values stored in the memory. The functionality uses a hash conversion and the 48 bit address will be changed to 8 bits.

A hash is generated corresponding to the current frame destination address and this is used to check the look-up table. If there is a match with the look-up table, the corresponding entry will be loaded into the current destination port. Else, the current destination port will be loaded with all the ports except the current source port. A hash

will also be generated for the source address and will be loaded into the look-up table. This is done by writing the address of the source frame, the current source port, time it was accessed and a valid bit as 1 [21].

This functionality of the Address Look-up Table is defined in RTL for the hardware version of the ALUT and is written in C for the software version of the ALUT. Having looked into the construction and the functionality of the ALUT, we move to the next section on how to create a verification environment for this DUT.

3.6. Verification Environment and the DUT:

We will be analyzing the SOC using this verification environment which comprises of a number of hardware interface UVCs, software UVC, one UVC for the hardware software interaction and a top level system environment. This also includes the DUT. The following sections will be used to explain the individual UVCs [22].

The Top level UVC: cdn_socv:

This top level verification environment is the one which is used to instantiate all other individual UVCs and extend them for our verification purpose [22].

The HW/SW interface UVC: cdn_hw_sw:

The purpose of this module is to interface the hardware and the software. This UVC consists of a virtual sequence driver, a monitor and a set of scoreboard. This module is used for two main purposes, the hardware software interface and the hardware software switch. This uses the basic concept explained in section 3.4.

The hardware software interface is used to transfer and check the data between the hardware interface and the SOC's internal memory. A scoreboard is instantiated for all the data transfer direction supported by the DUT. The following are the possible directions which include, data transfer from hardware interface to the memory in the DUT, data transfer from the memory in the DUT to the hardware interface, internal data transfer within the DUT memory. Data addition to the scoreboard and data checking from the scoreboard is enabled by a method port.

The hardware software switch is used to extend the filtering functionality implemented within the DUT by extending the Ethernet UVC [22].

The Software UVC: cdn_software:

This UVC contains a set of method calls which is used to activate the embedded processor (0R1K) through the generic software adapter. The hardware software UVC extends the cdn_software UVC so as to enable the sequences necessary to perform the co-verification [22].

The Ethernet UVC: vr_enet:

The Ethernet UVC is commercially available and it is used in this project by extending its sequence library so as to restrict the testing to basic Ethernet frames [22]. The Ethernet UVC has been explained in the section 3.3.3.

The Software:

The Software is written in C and it is compiled using the open RISC GNU tool chain. The compiled library is loaded into the DUT's memory and it runs on the OR1K processor. This software provides a C hardware driver for each peripheral in the DUT. The software UVC is used to control both the configuration methods and the control methods of the software.

The hardware can be configured by using the configuration methods. The configuration methods write to the registers in the peripheral so as to control the hardware. They also help in setting up the internal software state.

A transfer can be initiated by using the control methods. They write to the peripherals to do any transfers. There are two types of transfer. In the first type, the peripheral continue with the transfer without the software and then it interrupts the software after completion. In the other method, the software will copy or read the data into or from the FIFO. The peripheral will raise interrupt only when the FIFO needs any attention [22].

3.7. Benchmark Programs:

3.7.1 Introduction:

This section provides information on how the project has been designed and planned for executing the test cases. This section also gives an example test program with the information on how to run the test and collect the required data for further analysis.

3.7.2. Test Cases Flow Chart:

The flow chart explains how the tests are done. The tests are done on hardware ALUT and the software ALUT and the test cases will be similar. All the ports on the Ethernet MAC will be used for every test case. The test cases can be done using random seeds and mainly under two categories with ALUT empty and the ALUT full.

As mentioned above, the number of entries in the ALUT is 256. After completing the test cases the power information is collected in a TCF file through the *dumptcf* command in the Specman. This TCF file will be used for getting the power information. The detailed construction, procedure for executing the test cases, and collecting the outputs will be explained in the coming sections.

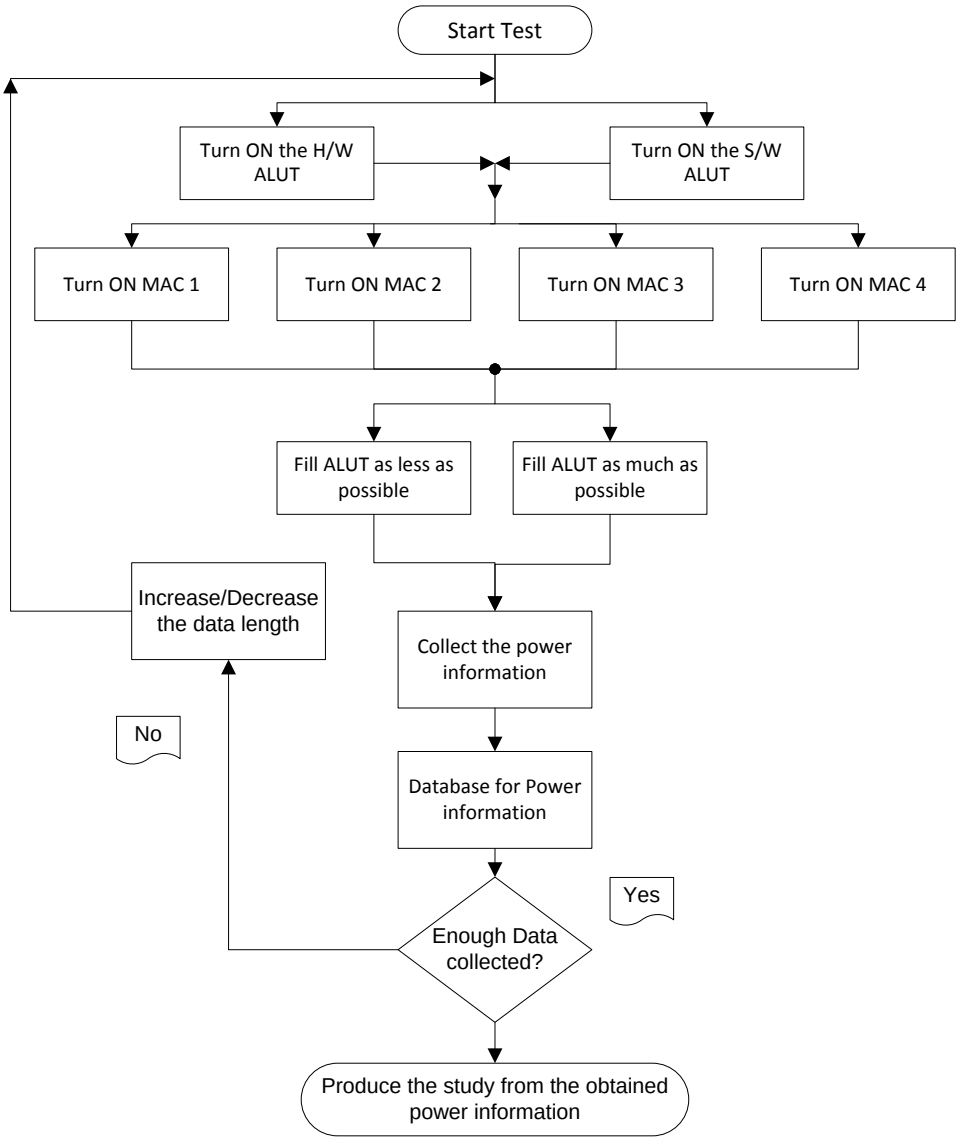


Figure 3.6 Test cases flow chart

The tests will be categorized under two main types.

i) Depending on the ALUT algorithm (hardware or software).

ii) Depending on the workload on ALUT (look-up with minimum or maximum entries).

The flow chart is self explanatory and it shows how the test flow for this project is planned and executed. The flow chart will be under each test case for a better understanding of the flow.

3.7.3. A Test Case snippet:

The test cases are written in *e* language and it uses the Ethernet UVC to create the Ethernet traffic. The data length, the number of frames and the maximum number of entries in ALUT can be changed in the Test cases. The following is a snippet of a test case. For test cases, please refer the information provided in the Appendices section.

```
...
...
extend has_switch has_gmii hw_sw_sequence_driver
{
        keep soft max_stations_per_port == 4;
};
extend MAIN cdn_hw_sw::hw_sw_sequence {
!hw_sw_switch : SWITCH hw_sw_sequence;
body() @driver.clock is only {
        do hw_sw_switch keeping {
                .count == 15;
                .max_length == 120;
                .quiesce_delay == 120000;
                };
        };
...
...
```

Figure 3.7 A section of a test case

The above test case imports the top level verification environment UVC which belong to the cdn_socv UVC. The purpose of including this is to enable the switching functionality of the DUT.

The system has four Ethernet ports and each of them is attached to a segment for the network. These segments can have multiple Ethernet ports connected to them which are called as stations. The number of stations created can be controlled in the test cases. For this we have to extend hardware software sequence driver which has a variable to control the maximum number of stations per port. Hardware software sequence driver has been extended to increase the number of stations that can be used in a port. Here, it has been increased to 4. This means that the number of entries in the ALUT will be to a maximum of 4X4 = 16 entries.

The number of frames has been set to five and the maximum length has been set to 120. Now let us see how this test case can be simple loaded using *irun* command after providing information about the locations of the directories in a script file.

