A PHYSICAL IMPLEMENTATION WITH CUSTOM LOW POWER EXTENSIONS OF A RECONFIGURABLE HARDWARE FABRIC

by

Gerold Joseph Dhanabalan

B.E.(Hons.), Birla Institute of Technology and Science, Pilani, 2002

Submitted to the Graduate Faculty of

Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical Engineering

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This thesis was presented

by

Gerold Joseph Dhanabalan

It was defended on

March 05, 2008

and approved by

Dr.Alex K. Jones, Assistant Professor, Electrical and Computer Engineering Department

Dr. Steven P. Levitan, John A. Jurenko Professor, Electrical and Computer Engineering Department

Dr.Jun Yang, Assistant Professor, Electrical and Computer Engineering Department

Thesis Advisor: Dr.Alex K. Jones, Assistant Professor, Electrical and Computer Engineering Department

A PHYSICAL IMPLEMENTATION WITH CUSTOM LOW POWER EXTENSIONS OF A RECONFIGURABLE HARDWARE FABRIC

Gerold Joseph Dhanabalan, M.S.

University of Pittsburgh, 2008

The primary focus of this thesis is on the physical implementation of the SuperCISC Reconfigurable Hardware Fabric (RHF). The SuperCISC RHF provides a fast time to market solution that approximates the benefits of an ASIC (Application Specific Integrated Circuit) while retaining the design flow of an embedded software system. The fabric which consists of computational ALU stripes and configurable multiplexer based interconnect stripes has been implemented in the IBM 0.13um CMOS process using Cadence SoC Encounter.

As the entire hardware fabric utilizes a combinational flow, glitching power consumption is a potential problem inherent to the fabric. A CMOS thyristor based programmable delay element has been designed in the IBM 0.13um CMOS process, to minimize the glitch power consumed in the hardware fabric. The delay element was characterized for use in the IBM standard cell library to synthesize standard cell ASIC designs requiring this capability such as the SuperCISC fabric. The thesis also introduces a power-gated memory solution, which can be used to increase the size of an EEPROM memory for use in SoC style applications. A macromodel of the EEPROM has been used to model the erase, program and read characteristics of the EEPROM. This memory is designed for use in the fabric for storing encryption keys, etc.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

"Many, O Jehovah my God, are the wonderful works which You have done, and Your thoughts toward us; There is none to compare with You. If I would declare and speak of them, they would be too numerous to count"

Psalm 40:5

As I pen this down, I think of all the great works God has done in my life. As a Father cares for his child, so has He protected and cared for me. All glory and honor be to God who has blessed every work of my hand and has made this education possible. I dedicate this thesis to Lord Jesus Christ who has been my strength throughout my life.

I like to express my sincere gratitude to Dr.Alex K. Jones for giving me an opportunity to work with him on my thesis. He has been a constant source of encouragement throughout the course of the research and has constantly supported my interests in analog circuit design. My sincere thanks for all the help and support he has given throughout the course of my study.

I like to express my thanks to Dr. Steven P. Levitan who has given me a strong foundation for a career in VLSI. The design basics that he has given have taken me a long way . I like to express my thanks to Dr.Jun Yang for being a part of my thesis defense committee and helping me out in my courses. I consider it a great previlege to be a student in many of her classes.

I like to express my sincere thanks to my parents Dhanabalan, Alphonse Mary and my wife Pio Clara who have shared my burden and have always encouraged me in my difficult times. Apart from her motivation, I also thank Pio for her valuable technical suggestions. I also like to thank my brothers Charles Daniel and Richard Carol, who have always helped me keep up with the stresses of graduate studies. I also like to thank my uncle Gnanasekar and aunt Anthuvan Grace for their constant support.

I also like to thank Dr.J.T. (Tom) Cain and Sandy Weisberg for their help with all my academic related issues. My special thanks to Sam Dickerson for his help with all HSPICE simulations. I like to thank Bill McGahey who has been the key person to contact customer support to get many CAD tool related issues solved. I also like to thank Gayatri Mehta for her help with the Prime Power simulations and Hariram for his help with the documentation.

I like to thank my friends Suresh and Gopinath who have helped me in my studies. I also like to thank Ajay Ogirala who has always been my “academic yellow pages”. I also like to thank my friends Anish, Gladys, Anand, Christy, Joshua, Manoj, Jasmine, Ben, Mekha, Ratnakar, Sabrina, Pastor Dave, Laura, Pastor Prabhu Isaac and family who have always prayed for me.

1.0 INTRODUCTION

Technological advances in multimedia have increased the demand on high performance systems. Hardware acceleration has become a necessity to cope with the challenges in processing speed especially for signal processing applications. On the other hand, market trends have shown that the economic success of a product is primarily determined by its time to market. Design of ASICs (Application Specific Integrated Circuits) based on a standard ASIC flow has always shown better performance and power characteristics at the expense of increased time-to-market.

ASICs are the choice when huge design times can be tolerated [1]. However, with the market for many applications, especially the consumer electronic products being very volatile, it is necessary that these products reach the market faster for a huge return on investment. Such a market driven by design time has led to the increased use of reconfigurable devices. FPGAs (Field Programmable Gate Arrays) are the most commonly used reconfigurable devices. Although FPGAs reduce the design time tremendously, the power consumed by the FPGAs prevents them from being used for battery-powered applications, which demand low power.

The wireless and mobile market has made the highest impact on the low-power design space. With consumer demands increasing at an exponential rate for low-power electronic gadgets, the research community has interesting opportunities to explore high performance low-power designs. This interest in developing a high-performance, low-power, reconfigurable hardware device led to the birth of an energy efficient SuperCISC Reconfigurable Hardware fabric. The objective of the SuperCISC RHF research is to develop a hardware design that has a reconfigurable architecture, high performance and low power [2] [3]. The RHF triangle as shown in Figure 1-1 below illustrates the three key design parameters that have been optimized by the design of the SuperCISC RHF.

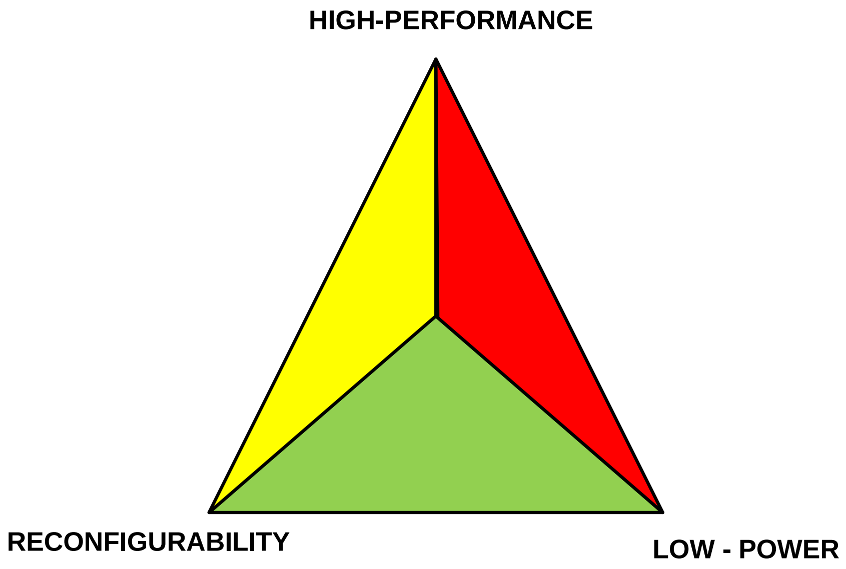


Figure 1-1: SuperCISC Reconfigurable Hardware Fabric Triangle

The work described in the document is about the physical implementation of the SuperCISC reconfigurable hardware fabric of a specific instance called 20X18. However, automation scripts have been developed to implement other instances of the hardware fabric. The physical implementation of the design is the final stage in the SuperCISC reconfigurable hardware design flow and is one of the most crucial stages in the design flow. The physical design of the SuperCISC RHF is essential to fabricate and commercialize the IP (Intellectual

Property) block. The physical design flow actually transforms the synthesized design into the layout that gets fabricated. Proper design considerations at the layout level are of utmost importance for the commercial success of the IP block.

The pre-layout power characteristics of the IP block, estimated after synthesizing the Verilog/VHDL netlist can use wireload models to model the interconnect parasitics. Such a model could be pessimistic or optimisitic dependent on the vendor library. For an accurate power estimation it is necessary to calculate the power consumed by the chip after placement and routing of the chip has been completed. Also, as the RHF has interconnects which are reconfigurable, it is necessary to understand the power consumed by the interconnects. Hence it is of prime importance to do a post-layout power analysis.

As the objective of the SuperCISC reconfigurable hardware fabric (RHF) is to achieve a low power reconfigurable solution, it is necessary to use custom circuit design techniques to reduce any source of wasteful power consumption in the fabric. As the SuperCISC RHF uses a complete combinational flow, glitching power is inherent to the fabric. The key design technique that has been adopted is to freeze the inputs to the computational units using latches until all the inputs to the computational unit have arrived. The latches are transparent (allow data to pass through) when enabled and hold the previous value when disabled. The 'enable' input to these latches is controlled by a delay element, which times the enabling of the latches. The value of the delay element to be used is determined using STA (Static Timing Analysis) at the mapping stage in the reconfigurable hardware fabric design flow. Hence, delay elements can be used to self-time the design. However, conventional delay elements, like the inverter chain, transmission gate, n or p-voltage controlled delay elements have a limitation on the range of delay that can be obtained. Also the signal integrity of these delay elements is significantly degraded because of the RC characteristics of the delay elements. More importantly, the power consumed by these delay elements is significant because of the short circuit power consumed by the degraded signal integrity or because of the large number of delay elements that are needed to obtain the delay.

Hence it is necessary to use a delay element that can suite the delay range requirements as well as have a low power characteristic. The CMOS thyristor based delay element is such a circuit that possesses good signal integrity properties with a wide delay range and low power. Circuit level changes to minimize the on-state and sub-threshold leakage power consumed by the CMOS thyristor based delay element have been suggested. Also, design techniques to minimize the number of delay elements used in the SuperCISC RHF are essential. So the design has been improved to make the delay element programmable for a specific range of delays.

EEPROMs have become an integral part of modern SoCs to store non-volatile information such as cryptographic keys etc. EEPROMs can be integrated with the SuperCISC reconfigurable hardware fabric to store such non-volatile information. However, the EEPROM needs to have a low power characteristic to save the power gained by the SuperCISC RHF. This thesis proposes a power gated EEPROM architecture level optimization which enables an increase in the size of the EEPROM with a minimum power overhead. A FLOTOX EEPROM macromodel described in HSPICE has been used to simulate the AC and DC characteristics of the EEPROM. Power simulations were then performed on a 2, 4, 8 bank EEPROM architecture to show the minimum power overhead in using the power-gated design.

Thus the objective of this thesis is to do a physical implementation of the SuperCISC reconfigurable hardware fabric and estimate its power characteristics after placement and routing. The thesis also shows the implementation of a CMOS thyristor based delay element that can be programmed for specific delay values to be used in the SuperCISC RHF. The final part of the thesis proposes a power gated memory architectural solution for use with non-volatile memories such as EEPROMs. Figure 1-2 shows the system-level diagram where the SuperCISC RHF, the delay elements and the EEPROM are integrated as part of the SuperCISC architecture. The glue logic interface shown in the figure is used to interface the processor core with the SuperCISC RHF and the power gated EEPROM. The control bits for programming the delay elements are a part of the control bus input to the RHF. Chapters 2, 6 and 8 contain the details of the SuperCISC RHF, delay elements and the power-gated EEPROM respectively.

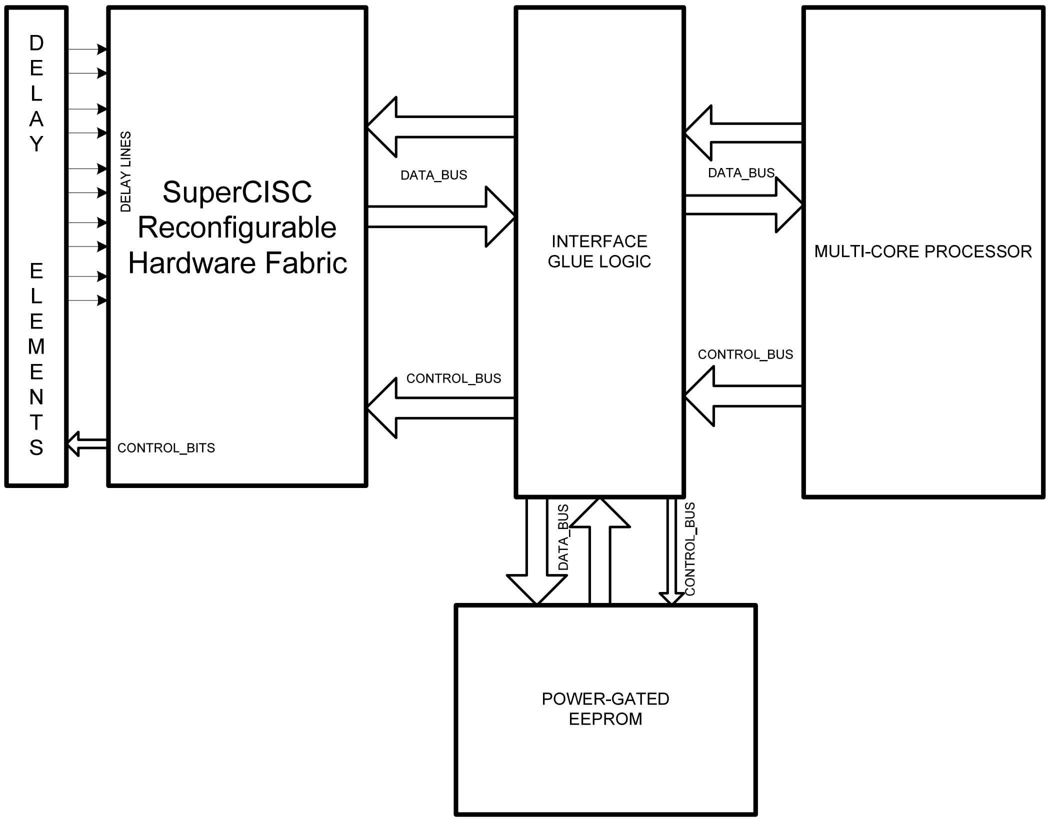


Figure 1-2: SuperCISC RHF System Integration Diagram

1.1 STATEMENT OF THE PROBLEM

The physical implementation of the SuperCISC Reconfigurable Hardware Fabric is important for the RHF design to be fabricated and commercialized. Also for an accurate power estimation of the IP block, post layout power analysis needs to be performed. As the objective of the SuperCISC RHF is to consume minimum power, it is necessary that any source of unwanted power consumption be eliminated. As the SuperCISC RHF uses a combinational flow, glitching power is inherent to the fabric. So, specialized circuits like the delay element are needed to time the design to minimize the glitching power. Also for SoCs that use an EEPROM integrated with the SuperCISC RHF, increasing the size of the EEPROM while keeping the active power low is essential. The work described in this thesis shows the methodology used for the physical implementation of the fabric. Also, the design of the CMOS delay element has been described along with the power gating technique used to increase the size of the EEPROM.

My contributions to the thesis have been on the physical implementation of the SuperCISC RHF in the IBM 0.13um CMOS technology. The parasitics extracted from the layout have been annotated to the design to estimate the post layout power consumption. I have also implemented a programmable delay element with delays of 4ns, 5ns and 7ns to be used as a standard cell in the IBM cell library. The standard cell has been characterized for various loads and transition times. To increase the size of the EEPROM while keeping the active power consumption low, a power gated memory solution has been proposed. Power measurements were conducted to show that the power-gated memory solution enables an increase in the size of the memory with a minimum power overhead.

2.0 SUPERCISC RECONFIGURABLE HARDWARE FABRIC

The SuperCISC Reconfigurable Hardware Fabric is a hardware acceleration unit, which can be embedded into the SuperCISC processor architecture [2]. The fundamental idea behind the SuperCISC architecture is to use a processor core to handle the control-intensive part of the code, while the SuperCISC RHF handles the computation intensive parts of the code, called kernels [2]. The kernels are converted into an entirely combinational hardware function generated automatically from C using a design automation flow. Using hardware predication, a Control Data Flow Graph (CDFG) can be converted into a Super Data Flow Graph (SDFG) [3] [4]. SDFG based hardware functions are asynchronous with respect to the processor core. The hardware functions generated by the SDFG get mapped onto the hardware fabric using a mapper. The entire SuperCISC RHF is a purely combinational data flow unit. Removing the sequential logic simplifies the implementation of the SDFG on the hardware fabric. Removal of sequential logic also saves operating power by avoiding clock trees and local storage elements [2]. The objective of the SuperCISC RHF is to provide a reconfigurable device which consumes low energy. The low energy consumed by the device partly comes from the limited programmability available.

2.1 ARCHITECTURE OF THE SUPERCISC RHF

The architecture of the SuperCISC RHF comprises an array of ALUs with interconnects between the rows of the ALU array. Each row in the reconfigurable fabric is called a stripe. The stripe of ALUs is called the *Computational Stripe* and the stripe of multiplexers which form the interconnect logic is called the *Interconnect Stripe*. The multiplexers in the *Interconnect Stripe* can be configured to select the appropriate operands to the ALU [3]. Figure 2-1 shows the architecture of the reconfigurable hardware fabric with computational and interconnect stripes alternating.

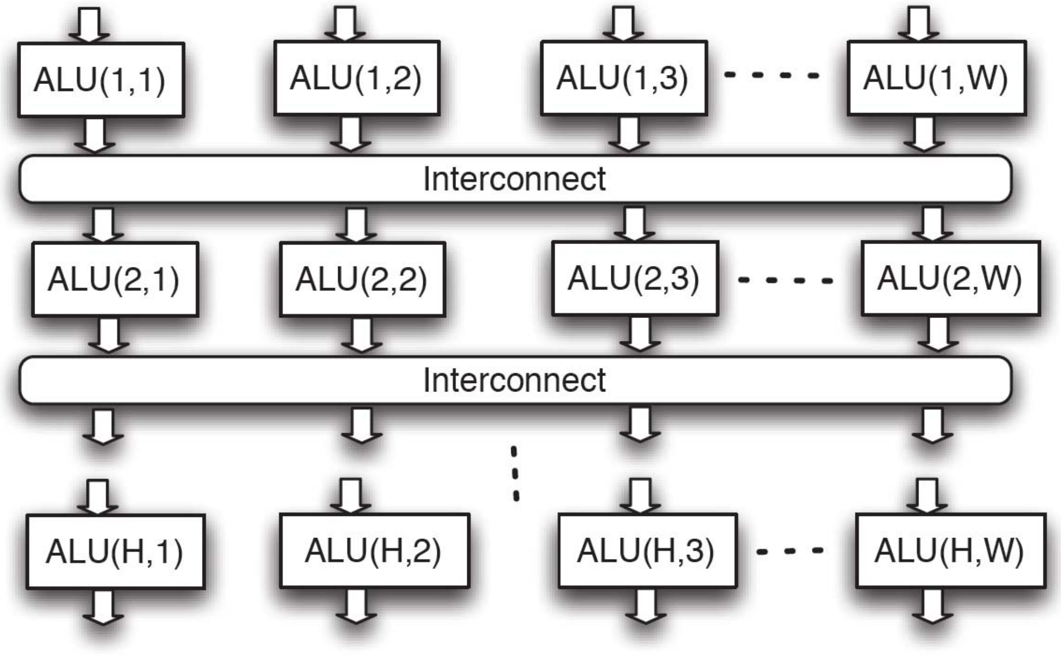


Figure 2-1: Architecture of SuperCISC Reconfigurable Hardware Fabric

The specifications of the hardware fabric are the number of ALUs per stripe and the number of *Computational (ALU)* stripes. A 20X18 configuration represents a RHF with 20 ALUs per stripe and 18 ALU stripes. The number of multiplexer stripes is one less than the number of the ALU stripes, as the ALU and MUX stripes alternate each other. The final ALU stripe in the design is followed by a special interconnect stripe called the FINALMUX stripe. The FINALMUX stripe has its inputs from the last ALU stripe and from specialized ALU stripes called “early exit” ALU stripes. The FINALMUX stripe multiplexes these inputs to the get the final output of the hardware fabric.

2.2 ARITHMETIC AND LOGIC UNIT (ALU)

The ALU performs conventional arithmetic and logical operations. In addition to the normal operations, a specialized hardware predication function is implemented within the ALU for implementing SDFGs [3] [4]. This operation requires a third single-bit operand called *predicator input* to be included in the ALU. The predicator input acts as a selector to choose one of the two operands INP1 and INP2 [3]. Figure 2-2 below shows the logical diagram of the ALU used in the hardware fabric. The control pins of the ALU select the logical operation performed by the ALU.

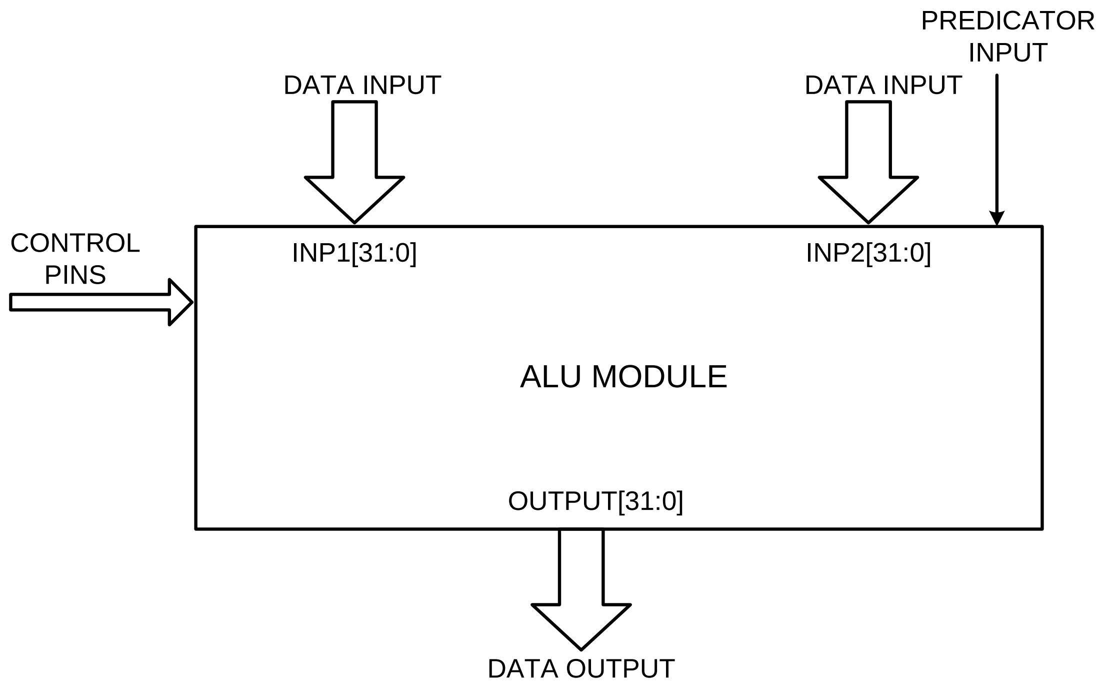


Figure 2-2: ALU Logical Diagram

2.3 INTERCONNECT MULTIPLEXER STRUCTURE (MUX)

The multiplexer implements the interconnect structure used to connect the computational stripes. One possible interconnect structure called 5:1 is shown in Figure 2-3. The 5:1 interconnect structure is basically a 5:1 multiplexer that is emulated using 4:1 multiplexers. The 5:1 multiplexer is undesirable from a power and performance perspective. Hence it is good to emulate the behavior using 4:1 multiplexers. The ALU has three operands, INP1, INP2 and the predicator input. Each operand of the ALU has a multiplexer associated with it. The multiplexer for operand INP1 selects from ALU0, ALU1, ALU2 or ALU3. The multiplexer for operand

INP2 selects from ALU1, ALU2, ALU3 or ALU4. The multiplexer for the predicator input is similar to the one for operand INP2.

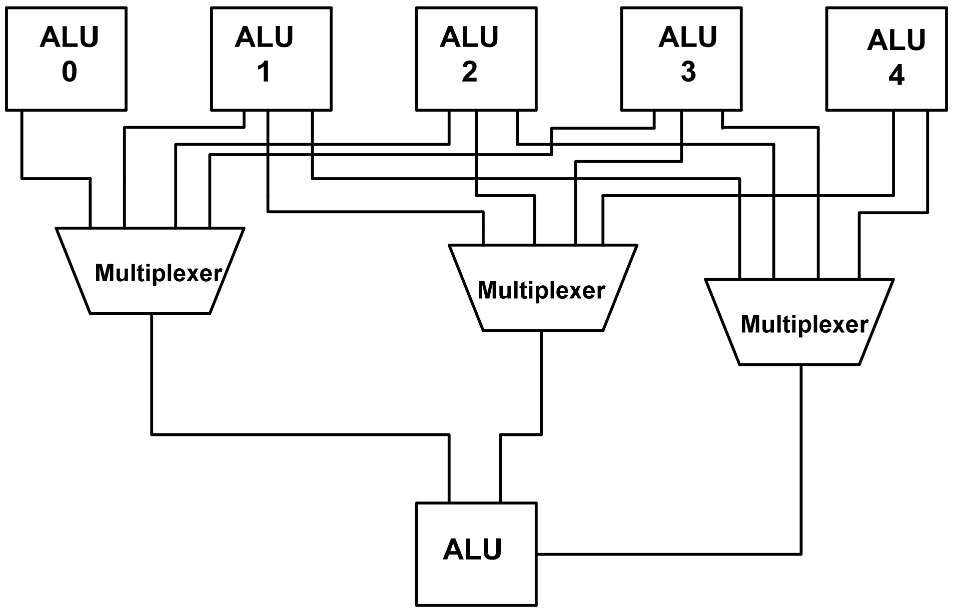


Figure 2-3: 5:1 MUX Interconnect Structure

2.4 FINAL MULTIPLEXER STRUCTURE (FINALMUX)

The third component of the hardware fabric is the FINALMUX stripe which is again a multiplexer stripe. The output of the fabric is physically connected to the output of the FINAL MUX stripe. A 4:1 FINALMUX stripe selects one out of four outputs. Let us consider a 20X18 RHF configuration with a 4:1 FINALMUX stripe and an ALU Stripe spacing of 4 as an example. For a 20X18 configuration having 18 computational stripes, the outputs can actually be derived from ALU stripe 6, ALU stripe 10, ALU stripe 14 and ALU stripe 18 for an ALU stripe spacing of 4. These connections are termed “early exits” and are a hardware provision to bypass the remaining computational stripes, for applications that need not use the remaining computational stripes. The ALU Stripe spacing determines the spacing between the “early exits” in the fabric. Figure 2-4 below shows the logical functionality of the FINALMUX stripe with inputs connected to the outputs of the respective early exits.

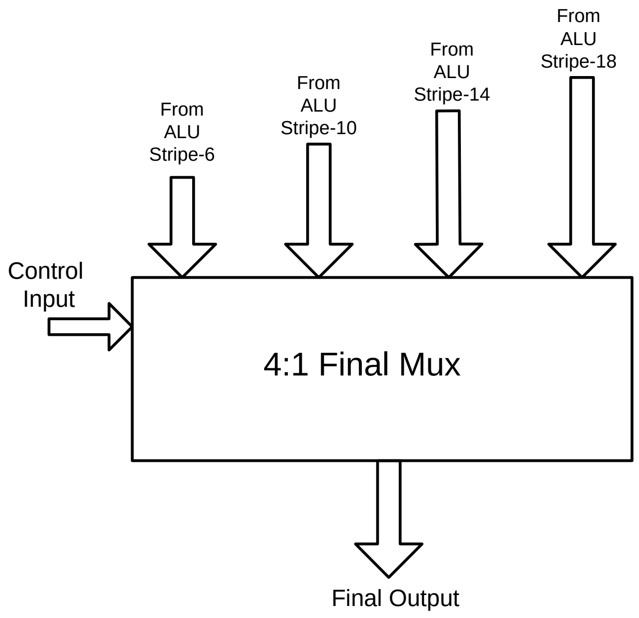


Figure 2-4: FINALMUX Stripe Logical Diagram

3.0 POWER ESTIMATION

The need for electronic gadgets to be mobile has driven the electronics industry for low-power design techniques, starting all the way from architecture level improvements down to transistor layout optimization. As the goal of the SuperCISC reconfigurable hardware fabric is to achieve high performance and low-power, it is necessary to estimate the power consumed by the fabric after synthesis at the pre-layout level as well as after the physical design at the post-layout level stage. This section of the document discusses the power consumption terminologies, the power estimation flow using Synopsys Prime Power and transistor level power estimation techniques for the delay element used in the fabric.

3.1 POWER DISSIPATION TERMINOLOGIES

The power consumed by a circuit can be broadly classified into two categories [5]:

(i) Static Power Consumption

(ii) Dynamic Power Consumption

Static Power Consumption in a CMOS circuit can be due to variety of sources, sub-threshold leakage, gate tunneling leakage and junction leakage. Dynamic power consumption can be classified into switching power and internal power. The power dissipation terminology tree shown in Figure 3-1 shows the various components of power that gets dissipated in a CMOS circuit [5]. The subsections discuss in detail about the various terminologies used in the design.

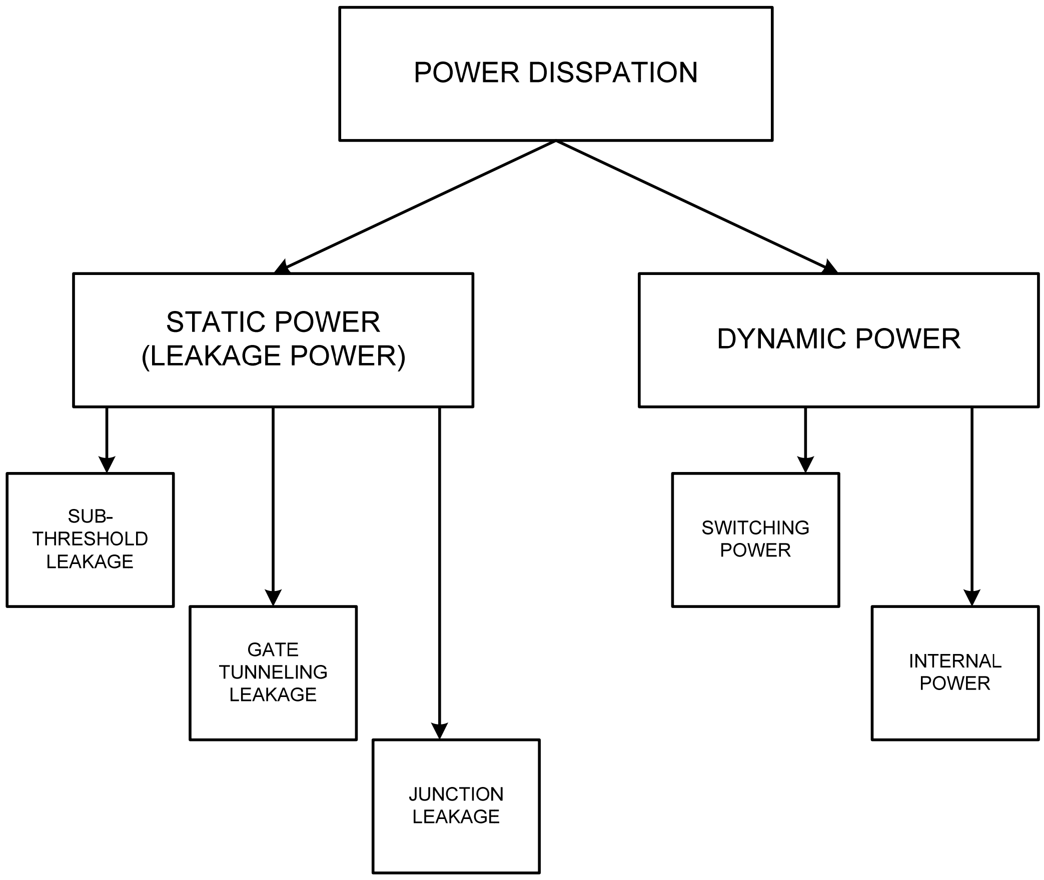


Figure 3-1: Power Dissipation Terminology Tree

3.1.1 Static Power Consumption

Static power is the power dissipated when the gate or circuit is not switching. In other words, the power consumed when the device is static or inactive can be called static power. The primary source of static power comes from the leakage of current from the supply rail to ground through various paths within the circuit. Hence, it has become conventional to refer static power as leakage power.

Sub-threshold leakage has been one of the main sources of leakage in nanometer CMOS technologies. Consider an NMOS transistor as an example. The sub-threshold leakage is due to the current flow from the drain to source of the transistor, even when the gate voltage is set below the threshold voltage of the transistor. This is because of the weak inversion layer that exists in the channel at voltages close to the threshold voltage of the transistor. Sub-threshold leakage has been shown to be the major leakage source in nanometer CMOS technologies. The other sources of leakage include gate tunneling leakage and junction tunneling leakage. Gate tunneling leakage is the current flow into or out of the gate terminal of a MOSFET. Gate tunneling leakage has become an issue with nanometer CMOS technologies because of the reduced gate oxide thickness. Junction leakage is the current flow from the drain or source diffusion region to the body (substrate) terminal of a MOSFET. This leakage current is because of the reverse biased junctions between the source/drain diffusion region and the substrate or the well [6].

3.1.2 Dynamic Power Consumption

Dynamic Power is the power dissipated when the circuit is active and switches. Dynamic power can be visualized to be the sum of two components:

(i) Switching Power

(ii) Internal Power

Switching Power

Switching power is the power dissipated by the charging and discharging of the load capacitance of the circuit. The switching power of an inverter shown in Figure 3-2 driving a load C_L is given by Equation 3-1 where *f* is the frequency of operation in Hz, C_L is the load capacitance in F, and V_{dd} is the supply voltage in V [7].

