2011-01-01

Accelerated Simulation And Testing Of Integrated Circuits Using High Performance Computing Machines

Matthew C. Markulik
University of Texas at El Paso, mcmarkulik@miners.utep.edu

ACCELERATED SIMULATION AND TESTING OF INTEGRATED CIRCUITS USING HIGH PERFORMANCE COMPUTING MACHINES

MATTHEW MARKULIK

Department of Electrical and Computer Engineering

APPROVED:

Eric MacDonald, Ph.D., Chair

John Moya, Ph.D.

Ryan Wicker, Ph.D.

Benjamin Flores, Ph.D.
Interim Dean of the Graduate School

Dedication

This thesis is dedicated to my family, friends and advisors.

ACCELERATED SIMULATION AND TESTING OF INTEGRATED CIRCUITS USING HIGH PERFORMANCE COMPUTING MACHINES

by

MATTHEW MARKULIK, MSEE

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

THE UNIVERSITY OF TEXAS AT EL PASO

December 2011

Acknowledgements

I would like to thank my advising professor, Dr. Eric MacDonald, for all of the guidance and practical perspective that he has shown me during each stage of my research. Having the opportunity to study under his guidance and mentorship has been a privilege and has expanded my view of the electrical engineering field.

I would also like to thank Dr. David Richie of Brown Deer Technologies for all of the dedication and knowledge he has brought to this research effort. The ability to work with a colleague of this caliber has helped to shape my engineering knowledge and thought process to a more optimal and efficient train of thought.

My final testimony of gratitude is to my family and friends for all of their support and encouragement over my scholastic career. The hectic schedule of a university student is a difficult one to put up with but my family and friends have continued to show their support throughout these hard times.

Abstract

As Moore's Law continues to hold true and transistor density becomes exponentially larger the need to rely on computer aided design (CAD) simulators has become more relevant and necessary. As we dive into the age of electronic reliance there presents a need to constantly improve methods to reduce time-to-solution for developing technologies. The reliance on CAD simulators for electronic development leaves engineers with an ability to improve the design process by improving these simulators or improving the methods in which these simulators are being utilized. With the advancement of quality fabrication processes and the ability to operate at sub-threshold levels the need for high fidelity simulations is ever present.

The digital circuit design process begins with a behavioral description of the intended circuit and is followed by a CAD simulator testing of the desired behavior. These initial circuit simulations are evaluated using a method of testing that does not account for all of the physics involved in transistor behavior but only take into account the average delays and behavior of generally manufactured transistors for a given manufacturing process. For the use of sub-threshold and related technologies these behavioral simulations cannot capture the needed physics level performance of each transistor to verify the functionality and operation of the design. For these digital circuit designs we must rely on higher fidelity simulators called Simulation Program with Integrated Circuit Emphasis (SPICE).

SPICE simulators are the paramount programs for digital circuit design testing and verification. These high fidelity simulators mimic the behavior of actual circuits to a degree of accuracy that rival prototype testing. The increase in accuracy also increases the simulation time, this increase in simulation time reduces the time-to-solution for the design. The present work generates a proposal for reducing time-to-solution while continuing to use the high fidelity circuit simulators by utilizing high performance computing machines. The increase in parallelization capabilities of modern SPICE simulator software and the ability to use multi-core

and multi-processor machines to run multiple simulations concurrently has the potential to further optimize the digital circuit design process.

The overall goal of this process is a set of Integrated Circuit (IC) design tools and methodologies that are scalable and extensible over all high performance computing platforms. This project is focused on design challenges that are outside the mainstream of commercial and open source tools (sub-threshold and extreme low-power designs). This objective will have the most significance on large scale designs and those that the initial modeling and design quality will greatly impact the quality of the final design.

The end objective of Markov Chain Monte Carlo (MCMC) parallel simulation of the Marine Corps TRSS magnetic field detector detection algorithm was preceded by testing of applicable sub-circuits within the detection algorithm digital circuit. Initial testing was conducted with a 64 bit Counter focusing on the energy per operation status over a design space that varied the V_{dd} of the circuit. Analysis showed that through 5,080 concurrent simulations in the Monte Carlo sampling, 4,469 produced correct operation under an acceptance rate of 75%. Of the simulations that produced correct results, an analysis of energy per operation as a function of V_{dd} was conducted and it was shown that two optimal V_{DD} values produced the lowest energy per operation while maintaining functionality. This analysis was completed in just under 7 hours utilizing high performance computing machines, had these simulations been conducted on common dual core workstations the run time would exceed 20,000 hours. The 99% reduction in simulation time of this simple circuit illustrates significant advantages of using high performance computing machines in the analysis of more complex systems especially those focused on low power, sub-threshold designs.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Motivation

As a consequence of Moore's law, transistor density of modern integrated circuits has grown so extensively that engineers are required to rely on computer aided design simulators, this requirement creates a major bottleneck in the design process. As a response to this growth of transistor density and the need for super low power electronics the use of parallel simulators has become a necessity in the design process. The emergence of the need of parallel simulators and simulation techniques has led to the development of the current research objective which has the potential to reduce the time-to-solution and cost of the integrated circuit design process.

The intent of the proposed work is to study the parallelization and scalability of high fidelity simulators and provide engineers with methods and tools in the area of microelectronic circuit modeling and design that utilize high performance computing machines to solve difficult problems that would be intractable without these resources. The goal is to provide a set of techniques and tools that utilize multiple-core and multiple-processor systems in order to employ a method of concurrent simulation that reduces the time-to-solution of the integrated circuit design process while maintaining high fidelity results.

Historically, the software developed in this area focused on workstation-scale computing most typical across the commercial industry. This focus has changed in recent years with the availability of more scalable software and workstations utilizing multiple-core processors. Presently the use of newly developed scalable software on high performance computing machines is limited to DoD research areas such as the current work. Applying the resources

within the high performance computing machines would generate drastic reductions in the simulation time of a large design space experiment requiring the use of high fidelity SPICE simulators.

The use of SPICE simulators on large-scale designs, (10,000+ transistor count) without extended simulation run times, requires a method of testing that allows for the capability to use concurrent math solvers. Math solvers that conduct their mathematic calculations using matrix algebra allows for the use of multiple-core/processor machines in order to solve multiple matrices contemporaneously. The ability to solve many matrices simultaneously allowing for multiple parallel transistor calculations to be achieved concurrently reduces the simulation time for each test bench generated. Combining the use of parallel math solver simulators and simulating multiple test benches simultaneously allows for engineers to converge their designs to an optimal circuit representation in a reduced time.

The present work focused on simulating large scale circuit designs, on the order of 10,000+ transistors, in these high fidelity simulators providing more accurate and reliable operational data. Utilizing high performance computing machines to simulate the large designs in high fidelity simulators allows for multiple test scenarios to run concurrently providing a widespread database for analysis in a drastically reduced timeframe. The present work focused on the ability to manipulate several design metrics at the SPICE level and perform a post simulation analysis in order to converge on (an) optimal operating parameter(s) for the detection algorithm of U.S. Marines' Tactical Remote Sensor System (TRSS).

The Tactical Remote Sensor System comprises of a novel fluxgate magnetometer that allows reconnaissance patrols to image armed individuals, even through a wall, and the system

can also be deployed in remote areas as an unattended ground sensor for persistent surveillance. Although this magnetometer does not require high performance control circuits, the necessary electronics must operate on close to zero power based on energy constraints imposed by using batteries. One possible solution is to run the digital electronics at very low voltages, far below traditional levels, as power is quadratically related to the supply voltage. The sub-threshold logic is gaining attention particularly in the DoD for providing significantly reduced energy per operation for applications in which high circuit performance is a secondary consideration, such as unattended ground sensors.

The detection circuit of the magnetometer contains the majority of the digital circuitry and complexity of the TRSS ASIC and as such was deemed the proper candidate for the current work. The work effort began with scale models of the detection algorithm, utilizing key features of the design in order to build upon successful simulations and ultimately simulate large-scale designs such as the full detect algorithm. Beginning with a basic counter of varying scale, i.e. from 8 bit through 256 bit counters, this project was able to show a reduction in simulation time with the use of a commercial tool, HSPICE, and an open source tool, NGSPICE. The capabilities of each simulator vary and thus a commercial baseline tool and an open source tool were utilized as experimental features. The results of these simulation times with respect to transistor count became dramatic as the design size and complexity increased. A basic 256 bit counter showed a 51% decrease in simulation time when utilizing the multi-core option of HSPICE.

Results of the high fidelity SPICE simulation using a 64 bit counter (Appendix A5) and varying input metrics such as V_{dd} showed proof of concept of simulating large-scale concurrent design while testing a large design space in order to converge on optimal system parameters and verify circuit functionality over changing environmental conditions. Evaluation of energy per operation with a scaling V_{dd} was deemed to be a favorable test factor for initial experiments of

the 64 bit counter. Results of this test illustrated that through 5,080 concurrent simulations there existed two optimal design configurations for low power operation. The use of high performance computing machines greatly reduced the simulation time and achieved the desired results of converging on a favorable low power configuration.

1.2 Organization of Thesis

This thesis focuses on the design of a methodology that reduces the time-to-solution of other than mainstream IC and its organization is as follows:

Chapter 2 focuses on previous works in the area of parallelized Verilog and SPICE simulators as well as a basic background into the simulators used for this project.

Chapter 3 describes the methods and approach taken to achieve the proposed goals and gives a more in depth view of all the previously discussed material.

Chapter 4 presents the results achieved during simulations and explains the courses of action that needed to be altered in order to achieve the desired results.

Chapter 5 discusses the overall conclusion of the research and provides thoughts on future work.

Appendix provides the scripts and code that were relevant to achieving the goals.

Chapter 2: Background

2.1 Previous Work

Previous work will focus on those works that presented data on the parallelization and scalability of Verilog and SPICE simulators and simulations. Due to the scarce amount of research presented in this area minimal previous work information will be provided.

2.1.1 Verilog

The history of Verilog is an extensive and rocky avenue of engineering that incorporated the efforts of several leading design corporations and engineers. The true beginning of a standardized Verilog simulator began with the Cadence sponsored Open Verilog International conference in 1991 where cadence, then the owner of Verilog, decided to release the language as an open source tool to allow for IEEE standardization. Upon the completion of the IEEE 1364 Verilog standardization in December 1995 multiple companies started developing more advanced and capable simulators utilizing the new Verilog XL algorithm (an efficient method for doing gate-level simulation).

The inherent design of Verilog simulators allows parallel simulation. As the main focus of Verilog simulators is that of time propagation and signal dependencies parallelizing the simulation of these calculations is intrinsically elementary.

2.1.2 SPICE

The development of SPICE software was a spinoff of several industry used tools that maintained only specific uses such as BIAS that only conducted analysis of bipolar junction

transistor operating points and SLIC that only performed small signal analysis. As the evolution of SPICE grew throughout the 1970's and 1980's engineers began to realize that there was a need to develop a decomposition algorithm to employ a parallel SPICE program. In 1991 Richard Chen published a journal on running SPICE in parallel utilizing a decomposition algorithm he developed which provides a basis for the current work.

The ability to run SPICE level simulations in parallel is one that relies completely on the math solver for that simulator. Utilizing matrix algebra to describe the integrated circuits allows for the ability to compute many transistor parameters concurrently and converge to a single state at a given point in time for each transistor in the design. Running multiple matrix calculations simultaneously greatly reduces the time multiple-core/processor machines need to generate the same data that a single-core/processor machine would.

2.2 Simulators

Stages of simulators that accomplish separate goals have been used in this design process. The two stages of IC simulation include Verilog and SPICE simulators. Verilog simulators are used to test the design behavior for basic functionality and operation to ensure that the desired behavior is achieved and the overall goal of the design is met. For more in depth simulations of IC design an engineer would turn to SPICE simulators. The use of SPICE simulators in modern design methodology is minimal due to the extended simulation time needed to calculate the physics level behaviors of each transistor at each step in time. Regardless of the level of simulation there are many options for simulators at each level. The following will explain the basic difference between SPICE level and Verilog level simulators and differences between the chosen simulators for this project.

2.2.1 Verilog

Hardware Description Language (HDL) is the basis of most digital systems in industry today, having the ability to describe a system quickly and accurately through either of two languages, Verilog and VHDL, allows for quicker design and more practical testing procedures than prototyping. Register Transfer Level (RTL) is the most common use of HDL languages allowing for a level of abstraction in circuit design that describes the flow of signals or data that is transferred within the system. Upon completion of an RTL code the next step would be to utilize a Verilog simulator to test the desired behavior and ensure proper functionality of the circuit designed. A major advantage of Verilog simulators is that they, generally, are efficient testing tools regardless of the fabrication technology that will be used due to vendors including ASIC libraries in their tools. Described in the following subsections are the two Verilog simulators used for this project, the first being a commercial tool that was used for baseline and benchmark operations and the second an open source instrument used as the project simulator called Veriwell.

2.2.1.1 NC-Verilog

NC-Verilog is a tool provided by a leading CAD tool developer named Cadence. Cadence has an extensive history in the development and sale of commercial CAD tools and NC-Verilog is one of its forerunning Verilog contrivances. This simulator provides high speed and high capacity Verilog simulation with the capability to provide transaction level signal analysis utilizing SimVision by Cadence.

As with all commercial Verilog simulators NC-Verilog complies with all IEEE 1364-1995 standards as well as most of the IEEE 1364-2001 Verilog standards. Keeping these compliance standards aids in the ability to unify code syntax and development procedures in order to provide more comprehensive and higher performance simulators. Although NC-Verilog

maintains all IEEE standards it also pushes the envelope when it comes to performance making it the optimal choice for the benchmark simulator used in this project.

2.2.1.2 VeriWell

Since the use of NC-Verilog requires licensing for every two cores or cpu's utilized, its use on the high performance computing machines was immediately deemed impractical. The sheer number of processing units within one of the high performance computing machines required either an open-source or freeware simulator that had the capabilities to compare to commercial simulators results. These demands led us to a single option for open-source comprehensive Verilog simulators, VeriWell.

VeriWell was originally a commercial Verilog simulator in the early 1990's before IEEE standardized the syntax and construction process. Being an early Verilog language, the support of IEEE 1364-1995 standards is limited to the open-source development teams' time and capabilities. Although VeriWell does not fully support the IEEE standards its capabilities to simulate the RTL code generated for this project was not impacted and after initial testing was deemed to be an adequate experiential tool for this project.

The following figure illustrates the comparison of NC-Verilog to VeriWell utilizing increasing sizes of counters. This illustration identifies the capabilities of each simulator without accounting for setup times of each simulator. The increased time of the NC-Verilog simulator is due to load times and not that of the simulation process. It should be noted that although the increased setup time of NCVerilog shows a slightly slower simulation time with respect to VeriWell the NCVerilog simulator is a more stable tool and the optimal option for Verilog simulations.

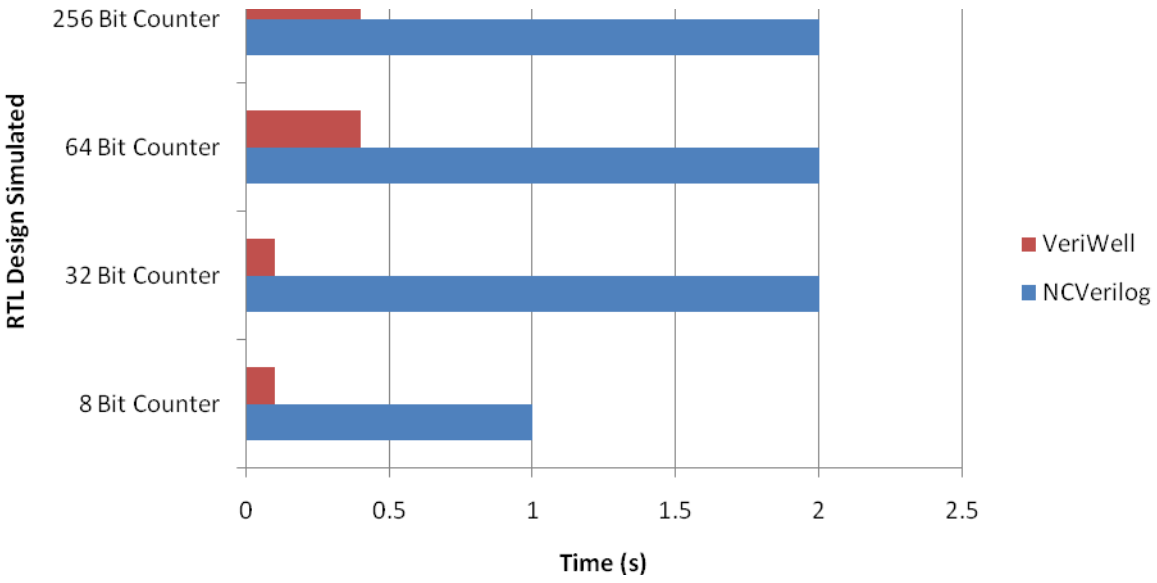

Figure 2.1 Commercial vs Freeware Verilog Simulators

2.2.2 SPICE

SPICE (Simulation Program with Integrated Circuit Emphasis) is a general-purpose open-source analog electronic circuit simulator. This is a powerful program that is used in integrated circuit design to check the integrity of circuit and predict circuit behavior. SPICE was originally developed by the University of California at Berkeley, and the programming syntax and device models are used as the basis of every commercial simulator available today and the original source code (version 3F5) is the cornerstone of a variety of open-source versions as well. Although the source code has been frozen at version 3F5 for the simulator, progress in device models is on-going at the University of California at Berkeley through collaboration with industry in order to capture recently understood short channel effects that appear in next generation contemporary transistors. These models (BSIM3.3 - Berkeley Short-channel Igfet Models) remain the standard for all circuit simulators.