3.7.4. Executing the Test Cases:

All the test cases can be simple run by using *irun* command in the command window. The following shows how the example test case shown above can be executed.

irun_or1k_gnu.csh -seed 1000 -mode interactive_debug -lp -test Example_Testcase.e

The *irun* command format shown in the table is used for executing most of the test cases. The *irun_or1k_gnu.csh* is a script file in which we have included the details of the all the file locations which includes the design topology, HDL files, topology defines, CPF constraints, verification environment, work directory etc. The one after the keyword *seed* selects the seed which can be any number or the keyword *random* which selects a random seed. The tests are executed in an interactive debug mode which is set by using the command *interactive_debug.* The *–lp* enables the low power testing. Finally the test case name is given at the end. Please note that the *irun* script file is placed in the work directory. If it is present in a different location the complete path has be present before the file name. Executing this command will bring up Simvision, Simulator, Specman, Waveform window and the Design Browser. The test case starts running after issuing the *run* command in the Simulator tab.

3.7.5. Accessing the ALUT information:

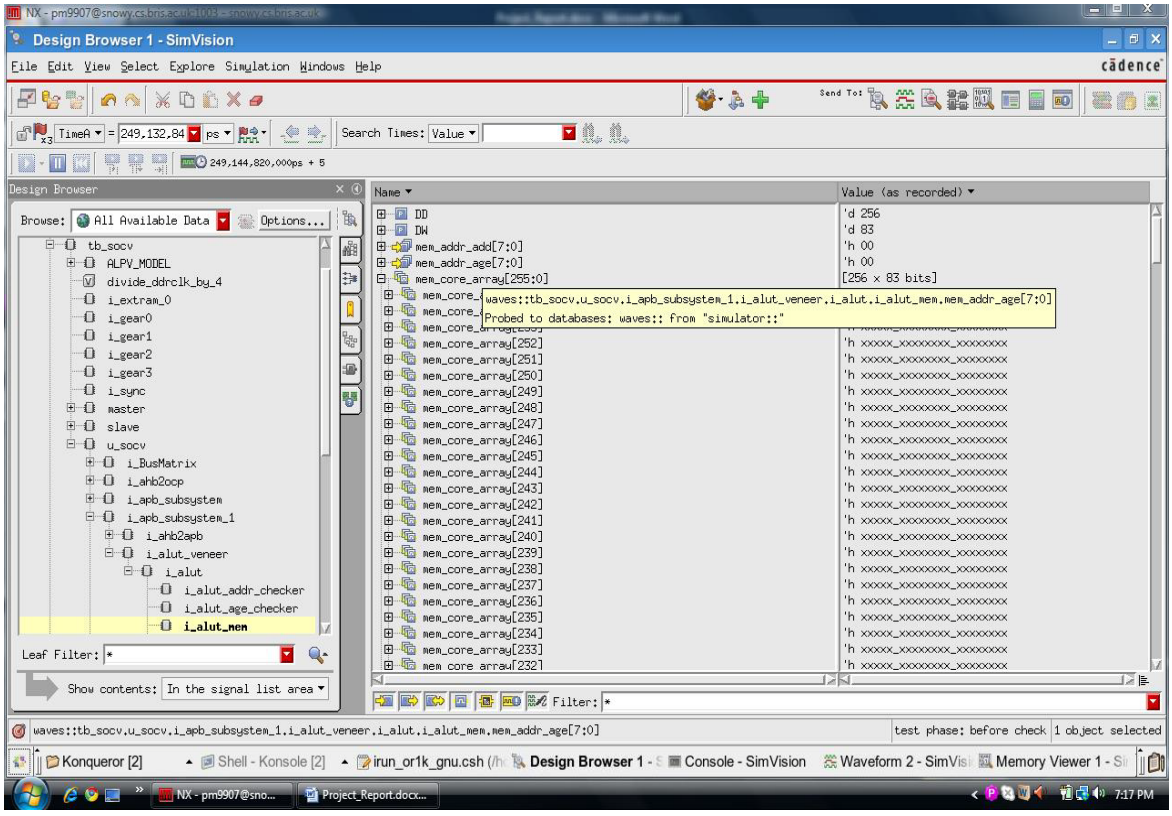


Figure 3.8 Design Browser showing the hierarchy of the design

The ALUT is a 2-dimension array named *mem_core_array*, present in the ALUT module to hold 256 entries of 83 bit information which is explained in the ALUT block section. This can be tracked down using the Design Browser which opens along with the Simvision. The Design Browser shows the memory location and its hierarchy. The snapshot shown as figure 3.6 explains the hierarchy clearly.

tb_socv -> u_socv -> i_apb_subsystem_1 - > i_alut_veneer -> i_alut -> i_alut_mem.

The snapshot shows the individual array elements present in the ALUT. This lets us to explore all the variables and their hierarchy. However there is a drawback depending on the Design Browser directly due to the fact that we will not be able to access the 2-D array elements directly. This brings us to the usage of probes for this particular array. When a variable is probed, all the changes in its values will be recorded into a database and these values can be taken for viewing later. Also, this allows the waveform window to show the values of the ALUT. This can also be accessed using the Memory viewer provided the variable has been probed issuing the command below.

probe -create tb_socv.u_socv.i_apb_subsystem_1.i_alut_veneer.i_alut.i_alut_mem –waveform

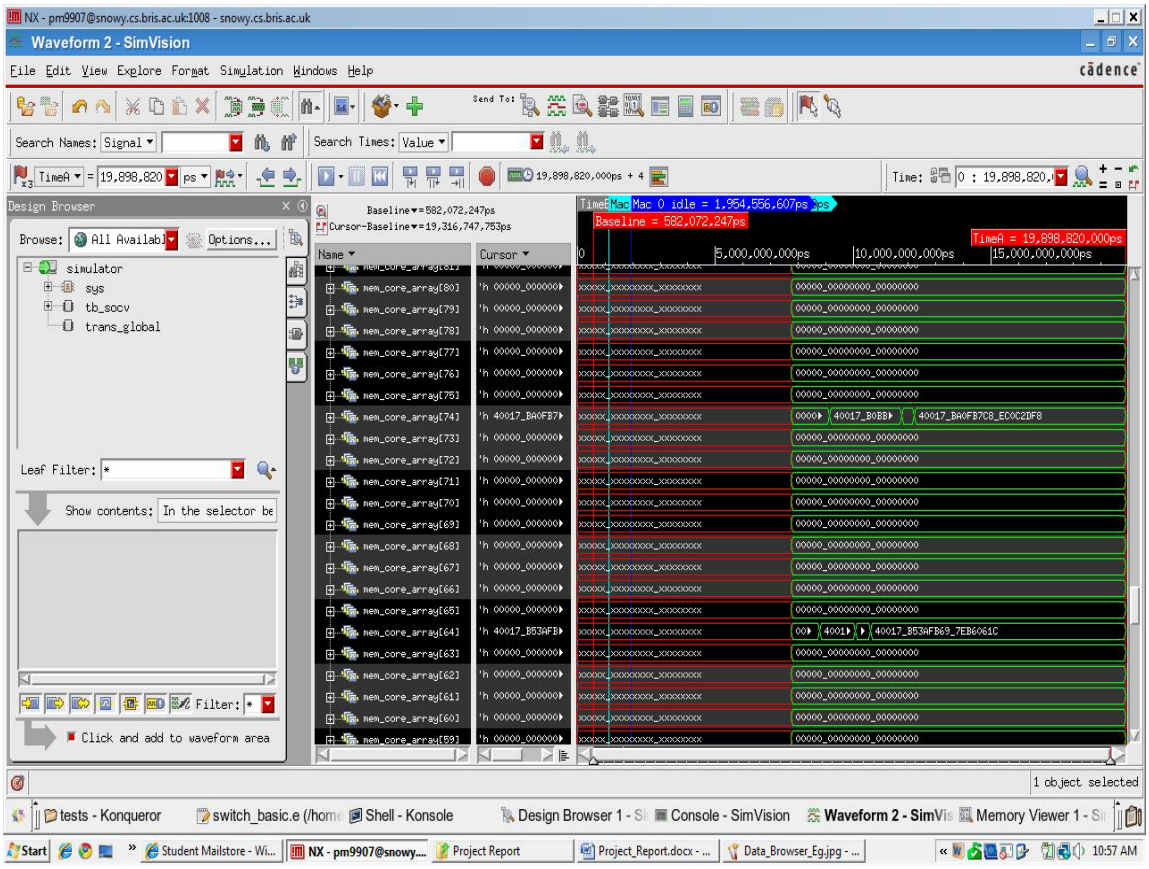


Figure 3.9 Look-up tables getting filled as seen by the waveform window

When a *run* command is issued, the execution starts with initialization and then proceeds with the data transfer. This data transfer can be observed in the log information in the Simulator and can also be observed through the waveform window.

It can be seen that only less than 16 entries have had happened in the ALUT memory. This is due to the restriction given in the Test case through the variable *max_stations_per_port*. The ALUT is designed in such a way that when it wants a new entry to be done and there is less number of locations to write, it simply reuses then previous locations. All these can be observed in the snapshot.

To ease the viewing of the ALUT, it is possible to check them through the Memory Viewer. It is to be noted that these possible viewings can happen only when a probe is created. This type of view helps us to find the number of ALUT entries very easily compared to the other type of views. A snapshot of the Memory Viewer for our sample test case is as shown below which clearly shows the current status of the ALUT with its values displayed.