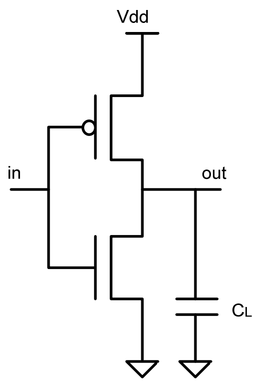


Figure 3-2: Schematic of an Inverter

$$\text{Power (W)} = f * C_L * V_{dd}^2$$

Equation 3-1: Dynamic Power of an Inverter

Internal Power

Internal Power is the power dissipated by the cell because of the charging and discharging of the nodes internal to the cell. In addition to that, short-circuit current that results when both the PMOS and NMOS transistors are ON (as in the case of an inverter) dissipates short-circuit power. This power is also accounted in the internal power consumption. Short-circuit power is a function of the input transition time. For slow transition times, the short circuit power is high as compared with fast transition times because the NMOS and PMOS transistors are simultaneously ON for a longer duration of time. Short circuit power is also influenced by the sizes of the transistors.

3.2 PRIMEPOWER POWER ESTIMATION FLOW

Prime power is gate-level power analysis tool for analyzing the power dissipation of standard-cell based designs [5]. Prime power supports event-based and statistical activity based power analysis at the gate level.

Event based power analysis uses event driven simulation to calculate the power consumed at every event. Event based power analysis is actually dependent on the input vector and when averaged over a large number of input vectors can give a very accurate estimate of the power consumed by the chip.

Statistical Activity based power analysis uses a probabilistic and statistical estimation algorithm to calculate the average power. It is mostly vector free or has weak vector dependence [5].

As event based power analysis was adopted as the design flow for the power estimation of the SuperCISC reconfigurable hardware fabric as it provides accurate time based information [5].

3.2.1 Prime Power's Event Based Power Estimation Flow

Figure 3-3 shows the event-based simulation flow used by Synopsys Prime Power. PrimePower's simulation flow is comprised of two distinct phases [5]:

(i) Switching activity Generation

(ii) Power Profile Generation.

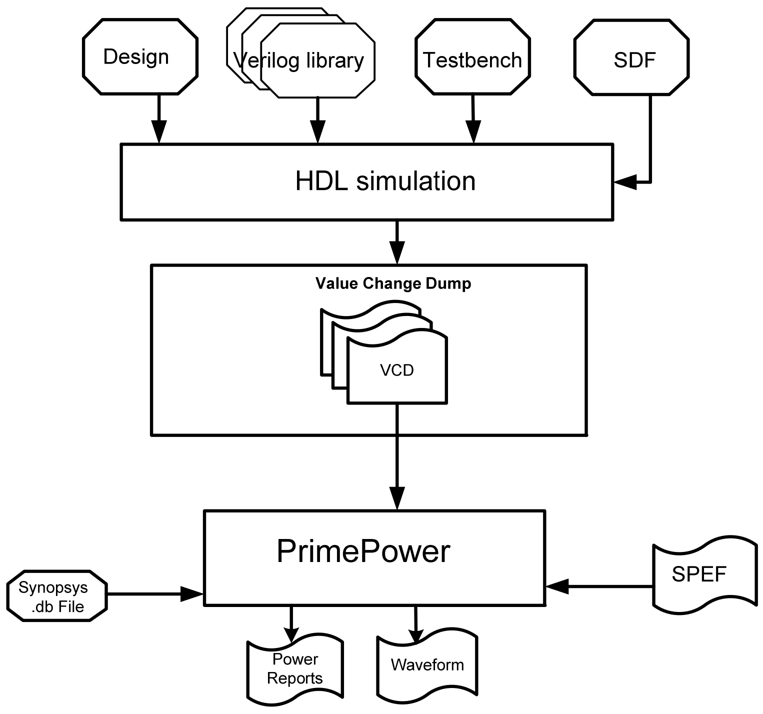


Figure 3-3: Prime Power Event Based Simulation Flow

Switching activity Generation

Switching activity generation is an important step in event based power estimation. Switching activity of the circuit is estimated by running a Modelsim simulation of the design using predetermined input vectors. The toggling of nets in the design is captured by the VCD (Value Change Dump) file. While generating the VCD file, the delays after place and route of the design can be annotated using the SDF (Standard Delay Format) file. The SDF file contains the delays of all the cells and interconnects in the design. The SDF file can be generated by the place and route tool like SoC Encounter using the command ‘delayCal’.

Although, the back-annotation of the SDF file is optional, back-annotation gives an accurate estimate of the delays in the design and is therefore highly recommended. Before the design has been placed and routed, an approximate estimate of the delay can be obtained by using the synthesized netlist along with wire load models. The use of SDF file is much better than using wire load models as the SDF file contains the actual calculated delays from the place and route information [38].

Power Profile Generation

The power profile can be generated by running a prime power script on the design which uses the VCD file. Using the VCD file which contains the information on all toggling nets in the design, dynamic energy consumed in every transition is computed. The static power consumed by every cell in the design is given by the standard cell library description provided by the technology vendor. The total power consumed by the design is the sum of the dynamic and static power consumed. The parasitic information extracted from the place and route tool can be annotated using the SPEF (Standard Parasitic Extraction Format) file generated by the tool.

3.2.2 Calculation of Static Power Dissipated using Prime Power

Prime Power computes the total leakage power of the design as the sum of the leakage power of the design's library cells [4] as shown in equation Equation 3-2.

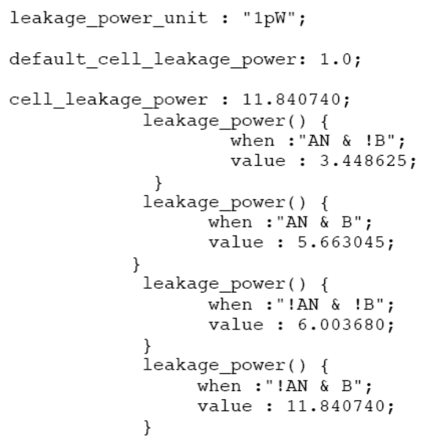
$$P_{\text{total\_leakage}} = \sum_{\forall i} P(i)_{\text{cell\_leakage}}$$

Equation 3-2: Leakage Power Estimation using Prime Power

$P_{\text{total\_leakage}}$ = Total leakage power dissipation of the design

$P_{\text{cell\_leakage}}(i)$ = Leakage power dissipation of each cell "i"

The standard cells in a standard cell library contain the approximate total leakage power dissipated by each cell. Leakage power models can be defined at the library level or at cell level. The models can be state dependent or independent and can be defined to be a single value or can be modeled using a polynomial equation. Figure 3-4 below shows the leakage power definition of a NAND cell. The "leakage_power_unit" attribute is common to the entire library of standard cells, while the "cell_leakage_power" attribute is defined for each standard cell. Figure 3-4 shows a state dependent leakage power definition. The cell leakage power is defined as the maximum value in the state dependent leakage declaration. Depending on the analysis done by the tool, either the "cell_leakage_power" value can be used or the state dependent leakage values can be used. The "default_cell_leakage_power" attribute defined in the standard cell library is a value used by the power analysis tool for standard cells that do not have a leakage power definition.



```
leakage_power_unit : "1pW";

default_cell_leakage_power: 1.0;

cell_leakage_power : 11.840740;
            leakage_power() {
                    when :"AN & !B";
                    value : 3.448625;
             }
            leakage_power() {
                  when :"AN & B";
                  value : 5.663045;
           }
            leakage_power() {
                  when :"!AN & !B";
                  value : 6.003680;
            }
            leakage_power() {
                 when :"!AN & B";
                 value : 11.840740;
            }
```

Figure 3-4: Leakage Power definition of a NAND Standard Cell

3.2.3 Calculation of Dynamic Power Dissipated using Prime Power

Dynamic power dissipated by a CMOS circuit is the sum of switching power and internal power [5]. While performing an event based power analysis, PrimePower uses the switching information available in the VCD file to note the nets that switch. The tool then sets the ramp-up times of the input ports and calculates the propagation of the ramp based on the capacitance annotation. Using the internal power table, the tool then estimates the switching power consumed by the device [5].

Calculation of Switching Power

Prime Power calculates the total switching energy by summing the switching energy associated with every rise transition [5]. Prime Power makes an assumption that switching energy is consumed only when the capacitance gets charged (rise transition) [5]. The total switching power is the total switching energy divided by the simulation time as shown in

Equation 3-3. The switching energy is given by the product of the load capacitance C_{load} and the square of the supply voltage $V_{dd.}$ The capacitance C_{load} includes the parasitic capacitance of the net and the gate and drain capacitances of all the devices connected on the net. The parasitic capacitance can be obtained from the specified wire-load model or from the back-annotation information specified by the SPEF file. The pin capacitance values are obtained from the Synopsys library.

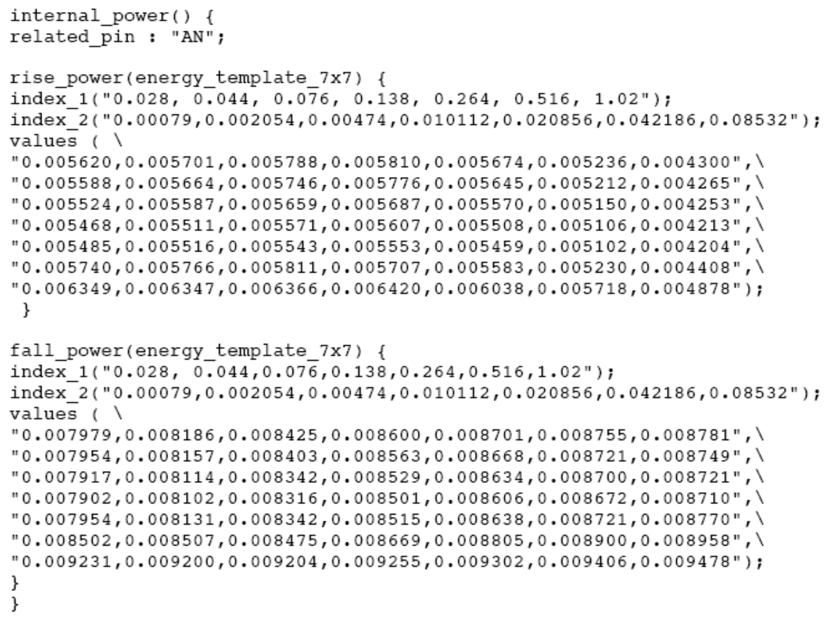
$$\text{Switching Power(W)} = \frac{\sum \text{Switching Energy(J)}}{\text{Total Simulation Time (s)}}$$

Equation 3-3: Prime Power Switching Energy Calculation

Calculation of Internal Power

Internal power can be modeled using NLPM (non-linear power models) or SPPM (Scalable Polynomial Power Model) [5]. NLPM allows designers to store the energy consumption of the cell based on the input transition time and output load. State dependencies and path dependencies can also be included as part of the model if desired. Figure 3-5 shows the NLPM definition of internal power using the Synopsys Liberty Format. The NLPM shown in Figure 3-5 uses a look-up table format. The first index, specified by “index_1” shows the input transition times. The second index, specified by “index_2” shows the various load capacitances.

The entire array contains information on the rise and fall power consumed by the cell for a specific value of the transition time and load capacitance. The total power is the sum of switching power, internal power and the leakage power consumed by the design.



```
internal_power() {
related_pin : "AN";

rise_power(energy_template_7x7) {
index_1("0.028, 0.044, 0.076, 0.138, 0.264, 0.516, 1.02");
index_2("0.00079,0.002054,0.00474,0.010112,0.020856,0.042186,0.08532");
values ( \
"0.005620,0.005701,0.005788,0.005810,0.005674,0.005236,0.004300",\
"0.005588,0.005664,0.005746,0.005776,0.005645,0.005212,0.004265",\
"0.005524,0.005587,0.005659,0.005687,0.005570,0.005150,0.004253",\
"0.005468,0.005511,0.005571,0.005607,0.005508,0.005106,0.004213",\
"0.005485,0.005516,0.005543,0.005553,0.005459,0.005102,0.004204",\
"0.005740,0.005766,0.005811,0.005707,0.005583,0.005230,0.004408",\
"0.006349,0.006347,0.006366,0.006420,0.006038,0.005718,0.004878");
 }

fall_power(energy_template_7x7) {
index_1("0.028, 0.044,0.076,0.138,0.264,0.516,1.02");
index_2("0.00079,0.002054,0.00474,0.010112,0.020856,0.042186,0.08532");
values ( \
"0.007979,0.008186,0.008425,0.008600,0.008701,0.008755,0.008781",\
"0.007954,0.008157,0.008403,0.008563,0.008668,0.008721,0.008749",\
"0.007917,0.008114,0.008342,0.008529,0.008634,0.008700,0.008721",\
"0.007902,0.008102,0.008316,0.008501,0.008606,0.008672,0.008710",\
"0.007954,0.008131,0.008342,0.008515,0.008638,0.008721,0.008770",\
"0.008502,0.008507,0.008475,0.008669,0.008805,0.008900,0.008958",\
"0.009231,0.009200,0.009204,0.009255,0.009302,0.009406,0.009478");
}
}
```

Figure 3-5: NLPM definition of Internal Power

3.3 POWER ESTIMATION USING HSPICE

The delay element (discussed in Chapter 4) is a custom CMOS circuit which can be used as a standard cell in a standard ASIC design flow. The delay element is used in the SuperCISC RHF to minimize the glitching power consumed in the design as shown in Chapter 6. Circuit level simulations were performed on the delay element to characterize its electrical properties. One of the prime metrics for the delay element is its power consumption, both dynamic and static. The sections below describe the power measurement simulation procedures for the delay element.

3.3.1 Calculation of Rise Power

The objective of calculating rise power is to get an average value for the energy consumed when the delay element makes a transition at its input from a logic low to a logic high value. The rise power in the context of the delay element is defined as the power measured from time t=0 (the input starts to rise) to t=delay + Δ. The Δ value was chosen to be 10ns in our case as almost all the transients settled within this time period even when the delay element was being overloaded. Figure 3-6 shows the rise power measurement window.

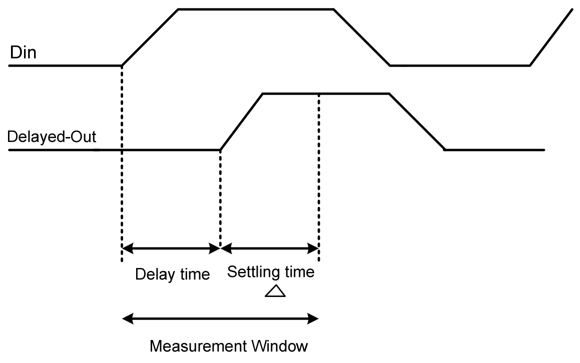


Figure 3-6: Rise Power Measurement window

The rise power is the product of the average current and the supply voltage during the measurement period. The rise energy is obtained by multiplying the rise power by the measurement period. The HSPICE command to calculate the average current during a rise transition is shown in Figure 3-7. The measurement was repeated many times to find the average power consumed.

```
.MEASURE TRAN rise_power AVG I(V0) FROM=1e-05 TO=1.0014e-05
```

Figure 3-7: HSPICE Rise Power Calculation command

3.3.2 Calculation of Fall Power

Fall Power in the context of the delay element is defined as the amount of power consumed when the D input signal makes a transition from a logic high to a logic low value. Fall power is calculated from the point where the Dinput begins to fall at t=0 to t= Δ. As the delay elements does not delay the falling edge, the measurement needs to be done only from t=0 to t=Δ. The HSPICE command for measuring the rise power and fall power are identical except for the time value. Figure 3-8 shows the fall power measurement window.

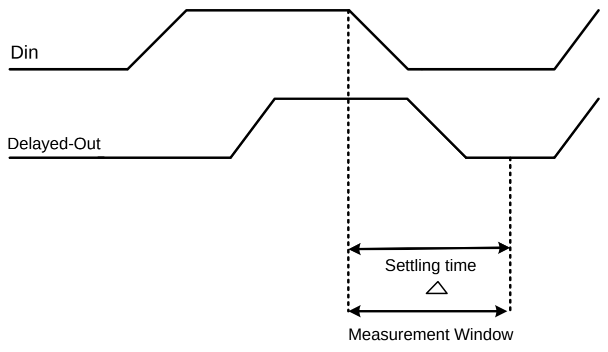


Figure 3-8: Fall Power Measurement window

3.3.3 Calculation of On-State Leakage

The leakage power measured when the Din input has a logic high value is defined as the on-state leakage power. The leakage power is measured after all the transients settle down and the device is in its steady state. On-state leakage is measured from the mid-point of the ON-time for about a 10ns duration. Figure 3-9 below shows the measurement window. The HSPICE command for calculating leakage power is same as the rise power calculation command except for the measurement window.

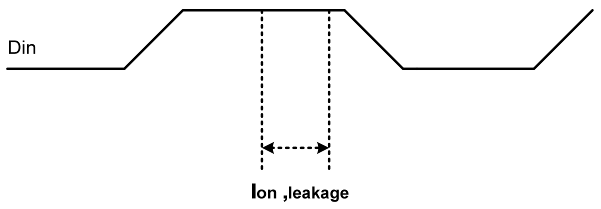


Figure 3-9: On-state leakage Power Measurement window

3.3.4 Calculation of Off-State Leakage

The leakage power measured when the Din input has a logic low value is defined as the off-state leakage power. The leakage power is measured after all the transients settle down and the device is in its steady state. For off-state leakage, the average power is measured from the mid-point of the OFF-time for about a 10ns duration. Figure 3-10 below shows the measurement window for off-state leakage. The HSPICE command similar to the on-state leakage can be used to measure the off-state leakage.

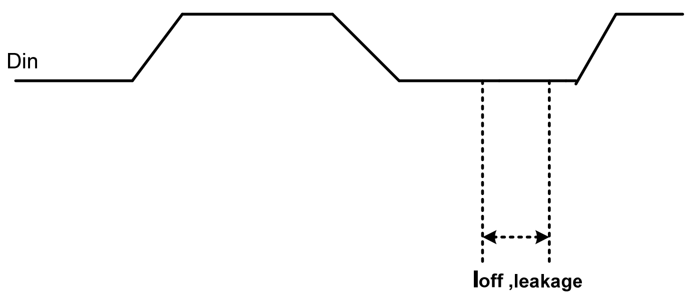


Figure 3-10: Off-state leakage Power Measurement window

4.0 ASIC DESIGN FLOW

A standard ASIC design flow starts right from system specification definition. Once the specifications of the system are clear, specifications are partitioned into logical modules. The logical modules are then implemented using a hardware description language such as VHDL or Verilog. The design is then tested for functional correctness using functional simulations. Once the modules are implemented, the design is synthesized to obtain the gate level netlist [8]. The design flow from specification definition to synthesis is typically called the front end design of the ASIC.

Once the design has been synthesized, the next step in the design process is the physical design including the placement and routing of the design to obtain the GDS file which contains the polygon information of the layout. The GDS file is handed over to the foundary for the chip to be fabricated. Once the chip has been manufactured, it is typically packaged. The packaged chip is tested for its datasheet specifications before being shipped to the customer. The part of the design flow from the synthesized netlist to the GDS file is called the backend design of an ASIC. The test process once the chip is packaged is typically called the post-silicon validation process. Figure 4-1 below shows the typical ASIC design flow right from input specifications all the way to GDSII generation [8].

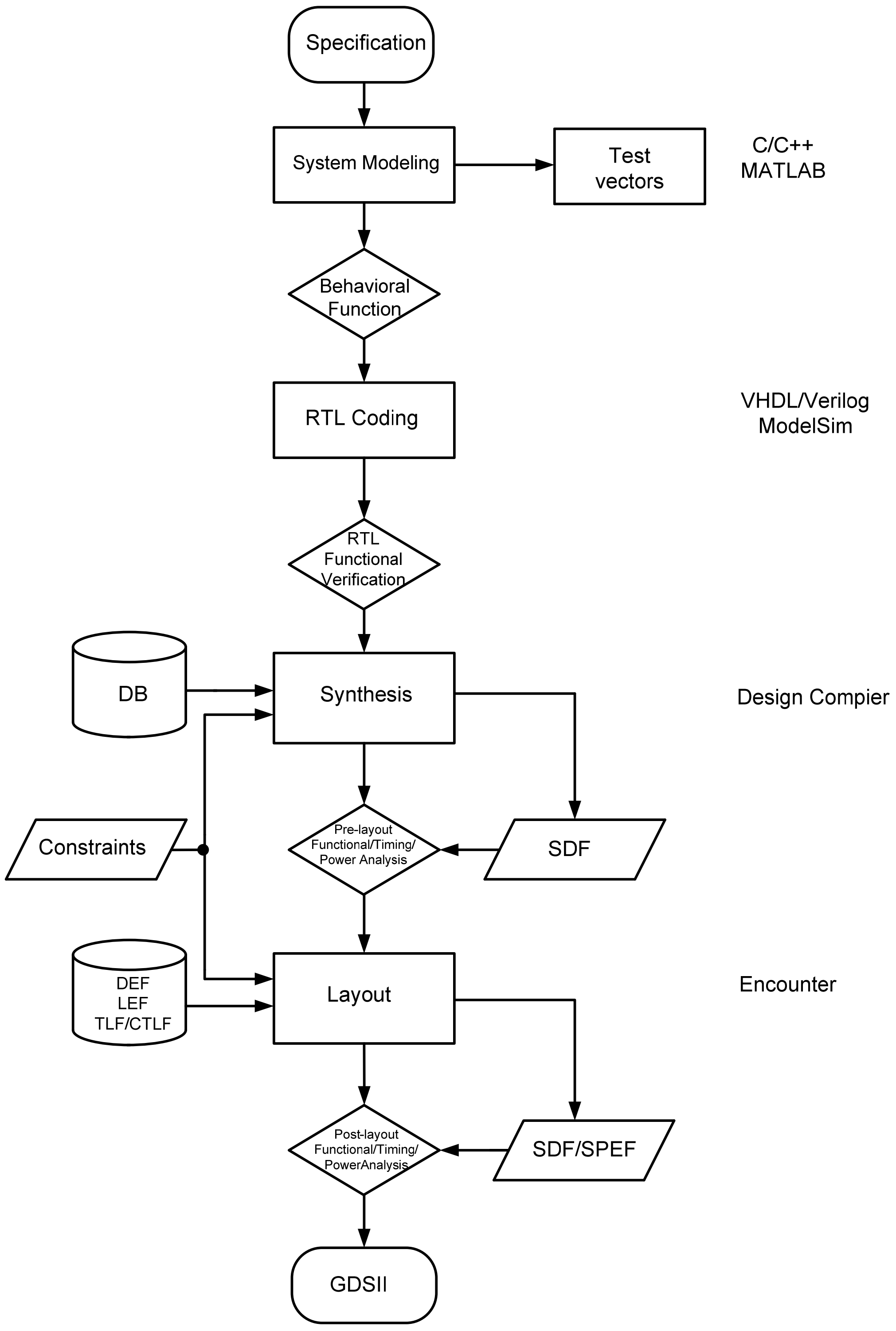


Figure 4-1 : Typical ASIC Design Flow

4.1 PHYSICAL DESIGN FLOW

The backend design of an ASIC called the ASIC Physical design involves placement and routing of the synthesized netlist obtained through logic synthesis. SoC Encounter from Cadence is an industry standard placement and routing tool. The physical design of the SuperCISC reconfigurable hardware fabric was implemented using SoC Encounter. Figure 4-2 shows the typical backend design flow using SoC Encounter.

4.1.1 Floorplanning

A typical ASIC design flow is hierarchical having a top level design and many modules and sub-modules within the design. Hierarchical design is very crucial to the success of any product. It is not only because of the reduced time to market but also because of CAD tools that limit them. A hierarchical design reduces the burden on the CAD tool as compared to a flattened design.

Floorplanning is the process of deciding the position and size of modules in a hierarchical design. A good floorplan is critical to ensure that the chip meets all the timing and power specifications of the design. The floorplan specifications include the core width, core height, and the DIE width and height. The part of the die where standard cells get placed is called the core of the chip. The space between the core and the die is typically used for routing POWER nets and for connecting the nets to the I/O (Input/Output).

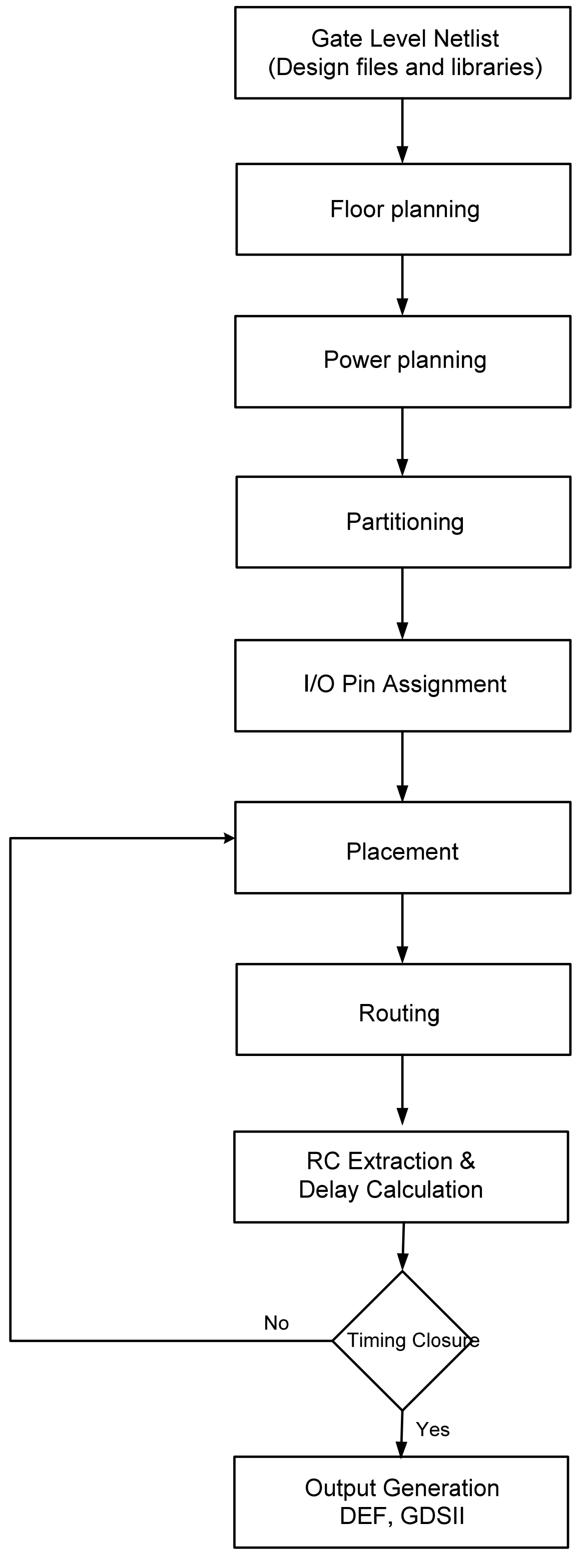


Figure 4-2: ASIC Physical Design Flow

Figure 4-3 below shows the core width/height definition and DIE width/height definitions. The core height has to be a multiple of the standard cell row height so that an integer number of standard cell rows can be placed in the core area. The DIE Width and height are defined using the CORE_TO_IO_LEFT, CORE_TO_IO_RIGHT, CORE_TO_IO_TOP and CORE_TO_IO _BOTTOM specifications. Figure 4-4 below shows the ALU stripe floor plan implemented for the SuperCISC reconfigurable hardware fabric.

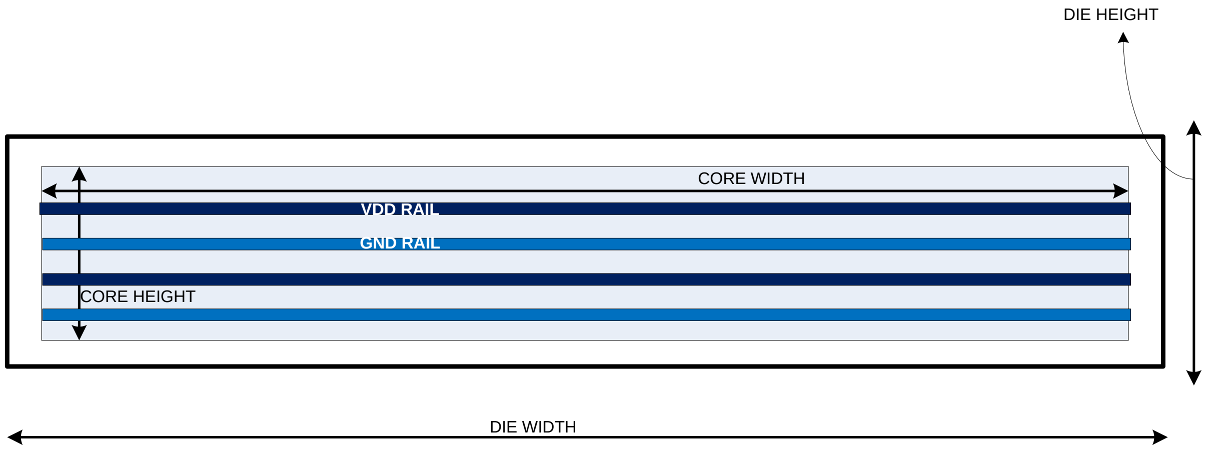


Figure 4-3: Floorplan specifications

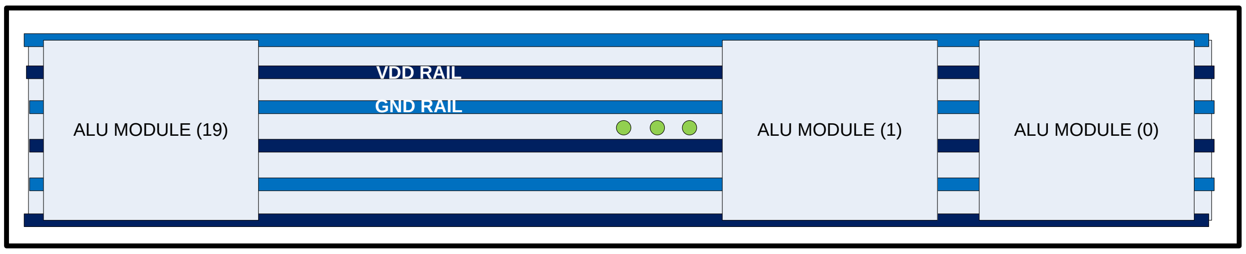


Figure 4-4: ALU Stripe Floor plan

4.1.2 Powerplanning

Power planning is the process of defining the power nets and the power ring around the core which can be used to connect the power supply rails of the standard cell rows. The standard cells have power and ground rails alternating which connect to the power ring on the left and right. Additionally power stripes can be added to reduce the power bus resistance. A thick power ring guarantees reduced power bus resistance and hence lesser IR drop on the power supply bus.

Figure 4-5 below shows the power ring and the power stripes in a design with a core height of five standard cell rows. Figure 4-5 also shows the standard cell power rails connected to the power ring.

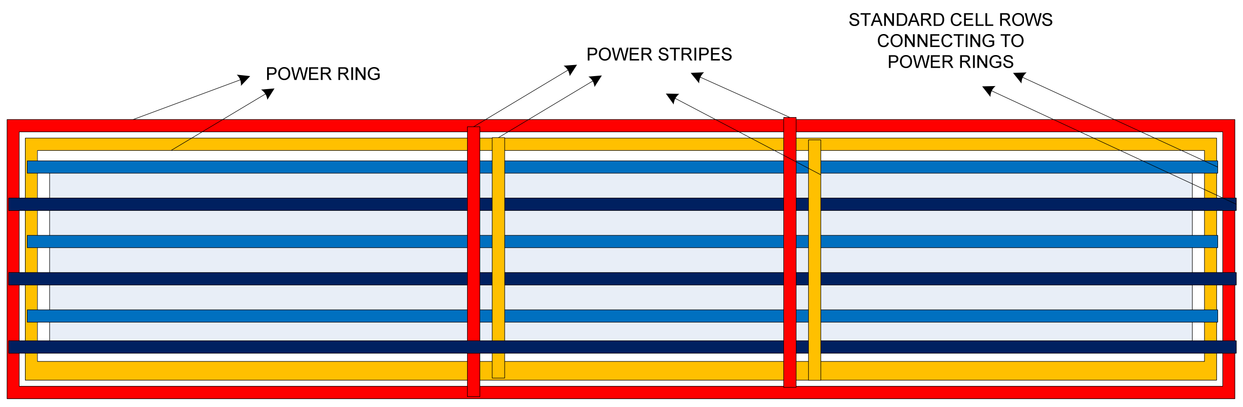


Figure 4-5: Power Planning

4.1.3 Partitioning a design

Partitioning the design is the process of breaking the top level design into modules which can be placed and routed independently. Modules specified as partitions are designs which can be worked upon independently. The place and routed partitions can be finally integrated into the top level design to make the complete chip. Modules that are not specified as partitions are automatically a part of the top level design and get placed and routed at the place and route phase of the top level design.

4.1.4 Routing Feedthrough

After the design has been partitioned, the entire metal stack along with the transistor area is reserved for the partition for its placement and routing. The top level design by default does not have any metal area reserved for routing nets that cross the partitions. This leads to routing congestions in a chip which has multiple partitions and the top level design has nets that run across partitions.

Figure 4-6 shows nets from partition-A getting connected to partition-C. If an inter-partition space is available as shown in Figure 4-6 the tool might route the nets around partition-B. However, if there are too many nets that need to be routed, the inter-partition space might get congested and may result in DRC violations. To avoid such disastrous situations, it is necessary to have specific areas in the metal layers reserved for top level routing. Such reserved metal areas are called routing feedthroughs. Figure 4-7 shows routing when routing feedthroughs are created in the design. Routing feedthroughs can minimize the length of the routes as well as avoid congestion for cases with a minimum inter-partition space.

