Fully Configurable Hierarchical Transaction Level Verifier for Functional Verification

Except where reference is made to the work of others, the work described in this Project is my own or was done in collaboration with my advisory committee. This Project does not include proprietary or classified information.

Chaitanya Bandi

Certificate of Approval:

Victor P. Nelson
Professor
Electrical and Computer Engineering

Vishwani D. Agrawal
James J. Danaher Professor
Electrical and Computer Engineering

Fa F. Dai
Professor
Electrical and Computer Engineering

George T. Flowers
Interim Dean, Graduate School

Fully Configurable Hierarchical Transaction Level Verifier for Functional Verification

Chaitanya Bandi

A Project

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Electrical Engineering

Auburn, Alabama
Feb 25, 2009

Fully Configurable Hierarchical Transaction Level Verifier for Functional Verification

Chaitanya Bandi

Permission is granted to Auburn University to make copies of this Project at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

Date of Graduation

Vita

Chaitanya Bandi, daughter of Pandu Ranga Reddy Bandi and Rajya Lakshmi Bandi was born on August 27th, 1985, in Krishna District, India. She received the Bachelor of Technology degree in Electronics and Communication Engineering from Nagarjuna University, India in 2006. She joined the Masters program in the Department of Electrical and Computer Engineering at Auburn University in January 2007. Her project work is focused on novel techniques for Functional Verification of Computer Aided Design circuits. This work was done at the NAND Solutions Group, Intel Corporation, Folsom, CA as a part of her Internship during June 2008- Dec 2008.

Project Abstract

Fully Configurable Hierarchical Transaction Level Verifier for Functional Verification

Chaitanya Bandi

Master of Electrical Engineering, Feb 25, 2009
(B.Tech., Nagarjuna University, 2006)

30 Typed Pages

Directed by Vishwani Agrawal

As the complexity of semiconductor design increases, the need for a faster design and verification environment arises. This has been accomplished by modeling and validating designs using Transaction Level Modeling (TLM), a higher level of abstraction than the Register Transfer Level (RTL). Transaction-level modeling (TLM) is a high-level approach to modeling digital systems where details of communication among modules are separated from the details of the implementation of functional units or of the communication architecture [1]. RTL is at a lower level of abstraction for validation of complex digital circuits, whereas validation at the Transaction Level loses details of the actual implementation at the RTL. A Fully Configurable Hierarchical Transaction Level Verifier which makes use of system behavior at the Transaction Level and signal behavior at the RTL leads to a faster functional verification of the chip.

In our approach, the system response to a set of stimuli is recorded as a sequence of transactions. These transactions can further be modeled as a sequence of sub transactions or packets, thus forming a hierarchical model. Only packets that are validated at the current level in the hierarchy are stepped into the next level for further investigation. The leaf in the hierarchical model will be at the RTL; hence only packets that pass all the layers above

it will be compared at the signal level. The verifier can choose whether or not to step into a transaction packet depending on user configuration, thus making it "fully configurable".

This Verifier has been designed to serve as an efficient regression environment to validate the functional correctness of the design revisions in "NAND Flash Memory design group, Intel Corporation, Folsom, CA". This work was done as a part of an Internship project under the supervision of the members of Design Automation during June 2008- Dec 2008.

Acknowledgments

First and foremost, I would express my sincere gratitude to my advisor Professor Vishwani Agrawal for his valuable guidance and support during my study and research at Auburn. Thanks for leading me into the research area of VLSI Design and Testing. I would like to express many thanks to all the members of Design Automation team at Intel's NAND Solutions Group, especially my mentor Anshuman Singh. Without their guidance and support, this project work would not be possible.

I also thank Professors Victor Nelson and Fa Dai for serving on my advisory committee. My thanks also go out to my colleagues in the VLSI Design and Testing courses for their valuable suggestions in my research. My special thanks to my friend Siddharth Dantu who made my stay at Auburn a pleasant one.

I am grateful to my parents and my sister for their consistent support and encouragement during my study and thanks to my friends for their kind presence whenever needed.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures x

1 Introduction 1
  1.1 Regression Testing 1
  1.2 Transaction Level Modeling 2

2 NAND Flash Memory Architecture 5
  2.1 Nand Flash Transistor Structure 5
  2.2 Principles of Operation 6
  2.3 Reading a Flash Cell 6
  2.4 Programming the NAND Cell 8
  2.5 Erasing the NAND Cell 8
  2.6 The Term "NAND Flash" 9
  2.7 Structure of NAND Flash Memory Chip 10

3 Verification in NAND Flash Design Group 11
  3.1 Proposed Solution 11

4 Design of the Transaction-level verifier 13
  4.1 Microcode Regression 14
  4.2 Comparison at a glance 17
  4.3 Controllable Comparison 17