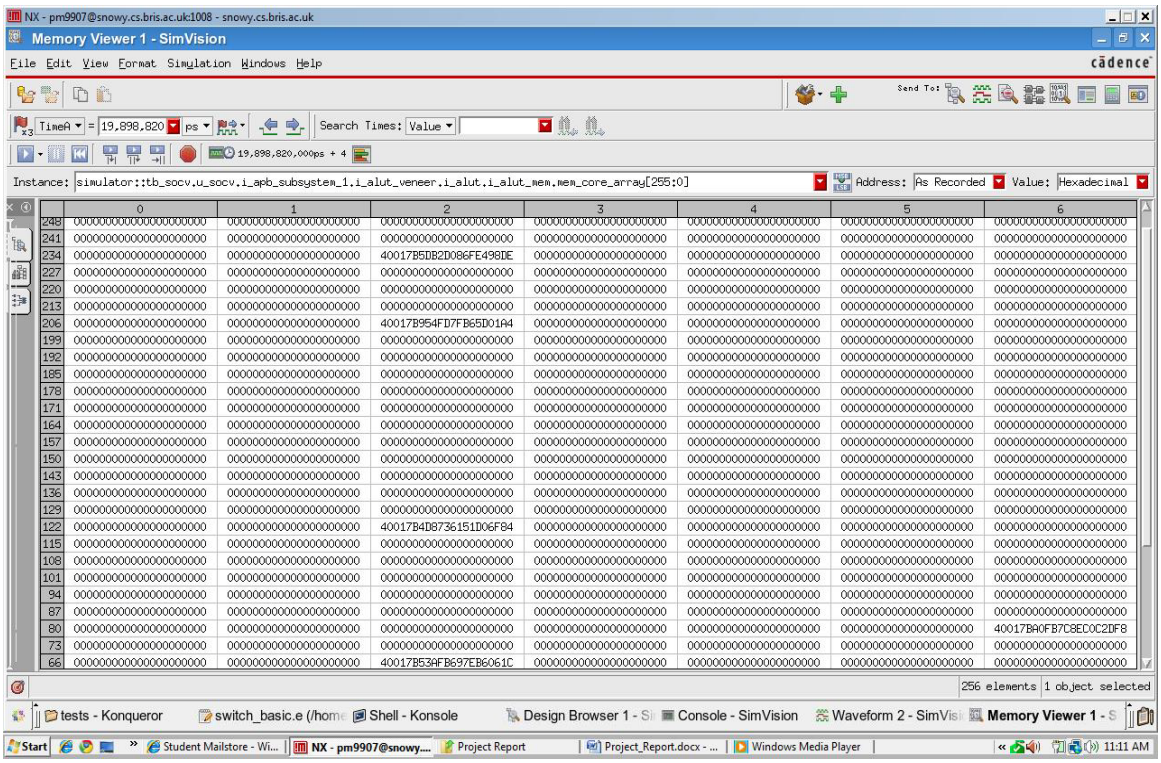


Figure 3.10 Memory window showing the look-up table live

3.7.6. Enabling the Software ALUT:

The software ALUT can be enabled by updating the firmware. The variable present in the *system.c* file will be changed. The variable *EnableSwAlut* will be set to TRUE so that the software look-up table is enabled. The system will not consider the hardware block for any look ups when this variable is set to TRUE.

The functionality for the look is performed by the file called *AlutDriver* in which the software is written in C programming language. This performs the same function that the hardware performs.

3.8. Expected Results:

The above test case should give us the number of ALUTs filled for a traffic of 15 frames and the toggle information and the number of high states for a particular point of time as well as for the whole run time. Now, what information can we get from the toggle file and the high states? They provide us the complete details on the number of high states during the test. This will give us the information on how many how many regions have gone active which will consume power.

This data can be taken by using the Dynamic Power Analysis (DPA) capability of the Cadence Palladium Emulator. The inputs would be the Toggle Count Format (TCF) file and the information from the ports. The results will be on a database file representing the number of high states and the number of toggle for each and every point of time. This information will show the power consumption of the hardware and the software embedded ALUT. As we know that in this example file we are considering the hardware ALUT, the next chapter will take us through the steps on how to get these data and how the analysis can be done.

We expect the hardware version of the look-up table to consume more power when compared to the software version. Also, the hardware version is expected to be faster compared to the software version. We will come to know the results only after the successful runs of all the test cases.

Chapter 4

Analysis of Work

4.1. Introduction:

In this chapter we will look into how each test is performed and how the data are collected. Then, we will move into the analysis part for all the results obtained. The analysis will be done on performance wise and as well as power wise.

4.2. Collecting Toggle Count Format File:

For performing the power analysis, we need to generate the Toggle Count Format (TCF) file after the simulation. The TCF is a standard developed by Cadence to describe the activities happening during switching. For accurate power analysis of a design, this switching activity information file is required.

The switching activity information file called as TCF, contains the toggle information indicating how often a pin or a net switch switches between its logic 1 and logic 0. It also provides us the probability of the pin or net switch being in logic state 1 which is given by the total time in logic state 1 divided by the total time during the TCF dump.

The TCF file can be extracted by using the command *dumptcf* in the simulator. In our example test file the dump command is given in the following way.

ncsim> dumptcf –output Example_Testcase.tcf

The following is a part of the TCF file taken from Simvision which will be used for further analysis. The following is for the *alut_mem* instance.

```
instance("i_alut_mem")
        {
                pin() {
                        "pclk" : "0.500000 1989882";
                        "mem_addr_add[7]" : "0.443796 47";
                        …
                        "mem_addr_add[0]" : "0.064662 36";
                        "mem_write_add" : "0.000015 30";
                        "mem_write_data_add[82]" : "1.000000 0";
                        "mem_write_data_add[81]" : "0.0 0";
                        …
                        "mem_write_data_add[2]" : "0.149748 6";
                        "mem_write_data_add[1]" : "0.070753 10";
                        "mem_write_data_add[0]" : "0.0 0";
```

```
                    "mem_addr_age[7]" : "0.000129 2";
                    "mem_addr_age[6]" : "0.000129 4";
                    …
                    "mem_addr_age[0]" : "0.000129 256";
                    "mem_write_age" : "0.000257 2";
                    "mem_write_data_age[82]" : "0.0 0";
                    "mem_write_data_age[81]" : "0.0 0";
                    …
                    "mem_write_data_age[1]" : "0.0 0";
                    "mem_write_data_age[0]" : "0.0 0";
            }
        }
```

Figure 4.1 A part of the generated TCF file

4.3. Collecting information from ports:

The information on the destination address port (*d_addr[47:0]*), source address port (*s_addr[47:0]*) and the port number (*s_port[1:0]*) are collected and these data are stored in a comma separated values format file by using the export option in Simvision. These values are collected for every lookups and they are present under the instance *i_alut_addr_checker*.

The reason for extracting and exporting the values of only *d_addr*, *s_addr* and the *s_port* is that they are used in the hash conversion and these three data are the ones that decide in writing a value to the ALUT. The *alut_addr_checker.v* performs the look-up based on these values.

The following is for our example file exported through Simvision DataViewer.

SimTime	d_addr[47:0]	s_addr[47:0]	s_port[1:0]
0	0	0	0
8433230000	0	0000D06FF186	0
8433310000	0	CA11D06FF186	0
8433410000	0000ECE83E72	CA11D06FF186	0
8433530000	7E2DECE83E72	CA11D06FF186	0
8433710000	7E2DECE83E72	CA11D06FF186	1
8787950000	7E2DECE83E72	CA117EB6061C	1
8788030000	7E2DECE83E72	FB697EB6061C	1
8788130000	7E2DEC0C2DF8	FB697EB6061C	1
8788250000	B7C8EC0C2DF8	FB697EB6061C	1
8788430000	B7C8EC0C2DF8	FB697EB6061C	2
9145030000	B7C8EC0C2DF8	FB69EC0C2DF8	2

9145110000	B7C8EC0C2DF8	B7C8EC0C2DF8	2
9145210000	B7C86FE498DE	B7C8EC0C2DF8	2
9145330000	2D086FE498DE	B7C8EC0C2DF8	2
9145510000	2D086FE498DE	B7C8EC0C2DF8	3
9508490000	2D086FE498DE	B7C8B65D01A4	3
9508570000	2D086FE498DE	FD7FB65D01A4	3
9508670000	2D08FFFFFFFF	FD7FB65D01A4	3
9508790000	FFFFFFFFFFFF	FD7FB65D01A4	3
9508970000	FFFFFFFFFFFF	FD7FB65D01A4	0
9865930000	FFFFFFFFFFFF	FD7FD06FF186	0
9866010000	FFFFFFFFFFFF	CA11D06FF186	0
9866110000	FFFFB65D01A4	CA11D06FF186	0
9866230000	FD7FB65D01A4	CA11D06FF186	0
9866410000	FD7FB65D01A4	CA11D06FF186	1
10071510000	FD7FB65D01A4	CA117EB6061C	1
10071590000	FD7FB65D01A4	FB697EB6061C	1
10071990000	FD7FB65D01A4	FB697EB6061C	2

Figure 4.2 A part of the exported CSV file from the database

4.4. Performing the Toggle Count Analysis:

The toggle count analysis is performed using the Palladium Emulator from Cadence that provides us the database which gives all the high states and the toggle information for all the modules selected. This information is enough for us to understand the power consumption data.

Incisive Palladium Dynamic Power Analysis:

We use the Incisive Palladium for identifying, capturing and analyzing the power switching activity. This allows us to perform the trade-offs between power and performance in a realistic system level environment. This tool is generally used to for peak identification, average power calculation for longer runs, Toggle Count Analysis (TCA) for data processing and filtering and it also supports CPF. One of the major advantages of using this tool is that it supports both RTL and gate level power estimation. We use the RTL based power estimation for our Ethernet IP [24].