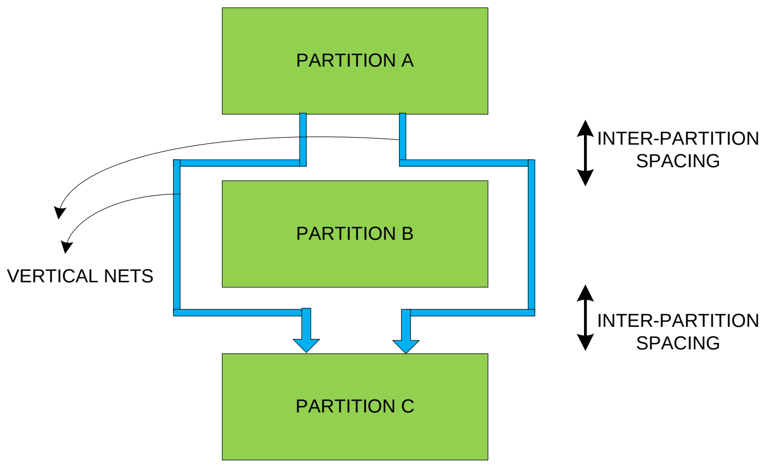


Figure 4-6: Routing in the absence of Routing Feedthroughs

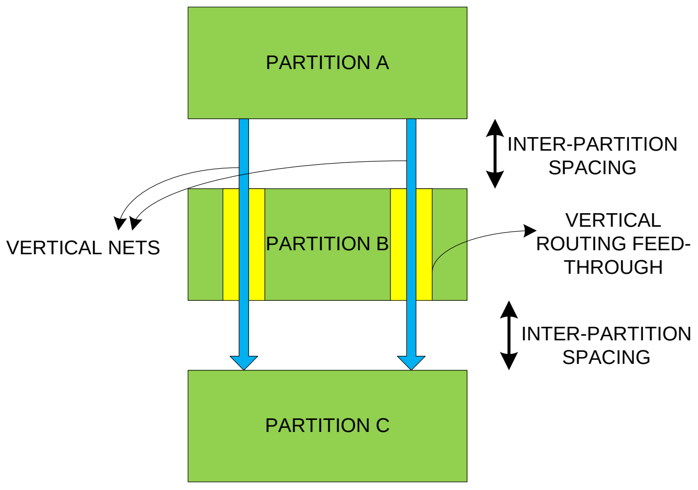


Figure 4-7: Routing in the presence of Routing Feedthroughs

4.1.5 Pin Assignment

Pin assignment defines the interface between the partitioned modules and the top-level design. Hence, it is mandatory that a partition is assigned all the necessary IO pins before being saved. Not only the partitions but also the top level design needs to have all IO pins defined before the partitions are saved.

4.1.6 Placement

During the placement phase, all standard cells in the design are placed within the core area. The placement is an entirely CAD tool dependent process which uses sophisticated CAD algorithms that determine the optimum placement of the standard cells to ensure minimum congestion while being routed. Figure 4-8 below shows an ALU module after being placed in the IBM 0.13um CMOS technology. The vertical columns on the left show the presence of vertical routing feedthroughs that were created on Metal Layer 6.

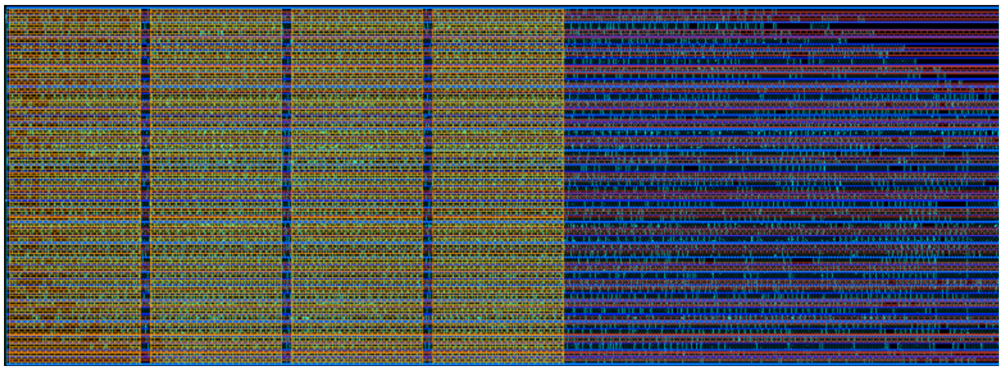


Figure 4-8: Placed ALU Module

4.1.7 Routing

Once the standard cells are placed, the next step in the physical design process is the routing of the design which physically draws polygons on metal layers to connect all the nets as per the netlist. Figure 4-9 below shows the placed and routed ALU designed in OKI 0.16um CMOS technology.

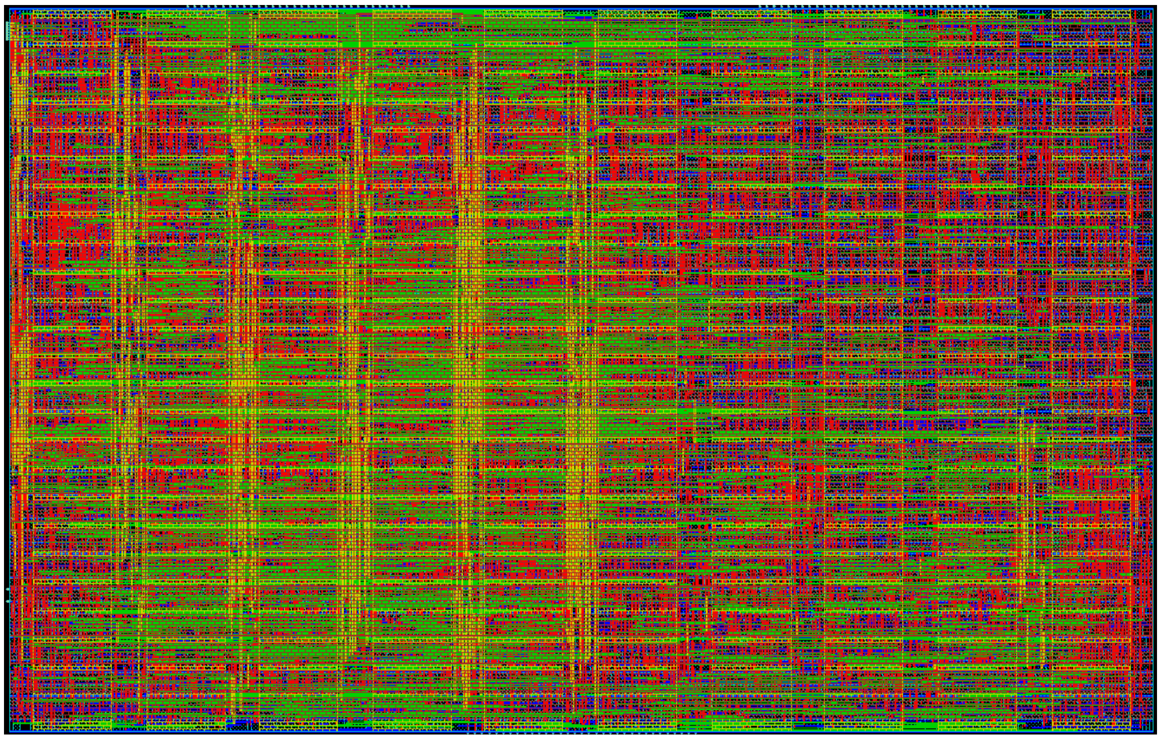


Figure 4-9: Routed ALU Module

5.0 PHYSICAL IMPLEMENTATION OF THE SUPERCISC RHF

The design of the SuperCISC reconfigurable hardware fabric can be divided into a number of modules such as the design of the ALU Stripe, MUX Stripe, FINALMUX Stripe and the design of the BIGFABRIC. The sections below describe the various implementation details of the SuperCISC RHF.

5.1 DESIGN OF THE ALU STRIPE

In a homogeneous ALU configuration, the ALU stripe has multiple identical ALUs in the same stripe. A 20X18 configuration has 20 ALUS in a row whereas a 13X18 configuration has 13 ALUs in a row. The fundamental component of the ALU stripe is the ALU Module.

5.1.1 ALU Module Specifications

The ALU Module specifications were derived from a place and route of the ALU module which showed a high routing density and standard cell area utilization. Table 5-1 shows some important specifications of the ALU Module. Figure 5-1 shows the placed and routed ALU Module implemented in the IBM 0.13um CMOS process.

Table 5-1: ALU Module Specifications

S.No.	ALU Module Specifications	Value
1	Width of ALU Module	500.8 um
2	Height of ALU Module	252 um
3	Die Area	0.126 sq.mm

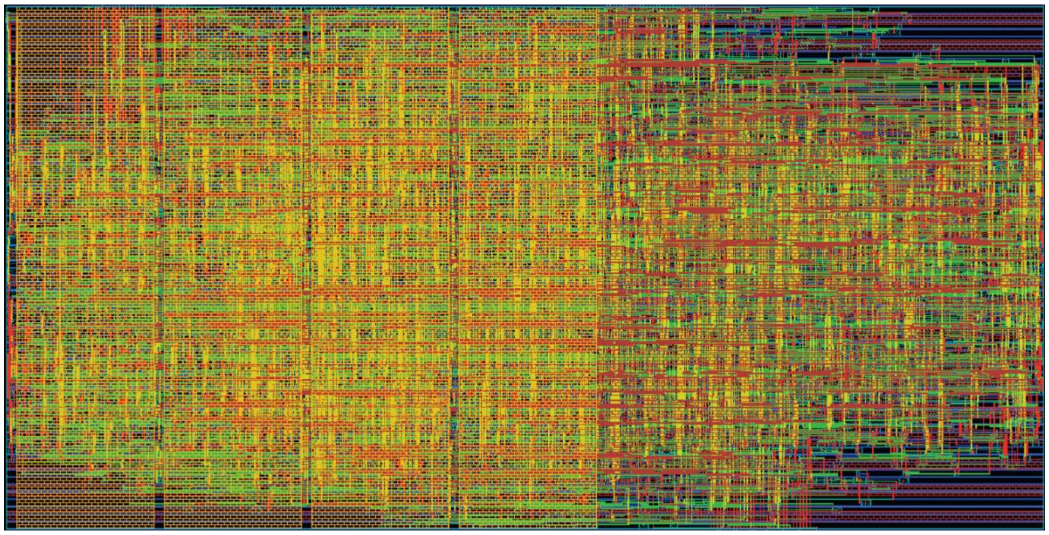


Figure 5-1: Placed and Routed ALU Module

5.1.2 ALU Stripe Specifications

The specifications of the ALU stripe were derived once the ALU module was designed. The stripe specifications are based on the width of the power bus that would be required to minimize drops on the power rail. Figure 5-2 shows a logical ALU stripe and Figure 5-3 shows the floorplan of the ALU stripe.

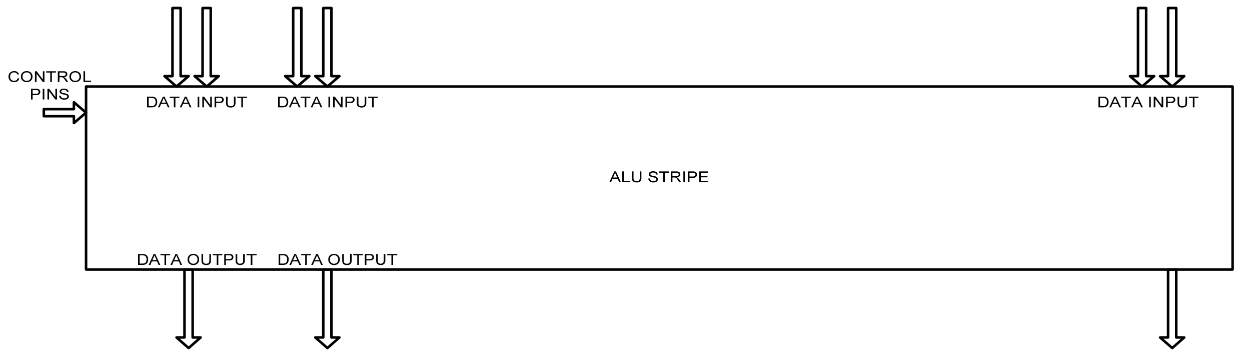


Figure 5-2: Logical ALU Stripe

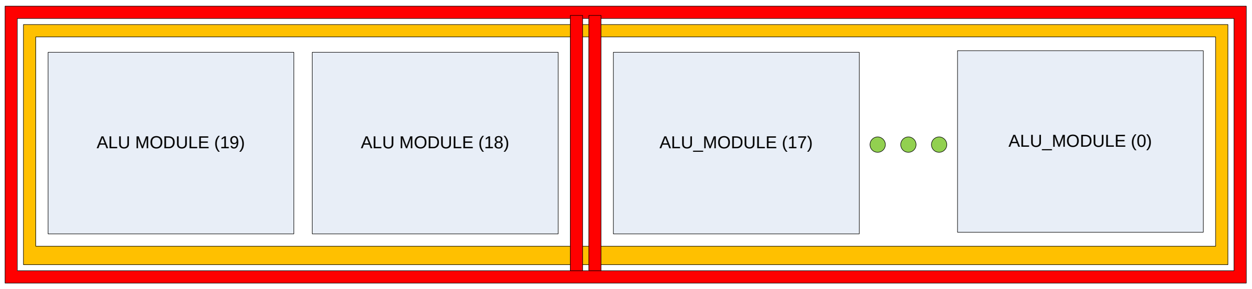


Figure 5-3: Floorplan of ALU Stripe

Table 5-2 shows the important specifications of the ALU stripe for a 20X18 hardware fabric configuration. Figure 5-4 shows the placed and routed ALU Stripe and Figure 5-5 shows a zoomed in view of the same. For the SuperCISC RHF, routing was done using the Nanoroute tool available with SoC Encounter.

Table 5-2: Specification of the ALU Stripe

S.No.	ALU Stripe Specifications (20X18 Hardware Fabric)	Value
1	Die Width of ALU Stripe	10,330.4 um
2	Die Height of ALU Stripe	285.6 um
3	Total Die Area	2.95 sq.mm
4	VDD/GND Power Bus Width	7.2um

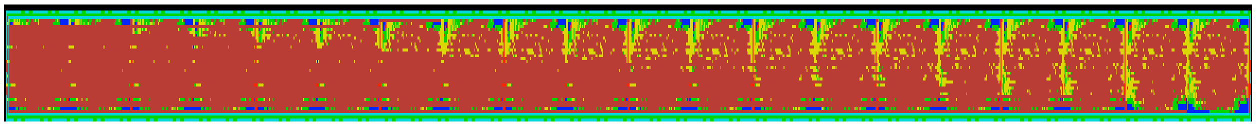


Figure 5-4: Placed and Routed ALU Stripe

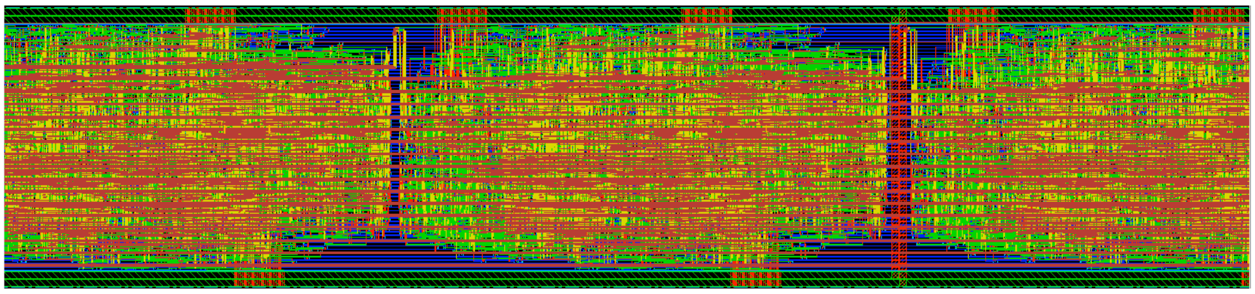


Figure 5-5: Zoomed-in view of Placed and Routed ALU Stripe

5.1.3 ALU Stripe Automation Scripts

As placement and routing is an iterative process, automation is essential to meet design deadlines. Also, the amount of manual work involved in designing the complete chip could be large if automation of the design flow was not possible. The scripts were all written in TCL (Tool Command Language).

The following scripts were written to automate the ALU Stripe design flow:

(i) ALU initialization routine

(ii) ALU Floorplanning routine

(iii) ALU Module Pin placement

(iv) ALU Stripe Pin placement

ALU initialization routine

The ALU Initialization routine initializes the die width and the die height of the ALU stripe. The "ibm_alu_init.tcl" is used for this purpose. It specifies the core area by specifying the distance between the core and the die on all the four sides. The core area is the actual area where standard cells get placed. The core height has to be a multiple of the standard cell height. Once the basic parameters are specified, the script can be use to initialize the ALU stripe using the "floorplan" command available with Encounter [9]. The routine also adds power rings in the space between the core and the IO of the chip using the "addRing" command [9]. The important parameters used in the script are described in Table 5-3.

Table 5-3: ALU Initialization Parameters

S.No.	ALU Initialization Parameters	Description
1	CORE_TO_LEFT CORE_TO_RIGHT CORE_TO_TOP CORE_TO_BOTTOM	Distance between core and IO on left/ right/ top/ bottom sides. Once the die width and height are specified, these values determine the core width and height
2	ALU_LEFT_DISTANCE ALU_RIGHT_DISTANCE	Each ALU has a certain space to its left and to its right to enable routing of wires.
3	INTER_ALU_DISTANCE_MIN	The distance between two ALUs without power stripes between them which is the sum of the ALU_LEFT_DISTANCE and ALU_RIGHT_DISTANCE
4	INTER_ALU_DISTANCE_MAX	The distance between two ALUs those have power stripes between them. The value is the sum of ALU_LEFT_DISTANCE, ALU_RIGHT_DISTANCE and POWER_STRIPE_TOTAL_WIDTH
5	POWER_RING_TOTAL_LEFT POWER_RING_TOTAL_RIGHT	The total width of the power ring on the left side. The width of the power ring includes the wdith of the VDD bus, GND bus, spacing between the VDD and GND bus, spacing between the VDD bus and IO and spacing between GND bus and the core
6	POWER_STRIPE_WIDTH	Width of the power bus
7	POWER_STRIPE_TOTAL_WIDTH	Sum of VDD Bus width, GND Bus width and spacing between them
7	POWER_STRIPE_SPACING	Spacing between the VDD bus and GND bus

In addition to power rings, the power bus resistance can be minimized by the addition of vertical power stripes connecting the power rings. The “addStripe” command can be used to add power stripes [9]. The final part of the initialization routine uses the “sroute” command to route the standard cell power and ground rails [9].

ALU Floor plan/ Power plan routine

The “ibm_setalumod.tcl” is used to floorplan the ALU modules. ALU modules are placed at pre-determined locations using the “setObjFPlanBox” command [9]. The script automatically calculates the X and Y co-ordinates of the ALU module. The parameter CUR_X and CUR_Y describe the X and Y co-ordinate of the lower left corner of the bounding box of the ALU module. The parameters NEXT_X and NEXT_Y describe the X and Y co-ordinate of the top right corner of the bounding box of the ALU module. The parameters CUR_X, CUR_Y, NEXT_X and NEXT_Y are internally generated based on the ALU module’s width and height and need not be given as an input to the user.

ALU Module Pin assignment

Pins can be assigned to a partition using the “preassignPin” command by specifying the X and Y co-ordinates of the pin location along with the layer on which the pin has to be assigned [9]. The “ibm_pinassign_alu_module.tcl” is used to assign pins to the ALU module. The important consideration during pin placement is that each process has a pre-defined horizontal and vertical pin grid given by the technology vendor. Hence, all pins placed in the horizontal direction have to be aligned on the horizontal pin grid. All pins placed on the vertical direction have to be aligned on the vertical pin grid. Also the grid spacing specification for the technology can be different depending on the metal layer being used for the pin placement. The pin placement routine assigns vertical and horizontal pins on a pin valid pin grid with proper spacing for each metal layer. Table 5-4 shows the parameters used in the ALU module pin assignment routine.

Table 5-4: ALU Module Pin Placement Parameters

S.No.	Parameters	Description
1	X_LOC	X Co-ordinate of the pin
2	Y_LOC	Y Co-ordinate of the pin
3	S1_OFFSET	Start position of the INP1 input pins
4	S2_OFFSET	Start position of the INP2 input pins
5	DOUT_OFFSET	Start position of the DOUT output pins. The position is shifted to the right for normal ALUs and is shifted to the left for ALUs that act as early exits. This is done to minimize congestion while routing at the top layer.
6	HOR_PIN_SPACING	Spacing between two pins in the horizontal direction.
7	VERT_PIN_SPACING	Spacing between two pins in the vertical direction.

The 20X18 SuperCISC RHF physical implementation uses five different ALU stripes, all which have the same netlist except for the position of the pins. In a 20X18 implementation of the design, ALU Stripe 6, ALU Stripe 10, ALU Stripe 14 and ALU Stripe 18 have specialized pin positions. All other ALU stripes have a default pin position. The specialized pin assignment is to avoid congestion while routing at the top layer and can be seen in the section that discusses the design of the BIGFABRIC. Figure 5-6 below shows the pin position for a default ALU module.

The default ALU module has its pins aligned to the right because the vertical feedthrough routes are aligned to the left. Figure 5-7 shows the pin alignment for the ALU modules which are part of the “early exits” in the reconfigurable hardware fabric. For a 20X18 configuration, ALU Stripe 6, ALU Stripe 10, ALU Stripe 15 and ALU Stripe 18 are treated as “early exits”.

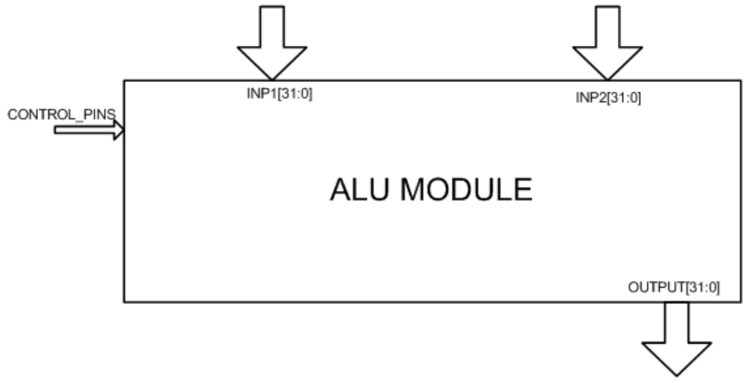


Figure 5-6: Default ALU Module Pin alignment

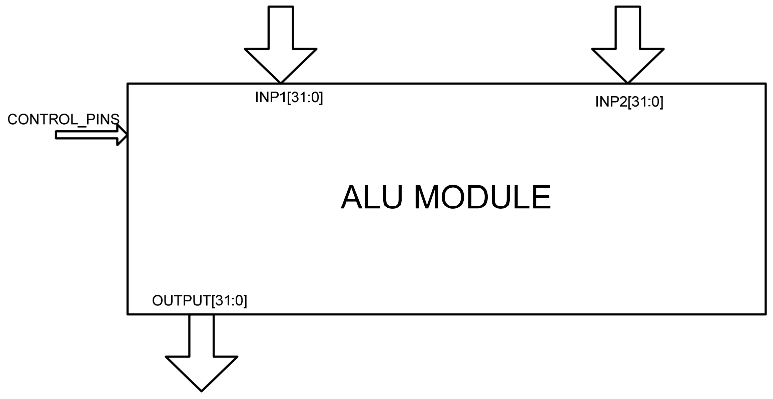


Figure 5-7: Early Exit ALU Module Pin alignment

ALU Stripe Pin Assignment

The “ibm_pinassign_alu_stripe.tcl” is used to assign pins to the ALU stripe. The input and output pins positioned at the top and bottom of the stripe are placed strategically so that they are aligned with the pins of the respective ALU module.

The control pins placed to the left of the stripe need to be routed to the respective ALU modules. To minimize the routing congestion, horizontal routing feedthroughs are inserted. The specifications of the routing feedthrough are shown in Table 5-5. Figure 5-8 shows the normal ALU stripe pin assignment and Figure 5-9 shows the pin assignment for ALUs that act as early exits.

Table 5-5: ALU Horizontal Routing Feedthrough Specification

S.No.	**ALU Horizontal feedthrough**	**Value**
1	FEEDTHROUGH_WIDTH	4.8um
2	NUMBER OF FEETHROUGHS	20
3	FEEDTHROUGH METAL LAYER	METAL 5
4	FEEDTHROUGH SPACING -TOP	3.6um
5	FEEDTHROUGH SPACING - BOTTOM	3.6um

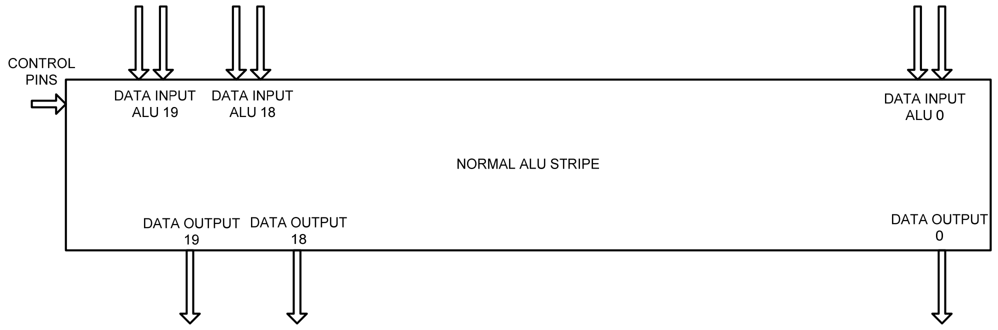


Figure 5-8: Normal ALU Stripe Pin assignment

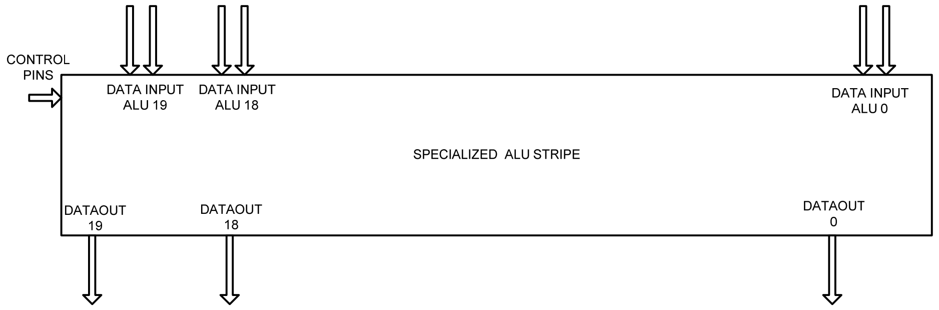


Figure 5-9: Specialized ALU Stripe Pin assignment

5.2 DESIGN OF THE MULTIPLEXER STRIPE

The design of the multiplexer stripe is slightly different from the design of ALU stripe. In addition to the space for the multiplexer modules which form the core of the multiplexer stripe, the stripe requires additional space for routing. As can be seen from the multiplexer architecture in Figure 2-3, the multiplexer stripe is route intensive. So, a dedicated routing space needs to be made available to avoid congestion.

5.2.1 MUX Module Specifications

As the multiplexer forms the interconnect structure, the inputs to the MUX stripe and the MUX modules are from the outputs of the ALU stripe. Figure 5-10 shows the MUX module's logical diagram. The MUX module specifications for the hardware fabric are shown in Table 5-6. Figure 5-11 shows the placed and routed MUX Module implemented in the IBM 0.13um CMOS process.

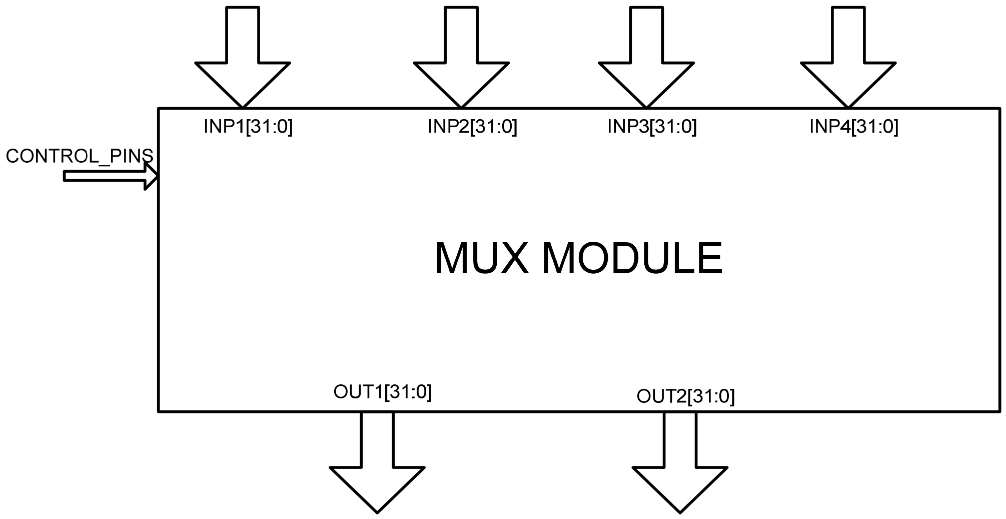


Figure 5-10: MUX Module Logical Diagram

Table 5-6: MUX Module Specification

S.No.	MUX Module Specifications	Value
1	Width of MUX Module	500.8 um
2	Height of MUX Module	36 um
3	Die Area	1802.88 sq.um

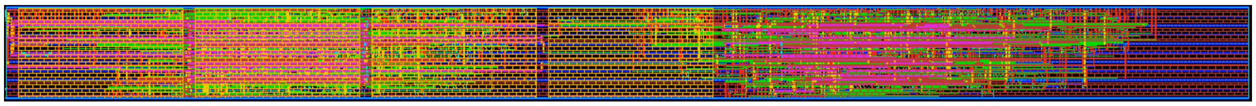


Figure 5-11: Placed and Routed MUX Module

5.2.2 MUX Stripe Specifications

The MUX stripe in addition to the space for the MUX module has additional space for routing. The height of the routing space in the design is 50.4um. Figure 5-12 shows the MUX Stripe logical diagram and Figure 5-13 shows the MUX stripe floorplan. The specifications of the multiplexer stripe are shown in Table 5-7. Figure 5-14 below shows the placed and routed MUX Stripe implemented in the IBM 0.13um CMOS process.

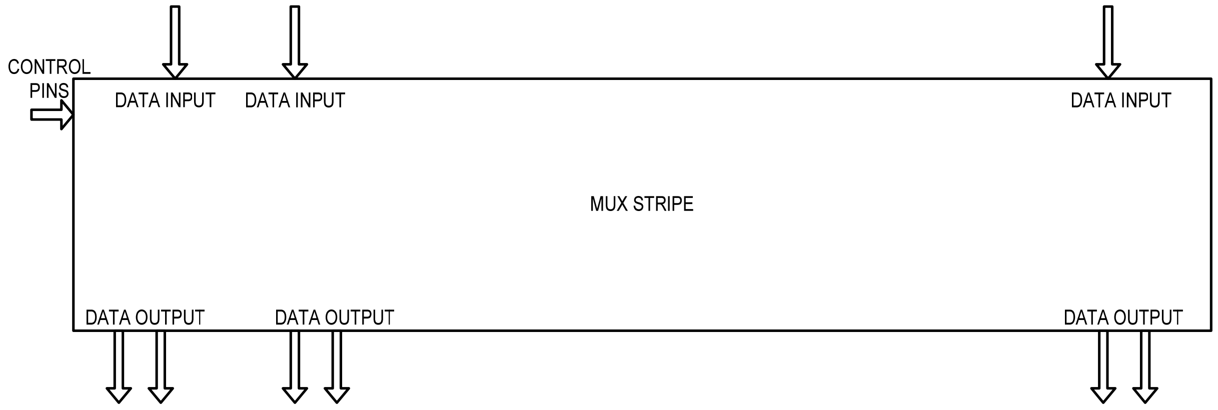


Figure 5-12: MUX Stripe Logical Diagram

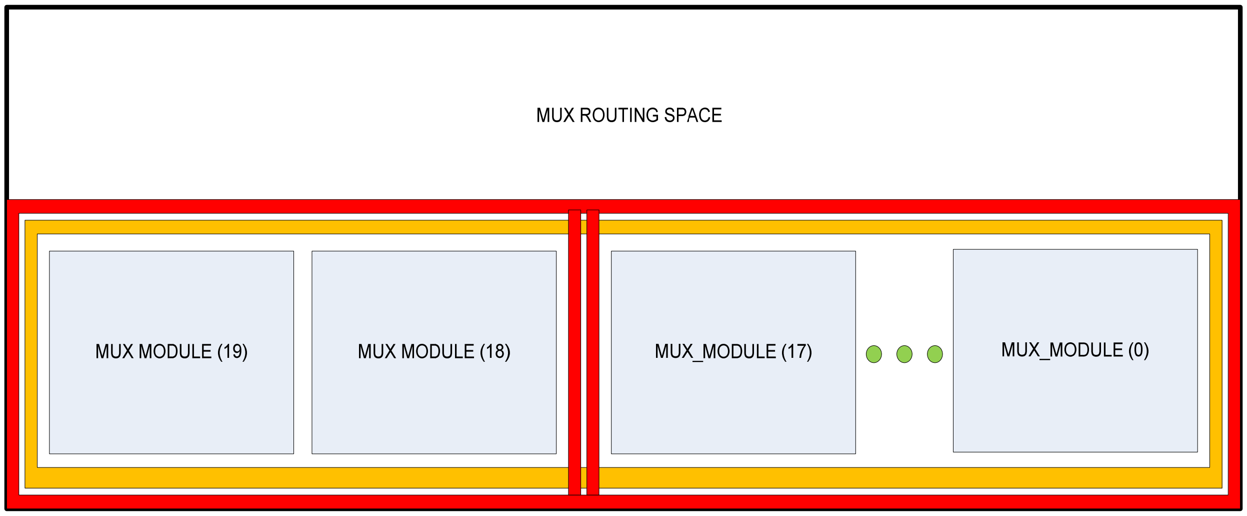


Figure 5-13: MUX Stripe Floorplan

Table 5-7: MUX Stripe Specifications

S.No.	MUX Stripe Specifications (20X18 SuperCISC Hardware Fabric)	Value
1	Die Width of MUX Stripe	10,330.4 um
2	Die Height of MUX Stripe	103.2 um
3	Total Die Area	1.066 sq.mm
4	VDD/GND Power Bus Width	7.2um
5	Routing Space within the Stripe	50.4um

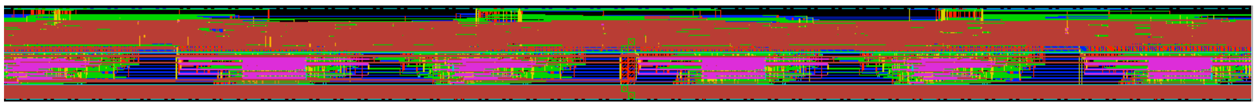


Figure 5-14: Placed and Routed MUX Stripe

5.2.3 MUX Stripe Automation Scripts

The following scripts were written to automate the design of the MUX Stripe:

(i) MUX initialization routine

(ii) MUX Floorplanning routine

(iii) MUX Module Pin placement

(iv) MUX Stripe Pin placement

MUX initialization routine

The MUX initialization routine initializes the die width and the die height of the MUX Stripe. The “ibm_mux_init.tcl” is used to initialize the MUX Stripe. The parameters used in the MUX Initialization routine are similar to the ALU initialization routine and are not repeated in this section again.