5 CONCLUSION 18

Bibliography 19

List of Figures

1.1 Intel's new 34nm 32Gb NAND Flash chip 3

2.1 NAND Flash Transistor Structure 5

2.2 Reading from a erased cell 7

2.3 Reading from a programmed cell 7

2.4 Programming the NAND cell 8

2.5 Erasing the NAND cell 9

2.6 Structure of NAND Vs NOR cell 10

4.1 Hierarchical Transaction Level Model 13

4.2 Microcode Comparison Algorithm 15

4.3 Determining good slices and bad slices 16

4.4 Slicing based comparison 16

4.5 Overview of the two-level comparison 17

Chapter 1

Introduction

Semiconductor non-volatile memory trends have led to the demand for more features within personal electronic devices such as cellular phones, handheld computers, faster and low power solid-state drives, etc. Flash memory has advanced rapidly to higher volume density and performance levels to meet these demands in the industry. NAND Flash memories have dominated the industry due to a tremendous decrease in price and due its wide range of applications. NAND Flash memory was widely initially in mp3s, USB, memory cards, etc., now with an application towards replacing hard disks with solid state drives. Despite this rise in design complexity, the faster solutions for design verification must be developed due to short time to market requirements. Verification of the NAND flash memory design involves checking that its implementation as a network of transistors implements the high-level view of a state machine storing and retrieving data at addressed locations [2]. This project involves designing a faster verification environment in NAND Flash memory design organization.

1.1 Regression Testing

Industry designs are huge and go through several revisions before they are ready for tape-out. These revisions are designed and simulated iteratively until the design is predicted to meet the desired quality goals. It is the responsibility of the verification engineers to make sure that these revisions don't break the functionality that already exists. This task is accomplished by comparing the design revision against a golden function model. By "comparing" we mean the response to a set of stimuli of the Design under Test (DUT) is

compared against the response to the same set of stimuli of a golden reference model of the design. This process is termed as Regression Testing.

Regression testing is a process that is applied after a design has been modified. It is a testing process in which sequences of tests are applied to a modified design to re-establish our confidence that the design will perform according to the specifications. In the initial stages of testing, a golden functional reference model that has been tested thoroughly is modeled to serve as a reference. Regression testing is a crucial step in the maintenance phase where the design can be corrected, adapted to new environment, or enhanced to improve its performance. Modifying the design might include creating new features to correct an error or to implement a change by adding new logic into an existing design. The new logic may involve minor modifications such as adding, deleting, rewriting a few lines of code, or major modifications such as adding, deleting or replacing one or more modules or subsystems. The major role of Regression testing is to check the correctness of the new logic, to ensure the continuous working of the unmodified portions of a program, and to validate that the modified program as a whole functions correctly [3].

1.2 Transaction Level Modeling

Transaction level modeling (TLM) addresses the problems of designing increasingly complex systems by raising the level of design abstraction above RTL. Transaction level models use software function calls to model the communication between blocks in a system, whereas in RTL and gate level models, signals are used to model the communication between blocks. For example, a transaction level model would represent a read or write transaction using a single function call, with an object representing the request and another object representing the response. An RTL HDL model would represent such a read or write transaction as a series of signal assignments and signal read operations [5].

Transaction level models helps in:

a) Embedded software development: It enables the hardware dependent software to be validated even before it runs on the prototype [7].

b) Functional Verification: Providing a reference model for the verification engineer. The "golden model" is used to generate tests which can later be applied to the implementation model to verify its functionality [7].

In our approach, we are going to utilize the benefits of TLM in functional verification to validate design revisions in NAND Flash Memory Design Group. The"golden model" in this case is derived from a previous version of the design that is manually verified to be functionally correct. In order to leverage the benefits of Transaction level verification in systems that have been designed using traditional RTL and don't contain TLM concept in their design, the verification environment must have an ability to extract transactions which then enables transaction-level comparison [8]. This feature has been included in our verification system by tracking the instructions that are being executed during a simulation in the microcode processing unit, called the controller. These microcode instructions are modeled as transactions in our case as they are at a higher level of abstraction than the signal values.

In the NAND Flash chip, the controller as in Fig. 1.1. is the heart of the functionality of the chip.

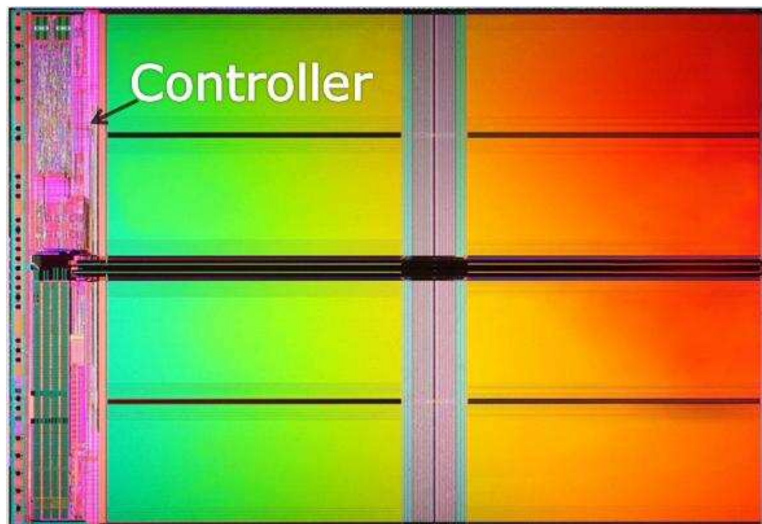

Figure 1.1: Intel's new 34nm 32Gb NAND Flash chip [1]

[1]Courtesy: http://www.intel.com/pressroom/archive/releases/20080529comp.htm

It interfaces with the outside world and sends out instructions to the bus accordingly. When the controller receives a “read” or a “write” instruction, it sends out signals on the bus to receive the data in the memory or to write to the memory respectively. During a simulation, these series of instructions are tracked as microcode instructions and modeled as transactions for the purpose of functional verification.