In our research, we use only the Toggle Count Analyzer feature of the Palladium emulator for the data processing. This has got the ability to process our input data mentioned in the previous section and give us the toggle count and the high count, which we can directly use as a power consumption data.

Palladium Execution Flow:

The Palladium dynamic power analysis engine is used in the following way in this project using the RTL, input TCF files and port information.

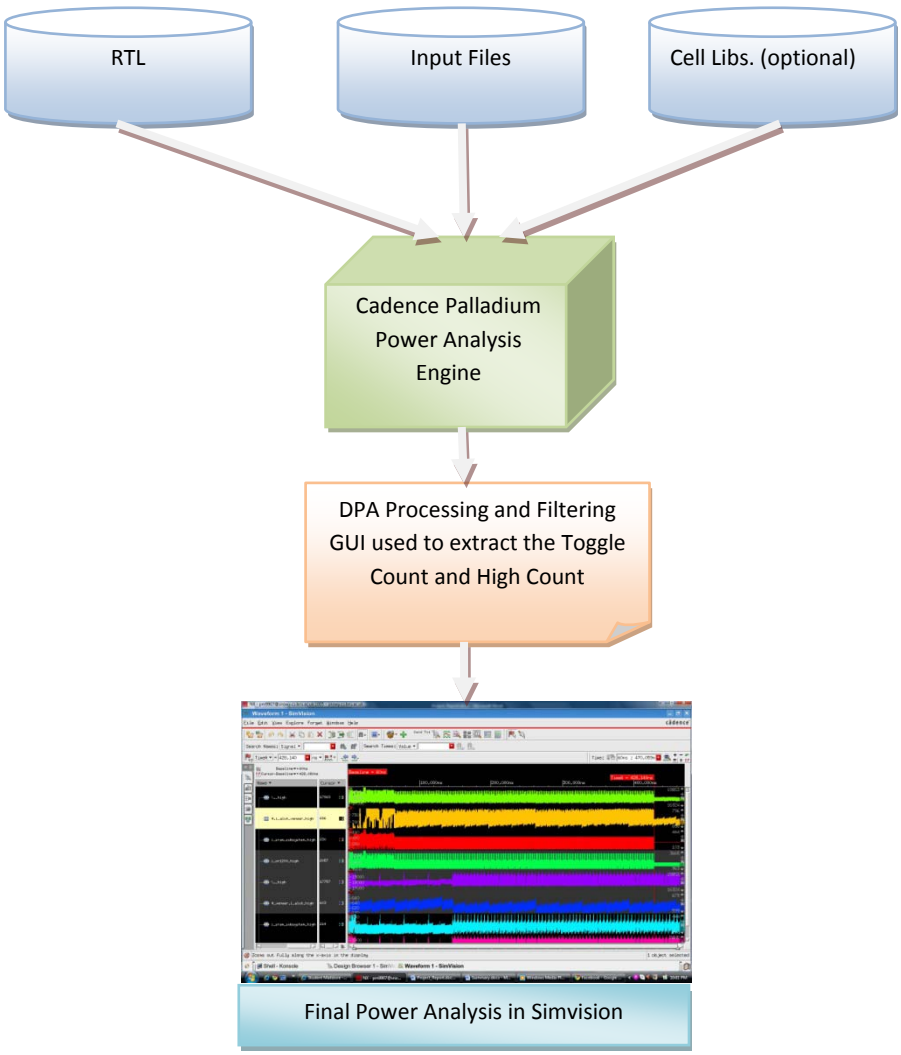


Figure 4.3 Execution flow in Palladium Emulator for the Test Cases

The start time and end time is given in the Palladium and the final database generated by the Palladium is accessible through Simvision. These values taken for our test cases are plotted in Simvision and its final waveform are shown below. They are used directly for our power analysis [24].

Result from Simvision after TCA:

The following table shows the end output taken from Simvision. This shows the high states and the toggle counts.

The total number of high states in the System	17984
The total number of high states in the ALUT module	415
The total number of high states in the SRAM module	280
The number of toggling activity in the System	415
The number of toggling activity in ALUT module	18
The number of toggling activity in SRAM module	48

Figure 4.4 Table showing the final output obtained from the Emulator

In the snapshot below we see the information for the overall System, ALUT and the SRAM. The information displayed is the number of high states and the toggle information. This snapshot represents the total number of high states for the time 360ns which is as given below.

Figure 4.5 Waveforms showing the high states and the toggle counts at time 360ns

We have given the above mentioned *csv* file and *tcf* file as inputs along with the time for execution. This results in the generation of a *.trn* file with the selected modules. This can be opened using Simvision and explored further for our analysis as shown above.

4.5. Test Cases:

In this section we move on to the main test cases. According to the flowchart mentioned we have two main branches of test cases. First one will concentrate on filling the ALUT to the maximum. And the second one will concentrate on filling the ALUT to the minimum. We will analyze the test results in this section. Both of these tests are done with respect to both, the software and the hardware algorithm.

The step by step approach for test cases will not be provided as the previous sections clearly explain how to create the test benches, run the tests and collect the data with an example.

4.5.1. Filling the look-up table as much as possible:

In this test case we will generate the traffic in such a way that the look-up table gets more entries and we will analyze the power for both the hardware and the software design flow.

The test cases extend the *hw_sw_sequence_driver* so as to select the maximum number of ports. This is achieved by giving the appropriate values for the variable *max_stations_per_port* whose default value is given as 2. This variable is declared under the file *cdn_hw_sw_switch.e.*

The test cases also extend the *hw_sw_sequence* so as to provide new values for the data length and the number of frames to be transmitted. The variables for selecting them are declared under the file *cdn_hw_sw_sequence_lib.e*

The basic constructions of these are explained under the *hardware software co-verification section* and also under the section *Verification environment and the DUT*. The detailed development of the file *cdn_hw_sw_sequence_lib.e* and *cdn_hw_sw_switch.e* will not be explained as it will be out of scope considering the aim of the project. Instead, the end product, test case file has been explained.

The test case one and test case two will be having the same *e* file as its test case file. However, they will be executed under two different modes of the ALUT.

Test 1: Testing with the Hardware ALUT:

The Hardware ALUT is enabled by setting the variable *EnableSwAlut* as FALSE so that the system does not take the software ALUT into account and works with the hardware ALUT which is written in Verilog. This is done before the software image was loaded into the memory.

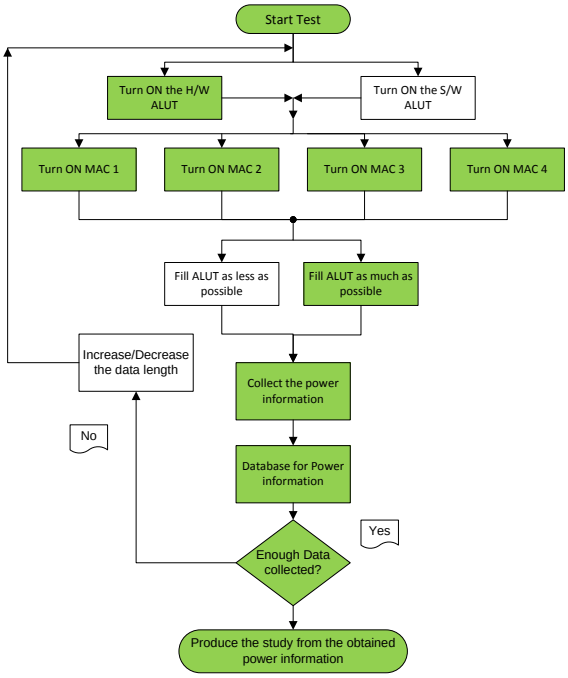


Figure 4.6 Test case on hardware ALUT considering maximum entries in look-up

This test case is designed in such a way that the system uses the maximum number of ALUTs and the number of frames is set to 1000. The test case was run in a similar way and the data are collected for the overall system, the ALUT module, CPU Module and the SRAM module as shown in the snapshot below. The frame size was fixed with a maximum size of 50.

The following is the data obtained after running this test and capturing the data. The following shows the number of high states and the number of toggle activity happened during the run in Palladium Emulator.

Module	High States	Toggles
Overall System	17705.3	640
ALUT Module	698.2	22.7
SRAM Module	227.2	43.1
CPU Module	1767.4	387.9
Others	15012.5	186.3

Figure 4.7 Table representing the average count on High states and the Toggle Count

The data obtained above is the average for the entire range of time which can be obtained by using the calculator option which comes with the waveform window of Specman. This average value represents the number of counts that happened within the time selected during the run in Palladium Emulator.

Test 2: Testing with the Software ALUT:

The Software ALUT is enabled by setting the variable *EnableSwAlut* as TRUE so that the system does takes the hardware ALUT into account and works with the software ALUT which is written in C. This is done before the software image was loaded into the memory. We have set the number of frames as 1000 as given in the hardware test. The test case was run in a similar way and the data are collected for the overall system, the ALUT module, CPU module and the SRAM module as shown in the snapshot below. The frame size was fixed with a maximum size of 50. This test case is exactly the same as the one ran for the hardware ALUT expect for the fact that this will be performed using the software ALUT.