MUX Floor plan/ Power plan routine

The “ibm_setmuxmod.tcl” script is used to floorplan the MUX Stripe by placing the MUX modules within the core area.

MUX Module Pin placement

The “ibm_pinassign_mux_module.tcl” is used to place pins inside the MUX module. Table 5-8 describes some implementation specific parameters used in the routine.

Table 5-8: MUX Module Pin Placement Parameters

S.No.	MUX Module Pin Placement Parameters	Description
1	INP1_OFFSET	Start Position of INP1 pins
2	INP2_OFFSET	Start Position of INP2 pins. INP3 pins are placed after INP2 pins.
3	OUT1_OFFSET	Start Position of OUTMUX1 pins
4	OUT2_OFFSET	Start Position of OUTMUX2 pins. OUTMUX3 pin is placed immediately after OUTMUX2 pins.

MUX Stripe Pin Assignment

The MUX Stripe is similar to the ALU Stripe and requires horizontal feedthroughs to route the control pins. Table 5-9 below shows the specifications of the feedthroughs used for the MUX Stripe. The ibm_pinassign_mux_stripe.tcl" is used to assign pins to the MUX Stripe. The parameters used in the script are described in Table 5-10 below.

As the width of the MUX stripe is much less than the ALU stripe width, the number of horizontal feedthroughs is less in case of the MUX stripe. The routing of the control pins is done by selecting the nets using the "selectNet" command and then the routing is performed only on the selected nets.

Table 5-9: MUX Stripe Feedthrough Specifications

S.No.	**MUX Horizontal feedthrough**	**Value**
1	FEEDTHROUGH_WIDTH	4.8um
2	NUMBER OF FEETHROUGHS	5
3	FEED THROUGH TYPE	REPEATED FEEDTHROUGH
4	FEEDTHROUGH LAYER	METAL 5
5	FEEDTHROUGH SPACING -TOP	1.2um
6	FEEDTHROUGH SPACING - BOTTOM	1.2um

Table 5-10: MUX Stripe Pin Placement Parameters

S.No.	MUX Stripe Pin Placement	Description
1	INPUT_OFFSET	Offset for input pins to the MUX Stripe
2	INTEROUT1_OFFSET	Offset for INTEROUT1 output pins of the MUX Stripe.
3	INTEROUT2_OFFSET	Offset for INTEROUT2 output pins of the MUX Stripe. INTEROUT3 pin is placed after the INTEROU2 pins.
4	INTEROUT1_BANK_SPACING	Spacing between the INTEROUT1 pins of two MUX modules
5	INTEROUT2_BANK_SPACING	Spacing between the INTEROUT2 pins of two MUX Modules
6	INPUT_PIN_SPACING	Spacing between two input pins
7	OUTPUT_PIN_SPACING	Spacing between two output pins

Similar to the ALU Stripe, the physical implementation of the MUX Stripe has five specific implementations. The default MUX Stripe implements all the MUX Stripes except the MUX Stripes connected to the ALUs which act as early exits. So, for a 20X18 configurations, MUX Stripe 6, MUX Stripe 10, MUX Stripe 14 and MUX Stripe 18 are specialized stripes which differ from the default MUX Stripes in their pin positions. The netlist of all the five implementations are identical.

5.3 DESIGN OF THE FINALMUX STRIPE

The FINALMUX stripe is different from the MUX Stripe, because the inputs to the final mux stripe are from the "early exits" defined in the reconfigurable hardware fabric. For a 20X18 fabric, with a 4:1 Final MUX Stripe, ALU Stripe 6, Stripe 10, Stripe 14 and Stripe 18 are defined as early exits. The early exits define dedicated routing highways from the specific ALUs to the output through the FINALMUX Stripe.

5.3.1 FINALMUX Module Specifications

The FINALMUX stripe has a FINALMUX Module as its core logic. The logical diagram of the final mux module is shown in Figure 5-15 and the specifications of the FINALMUX Module are given in Table 5-11. Figure 5-16 shows the placed and routed FINALMUX Module implemented in the IBM 0.13um CMOS process.

Table 5-11: FINALMUX Module Specification

S.No.	FINALMUX Module Specifications	Value
1	Width of FINALMUX Module	500.8 um
2	Height of FINALMUX Module	36 um
3	Die Area	1802.88 sq.um

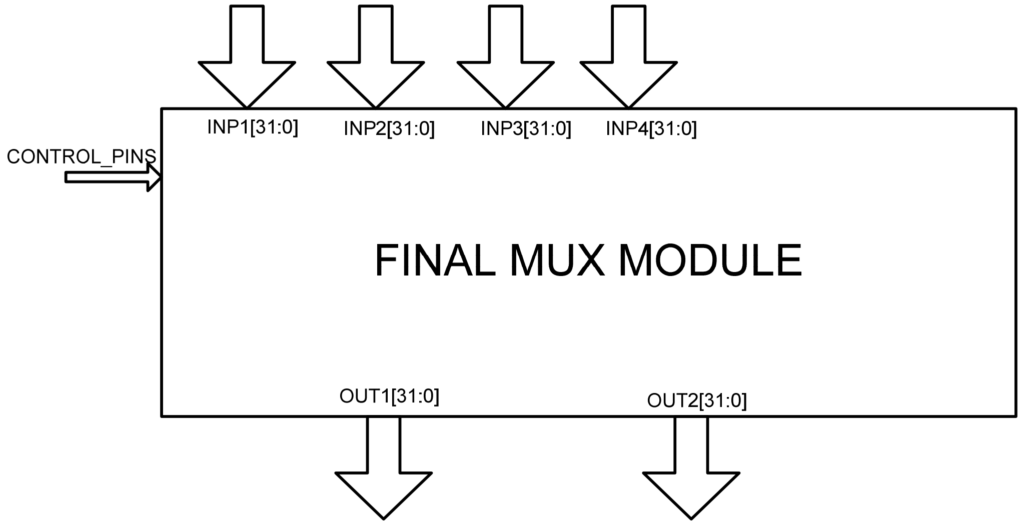


Figure 5-15: FINALMUX Module Logical Diagram

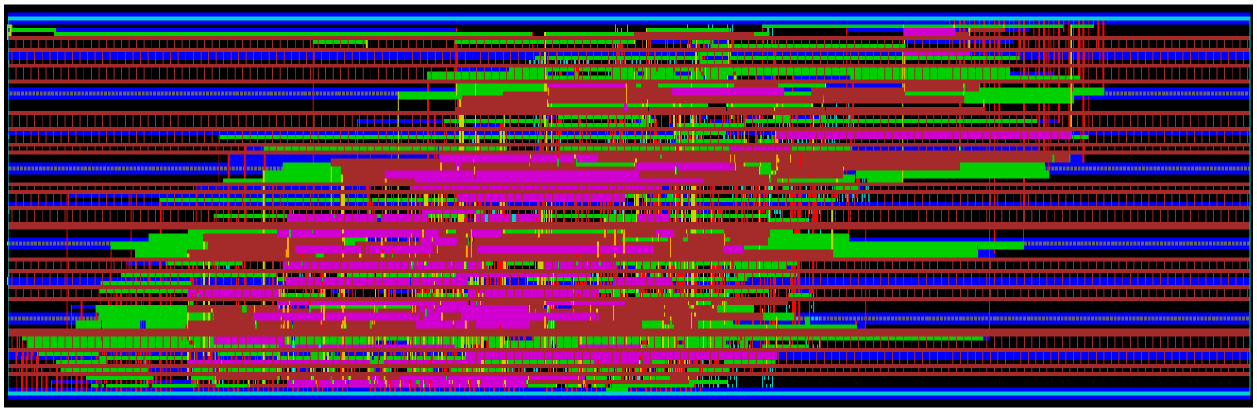


Figure 5-16: Placed and Routed FINALMUX Module

5.3.2 FINALMUX Stripe Specifications

Figure 5-17 shows the FINALMUX stripe logical diagram and Figure 5-18 shows the floorplan of the FINALMUX stripe. Table 5-12 shows the specifications of the FINALMUX stripe. Figure 5-19 shows the placed and routed FINALMUX Stripe implemented in the IBM 0.13um CMOS process.

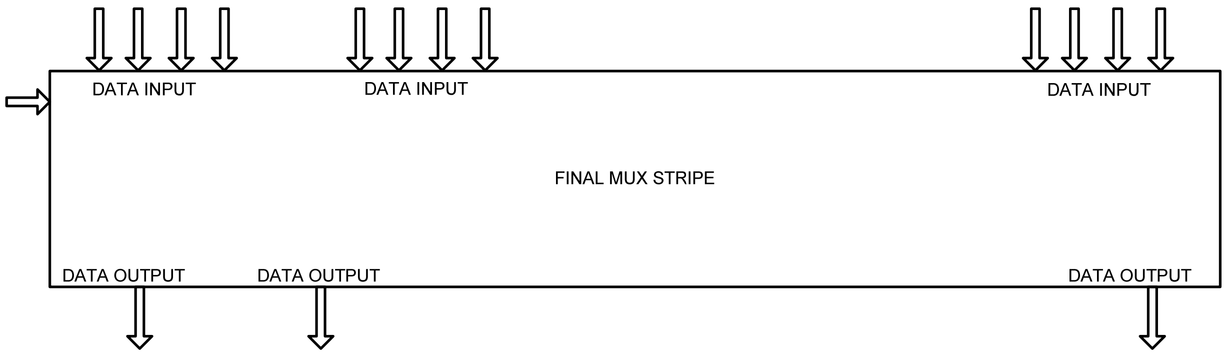


Figure 5-17: Final MUX Stripe Logical Diagram

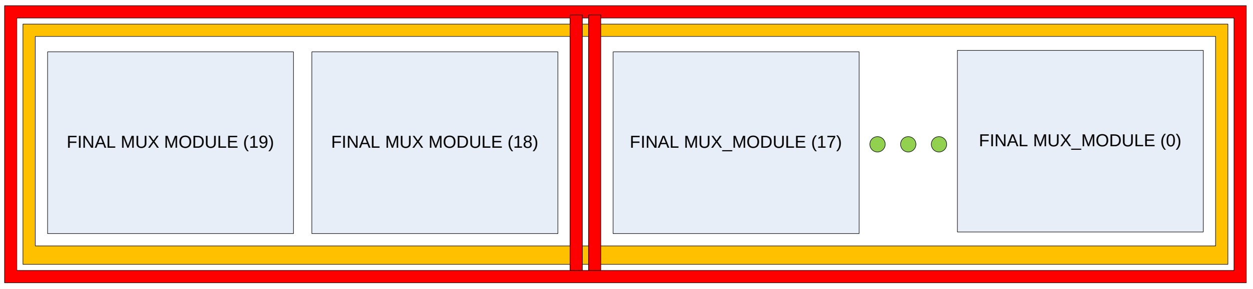


Figure 5-18: Final MUX Stripe Floorplan

Table 5-12: Final MUX Stripe Specifications

S.No.	Final MUX Stripe Specifications (20X18 Hardware Fabric)	Value
1	Die Width of FINALMUX Stripe	10,330.4 um
2	Die Height of FINALMUX Stripe	69.6 um
3	Total Die Area	0.7189 sq.mm
4	VDD/GND Power Bus Width	7.2um

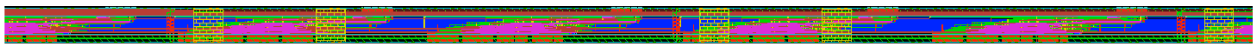


Figure 5-19: Placed and Routed FINALMUX Stripe

5.3.3 FINALMUX Stripe Automation Scripts

The following scripts were written to automate the FINALMUX Stripe design:

(i) FINALMUX initialization routine

(ii) FINALMUX Floor planning routine

(iii) FINALMUX Module Pin placement

(iv)FINALMUX Stripe Pin placement

FINALMUX initialization routine

The FINALMUX initialization routine initializes the die width and the die height of the FINALMUX stripe. The script is similar to the initialization scripts for the ALU Stripe and the MUX Stripe, except for the values specific to the FINALMUX module design. Also, the FINALMUX Stripe does not have additional routing space like the MUX Stripe. This is because the connections from the FINALMUX Stripe to the FINALMUX Module are straight without any routing complexity.

FINALMUX Floorplan routine

The FINAMUX Floorplan routine places the FINALMUX modules inside the core of the FINALMUX stripe. The “ibm_setfinalmuxmod.tcl” script is used to perform the floorplanning.

FINALMUX Module Pin placement

The “ibm_pinassign_finalmux_module.tcl” is used to place pins inside the FINALMUX module. The pins of the FINALMUX module are aligned with the pins of the FINALMUX stripe to avoid routing congestion.

FINALMUX Stripe Pin Assignment

The “ibm_pinassign_finalmux_stripe.tcl” is used to assign pins in the FINALMUX Stripe. For a 20X18 configuration with 4:1 FINALMUX configuration and a spacing of 4, the FINALMUX stripe has 4 inputs. The first input is routed from ALU Stripe 6, second from ALU Stripe 10, third from ALU Stripe 14 and fourth from ALU Stripe 18. The position of the input pins in the FINALMUX stripe is placed in order with the pin positions from the respective ALU stripe. If the pins are not aligned, it would increase the routing complexity tremendously.

The routes to the FINALMUX stripe are routed from the respective ALU Stripes using the dedicated vertical routing feedthroughs. The vertical routing feedthroughs are discussed in the section dealing with the design of the BIGFABRIC.

5.4 DESIGN OF THE BIGFABRIC

The BIGFABRIC is used to describe the top level design of the reconfigurable hardware fabric, instantiating the ALU Stripes, MUX Stripes and the FINALMUX Stripe. The design of the ALU Stripe, MUX Stripe and the FINALMUX Stripe followed a top-down design methodology starting with the Verilog code for each of the stripes down to the place and route of the individual stripes. The design of the “big_fabric” leverages the designed modules and follows a bottom-up flow to integrate the modules that were designed. To accommodate the vertical routing feedthroughs for connecting the early exits to the final mux stripe, dedicated physical feedthroughs were added to the design.

5.4.1 BIGFABRIC Chip Specifications

As the “big_fabric” is built upon the ALU Stripes, MUX Stripes and the final MUX Stripes, these components form the modules for the “big_fabric” design. Table 5-13 below summarizes the “big_fabric” module specifications.

As the BIGFABRIC is the SuperCISC reconfigurable hardware fabric, the specifications of the design are very important. Table 5-14 summarizes the key specifications of the 20X18 SuperCISC reconfigurable hardware fabric.

Table 5-13: BIGFABRIC Module Specifications

S.No.	**BIG FABRIC Module Specifications**	**Value**
1	Height of ALU Stripe	285.6 um
2	Width of ALU Stripe	10,330.4 um
3	Height of MUX Stripe	103.2 um
4	Width of MUX Stripe	10,330.4 um
5	Height of Final MUX Stripe	69.6 um
6	Width of Final MUX Stripe	10,330.4 um

Figure 5-20 shows the “big_fabric” logical diagram and Figure 5-21 shows the top-level routing of the “big_fabric” implemented in the IBM 0.13um CMOS technology. Figure 5-21 also shows the vertical routing feedthroughs being used at the top-level to route the nets from the “early exits” to the FINALMUX stripe. The FINALMUX Stripe is placed at the top to enable an easy access of the IO pins of the fabric when integrated with the processor core/memory in a larger system. Figure 5-22 shows the place and routed BIGFABRIC implemented in OKI 0.16um technology. Figure 5-22 shows only the ALU and MUX Stripes and does not have the FINALMUX Stripe.

Table 5-14: Reconfigurable Hardware Fabric Chip Specifications

S.No.	20X18 Reconfigurable Hardware Fabric	Value
1	Die Width of Chip	10.364 mm
2	Die Height of Chip	7.3164 mm
3	Total Die Area	75.827 sq.mm
4	VDD/GND Power Bus Width	7.2um

5.4.2 BIGFABRIC Automation Scripts

The BIGFABRIC automation scripts are similar to the automation scripts written to automate the design of ALU, MUX, and FINALMUX Stripes. Appendix C contains the TCL (Tool Command Language) scripts that were written to automate the design of the BIGFABRIC.

The following scripts were written to automate the BIGFABRIC design:

(i) BIGFABRIC initialization routine

(ii) BIGFABRIC Floorplanning routine

(iii) BIGFABRIC Module Pin placement

(iv) BIGFABRIC Stripe Pin placement

BIGFABRIC initialization routine

The BIGFABRIC initialization routine initializes the die width and the die height of the SuperCISC reconfigurable hardware fabric chip. The “ibm_bigfabric_init.tcl” is used for this purpose.

BIGFABRIC Floorplan routine

The BIGFABRIC Floorplan routine is used to floor plan the various stripes within the top level design. The “ibm_setbigfabricmod.tcl” is used to automate the floorplanning process.

BIGFABRIC Module Pin placement

As the ALU Stripe, MUX Stripe and FINALMUX Stripes are modules in the “big_fabric”, assignment of pins to the various modules is done using the pin assignment routines of the respective stripes. However, vertical routing feedthroughs (VRF) need to be added to the modules after they are partitioned.

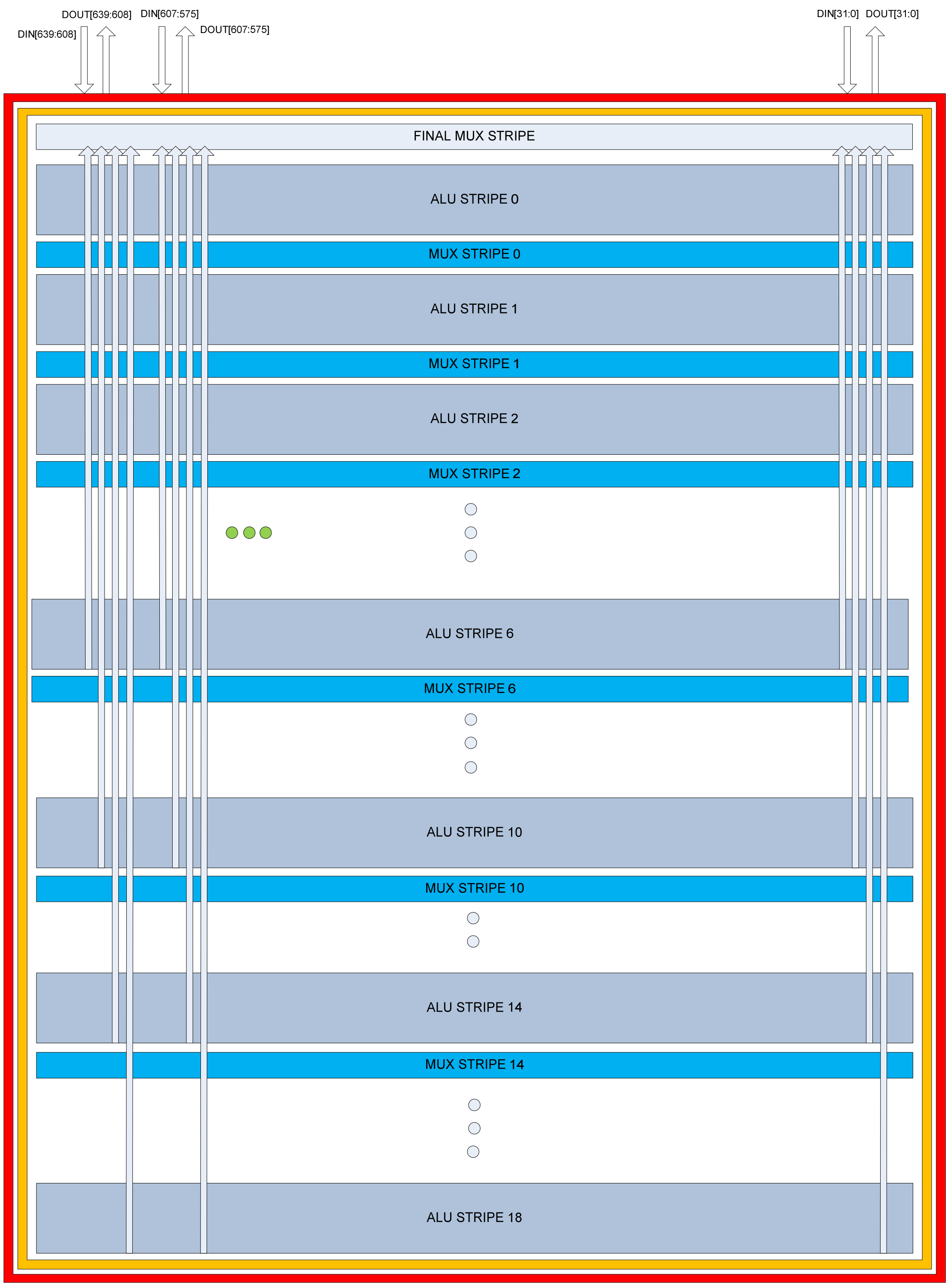


Figure 5-20: BIGFABRIC Logical Diagram

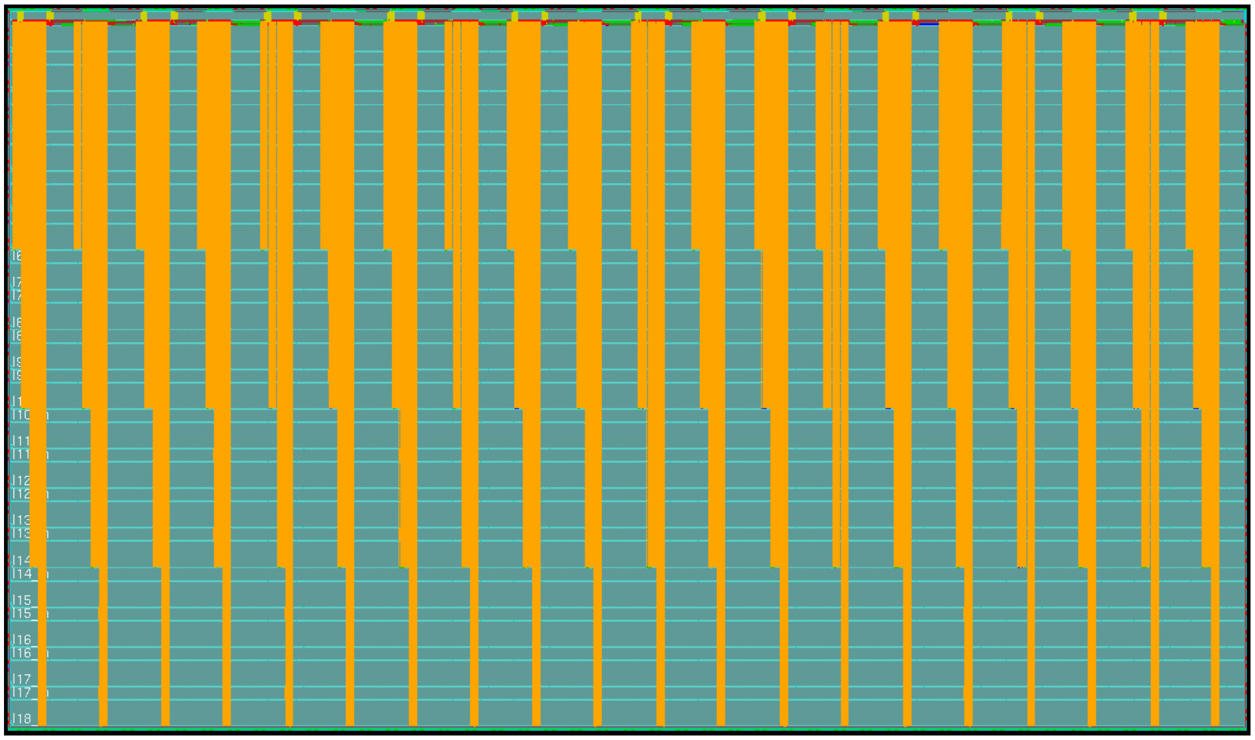


Figure 5-21: Top-level routing of the BIGFABRIC

Four vertical feedthrough start points are specified. Each feedthrough is repeated across the length of the entire chip to provide feedthroughs for every module in the “early exit” stripe. Table 5-15 shows the position of the start points for the four vertical feedthroughs with the spacing for the FINALMUX Stripe. Table 5-16 shows the position of the start points for the vertical feedthroughs for the ALU and MUX stripes in the BIGFABRIC partitions.

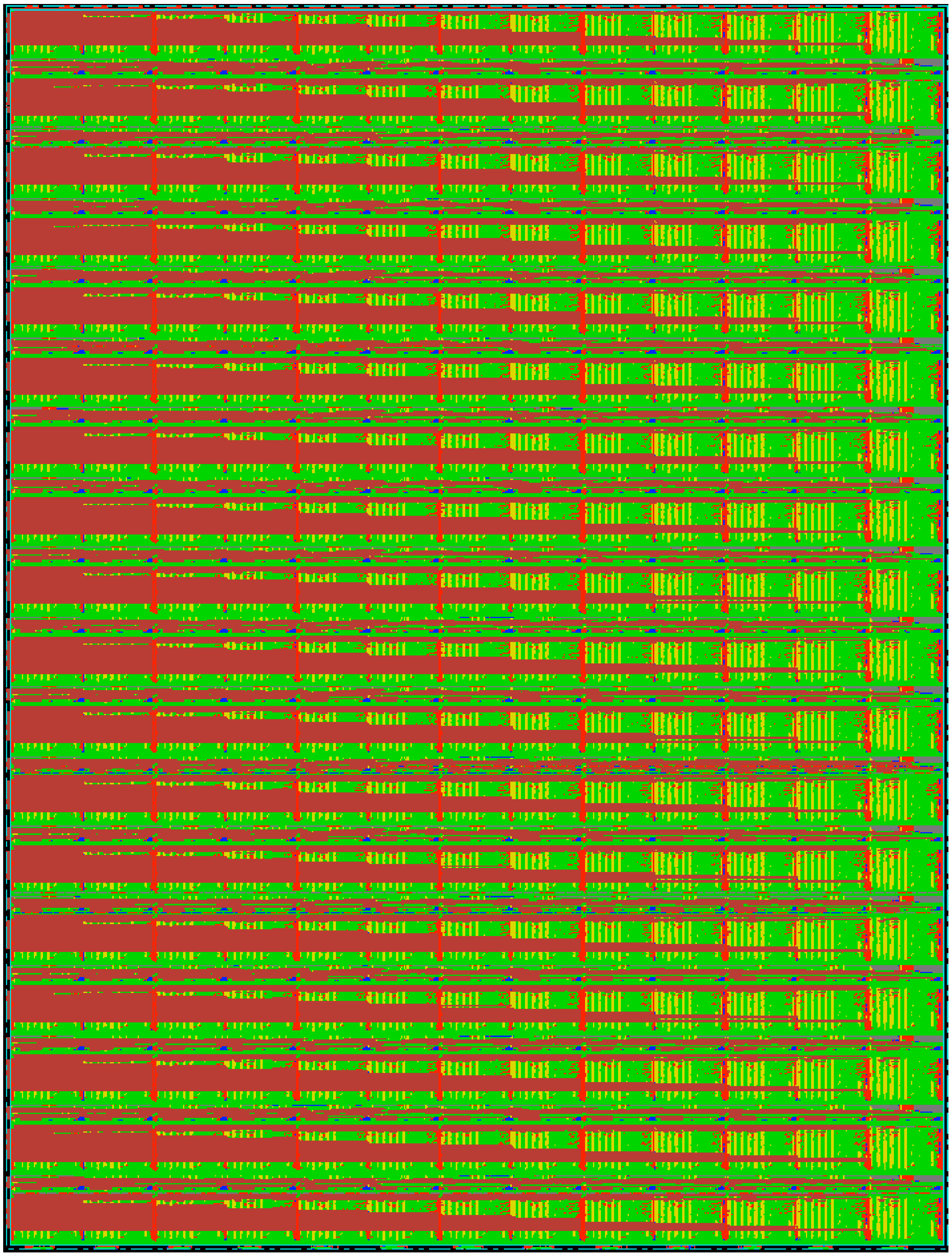


Figure 5-22: Place and Routed BIGFABRIC in OKI 0.16um

Table 5-15: BIGFABRIC VRF for FINALMUX stripe

	I	II	III	IV
Start Point	56.4um	306.4um	564.4um	814.4
Feedthrough width	59.2um	59.2um	59.2um	59.2um
Feedthrough Spacing	972um	972um	972um	972um
Type	Repeated	Repeated	Repeated	Repeated

Table 5-16: BIGFABRIC VRF for ALU and MUX stripe

	I		II		III		IV	
Start Point	22um	93.2	164.4um	235.6um	530.0um	601.2um	672.4um	743.6um
Feedthrough width	66um	66um	66um	66um	66um	66um	66um	66um
Feedthrough Spacing	965.2um	965.2um	965.2um	965.2um	965.2um	965.2um	965.2um	965.2um
Type	Repeated	Repeated	Repeated	Repeated	Repeated	Repeated	Repeated	Repeated

BIGFABRIC Stripe Pin Assignment

As the “big_fabric” is the top level chip which gets integrated into a system, it is necessary that the RHF interface is easier. To make it easier the input and output pins are made accessible from the top of the chip. The placement of the FINALMUX stripe is near the top of the chip, making the output pins easier to access. The “ibm_pinassign_chip_alu_ctrl.tcl” is used to assign the ALU related control pins to the top level chip. The “ibm_pinassign_chip_alu_data.tcl” is used to assign ALU related data pins which act as inputs to the SuperCISC reconfigurable hardware fabric. The “ibm_pinassign_chip_mux_ctrl.tcl” is used to assign MUX related control pins to the fabric. The “ibm_pinassign_chip_finalmux_ctrl.tcl” is used to assign FINALMUX related control pins. The “ibm_pinassign_chip_finalmux_data.tcl” is used to assign the output related pins of the FINALMUX stripe which are actually the outputs of the hardware fabric.

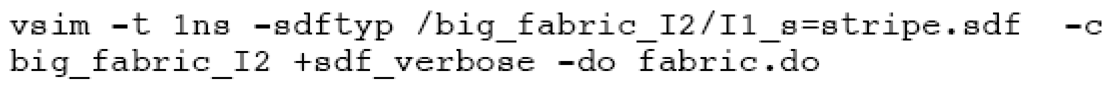
5.5 POWER ANALYSIS OF THE CHIP

Power Analysis was completed using Synopsys Prime Power to estimate the power consumed by the chip after placement and routing. As described in section 4, post place and route information was captured using the SDF (Standard Delay Format) and the SPEF (Standard Parasitic Extraction Format) files. The SDF captures the delay incurred due to interconnects in the design. The SPEF captures the parasitic capacitance and resistance of the nets.

To simulate the chip, the design is broken down into a series of ALU stripes and MUX Stripes. The first ALU stripe is simulated using a set of input vectors. The toggling information in the ALU stripe is captured using the VCD (Value Change Dump) file. The next step is the simulation of the design containing the first ALU stripe and the first MUX stripe. In this simulation, the entire ALU stripe is replaced with the toggling information available in the VCD file generated by the first simulation. This second simulation generates the VCD file containing the toggling information of the MUX stripe. The procedure is continued until all the stripes in the design are simulated. While generating the VCD file, the design needs to be annotated with the SDF file to capture the delays due to the placement and routing of the chip.

The SDF file can be generated using the “delayCal –sdf filename.sdf” command available in SoC Encounter [11]. The tool uses the default delay calculation engine to calculate the delays [12]. The delays are calculated for the interconnect and the cells.

Figure 5-23 shows the Modelsim command used to simulate the ALU stripe with SDF back annotation. The “.do” files contain the necessary Modelsim commands for the VCD file generation along with the input vectors for the simulation.



```
vsim -t 1ns -sdftyp /big_fabric_I2/I1_s=stripe.sdf  -c
big_fabric_I2 +sdf_verbose -do fabric.do
```

Figure 5-23: Modelsim Command for SDF back annotation

The SPEF file can be generated using the “rcOut –spef filename.spef” command available in SoC Encounter [10]. The command essentially reads the parasitic database and outputs the parasitics in the SPEF format. The parasitic database has to be created before issuing the “rcOut” command. The “extractRC –outfile design.cap” extracts the parasitic capacitance for the interconnects and outputs the results to a file. The extraction command uses the CPERSQDIST and EDGECAPACITANCE values specified in the LEF file for capacitance calculation. The parasitic estimates are based on the METAL layer used, the length and width of the routed wires [9] [10]. Similarly, the resistance extraction uses the resistivity coefficients for the layers and via resistance specified in the LEF file.