SPICE simulations are rigorous and utilize detailed models of the transistors used to physically construct the logic of a large digital circuit and are specialized to capture the specific behavior for a given semiconductor technology node that will eventually be used fabricate actual devices. The higher fidelity modeling, relative to simple RTL simulation, coincides with longer simulation run times. Longer run times mean longer time-to-solution. Although having a more effective simulation allows for better analysis of circuit behavior under varying conditions, the amount of time taken to explore these varying conditions can amount to weeks or months of just testing.

Strides have been taken in recent years to improve simulation time by allowing multi-threading and multi-processor use during simulations. Since the majority of SPICE simulation time is spent evaluating model parameters for every transistor in the circuit allowing SPICE to evaluate multiple matrices at one time by multi-threading has the potential to greatly reduces simulation time. SPICE must calculate transistor model parameters for each transistor at each transient step in order to arrive at the high fidelity solutions for each step in time. With varying levels of transistor models the accuracy of each simulation will vary. Transistor models include many factors such as intrinsic capacitance, channel dimensions, junction sizes and additional relevant transistor properties. Each transistor model is specific for a target silicon production size, e.g., 250nm channel length.

SPICE simulators represent a class of simulation software derived from the original open-source Berkeley program, and there are many simulators to choose from ranging from commercial packages to free and open-source simulators developed by various organizations. With the range of available simulators, there is also a range in capability and advanced features, as well as subtleties in their interfaces and input file syntax.

2.2.2.1 HSPICE

HSPICE is the Synopsys SPICE tool and the industry standard for SPICE level circuit simulation tools. With over 25 years of development and utilization HSPICE has become the most comprehensive and robust circuit simulator in industry. As workstations move toward multi-core/processor systems the need for a multi-core enabled circuit simulator is a must and that is what HSPICE delivers with the ability to utilize up to 8 cores per simulation. For the current work the use of a multi-core enabled simulation is ideal not only for final effect but in order to reduce research time by providing quicker results without any loss of data accuracy. The design of HSPICE is one that has focused in the area of scalability and design for yield resulting in a simulator that flows well with the current works use of the Monte Carlo approach to decrease time-to-solution.

2.2.2.2 NGSPICE

NGSPICE is an open-source circuit simulator that has come from the compilation of three circuit simulation tools. The basis of NGSPICE, and all SPICE simulators, is the U.C. Berkeley SPICE code Spice3f5. The Spice3f5 circuit simulator tool is the latest stable version of U.C. Berkeley SPICE code that utilizes the BSIM3v3 and higher device models allowing for parallelization and higher transistor modeling accuracy. The next tool used in the development of NGSPICE is CIDER, a mixed-level circuit and device simulator that provides a direct link between technology parameters and circuit performance. A mixed-level circuit and device simulator can provide greater simulation accuracy than a stand-alone circuit or device simulator by numerically modeling the critical devices in a design netlist.

Overall the functionality and simulation results of NGSPICE are comparable to the commercial tool HSPICE. With minor syntactical differences the ability to import an HSPICE netlist to NGSPICE is a trivial matter once the syntax differences are learned. The following

graph illustrates the performance difference between the commercial and open-source tools on a logarithmic scale with varying levels on transistor density.

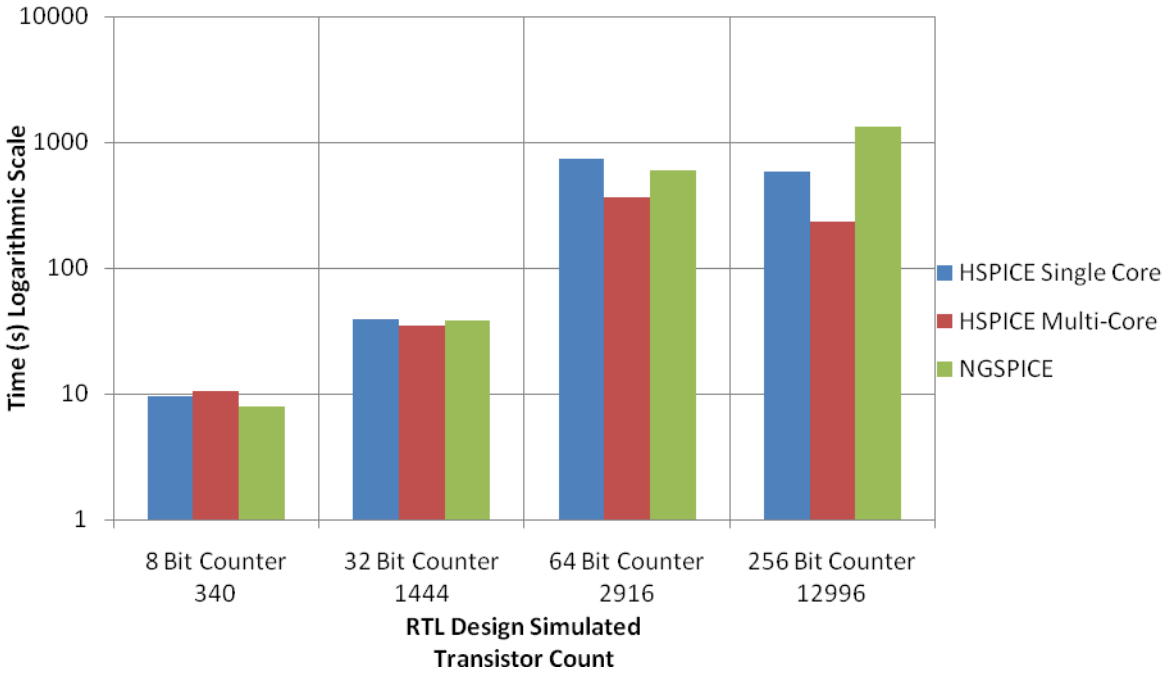

Figure 2.2 Commercial vs Freeware SPICE Simulators

Chapter 3: Implementation

3.1 Design Methodologies

Design methodologies over the past three decades have evolved to include simulation with tiers of fidelity, from devices to circuits to logic to architecture to software. The higher the accuracy required, the lower the limit of circuit size that could be simulated. To simulate larger circuits, levels of abstractions were necessary to simplify the simulations to allow modern computers systems to manage the simulation and corresponding memory and processor requirements.

The implementation of sub-threshold circuits with corresponding exponential dependencies on variations require that large scale designs be simulated with high fidelity SPICE simulators. This effort focused on utilizing the Markov Chain Monte Carlo (MCMC) simulation technique in order to take full advantage of the large number of cores accessible in the high performance machines. The first phase of this effort concentrated on implementing the MCMC technique on the Verilog design script of the detection algorithm. The second phase of the project revolved around designing a tool that made it possible to automate the netlist conversion process thus allowing for a quick and practical method to convert a Verilog netlist to a SPICE syntax netlist. Upon the completion of the automated netlist conversion tool the final phase delved into simulating large scale SPICE netlists in high fidelity simulators on high performance computing machines.

3.2 Verilog MCMC Approach

The Verilog design comes from a mature Advanced Magnetic Detector system developed for the Marine Corps by SSC_Pacific and has stringent power requirements. Utilizing a mature Verilog design eliminates the precursor issues of functionality and operation allowing for advancement to the simulation process and testing.

Developing an MCMC process with a choice Verilog simulator required the development of several coexisting tools in order to test a large design space over thousands of simualtions concurrently. Each individual simulation required developing a testbench that varied a set of input parameters allowing for a controlled experiential design space. The ability to generate such controlled random testbenches required the development of a pre-processing (Appendix A2) script that would take random inputs from a master code (Appendix A4). The MPI master code would pass arguments to the pre-processing script that would in-turn output a testbench utilized by the simulator for circuit stimulus. This process allows for a controlled test space and the ability to analyze the output data and feed it back to the input for a dynamic test facility.

The following figure illustrates the simulation process from the MPI code through the output analysis, this figure allows for a visualization of the dynamic process flow of the HPC simulation process.

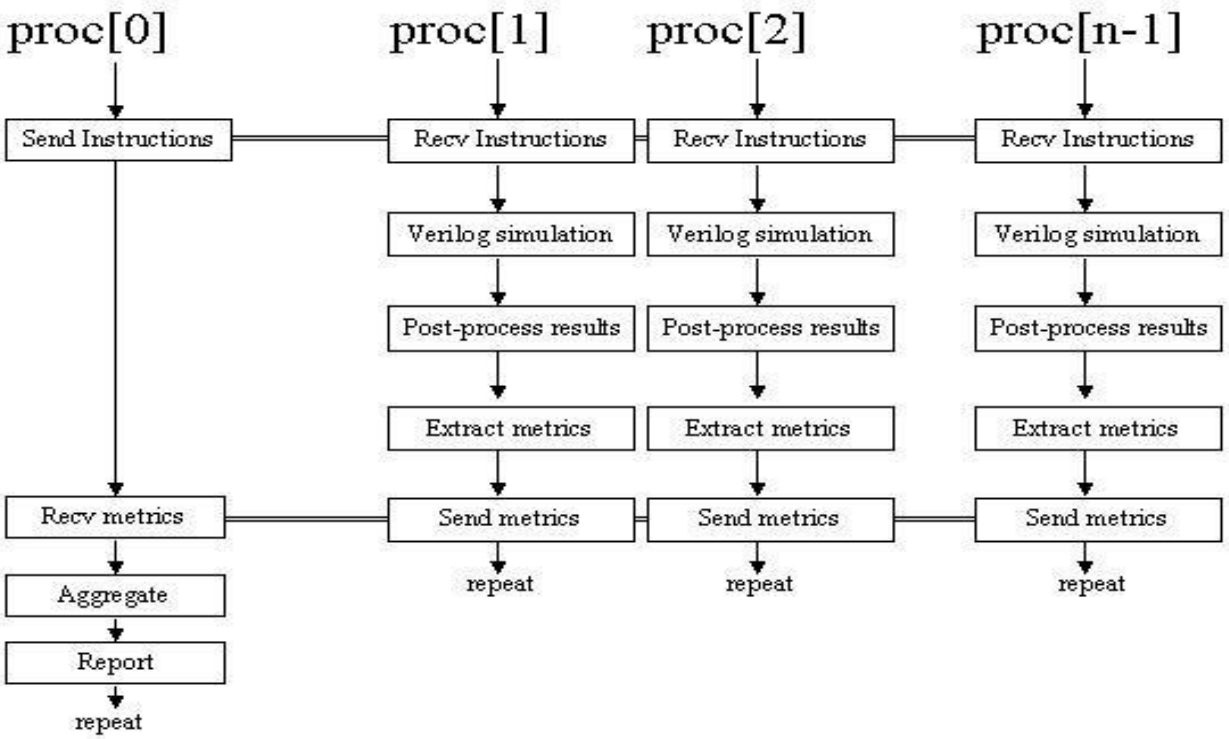

Figure 3.1 Verilog MCMC Simulation Approach

The MPI run is first launched with *n* processes and a simple master-worker model is used to initiate repeated cycles of *n-1* simulations. The master process, *proc[0]*, first sends instructions to each of the workers, *proc[1] ... proc[n-1]*, which can be used to control the individual simulations for a given iteration. Each worker process then generates a unique Verilog test-bench for the simulation. Some parameters are generated so as to explore state space following the standard Metropolis algorithm for generating a Markov Chain Monte Carlo simulation. The random selection utilizes SPRNG, which is a parallel random number generator designed for accurate large-scale parallel simulations and suitable for Monte Carlo studies. Upon completion, a post-processing script (Appendix A3) is run to extract information from the simulation used to evaluate a metric to determine the quality of the results. The results from individual workers can be packed and sent to the master MPI process. The master MPI process can aggregate the results at which point they can be used to alter the next round of instructions

for the next cycle. Thus, the design allows for feedback and control from the master process suitable to Monte Carlo studies.

3.3 SPICE Netlist Extraction

Whereas the conventional workflow involving SPICE simulation involves constructing a representative element of a larger circuit from which parameters are to be extracted, here we are targeting the SPICE simulation of a much larger and complex circuit. Moreover, the definition of this circuit is given in the form of an RTL Verilog specification. This introduces the requirement for a non-trivial step that must be automated whereby the Verilog netlist generated as part of a conventional synthesis process is mapped to a representation suitable for SPICE simulation at the transistor level.

Synthesis is an RTL (register Transfer Level) tool that generates a gate-level representation of an abstract Verilog script. Synthesis tools use standard cell libraries, a list of logic gates and their corresponding metrics, in order to hardcode the available logic gates into a netlist that is equivalent to that of the abstract Verilog code. In order to employ the synthesis tool the development of a synthesis script that outlines items suchs as the standard cell library, HDL design, timing constraints and output files is needed. Once the synthesis tool completes its calculations and generates a logic-level circuit it prunes any logic not feeding a next stage thus genrerating an optimized netlist.

Extracting a SPICE level netlist from a synthesized Verilog netlist is a non-trivial procedure that requires in-depth knowledge of both the synthesis tool and the SPICE simulator. Understanding how the synthesis tool contructs a viable netlist is key to generating a tool that

reworks this netlist into a SPICE suitable option. Synthesis generates netlists that utilize call-names for nodal connections such as input(.ip in Verilog), output(.op in Verilog) and clock(.ck in Verilog). The use of such call-names interprets into a netlist that does not require any order of the inputs, outputs and clocks for each sub-circuit. A comparison of nodal attachments for SPICE netlists shows that the hierarchical structures require an exact order of the inputs, outputs and clocks matching the standard cell library logic. Noting the difference in ordered vs non-ordered nets of each simulator presents a significant challenge for SPICE netlist extraction. Development of a tool capable of automating the netlist extraction process was implemented in PERL (Appendix A1). The output of this tool was verified over several SPICE simulators to include the commercial HSPICE tool and the freeware NGSPICE and SpiceOpus simulators. Verification of a common syntax among the three leading simulators was conducted and simulator-specific options were respected in order to confirm correct operation of the automatically generated netlist with required parameters.

Whilst developing the tool and technique for a functional SPICE netlist from the synthesis process, there arose a need to develop several reduced circuit designs that captured varying levels of complexity and features from the original design in order to reduce simulation time and testing.

3.4 SPICE Netlist Testing and Development

The essence of this project focused on simulating large-scale digital designs in high fidelity SPICE simulators using high performance computing machines. Immediatley upon initial testing of the full detect algorithm it was determined that there was a need to develop a

reduced toplogy that would retain critical features of the original netlist. Upon identification critical components of the detect algorithm key features were manipulated in order to reduce transistor count and simulation time while maintining the basic functionality of the original design. Reconfiguring the reduced algorithm from an asynchronous design to that of a synchronous one allowed for drastic simulation time diminution. The final stage of reducing the original design began with reducing the complexity of several modules within the design. This was accomplished by reducing items such as the number of autoregressive moving-average models used in the design and curtailing the number of wires used by replacing them with registers.

The second phase of SPICE simulation began with thorough research into the operation of these simulators. Large data set designs increase the probability of dc convergence issues due to the number of memory based components within the design. SPICE simulators begin their modeling by determining an initial value for every net within the design and setting values to begin transient analysis. Memory based components that are not initialzed with either a high or low value will be given a high impendance (Z) state. SPICE simulators can handle low numbers of high impedance states if the logic is resettable, if this is not the case or there are large numbers of high impedance components the simulator will fail dc convergence and abort the transient analysis. In order to avoid dc convergence issues it was determined that all memory based components need to be initialized using either the .NODESET or .IC options of the simulators. The design of the current work utilized many flip-flop memory circuits and as such there arose a need to initialize each latch within the flip-flops. As SPICE netlists tend to the hierarchical design method initialization of individual latches was not possible thus the need to *"flatten"* the design was deemed appropriate. Flattening of a SPICE netlist entails removing levels of

hierarchy in order to further manipulate the circuit to meet simulation requirements. By flattening the netlist it led to the capability of using the .NODESET option in order to initialize each latch with a low value and thus giving way to a netlist that would pass dc convergence.

The final stage of netlist testing and development began with exploring the simulation of increasingly complex and transistor dense designs beginning with basic counters through the full detect algorithm. Initial testing was done with basic counters through each simulator to verify syntax and simulator option capabilities. Verifying dc convergence and transient analysis outputs of these basic circuits allowed for tesing of the SPICE MCMC simulation methodology on the larger high performance computing machines.

3.5 SPICE MCMC Approach

Integrating SPICE simualtions into the high performance computing environment followed the previous methodology developed for RTL simulation. Applying the MPI code to incorporate the high fidelity SPICE simulator was trivial since it regarded the simulator as a closed tool just as in the RTL simulation process. The same basic framework was used and is illustrated in Figure 3.2.