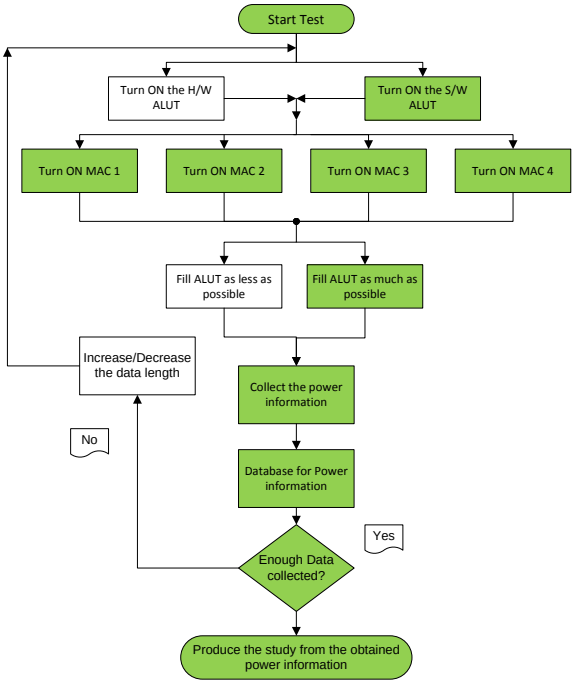


Figure 4.8 Test case on software ALUT considering maximum entries in look-up

In a similar way, the following is the data obtained while running this test and capturing the data. The following shows the number of high states and the number of toggle activity happened during the run in Palladium Emulator.

Module	High States	Toggles
Overall System	17934.3	702
ALUT Module	622.2	20.7
SRAM Module	223.1	32.9
CPU Module	1881.7	488.6
Others	15207.3	159.8

Figure 4.9 Table representing the average count on High states and the Toggle Count

Figure 4.10 Snapshot representing all the eight instances for test 1 and test 2 with filled ALUT.

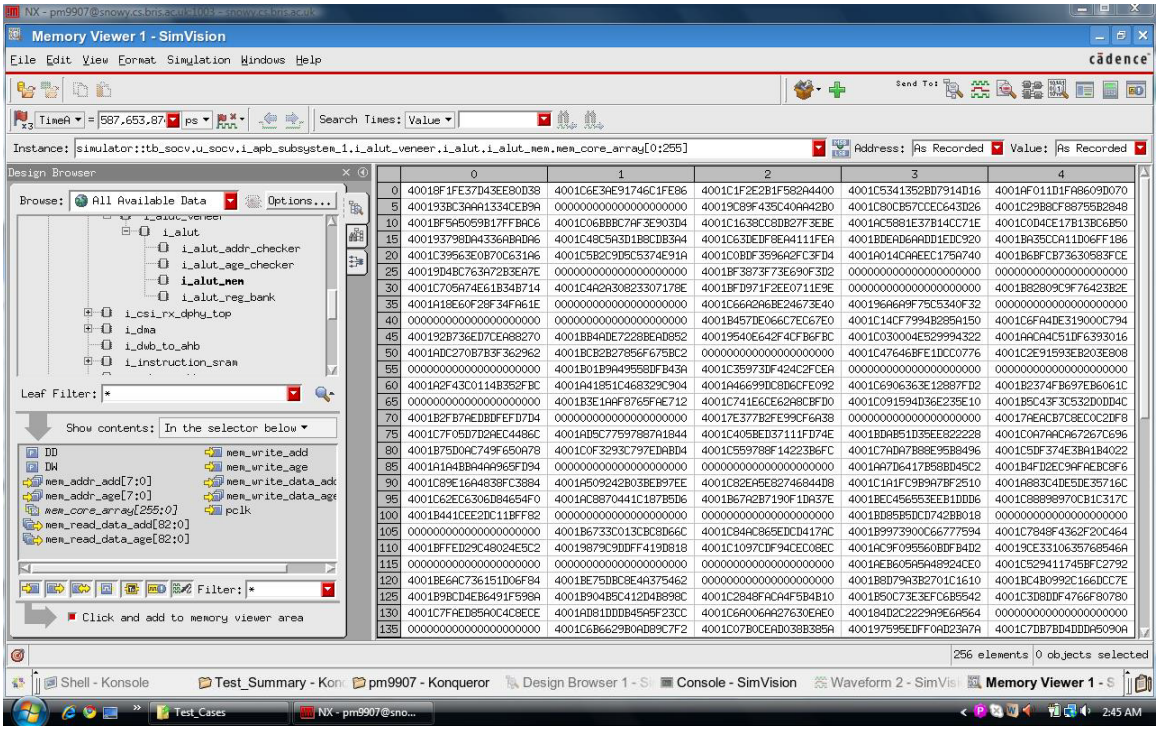


Figure 4.11 Snapshot showing the look-up table getting filled during the run for test 1 and test 2

4.5.2. Tests: Filling the ALUT as less as possible:

In this test case we will generate the traffic in such a way that the look-up table gets less information and we will analyze the power for both the hardware and the software design flow.

This test case is done according to the ALUT hashing algorithm and thus it is designed to create a maximum number of evictions in the look-up table. This test helps us identify the power consumption between the hardware and the software considering the fullness of the ALUT which is related to workload of the ALUT.

Test 3: Testing with Hardware ALUT:

This Test case differs from the previous test cases for the fact that we have added functionality in our test case to evict the ALUT entries. The hardware ALUT is enabled in a similar way by setting the variable *EnableSwAlut* as FALSE so that the system works with the hardware ALUT.

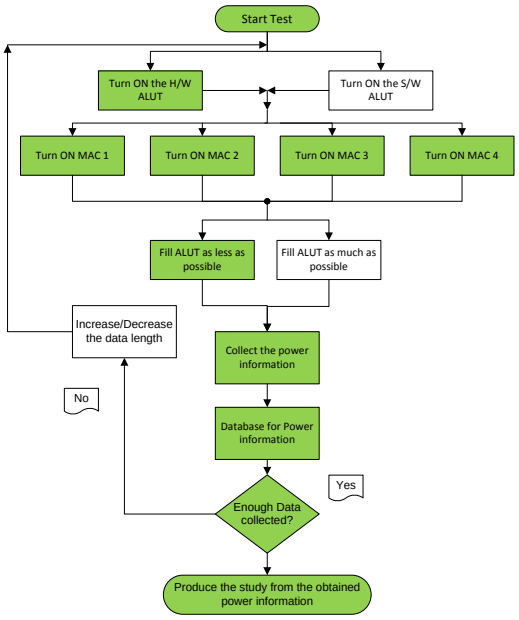


Figure 4.12 Test case on Hardware ALUT considering minimum entries in look-up

We use the same number of frames used in the previous test cases and it is set to 1000. The test case was run in a similar way and the data are collected for the overall system, the ALUT module, CPU Module and the SRAM module. The frame size is fixed with a maximum size of 50. However, a different seed was used to run this test case.

The following is the data obtained while running this test and capturing the data. The following shows the number of higher states and the number of toggle activity happened during the run in Palladium Emulator. The following shows the average values.

Module	High States	Toggles
Overall System	17754	657.2
ALUT Module	699.4	23.7
SRAM Module	228.7	44.8
CPU Module	1801.9	399
Others	15012.5	189.7

Figure 4.13 Table representing the average count on High states and the Toggle Count

Test 4: Testing with Software ALUT:

Compared to the previous test case, the software ALUT is enabled by setting a corresponding value in the variable *EnableSwAlut*. We use the same number of frames used in the previous test cases and it is set to 1000. The test case was run in a similar way and the data are collected for the overall system, the ALUT module, CPU module and the SRAM module as shown in the snapshot below.

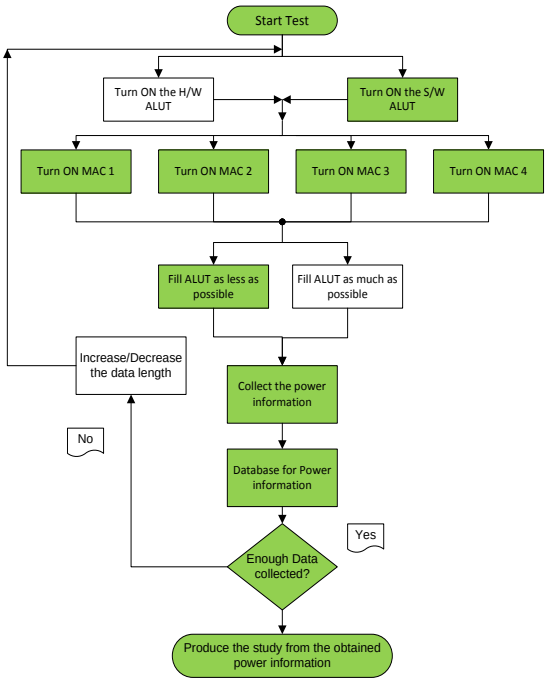


Figure 4.14 Test case on Software ALUT considering maximum entries in look-up

The frame size is fixed with a maximum size of 50. We have used the same seed which was used in the hardware test. The following is the data obtained while running this test and capturing the data. The following shows the number of higher states and the number of toggle activity happened during the run in Palladium Emulator.