Chapter 2
NAND Flash Memory Architecture

In this chapter, the basic operations in NAND Flash memory devices like Read, Erase and Program will be discussed. We will also discuss the working of the Flash Transistor and how it is termed as Non-Volatile. Most of the discussions in this chapter are learnings from [16].

2.1 Nand Flash Transistor Structure

The structure of the NAND Flash cell is similar to a MOS device with an extra polysilicon layer that acts as a floating gate between the gate and the channel as in Fig. 2.1. This extra polysilicon layer is not connected to anything else, and hence the name floating gate. The Floating gate transistor forms the core of virtually every NAND Flash memory built today. Floating gate stores charge for extended periods of time even though power supply is turned off, thus making the flash devices as non-volatile.

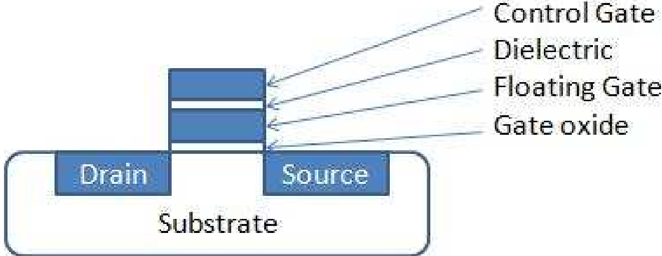

Figure 2.1: NAND Flash Transistor Structure

2.2 Principles of Operation

Flash memory stores the information in an array of transistors. By adding or removing electrons from the floating gate, we can store the information in a flash transistor. We can add charge to the floating gate by a phenomenon called as Tunneling. Tunneling is described as a phenomenon in which the electrons acquire enough energy to move across the insulating gate oxide layer in the presence of a huge potential difference. The electrons thus accumulated in the floating gate are trapped in between two insulating layers. Adding or removing electrons to the floating gate will increase or decrease the threshold voltage of the transistor. Threshold voltage, Vth is the voltage that has to be applied to the control gate of the transistor in order to turn on the device, i.e., to allow a flow of from the Source terminal to the Drain terminal. The threshold voltage can be modulated by controlling the charge on the floating gate. This property is termed as Threshold voltage modulation.

2.3 Reading a Flash Cell

The purpose of reading a flash cell is to distinguish between a memory cell that is programmed and a memory cell that is erased.

a) Let us consider a memory cell that is erased and read the contents of the Flash transistor. By 'erased' we mean that there is no charge accumulated on the floating gate. A voltage applied on the gate $Vg > Vth$ of the device should turn on the device and enable current flow from the source to the drain as in Fig. 2.2. Reading an erased cell results in a current flow which is read as logic '1' stored in the flash cell.

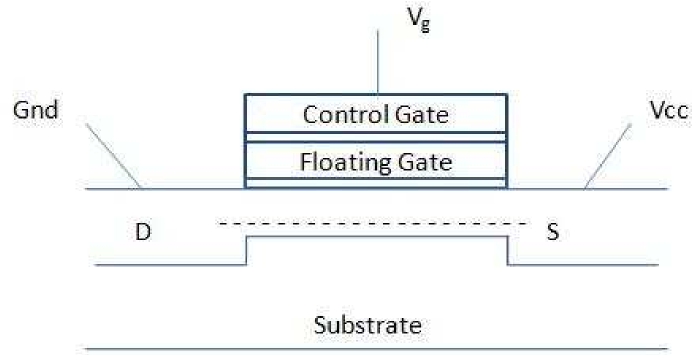

Figure 2.2: Reading from a erased cell

b) Let us consider a memory cell that is programmed and read the contents of the Flash transistor. By 'programmed' we mean that there is a charge accumulated on the floating gate. A voltage applied on the gate Vg > Vth of the device is not sufficient to turn on the device as there are additional electrons on the floating gate hindering the flow of current from source to drain as in Fig. 2.3. Reading a programmed cell does not result in a current flow which is read as logic '0' stored in the flash cell.

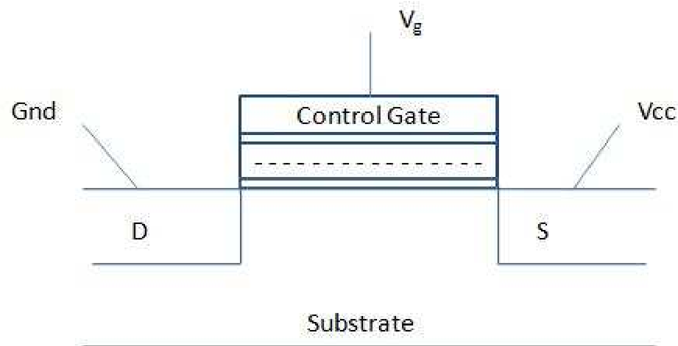

Figure 2.3: Reading from a programmed cell

2.4 Programming the NAND Cell

Programming a NAND cell can be interpreted as the process in which electrons are accumulated on to the floating gate. This operation is achieved by the phenomenon of tunneling. For this phenomenon to take place a huge voltage is to be applied to the gate of the device to ensure that there is enough potential difference for the electrons to tunnel from the substrate to the floating gate as in Fig. 2.4. Hence we apply a high voltage of about 20V to the gate to ensure that tunneling takes place.