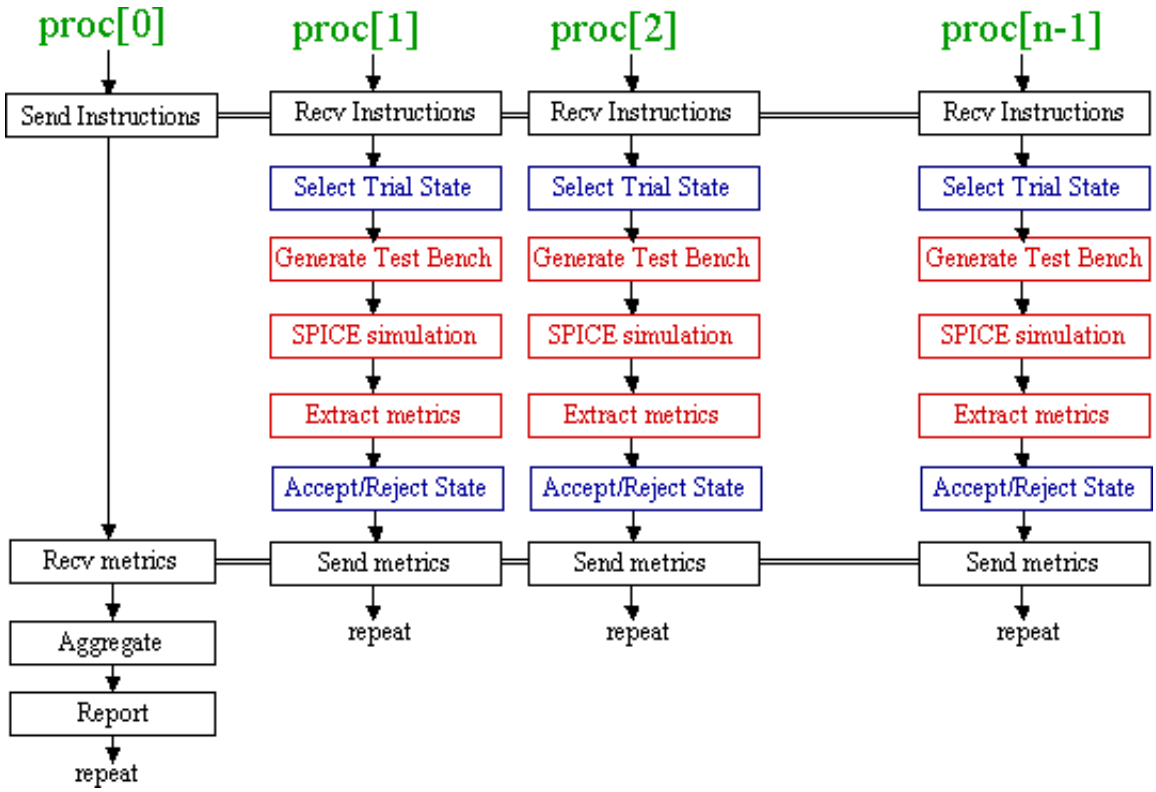

Figure 3.2 SPICE MCMC Simulation Approach

The MPI run is first launched with *n* processes and a simple master-worker model is used to initiate repeated cycles of *n-1* simulations. The master process, *proc[0]*, first sends instructions to each of the workers, *proc[1] ... proc[n-1]*, which can be used to control the individual simulations for a given iteration. Each worker process then generates a unique SPICE test-bench for the simulation, which will include the SPICE netlist and parameters controlling the simulation. Some parameters are generated so as to explore state space following the standard Metropolis algorithm for generating a Markov Chain Monte Carlo simulation. The random selection utilizes SPRNG, which is a parallel random number generator designed for accurate large-scale parallel simulations and suitable for Monte Carlo studies. Once the testbench has been generated, a SPICE sub-simulation is then initiated from the worker process. Upon completion, a post-processing script is run to extract information from the simulation used to

evaluate a metric to determine the quality of the results. The results from individual workers can be packed and sent to the master MPI process. The master MPI process can aggregate the results at which point they can be used to alter the next round of instructions for the next cycle. Thus, the design allows for feedback and control from the master process suitable to Monte Carlo studies.

The key modifications required to adapt the original simulation framework to support SPICE simulation involved replacing the three steps unique to the type of sub-simulations being performed. Specifically, this involved modifying the steps for (1) generating a testbench, (2) executing a SPICE sub-simulation, and (3) extracting metrics from the sub-simulation. Additional modifications were required in the specific implementation of the Metropolis algorithm to account for the state space being explored.

The standard Metropolis algorithm was used to sample the space spanned by V_{dd}, V_{enable}, as well as the transition time and period of the input clock. For each circuit parameter defining the state space of the Monte Carlo sampling, a random change was selected from a Gaussian proposal distribution. These parameters were restricted to the ranges 0 – 2.5 V for V_{dd} and V_{enable} and 20-2000 nsec for the clock period. A metric was required to measure the quality of a given state based on the output of the simulation. The following metric was constructed to reflect a reasonable measure of the quality of the results with a focus on requiring that the circuit operated to completion with a preference for lower energy usage:

$$F = \exp(+10 \cdot [-f_1^2 / 100 - f_2^2 / 500])$$

where f_1 provides a penalty based on circuit failure and is proportional to the $\log_2$ of the clock cycle where failure occurred, and f_2 provides a penalty for energy usage being proportional to the $\log_{10}$ of the total energy consumed in the operation of the circuit.

This metric was used following the standard Metropolis algorithm to determine whether a new state should be accepted or rejected in the course of constructing the Markov chain exploring state space. The developed code represents a demonstration of the type of circuit optimizations that can be performed using large-scale HPC resources. The modifications to the previous simulation framework necessary to apply it to parallel SPICE Monte Carlo simulations encountered several technical challenges that are described in the remaining sections below.

3.6 Pre- and Post-Processing Script Development

The need for a pre-processing script that generates both a Verilog test bench and a SPICE input file that have matching input data presented itself when developing metrics in order to compare the Verilog results to the SPICE simulations. The development of this PERL script began with the design of a random number generator (RNG). With the aforementioned random number generator and a list of dynamic command line inputs the generation of a spice input file along with a Verilog test bench became an issue of format. The inputs accounted for include but are limited to sensitivity, number of data points before enable is set, number of data points after enable is set and standard deviation. The use of these as well as other pre-processing script inputs allows for the simulation of a large state space evaluating multiple metrics simultaneously and letting the Monte Carlo approach converge to an optimal design fixture.

Formatting of the Verilog test bench is a simple task as there exists an industry standard for Verilog syntax. The case of SPICE input files does not follow these same guidelines and the format varies within each simulator. Issues of the SPICE input file began to arise with errors due to exceeding line size and syntactical errors such as next line characters. The research of

differences between the commercial baseline simulators and the open source simulators used in this work led to the development of a neutral input file syntax that allowed for operation within both simulators.

A post-processing script that reads in the log files of both the Verilog and spice simulations was a necessity in order to compare the results of a basic simulator to that of a high fidelity physics based simulator. The major challenge of a post-processing script comparing the output of two different types of simulators is aligning the data by clock cycles and verifying the outputs. Generating a post-processing script requires discarding the many excess data points of the high fidelity simulators in order to compare a single detect value per clock cycle in each log file. Utilizing the basic, transistor on/off, Verilog simulator as a baseline for comparison the challenge of changing the many variables of the large state space of the high fidelity SPICE simulator and evaluating the inputs and the results has become the main focus of the post-processing script. When reading the outputs of the simulators the need to know the inputs has become relevant in order to ensure proper functionality of the design.

As part of the post-processing script development the identification of required metrics is key to successful evaluation of design behaviors. Metric identification and design began with distinguishing the key functionality of the design at hand. For the detection algorithm a measurement of the number of real detects, false detects and a comparison of an intermediate 24 bit value were deemed to be efficient evaluation factors of the design.

Chapter 4: Results

4.1 Overview of Designs Tested

The first test circuit examined was a 64-bit counter, which contains an adder with significant propagation delay to warrant study in terms of reduced voltage and variable clock frequency operation. This circuit uses only fundamental logic gates, e.g., nands, ors, xors, flip-flops and inverters. The netlist for this circuit contains 1602 nodes, 2940 elements and 2936 transistors. Once synthesized using a clock of 40ns the Verilog netlist was converted to a spice netlist using a perl script that coincides with the Virginia tech standard cell library. This netlist was initially verified using HSPICE. After netlist operation was confirmed on the commercial-grade software, simulation was then performed with Ngspice where correct operation and performance comparisons could be made. The results for this extracted SPICE netlist are shown in Table 2 and include synthesis information with the logical gates and transistors required to implement the design**.**

The second test circuit is of a slightly more complex design that allows for use of combinatorial logic and basic logic gates. The single moving average circuit utilizes multiplexers and full-adders as well as the basic logic gates. Applying the same timing requirements as the 64 bit counter on this more robust circuit requires that the basic logic gates be more enhanced to meet the timing requirements. This spice netlist contains 1316 nodes, 2645 elements and 2614 transistors. Although the overall transistor count of this circuit is lower than that of the counter the complexity of the gates makes simulation times longer and device modeling in the simulators more difficult. A modified version of the PERL script was generated in order to convert the Verilog netlist into a spice syntax netlist and this spice netlist was then

verified using HSPICE. Upon verifying the functionality of the spice netlist in HSPICE we then ran the circuit in NGSPICE for a baseline on a workstation.

The largest test circuit generated is the detect algorithm with a hardcoded configuration that allows for easier testability. The detect algorithm is very complex circuit that utilizes the counter and moving average circuits within itself. The complexity of this circuit is ideal for spice simulation as the average and error calculations at the sub-threshold level require a more precise level of fidelity. Within this circuit all levels of logic gates are utilized from basic logic to more advanced combinatorial and cascaded logic. This SPICE netlist contains 8924 nodes, 16871 elements and 16832 transistors making it the largest test circuit we have developed to date. This netlist was also verified using HSPICE and through the working of HSPICE the dc convergence issues that have afflicted the design were measured and corrected. Porting the dc convergence methods into NGSPICE was not a trivial task as the syntax of SPICE simulators is not universal and a rework of the dc convergence issue was researched and implemented in order to generate a working design in NGSPICE. The inherent hierarchical structure of SPICE was deemed to be the main cause of dc convergence issues and was rectified by falttening the designs in order to initialize each latch within the memeory circuits of the netlist.

Simulating the full detect algorithm in SPICE thoroughly tested the expansive mathematical models developed for these high fidelity simulators and tested the comprehensive design parameters of the identified tools. The use of the commercial HSPICE tool allowed for proof of theory that the large design spaces and netlists could be simulated of high fidelity tools and the produce viable, reliable results. Porting designs of this magnitude to the open source tools such as NGSPICE presented tool specific issues that were overcome in the current effort.

Circuit	Logic Elements	Transistors	Simulation Time (number of cores)	
			HSPICE	Ngspice
64-bit Counter	2940	2936	905 sec (6 cores)	2955 sec (1 core)
Moving-Average	2645	2,614	367 sec (5 cores)	1,993 sec (1 core)
Error & Delta Calculations	10,845	10,816	7,186 sec (6 cores)	75,792 sec (1 core)
Rolling Error	10,615	10,586	15,829 sec (8 cores)	80,613 sec (1 core)
Reduced Detect	3,679	3,640	1,675 sec (5 cores)	9,784 sec (1 core)
Full Detect	16,871	16,832	32,451 sec (8 cores)	98,405 sec (1 core)

4.2 Parallel Monte Carlo Simulation

The methodology developed in this work was tested by performing parallel Monte Carlo SPICE simulations for a 64-bit counter. A total of 5,080 SPICE sub-simulations were run on 128 cores of the ARL-DSRC Emperor platform that was used for the initial development and testing. The run time for this simulation was approximately 7 hours. Of the 5,080 simulations generated in the Monte Carlo sampling, 4,469 simulations produced correct operation. The acceptance rate in the Monte Carlo sampling was initially 75%, which is somewhat higher than the 25% acceptance rate recommended for efficient Monte Carlo sampling. The consequence of a higher acceptance rate is one of efficiency insofar as the state space will be explored more slowly than desired. Increasing the width of the trial distributions used to generate new states in the sampling reduced the acceptance rate to 64% with 3,849 simulations showing correct operation. An examination of the autocorrelation functions (ACFs) for the state space parameters, shown in Figure 4.1, exhibited a corresponding reduction in the correlation between samples. The ACFs exhibited the correct exponential form expected for MCMC simulation. However, the correlation for the clock period remained higher than desired, which can be adjusted with further tuning. Nevertheless, the initial simulations showed a good sampling of the operating parameters of the circuit.

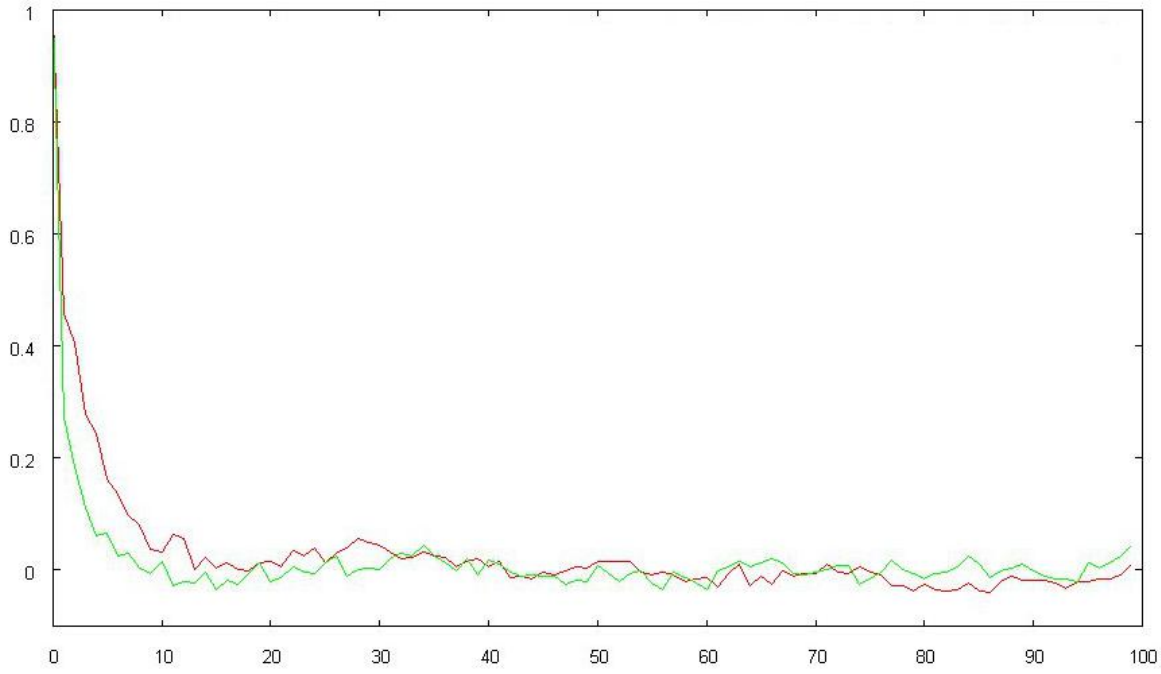

Figure 4.1 Auto-Correlation Function (ACF) for the Monte Carlo samples of V_{dd} (green) and energy (red).

Sub-threshold circuits are of particular interest for applications in which obtaining the minimum energy per operation is paramount, but in which performance is less important (e.g. unattended ground sensors, unmanned aerial vehicles, etc). However, due to the exponential dependence on environmental factors, the traditional design and simulation methodologies of normal electronics are insufficient for sub-threshold circuits and thus require circuit level simulators in place of faster logic level simulators. To overcome the dramatic increase in processing power required by the circuit simulators, we have proposed to leverage the massive parallelism offered by the high performance computing systems.

To create the baseline data to provide a demonstration that the simulators and the Monte Carlo infrastructure worked correctly, a counter circuit in Verilog was chosen as a testbed. Counters are well suited to serve as the baseline as these circuits are easily scaled up in terms of transistor count and include sequential elements as well as full adder cells - both of which are central to the operation of the targeted circuit. The final targeted circuit is a DSP detection

algorithm in a magnetometer control chip being developed by Navy's SPAWAR in San Diego. Counters are complex enough to provide a rigorous test for the methodology while simultaneously being sufficiently regular in structure to allow for reasonable debug on a massive scale in terms of transistor count. The circuits can be described in Verilog hardware description language as is the case with contemporary digital circuits and the developed methodology begins by synthesizing Verilog source code into Verilog netlists and then converting the netlist into Spice format. The Monte Carlo infrastructure then provides a mechanism to simulate the circuits in SPICE with a huge number of test runs to explore the corresponding state space. Consequently, counters exercise the developed methodology from the initial steps to the final analysis.

The inputs that were varied in the analysis include the supply voltage (V_{dd}), the input enable voltage (V_{enable}) to provide data on the DC specification of the circuit (V_{ih}), the temperature, and the transition time and period of the input clock. The correct functional operation of the device was checked as the counters were held in reset for three cycles and then allowed to operate counting to the value of 127 (final count of a seven bit adder). Critical timing paths were anticipated to cause timing failures at the log base two transitions in the count value (15 to 16, 31 to 32, 63 to 64) increasing in difficulty with a linear relationship with the bit value.