Module	High States	Toggles
Overall System	17934.3	702
ALUT Module	622.2	20.7
SRAM Module	223.1	32.9
CPU Module	1881.7	488.6
Others	15207.3	159.8

Figure 4.15 Table representing the average count on High states and the Toggle Count

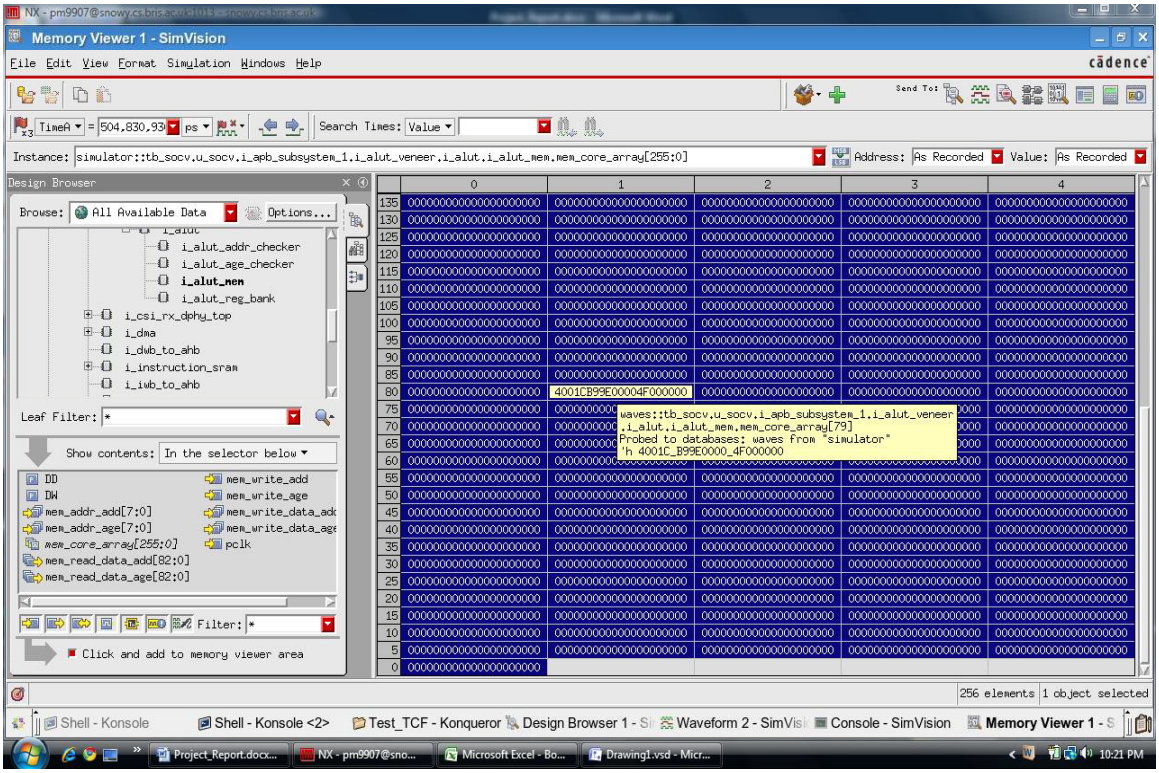


Figure 4.16 Snapshot showing that only one ALUT got filled for Test 3 and Test 4

4.6. Power Analysis:

Having done all the test cases, we have got enough data to perform the analysis. The results will be analyzed separately for the test cases depending on the ALUT entries.

4.6.1. Filling the look-up table as much as possible:

Considering the hardware and software versions of the ALUT, let us summarize the values obtained to see which version has consumed more power when the ALUT got filled to the maximum.

The following table represents the data for Test 1 and Test 2,

Module	Hardware ALUT High States	Software ALUT High States		Hardware ALUT Toggles	Software ALUT Toggles
Overall System	17705.3	17934.3		640	702
ALUT Module	698.2	622.2		22.7	20.7
SRAM Module	227.2	223.1		43.1	32.9
CPU Module	1767.4	1881.7		387.9	488.6
Others	15012.5	15207.3		186.3	159.8

Figure 4.17 Table representing the overall values for test 1 and test 2

Analysis of Test 1 and Test 2:

Considering the Test 1 and Test 2, we have filled the look-up table and have observed the high states and the toggle counts using the Emulator for both the software version and the hardware version. The values in the above table representing the values have been colour coded to represent the ones in green to show that they have taken less power and the ones in red to show that they have consumed more power.

ALUT Module:

Let us begin our detailed analysis with the hardware ALUT module. The hardware version of the ALUT has got an average of 698.2 high states overall, which is higher than the software version having only 622.2. This shows that the ALUT block during the hardware has consumed more power compared to the run with the software version.

Even though the hardware is having more high states compared to the software, considering the difference, the hardware has taken less numbers. This is because, during the run with the software version of the ALUT, the ALUT block in system (RTL) does not perform any operation and the numbers shown for software are only due to the clocking

happened. This can be avoided by gating technology as discussed in the section 2.3. We will look more into this in our critical analysis section.

Looking into the construction of both the hardware and software versions of the look-up table design we find that in the software version, all the functionalities which includes the hash key generation, age checker, address checker and others basic functions are all included in a file called the *AlutDriver.c* which also acts as the firmware for our system.

The hardware version of the look-up table which has got separate files to perform every operation, takes up additional size in the overall hardware. As a conclusion, for a data transfer, with the use of the look-up table, the hardware version takes up more power compared to the software version.

SRAM Module:

From the tables it is clear that using the hardware version consumes more power in the SRAM module when compared while using the software version of the look-up table. This is due to the fact that the hardware block has to perform the data transfer through the APB and the AHB route which is clear from the block diagram.

The data transfer will be from the second APB-Subsystem to the on-chip memories which happens from ALUT to APB to bridge to AHB and then to the Memory. Considering this aspect, using the hardware ALUT is not a good option when saving power is considered. We will analyze in detail how we can overcome this problem which is with respect to the positioning of the ALUT module, in the critical analysis section.

Processor Module:

The processor module is one of the important modules to be considered since the entire system's performance depends on this module. As mentioned in the block diagram, we are using the OR1K processor which is connected through the AHB to the other parts of the system. The software and hardware versions of ALUT may have different workloads for the processor module which in turn will consume more or less power. We will look in detail the power consumption by the processor (CPU) module.

Acting differently when compared with the previously analyzed modules, where the hardware consumes more power, here, the software consumes more power for processor. The value for the high states average is 1767.4 for the hardware which is far less than the value 1881.7 obtained while using the software version. This clearly shows that the power and the workload given to the processor is more when we use the software version.

This extra load and power consumed is due to the fact that the software version of the ALUT does all its operation through the OR1K processor since it doesn't have any dedicated block of its own. On the other hand the hardware version performs all its

calculations and the checking using its own dedicated hardware. This, in turn reduces the workload given to the processor and hence the power consumed is also low. However, are there any advantages of using software version which gives additional load to the processor? This will be analyzed in detail in the critical analysis section.

Overall System Level:

On the whole, the power consumption depends on this set of data which involves all the individual modules including the previously analyzed modules. Comparing both the versions, the hardware has 17705.3 high states on the average and the software has got much higher than this which stands to 17934.3. When it comes to the toggle count, the hardware has got only 640 counts and the software on the other has got 702.

This information clearly shows that the hardware uses less amount of power to do the same operation performed by the software ALUT. When we take the CPU and the overall system's numbers, we find that the software version uses most of the power of the processor and ends up drawing more power altogether. We will analyze more into these results in the critical analysis section also considering the test cases which use less number of entries in the ALUT.

4.6.2. Filling the ALUT as less as possible:

Considering the hardware and software versions of the ALUT, let us summarize the values obtained to see which version has consumed more power when the ALUT got filled to the minimum.

The following table represents the data for Test 3 and Test 4,

Module	Hardware ALUT High States	Software ALUT High States		Hardware ALUT Toggles	Software ALUT Toggles
Overall System	17754	17934.3		657.2	702.6
ALUT Module	699.4	622.2		23.7	20.8
SRAM Module	228.7	223.1		44.8	33
CPU Module	1801.87	1881.7		399	488.6
Others	15024	15207		189.7	160.2

Figure 4.18 Table representing the average High states and the Toggle Count for test 3 and test 4

Analysis of Test 3 and Test 4:

Unlike Test 1 and Test 2, here, we have used only one entry in the look-up table and have observed the high states and the toggle counts using the Emulator for both the software version and the hardware version. The vales in the above table representing the values

have been colour coded to represent the ones in green to show that they have taken less power and the ones in red to show that they have consumed more power. This observed test result shows that the software version is using more power overall when compared to the hardware version.

The results obtained for tests 3 and 4 are similar to the results obtained for tests 1 and 2. Hence, we will not analyze the results in detail again. This shows that the power consumption is independent of the number of entries in the ALUT.

Also, test cases with different number of entries in ALUT behave in a similar way. So, it is not required to perform any more test cases with respect to the number of entries in the ALUT.

4.7 Performance Analysis:

We have analysed the power consumption of both hardware and the software versions of the ALUT. We will look into their performance in this section.