Prime Power uses the switching information available in the VCD file to generate power characteristics of every stripe. The total power consumed by the chip is the sum of the power consumed by every stripe. Figure 5-24 shows a sample prime power script along with parasitic annotation to read the SPEF file. Figure 5-25 summarizes the power analysis flow that has been used to simulate the post placement and routed power characteristics of the design.

A number of signal processing benchmarks have been mapped to the SuperCISC RHF to analyze its power characteristics. The benchmarks that have been mapped are ADPCM Encoder, ADPCM Decoder, IDCT Row, IDCT Column, Sobel and Laplace.

```
set search_path "/vol/ibm_library"
set link_library "* typical.db"
read_verilog ./stripe.v
current_design stripe
link
read_parasitics ./stripe.spef
read_vcd ./fabric.vcd
calculate_power -statistics -time {0 4256}
report_power -file power_s
quit
```

Figure 5-24: SPEF Annotation in Prime Power

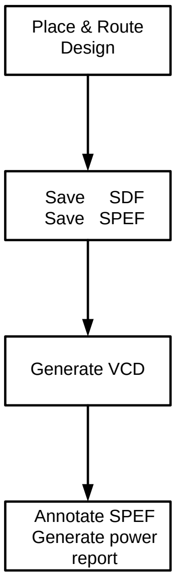


Figure 5-25: Summary of Power Analysis Flow

5.5.1 Power Results for ADPCM Encoder Bench Mark

Table 5-17 summarizes the power consumed by the chip for every stripe in the Super-CISC reconfigurable hardware fabric design when the ADPCM (Adaptive Differential Pulse Code Modulation) Encoder bench mark kernel was mapped to the fabric. The table clearly shows that parasitic annotation of the design has increased the power. The pre-layout power consumption of the chip was 5.948mW and the post-layout power consumption was 7.869mW.

Table 5-17: ADPCM Encoder Post Layout Power Simulation

	ALU Stripe		**MUX Stripe**	
Stripe Number	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**
S1	292	370.8	139.2	220.1
S2	500.2	621.9	145	215.4
S3	444.4	582.6	170.5	251.2
S3	463.8	599.7	176.9	271.1
S5	341.3	425.2	142	215.4
S6	379.3	473.3	130.8	191
S7	240.5	300.4	105	156.9
S8	348.3	430.6	133.6	203.9
S9	219.7	270.2	87.03	126.5
S10	210.6	257.6	64.28	92.01
S11	131.4	166.1	65.43	96.33
S12	157.6	181.6	67.1	102.7
S13	85.59	102.1	48.14	73.34
S14	130.9	158.8	46.66	68.02
S15	70.58	84.15	49.66	76.24
S16	73.39	87.22	36.36	52.99
S17	94.05	121	36.36	52.99
S18	94.05	121	26.7	49.29
Total Power	**4277.6**	**5354.27**	**1670.72**	**2515.41**

5.5.2 Power Results for ADPCM Decoder Bench Mark

Table 5-18 summarizes the power consumed by the chip for every stripe in the SuperCISC reconfigurable hardware fabric design when the ADPCM (Adaptive Differential Pulse Code Modulation) decoder bench mark was run on the fabric. The pre-layout power consumption of the chip was 0.917mW and the post-layout power consumption was 1.168mW.

Table 5-18: ADPCM Decoder Post Layout Power Simulation

	ALU Stripe		**MUX Stripe**	
Stripe Number	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**
S1	81.3	101.9	29.35	41.75
S2	62.69	75.06	25.48	37.41
S3	66.42	80.55	27.78	41.62
S4	51.45	61.23	24.75	34.72
S5	50.17	60.9	23.86	35.46
S6	43.38	52.16	18.9	27.41
S7	41.86	50.89	17.9	26.88
S8	35.37	42.55	17.27	24.29
S9	48.63	57.21	26.61	38.42
S10	31.19	34.65	19.68	28.95
S11	36.02	41.96	12.64	18.61
S12	22.66	25.72	10.23	14.49
S13	21.29	24.03	15.09	21.08
S14	26.89	32.6	10.52	14.75
S15	6.849	6.901	0.1199	0.1199
S16	2.307	2.307	0.1199	0.1199
S17	2.307	2.307	0.1199	0.1199
S18	2.307	2.307	3.536	6.929
Total Power (uW)	**633.09**	**755.232**	**283.9557**	**413.1287**

5.5.3 Power Results for IDCT Row Bench Mark

Table 5-19 summarizes the power consumed by the chip for every stripe in the SuperCISC reconfigurable hardware fabric design when the IDCT (Inverse Discrete Cosine Transform) Row bench mark was run on the fabric. The pre-layout power consumption of the chip was 10.82mW and the post-layout power consumption was 16.204mW.

Table 5-19: IDCT Row Post Layout Power Simulation

	ALU Stripe		MUX Stripe	
Stripe Number	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**
S1	1023	1371	184.8	281
S2	987.3	1833	192.9	297.8
S3	527	631.4	205	305.7
S4	660	770	256.7	390.7
S5	699.7	830.9	213.3	326.2
S6	1425	3085	232.5	356.1
S7	647.1	822.3	256.3	393
S8	695.9	910.6	268	412.9
S9	764.6	934.4	254.1	373.8
S10	794.4	1053	292.1	406.5
S11	141.7	143.3	0.1199	0.1199
S12	2.307	2.307	0.1199	0.1199
S13	2.307	2.307	0.1199	0.1199
S14	2.307	2.307	0.1199	0.1199
S15	2.307	2.307	0.1199	0.1199
S16	2.307	2.307	0.1199	0.1199
S17	2.307	2.307	0.1199	0.1199
S18	2.307	2.307	82.31	259.3
Total Power (uW)	**8381.849**	**12401.05**	**2438.849**	**3803.839**

5.5.4 Power Results for IDCT Column Bench Mark

Table 5-20 summarizes the power consumed by the chip for every stripe in the SuperCISC reconfigurable hardware fabric design when the IDCT (Inverse Discrete Cosine Transform) Column bench mark was run on the fabric. The pre-layout power consumption of the chip was 13.995mW and the post-layout power consumption was 19.979mW.

Table 5-20: IDCT Col Post Layout Power Simulation

	ALU Stripe		MUX Stripe	
Stripe Number	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**	**Pre-Layout Power (uW)**	**Post-Layout Power(uW)**
S1	1109	1607	168.5	247.2
S2	938.1	1511	164.1	251.1
S3	504.3	638.9	193.1	294.2
S4	505.6	595.7	163.1	234.7
S5	455.2	599.1	165.5	248.2
S6	495.4	629.9	179.7	272
S7	550	657.6	214.3	314.5
S8	617	734.1	188.6	288.9
S9	1537	3000	202.5	309
S10	605.2	764.2	253.2	378.2
S11	682.3	892.5	290.5	426.6
S12	820.7	993.7	329.7	492.1
S13	911.6	1197	366.9	532.7
S14	808.6	1041	321.1	451.7
S15	153.8	155.5	0.1199	0.1199
S16	2.307	2.307	0.1199	0.1199
S17	2.307	2.307	0.1199	0.1199
S18	2.307	2.307	93.26	213.5
Total Power (uW)	**10700.72**	**15024.12**	**3294.42**	**4954.96**

5.5.5 Power Results for Sobel Bench Mark

Table 5-21 summarizes the power consumed by the chip for every stripe in the SuperCISC reconfigurable hardware fabric design when the Sobel benchmark was run on the fabric. The pre-layout power consumption of the chip was 1.512mW and the post-layout power consumption was 2.099mW.

Table 5-21: Sobel Post Layout Power Simulation

	ALU Stripe		**MUX Stripe**	
Stripe Number	**Pre-Layout Power (uW)**	**Post-Layout Power (uW)**	**Pre-Layout Power (uW)**	**Post-Layout Power (uW)**
S1	313.9	571.5	71.69	107.3
S2	126.6	142.4	30.09	44.22
S3	93.7	110.5	55.86	85.19
S4	224.9	270.8	86.44	123.4
S5	152.5	190	57.77	82.56
S6	107	122.6	38.32	58.34
S7	36.28	40.19	11.16	17.29
S8	32.64	38.7	8.296	13.45
S9	15.46	17.96	5.804	8.818
S10	14.64	18.06	4.077	6.096
S11	4.632	4.666	0.1199	0.1199
S12	2.307	2.307	0.1199	0.1199
S13	2.307	2.307	0.1199	0.1199
S14	2.307	2.307	0.1199	0.1199
S15	2.307	2.307	0.1199	0.1199
S16	2.307	2.307	0.1199	0.1199
S17	2.307	2.307	0.1199	0.1199
S18	2.307	2.307	3.465	8.921
Total Power (uW)	**1138.401**	**1543.525**	**373.8113**	**556.4243**

5.5.6 Power Results for Laplace Bench Mark

Table 5-22 summarizes the power consumed by the chip for every stripe in the SuperCISC reconfigurable hardware fabric design when the Laplace benchmark was run on the fabric. The pre-layout power consumption of the chip was 1.296mW and the post-layout power consumption was 1.623mW.

Table 5-22: Laplace Post Layout Power Simulation

	ALU Stripe		**MUX Stripe**	
Stripe Number	**Pre-Layout Power (uW)**	**Post-Layout Power (uW)**	**Pre-Layout Power (uW)**	**Post-Layout Power (uW)**
S1	276.4	323.6	80.77	120.3
S2	169.7	196	59.17	89.53
S3	116	134.3	42.71	67.25
S4	88.91	100.3	29.75	45.98
S5	76.71	88.63	57.37	85.86
S6	119.9	139.9	38.09	55.56
S7	35.62	39.22	9.6	14.77
S8	16.22	18.88	7.233	10.88
S9	17.44	21.67	7.233	10.88
S10	17.44	21.67	5.136	7.551
S11	5.135	5.175	0.1199	0.1199
S12	2.307	2.307	0.1199	0.1199
S13	2.307	2.307	0.1199	0.1199
S14	2.307	2.307	0.1199	0.1199
S15	2.307	2.307	0.1199	0.1199
S16	2.307	2.307	0.1199	0.1199
S17	2.307	2.307	0.1199	0.1199
S18	2.307	2.307	3.191	8.585
Total Power (uW)	**955.624**	**1105.494**	**341.0923**	**517.9853**

6.0 DELAY ELEMENTS FOR LOW POWER FABRIC

Delay elements are circuits that introduce a specific delay between the input and the output. Delay elements are widely used in asynchronous or self-timed digital systems. For a self-timed system, the delay element generates a task-complete signal to flag the completion of the task [11][12].Delay elements are also used in circuits that perform mathematical computations like the Discrete Cosine Transform [11] [13]. Delay elements are also used in Phase Locked Loops(PLLs) and Delay Locked Loops (DLL) [11] [14].

Circuits can be designed to provide the specific delay as required. The challenge in designing delay element circuits is to have circuits that show good signal integrity and low power consumption. Also the circuits should show very little variation across supply voltage, temperature and process. An exhaustive study on delay elements has been conducted in the past to analyze the pros and cons of various delay elements. Some of the conventional delay elements are the Transmission gate, Cascaded inverter chain, Voltage-Controlled delay elements, and transmission gate with Schmitt trigger [15][16].

6.1 DELAY ELEMENT TOPOLOGIES

A wide variety of delay element topologies have been described in [15] [16]. Each of the delay element topology has a unique delay, area, power and signal integrity characteristic.

Transmission Gate Based Delay Element

A transmission gate (TG) based delay element has a NMOS and a PMOS connected in parallel. The gates of the PMOS and NMOS are connected to GND and VDD respectively. The delay in a TG based delay element is from the on-resistance of the parallel combination of the PMOS and NMOS through which a load capacitance C_L is charged and discharged. The on-resistance of the delay element can be varied by adjusting the transistor sizes appropriately. The reason for using both PMOS and NMOS in the structure is that a PMOS passes a logic high value without degradation and a NMOS passes a logic low value without degradation.

The power consumed by the transmission gate delay element is from the charging and discharging of the load capacitance. Hence the power consumed by the delay element is minimal. However, because of the slow rise and fall times, the signal integrity of the delay element is considerably degraded. In this context, signal integrity is said to be good when the rise and fall times measured from 10% to 90% of the output is considerably small [15]. The delay element cannot be used to generate large delay values because of the slow rise times. The slow rise and fall times will cause a huge short circuit current consumption on the logic gate that is being driven. The TG based delay element has shown to vary considerably with supply voltage variations [11]. The TG can be used for delays in the range from 200-300ps [11]. Figure 6-1 shows the schematic of a transmission gate based delay element.

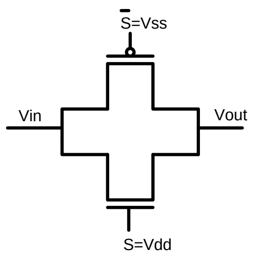


Figure 6-1: Transmission gate based delay element

Transmission gate with Schmitt Trigger

The signal integrity of the TG based delay element can be improved by placing a Schmitt trigger in series with the TG. The Schmitt trigger circuit produces a fast rising edge signal from a noisy and slow varying signal [15]. Apart from making the output signal clean, the Schmitt Trigger minimizes the short-circuit current consumption due to a slowly rising signal. However, the short-circuit current from the input transistor cannot be avoided. Hence, the use of TG always costs significant amount of power. The delay of a TG with Schmitt trigger circuit can be changed by varying the sizes of the TG or by changing the switching threshold of the Schmitt Trigger [15].

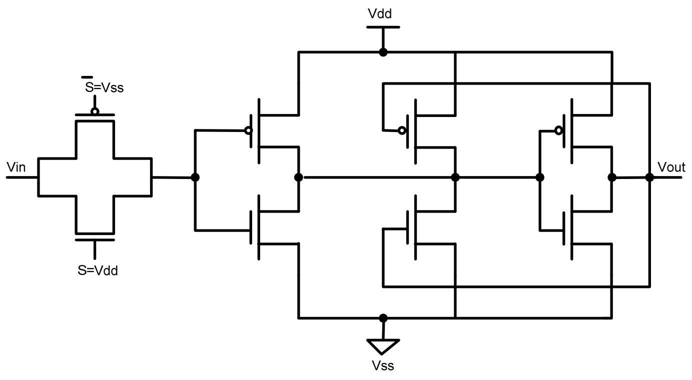


Figure 6-2: Transmission Gate with Schmitt Trigger

Cascaded Inverter Based Delay Element

A series of inverters can be used as a delay element. Such a delay element uses the propagation delay of the inverters in the chain to obtain the specified delay value. The propagation delay of an inverter is dependent upon the time to charge and discharge the load capacitance. The cascaded inverter chain is the simplest delay element topology and is the most common delay element because of the ease of design. However, such delay elements are sensitive to environmental variations such as supply voltage and temperature.

The inverter chain shows considerably less supply voltage variation compared to a TG based delay element [11]. However, the number of inverters in series increases tremendously for large delay values. With the increase in the number of inverters, the power consumption also increases. The inverter chain can be used for delays in the range between 1ns to 3ns [16]. Figure 6-3 below shows the schematic of the inverter based delay chain.

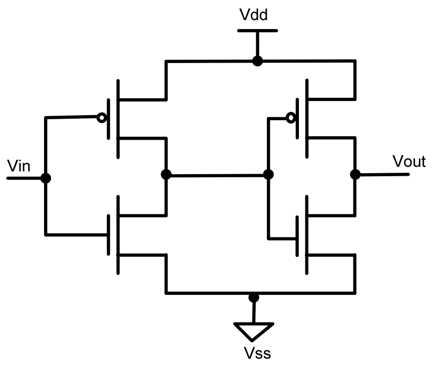


Figure 6-3 Cascaded inverter based delay element

NP- Voltage Controlled

A NP-voltage controlled delay element consists of a cascaded inverter pair with a series connected NMOS and PMOS transistor in the pull-up and pull-down path [15]. The NMOS and

PMOS transistors have their gate inputs connected to a control voltage designated as Vn and Vp respectively. The control voltage together with the transistor sizes control the charging and discharging time of the load capacitance and hence the total delay between the input and the output.

Figure 6-4 shows the schematic of a NP-voltage controlled delay element. The N-Voltage controlled and P-Voltage controlled delay elements are different flavors of this design. The signal integrity of the NP voltage controlled delay element is poor because of the slow rise and fall times associated with the output.

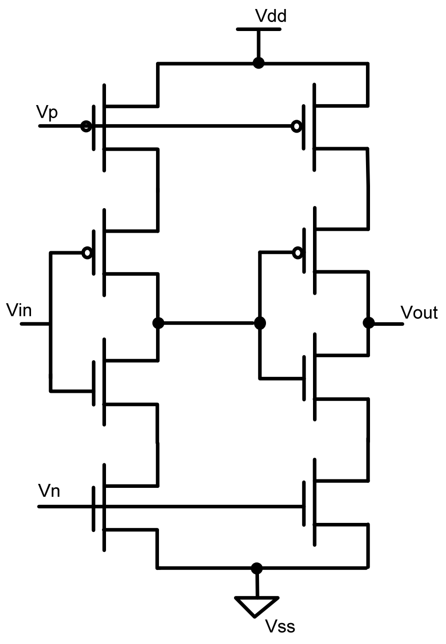


Figure 6-4: NP-Voltage Controlled delay element

NP-Voltage controlled with Schmitt Trigger

The signal integrity of a NP-voltage controlled delay element can be improved by using a Schmitt trigger in series which converts the slow edges to fast edges [15]. However, as in many other delay elements, the slow edges due to the NP voltage controlled delay element will cause short circuit power dissipation at the input stage of the Schmitt trigger.

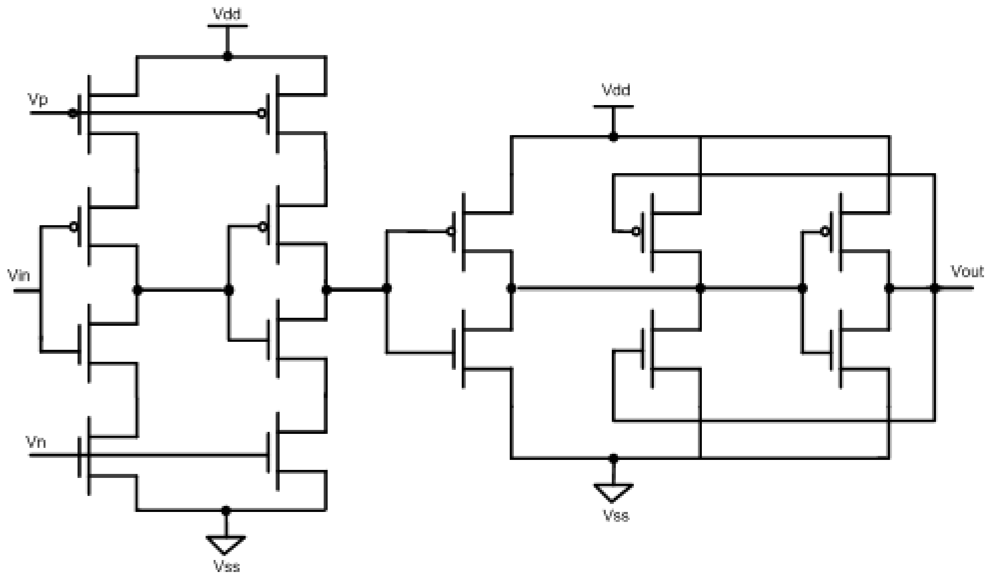


Figure 6-5: NP-Voltage Controlled delay element with Schmitt Trigger

N-Voltage Controlled

For an N-Voltage controlled delay element, varying the control voltage Vn changes the delay by changing the discharging current. When there is a rise transition at the input, the first inverter discharges through the control transistor path while the second inverter charges the output normally. The signal integrity is similar to a cascaded inverter case. The power consumption will be similar to the cascaded inverter case, except for the additional control voltage transistors, which contribute additional power due to the additional diffusion capacitance [15]. Figure 6-6 below shows the N-voltage controlled delay element.

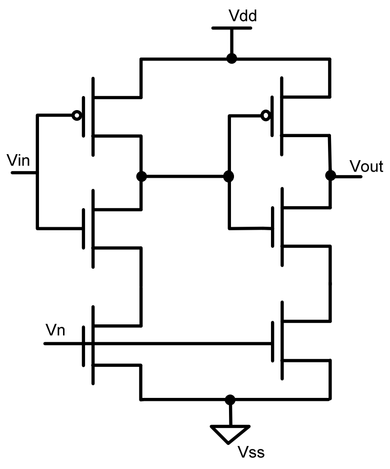


Figure 6-6: N-Voltage Controlled delay element

P-Voltage Controlled

Another variant of the NP-voltage controlled delay element is the P-voltage controlled delay element. The control voltage Vp changes the delay by changing the charging current. The analysis of the P-voltage controlled delay element is similar to the N-voltage and NP-voltage controlled delay elements. Figure 6-7 below shows the P-voltage controlled delay element.

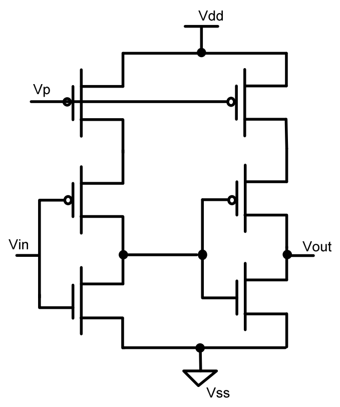


Figure 6-7: P-Voltage Controlled delay element

Current Starved Cascaded Inverter

The structure is similar to a NP-voltage controlled transistor with the modification that the PMOS control voltage is generated using a current mirror structure. The signal integrity of this delay element is also poor because of the slow rise and fall times at the output.

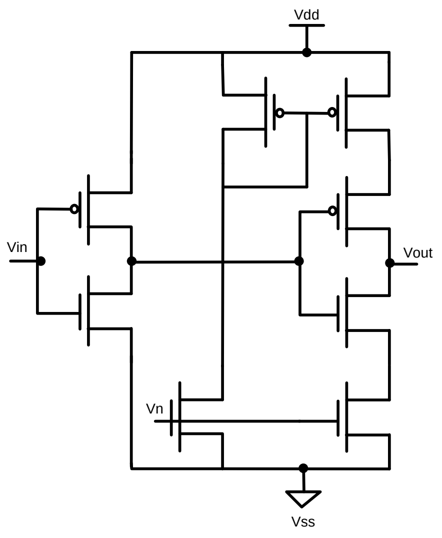


Figure 6-8: Current Starved Cascaded Inverter

m-Transistor Cascaded Inverter

This delay element has a cascode like structure with series connected PMOS and NMOS in its pull-up and pull-down path respectively [15]. The gates of all these PMOS and NMOS transistors are connected to the input. The cascode like structure has an increased ON-resistance and capacitance. The additional capacitance arises from the diffusion capacitances of the source and drain regions of the additional series transistors [15].

The m-transistor cascade has poor signal integrity because of the increased charging and discharging resistance [15]. The power consumption is slightly higher than a simple inverter chain because of the additional node capacitances in this structure.

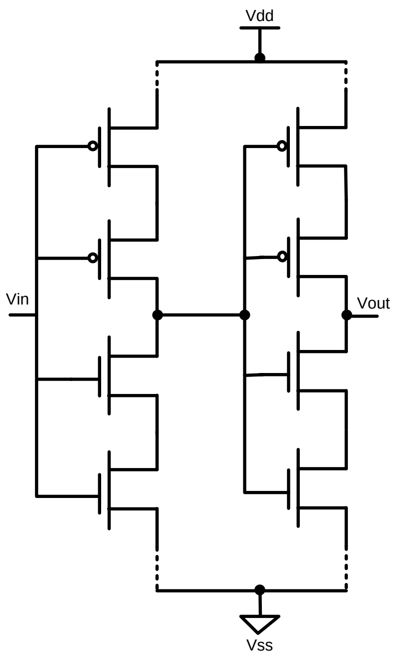


Figure 6-9: m-Transistor Cascaded Inverter

Staged Cascaded Inverter

The Staged cascaded inverter based delay element has stages of inverters. The first stage has two inverters and the second stage has a single inverter. The first stage is called the input stage and the second stage the output stage. The output of the first stage of inverters controls the transistors of the next stage [15].

The primary advantage of this design is that the short circuit power dissipated on the output stage when both the transistors are ON can be virtually eliminated by sizing the input transistors appropriately [15]. The sizing of the input transistors can ensure that the output PMOS is turned OFF before turning ON the output NMOS and vice versa. The input stage inverters primarily control the delay of the delay element. The range of delay that can be obtained with this delay element topology is minimal.

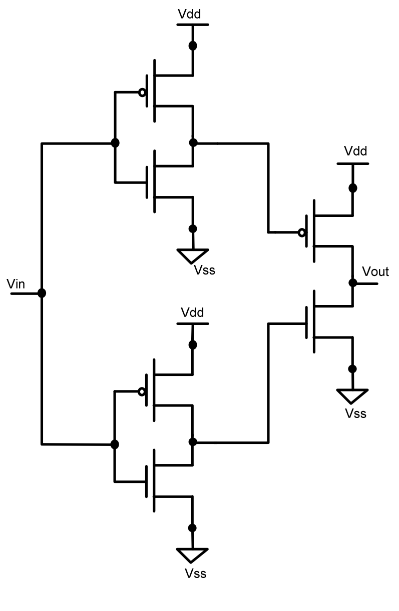


Figure 6-10: Staged Cascaded Inverter

6.2 LOW POWER FABRIC USING DELAY ELEMENTS

Delay elements can be used in the SuperCISC Reconfigurable Hardware Fabric to save a significant amount of glitching power. As the entire hardware fabric uses a combinational flow, timing differences between two paths can cause glitches at the output of the combinational block. These glitches contribute significant amount of power in the operation of the device. Delay elements are used to enable the latches at the data inputs to the functional unit. The latches are enabled at time t0 which is the maximum of the all the data path delays to the functional unit. Thus freezing of the gate inputs until all data inputs become available minimizes the glitch power. Thus delay elements can be used to minimize the power.

For an example consider the combinational circuit shown in Figure 6-11 . The adders have a propagation delay of 1.2ns, multipliers have a delay of 4ns and subtractors have a delay of 1.25ns. The critical path on the left side of the final multiplier M1 is 5.25ns and on the right side is 4ns. The final multiplier switches continuously during the entire duration of 9.25ns,which is the sum of the maximum critical path delay of the operands (5.25ns) and the propagation delay of the multiplier M1(4ns) itself.

Figure 6-12 shows the latches (L) that are introduced at the inputs to the operands of the multiplier M1. The latches L are enabled by a 5ns delay element. As the inputs to the multiplier M1 are enabled only after 5ns, M1 switches for 0.25ns due to the left input operand and for 4ns due to its own propagation delay. Thus the total switching time is only 4.25ns. As shown, the delay element has reduced the switching time from 9.25ns to 4.25ns, hence reducing the glitching power consumption of the system.

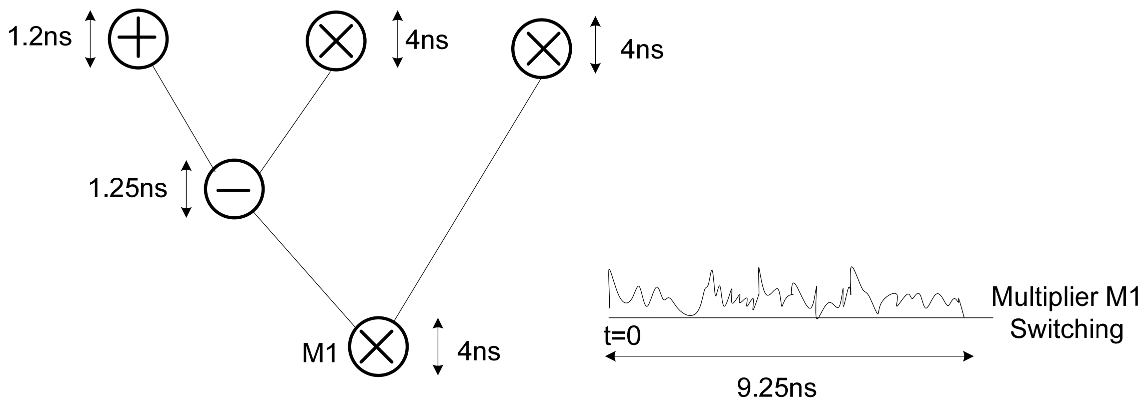


Figure 6-11: Combinational switching without delay elements

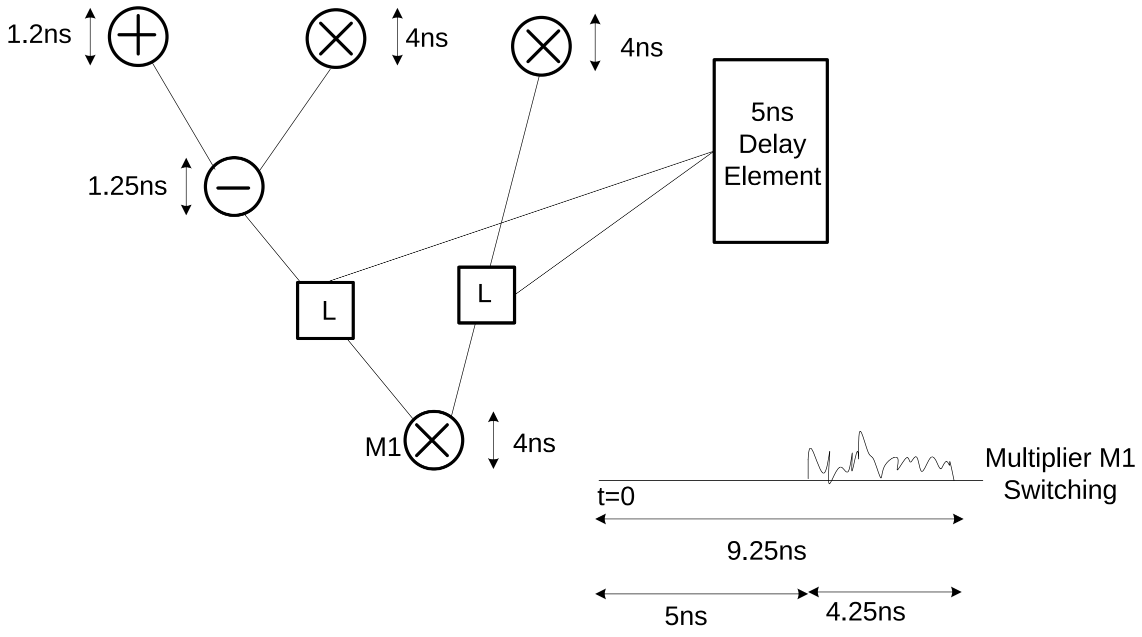


Figure 6-12: Combinational switching with delay elements

The use of delay elements in the hardware fabric requires that the delay elements have a wide delay range. Also, as the delay elements are used to enable latches, which are in the critical path of the design, increase in the delay value can cause performance degradation while a decrease in the delay value can cause an increase in the glitch power. Variations in delay values are considerable with supply voltage, temperature and process variations. Hence, a delay element which is less sensitive to these variations is essential to reduce power in the hardware fabric with minimal performance degradation.

6.3 THYRISTOR BASED DELAY ELEMENT

The need for a delay element with a low power characteristic and low supply and temperature variation was the driving factor in the use of the CMOS thyristor based delay element. In addition, this delay element also possesses good signal integrity. The thyristor based delay element as proposed by [17] is less sensitive to voltage and temperature variations as the major component of the delay is controlled by the current source.

6.3.1 CMOS Thyristor Concept

The working of the thyristor based delay element is based on the principle described below. Consider the circuit as shown in Figure 6-13. Let node Q~ be charged to Vdd and node Q be discharged to ground in the beginning. This assumption is valid as can be seen in the dynamic triggered delay element design. Let Penable be asserted high and Nenable be asserted low. When the input signal D transitions, node Q~ begins to discharge with current Ictrl. Once the voltage at node Q~ drops below the threshold voltage of the PMOS transistor M3, the transistor M3 turns on and begins to charge node Q. Once node Q has charged to the threshold voltage of the NMOS transistor M2, the transistor turns on and discharges node Q~ faster. The transistors M3 and M2 now form a positive feedback which makes the switching action faster. As the delay in the positive feedback action is small, the total delay is dominated by the delay due to discharge of Q~ using the discharge current I_{ctrl} and the charging of Q. The signal integrity of the thyristor based delay element is good because of the positive feedback action which causes the output edges to be sharp [16].

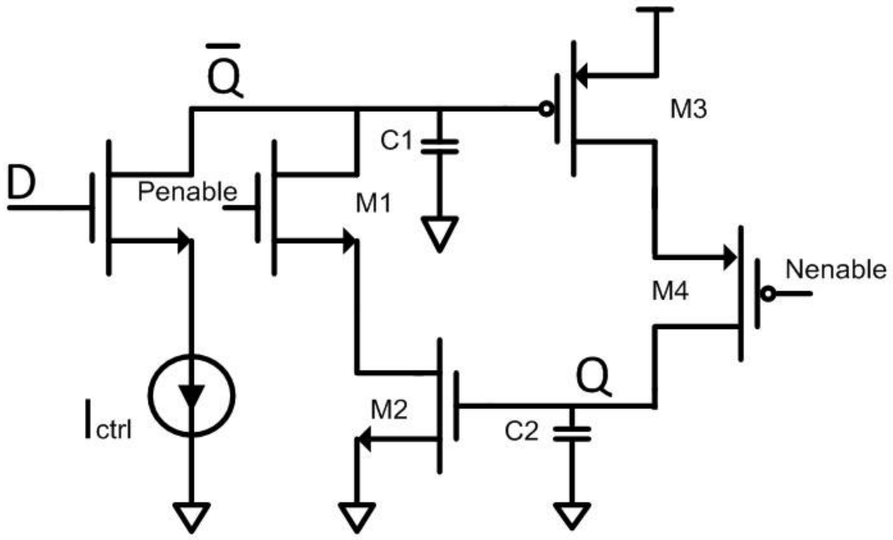


Figure 6-13: CMOS Thyristor structure

6.3.2 Dynamic Triggering Scheme

A dynamic triggering scheme [17] based delay element using the CMOS Thyristor is shown in Figure 6-14. The delay element uses two CMOS thyristors, the left half and the right half. The left half CMOS thyristor ensures delay on the rising edge of D and the right half ensures the delay on the falling edge of D considering Q as the output. Also the two CMOS thyristors help to restore the charge on the nodes Q and Q~. To be specific, let us consider Figure 6-13. Once node Q~ gets discharged, the standalone CMOS thyristor does not have a way to restore the charge on node Q~ for the next cycle. The two CMOS thyristors help to avoid this situation. Also the signals PENABLE and NENABLE help to avoid the shunt current condition during activation [17].