Figure 4.2 illustrates the relationship that emerges between supply voltage and total energy for successfully passing simulations. From the graph a structure becomes apparent in which a local minimum energy is seen at 2.0V and then the energy declines as supply voltage approaches zero. In this case, a well understood relationship between supply voltage and dynamic energy explains the general decrease in energy consumption. The local increase seen

from 2.0 down to 1.5 V is suspected to be related to increased transition times in the operation of the switch that result in increases in short circuit current. Below 1.5V the decrease in dynamic power begins to dominate.

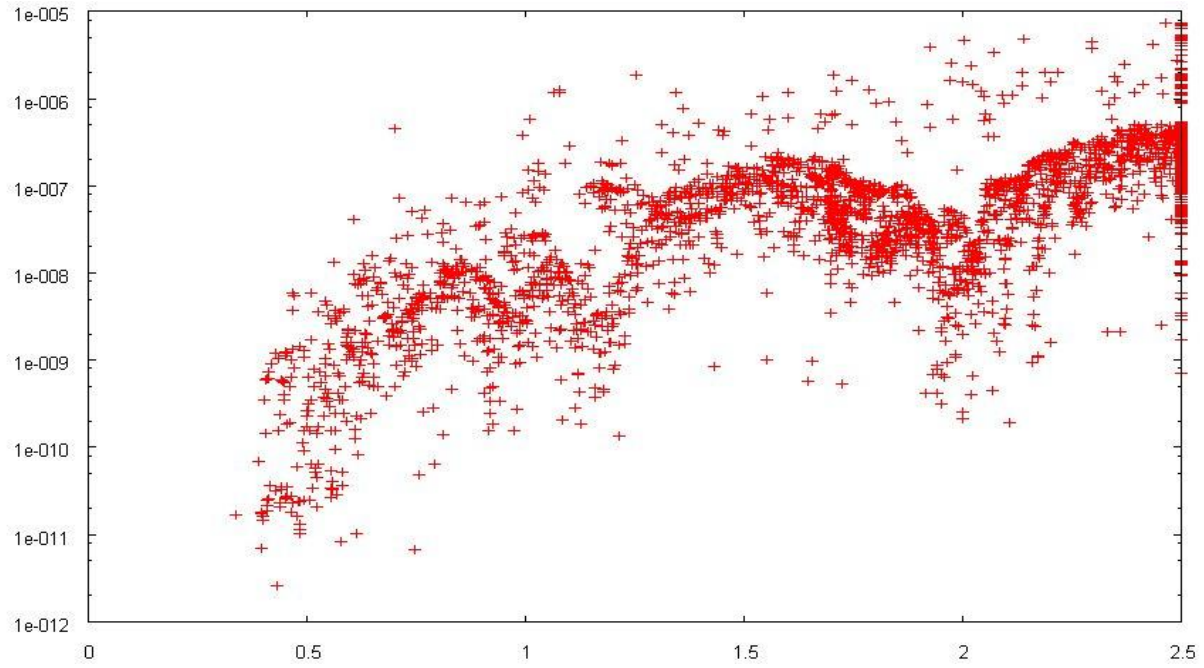

Figure 4.2 Energy Consumed as a Function of V_{dd}

Figure 4.3 illustrates how even though energy per operation should be unrelated to frequency or clock period in a dynamic sense, that static power (sub-threshold leakages) begin to increase the energy as the simulation time is increased. The plot also shows two distinct lines, which likely are the clustering of simulations around two supply voltages that include the local minimum of 2.0V and the lower absolute minimum.

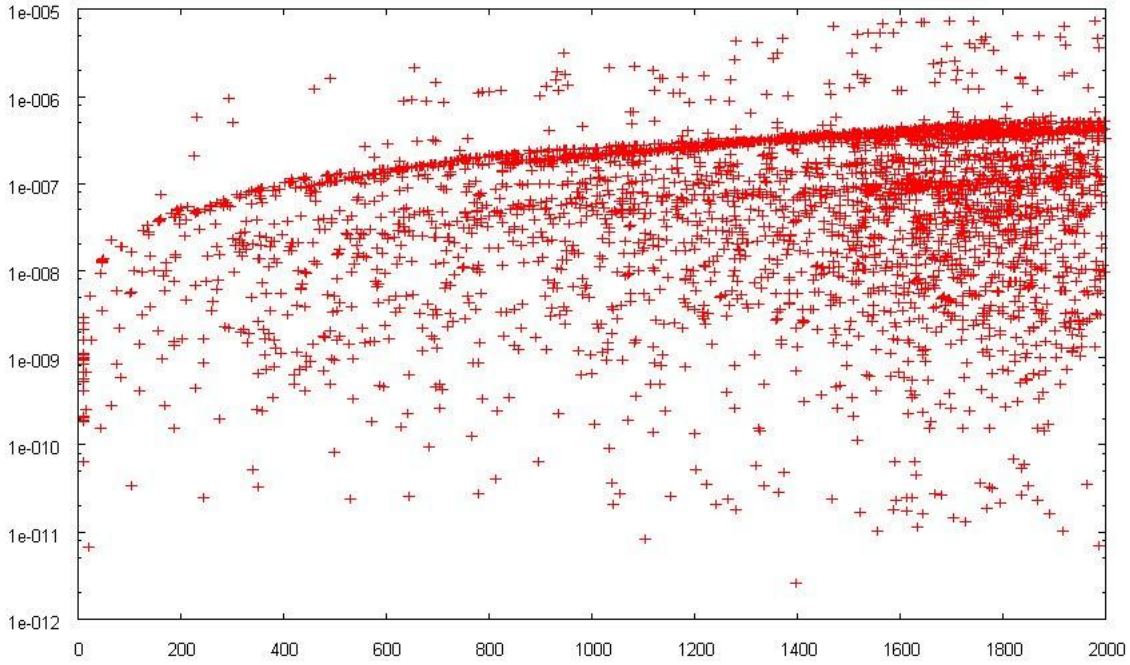

Figure 4.3 Energy Consumed as a Function of the Clock Period Δt.

With the exception of a simulator dynamic time step problem for circuits simulated with long clock periods, the majority of simulations provided insight into the operation of the circuit across a wide state space, while simultaneously providing strong evidence that the simulations are accurate and that the methodology is sound.

4.3 Summary

The results of this project provide clear evidence that a significant advantage can be gained from employing HPC systems to perform many of the important functions in the chip design and evaluation flow. Specifically, integrated circuits with large state spaces can be explored more rapidly using Monte Carlo techniques with high fidelity, mathematically complex circuit simulators for circuits the size of which would normally preclude simulation. These circuit simulations are in turn well suited for HPC systems with very large numbers of cores. The project has demonstrated a pragmatic approach that can integrate existing software to target project-specific analyses supporting design and simulation. As a proof-of-concept, a Monte

Carlo simulation was constructed as a high-level simulation flow driving SPICE sub-simulations targeting a digital counter circuit that captures most of the central functions of the circuit that will eventually be characterized for SSC Pacific. Using 128 cores, more than 5,000 SPICE sub-simulations were performed as part of a Monte Carlo sampling of parameters impacting the operation of the digital circuit. The work was based on an open-source simulator (Ngspice) that is based on Berkeley's original SPICE simulator, with performance in terms of simulation time was within 50% of commercial tools and provided similar fidelity. The methodology developed in this work encountered significant technical challenges that were successfully addressed. A method for automatically extracting a SPICE-level netlist from the output of an RTL synthesis was implemented. The treatment of large data sets generated when running thousands of SPICE simulations was addressed by integrating a real-time data compression engine into the Ngspice simulator. Circuits were simulated comprehensively that contained almost 3,000 transistors – generally much larger than typical industrial circuit simulations - and provided a benchmark for further investigation of more targeted and larger circuits.

References

[1] Two projects at SSC-Pacific are developing compact spectrum analyzers on a chip (Code 71730). The project depends heavily on non-traditional circuit design and can benefit from the effort described here, and which are supported by the Air force and the Navy (Code 30).

[2] The Advanced Dynamic Magnetometer project (SSC-Pacific Code 71730) is a mature compact magnetic field sensor with stringent power requirements. The project can directly benefit from the effort described here by advancing the development of a low power digital core as well as analog circuitry. The project is supported by the Marines and is in evaluation for use in their TRISS and TUSS systems.

[3] The goal Ultra Low Power Electronics project at SSC-Pacific (Code 71730) is to develop ultra low power standard cell libraries in support of DoD projects that require digital ICs. Because the project focuses on pushing the limits of sub-threshold circuit design, the use of large-scale parameter simulations is of critical importance. The project has the potential to benefit a great deal by the effort described here. The project is supported by The Navy In-House Laboratory Independent Research program (ILIR).

[4] A. Wang and A. Chandrakasan, "A 180-mV Subthreshold FFT Processor using a Minimum Energy Design Methodology," IEEE Journal of Solid-State Circuits, vol. 40, no. 1, pp. 310-319, January, 2005.

[5] H. Soeleman, K. Roy, B. Paul, "Robust Ultra-Low-Power Subthreshold DTMOS Logic," Proc. Int. Symp. Low-Power Electronics Design, 2000, pp. 25-30.

[6] B. Calhoun, A. Chandrakasan, "Ultra-Dynamic Voltage Scaling (UDVS) using sub-threshold operation and local voltage dithering," IEEE Journal of Solid-State Circuits, Vol. 41, No. 1, pp. 238 – 245, Jan 2006.

[7] Final Technical Report for PETTT Project PP-ENS-KY01-006-P3, "Distributed Integrated Circuit Design," David Richie, Eric MacDonald and Matthew Markulik, August 31, 2011.

[8] HSPICE is a commercial SPICE simulator provided by Synopsys Inc. (http://www.synopsys.com)

[9] SPECTRE is a commercial SPICE simulator provided by Cadence Inc. (http://www.cadence.com)

[10] Xyce is a SPICE simulator developed by the DOE. (http://xyce.sandia.gov).

[11] NGSPICE is an open-source SPICE 3F5 simulator. (http://sourceforge.net/projects/ngspice/)

[12] Spice-Opus is a free SPICE 3E4 simulator (binary only) maintained by the University of Ljubljana, Slovenia. (http://www.spiceopus.com).

[13] StormRT is an open-source library that supports real-time temporal data compression and feature detection. (http://www.browndeertechnology.com/stormrt.htm)