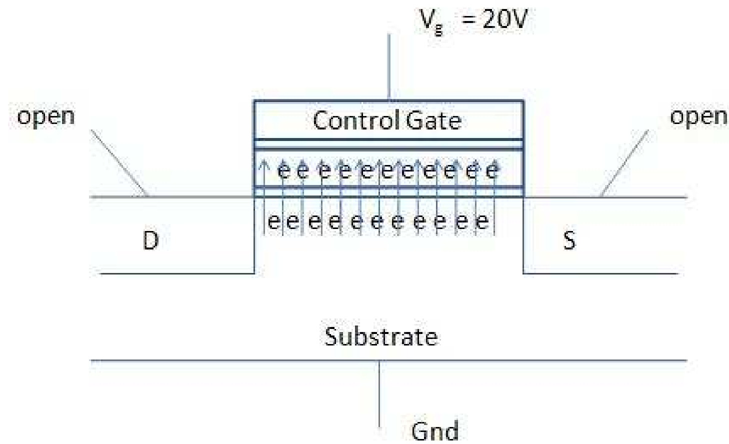

Figure 2.4: Programming the NAND cell

2.5 Erasing the NAND Cell

Erasing a NAND cell can be interpreted as the process in which electrons that are accumulated on the floating gate are tunneled back to the substrate. This operation is exactly reverse to the programming of the NAND cell. For this phenomenon to take place a huge voltage is to be applied to the substrate of the device to ensure that there is enough potential difference for the electrons to tunnel from the floating gate back to the substrate Fig. 2.5. Hence we apply a high voltage of about 20V to the substrate to ensure that tunneling takes place.

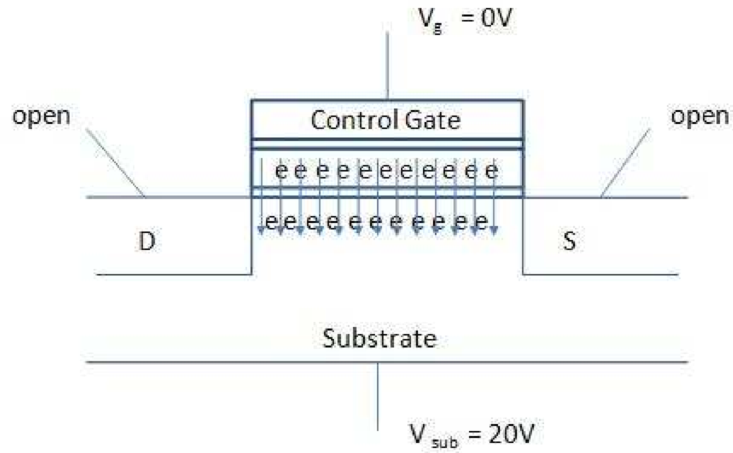

Figure 2.5: Erasing the NAND cell

2.6 The Term "NAND Flash"

The term "NAND" Flash is originated from the way in which these flash transistors are connected to each other. In NAND Flash memory, the source of one transistor is connected to the drain of the next transistor, which results in reading blocks of data at a time as in Fig. 2.6. In NOR Flash memory, each transistor is available independently for random access of data. This arrangement of transistors results in the name NAND Flash [18].

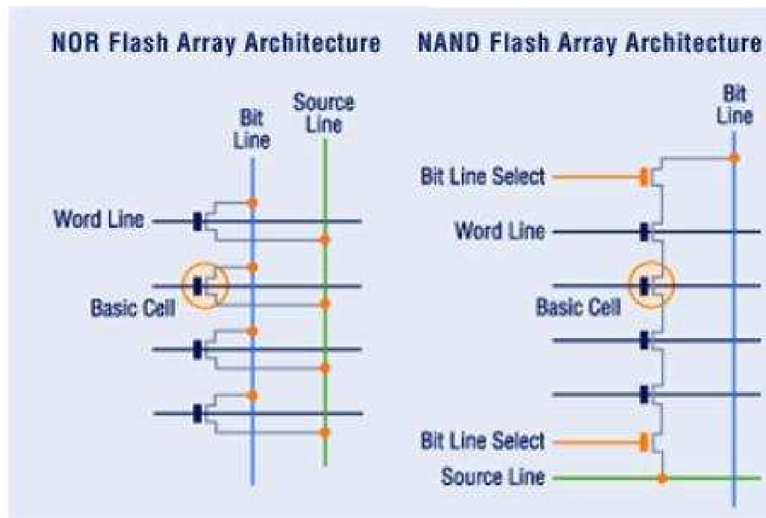

Figure 2.6: Structure of NAND Vs NOR cell

2.7 Structure of NAND Flash Memory Chip

The NAND Flash memory accounts for a major portion in the size of the chip. Peripheral circuits are located around the memory array such as sense amplifiers, control signals and row decoders. The Flash chip contains a "controller" that sends and receives instructions from the outside world to the memory array. It controls the data that is to be written and sends out the data that is read from these memory arrays. The controller contains a micro processing unit that will take into account the series of commands that are to be given to the Flash memory in case of a Read, Program or Erase operation.