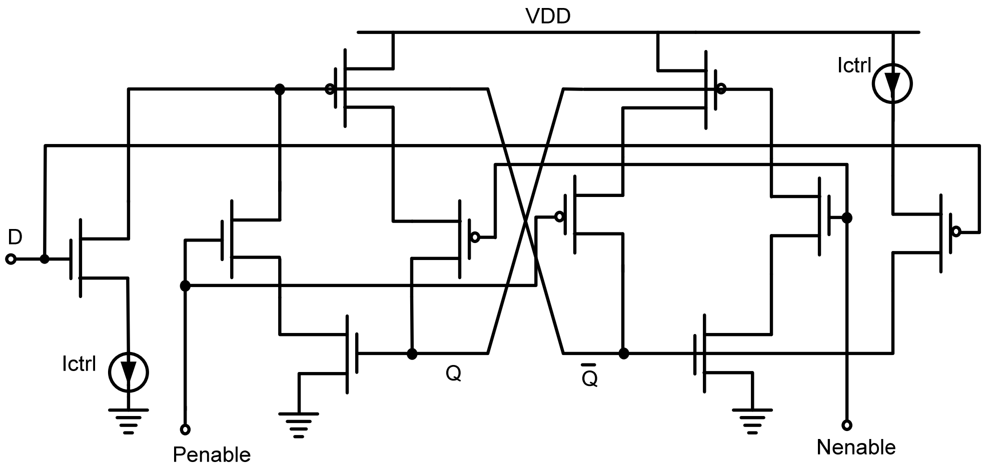


Figure 6-14: CMOS Thyristor Dynamic Triggering Scheme

Dynamic triggering prevents shunt current that would cause higher power consumption and spike noise on the power line [17]. But this requires an internal delay element τ for dynamic timing generation [17]. The delay τ is required for disconnecting the path between Vdd and ground using the Penable and Nenable signals. The delay τ can be generated using an inverter chain. The delay between the input and output is not affected by the internal delay τ required for generating the Penable and Nenable signals [17].

6.3.3 Static Triggering Scheme

A simpler version of the dynamic triggering scheme is the static triggering scheme [17] shown in Figure 6-15. The static triggering scheme avoids the generation of Penable and Nenable signals and this causes shunt current in the design.

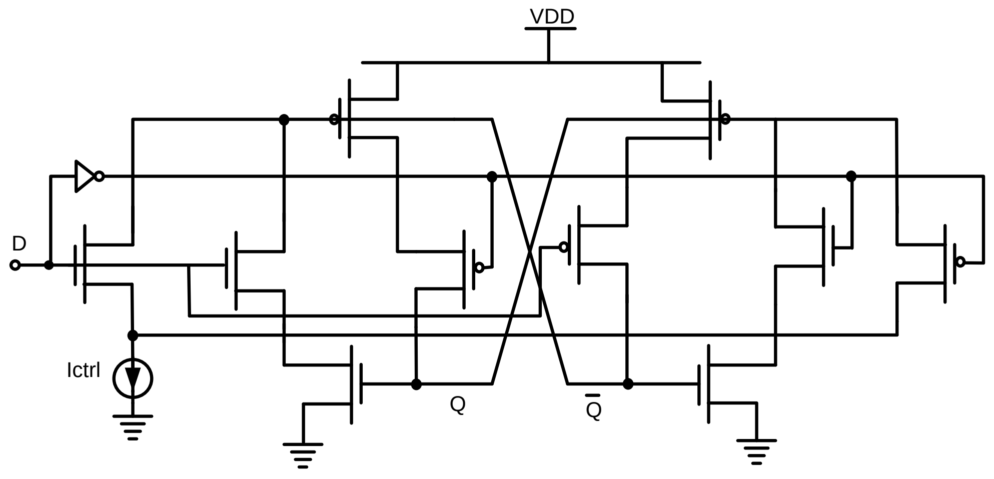


Figure 6-15: CMOS Thyristor Static Triggering Scheme

Figure 6-16 shows the shunt current condition when the D transitions from a logic low to a logic high state. When D transitions from a logic low to a logic high, the left thyristor turns on while the right thyristor turns off. These simultaneous actions cause a shunt current path from Vdd to ground. This shunt current can be avoided by turning off the right thyristor once node Q~ reaches Vdd. The Penable signal is delayed version of signal Q~ which can be used to turn of the right thyristor and turn on the left thyristor for the next cycle. Figure 6-17 shows the shunt current condition when D transitions from a logic high to a logic low state. The Nenable signal is a delayed version of signal Q which can be used to turn off the left thyristor and turn on the right thyristor. The Nenable prevents the shunt current during a transition from a high state to a low state on the D input.

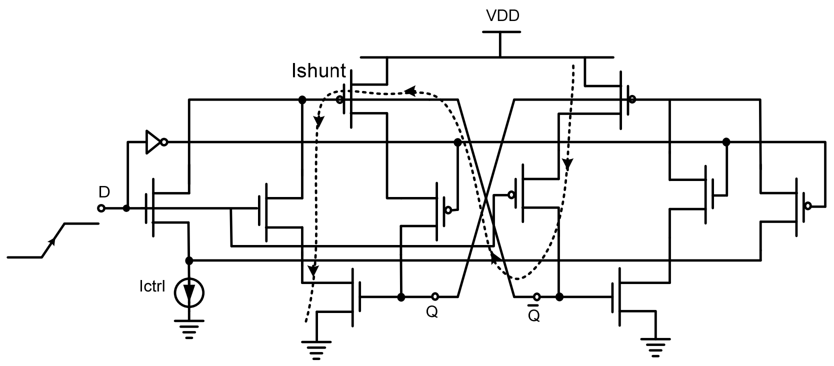


Figure 6-16: CMOS Thyristor Shunt current when D transitions to a high

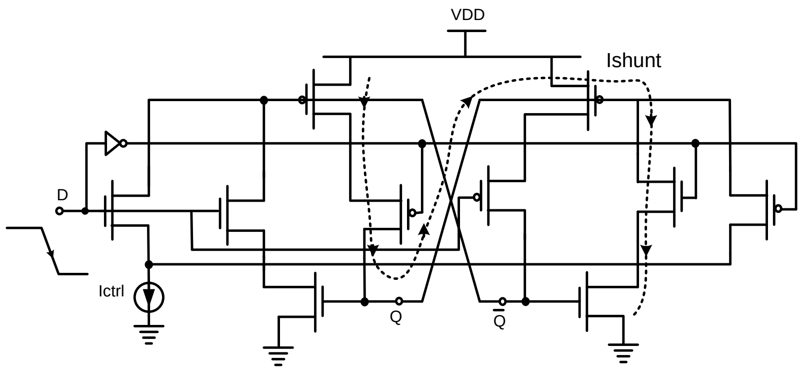


Figure 6-17: CMOS Thyristor shunt current when D transitions to a logic low

6.3.4 Delay Analysis of the Thyristor Based Delay Element

The total delay using the thyristor based delay element can be split into three components and an analytical model has been derived by [17][18]. Let us consider the rising edge of the input signal. The first component of the delay can be attributed to the delay in discharging node Q~ from Vdd to Vdd-Vtp, where Vtp is the threshold voltage of the PMOS. The second component of the delay can be attributed to the delay in charging node Q from 0V to Vtn, where Vtn is the threshold voltage of the NMOS. The third component of the delay is from the regeneration time of the CMOS thyristor during positive feedback. Equation 6-1 shows the total delay in the delay element.

$$\text{Total Delay} = t_{d1} + t_{d2} + \delta_{\text{t}}$$

Equation 6-1: Thyristor Delay Components

t_{d1} is the delay in discharging node Q~ from Vdd to Vdd-Vtp. t_{d2} is the delay in charging node Q from 0V to Vtn and δt is the regeneration time of the CMOS thyristor. The delay in discharging node Q~ from Vdd to Vdd-Vtp is determined by the capacitance at node Q~ and the value of the current source used for discharging. Using a current value of I_{ctrl} the delay *t_{d1}* is given by Equation 6-2.

$$t_{d1} = \frac{C_1 V_{tp}}{I_{ctrl}}$$

Equation 6-2: Delay due to Control Current

After discharging node Q~ to Vdd-Vtp ,transistor M3 as shown in Figure 6-13 turns on. As transistor M3 is in saturation when it turns on, the drain current I_{d1} is given by Equation 6-3.

$$\begin{aligned} V_{gs1} &= \left| V_{Q\sim} - V_{dd} \right| \\ &= V_{tp} + \frac{I_{ctrl}\, t}{C_1} \\ I_{d1} &= \frac{\mu_p C_{ox}}{2} \frac{W}{L} (V_{gs1} - V_{tp})^2 \\ &= \frac{\mu_p C_{ox}}{2} \frac{W}{L} \frac{I_{ctrl}{}^2 t^2}{C_1^2} \end{aligned}$$

Equation 6-3: Drain Current in CMOS Thyrsitor

The delay t_{d2} in charging the node Q from ground to Vtn using the drain I_{d1} is given by Equation 6-4. The total thyristor delay t_d is given by Equation 6-5.

$$\int_{0}^{td1} I_{d1} dt = V_{tn} C_2$$

$$t_{d2} = \sqrt[3]{\frac{6C_1{}^2 C_2 V_{tn}}{\mu_p C_{ox} (W/L)}}$$

Equation 6-4: Delay in charging node Q

$$td = \frac{C_1 V_{tp}}{I_{ctrl}} + \sqrt[3]{\frac{6C_1{}^2 C_2}{\mu_p C_{ox} (W/L) I_{ctrl}{}^2} V_{tn}} + \delta t$$

Equation 6-5: Total Thyristor Delay

δt is the regeneration time of the CMOS thyristor and is a minor component of the delay value but is sensitive to the variation in the supply voltage. Since the major components of the delay values are insensitive to the supply voltage, the characteristic of this delay element is less sensitive to the variation in the supply voltage.

As shown in Equation 6-5, with smaller I_{ctrl}, i.e., larger delay value, the delay element gets less sensitive to the supply voltage variation. Equation 6-5 clearly shows that the delay value is controlled by the control current I_{ctrl}. The second term in the equation shows the temperature dependency of the delay with mobility μ_p.

6.4 MODIFIED CMOS THYRISTOR DELAY ELEMENT

The CMOS thyristor based delay element as suggested by [17] suffers from some drawbacks. The fundamental drawback arises from charge sharing between the nodes. The assumption that node Q~ is at Vdd before being discharged by the current source is not valid. This is because the rising edge of the D input and the signal Penable cause the charge stored on the parasitic capacitance C0 (shown in Figure 6-18) to be shared with capacitances C1 and C2. The charge sharing effect causes the voltage at node Q~ to drop from Vdd to Vdd * (C0/(C0+C1+C2)). This reduction in voltage affects the delay because the node Q~ instead of getting discharged from Vdd , gets discharged from a lower voltage. Thus the charge sharing phenomenon causes a decrease in the delay value.

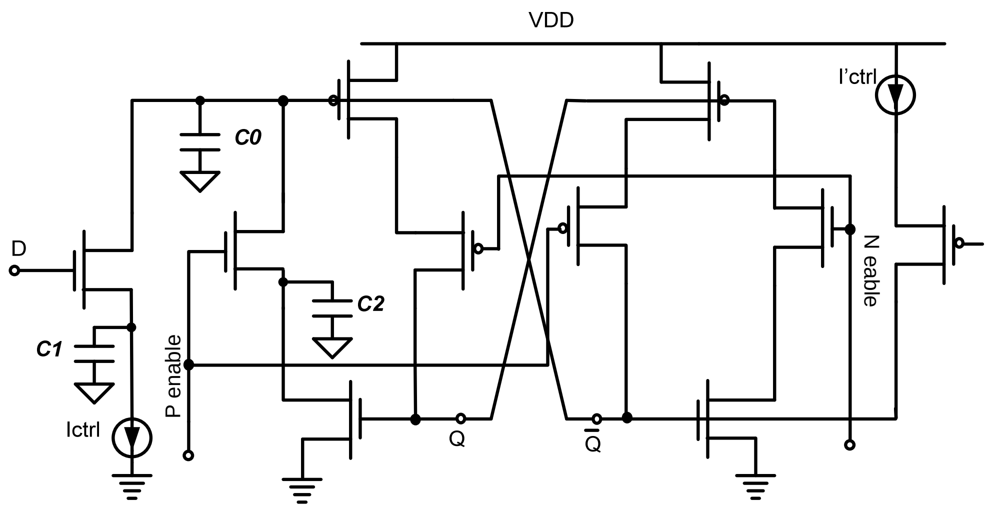


Figure 6-18: CMOS Thyristor showing Parasitic Capacitances

Also mismatches between the current sources Ictrl and Ictrl` can cause the delays between the rising edge to be different from the falling edge. As the delay element used for timing the SuperCISC reconfigurable hardware fabric does not use a delay on the falling edge of the signal, this effect is not pertinent to the application in hand.

An architecture modified to solve the above issues has been proposed by [18]. The schematic of the modified CMOS thyristor delay element is shown in Figure 6-19. The signals Qcharge and Q~charge are used to cancel the charge sharing effect. Qcharge is generated by the NAND of inputs D and Nenable. Q~charge is generated by the NAND of inputs Dbar and PEN. The prime motive in having signal Q~charge is to enable the PMOS transistor that can be used to replenish the charge on node Q~ when PEN gets enabled. The signal Q~ charge needs to be asserted low only as long as the D input is low. Hence, a NAND of inputs PEN and Dbar accomplishes this purpose. Similarly Qcharge gets activated to replenish the charge on node Q when NEN gets enabled. As the Qcharge and Q~charge are driving the gates of PMOS transistors, they are active low signals. Figure 6-20 below shows the timing diagram of the modified CMOS thyristor based delay element.

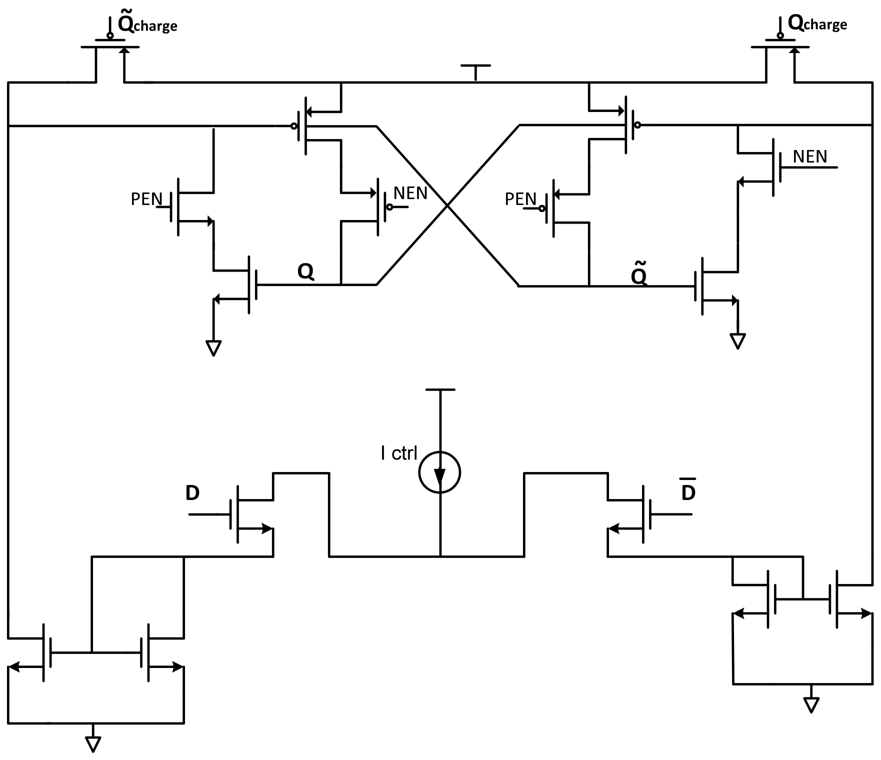


Figure 6-19: CMOS Modified Thyristor Delay Element

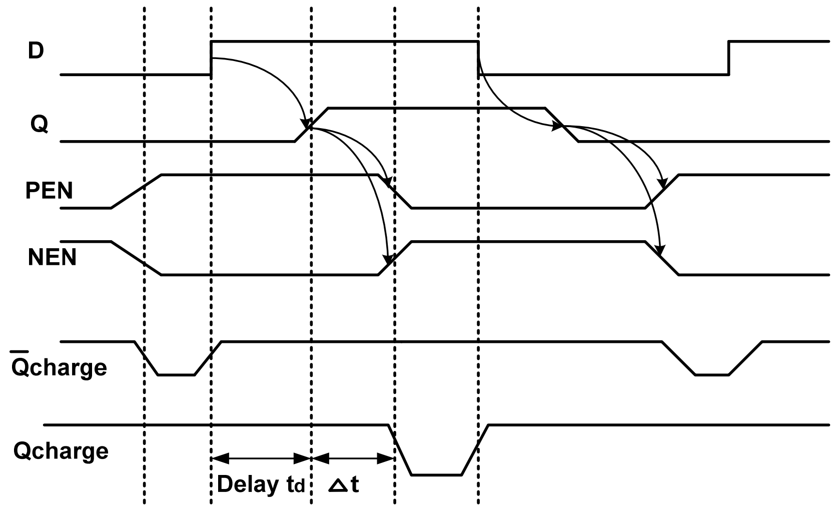


Figure 6-20: Timing diagram of the modified CMOS delay element

6.5 CUSTOM CMOS THYRISTOR DELAY ELEMENT FOR FABRIC

The original circuit of Figure 6-19 has been modified to suit the application needs of the SuperCISC Reconfigurable Hardware Fabric. The circuit schematic of the delay element used in the fabric is shown in Figure 6-21.

The delay element to be used in the SuperCISC reconfigurable hardware fabric, drives the enable inputs of the latches. The latches when disabled, freeze the inputs to the computational unit. The latches when enabled pass the inputs to the computational unit. The time between the input to the hardware fabric and the valid output from the hardware fabric can be considered as one processing cycle of the fabric. All latches used in the hardware fabric for delaying the inputs are to disabled at the beginning of the processing cycle and need to be enabled at the time points when they start the computation. All latches used in the context of delay elements are enabled on a logic high and disabled on a logic low. As can be inferred from this application, only the rising edge of latches should be delayed while the falling edge should not be delayed. Hence, the delay elements that drive these latches need to have the same property.

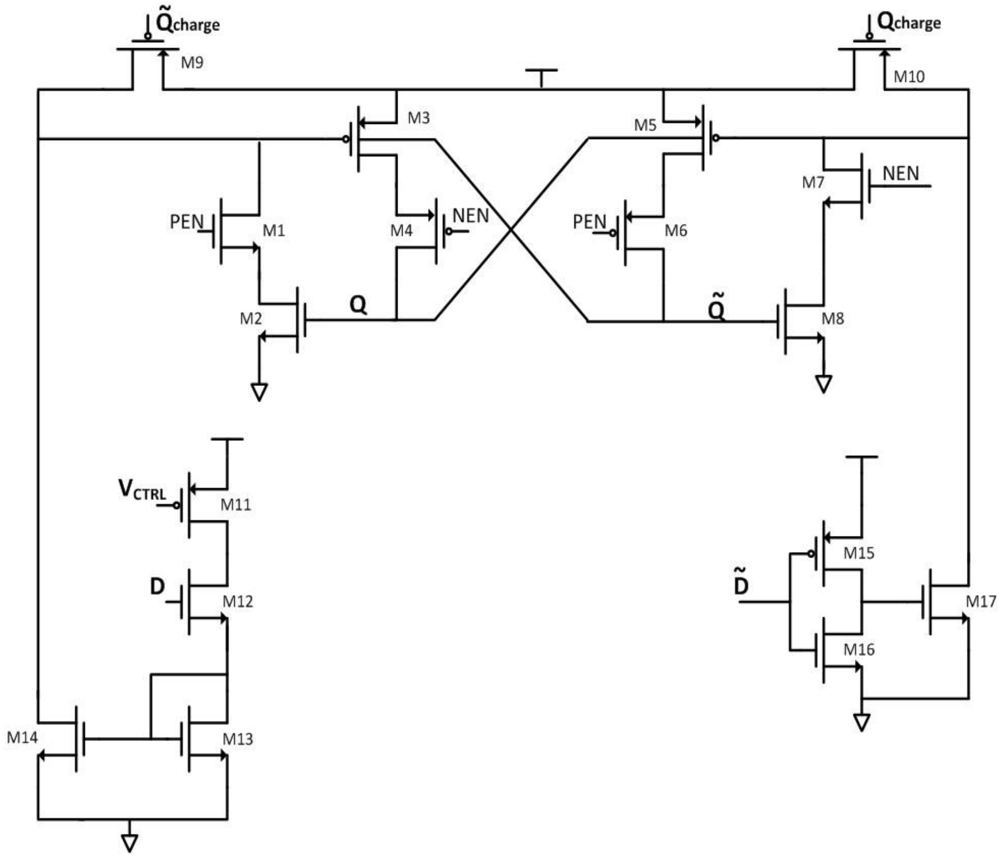


Figure 6-21: Custom CMOS Thyristor based Delay Element

To meet the design requirement of having an undelayed falling edge, the current source required to discharge Q has been replaced with an NMOS transistor. The NMOS transistor M17 gets enabled when D input makes a transition from a logic high to a logic low. The NMOS transistor discharges node Q at a fast rate and hence the delay element does not have a delay on the falling edge of the signal.

Input Vctrl of transistor M11 acts as the current source for the delay element. The signal Vctrl is generated from a constant *gm* based current source circuit and the transistor M11 mirrors the current. The current source used in the design is based on a 100nA current source.

6.5.1 Programmability

For a 20X18 ALU configuration of the hardware fabric, there would be 360 ALU nodes (assuming homogeneous ALU configuration).The delay elements that are used can have value of 4ns, 5ns and 7ns, which totals to 3 delay elements per ALU node. Use of 3 delay elements per ALU node makes the number of delay elements used in the fabric to be around 3*360. The use of so many delay elements is prohibitive from an area perspective. So the idea of using a programmable delay element based on scaling the current source helps to overcome the overhead in having multiple delay elements. This programmability reduces the number of delay elements from 1080 to 360.

One possible configuration would be to scale the input current mirror transistor connecting to signal Vtrl by having multiple PMOS transistors in parallel with enable switches connected to them in series. The major drawback behind this application is from the fact that the sizes of the input current mirror transistors are large. So, when the D input is off, charge gets stored on the drains of the large current mirror transistors and when the D input gets turned ON charge sharing happens between the drain of the current mirror transistor and the drain of the diode connected NMOS transistor. This charge sharing causes a huge current to be mirrored on the path that discharges Q~. This limits the range of delay that can be achieved with the programmable delay element.

An alternative solution that has been used is to scale the current mirror transistors in the path that discharges Q~. The inputs C0,C1,C2 connected to the gates of the NMOS transistors M22,M24 and M26 as shown in Figure 6-22 act as control switches that can be turned ON to scale the discharging current. As the switches are statically configured before the start of operation, the appropriate switches are always ON or always OFF. Using this approach, a well-controlled delay range has been obtained.

The use of programmable current mirrors comes at the expense of an increase in the gate capacitance seen at the gate of the diode connected NMOS transistor M13. Under normal operating conditions, when D is turned ON, the gate of the diode-connected transistor rises to a potential depending on the charging current. When D gets turned OFF, the gate gets discharged to the threshold voltage of the diode connected NMOS transistor. With the increase in the gate capacitance, the time to discharge the node to Vtn (threshold voltage of NMOS) becomes higher. Also, during this time as Q~ gets replenished through Q~charge transistor M9, a static current path exists between node Q~ and ground because of the potential being above the Vtn of the NMOS transistors. Even under normal operating conditions, when D is OFF, the Vtn voltage on the diode-connected NMOS causes sub-threshold current leakage from node Q~ to ground. To alleviate this problem, an additional NMOS transistor M19 has been added with the drain connected to the gate of the diode-connect NMOS and the source connected to ground. The gate of transistor M19 is connected to dbar (D after inverted). So, when D is OFF, dbar gets enabled and discharges the gate of the diode-connected NMOS to ground. Thus the circuit eliminates static power consumption. Appendix D which contains the characterization data for the delay elements shows the off-state leakage power consumed by the device.

After the delay has been accomplished, the current path from Vdd to ground through transistors M11,M12 and M13 when D is ON is a source of unwanted power consumption. So the use of a control signal Dctrl to turn OFF the input to transistor M12 after the delay happens, eliminates the unwanted current consumption. Dctrl is generated using the AND of inputs D and PEN as shown in Figure 7. The circuit schematic of a programmable delay element is shown in Figure 6-22. Figure 6-23 shows the AND gate to generate the Dctrl signal. The control bits for programming the delay value can be set during the configuration phase of the reconfigurable hardware fabric.

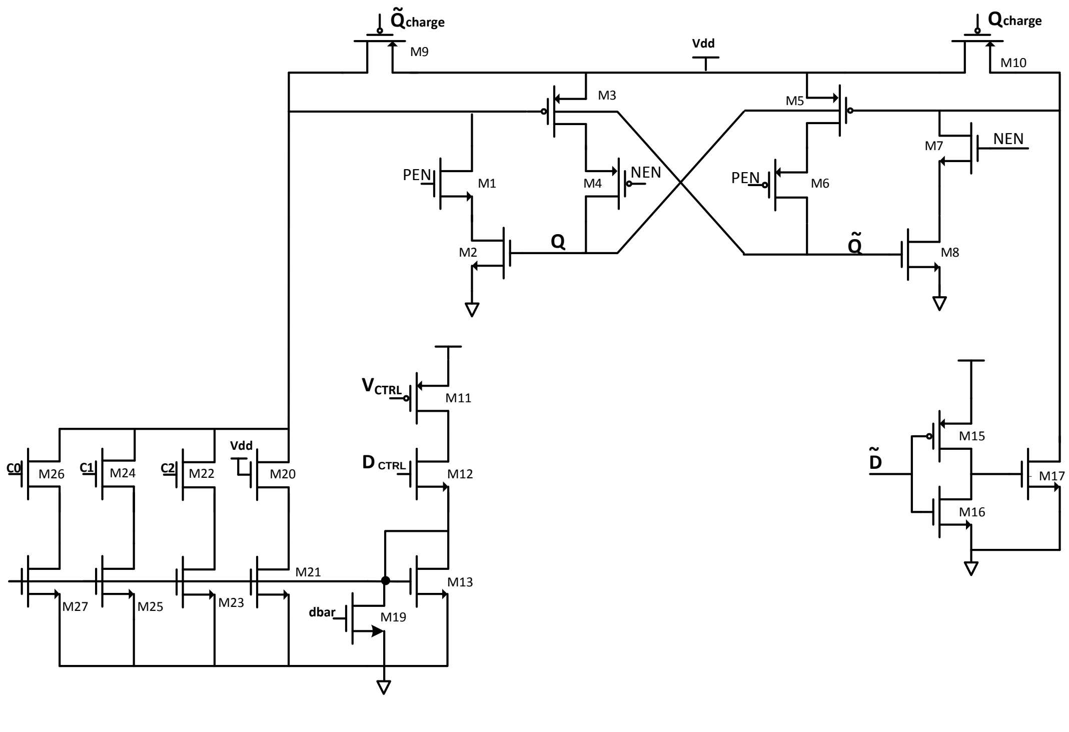


Figure 6-22: Programmable delay element

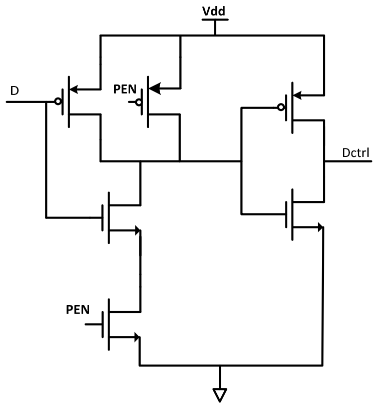


Figure 6-23: AND gate to generate Dctrl

6.5.2 Layout of the Programmable Delay Element

The layout of the programmable delay element was implemented in Cadence Virtuoso XL using the IBM 0.13um CMOS process design kit. The delay element has a size of about 97.8um X 3.6um giving a total area of about 352.08sq.um. This area does not include the buffers used to characterize the delay cell. Figure 6-24 below shows the layout of the programmable delay element. The design was DRC checked and LVS verified using Mentor Graphics Calibre.

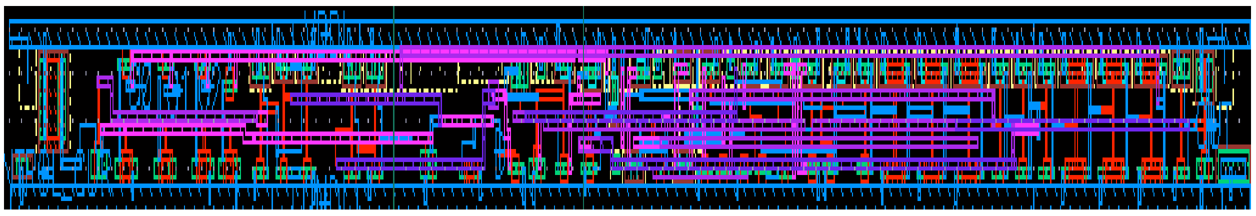


Figure 6-24: Layout of the Programmable Delay Element

6.5.3 Parasitic Extraction of the Delay Element

To annotate the design with parasitics from the layout, parasitic extraction was done using Mentor Graphics Calibre xRC. The parasitic capacitance annotated HSPICE netlist was used for characterizing the complete design. The tool annotates both the wire capacitance to ground as well as the coupling capacitance between nets.

6.5.4 Cell Characterization and HSPICE Post-Processing

The objective of building the delay element is to characterize its power characteristics for various delay values. The characterized data can be transformed into Synopsys Liberty Format to be used in the standard cell based ASIC design flow. The HSPICE post-processing script automates the cell characterization procedure. Appendix B contains the HSPICE post-processing script written in Perl to automate the cell characterization. The delay element cell was characterized for various values of load capacitance and input transition times. The HSPICE post-processor reads the HSPICE netlist and changes the load capacitance and transition times appropriately and the measurement commands suited for the simulation. The netlist is simulated in HSPICE after the post-processing step. The measurement data is written into a “netlist.mt0” file by HSPICE. The simulated data is read and the data format is changed to be readable by the user.

Three different versions of the standard cell were created with different drive strengths. These delay elements differ only in the output buffer used to drive the load. The first version of the delay element has the capability to drive a 80fF load. The next has the capability to drive a 160fF load and third has the capability to drive a 640fF load. Figure 6-25 shows the layout of a buffer with a drive capacity of 640fF. Multi-finger transistors were used to fit the width of the transistors within the standard cell height. Figure 6-26 shows a buffer with a drive capacity of 160fF. Figure 6-27 shows a buffer with a drive capacity of 80fF. The appropriate delay element can be chosen under a given load condition.

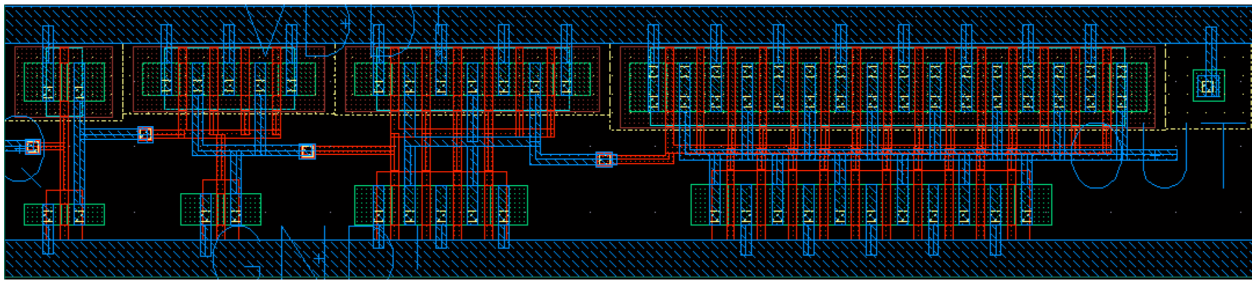


Figure 6-25: Layout of a 640fF drive Buffer

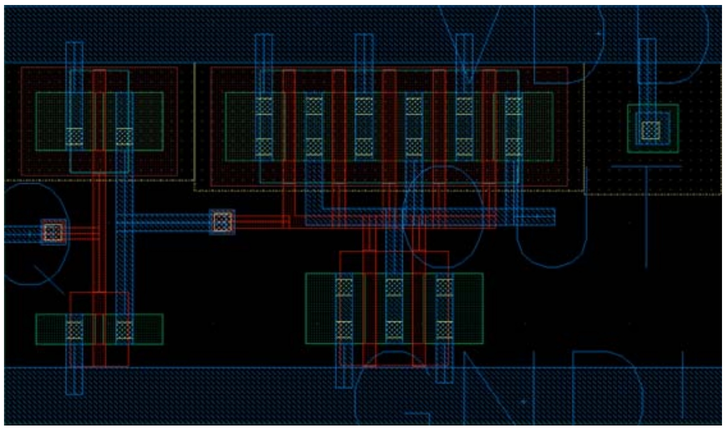


Figure 6-26: Layout of a 160fF drive buffer

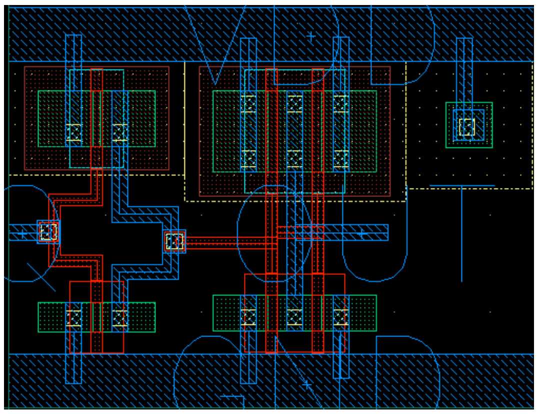


Figure 6-27: Layout of a 80fF drive buffer

The Liberty Format description of a delay element includes its rise power, fall power, cell rise delay, cell fall delay, rise transition time, fall transition time, leakage power during ON state, leakage power during OFF state. The Synopsys Liberty Format terminologies are described below.