The total time taken for transfer by the hardware = 448,282 ns
The total time taken for transfer by the software = 584,620 ns

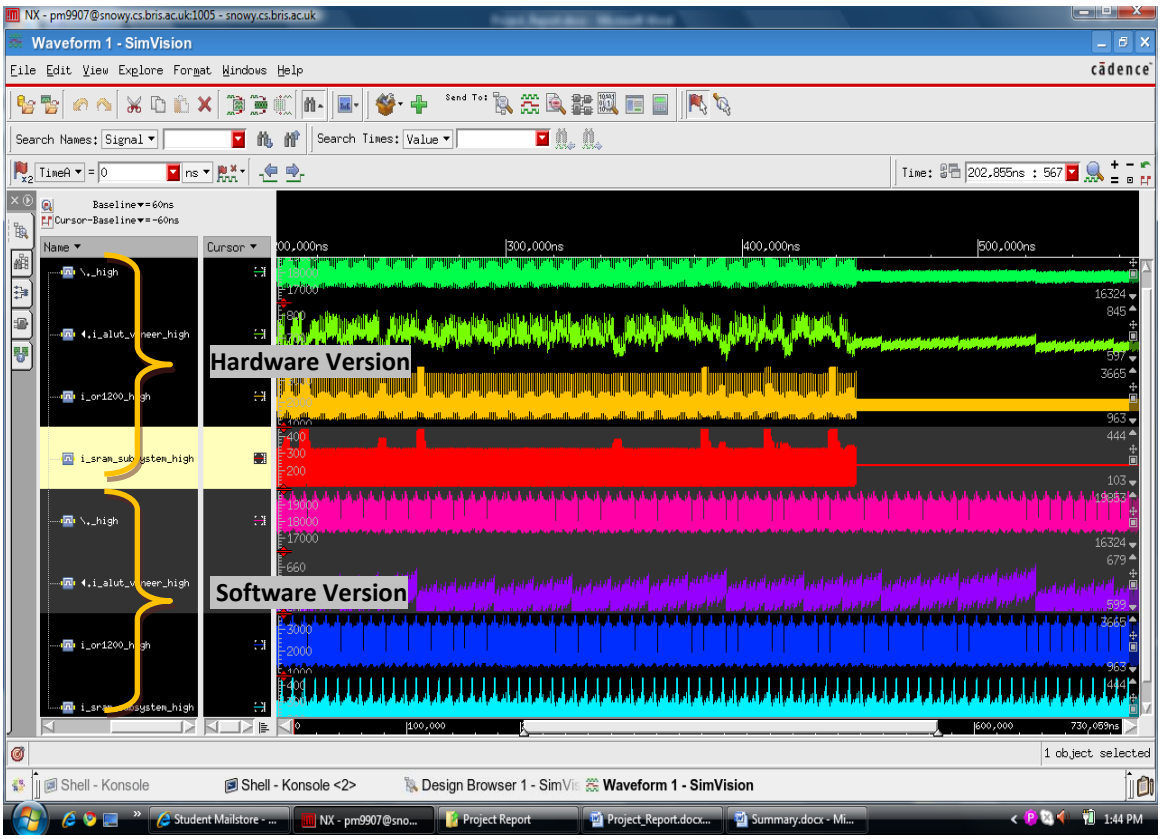


Figure 4.19 Performance Analyses of Hardware and Software Algorithms

This data clearly shows that the speed (performance) of the software version is slower since it does the same operation in a greater amount of time. The result for the performance analysis obtained was similar for the test cases dealing with the fullness of the ALUT. It means that in either case the hardware performed better than the software.

Shown in Figure 4.19 is the snapshot of both hardware and the software algorithms. The first four data represents the hardware and the last four represents the software version. This performance difference between the two versions is due to the fact that the hardware ALUT is dedicated to perform only the look-up operation and it does not have to perform any other operations. On the other hand the software version is present in the memory and it relies on the processor to perform the operation. However, processor in the mean time also performs other tasks so it cannot dedicate itself for only performing the look-up task.

This performance of the software and hardware can be improved further in certain ways. We will look into improving the performance of both the algorithms in the critical analysis section. Thus, after the detailed analysis of the waveform we find that the hardware algorithm is way faster than the software version.

This page is intentionally left blank.

Chapter 5

Critical Evaluation

5.1 Introduction:

We have analyzed in detail the results achieved with respect to (i) The workload of the look-up table (ii) The power aspect and (iii) The performance aspect in the previous chapter, section 4.6 and section 4.7. In this chapter, we will undertake a detailed analysis of using the hardware and the software look-up tables. In addition, we will be providing suggestions on improving the performance of both hardware and the software look-up tables.

5.2 The Software Algorithm:

The software version of the look-up table is coded in C programming language and will be loaded along with the firmware. The major advantage of this is that whenever we find a new algorithm to implement the look-up table, new software for the look-up table can be developed and loaded in the hardware. This can also be done after bug fixing, if we find any bugs after the complete development of the system. Loading the software with its new version is simple.

When considering the size of the system, the hardware address look-up table consumes a lot of space which can be totally avoided by using the software version. This will result in the removal of the entire APB subsystem 2 as shown in the system block diagram. This will result in a compact version of the system. As the cost of using memory has come down, loading the software version of the look-up table will not affect the cost of the system.

A key question is raised while asserting that the software algorithm is not going to take up additional space in the system since it gets loaded in the memory, that is, how is the work done? The software algorithm uses and depends on the system processor to perform its operation. The load of processor gets increased since it is loaded with additional responsibilities. This will in turn reduce the performance of the processor and the overall system.

Cost of production is an important factor when thinking in the system point of view. Developing and testing software for the look-up table will definitely bring down the overall price of the system. The verification can also be done in an undemanding way for software. When compared to doing the same for hardware algorithm of the address look-up table, this time will be fewer. Since verification time and cost goes high for a product of superior range, using the software version will definitely bring down the cost.

Having analyzed the pro and cons of the software version of the ALUT in general, let us analyze the pros and cons with respect to the results obtained from our test cases.

We have analyzed in detail on the results obtained in the section 4.2. This shows that the processor has been given more loads. The power consumed by the processor while working with the software version of the ALUT is much higher. However, all that matters is the overall power consumed by the system. This is also higher when used with the software algorithm. Also, our test cases show that the performance of software version is also low.

5.3. The Hardware Algorithm:

The hardware version of the look-up table is written in Verilog and its RTL cannot be changed after production. This is a drawback when an upgrade is considered. Unlike the software version the hardware version cannot be updated frequently. The entire block has to be replaced.

When the size of the system is considered, the hardware ALUT will take up additional space and it also requires a bridge to communicate through the APB. This bridge is again connected to the main AHB to communicate with the other devices. This setup has been done in the AHB Subsystem 2 block. This block can be removed when we don't need the hardware version which reduces the size of the system hardware.

The best part of the hardware ALUT version is that it does its operation by itself and does not involve the processor. This gives the processer time to do some other important operations. This in turn will increase the performance of the overall system. The workload of the processor is thus reduced since the ALUT hardware takes care of all the activities by itself.

The cost of building the hardware ALUT is more, compared to the software version. This also includes the time required additional resources required for its verification. This depends on the application that the system will be working on. For critical applications, the cost will not be an issue. For other simple applications, the cost does matter and the hardware algorithm will definitely shoot up the cost.

Having analyzed the pros and cons of the hardware algorithm in general, let us move on to the analysis with respect to the power constraint. The tables and results analysis of section 4.2 clearly shows that the hardware consumes more power on certain blocks. However, it is also made clear that the hardware algorithm saves a lot of power in the CPU and also the power consumed by the overall system is fairly low when compared to the software algorithm.

Thus using the hardware ALUT saves power. When we say that the power consumed by the processor is low, it means that the work load of the processor is also low. This will result in a better performance for the overall system and also save power. This is again proved from our test cases.

5.4. How to improve the System:

Having studied the operation of both the hardware and the software versions of the address look-up table, it is time for us to provide valuable suggestions on improving the efficiency of the system on both power and performance wise. We have encountered the power values for various blocks during the data transfer. Are there any ways to improve the speed and reduce the power consumption? We will look into the system in these aspects in the following sections.

5.4.1. Improving the Hardware version of the ALUT:

In the section 5.3 we encountered a few drawbacks while using the hardware version of the ALUT. Considering those points, here we present some valuable suggestions for improving the performance of the system while using the hardware ALUT.

The ALUT block can be moved from the APB subsystem 2 to the AHB system which will increase the performance of the look-up process. This increase in the performance will be due to the improvement in accessing the SRAM. This will not only improve the performance but it will also reduce the size and the additional power consumed by the APB subsystem.

Upgrading the processor from OR1K to a better commercially available processor will give a better performance for the system. Though the impact will be far better for the software version, a superior processor will speed up the firmware and thus the overall systems performance will be improved.

Another improvement on the hardware algorithm is that it can be given a separate SRAM or a Content-Addressable Memory (CAM). This will give a better performance for the hardware algorithm due to the fact that the memory accessing time will be reduced drastically.

5.4.2. Improving the Software version of the ALUT:

When the system is using the software version of the ALUT, there are power losses in the AHB subsystem 2. This power wastage is due to the fact that clocking has been happening. This can be avoided by using the clock gating technology as discussed in the *Background* section of chapter 2. This leaves us with a lot of saving in power.

The other major improvement in software version can be done by including a cache memory which is several times faster than the SRAM. By decreasing the access time, the overall system efficiency can be improved to a greater extend. This not only increases the overall efficiency, it also saves a lot of power and paves the way for a low power and a faster system.

This system uses the open source OR1K processor which can be upgraded to a more efficient low-power embedded processor which may slightly increase the cost. However, the system will definitely show a better performance.

The software version can also be improved by using various code optimization techniques available for C programming. They should be used to the maximum, to make the system faster. Writing the frequently used part of the code in assembly will also make the system faster.

5.5 Which version to choose?

As given under the aim of the project, the ultimate plan is to correlate the power consumption data from the hardware and the software point of view. With the help of all our experiments, we now come to a conclusion that this system without any modifications given in section 5.4 is consuming more power when it is used with the software version and the performance is also low. We suggest the usage of hardware ALUT when power saving and better performance is needed.