Chapter 3
Verification in NAND Flash Design Group

In this chapter, we describe how netlist revisions were manually verified in NAND Flash design group at Intel, and suggest ways to improve the verification time.

Currently, functional verification is done by reviewing the log files after a fullchip netlist revision is simulated, and by checking the waveforms to verify if the algorithm flow is fully functional or not. This is a step-by-step, manual review of the microcode execution using full-chip simulation information. This is obviously a manual process and it requires extensive knowledge of the full-chip functionality for verification. This resulted in a turnaround time of few days for validating each netlist revision. Several automation methods have been tried in this context like standard waveform comparison tools, but they resulted in irrelevant differences [4].

3.1 Proposed Solution

RTL is at a low level of abstraction for functional verification. Verification at a higher level of abstraction yields better results in terms of time; but loses information on the signals. By the use of fully configurable hierarchical transaction level modeling and configuring it based on the design; debugging and validation of the system is greatly simplified. The data flow through the chip can be validated at higher levels of abstraction, then stepping through each transaction and further validating it at a lower level of abstraction. This approach is efficient than

a) Comparing the individual waveforms at signal level where the available data is in a raw format in-comprehendible to view if the microcode is being executed as per desired.

b) Comparing only the transactions at higher level of abstraction which loses information on the signals.

To validate the behavior of a new model, a golden functional reference model is needed to compare new revisions with. This golden model is developed based on previous versions of the design that have been tested thoroughly.

Chapter 4

Design of the Transaction-level verifier

Separating the communication among modules with the details of the actual implementation is the key to Transaction Level Modeling [13]. The first step to this approach is to model the system functionality as transaction packets. Each packet can further contain multiple transactions based on the user configuration. The leaf in the hierarchy tree is signal values pertaining to the parent packet above it. As we go down in the hierarchical tree, the implementation details are enhanced. The user can configure whether to step into a transaction packet or not based on its relevance in the hierarchy tree. The signals that are to be compared can also be configured by their relevance to the parent packet. In NAND Flash Memory design, the full-chip model contains a microcode processing unit

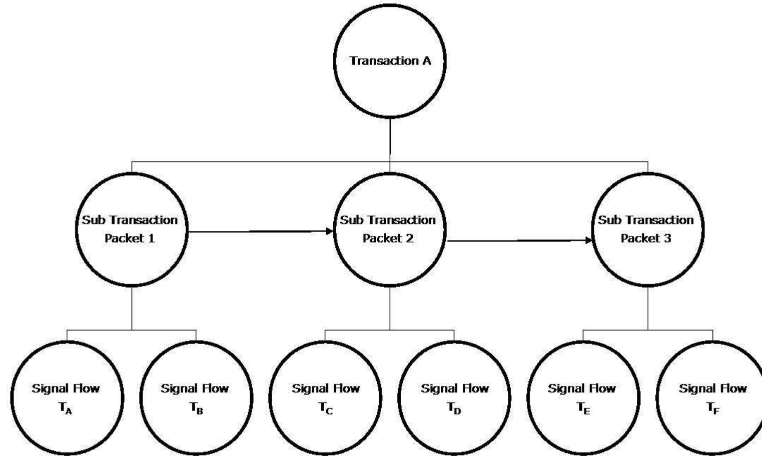

Figure 4.1: Hierarchical Transaction Level Model

that contains the instruction being executed in the ROM. These microcode instructions are modeled as transactions in this case as they are at a higher level of abstraction than the signal values. Validating the microcode flow is done by using Microcode Regression and

deep in the hierarchy tree lays the actual implementation at the signal level and validation at this level is done using event-based waveform compare algorithm[4].

4.1 Microcode Regression

Every time the model owner releases a new version of the netlist, it has to be validated against the golden reference model. A set of stimulus is applied to both these models and the behavior is determined accordingly. Microcode flow is extracted from the response to the stimuli by tracking the Program Counter. An Intelligent Microcode regression is performed to compare and track the microcode flow. The regression environment tries to find a maximum match between the golden and test microcode flow. Microcode flow is modeled as a set of instruction-time pairs; time stamps as in Fig. 4.3. and Fig. 4.4. here are needed to

a) Store the time intervals at which each microcode execution takes place.

b) Model a threshold time frame beyond which no microcode match tries to take place.

Let the microcode flow in the golden and test cases can be modeled as M_G and M_T where $M_G = \{ mG_1, mG_2, ... mG_i,,mG_n\}$ and $M_T = \{ mT_1, mT_2, ,mT_i,...,mT_n\}$. Each microcode instruction is associated with a time stamp along with it. Let the time at which the golden microcode instruction mG_i is executed be denoted by tG_i and the test microcode instruction mT_i is executed be denoted by tT_i.

Algorithm 1 is used for the Microcode Regression.