Rise Power

Energy Consumed by the cell when the input makes a transition from a logic low to a logic high value.

Fall Power

Energy Consumed by the cell when the input makes a transition from a logic high to a logic low value.

Cell Rise Delay

The Cell Rise Delay is measured when the output makes a transition from a logic low to a logic high. Delay measured between the input reaching 50% of its final value to the output reaching 50% of its final value is defined as the Cell Rise Delay. Figure 6-28 shows the Cell Rise delay measurement waveform.

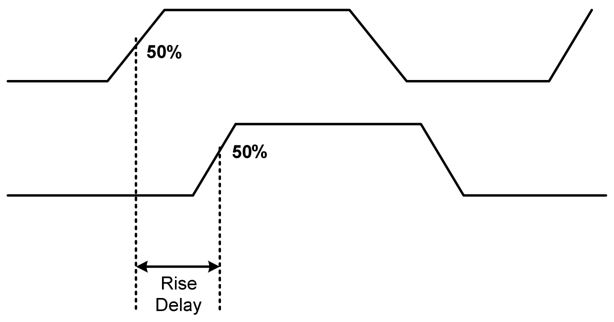


Figure 6-28: Cell Rise Delay Measurement

Cell Fall Delay

The Cell Fall Delay is measured when the output makes a transition from a logic low to a logic high. Delay measured between the input reaching 50% of its final value to the output reaching 50% of its final value is defined as the Cell Rise Delay. Figure 6-29 shows the Cell Fall delay measurement waveform.

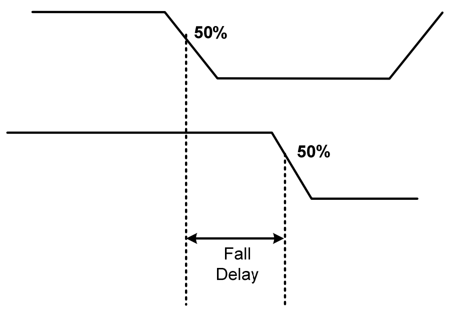


Figure 6-29: Cell Fall Delay Measurement

Rise Transition Time

The time taken for the output to reach from 10% of its final value to 90% of its final value is defined as the rise transition time. Figure 6-30 below shows the rise transition time measurement waveform.

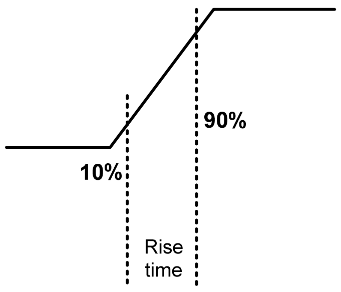


Figure 6-30: Rise Time Measurement Waveform

Fall Transition Time

The time taken for the output to reach from 90% of its final value to 10% of its final value is termed the fall transition time. Figure 6-31 below shows the fall transition time measurement waveform.

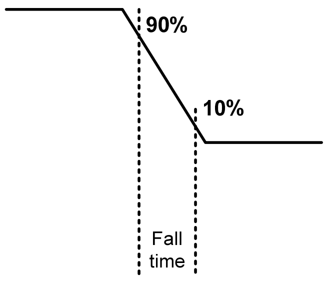


Figure 6-31: Fall Time Measurement Waveform

Leakage Power during ON State

The ON-state leakage power is measured after the input has transitioned to a logic high value and is in its steady state.

Leakage Power during OFF State

The OFF-state leakage power is measured after the input has transitioned to a logic low value and is in its steady state.

6.5.5 Characterization Results for a 4ns delay element

Figure 6-32 shows the input and the delayed waveforms for a 4ns delay element. Figure 6-33 shows the Q and Q~ nodes of the internal CMOS thyristor. Appendix D shows the characterization data collected for the 4ns delay element with different drive strengths.

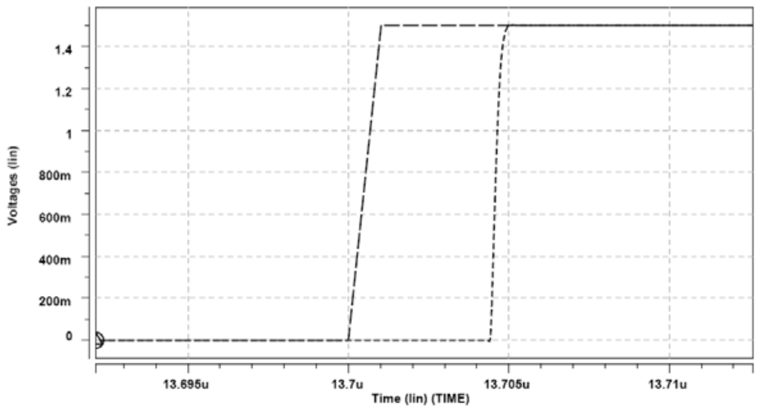


Figure 6-32: Input and Delayed waveforms for 4ns delay element

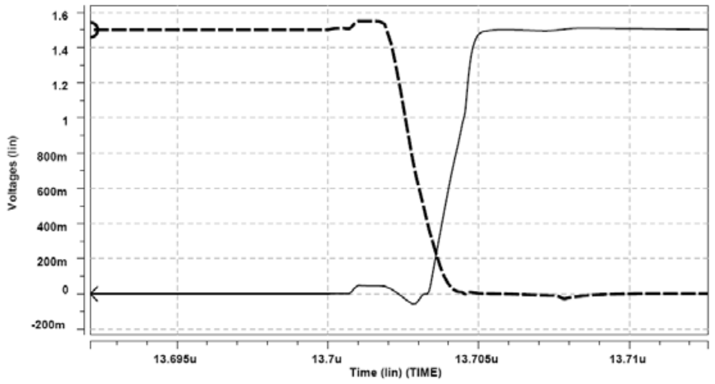


Figure 6-33: Q and Q~ discharge waveforms for 4ns delay element

6.5.6 Characterization Results for a 5ns delay element

Figure 6-34 shows the input and the delayed waveforms for a 4ns delay element. Figure 6-35 shows the Q and Q~ nodes of the internal CMOS thyristor. Appendix D shows the characterization data collected for the 5ns delay element with different drive strengths.

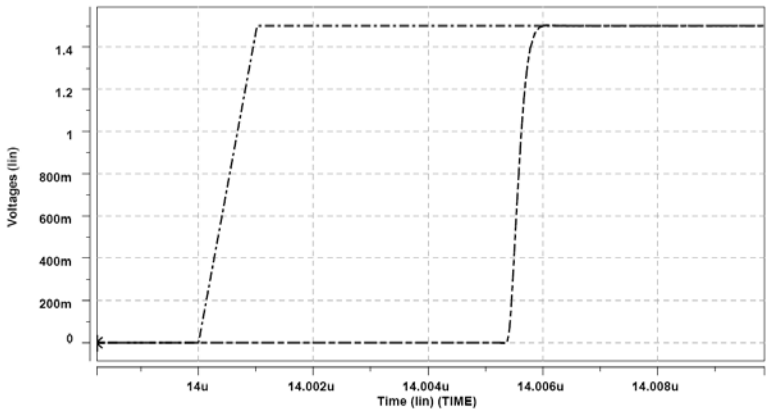


Figure 6-34: Input and delayed waveforms for a 5ns delay element

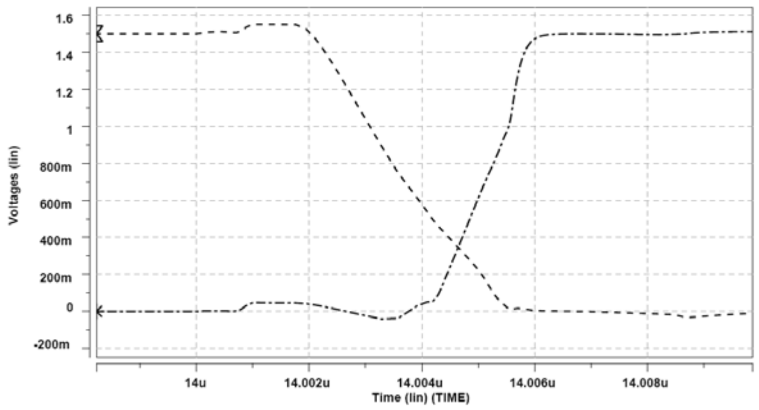


Figure 6-35: Q and Q~ discharge waveforms of a 5ns delay element

6.5.7 Characterization Results for a 7ns delay element

Figure 6-36 shows the input and the delayed waveforms for a 4ns delay element. Figure 6-37 shows the Q and Q~ nodes of the internal CMOS thyristor. Appendix D shows the characterization data collected for the 7ns delay element with different drive strengths.

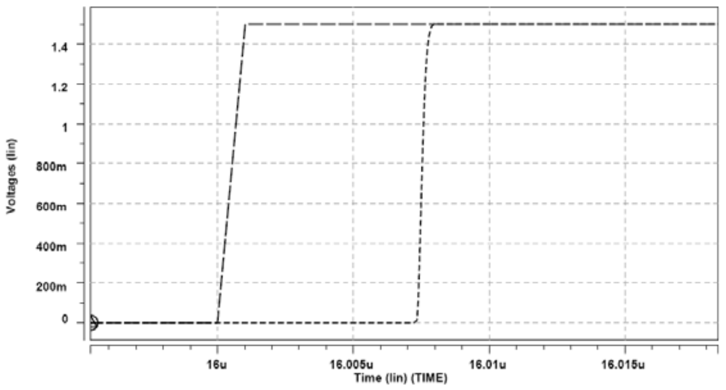


Figure 6-36: Input and delayed waveforms of a 7ns delay element

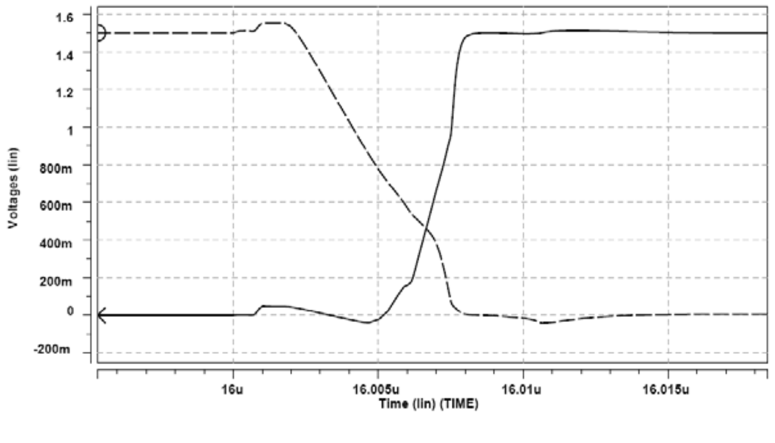


Figure 6-37: Q and Q~ waveforms of a 7ns delay element

7.0 EEPROM CIRCUIT DESIGN

EEPROM (Electrically Erasable/Programmable Read Only Memory) is a non-volatile memory device which has the capability to be read like an ordinary ROM as well as to be erased /programmed electrically to a new data value that needs to be stored. The fundamental component in the EEPROM cell is the FLOTOX (Floating Gate Tunnel Oxide) transistor. The following sections describe the implementation of the EEPROM design in a 0.35um process.

7.1 EEPROM CELL

The FLOTOX transistor is a device, which can be used to store a "bit" value. The EEPROM cell operates by changing the threshold voltage of the FLOTOX transistor [19]. When the transistor's threshold voltage is increased, the transistor does not conduct for normal applied voltages and is said to be erased. When the transistor's threshold voltage is reduced, the transistor begins to conduct even for a zero gate to source voltage and is said to be written. The structure of the FLOTOX shown in Figure 7-1 shows the gate oxide and tunnel oxide regions of the transistor.

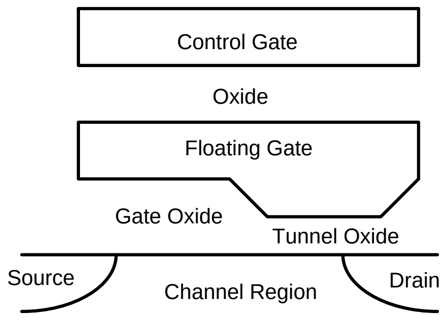


Figure 7-1: Structure of the FLOTOX Transistor

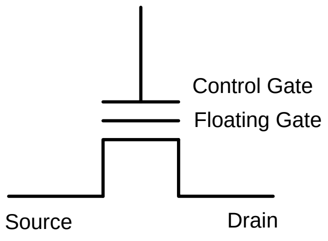


Figure 7-2: Symbol of a FLOTOX transistor

Figure 7-2 shows the symbol of a FLOTOX transistor. The EEPROM FLOTOX transistor has a Control Gate (CG) terminal, a Floating gate terminal (FG), a Source (S), a Drain (D) and a Substrate (B) terminal. The FLOTOX transistor is fundamentally an n-channel double-poly transistor in which the middle layer is floating and is called the floating gate [19]. The FLOTOX EEPROM has a region where the oxide thickness is very low and is called the tunneling window. The oxide at the tunneling window region is called the tunnel oxide. Figure 7-3 shows the IV characteristics of the FLOTOX transistor in its virgin state [19]. Virgin state of the FLOTOX transistor is the state when the transistor has neither been erased nor written.

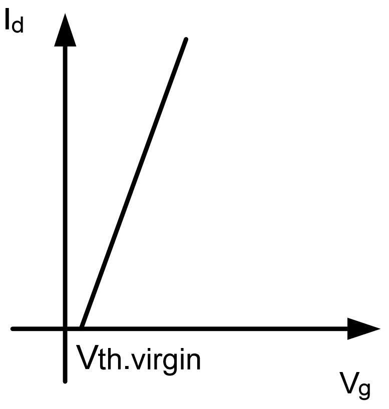


Figure 7-3: IV Characteristics of a virgin FLOTOX transistor

7.1.1 Erase Operation

An ERASE operation is said to be performed when a positive pulse is applied to the CG terminal while the S,B and D are grounded. The positive pulse causes a large electric filed across the tunneling oxide region. Fowler-Nordheim (FN) tunneling causes electrons to flow from the drain to the floating gate where they are stored. The negative charge on the floating gate attracts holes in the channel between the drain and the source. This effectively increases the threshold voltage of the device. Under this condition, conduction will not occur between the drain and source terminals for normal voltages that are applied to the gate terminal. The device is then said to be in an erased state. Figure 7-4 shows the physical operation using FN tunneling when the EEPROM cell is erased [19] and Figure 7-5 shows the charge on the floating gate after an erase operation [19].

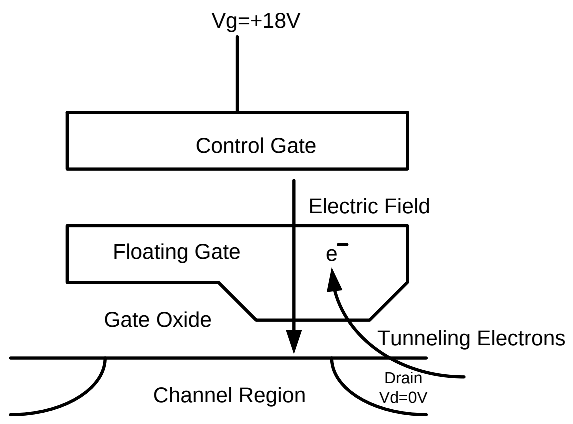


Figure 7-4: EEPROM Erase Physical Operation

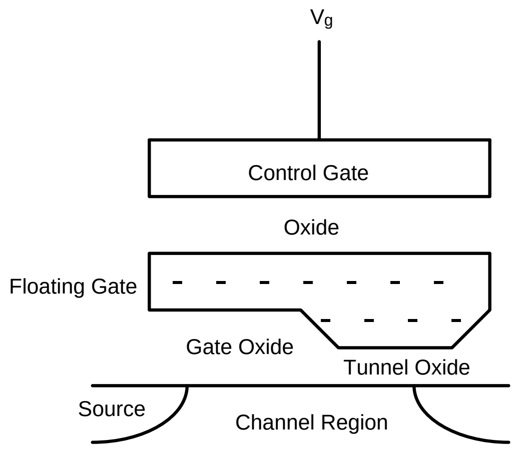


Figure 7-5: Charge on Floating Gate after erase operation

Figure 7-6 below shows the IV characteristics of the EEPROM in the erased state [19]. The figure shows an increase in the threshold voltage of the FLOTOX transistor after an erase operation.

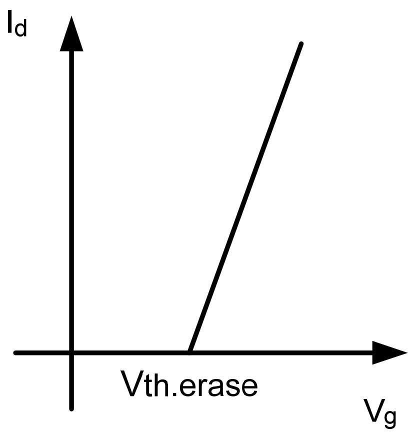


Figure 7-6: IV Characteristics of an erased FLOTOX transistor

7.1.2 Write Operation

A write operation is said to be performed when a positive pulse is applied to the drain while the source, bulk and control gate are grounded. Under such a condition, electrons flow from the floating gate to the drain because of Fowler-Nordheim tunneling. Thus the floating gate accumulates more positive charge and thus induces electrons to be attracted to the oxide - substrate interface. This causes a decrease in the threshold voltage of the transistor. Under such a state, the device allows conduction between the drain and the source for normal voltages applied to the gate terminal of the device. Figure 7-7 shows the physical operation using FN tunneling when the EEPROM cell is written [19] and Figure 7-8 shows the charge on the floating gate after the write operation [19].

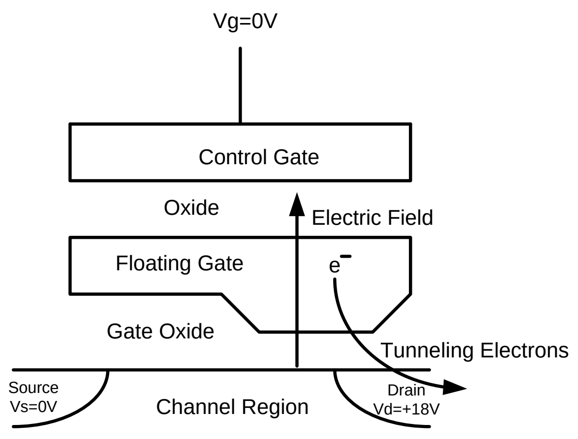


Figure 7-7: EEPROM Write Physical Operation

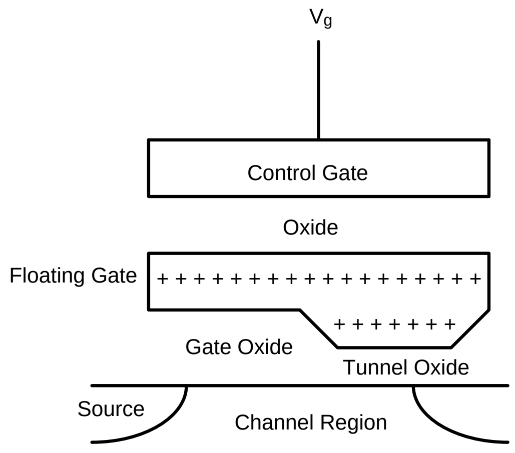


Figure 7-8: Charge on floating gate after write operation

Figure 7-9 shows the IV characteristics of the cell when the FLOTOX transistor is written [19]. The characteristic shows a decrease in the threshold voltage of the FLOTOX transistor when the cell is written.

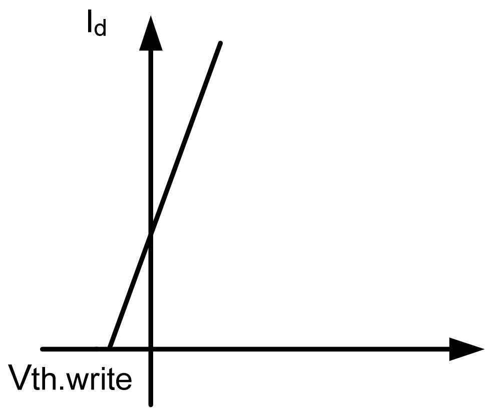


Figure 7-9: IV Characteristics of a written FLOTOX transistor

During the erase operation, the voltage on the floating gate node follows the gate control voltage until electrons get injected because of Fowler-Nordheim tunneling [20]. At the onset of tunneling, the floating gate voltage begins to decrease. Once the erase operation is completed by setting the gate control voltage to zero volts, the floating gate potential reaches a negative value depending on the amount of charges trapped on the floating gate [20]. The floating gate behavior is similar during a write operation.

7.2 EEPROM MEMORY ARCHITECTURE

The EEPROM memory in addition to the FLOTOX transistor requires special circuits for its operation. Some components of the design are similar to the SRAM (Static Random Access Memory) array while most of them are different. The following components are a part of the EEPROM array:

1. EEPROM cell
2. Sense Amplifier
3. High Voltage Generator Using Charge Pump
4. Ramp Generator
5. Word Line Level shifter
6. Column Latch for the bit lines
7. Power multiplexer

EEPROM Cell

Practical implementation of the EEPROM requires a memory process design kit with FLOTOX transistor models. In the absence such a design kit, EEPROM macromodels which model the DC and transient characteristics can be used to simulate the EEPROM. The EEPROM model used in the design has been based on the macromodel described in [21].

Macromodel Description

The macromodel description is entirely based on the model proposed by the paper [21]. The Figure 7-10 shows the circuit schematic of the macromodel description. Transistor M1 is an NMOS transistor. The Fowler-Nordheim tunneling current is modeled as a voltage-controlled current source GFN. The value of GFN is given by the expression shown in Equation 7-1 [21].

$$\mathrm{GFN} = \mathrm{A}_{\mathrm{tun}}\alpha\left|\frac{V_{fg}-V_t}{t_{tun}}\right|\frac{(V_{fg}-V_t)}{t_{tun}}.\exp(-\beta\left|\frac{t_{tun}}{V_{fg}-V_t}\right|)$$

Equation 7-1: Fowler-Nordheim Tunneling Current

The current source IFN, resistor RT and capacitor CT are used to model the charge storage and retention capabilities of the EEPROM [21]. The IFN is a current-controlled current source which senses the tunneling current GFN with a gain of unity. RT is chosen to be a very high value, so that most of the current IFN is supplied to the capacitor CT [21].

The floating gate is modeled by the controlled source EFG. The value of the floating gate voltage is given by Equation 7-2 [21].

$$V_{fg} = \alpha_d V_d + \alpha_s V_s + \alpha_{cg} V_{cg} + \alpha_b V_b + \alpha_{tun} V_{tun} + \frac{Q_{fg,initial}}{C_t} + \frac{\Delta Q_{fg}}{C_t}$$

Equation 7-2: Expression for Floating Gate Voltage

The remaining part of the circuit is used to model Φsi, the surface potential of silicon under the tunnel oxide. The original description of the SPICE code as shown in the paper has been translated to HSPICE for use in HSPICE simulations. The HSPICE description of the macromodel is shown in Figure 7-11.

The low-voltage and high voltage device models are based on the description given in [22]. The high voltage devices had an effective length of 1um and a tox (oxide thickness) of 275 Angstrom [22]. The low voltage devices had an effective length of 0.35um and a thickness of 75 Angstrom [22].

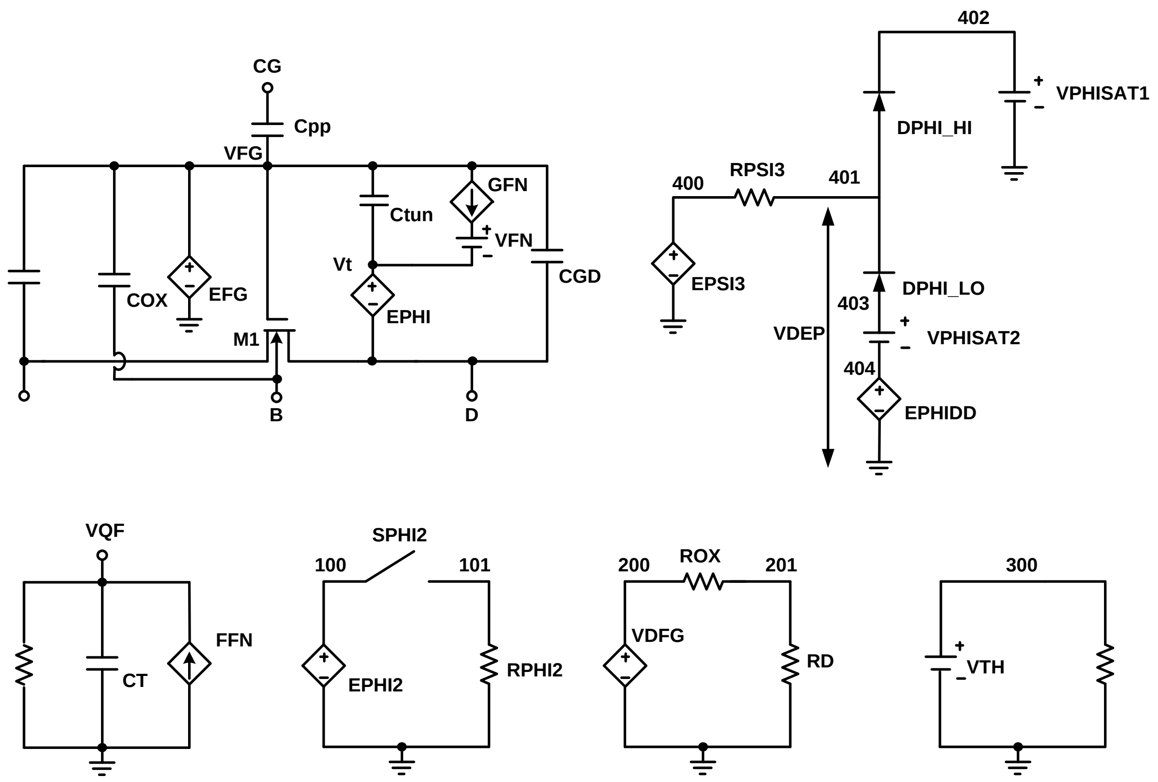


Figure 7-10: FLOTOX EEPROM Macromodel Schematic

Figure 7-12 shows the simulation results of the macromodel. The figure shows the control gate voltage during an erase operation. The figure shows the drain voltage during a write operation, the Fowler-Nordheim tunneling current and the charge on the floating gate during the erase and write operations.

```
.SUBCKT EEPROM_CELL_G2 B CG D QF S
+VDDTHR=10.5 VPHISAT=-1.12 CPP=80.535F
+CGS=7.386F COX=11.4F Ctun=4F CGD=12.34F CTOTAL=115.661F
+A_G=.6963 A_D=.1067 A_t=.0346 A_S=.0639 A_B=.0986
+TTUN=100E-10 ATUN=1.16E-12 A=3.69E-7 B=223E6
+VINITIAL=0 VTO=2.4 W=5.5U L=3U

CPP CG FG 'CPP'
CGS FG S 'CGS'
COX FG B 'COX'
Ctun FG 2 'Ctun'
EPHI 2 D 101 0 1
CGD FG D 'CGD'
EFG FG 0 VOL ='A_G*V(CG)+ A_D*V(D)+ A_S*V(S) +A_t*(+V(2))+
+A_B*V(B) +V(QF) +VINITIAL'
GFN FG 1 VOL = 'ATUN*A*ABS((V(FG)-V(2))/TTUN )*((V(FG)-
+V(2)) /TTUN)*EXP(-B*100*ABS(TTUN/(V(FG)-V(2 ))))'
VFN 1 2 DC 0
M1 D FG S B NTYPE

.MODEL NTYPE NMOS VTO='VTO' W='W' L='L'
FFN QF 0 VFN 1
CT QF 0 '(CTOTAL)' IC=0
RCT QF 0 1E20
EPSI_2 100 0 401 0 1
GSPHI_2 100 101 VCR  FG D  -1MV,1 1M,.1MEG
RPSI_2 101 0 100
EFGD 200 0 D FG 1
ROX 200 201 1K
RD 201 0 5K
VTH 300 0 'VDDTHR'
RTH 300 0 1000
EPSI_3 400 0 201 300 1k
RPSI_3 400 401 10K
DPHI_HI 401 402 DMOD
DPHI_LO 403 401 DMOD
VPHISAT1 402 0 'VPHISAT-.7'
VPHISAT2 403 404 'VPHISAT +0.7'

.MODEL DMOD D (CJO=10P N=0.75 IS=1e-10)
EPHI_DD 404 0 VALUE='-ABS(V(201))'
.ENDS EEPROM_CELL_G2
```

Figure 7-11: HSPICE Description of the FLOTOX EEPROM Macromodel

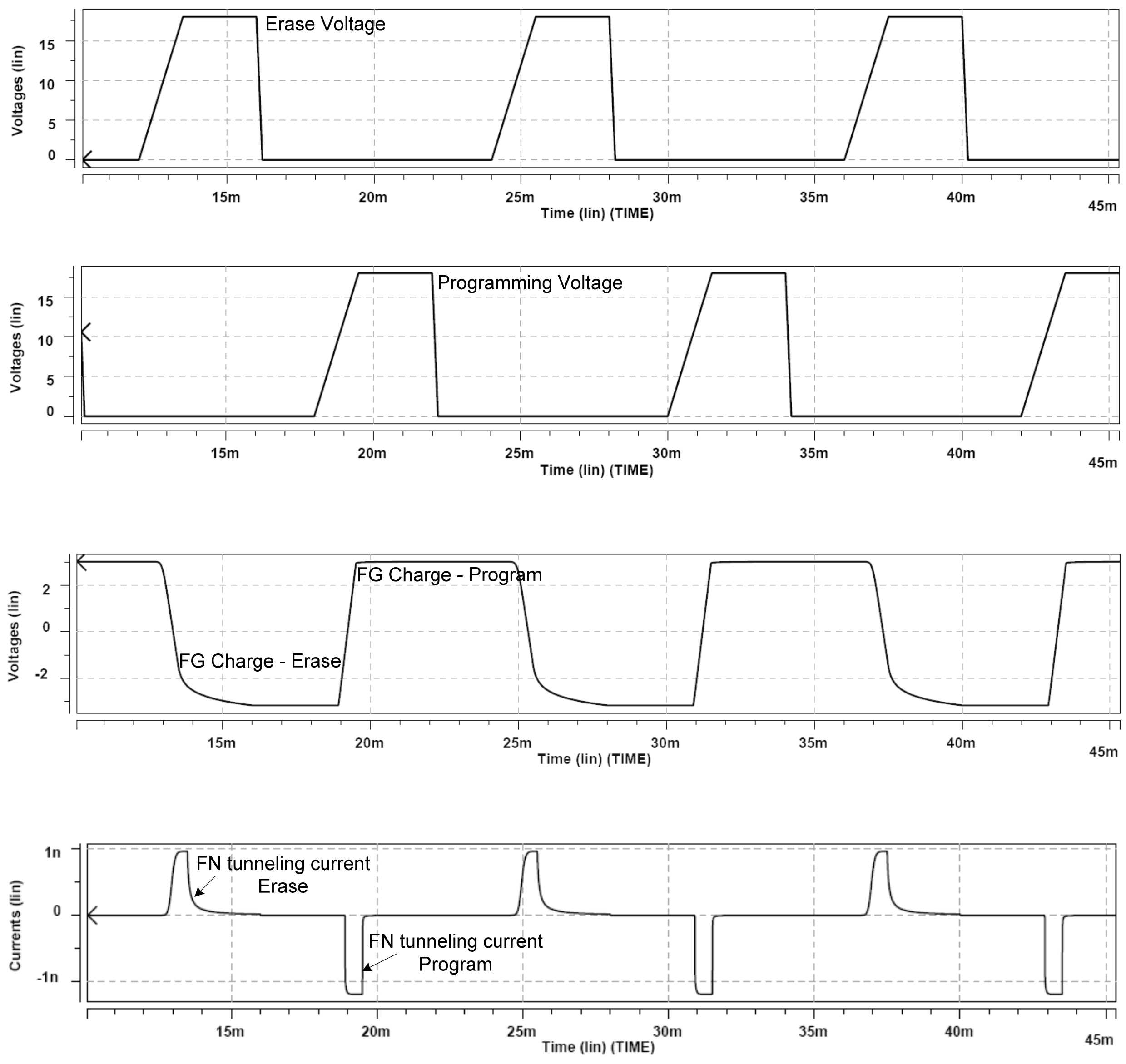


Figure 7-12: FLOTOX EEPROM Cell HSPICE Simulation

7.2.1 Ramp Generator

The ramp generator has been implemented based on the description given in patent number US 6,650,153 B2 [23]. The schematic of the ramp generator is shown in Figure 7-13. The ramp generator design consists of a differential pair powered by the voltage Vpp, generated using a charge pump. The differential pair acts as an OPAMP with a high gain which forces the inputs IN+ and IN- to be virtually at the same potential. The IN+ terminal of the OPAMP is connected to Vref. Vref used in the design is 1.33V (bandgap reference voltage). The IN- terminal of the OPAMP is connected to the feedback node. Feedback ensures that the voltage at the feedback node stays close to 1.33V. The presence of a constant current source forces a constant current through the ramp capacitor C.