This leaves us with the hardware version with one drawback, the size. Possible solutions for reducing the size by positioning the block in a different area and thus increasing the speed and performance is discussed in the section 5.4.1.

Chapter 6

Further Work and Possible Improvements

As mentioned in the previous section, there are several improvements that can be done to make our system a faster and a less power consuming system. The suggestions given are for both the hardware and the software algorithms. However we never know which algorithm will consume less power after implementing the suggestions. Repeating these experiments will definitely reveal those secrets and an ultimate low power system can be thus found.

Address look-up tables are widely used these days, from fuel injection systems for automobile engines to simple code translation units. All of these embedded systems can have either hardware or a software constructed address look-up tables and there can be various scenarios in which they will have to be used. This research can be done on those systems as well, to give an appropriate or exact figure on which one is power efficient.

This research does not stop with only analyzing the systems with the look-up tables. This can be performed on various applications with modules given in hardware and software. The manufactures can provide both the algorithms and provide the software engineers a data sheet mentioning which version to use depending on the application the software engineers intend to develop. This can also be done by giving the software engineers an option to dynamically select them during the run time so as to save the power.

This research was jointly done with Cadence Design Systems. They have considered this project for further exploration into their research on hardware and software power tradeoff.

This page is intentionally left blank.

Chapter 7

Conclusion

Before the starting of this project in finding which algorithm is power efficient, I had discussed about the possible version which would consume more power. The answers and thoughts received were of mixed. However, with all the experiments it became clear that it was only the software version which consumes more power.

As a conclusion, considering this SoC, the overall power utilized by the software version of the ALUT is higher compared to the hardware version of the ALUT irrespective of the number of entries in ALUT. It has also been found that the hardware version not only takes less power but it also gives a better performance.

Valuable suggestions have been given on improving both the versions along with the explanation for their present performance. There are many unexplored systems on which this investigation can be done so as to find the possible ways to save power in this power thirsty world.

This page is intentionally left blank.

Bibliography

1. *A Practical Guide to Low-Power Design User Experience with CPF* accessed on 1st September, 2010 from www.**power**forward.org/lp_**guide/**

2. *Verifying System-Level Power Management* (San Jose: Cadence Design Systems, Inc., 2007-2010)

3. Sperling, Ed, *Verifying Low Power Design,* accessed on 1st September, 2010 from http://chipdesignmag.com/lpd/blog/2010/01/14/verifying-low-power-designs/

4. Flautner, Krisztian, and David Flynn, *A Combined Hardware-Software Approach for Low-Power SoCs: Applying Adaptive Voltage Scaling and Intelligent Energy Management Software*, DesignCon 2003 Proceedings, January 2003.

5. Piziali, Andrew, *Verification Planning to Functional Closure of Processor-Based SoCs*. DesignCon, Feb. 2006. Accessed on 5th September, 2010 from www.designcon.com/2006/pdf/3-tp2_piziali.pdf

6. Stump, Holly, and George Harper, *ESL Synthesis + Power Analysis = Optimal Micro-Architecture,* Chip Design Magazine, Jan. 2007 Accessed on 5th September, 2010 from www.chipdesignmag.com/display.php?articleId=963&issueId=20

7. Saito, Toshiyuki, *Integrating Power Awareness into IC Design*, NEC Electronics Corporation EDA DesignLine 03/01/2007 Accessed on 5th September, 2010 from www.edadesignline.com/howto/showArticle.jhtml?articleID=197700296

8. Schulz, Steve, *A Practical Case Study in Low Power Design Methodology*, EPN Online Accessed on 7th September, 2010 from http://www.epn-online.com/page/new56459/a-practical-case-study-in-low-power-design-methodology.html

9. Sivaramakrishnan, Krishnan, *Low power challenges push for system-level EDA June, 2007,* Accessed on 7th September, 2010 from http://www.eetindia.co.in/ART_8800466314_1800000_NT_42c851c2.HTM

10. *Cadence 65nm Low-Power Reference Flow for Common Platform Technology* IBM Corporation July 2007, Accessed on 7th September, 2010 from http://www.cadence.com/rl/Resources/conference_papers/common%20platform.pdf

11. Menager, Herve, *Words of Power: Reusable, Holistic, Scalable Multi-voltage Design*. EPD Conference, 12 April 2007. NXP. Accessed on 7th September, 2010 from www.eda.org/edps/edp07/ApprovedPapers/01%20Herve%20Menager.pdf

12. Hung, W. , Y. Xie, N. Vijaykrishnan, M. Kandemir, M. J. Irwin and Y. Tsai, *Total Power Optimization through Simultaneously Multiple-VDD Multiple-VTH Assignment and Device Sizing with Stack Forcing,* Accessed on 10th September, 2010 from http://www.cse.psu.edu/~yuanxie/Papers/ISLPED04-Hung.pdf

13. Munch, M., B. Wurth , R. Mehra, J. Sproch and N. When, *Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths* (New York: ACM Publication, 2000)

14. Waldo, Brandon, and Jim Flynn, *Power Management in Complex, SoC Design*, Accessed on May 2nd, 2010 from http://www.techonline.com/article/pdf/showPDF.jhtml?id=1931025601&_requestid=46725

15. Girard, Patrick, Nicola Nicolici, Xiaoqing Wen, *Power-Aware Testing and Test Strategies for Low Power Devices* (New York: Springer Science+Business Media, LLC 2010)

16. Palnitkat, Samir, *Design Verification with e* (New Jersey: Pearson Education, Inc., 2004)

17. Iman Sasan, Sunitha Joshi, *The e Hardware Verification Language* (Norwell: Kluwer Academic Publishers, 2004)

18. *Introduction to the e Hardware Verification Language* (San Jose: Cadence Design Systems, Inc., 2007-2009)

19. *Open Verification Methodology* (San Jose: Cadence Design Systems, Inc., 2006-2010)

20. *Hardware-Software Co-Verification Methodology* (San Jose: Cadence Design Systems, Inc., 2006-2010)

21. *Address Look-Up Table Design* Document (San Jose: Cadence Design Systems, Inc., February 2007)

22. *Hardware/Software Co-Verification Flow Guide* (San Jose: Cadence Design Systems, Inc., 2006-2010)

23. *Incisive VIP Ethernet UVC e User Guide* (San Jose: Cadence Design Systems, Inc., 2005-2010)

24. *INCISIVE Palladium III Dynamic Power Analysis Datasheet* (San Jose: Cadence Design Systems, Inc., 2008)

25. *Common Power Format Language Reference* (San Jose: Cadence Design Systems, Inc., 2007-2009)

Appendices

This project has been done in collaboration Cadence Design Systems. I am not allowed to upload or document the code (Test cases) due to confidentiality.

I have attached a letter from Nick Heaton, Senior Solution Architect of Cadence Design Systems, UK, mentioning the successful completion/execution of the Project.

Please refer next two pages.

cādence®

28th September 2010

Nick Heaton, Senior Solution Architect, Cadence Design Systems

To Whom it may concern

I have been mentoring Padmanabhan Mohan Das (Paddy) during his MSc project and wanted to provide my overview of the work he has done and results achieved.

The project has been centred around a System-on-Chip (SoC) designed to perform as a 4-port Ethernet Switch. The design comprises a processor, memory, 4 Ethernet MACs and assorted peripherals such as UART and GPIO.

Ontop of this system, embedded SW is run to perform the Switching algorithm which manages a hash table of MAC address to port mappings in order to optimize the broadcast traffic generated on each of the 4 network segments. In order to perform this function more efficiently a specialized hardware IP is included to perform Address Look-Up Table (ALUT) in hardware, this frees the processor of the burden of implementing the algorithm. This SoC is instantiated inside a constrained random testbench designed to simulate realistic Ethernet traffic scenarios from high level 'e' test cases using constraints, the testbench uses commercial Ethernet Verification IP (VIP) components to simplify the construction.

The goal of the project was to investigate the impact on power consumption of using the hardware ALUT accelerator or implementing it in software. In order to obtain the results Paddy had to fully understand the overall application, the architecture of the SoC, the architecture of the Embedded SW including how to switch between using the ALUT HW and the SW implementation. Using this knowledge he then needed to create top-level scenarios using constraints to exercise the System and capture toggle data from the simulation which Cadence could then run through a remote HW Accelerator (in San Jose, California) in order to get power consumption estimates.

Cadence Design Systems – Bagshot Road Bracknell, Berkshire RG12 3PH United Kingdom / Ph: +44.1344.360333

cādence

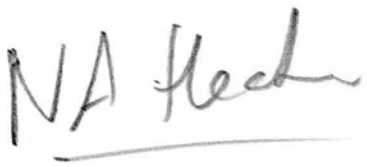
For each of the tests Paddy ran he delivered the toggle data (TCF file) and using this data the power consumption results were delivered back for his analysis.

As might be expected it took Paddy some time to assimilate the concepts and all the various components and languages that comprise the simulation environment he needed to run his testcases. However once he got a good understanding of this complex environment he produced a number of testcases in order to test actual versus expected results.

Paddy's results show as might be expected that there is a power benefit to running the algorithm in a dedicated piece of HW given the current implementation. Given the target SoC uses just a simple OpenCores processor one might expect the savings to be even greater on a more complex commercial embedded processor.

Paddy's work has been very valuable in streamlining our tool flow for this design and will provide the basis for further more detailed explorations into HW/SW power trade-offs.

Yours sincerely



Nick Heaton

Cadence Design Systems – Bagshot Road Bracknell, Berkshire RG12 3PH United Kingdom / Ph: +44.1344.360333