Appendix

A1: Netlist Extraction Tool

(Script written in PERL, Specific to the Virginia Tech Standard Cell Library)

```
#!/usr/bin/perl

#######################################################################
##
## Collapse Lines
##
#######################################################################
open(IN, "./netlist.v") || die("ERROR: Cannot open input file");  ##
Open input file
open(OUT, ">./tmp.dat") || die("ERROR: Cannot open output file"); ##
Open output file

@search_gate = <IN>;
$d=0;
until(@search_gate[$d] =~ /endmodule/)        #Begin search loop

if(@search_gate[$d] =~ /[\r]+|[\t]+|[\n]/){   #Search for key words
  @search_gate[$d] =~ s/[\r]+//g; #Search for tabs, carriage returns
and new lines
  @search_gate[$d] =~ s/[\n]+//g;
  @search_gate[$d] =~ s/[\t]+//g;
  @search_gate[$d] =~ s/;/ Vdd Gnd\n/; #Replace colon with Vdd and Gnd
and new line
  @search_gate[$d] =~ s/[ ]+/ /g;
}                                             ##  Close if statement
$d++;
}

print OUT (@search_gate);
close OUT;

#######################################################################
##
## Delete unwanted variables
##
#######################################################################
open(IN, "./tmp.dat") || die("ERROR: Cannot open input file");    ##
Open input file
open(OUT,">./netlist.sp") || die("ERROR: Cannot open output file");
     ##  Open output file

@search_gates = <IN>;                 ##  Set search variable
$n=0;
```

```
until(@search_gates[$n] =~ /endmodule/){    #  Begin search loop
     if(@search_gates[$n] !=~ /wire/ || @search_gates[$n] !=~ /input/
|| @search_gates[$n] !=~ /output/){   ##  Search for key words
          @search_gates[$n] =~ s/wire[ \d\D]+//g;     ##  Search wire
          @search_gates[$n] =~ s/input[ \d\D]+//g;    ##  Search input
          @search_gates[$n] =~ s/output[ \d\D]+//g;   ##  Search
output
          @search_gates[$n] =~ s/[\/]+//g;            ##  Search
backslash
     }                                                ##  Close if
statement
     $n++;
}                                              ##  Close search loop

#######################################################################
##
##   Search for unneeded Verilog Identifiers
##
#######################################################################
@search_gates1 = @search_gates;
$n=0;
until(@search_gates1[$n] =~ /endmodule/){   ##  Begin search loop

     @search_gates1[$n] =~ s/\.op//g;
          @search_gates1[$n] =~ s/,//g;
               @search_gates1[$n] =~ s/\.ip[\d]//g;
                    @search_gates1[$n] =~ s/\[(\d)\]/_$1/g;

     @search_gates1[$n] =~ s/\[(\d\d)\]/_$1/g;
          @search_gates1[$n] =~ s/\.ck//g;
               @search_gates1[$n] =~ s/\(+//g;
                    @search_gates1[$n] =~ s/\)+//g;
                         @search_gates1[$n] =~ s/\\//g;
                              @search_gates1[$n] =~ s/\.sb//g;
                                   @search_gates1[$n] =~ s/\.ip//g;
                                        @search_gates1[$n] =~
s/\.rb//g;
     @search_gates1[$n] =~ s/\.s//g;
          @search_gates1[$n] =~ s/\ .q//g;
               @search_gates1[$n] =~ s/[ ]+/ /g;
                    @search_gates1[$n] =~ s/\.a//g;
                         @search_gates1[$n] =~ s/\.b//g;
                              @search_gates1[$n] =~ s/\.co//g;
                                   @search_gates1[$n] =~ s/\.ci//g;
                                        @search_gates1[$n] =~
s/[\n]+//g;
$n++;
}

#######################################################################
##   Rewrite Netlist into SPICE syntax
##
```

```
#####################################################################
@search_gates = @search_gates1;
$n=0;
until(@search_gates[$n] =~ /endmodule/){    ##  Begin search loop

     if(@search_gates[$n] =~ /buf_1/){

          @search_gates[$n] =~ s/(buf_1)([( \d\D)+])/X/g;
               @search_gates[$n] = join('
',@search_gates[$n],"buf_1\n");                  }

     elsif(@search_gates[$n] =~ /buf_2/){
               @search_gates[$n] =~ s/(buf_2)([( \d\D)+])/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"buf_2\n");                  }

     elsif(@search_gates[$n] =~ /buf_4/){
               @search_gates[$n] =~ s/(buf_4)([( \d\D)+])/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"buf_4\n");                  }

     elsif(@search_gates[$n] =~ /inv_1/){
               @search_gates[$n] =~ s/(inv_1)[( \d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"inv_1\n");                  }

     elsif(@search_gates[$n] =~ /inv_2/){
               @search_gates[$n] =~ s/(inv_2)[( \d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"inv_2\n");                  }

     elsif(@search_gates[$n] =~ /inv_4/){
               @search_gates[$n] =~ s/(inv_4)[( \d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"inv_4\n");                  }

     elsif(@search_gates[$n] =~ /nand2_1/){
               @search_gates[$n] =~ s/(nand2_1)[( \d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"nand2_1\n");                }

     elsif(@search_gates[$n] =~ /nand2_2/){
               @search_gates[$n] =~ s/(nand2_2)[( \d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"nand2_2\n");                }

     elsif(@search_gates[$n] =~ /nand2_4/){
               @search_gates[$n] =~ s/(nand2_4)[( \d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"nand_4\n");                 }

     elsif(@search_gates[$n] =~ /nand3_1/){
```

```
                @search_gates[$n] =~ s/(nand3_1)[( \d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nand3_1\n");                }

     elsif(@search_gates[$n] =~ /nand3_2/){
                @search_gates[$n] =~ s/(nand3_2)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nand3_2\n");                }

     elsif(@search_gates[$n] =~ /nand4_1/){
                @search_gates[$n] =~ s/(nand4_1)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nand4_1\n");                }

     elsif(@search_gates[$n] =~ /nand4_2/){
                @search_gates[$n] =~ s/(nand4_2)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nand4_2\n");                }

     elsif(@search_gates[$n] =~ /xnor2_1/){
                @search_gates[$n] =~ s/(xnor2_1)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"xnor2_1\n");                }

     elsif(@search_gates[$n] =~ /xnor2_2/){
                @search_gates[$n] =~ s/(xnor2_2)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"xnor2_2\n");                }

     elsif(@search_gates[$n] =~ /nor2_1/){
                @search_gates[$n] =~ s/(nor2_1)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nor2_1\n");                }

     elsif(@search_gates[$n] =~ /nor2_2/){
                @search_gates[$n] =~ s/(nor2_2)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nor2_2\n");                }

     elsif(@search_gates[$n] =~ /nor2_4/){
                @search_gates[$n] =~ s/(nor2_4)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nor2_4\n");                }

     elsif(@search_gates[$n] =~ /nor3_1/){
                @search_gates[$n] =~ s/(nor3_1)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"nor3_1\n");                }

     elsif(@search_gates[$n] =~ /nor3_2/){
                @search_gates[$n] =~ s/(nor3_2)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
```

```
',@search_gates[$n],"nor3_2\n");                }

     elsif(@search_gates[$n] =~ /nor4_1/){
               @search_gates[$n] =~ s/(nor4_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"nor4_1\n");                }

     elsif(@search_gates[$n] =~ /nor4_2/){
               @search_gates[$n] =~ s/(nor4_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"nor4_2\n");                }

     elsif(@search_gates[$n] =~ /xor2_1/){
               @search_gates[$n] =~ s/(xor2_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"xor2_2\n");                }

     elsif(@search_gates[$n] =~ /xor2_2/){
               @search_gates[$n] =~ s/(xor2_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"xor2_2\n");                }

     elsif(@search_gates[$n] =~ /or2_1/){
               @search_gates[$n] =~ s/(or2_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or2_1\n");                 }

     elsif(@search_gates[$n] =~ /or2_2/){
               @search_gates[$n] =~ s/(or2_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or2_2\n");                 }

     elsif(@search_gates[$n] =~ /or2_4/){
               @search_gates[$n] =~ s/(or2_4)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or2_4\n");                 }

     elsif(@search_gates[$n] =~ /or3_1/){
               @search_gates[$n] =~ s/(or3_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or3_1\n");                 }

     elsif(@search_gates[$n] =~ /or3_2/){
               @search_gates[$n] =~ s/(or3_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or3_2\n");                 }

     elsif(@search_gates[$n] =~ /or4_1/){
               @search_gates[$n] =~ s/(or4_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or4_1\n");                 }
```

```
     elsif(@search_gates[$n] =~ /or4_2/){
               @search_gates[$n] =~ s/(or4_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"or4_2\n");                 }

     elsif(@search_gates[$n] =~ /and2_1/){
               @search_gates[$n] =~ s/(and2_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and2_1\n");                }

     elsif(@search_gates[$n] =~ /and2_2/){
               @search_gates[$n] =~ s/(and2_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and2_2\n");                }

     elsif(@search_gates[$n] =~ /and3_1/){
               @search_gates[$n] =~ s/(and3_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and3_1\n");                }

     elsif(@search_gates[$n] =~ /and3_2/){
               @search_gates[$n] =~ s/(and3_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and3_2\n");                }

     elsif(@search_gates[$n] =~ /and4_1/){
               @search_gates[$n] =~ s/(and4_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and4_1\n");                }

     elsif(@search_gates[$n] =~ /and4_2/){
               @search_gates[$n] =~ s/(and4_2)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and4_2\n");                }

     elsif(@search_gates[$n] =~ /and4_4/){
               @search_gates[$n] =~ s/(and4_4)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"and4_4\n");                }
     elsif(@search_gates[$n] =~ /cd_8/){
               @search_gates[$n] =~ s/(cd_8)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"cd_8\n");                  }

     elsif(@search_gates[$n] =~ /cd_12/){
               @search_gates[$n] =~ s/(cd_12)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"cd_12\n");                 }

     elsif(@search_gates[$n] =~ /dp_1/){
               @search_gates[$n] =~ s/(dp_1 )/X/g;
                    @search_gates[$n] = join('
```

```
',@search_gates[$n],"dp_1\n");                    @search_gates[$n] =~
s/(X([\d\D]+) clk (n_\d+) ([\d\D ]+) Vdd Gnd dp_1)/
MN$2_0 $2net44 $3 Gnd Gnd NM W=840.0n L=240.0n
MN$2_1 $2net38 clk Gnd Gnd NM W=840.0n L=240.0n
MN$2_2 $2net35 $2net38 $2net44 Gnd NM W=840.0n L=240.0n
MN$2_3 $2net32 clk $2net35 Gnd NM W=840.0n L=240.0n
MN$2_4 $2net32 $2net78 Gnd Gnd NM W=840.0n L=240.0n
MN$2_5 $2net78 $2net35 Gnd Gnd NM W=840.0n L=240.0n
MN$2_6 $2net20 $2net78 Gnd Gnd NM W=840.0n L=240.0n
MN$2_7 $2net63 clk $2net20 Gnd NM W=840.0n L=240.0n
MN$2_8 $2net12 $2net38 $2net63 Gnd NM W=840.0n L=240.0n
MN$2_9 Gnd $4 $2net12 Gnd NM W=840.0n L=240.0n
MN$2_10 $4 $2net63 Gnd Gnd NM W=840.0n L=240.0n
MP$2_0 $2net88 $3 Vdd Vdd PM W=1.68u L=240.0n
MP$2_1 $2net38 clk Vdd Vdd PM W=1.68u L=240.0n
MP$2_2 $2net35 clk $2net88 Vdd PM W=1.68u L=240.0n
MP$2_3 $2net80 $2net38 $2net35 Vdd PM W=1.68u L=240.0n
MP$2_4 Vdd $2net78 $2net80 Vdd PM W=1.68u L=240.0n
MP$2_5 $2net78 $2net35 Vdd Vdd PM W=1.68u L=240.0n
MP$2_6 $2net68 $2net78 Vdd Vdd PM W=1.68u L=240.0n
MP$2_7 $2net63 $2net38 $2net68 Vdd PM W=1.68u L=240.0n
MP$2_8 $2net60 clk $2net63 Vdd PM W=1.68u L=240.0n
MP$2_9 Vdd $4 $2net60 Vdd PM W=1.68u L=240.0n
MP$2_10 $4 $2net63 Vdd Vdd PM W=1.68u L=240.0n
.IC v($2net35)=2.5 v($3)=0 v($2net78)=0 v($2net63)=2.5 v($4)=0\n/g
          }
     }

     elsif(@search_gates[$n] =~ /ABorC/){
               @search_gates[$n] =~ s/(ABorC)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"ABorC\n");                  }

     elsif(@search_gates[$n] =~ /not_ab_or_c_or_d/){
               @search_gates[$n] =~
s/(not_ab_or_c_or_d)[(\d\D)+]/X/g;               @search_gates[$n] =
join(' ',@search_gates[$n],"not_ab_or_c_or_d\n");
     }
          elsif(@search_gates[$n] =~ /ab_or_c_or_d/){
                    @search_gates[$n] =~
s/(ab_or_c_or_d)[(\d\D)+]/X/g;
     @search_gates[$n] = join(' ',@search_gates[$n],"ab_or_c_or_d\n");
          }
          elsif(@search_gates[$n] =~ /drp_2/){
                    @search_gates[$n] =~ s/(drp_2)[(\d\D)+]/X/g;
                         @search_gates[$n] = join('
',@search_gates[$n],"drp_2\n");                  }

     elsif(@search_gates[$n] =~ /dksp_1/){
               @search_gates[$n] =~ s/(dksp_1)[(\d\D)+]/X/g;
                    @search_gates[$n] = join('
',@search_gates[$n],"dksp_1\n");                 }
```

```
     elsif(@search_gates[$n] =~ /mux2_1/){
                @search_gates[$n] =~ s/(mux2_1)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"mux2_1\n");                }

     elsif(@search_gates[$n] =~ /mux2_2/){
                @search_gates[$n] =~ s/(mux2_2)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"mux2_2\n");                }

     elsif(@search_gates[$n] =~ /fulladder/){
                @search_gates[$n] =~ s/(fulladder)[(\d\D)+]/X/g;
                     @search_gates[$n] = join('
',@search_gates[$n],"fulladder\n");              }

     elsif(@search_gates[$n] =~ /endmodule/){
                @search_gates[$n] =~ s/endmodule/.Include
models.model\r.Include standard_cells.dat\r.END/g;
          }

                $n++;                ##  Increment for search loop
}                          ##  Close search loop
#print OUT (@search_gates)

print OUT (@search_gates);
```

A2: Pre-Processing Script

(Written in PERL).

```
#!/opt/local/bin/perl
#////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////
#//
#//  PETTT Project - SPAWAR Detection Algorithm Massively Parallel Simulation Generator
#//                Spice netlist Monte Carlo testcase generator
#//                Version 1.0.2 - 1/30/2011
#//
#////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////
#
#    USAGE: pett_preprocess.pl <option flags> <outputfile_name>
#
#        -q          for quiet output
#        -h          for help
#
#   note: try pdat_gui.pl for graphical interface

$num_of_args = $#ARGV + 1;
$clock_frequency = 20;
$offset_delay = 1000;
$spice_rise_time = 0.1;
$num_data_points_before = 50;
$sensitivity = 10;
$config = 1011;
$num_data_points_after = 50;
$num_stable_points = 50;
$rise_time = 20;
$offset = 20;
$standard_deviation = 3;

if(($ARGV[0] =~ /^-[hH]/)|($num_of_args < 7)) {
          printf("\n\nPETT Simulation Generator                                    \n ");
      printf("usage example - spice_preprocessing.pl -nb=100 -sen=11 -con=13 -na=100 -rt=5 -
sn=20 -o=5 -sd=8 -of=tb.v -sf=input.sp \n");
          printf("            Version 1.0.2 - 1/30/2011                    \n\n ");
      printf("-q          for quiet output                                 \n ");
      printf("-config=NUMBER   for morphing design - 0000 to 1111          \n ");
      printf("-nb=NUMBER   for selecting number of data points before      \n ");
      printf("-na=NUMBER   for selecting number of data points after       \n ");
      printf("-rt=NUMBER   for selecting number of rise time of step       \n ");
      printf("-sn=NUMBER   for selecting number of stable points           \n ");
      printf("-sen=NUMBER  for selecting sensitivity input (o to 1023)     \n ");
```

```
        printf("-o=NUMBER    for selecting offset of magnetic field             \n ");
        printf("-sd=NUMBER   for selecting standard deviation of data           \n ");
        printf("-of=NAME   for name of output verilog                           \n ");
        printf("-sf=NAME   for name of output spice                             \n ");
        printf("-h          for help                                            \n ");
           exit;}


for($index=0; $index <=$num_of_args; $index++) {
if (@ARGV[$index] =~ /^-q/)          { $quiet = 1;}
if (@ARGV[$index] =~ /^-nb\=([0-9]+)/){ $num_data_points_before = $1;}
if (@ARGV[$index] =~ /^-sen\=([0-9]+)/){ $sensitivity = $1;}
if (@ARGV[$index] =~ /^-con\=([0-1]+)/){ $config = $1;}
if (@ARGV[$index] =~ /^-na\=([0-9]+)/){ $num_data_points_after = $1;}
if (@ARGV[$index] =~ /^-sn\=([0-9]+)/){ $num_stable_points = $1;}
if (@ARGV[$index] =~ /^-rt\=([0-9]+)/){ $rise_time = $1;}
if (@ARGV[$index] =~ /^-o\=([0-9]+)/) { $offset = $1;}
if (@ARGV[$index] =~ /^-of\=([a-zA-Z\.]*)/) { $output_file       = $1; printf("verilog
$output_file"); }
if (@ARGV[$index] =~ /^-sf\=([a-zA-Z\.]*)/) { $spice_output_file = $1; printf("spice
$spice_output_file");}
if (@ARGV[$index] =~ /^-sd\=([0-9]+)/){ $standard_deviation = $1;}
                    }
@starttbarray = (
'module tb();',
'reg            clk;',
'reg            reset;',
'reg   [23:0]    data_input;',
'reg   [23:0]    old_data;',
'wire           detect_out;             ',
'reg            data_enable;',
'wire           detect_out_sticky;              ',
'',
'////////////////////////////////////////////////////',
'// clock stimilus',
'////////////////////////////////////////////////////',
'initial              clk = 0;',
'initial forever #20   clk = ~clk;',
'',
'reg  [7:0]       sample_counter;',
'always @(posedge clk)',
'     if(reset)  sample_counter = 0;',
'     else       sample_counter = sample_counter + 1;',
'',
'wire            sample_clk = (sample_counter == 8\'hFF);',
'',
```

```
'////////////////////////////////////////////////////',
'// Reset and completion logic ',
'////////////////////////////////////////////////////',
'initial ',
' begin',
'               reset             <= 1;',
'#1000          reset             <= 0;',
'#1000          reset             <= 1;',
'#1000          reset             <= 0;',
'               data_enable       <= 0;',
'ENDENDEND'                        ,
);


@middletbarray = (
'               $display("enabled with sensitivity of %%d and config of %%b",
tb.detect_module.sensitivity, tb.detect_module.config);',
'               data_enable       <= 1;',
'ENDENDEND'                        ,
);


@endtbarray = (
'#100000     $finish;',
'end',
'',
'always @(negedge sample_clk or posedge reset)  ',
'             if(reset)       old_data   <= 0;',
'             else            old_data <= data_input;',
'',
'detect_module detect_module',
'            (.clk               ( clk               ),',
'             .soft_reset        ( reset             ),',
'             .sensitivity       ( 10\'dname         ),   ///modsen',
'             .config            ( 4\'bname          ),   ///modcon',
'             .data_available      ( sample_clk        ),',
'             .RTD               ( data_input        ),',
'             .last_RTD          ( old_data          ),',
'             .enable            ( data_enable       ),',
'             .detect_sticky       ( detect_out_sticky    ));',
'',
'',
'always @(negedge sample_clk) $display("di=%%x, ma0=%%x, ma1=%%x, error=%%x,
merror=%%x, thresh=%%x, delta detect=%%x, error detect= %%x, sticky=%%x",',
'                                                    tb.detect_module.RTD,',
'                                                    tb.detect_module.moving_average0,',
```

```
'                                                       tb.detect_module.moving_average1,',
'                                                       tb.detect_module.error,',
'                                                       tb.detect_module.moving_error,',
'                                                       tb.detect_module.error_detect_thresh,',
'                                                       tb.detect_module.delta_detect,',
'                                                       tb.detect_module.error_detect,',
'                                                       tb.detect_out_sticky);',
",
'////////////////////////////////////////////k////////',
'// Waveform Generation ',
'////////////////////////////////////////////////////',
'initial  $dumpfile("verilog.vcd"); ',
'initial  $dumpvars(); ',
",
'endmodule',
'ENDENDEND'
);



@startspicearray = (
'module tb();',
'reg            clk;',
'reg            reset;',
'reg   [23:0]    data_input;',
'reg   [23:0]    old_data;',
'wire           detect_out;            ',
'reg            data_enable;',
'wire           detect_out_sticky;            ',
",
'////////////////////////////////////////////////////',
'// clock stimilus',
'////////////////////////////////////////////////////',
'initial              clk = 0;',
'initial forever #20   clk = ~clk;',
",
'reg  [7:0]       sample_counter;',
'always @(posedge clk)',
'      if(reset)  sample_counter = 0;',
'      else       sample_counter = sample_counter + 1;',
",
'wire            sample_clk = (sample_counter == 8\'hFF);',
",
'////////////////////////////////////////////////////',
'// Reset and completion logic ',
'////////////////////////////////////////////////////',
```

```
'initial ',
' begin',
'               reset             <= 1;',
'#1000            reset             <= 0;',
'#1000            reset             <= 1;',
'#1000            reset             <= 0;',
'               data_enable        <= 0;',
'ENDENDEND'                         ,
);
```

```
#####################################################################
# Add front material here
#####################################################################
# building the tb.v here
$index = 0;
while (@starttbarray[$index] !~ /ENDENDEND/){
            @tboutput[$index] = @starttbarray[$index];
            $index++;
   }
$tb_length = $index + 1;

#####################################################################
# Add X number of stable inputs
#####################################################################
for($index=$tb_length; $index <= $num_stable_points+$tb_length; $index++){
        @tboutput[$index] =  "@(posedge sample_clk) data_input <= 0;  ";
  }
$tb_length = $tb_length + $num_stable_points;


#####################################################################
# Enable detection
#####################################################################
# building the tb.v here
$index = 0;
while (@middletbarray[$index] !~ /ENDENDEND/){
            @tboutput[$index + $tb_length] = @middletbarray[$index];
            $index++;
   }
$tb_length = $tb_length + $index + 1;
```

```
$spice_length = 0;
$second_index = 0;

#####################################################################
# Add random data points
#####################################################################
for($index=$tb_length; $index <= $num_data_points_before+$tb_length; $index++){
        $value = int(gaussian_rand()*$standard_deviation);
        @binarray[$second_index] = $value;
        @tboutput[$index] =  "@(posedge sample_clk) data_input <= $value;  ";
        $second_index++;
  }

$spice_length = $spice_length + $second_index;
$tb_length = $tb_length + $num_data_points_before;

@tboutput[$tb_length] =  '$display("Offset in magnetic field starts...");  ';
$tb_length++;


#####################################################################
# Add step with rise time
#####################################################################
for($index=$tb_length; $index <= ($rise_time+$tb_length); $index++){
        $rise = (($index - $tb_length)*$offset)/$rise_time;
        $value = int(gaussian_rand()*$standard_deviation + $rise);
        @binarray[$second_index] = $value;
        @tboutput[$index] =  "@(posedge sample_clk) data_input <= $value;  ";
        $second_index++;

  }
        $tb_length = $tb_length + $rise_time;
        $spice_length = $spice_length + $rise_time;

        @tboutput[$tb_length] =  '$display("Offset in magnetic field is complete..."); ';
        $tb_length++;


#####################################################################
# Add random data points with offset
#####################################################################
for($index=$tb_length; $index <= $num_data_points_after+$tb_length; $index++){
        $value = int(gaussian_rand()*$standard_deviation + $offset);
        @binarray[$second_index] = $value;
```

```
        @tboutput[$index] =  "@(posedge sample_clk) data_input <= $value;  ";
        $second_index++;
  }

        $spice_length = $spice_length + $rise_time;
        $tb_length = $tb_length + $num_data_points_after;


#################################################################
# Finish testbench
#################################################################
# building the tb.v here
$index = 0;
while (@endtbarray[$index] !~ /ENDENDEND/){
   if (@endtbarray[$index] =~ /(.*)name(.*)\/\/modcon.*/ ) {
            @tboutput[$index + $tb_length] = $1.$config.$2;
            $index++;
        }
   if (@endtbarray[$index] =~ /(.*)name(.*)\/\/modsen.*/ ) {
            @tboutput[$index + $tb_length] = $1.$sensitivity.$2;
            $index++;
        }
   else {
            @tboutput[$index + $tb_length] = @endtbarray[$index];
            $index++;
        }
   }
$tb_length = $tb_length + $index + 1;


if(!$quiet)
{printf("\n++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++ \n");}
if(!$quiet)
{printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++   \n");}
if(!$quiet) {printf("    PETTT tb generator                                    \n");}
if(!$quiet) {printf("          Generates verilog test bench and spice input          \n");}
if(!$quiet) {printf("          for the detection algorithm.                           \n");}
if(!$quiet)
{printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++   \n");}
if(!$quiet) {printf("+++++           Version 1.0.2 - 1/30/2011             ++++++++++   \n");}
if(!$quiet)
{printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++   \n");}
```

```
if(!$quiet)
{printf("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++   \n");}



open(OUT,"> $output_file") || die "can not open verilog file $output_file for writing$!";
$index = 0;
for($index=0;$index <= $tb_length;$index++){
     printf(OUT "@tboutput[$index] \n");
   }
close(OUT);


open(OUTS,"> $spice_output_file") || die "can not open spice file $spice_output_file for
writing$!";

$oldbinval = 0;
$binval = 0;
printf(OUTS "vr0 RTD_0 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{31}([0-1]).*/){$bit = $1;}
      $starttime = $offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr1 RTD_1 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{30}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
```

```
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr2 RTD_2 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{29}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr3 RTD_3 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{28}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");
```

```
$oldbinval = 0;
$binval = 0;
printf(OUTS "vr4 RTD_4 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{27}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr5 RTD_5 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{26}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr6 RTD_6 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{25}([0-1]).*/){$bit = $1;}
```

```
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr7 RTD_7 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{24}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr8 RTD_8 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{23}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
```

```
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr9 RTD_9 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{22}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr10 RTD_10 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{21}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr11 RTD_11 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
```

```
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{20}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");

$oldbinval = 0;
$binval = 0;
printf(OUTS "vr12 RTD_12 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{19}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr13 RTD_13 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{18}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
```

```
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");



$oldbinval = 0;
$binval = 0;
printf(OUTS "vr14 RTD_14 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{17}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");



$oldbinval = 0;
$binval = 0;
printf(OUTS "vr15 RTD_15 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{16}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");
```

```
$oldbinval = 0;
$binval = 0;
printf(OUTS "vr16 RTD_16 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{15}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr17 RTD_17 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{14}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr18 RTD_18 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{13}([0-1]).*/){$bit = $1;}
```

```
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr19 RTD_19 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{12}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
    }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr20 RTD_20 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
        $oldbit = $bit;
        $binval = dec2bin(@binarray[$index]);
        if ($binval =~ /^[0-1]{11}([0-1]).*/){$bit = $1;}
        $starttime =$offset_delay + $index*$clock_frequency;
        $endtime =$starttime + $spice_rise_time;
        $startvoltage = $oldbit * 2.5;
        $endvoltage = $bit * 2.5;
        $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
        $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
        printf(OUTS "$startstring \n");
        printf(OUTS "$endstring \n");
```

```
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr21 RTD_21 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{10}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr22 RTD_22 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{9}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");


$oldbinval = 0;
$binval = 0;
printf(OUTS "vr23 RTD_23 0 PWL(0   0 \n ");
for($index=0;$index <= $spice_length;$index++){
```

```
      $oldbit = $bit;
      $binval = dec2bin(@binarray[$index]);
      if ($binval =~ /^[0-1]{8}([0-1]).*/){$bit = $1;}
      $starttime =$offset_delay + $index*$clock_frequency;
      $endtime =$starttime + $spice_rise_time;
      $startvoltage = $oldbit * 2.5;
      $endvoltage = $bit * 2.5;
      $startstring = "    " . $starttime . "n \t\t" . $startvoltage;
      $endstring   = "    " . $endtime . "n \t\t" . $endvoltage;
      printf(OUTS "$startstring \n");
      printf(OUTS "$endstring \n");
   }
printf(OUTS "     ) \n ");

close(OUTS);

sub gaussian_rand {
    my ($u1, $uw);
    my $w;
    my ($g1, $g2);

  do {
     $u1 = 2 * rand() -1;
     $u2 = 2 * rand() -1;
     $w  = $u1*$u1 + $u2*$u2;
    } while ( $w >= 1);

   $w = sqrt( (-2 * log($w)) / $w );
   $g2 = $u1 * $w;
   $g1 = $u2 * $w;
   return $g1;
}


sub dec2bin {
          my $str = unpack("B32", pack("N", shift));
#           $str =~ s/^[0+(?=\d)//;
          return $str;
}
```