Algorithm 1 Microcode Comparison Algortihm

- Compare the microcode instruction of Golden and Test cases mG_i and mT_i
- If mG_i and mT_i matches, store the time slices between which these instructions take place. Mark it as a good time slice.
- If mG_i and mT_i don't match, store the time slices between which these instructions take place. Mark it as a bad time slice.
- Perform an event based signal comparison only during the time slices at which there is a microcode match.

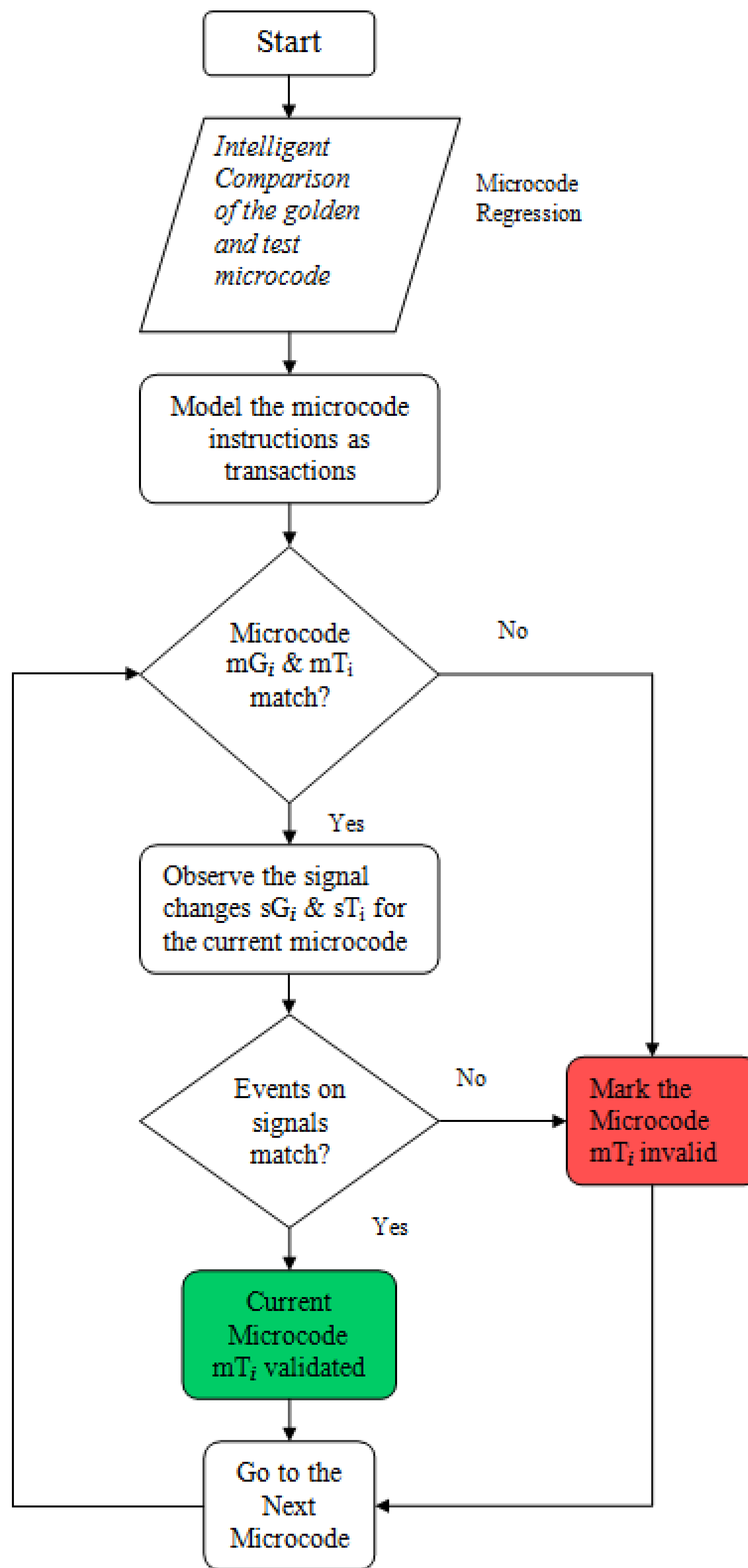

Figure 4.2: Microcode Comparison Algorithm

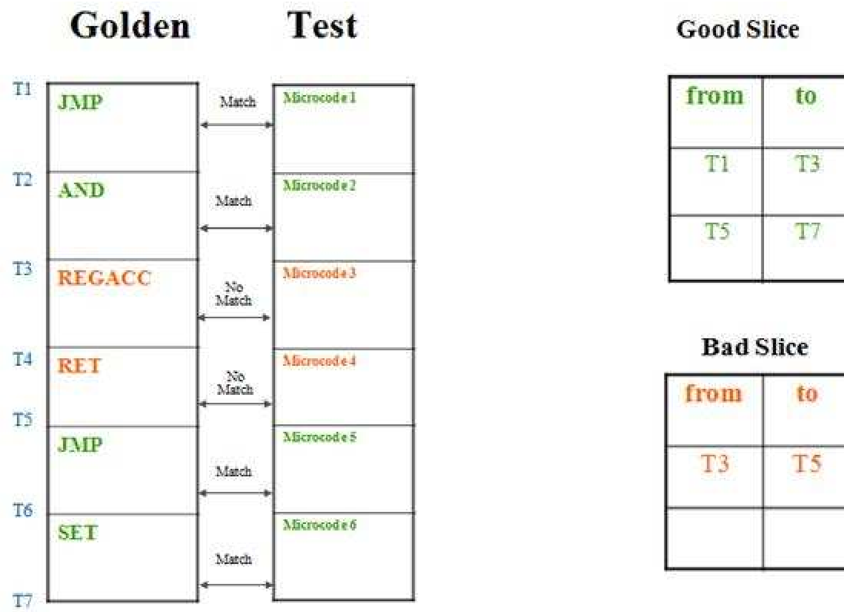

Figure 4.3: Determining good slices and bad slices

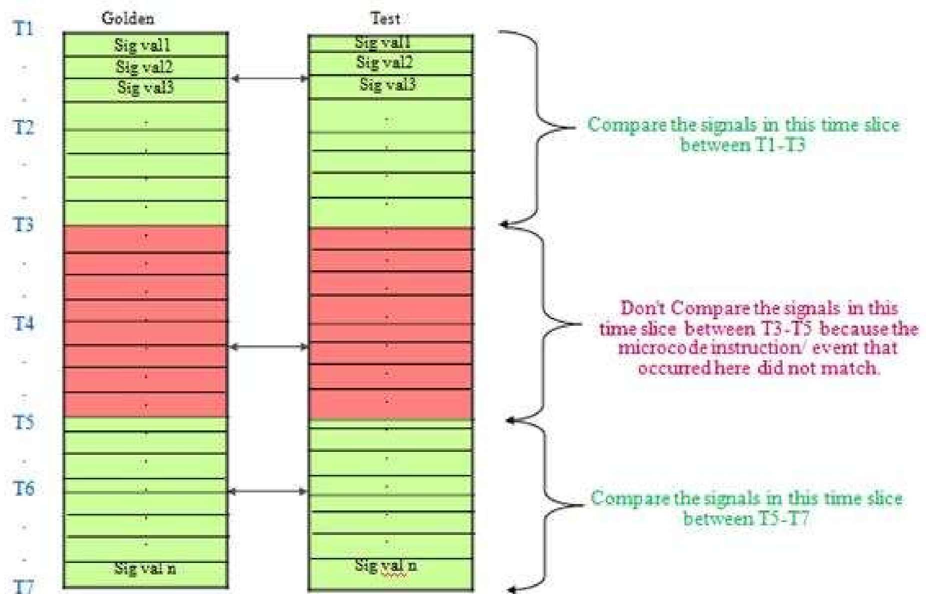

Figure 4.4: Slicing based comparison

4.2 Comparison at a glance

We have modeled a two-level hierarchical verifier in our regression environment, where the "microcode" is at the Transaction-Level and the "signals" are at the Register Transfer Level.

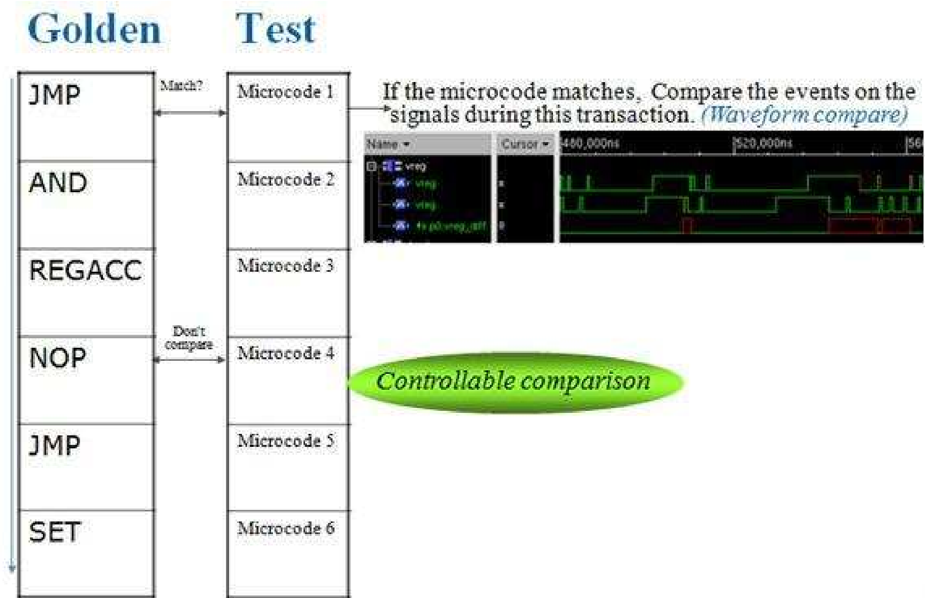

Figure 4.5: Overview of the two-level comparison

4.3 Controllable Comparison

By "controllable comparison" we mean that the verification engineer can control which microcode instructions to compare. Instructions like "NOP", "HALT", etc. need not be compared based on the verifier's configuration as in Fig. 4.5. The verifier can configure which signals to compare during each transaction. Instructions like "SET", "RST", the verifier chooses to compare only the register values and in instructions like "JMP", the verifier chooses to compare the program counter values. This controllable comparison will yield in faster comparison by eliminating unnecessary and redundant checking of irrelevant signals for each microcode.