A3: Post-Processing script

(Written in PERL)

```
#!/usr/bin/perl
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////
#//
#//  PETTT Project - SPAWAR Detection Algorithm Massively Parallel
Simulation Generator
#//                  Spice netlist Monte Carlo post-processing script.
#//                  Version 1.0.0 - 8/8/2011
#//
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////
#
#     USAGE: pettt_postprocess_1_0_0.pl <outputfile_name>
#
#     Authored By:  Matthew Markulik
#
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////
#
#     Command Line options and print out
#
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////

printf("\n\n\t\t\t\t\tPETTT Post-Processing Script\n");
printf("\t\t\t\t\t  Version 1.0.8 - 8/8/2011\n\n");

if(@ARGV[0] =~ /^-[hH]/){
printf("\t\tperl pettt_postprocessing_1_0_0.pl <verilog input file>
<spice input file> <output file>\n\n");
printf("\t\t\t\t **********   Do NOT include file extensions
**********\n\n\n");
exit;}
printf("\t\t\t\t   \*\*\*  Output Directed to @ARGV[2].dat
\*\*\*\n\n");

open(IN, "./@ARGV[0].log")     || die("\t\t\tERROR: Cannot Open Input
File @ARGV[0].log   \n\n");
open(IN0,"./@ARGV[1].splog")   || die("\t\t\tERROR: Cannot Open Input
File @ARGV[1].splog \n\n");
open(IN1,"./input.dat")        || die("\t\t\tERROR: Cannot Open Input
File input.dat      \n\n");
open(OUT,">./@ARGV[2].dat")    || die("\t\t\tERROR: Cannot Open Output
File @ARGV[2].dat  \n\n");

@verilog = <IN>;
```

```
@spice   = <IN0>;
@input   = <IN1>;

#////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////
#
#      Verilog Analysis
#
#////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////

@verilog_detect      = grep(/detect=[\d]/,@verilog);
@verilog_detect_high = grep(/detect=1/,@verilog);
@verilog_abs_delta   = grep(/abs_delta=[0-1xz]{24}/,@verilog);

$verilog_count       = @verilog_detect;
$verilog_high_count  = @verilog_detect_high;

$n = 0;

while(@verilog_abs_delta[$n] ne ""){
      @verilog_abs_delta[$n] =~ s/detect=[\d] (abs_delta=[0-1a-
z]{24})/$1/g;
      $n++;
}

$n = 0;

while(@verilog_detect[$n] =~ /detect/){
      @verilog_detect[$n] =~ s/(detect=[\d]) abs_delta=[\dxz]+/$1/g;
      $n++;
}

$n = 0;
$o = 0;
$p = 0;
$number_verilog_detects = 0;

while(@verilog_detect[$n] ne ""){
      if(@verilog_detect[$n] =~ /detect=1/){
         splice (@verilog_detect, $n+1, 0, "high\n");
           @verilog_count_till_high[$o] = $p - 1;
           $p = 0;
           $o++;
           $number_verilog_detects++;
      }
      $p++;
      $n++;
}

$o = 0;
```

```
while(@verilog_abs_delta[$o] ne ""){

if(@verilog_abs_delta[$o] =~ /abs_delta=[1-9]{24}/){
  for($n = 0; $n <= 23; $n++){
   $verilog_abs_delta1 = @verilog_abs_delta[$o];
    if($n == 0){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$1/g;
       @delta_val[0] = $verilog_abs_delta1;
         if(@delta[0] =~ /1/){
            @delta[0] = 1;
         }
         else{
            @delta_val[0] = 0;
         }
    }
    if($n == 1){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$2/g;
       @delta_val[1] = $verilog_abs_delta1;
          if(@delta_val[1] =~ /1/){
             @delta_val[1] = 2;
          }
          else{
             @delta_val[1] = 0;
          }
    }
    if($n == 2){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$3/g;
       @delta_val[2] = $verilog_abs_delta1;
          if(@delta_val[2] =~ /1/){
             @delta_val[2] = 4;
          }
          else{
             @delta_val[2] = 0;
          }
    }
    if($n == 3){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$4/g;
       @delta_val[3] = $verilog_abs_delta1;
          if(@delta_val[3] =~ /1/){
             @delta_val[3] = 8;
```

```
            }
            else{
               @delta_val[3] = 0;
            }
    }
    if($n == 4){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$5/g;
       @delta_val[4] = $verilog_abs_delta1;
          if(@delta_val[4] =~ /1/){
             @delta_val[4] = 16;
          }
          else{
             @delta_val[4] = 0;
          }
    }
    if($n == 5){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$6/g;
       @delta_val[5] = $verilog_abs_delta1;
          if(@delta_val[5] =~ /1/){
             @delta_val[5] = 32;
          }
          else{
             @delta_val[5] = 0;
          }
    }
    if($n == 6){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$7/g;
       @delta_val[6] = $verilog_abs_delta1;
          if(@delta_val[6] =~ /1/){
             @delta_val[6] = 64;
          }
          else{
             @delta_val[6] = 0;
          }
    }
    if($n == 7){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$8/g;
       @delta_val[7] = $verilog_abs_delta1;
          if(@delta_val[7] =~ /1/){
             @delta_val[7] = 128;
```

```
            }
            else{
               @delta_val[7] = 0;
            }
    }
    if($n == 8){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$9/g;
       @delta_val[8] = $verilog_abs_delta1;
          if(@delta_val[8] =~ /1/){
             @delta_val[8] = 256;
          }
          else{
             @delta_val[8] = 0;
          }
    }
    if($n == 9){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$10/g;
       @delta_val[9] = $verilog_abs_delta1;
          if(@delta_val[9] =~ /1/){
             @delta_val[9] = 512;
          }
          else{
             @delta_val[9] = 0;
          }
    }
    if($n == 10){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$11/g;
       @delta_val[10] = $verilog_abs_delta1;
          if(@delta_val[10] =~ /1/){
             @delta_val[10] = 1024;
          }
          else{
             @delta_val[10] = 0;
          }
    }
    if($n == 11){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$12/g;
       @delta_val[11] = $verilog_abs_delta1;
          if(@delta_val[11] =~ /1/){
             @delta_val[11] = 2048;
```

```
            }
            else{
               @delta_val[11] = 0;
            }
    }
    if($n == 12){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$13/g;
       @delta_val[12] = $verilog_abs_delta1;
          if(@delta_val[12] =~ /1/){
             @delta_val[12] = 4096;
          }
          else{
             @delta_val[12] = 0;
          }
    }
    if($n == 13){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$14/g;
       @delta_val[13] = $verilog_abs_delta1;
          if(@delta_val[13] =~ /1/){
             @delta_val[13] = 8192;
          }
          else{
             @delta_val[13] = 0;
          }
    }
    if($n == 14){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$15/g;
       @delta_val[14] = $verilog_abs_delta1;
          if(@delta_val[14] =~ /1/){
             @delta_val[14] = 16384;
          }
          else{
             @delta_val[14] = 0;
          }
    }
    if($n == 15){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$16/g;
       @delta_val[15] = $verilog_abs_delta1;
          if(@delta_val[15] =~ /1/){
             @delta_val[15] = 32768;
```

```
            }
            else{
               @delta_val[15] = 0;
            }
    }
    if($n == 16){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$17/g;
       @delta_val[16] = $verilog_abs_delta1;
          if(@delta_val[16] =~ /1/){
             @delta_val[16] = 65536;
          }
          else{
             @delta_val[16] = 0;
          }
    }
    if($n == 17){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$18/g;
       @delta_val[17] = $verilog_abs_delta1;
          if(@delta_val[17] =~ /1/){
             @delta_val[17] = 131072;
          }
          else{
             @delta_val[17] = 0;
          }
    }
    if($n == 18){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$19/g;
       @delta_val[18] = $verilog_abs_delta1;
          if(@delta_val[18] =~ /1/){
             @delta_val[18] = 262144;
          }
          else{
             @delta_val[18] = 0;
          }
    }
    if($n == 19){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$20/g;
       @delta_val[19] = $verilog_abs_delta1;
          if(@delta_val[19] =~ /1/){
             @delta_val[19] = 524288;
```

```
            }
            else{
               @delta_val[19] = 0;
            }
    }
    if($n == 20){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$21/g;
       @delta_val[20] = $verilog_abs_delta1;
          if(@delta_val[20] =~ /1/){
             @delta_val[20] = 1048576;
          }
          else{
             @delta_val[20] = 0;
          }
    }
    if($n == 21){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$22/g;
       @delta_val[21] = $verilog_abs_delta1;
          if(@delta_val[21] =~ /1/){
             @delta_val[21] = 2097152;
          }
          else{
             @delta_val[21] = 0;
          }
    }
    if($n == 22){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$23/g;
       @delta_val[22] = $verilog_abs_delta1;
          if(@delta_val[22] =~ /1/){
             @delta_val[22] = 4194304;
          }
          else{
             @delta_val[22] = 0;
          }
    }
    if($n == 23){
       $verilog_abs_delta1 =~ s/abs_delta=([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-1])([0-
1])/$24/g;
       @delta_val[23] = $verilog_abs_delta1;
          if(@delta_val[23] =~ /1/){
             @delta_val[23] = 8388608;
```

```
          }
          else{
             @delta_val[23] = 0;
          }
    }
  }

}
else{
    @delta_val = 0;
}

#print OUT (@delta_val);
@abs_delta_val[$o] = @delta_val[0] + @delta_val[1] + @delta_val[2] +
@delta_val[3] + @delta_val[4] + @delta_val[5] + @delta_val[6] +
@delta_val[7] + @delta_val[8] + @delta_val[9] + @delta_val[10] +
@delta_val[11] + @delta_val[12] + @delta_val[13] + @delta_val[14] +
@delta_val[15] + @delta_val[16] + @delta_val[17] + @delta_val[18] +
@delta_val[19] + @delta_val[20] + @delta_val[21] + @delta_val[22] +
@delta_val[23];

$o++;
}

#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////
#
#        SPICE Analysis
#
#////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////

$n = 0;

while(@input[$n] ne ""){
      if(@input[$n] =~ /vck/){
     @input[$n] =~ s/vck clk 0 PULSE\(0 [\d].[\d] 0 [\d]+p [\d]+p
[\d]+n ([\d]+)n\)/$1e-9/g;
         $period    =  @input[$n];
      }
      if(@input[$n] =~ /v1/){
         @input[$n] =~ s/v1 Vdd 0 dc ([\d].[\d])/$1/g;
         $vdd = @input[$n];
      }
      $n++;
}

$n = 0;

while(@spice[$n] ne ""){
      @spice[$n] =~ s/\t/ /g;
      @spice[$n] =~ s/[ ]+/ /g;
```

```
      $n++;
}

$n = 0;
$o = 0;

while(@spice[$n] ne ""){
      if(@spice[$n] =~ /index/){
         @info[$o] = @spice[$n];
         $o++;
      }
      $n++;
}

$n = 0;
$o = 0;

while(@spice[$n] ne ""){
      if(@spice[$n] =~ /index/){
         @info1[$o] = @spice[$n];
         $o++;
      }
      $n++;
}

shift(@info);
shift(@info1);

$n = 0;
$o = 0;

while(@info[$n] ne ""){
      @info[$n] =~ s/(index [\d]+ val ([\d].[\d]{6}e-
[\d][\d])[\d\D]+)/$2\n/g;
      @time[$o] = @info[$n];
      $o++;
      $n++;
}

$n = 0;
$o = 0;

while(@info1[$n] ne ""){
      @info1[$n] =~ s/(index [\d]+ val ([\d].[\d]{6}e-[\d][\d]) val
([\d\D]+.[\d]+e[\D][\d][\d]))/$3/g;
      @detect[$o] = @info1[$n];
      $o++;
      $n++;
}

$o = 0;
$t = 0;
```

```
$length = @time;

for($n = 0; $n <= $length; $n++){
    if(($t - 30e-9) <= @time[$n] && @time[$n] <= ($t + 30e-9)){
       @time_new[$o]   = @time[$n];
       @detect_new[$o] = @detect[$n];
       $o++;
       $t = $t + $period;
#       print OUT ("$time_new\n");
    }
}

$on  = .9*$vdd;
$off = .1*$vdd;
$n   = 0;

while(@detect_new[$n] ne ""){
      if(@detect_new[$n] >= $on){
         @detect_new[$n] = "1\n";
      }
      else{
         @detect_new[$n] = "0\n";
      }
      $n++;
}


#////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////
#
#        Simulator Comparison
#
#////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////

$n = 0;

while(@time_new[$n] ne ""){
      @combined[$n] = "@time_new[$n]    @detect_new[$n]";
        $n++;
}

$n = 0;
$o = 0;
$p = 0;
$number_spice_detects = 0;

while(@detect_new[$n] ne ""){
      if(@detect_new[$n] =~ /1/){
         splice (@verilog_detect, $n+1, 0, "high\n");
           @spice_count_till_high[$o] = $p - 1;
           $p = 0;
```

```
            $o++;
            $number_spice_detects++;
      }
      $p++;
      $n++;
}


$n = 0;
$o = 0;
$p = 0;

while(@verilog_count_till_high[$n] ne "" && @spice_count_till_high[$n]
ne ""){
      if(@verilog_count_till_high[$n] == @spice_count_till_high[$n]){
          $o++;
      }
      if(@verilog_count_till_high[$n] != @spice_count_till_high[$n]){
          $p++;
          @number_clk_cycles_off = abs($number_verilog_detects[$n] -
$number_spice_detects[$n]);
      }
      $n++;
}

$mismatch = $p;
$alignments = $o;

if($number_verilog_detects == $number_spice_detects){
   $number_false_detects = 0;
}
if($number_verilog_detects != $number_spice_detects){
   $verilog_or_spice_has_more = $number_verilog_detects -
$number_spice_detects;
}
if($number_verilog_detects < $number_spice_detects){
   $number_false_detects = abs($number_verilog_detects -
$number_spice_detects);
}
if($number_verilog_detects > $number_spice_detects){
   $number_detects_not_in_spice = abs($number_verilog_detects -
$number_spice_detects);
}
if($verilog_or_spice_has_more < 0){
   $verilog_or_spice_has_more = "SPICE";
}
if($verilog_or_spice_has_more > 0){
   $verilog_or_spice_has_more = "VERILOG";
}
if($verilog_or_spice_has_more == 0 && $number_verilog_detects == 0 &&
$number_spice_detects == 0){
   $verilog_or_spice_has_more = "No Detects in Either Simulation";
```

```
}
if($verilog_or_spice_has_more == 0 && $number_verilog_detects != 0 or
$number_spice_detects != 0){
   $verilog_or_spice_has_more = "Equal Number of Detects in both
Simulators";
}

print OUT ("Number of mismatched detects = $mismatch\n");
print OUT ("Number of aligned detects    = $alignments\n");
print OUT ("Number Verilog detects       =
$number_verilog_detects\n");
print OUT ("Number Spice detects         = $number_spice_detects\n");
print OUT ("Number False Detects         = $number_false_detects\n");
print OUT ("Number Detects not in SPICE  =
$number_detects_not_in_spice\n");
print OUT ("Simulator with more detects  =
$verilog_or_spice_has_more\n");
print OUT (@detect_new);
```