Chapter 5
CONCLUSION

The Transaction-level verifier has been designed in PERL scripting language. The script looks for the simulation results of the golden and test netlists and reports where there is a mismatch in the microcode flow or signal values. The entire process is automated and hence the turnaround time for validating each netlist revision is reduced greatly. In the event of a mismatch in the simulation results, the verification engineer can identify which portions of the microcode flow is not being validated and take measures appropriately.

The simulation is perfomed in $NC - Verilog^{TM}$ [21], microcode comparison is done using the Transaction-Level verifier in PERL scripting language [20] and the signals are compared using the EventCom waveform comparison tool that uses $SimVision^{TM}$ [22] database.

The Transaction-level verifier has been designed and tested on a previous version of the project that has been taped-out and has yielded good results in terms of time and performance. This verifier will be used in an ongoing project with minor modification in the script to validate design revisions faster and result in a high-quality tape out on time.

In future, we can have additional features to have the capability of viewing the transactions in the waveforms. This will enable the verification engineers to graphically visualize the transactions and signals simultaneously in the waveform window.

Bibliography

[1] T. Grtker, S. Liao, G. Martin, S. Swan, “System Design with SystemC”, Springer, 2002, ISBN 1402070721.

[2] R. Sandip, B. Jayanta, “Abstracting and verifying flash memories”, Non-Volatile Memory Technology Symposium, 2008. NVMTS 2008.

[3] H. K. N. Leung and L. White, “Insights into Regression Testing”, Proc. Conference on Software Maintenance, pp. 60-69, October 1989.

[4] A. K. Singh, “EventCom: A Transaction-Based Waveform Comparison tool”, Proc. of Design and Test Technology Conference, 2007.

[5] S. Swan; “SystemC transaction level models and RTL verification”, 43rd ACM/IEEE Design Automation Conference 2006.

[6] A. H. H. Ngu, “Conceptual transaction modeling”, IEEE Transactions on Knowledge and Data Engineering, Dec 1989.

[7] F. Ghenassia, L. Maillet-Contoz, “TLM Concepts and Applications for Embedded Systems”, Ch. 2: Transaction-Level Modeling: An Abstraction beyond RTL, SpringerLink, ISBN: 978-0-387-26232-1.

[8] T. Bultiaux, S.Guenot, S. Hustin, A. Blampey, J. Bulone, M. Moy, “TLM Concepts and Applications for Embedded Systems”, Ch. 5: Functional Verification from the TLM Perspective, SpringerLink, ISBN: 978-0-387-26232-1.

[9] S. P. Goldman, L. M. Mohr , D. R. Smith, “Using microcode in the functional verification of an I/O chip”, IBM Journal of Research and Development, v. 49 n.4/5, pp. 581-588, July 2005.

[10] M. Chiodo, P. Giusto, A. Jurecska, H.C.Hsieh, A. Sangiovanni-Vincentelli, L. Lavagno, “Hardware-Software Codesign of Embedded Systems”, in IEEE Micro JNL Volume 14, Issue 4, pp. 26-36, Aug. 1994.

[11] M. D. McKinney, “Integrating Perl, Tcl and C++ into simulation-based ASIC verification environments”, Proc. Sixth IEEE International High-Level Design Validation and Test Workshop, 2001.

[12] A. Bernstein, M. Burton, F. Ghenassia, “How to bridge the abstraction gap in system level modeling and design”, IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.

[13] A. Clouard, K. Jain, F. Ghenassia, L. Maillet-Contoz, J. Strassen, "Using Transaction-level models in SoC design flow", in SystemC: Methodologies and Applications, W. Mller, W. Rosenstiel, and J. Ruf, Eds., pp. 2964, Kluwer Academic Publishers, USA, 2003.

[14] W. D. Brown, J. D. Brewer, "Nonvolatile Semiconductor Memory Technology, a Comprehensive Guide to Understanding and Using NVSM Devices", IEEE Press New York. 1998.

[15] P. Cappelleti, "Flash Memories" , Kluwer Academic Publishers, ISBN 0-7923-8487-3, 1999.

[16] J. M. Rabaey, A. P. Chandrakasan, B. Nikolic , "Digital Integrated Circuits: A Design Perspective", Prentice Hall Edition, ISBN-13: 978-0130909961, 2003.

[17] R. Bez, E. Camerlenghi, A. Modelli, A.Visconti,"Introduction to flash memory", in Proceedings of the IEEE, pp. 489 - 502, Volume 91, Issue 4, April 2003

[18] http://www.data-io.com/pdf/NAND/MSystems/MSystems_NOR_vs_NAND.pdf .

[19] R. L. Schwartz, T. Phoenix, B. D. Foy, "Learning Perl", O'Riley and Associates, ISBN-0-596-10105-8.

[20] "Perl version 5.10.0 documentation", http://perldoc.perl.org/

[21] "Product Documentation for Cadence NC-Verilog Simulator", 2007.

[22] "Product Documentation for Cadence SimVision Reference Manual", 2007.