A4: 64-Bit Counter Netlist Extracted Using Netlist Conversion Tool

(NGSPICE Syntax)

```
v0 Gnd 0 dc 0
vck clk 0 pulse(0 2.5 0 200p 200p 5n 10n)
vreset reset 0 pwl(0 2.5 200n 2.5 200.1n 0)
v1 Vdd 0 dc 2.5


 Xout_reg_63 clk n_261 out_63 Vdd Gnd dp_1
 Xg3735 reset n_260 n_261 Vdd Gnd nor2_2
 Xout_reg_62 clk n_259 out_62 Vdd Gnd dp_1
 Xg3738 out_63 n_258 n_260 Vdd Gnd xor2_2
 Xg3737 reset n_257 n_259 Vdd Gnd nor2_2
 Xout_reg_61 clk n_256 out_61 Vdd Gnd dp_1
 Xg3743 out_62 n_255 n_258 Vdd Gnd nand2_2
 Xg3741 out_62 n_254 n_257 Vdd Gnd xor2_2
 Xg3740 reset n_253 n_256 Vdd Gnd nor2_2
 Xg3747 n_254 n_255 Vdd Gnd inv_2
 Xout_reg_60 clk n_252 out_60 Vdd Gnd dp_1
 Xg3745 out_61 n_250 n_253 Vdd Gnd xor2_2
 Xg3748 out_61 n_251 n_254 Vdd Gnd nand2_2
 Xg3744 reset n_249 n_252 Vdd Gnd nor2_2
 Xout_reg_59 clk n_248 out_59 Vdd Gnd dp_1
 Xg3752 n_250 n_251 Vdd Gnd inv_2
 Xg3750 out_60 n_246 n_249 Vdd Gnd xor2_2
 Xg3753 out_60 n_247 n_250 Vdd Gnd nand2_2
 Xg3749 reset n_245 n_248 Vdd Gnd nor2_2
 Xg3757 n_246 n_247 Vdd Gnd inv_2
 Xout_reg_58 clk n_244 out_58 Vdd Gnd dp_1
 Xg3755 out_59 n_241 n_245 Vdd Gnd xor2_2
 Xg3758 out_59 n_242 n_246 Vdd Gnd nand2_2
 Xg3754 reset n_240 n_244 Vdd Gnd nor2_2
 Xg3762 n_241 n_242 Vdd Gnd inv_2
 Xout_reg_57 clk n_239 out_57 Vdd Gnd dp_1
 Xg3760 out_58 n_236 n_240 Vdd Gnd xor2_2
 Xg3763 out_58 n_237 n_241 Vdd Gnd nand2_2
 Xg3759 reset n_235 n_239 Vdd Gnd nor2_2
 Xg3767 n_236 n_237 Vdd Gnd inv_2
 Xout_reg_56 clk n_234 out_56 Vdd Gnd dp_1
 Xg3765 out_57 n_232 n_235 Vdd Gnd xor2_2
 Xg3768 out_57 n_233 n_236 Vdd Gnd nand2_2
 Xg3764 reset n_231 n_234 Vdd Gnd nor2_2
 Xg3772 n_232 n_233 Vdd Gnd inv_2
 Xout_reg_55 clk n_230 out_55 Vdd Gnd dp_1
 Xg3770 out_56 n_227 n_231 Vdd Gnd xor2_2
 Xg3773 out_56 n_228 n_232 Vdd Gnd nand2_2
 Xg3769 reset n_226 n_230 Vdd Gnd nor2_2
 Xg3777 n_227 n_228 Vdd Gnd inv_2
 Xout_reg_54 clk n_225 out_54 Vdd Gnd dp_1
```

```
Xg3778 out_55 n_224 n_227 Vdd Gnd nand2_2
Xg3775 out_55 n_223 n_226 Vdd Gnd xor2_2
Xg3774 reset n_222 n_225 Vdd Gnd nor2_2
Xg3782 n_223 n_224 Vdd Gnd inv_2
Xout_reg_53 clk n_221 out_53 Vdd Gnd dp_1
Xg3780 out_54 n_219 n_222 Vdd Gnd xor2_2
Xg3783 out_54 n_220 n_223 Vdd Gnd nand2_2
Xg3779 reset n_218 n_221 Vdd Gnd nor2_2
Xg3787 n_219 n_220 Vdd Gnd inv_2
Xout_reg_52 clk n_217 out_52 Vdd Gnd dp_1
Xg3785 out_53 n_214 n_218 Vdd Gnd xor2_2
Xg3788 out_53 n_215 n_219 Vdd Gnd nand2_2
Xg3784 reset n_213 n_217 Vdd Gnd nor2_2
Xg3792 n_214 n_215 Vdd Gnd inv_2
Xout_reg_51 clk n_212 out_51 Vdd Gnd dp_1
Xg3790 out_52 n_210 n_213 Vdd Gnd xor2_2
Xg3793 out_52 n_211 n_214 Vdd Gnd nand2_2
Xg3789 reset n_209 n_212 Vdd Gnd nor2_2
Xg3797 n_210 n_211 Vdd Gnd inv_2
Xout_reg_50 clk n_208 out_50 Vdd Gnd dp_1
Xg3795 out_51 n_206 n_209 Vdd Gnd xor2_2
Xg3798 out_51 n_207 n_210 Vdd Gnd nand2_2
Xg3794 reset n_205 n_208 Vdd Gnd nor2_2
Xout_reg_49 clk n_204 out_49 Vdd Gnd dp_1
Xg3802 n_206 n_207 Vdd Gnd inv_2
Xg3800 out_50 n_201 n_205 Vdd Gnd xor2_2
Xg3803 out_50 n_202 n_206 Vdd Gnd nand2_2
Xg3799 reset n_200 n_204 Vdd Gnd nor2_2
Xg3807 n_201 n_202 Vdd Gnd inv_2
Xout_reg_48 clk n_199 out_48 Vdd Gnd dp_1
Xg3805 out_49 n_196 n_200 Vdd Gnd xor2_2
Xg3808 out_49 n_197 n_201 Vdd Gnd nand2_2
Xg3804 reset n_195 n_199 Vdd Gnd nor2_2
Xg3812 n_196 n_197 Vdd Gnd inv_2
Xout_reg_47 clk n_194 out_47 Vdd Gnd dp_1
Xg3810 out_48 n_192 n_195 Vdd Gnd xor2_2
Xg3813 out_48 n_193 n_196 Vdd Gnd nand2_2
Xg3809 reset n_191 n_194 Vdd Gnd nor2_2
Xg3817 n_192 n_193 Vdd Gnd inv_2
Xout_reg_46 clk n_190 out_46 Vdd Gnd dp_1
Xg3815 out_47 n_188 n_191 Vdd Gnd xor2_2
Xg3818 out_47 n_189 n_192 Vdd Gnd nand2_2
Xg3814 reset n_187 n_190 Vdd Gnd nor2_2
Xg3822 n_188 n_189 Vdd Gnd inv_2
Xout_reg_45 clk n_186 out_45 Vdd Gnd dp_1
Xg3823 out_46 n_185 n_188 Vdd Gnd nand2_2
Xg3820 out_46 n_184 n_187 Vdd Gnd xor2_2
Xg3819 reset n_183 n_186 Vdd Gnd nor2_2
Xg3827 n_184 n_185 Vdd Gnd inv_2
Xout_reg_44 clk n_182 out_44 Vdd Gnd dp_1
Xg3825 out_45 n_179 n_183 Vdd Gnd xor2_2
Xg3828 out_45 n_180 n_184 Vdd Gnd nand2_2
```

```
Xg3824 reset n_178 n_182 Vdd Gnd nor2_2
Xg3832 n_179 n_180 Vdd Gnd inv_2
Xout_reg_43 clk n_177 out_43 Vdd Gnd dp_1
Xg3830 out_44 n_174 n_178 Vdd Gnd xor2_2
Xg3833 out_44 n_175 n_179 Vdd Gnd nand2_2
Xg3829 reset n_173 n_177 Vdd Gnd nor2_2
Xg3837 n_174 n_175 Vdd Gnd inv_2
Xout_reg_42 clk n_172 out_42 Vdd Gnd dp_1
Xg3835 out_43 n_170 n_173 Vdd Gnd xor2_2
Xg3838 out_43 n_171 n_174 Vdd Gnd nand2_2
Xg3834 reset n_169 n_172 Vdd Gnd nor2_2
Xg3842 n_170 n_171 Vdd Gnd inv_2
Xout_reg_41 clk n_168 out_41 Vdd Gnd dp_1
Xg3840 out_42 n_166 n_169 Vdd Gnd xor2_2
Xg3843 out_42 n_167 n_170 Vdd Gnd nand2_2
Xg3839 reset n_165 n_168 Vdd Gnd nor2_2
Xg3847 n_166 n_167 Vdd Gnd inv_2
Xout_reg_40 clk n_164 out_40 Vdd Gnd dp_1
Xg3845 out_41 n_162 n_165 Vdd Gnd xor2_2
Xg3848 out_41 n_163 n_166 Vdd Gnd nand2_2
Xg3844 reset n_161 n_164 Vdd Gnd nor2_2
Xg3852 n_162 n_163 Vdd Gnd inv_2
Xout_reg_39 clk n_160 out_39 Vdd Gnd dp_1
Xg3850 out_40 n_158 n_161 Vdd Gnd xor2_2
Xg3853 out_40 n_159 n_162 Vdd Gnd nand2_2
Xg3849 reset n_157 n_160 Vdd Gnd nor2_2
Xg3857 n_158 n_159 Vdd Gnd inv_2
Xout_reg_38 clk n_156 out_38 Vdd Gnd dp_1
Xg3855 out_39 n_154 n_157 Vdd Gnd xor2_2
Xg3858 out_39 n_155 n_158 Vdd Gnd nand2_2
Xg3854 reset n_153 n_156 Vdd Gnd nor2_2
Xg3862 n_154 n_155 Vdd Gnd inv_2
Xout_reg_37 clk n_152 out_37 Vdd Gnd dp_1
Xg3860 out_38 n_150 n_153 Vdd Gnd xor2_2
Xg3863 out_38 n_151 n_154 Vdd Gnd nand2_2
Xg3859 reset n_149 n_152 Vdd Gnd nor2_2
Xg3867 n_150 n_151 Vdd Gnd inv_2
Xout_reg_36 clk n_148 out_36 Vdd Gnd dp_1
Xg3865 out_37 n_146 n_149 Vdd Gnd xor2_2
Xg3868 out_37 n_147 n_150 Vdd Gnd nand2_2
Xg3864 reset n_145 n_148 Vdd Gnd nor2_2
Xg3872 n_146 n_147 Vdd Gnd inv_2
Xout_reg_35 clk n_144 out_35 Vdd Gnd dp_1
Xg3870 out_36 n_142 n_145 Vdd Gnd xor2_2
Xg3873 out_36 n_143 n_146 Vdd Gnd nand2_2
Xg3869 reset n_141 n_144 Vdd Gnd nor2_2
Xg3877 n_142 n_143 Vdd Gnd inv_2
Xout_reg_34 clk n_140 out_34 Vdd Gnd dp_1
Xg3875 out_35 n_138 n_141 Vdd Gnd xor2_2
Xg3878 out_35 n_139 n_142 Vdd Gnd nand2_2
Xg3874 reset n_137 n_140 Vdd Gnd nor2_2
Xout_reg_33 clk n_136 out_33 Vdd Gnd dp_1
```

```
Xg3882 n_138 n_139 Vdd Gnd inv_2
Xg3880 out_34 n_134 n_137 Vdd Gnd xor2_2
Xg3883 out_34 n_135 n_138 Vdd Gnd nand2_2
Xg3879 reset n_133 n_136 Vdd Gnd nor2_2
Xg3887 n_134 n_135 Vdd Gnd inv_2
Xout_reg_32 clk n_132 out_32 Vdd Gnd dp_1
Xg3885 out_33 n_130 n_133 Vdd Gnd xor2_2
Xg3888 out_33 n_131 n_134 Vdd Gnd nand2_2
Xg3884 reset n_129 n_132 Vdd Gnd nor2_2
Xg3892 n_130 n_131 Vdd Gnd inv_2
Xout_reg_31 clk n_128 out_31 Vdd Gnd dp_1
Xg3890 out_32 n_126 n_129 Vdd Gnd xor2_2
Xg3893 out_32 n_127 n_130 Vdd Gnd nand2_2
Xg3889 reset n_125 n_128 Vdd Gnd nor2_2
Xg3897 n_126 n_127 Vdd Gnd inv_2
Xout_reg_30 clk n_124 out_30 Vdd Gnd dp_1
Xg3895 out_31 n_122 n_125 Vdd Gnd xor2_2
Xg3898 out_31 n_123 n_126 Vdd Gnd nand2_2
Xg3894 reset n_121 n_124 Vdd Gnd nor2_2
Xout_reg_29 clk n_120 out_29 Vdd Gnd dp_1
Xg3902 n_122 n_123 Vdd Gnd inv_2
Xg3900 out_30 n_118 n_121 Vdd Gnd xor2_2
Xg3903 out_30 n_119 n_122 Vdd Gnd nand2_2
Xg3899 reset n_117 n_120 Vdd Gnd nor2_2
Xg3907 n_118 n_119 Vdd Gnd inv_2
Xout_reg_28 clk n_116 out_28 Vdd Gnd dp_1
Xg3905 out_29 n_114 n_117 Vdd Gnd xor2_2
Xg3908 out_29 n_115 n_118 Vdd Gnd nand2_2
Xg3904 reset n_113 n_116 Vdd Gnd nor2_2
Xout_reg_27 clk n_112 out_27 Vdd Gnd dp_1
Xg3912 n_114 n_115 Vdd Gnd inv_2
Xg3910 out_28 n_110 n_113 Vdd Gnd xor2_2
Xg3913 out_28 n_111 n_114 Vdd Gnd nand2_2
Xg3909 reset n_109 n_112 Vdd Gnd nor2_2
Xg3917 n_110 n_111 Vdd Gnd inv_2
Xout_reg_26 clk n_108 out_26 Vdd Gnd dp_1
Xg3915 out_27 n_106 n_109 Vdd Gnd xor2_2
Xg3918 out_27 n_107 n_110 Vdd Gnd nand2_2
Xg3914 reset n_105 n_108 Vdd Gnd nor2_2
Xg3922 n_106 n_107 Vdd Gnd inv_2
Xout_reg_25 clk n_104 out_25 Vdd Gnd dp_1
Xg3920 out_26 n_102 n_105 Vdd Gnd xor2_2
Xg3923 out_26 n_103 n_106 Vdd Gnd nand2_2
Xg3919 reset n_101 n_104 Vdd Gnd nor2_2
Xg3927 n_102 n_103 Vdd Gnd inv_2
Xout_reg_24 clk n_100 out_24 Vdd Gnd dp_1
Xg3925 out_25 n_98 n_101 Vdd Gnd xor2_2
Xg3928 out_25 n_99 n_102 Vdd Gnd nand2_2
Xg3924 reset n_97 n_100 Vdd Gnd nor2_2
Xg3932 n_98 n_99 Vdd Gnd inv_2
Xout_reg_23 clk n_96 out_23 Vdd Gnd dp_1
Xg3930 out_24 n_94 n_97 Vdd Gnd xor2_2
```

```
Xg3933 out_24 n_95 n_98 Vdd Gnd nand2_2
Xg3929 reset n_93 n_96 Vdd Gnd nor2_2
Xg3937 n_94 n_95 Vdd Gnd inv_2
Xout_reg_22 clk n_92 out_22 Vdd Gnd dp_1
Xg3935 out_23 n_90 n_93 Vdd Gnd xor2_2
Xg3938 out_23 n_91 n_94 Vdd Gnd nand2_2
Xg3934 reset n_89 n_92 Vdd Gnd nor2_2
Xg3942 n_90 n_91 Vdd Gnd inv_2
Xout_reg_21 clk n_88 out_21 Vdd Gnd dp_1
Xg3940 out_22 n_86 n_89 Vdd Gnd xor2_2
Xg3943 out_22 n_87 n_90 Vdd Gnd nand2_2
Xg3939 reset n_85 n_88 Vdd Gnd nor2_2
Xg3947 n_86 n_87 Vdd Gnd inv_2
Xout_reg_20 clk n_84 out_20 Vdd Gnd dp_1
Xg3945 out_21 n_82 n_85 Vdd Gnd xor2_2
Xg3948 out_21 n_83 n_86 Vdd Gnd nand2_2
Xg3944 reset n_81 n_84 Vdd Gnd nor2_2
Xg3952 n_82 n_83 Vdd Gnd inv_2
Xout_reg_19 clk n_80 out_19 Vdd Gnd dp_1
Xg3950 out_20 n_78 n_81 Vdd Gnd xor2_2
Xg3953 out_20 n_79 n_82 Vdd Gnd nand2_2
Xg3949 reset n_77 n_80 Vdd Gnd nor2_2
Xg3957 n_78 n_79 Vdd Gnd inv_2
Xout_reg_18 clk n_76 out_18 Vdd Gnd dp_1
Xg3955 out_19 n_74 n_77 Vdd Gnd xor2_2
Xg3958 out_19 n_75 n_78 Vdd Gnd nand2_2
Xg3954 reset n_73 n_76 Vdd Gnd nor2_2
Xg3962 n_74 n_75 Vdd Gnd inv_2
Xout_reg_17 clk n_72 out_17 Vdd Gnd dp_1
Xg3960 out_18 n_70 n_73 Vdd Gnd xor2_2
Xg3963 out_18 n_71 n_74 Vdd Gnd nand2_2
Xg3959 reset n_69 n_72 Vdd Gnd nor2_2
Xg3967 n_70 n_71 Vdd Gnd inv_2
Xout_reg_16 clk n_68 out_16 Vdd Gnd dp_1
Xg3965 out_17 n_66 n_69 Vdd Gnd xor2_2
Xg3968 out_17 n_67 n_70 Vdd Gnd nand2_2
Xg3964 reset n_65 n_68 Vdd Gnd nor2_2
Xg3972 n_66 n_67 Vdd Gnd inv_2
Xout_reg_15 clk n_64 out_15 Vdd Gnd dp_1
Xg3970 out_16 n_62 n_65 Vdd Gnd xor2_2
Xg3973 out_16 n_63 n_66 Vdd Gnd nand2_2
Xg3969 reset n_61 n_64 Vdd Gnd nor2_2
Xout_reg_14 clk n_60 out_14 Vdd Gnd dp_1
Xg3977 n_62 n_63 Vdd Gnd inv_2
Xg3975 out_15 n_58 n_61 Vdd Gnd xor2_2
Xg3978 out_15 n_59 n_62 Vdd Gnd nand2_2
Xg3974 reset n_57 n_60 Vdd Gnd nor2_2
Xg3982 n_58 n_59 Vdd Gnd inv_2
Xout_reg_13 clk n_56 out_13 Vdd Gnd dp_1
Xg3980 out_14 n_54 n_57 Vdd Gnd xor2_2
Xg3983 out_14 n_55 n_58 Vdd Gnd nand2_2
Xg3979 reset n_53 n_56 Vdd Gnd nor2_2
```

```
Xg3987 n_54 n_55 Vdd Gnd inv_2
Xout_reg_12 clk n_52 out_12 Vdd Gnd dp_1
Xg3985 out_13 n_50 n_53 Vdd Gnd xor2_2
Xg3988 out_13 n_51 n_54 Vdd Gnd nand2_2
Xg3984 reset n_49 n_52 Vdd Gnd nor2_2
Xout_reg_11 clk n_48 out_11 Vdd Gnd dp_1
Xg3992 n_50 n_51 Vdd Gnd inv_2
Xg3990 out_12 n_46 n_49 Vdd Gnd xor2_2
Xg3993 out_12 n_47 n_50 Vdd Gnd nand2_2
Xg3989 reset n_45 n_48 Vdd Gnd nor2_2
Xg3997 n_46 n_47 Vdd Gnd inv_2
Xout_reg_10 clk n_44 out_10 Vdd Gnd dp_1
Xg3995 out_11 n_42 n_45 Vdd Gnd xor2_2
Xg3998 out_11 n_43 n_46 Vdd Gnd nand2_2
Xg3994 reset n_41 n_44 Vdd Gnd nor2_2
Xg4002 n_42 n_43 Vdd Gnd inv_2
Xout_reg_9 clk n_40 out_9 Vdd Gnd dp_1
Xg4000 out_10 n_38 n_41 Vdd Gnd xor2_2
Xg4003 out_10 n_39 n_42 Vdd Gnd nand2_2
Xg3999 reset n_37 n_40 Vdd Gnd nor2_2
Xg4007 n_38 n_39 Vdd Gnd inv_2
Xout_reg_8 clk n_36 out_8 Vdd Gnd dp_1
Xg4008 out_9 n_35 n_38 Vdd Gnd nand2_2
Xg4005 out_9 n_34 n_37 Vdd Gnd xor2_2
Xg4004 reset n_33 n_36 Vdd Gnd nor2_2
Xout_reg_7 clk n_32 out_7 Vdd Gnd dp_1
Xg4012 n_34 n_35 Vdd Gnd inv_2
Xg4013 out_8 n_31 n_34 Vdd Gnd nand2_2
Xg4010 out_8 n_30 n_33 Vdd Gnd xor2_2
Xg4009 reset n_29 n_32 Vdd Gnd nor2_2
Xg4017 n_30 n_31 Vdd Gnd inv_2
Xout_reg_6 clk n_28 out_6 Vdd Gnd dp_1
Xg4015 out_7 n_26 n_29 Vdd Gnd xor2_2
Xg4018 out_7 n_27 n_30 Vdd Gnd nand2_2
Xg4014 reset n_25 n_28 Vdd Gnd nor2_2
Xg4022 n_26 n_27 Vdd Gnd inv_2
Xout_reg_5 clk n_24 out_5 Vdd Gnd dp_1
Xg4020 out_6 n_22 n_25 Vdd Gnd xor2_2
Xg4023 out_6 n_23 n_26 Vdd Gnd nand2_2
Xg4019 reset n_21 n_24 Vdd Gnd nor2_2
Xg4027 n_22 n_23 Vdd Gnd inv_2
Xout_reg_4 clk n_20 out_4 Vdd Gnd dp_1
Xg4025 out_5 n_18 n_21 Vdd Gnd xor2_2
Xg4028 out_5 n_19 n_22 Vdd Gnd nand2_2
Xg4024 reset n_17 n_20 Vdd Gnd nor2_2
Xout_reg_3 clk n_16 out_3 Vdd Gnd dp_1
Xg4032 n_18 n_19 Vdd Gnd inv_2
Xg4030 out_4 n_14 n_17 Vdd Gnd xor2_2
Xg4033 out_4 n_15 n_18 Vdd Gnd nand2_2
Xg4029 reset n_13 n_16 Vdd Gnd nor2_2
Xout_reg_2 clk n_12 out_2 Vdd Gnd dp_1
Xg4036 n_14 n_15 Vdd Gnd inv_2
```

```
 Xg4035 out_3 n_10 n_13 Vdd Gnd xor2_2
 Xg4037 out_3 n_11 n_14 Vdd Gnd nand2_2
 Xout_reg_1 clk n_9 out_1 Vdd Gnd dp_1
 Xg4034 reset n_8 n_12 Vdd Gnd nor2_2
 Xout_reg_0 clk n_7 out_0 Vdd Gnd dp_1
 Xg4041 n_10 n_11 Vdd Gnd inv_2
 Xg4038 reset n_6 n_9 Vdd Gnd nor2_2
 Xg4039 n_4 out_2 n_8 Vdd Gnd xor2_2
 Xg4042 out_2 n_5 n_10 Vdd Gnd nand2_2
 Xg4044 reset out_0 n_7 Vdd Gnd nor2_2
 Xg4043 out_0 out_1 n_6 Vdd Gnd xnor2_2
 Xg4045 n_4 n_5 Vdd Gnd inv_2
 Xg4046 out_1 out_0 n_4 Vdd Gnd nand2_2

.Include model8.mod
.Include standard_cell_new.dat

.option post=1 accurate=1 symb=1
.op all
.TRAN 25n 1.5u
.print TRAN v(out_0) v(out_1) v(out_2) v(out_3) v(out_4)
+v(out_5) v(out_6) v(out_7) v(out_8) v(out_9)
+v(out_10) v(out_11) v(out_12) v(out_13) v(out_14)
.plot TRAN v(out_0) v(out_1) v(out_2) v(out_3) v(out_4)
+v(out_5) v(out_6) v(out_7) v(out_8) v(out_9)
+v(out_10) v(out_11) v(out_12) v(out_13) v(out_14)
.END
```

A5: BSIM3v3 Models Utilized for Testing:

```
* DATE: May  5/10
* LOT: V01Z                  WAF: 7007
* Temperature_parameters=Default
.MODEL NM NMOS (                                LEVEL   = 8
+VERSION = 3.1           TNOM    = 27            TOX     = 5.7E-9
+XJ      = 1E-7          NCH     = 2.3549E17     VTH0    = 0.3865501
+K1      = 0.4719559     K2      = 3.869181E-6   K3      = 1E-3
+K3B     = 2.0172167     W0      = 1E-7          NLX     =
1.893538E-7
+DVT0W   = 0             DVT1W   = 0             DVT2W   = 0
+DVT0    = 1.0569689     DVT1    = 0.5068122     DVT2    = -
0.0331345
+U0      = 311.148287    UA      = -1.2388E-9    UB      =
2.539912E-18
+UC      = 4.072175E-11  VSAT    = 1.285416E5    A0      = 1.9635421
+AGS     = 0.3718654     B0      = -1.982096E-8  B1      = 0
+KETA    = -0.0126039    A1      = 1.787007E-4   A2      = 0.5219778
+RDSW    = 167.3631056   PRWG    = 0.4987461     PRWB    = -0.2
+WR      = 1             WINT    = 0             LINT    =
1.689392E-9
+XL      = 0             XW      = -4E-8         DWG     = -
1.924072E-8
+DWB     = 2.01265E-9    VOFF    = -0.1024002    NFACTOR = 1.1874144
+CIT     = 0             CDSC    = 2.4E-4        CDSCD   = 0
+CDSCB   = 0             ETA0    = 6.201169E-3   ETAB    =
6.889911E-5
+DSUB    = 0.0453464     PCLM    = 1.6166621     PDIBLC1 = 0.9630962
+PDIBLC2 = 2.722713E-3   PDIBLCB = 0.1           DROUT   = 1
+PSCBE1  = 6.834619E8    PSCBE2  = 2.334871E-4   PVAG    =
9.828743E-3
+DELTA   = 0.01          RSH     = 3.9           MOBMOD  = 1
+PRT     = 0             UTE     = -1.5          KT1     = -0.11
+KT1L    = 0             KT2     = 0.022         UA1     = 4.31E-9
+UB1     = -7.61E-18     UC1     = -5.6E-11      AT      = 3.3E4
+WL      = 0             WLN     = 1             WW      = 0
+WWN     = 1             WWL     = 0             LL      = 0
+LLN     = 1             LW      = 0             LWN     = 1
+LWL     = 0             CAPMOD  = 2             XPART   = 0.5
+CGDO    = 4.18E-10      CGSO    = 4.18E-10      CGBO    = 1E-12
+CJ      = 1.864911E-3   PB      = 0.99          MJ      = 0.4832148
+CJSW    = 4.310637E-10  PBSW    = 0.8           MJSW    = 0.3140839
+CJSWG   = 3.29E-10      PBSWG   = 0.8           MJSWG   = 0.3140839
+CF      = 0             PVTH0   = -3.610364E-3  PRDSW   = -10
+PK2     = 2.965885E-3   WKETA   = 0.0114362     LKETA   =
2.627895E-3     )
*
.MODEL PM PMOS (                                LEVEL   = 8
+VERSION = 3.1           TNOM    = 27            TOX     = 5.7E-9
```

```
+XJ      = 1E-7           NCH     = 4.1589E17      VTH0    = -
0.5613236
+K1      = 0.6414791      K2      = -7.998217E-4   K3      = 0.0974888
+K3B     = 5.9844014      W0      = 1E-6           NLX     =
8.422015E-9
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0    = 2.3150515      DVT1    = 0.8309842      DVT2    = -
0.1745928
+U0      = 100            UA      = 9.035839E-10   UB      = 1E-21
+UC      = -1E-10         VSAT    = 1.326848E5     A0      = 0.9931835
+AGS     = 0.1277387      B0      = 1.297419E-6    B1      = 5E-6
+KETA    = 0.0179736      A1      = 0.0461129      A2      = 0.3
+RDSW    = 1.384905E3     PRWG    = 0.0112159      PRWB    = -
0.1122058
+WR      = 1              WINT    = 0              LINT    = 3.25565E-
8
+XL      = 0              XW      = -4E-8          DWG     = -
3.900507E-8
+DWB     = -3.965031E-9   VOFF    = -0.1146119     NFACTOR = 1.2317619
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0              ETA0    = 0.2009717      ETAB    = -
0.2535502
+DSUB    = 1.0057681      PCLM    = 1.3709724      PDIBLC1 = 5.25257E-
3
+PDIBLC2 = -1E-5          PDIBLCB = -6.466352E-4   DROUT   = 0.0738757
+PSCBE1  = 1.054541E10    PSCBE2  = 2.44303E-9     PVAG    = 0
+DELTA   = 0.01           RSH     = 2.9            MOBMOD  = 1
+PRT     = 0              UTE     = -1.5           KT1     = -0.11
+KT1L    = 0              KT2     = 0.022          UA1     = 4.31E-9
+UB1     = -7.61E-18      UC1     = -5.6E-11       AT      = 3.3E4
+WL      = 0              WLN     = 1              WW      = 0
+WWN     = 1              WWL     = 0              LL      = 0
+LLN     = 1              LW      = 0              LWN     = 1
+LWL     = 0              CAPMOD  = 2              XPART   = 0.5
+CGDO    = 5.06E-10       CGSO    = 5.06E-10       CGBO    = 1E-12
+CJ      = 1.893734E-3    PB      = 0.9889579      MJ      = 0.4705132
+CJSW    = 3.124347E-10   PBSW    = 0.8            MJSW    = 0.2786992
+CJSWG   = 2.5E-10        PBSWG   = 0.8            MJSWG   = 0.2786992
+CF      = 0              PVTH0   = 3.966648E-3    PRDSW   = -
15.6092787
+PK2     = 1.822963E-3    WKETA   = 0.0158334      LKETA   = -
5.813077E-3    )
*
```

Vita

Matthew C. Markulik was born on November 26, 1981 in Albuquerque, New Mexico. The second born son of Charles and Brenda Markulik, he graduated high school from Tularosa High School in Tularosa, NM. He entered the United States Army in 2000 and became a combat veteran participating in Operation Iraqi Freedom and received several accommodations for his efforts. Upon completion of his military service he worked in the field of electronics technician for several government contractors. In the spring of 2006 Matthew entered into his degree plan and graduated with a BSEE from the University of Texas at El Paso in May 2010 with a concentration in device physics. Immediately upon graduation he entered the Masters program at the same university with an emphasis in digital circuit design.

Permanent address: 3237 Mountain Ridge Dr.
El Paso , TX 79904

This thesis was typed by Matthew Markulik.