The current through the capacitor and the voltage across the capacitor are related as shown in Equation 7-3. The current source is constant and so the voltage across the capacitor ramps with the slope of I/C as shown in Equation 7-4. The slope can be varied by varying C or I.

$$I = C\frac{dV}{dT}$$

Equation 7-3: Capacitor Current Equation

$$V = \int \frac{I\,dt}{C} = \frac{It}{C}$$

Equation 7-4: Capacitor Voltage Ramp

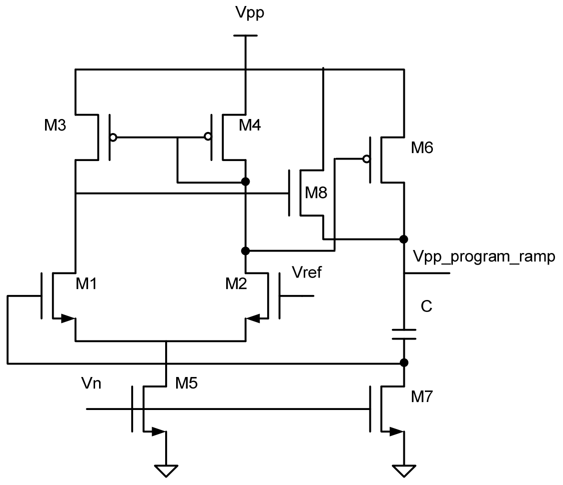


Figure 7-13: Ramp Generator Schematic

7.2.2 High Voltage Generation Using Charge Pump

The high voltage Vpp required for erase and programming operations is typically generated using a charge pump. The Dickson II Charge Pump circuit is one of the common charge pump circuits used for this purpose [24].

7.2.3 Word Line Level Shifter

The level shifter circuit shown in Figure 7-14 has been designed based on [25]. Asserting the input, sets inp1 signal high and inp2 signal low. This action pulls node X low and Y to Vpp. Pulling node X low, enables the pass transistor M5 which drives the high voltage at the output. On the other hand, de-asserting the input, pulls node Y low and X to Vpp which sets the gate voltage of the pass transistor M5 at Vpp. This effectively turns off the pass transistor and the ouput is tri-stated.

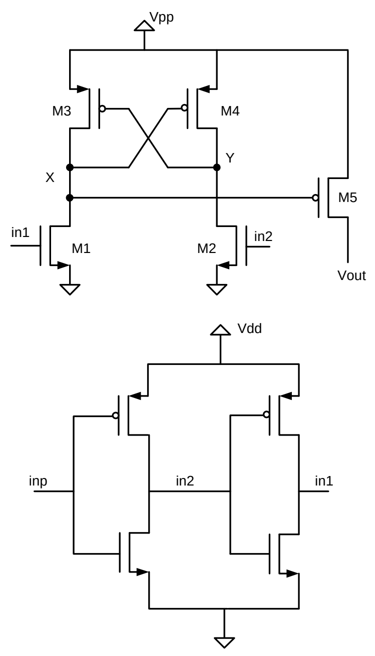


Figure 7-14: Schematic of Voltage Level Shifter

7.2.4 Column Latch for Bitlines

The column latch is a circuit within the EEPROM memory array to store the data that needs to be written into the column of the memory array. The FLOTOX transistor that gets activated in the selected row gets programmed through the new data value available on the column bit lines.

The design of the column latch to drive the drain of the FLOTOX transistor with the programming voltage is based on the design described in US Patent US6,859,391[26] with minor modifications to the design. Data is written into the EEPROM cell through a three step process, consisting of loading the data, followed by erasing the data and finally programming the data. To load the data into the column latch, the LOAD signal as shown in

Figure 7-15 is asserted for a short duration of time. When LOAD gets asserted, the new data value that needs to be written into the EEPROM cell gets loaded into the latch and the value is stored at node X by the cross coupled inverter even after LOAD gets de-asserted. Under normal conditions, DIS_BL_CTRL gets asserted and signal IN2 is asserted and signal IN1 is de-asserted pulling the output of the level shifter to Vpp. This effectively turns off the pass transistor connected to the bit line. When the data needs to be programmed into the EEPROM cell, DIS_BL_CTRL gets de-asserted and DATA_CTRL is asserted. This loads the data into nodes IN2 and IN1. Depending on the value of IN2 and IN1, the level shifter either discharges or charges to enable the pass transistor as shown in Figure 7-16. The enabling of the pass transistor connects the programming voltage to the bit line and the disabling of the pass transistor tri-states the bitline. For an EEPROM cell to be programmed the pass transistor is enabled and the bit line is connected to the programming ramp generated by the ramp generator as described in the section 7.2.2. The DIS_BL, shown in Figure 7-16 is asserted to pull the BL node to ground during an erase operation. The DIS_BL can also be asserted during a read operation to discharge any unwanted voltages that get coupled to the bit line.

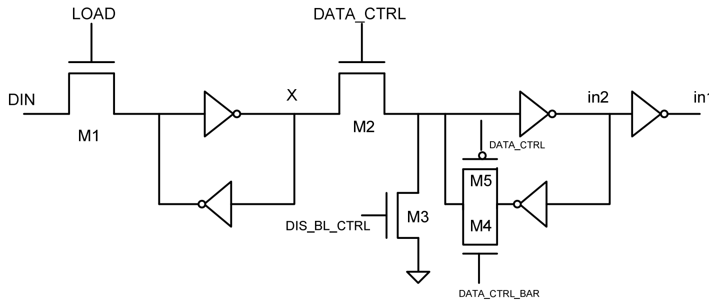


Figure 7-15: Data Latch Schematic of Column Latch

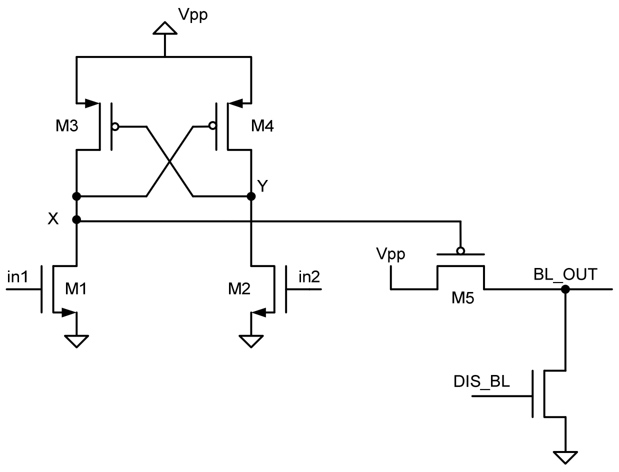


Figure 7-16: Level Shifter and Pass Transistor for Column Latch

7.2.5 Power Multiplexer

The design of the power multiplexer is based on the patent US 7,005,911 [27]. As the EEPROM erase and program operations are high voltage operations, and the read being a low voltage operation, it is necessary to have a power multiplexer which could choose between the high and low voltages depending on the operation performed.

Such circuits are essential at the input to the word-line level shifter. The level shifter would receive a high voltage from the power multiplexer for an erase and program operation and a low voltage for a read operation.

Power multiplexers are also used to connect to the column gates, which output a high voltage during the erase operation, a ground voltage during program operation and a low voltage during a read operation. Figure 7-17 shows the schematic of the power multiplexer used in the design. The LS1 and Vdd_switch as shown in Figure 7-17 are essentially level shifters used to shift the output to Vpp or Vdd depending on the enable input.

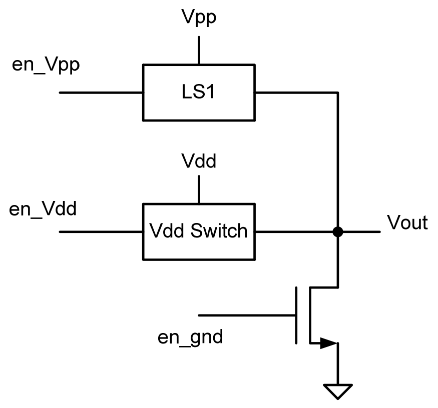


Figure 7-17: Power Multiplexer Schematic

7.2.6 Sense Amplifier

The sense amplifier is the circuit that senses the voltage at the output of the EEPROM cell during a read operation. The design of the sense amplifier is based on the voltage sensing scheme as proposed by [24]. Under erase or program conditions, signal C1 is asserted pulling the source of the FLOTOX transistor to ground. C2 is asserted under normal conditions. During a read operation, C1 gets de-asserted rising the voltage at node "ee_cs" to the supply Vdd. The control voltage is set to the supply voltage Vdd. The wordline is enabled, effectively turning on the select transistor and the control gate transistor. If the FLOTOX transistor is programmed, the voltage at the gate terminal of the FLOTOX transistor, causes the transistor to conduct, raising the voltage of the bit line from 0V to Vx. The value of Vx dpends on the wordline voltage. If the voltage at the wordline is high because of use the of wordline boosters, the select transistor would not incur a Vth drop across it. Otherwise the voltage at the bit line would be reduced by the threshold voltage drop across the select transistor.

The inverter pair connected to the bit line senses the bit line voltage to produce a logic high or a logic low. Figure 7-19 shows the sense amplifier reading a logic ‘1’ when the FLOTOX transistor has been programmed. If the FLOTOX transistor had not been programmed, then the transistor would not conduct and the bit line voltage would not rise. Figure 7-20 shows the sense amplifier reading a logic ‘0’ when the FLOTOX transistor has not been programmed.

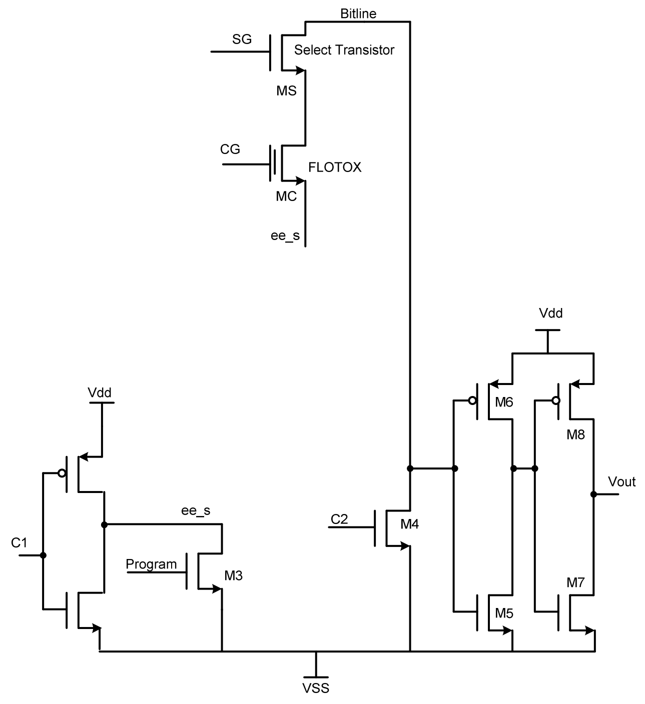


Figure 7-18: Sense Amplifier Schematic

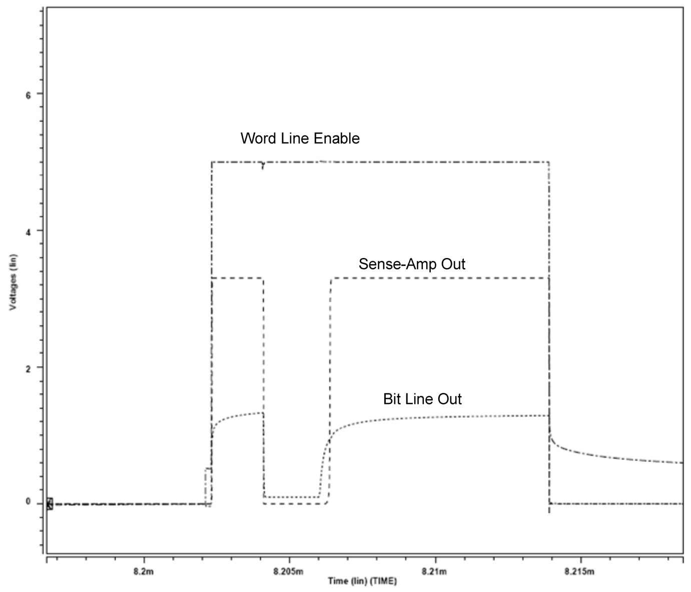


Figure 7-19: Sense Amplifier Reading a Logic ‘1’

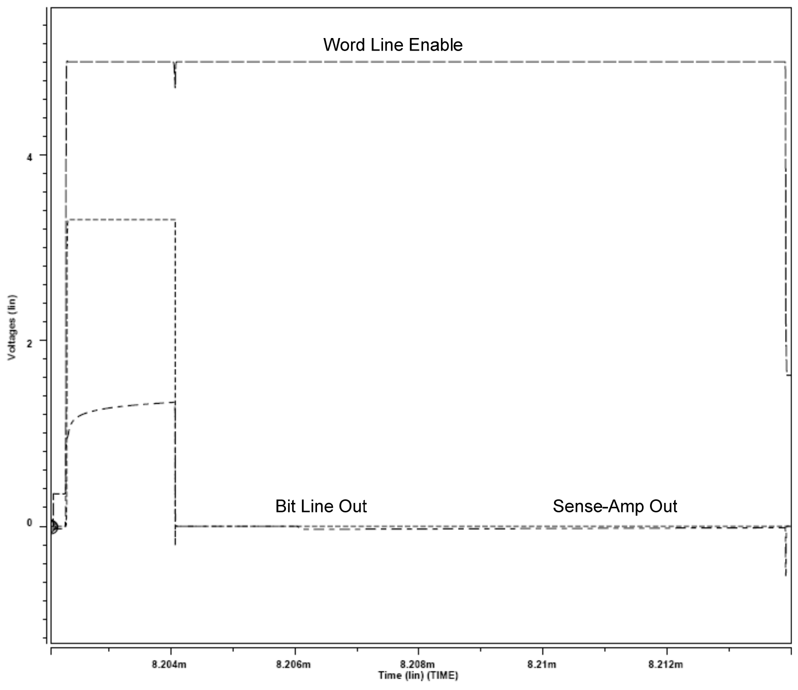


Figure 7-20: Sense Amplifier Reading a Logic ‘0’

7.2.7 Memory Bank Architecture

Figure 7-21 shows the schematic of the memory bank architecture that has been implemented. The implementation has two power multiplexer configurations, *wordline power multiplexer* and *control gate power multiplexer*. The *wordline power multiplexer* is used to multiplex between the high voltage (Vpp), boosted voltage (Vdd_boost) and ground (gnd). This is because the wordline output is Vpp during erase and program operations, Vdd_boost during a read operation and gnd when the memory is disabled. The wordline is asserted for the appropriate row that gets selected from the address decoding action of the row address decoder. The level shifter in the wordline path asserts the appropriate voltages only to the wordline that is activated. All other wordlines are disconnected and are not enabled.

The *control gate power multiplexer* is used to multiplex Vpp, Vdd and gnd voltages to the control gate of the EEPROM cells. The multiplexer outputs Vpp when the device needs to be erased, Vdd during a read operation and gnd when the device needs to be programmed or when the device is disabled. The Vpp to the control gate multiplexer is generated by the erase ramp generator circuit to meet the appropriate ramp conditions.

The column data latch is used to latch the data value that needs to be written into the memory. Depending on the latched value, the column latch outputs either a Vpp or a tri-state output. The Vpp to the column latch is generated by the program ramp generator. The column select lines “col0,”, “col1” are used to select the appropriate column during a read operation. The column select lines are activated by a column address decoder. The sense amplifier finally reads the voltage on the bit lines and decides the logic value of the voltage that is being read out. The “bl_s” generator circuit is used to generate the precharge voltages that are necessary to precharge the sources of the EEPROM cells.

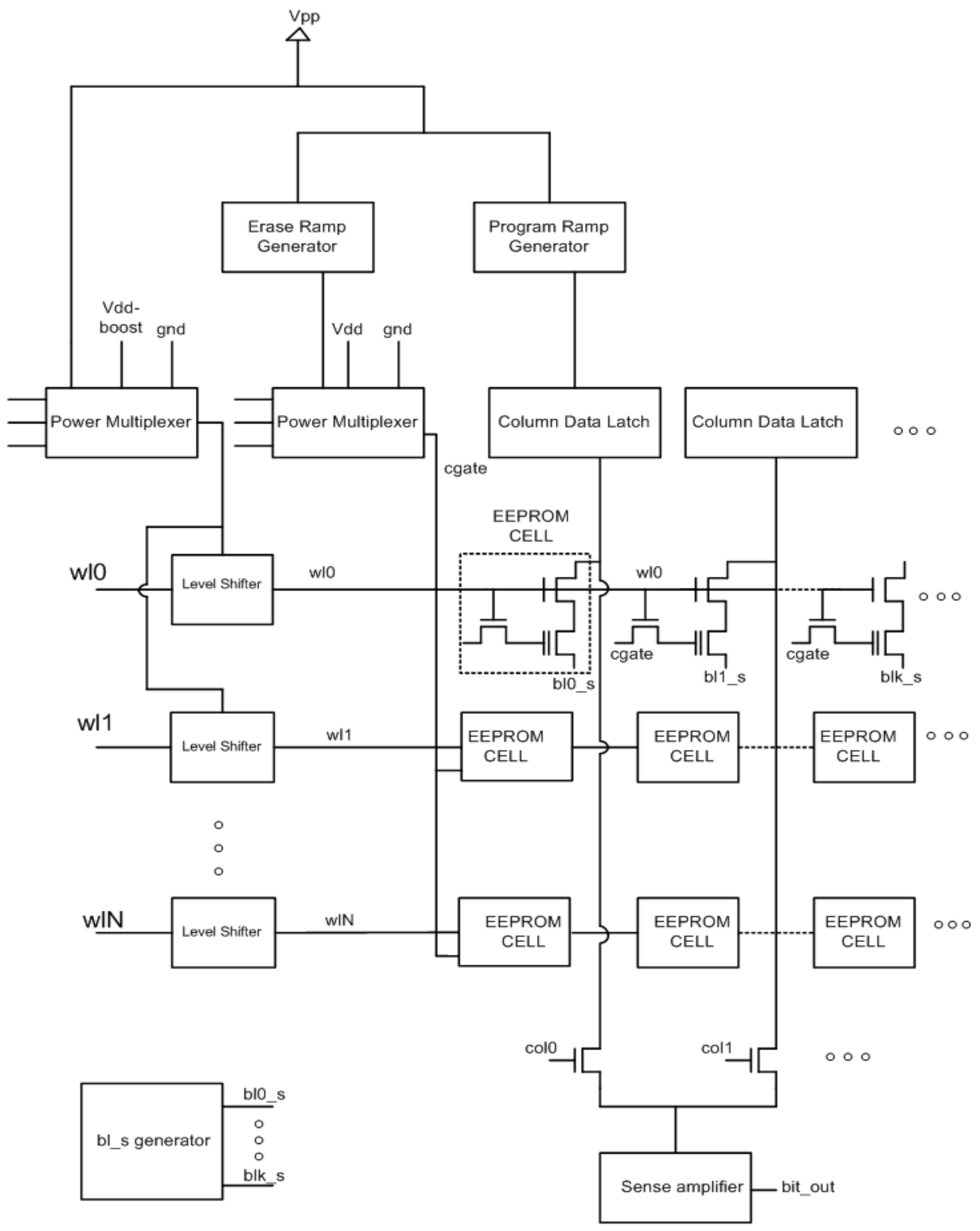


Figure 7-21: Memory Bank Architecture

7.2.8 Memory Bank Simulation

Figure 7-22 shows the erase and programming of two bits in the memory bank. The first bit gets erased and is not programmed while the second bit gets erased and is programmed. Figure 7-22 shows the floating gate voltage of the EEPROM cell that's has been programmed and that been only erased.

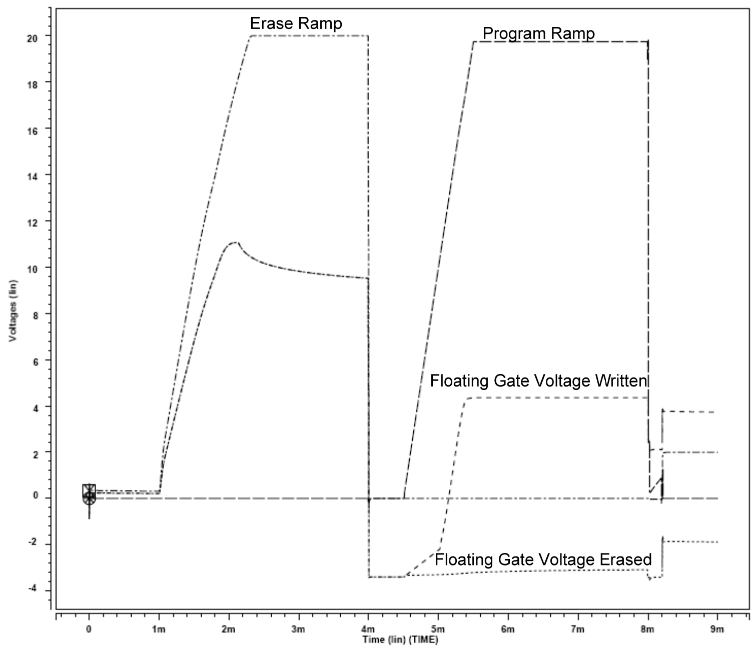


Figure 7-22: EEPROM Bank Simulation

8.0 POWER GATED EEPROM DESIGN

High power consumption has always been an issue with EEPROM design because of the inherent nature of the FLOTOX memory processes that use a high voltage to program these devices. Many circuit level optimizations have been made in the past to reduce the power consumption of the basic EEPROM block. The thesis proposes an architectural level power optimization using power gating technique to increase the size of the memory while keeping the active power low. The power-gated memory design minimizes both the dynamic and leakage power associated with memories. Simulations have been performed using HSPICE models for a 0.35um CMOS process.

Our solution is to build a hierarchical low-power memory block using power-gating technique for the non-accessed memory block. The concept of multi-block architecture [28] is a common low power technique used in memory design. However, the use of power gating reduces the power consumption significantly as compared to other sleep mode techniques.

8.1 ARCHITECTURE OF THE POWER GATED MEMORY

A ``power enable" PMOS device is added in series with the memory block as shown in Figure 8-1. This allows only the block that is actually addressed to be powered, while the remaining blocks remain disconnected from power. As the EEPROM uses both a high voltage

(Vpp) and a normal voltage (Vdd), the “power enable” voltage depends on the power supply that the device is connected to. For normal voltage sections, the “power enable” signal voltage is normal (Vdd) to disable the memory block. For high voltage sections, the “power enable” signal voltage is high (Vpp) to disable the memory block. To select the bank that needs to be activated, an address decoder is required that decodes the address and asserts the corresponding power enable signal. As the address decoder output is a low voltage output, a level shifter is required to translate the low voltage to a high voltage for the high power sections of the memory block. An overview of this architecture is shown in Figure 8-2 N is the number of memory blocks and M is the number of inputs to the decoder. Typically, the higher order address bits of the memory are connected as inputs to the block decoder.

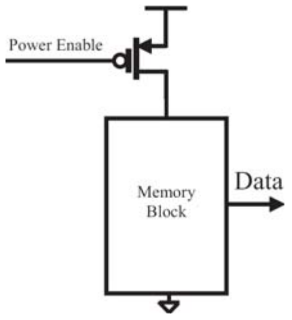


Figure 8-1: Memory Block with power gate

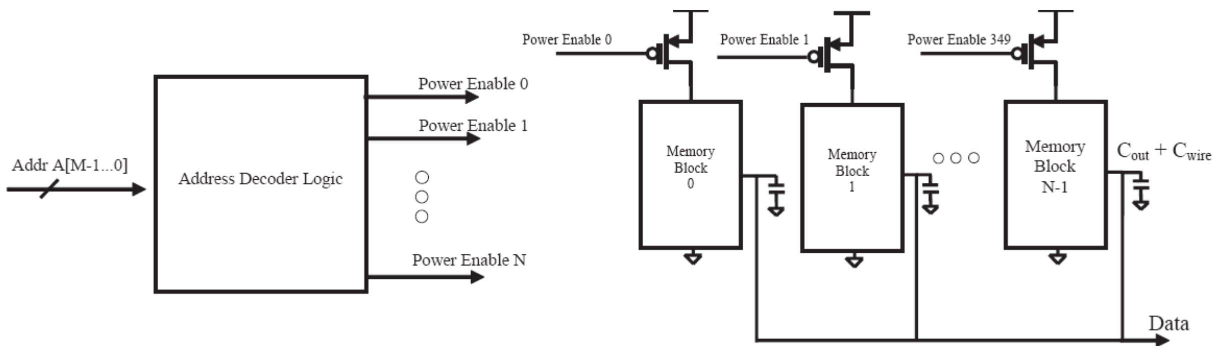


Figure 8-2: Power Gated Memory Design

8.1.1 Memory Block with Power Gate

The primary benefit of the ``power gated" implementation is that both the static and dynamic power can be eliminated from all but the active memory block. The addition of a series power enable PMOS device does not affect the average power consumed by the device, but reduces the peak power consumed by the device. This technique has been previously applied to I/O buffers to reduce the SSN (simultaneous switching noise) produced on the supply lines when the output buffers switch [29]. The design implication with the reduction of peak power consumed by the device is the on-chip regulator’s efficiency can be improved. As EEPROMS typically operate at a low frequency, the addition of the PMOS device does not have an adverse impact on the speed of the memory. However, the width of the power enable PMOS device should be large enough to supply enough current during the normal operation of the memory.

8.1.2 Dynamic Decoder

The address decoder to select the memory bank that needs to be enabled can be constructed using static or dynamic CMOS circuits. Our design uses a dynamic CMOS gate with a pre-charge transistor acting as the pull up network [6]. The dynamic decoder is a low-power alternative [30]. The pre-charge signal is driven by a periodic signal related to the clock. In our case we can use the system clock directly as the pre-charge signal. For a rising edge triggered memory, when pre-charge is `1' the pre-charge PMOS transistor is off and the pre-charge NMOS transistor is on, allowing the pull down network or evaluation network to pull the appropriate power enable line to ground, thus turning on the associated memory block. On the back half of the cycle, pre-charge is '0' turning on the pre-charge PMOS and disconnecting the power enable line from ground, thus turning off all the memory blocks.

8.2 MEMORY BLOCK POWER CONSUMPTION

Many advanced low power circuit design techniques for Embedded EEPROMS have been reported [24] [25]. The EEPROM power consumption can be categorized as 1) read power consumption and 2) write power consumption.

The read power consumed during a read operation includes the power required to precharge the source terminal of the FLOTOX transistor, the power consumed by the sense amplifier circuitry, and the row and column decoder power. The read operation is typically a low voltage operation mode except for the word line boosting and bit line boosting schemes are used to decrease the read time.

The write power consumption dominates the read power consumption because of the high voltage used to program the device. The major power consuming component during a write operation is the high voltage charge pump which generates the voltage required to erase and program the devices [24]. Many circuit level optimizations have been proposed by [24] to keep the charge pump power low during the write operation. Apart from the charge pump, power is consumed in charging and discharging the word lines and the control gate line to Vpp during the erase operation and in charging the bit lines to Vpp during a write operation, where Vpp is the high voltage supplied by the charge pump.

During a write operation, the choice of the programming voltage is dominated by the EEPROM process rather than by the designer. Hence, to keep the dynamic power low it becomes necessary to keep the charging capacitance low at a fixed operating frequency. With the increase in size of the memory from 128 bytes to 256 bytes, for example, an architectural decision to increase the number of rows or the number of columns is required. Increasing the number of columns increases the page width, which in turn necessitates an increase in the current required to program the device. This design choice mandates an increase in the size of the charge pump and hence a proportionate increase in power. On the other hand, an increase in the number of rows, increases the bit line capacitance that needs to be charged during a write operation. Hence, scaling the memory while keeping the dynamic power low is a design challenge. Hence, a power gated memory solution enables to expand the memory size with minimal penalty because the expanded memory block is disabled and is enabled only when required.

8.3 POWER-ON RESET

One problem with the power-gating technique occurs in particular when the supply voltage is being ramped up to Vdd. Initially, the power enable lines are low because of the power up delay of the decoder. This effectively allows each of the memory blocks to power up as if they were not power gated, which causes a tremendous amount of power consumption during power up. This is particularly problematic for EEPROMs in passive RFID (Radio Frequency Identification Devices) tags because the power up time for Vdd from RF energy harvesting is long. Figure 8-3 shows the assumptions for our power on condition of the power supply and the power enable input lines. The power supply charges to Vdd linearly in time t_P (100us) and the power enable lines are delayed by a time t_D (40us) where $t_D << t_P$.

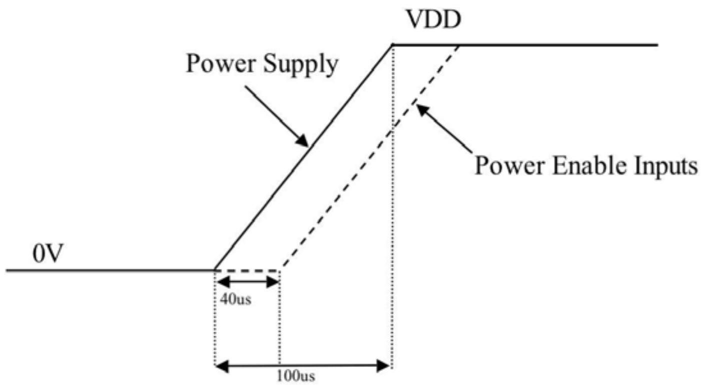


Figure 8-3: Simulation of Power on condition

To solve this problem a ``Power-on Reset'' PMOS transistor is added to each of the decoder lines to rise all the power enable lines high and effectively block any of the memory blocks from charging until V_{dd} has fully powered up. This is shown for the power enable 0 line in Figure 8-4 below.

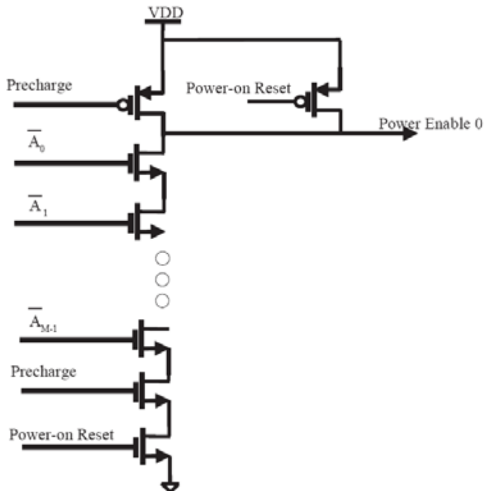


Figure 8-4: Dynamic Decoder with power-on reset

The Power-on Reset transistor is enabled with the "Power-on-Reset" signal held low until the power on reset circuit shuts off the transistor by setting "Power-on-Reset" high. This is assumed to happen well after the power supply is fully charged as shown in Figure 8-5 where V_{dd} is activated well after the power supply has powered up. This circuit causes all the power enable lines to rise at approximately the same speed as the power supply. Thus, the power enable transistor from Figure 8-1 has a $V_{gs} \approx 0$ during power up and does not turn on.

As POR (power-on-reset) circuits are a part of any digital system, the use of POR circuit does not cost additional power during the normal operation of the circuit. Ramos et.al. [31] gives an example of a low power power-on-reset circuit.

Figure 8-6 shows the modified charge capacitor circuit as proposed by [31] which uses an injection capacitor C_{inj}. When Vdd ramps up, the input to the power-on-reset inverter becomes high because of the capacitive coupling through C_{inj}. Hence the power-on-reset output is held low. After a certain amount of delay (power-on-reset delay), the C_{timing} capacitor gets charged through the PMOS pull-up path, which pulls the node connected to the C_{inj} to ground. This causes the power-on-reset ouput to reach Vdd. The delay is primarily controlled by the PMOS pull-up path and the value of the C_{timing} capacitor.

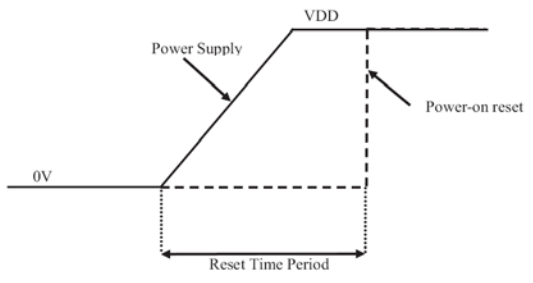


Figure 8-5: Power-on reset timing diagram

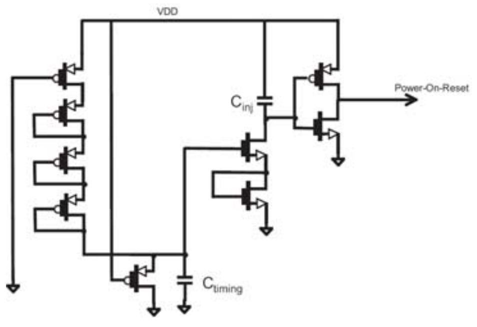


Figure 8-6: Power-on reset circuit

8.4 RESULTS

For our fundamental memory building block, we used the model of a 64 byte EEPROM memory built in a 0.35 um CMOS process operating at 5.0V. Table 8-1 shows the power-gated memory power simulation results. The table compares the power consumed by the active memory bank and the power-gated memory banks in a power-gated design. The address decoder overhead is also shown in the table. The total read power overhead is 298.12nW for a 2-bank memory, 390.71nW for a 4-bank memory and 570.7nW for an 8-bank memory. The total write power overhead is 143.55nW for a 2-bank memory, 203.69nW for a 4-bank memory and 339.67nW for an 8-bank memory.

Table 8-1: Power Gated Memory Simulation Results

	2 Bank (Bank Size 64 Bytes)			4 Bank (Bank Size 64 Bytes)			8 Bank (Bank Size 64 bytes)		
	Active Bank	Idle Bank (nW)	Decoder Overhead (nW)	Active Bank	Idle Banks (nW)	Decoder Overhead (nW)	Active Bank	Idle Banks (nW)	Decoder Overhead (nW)
Read Power	700.68uW	9.77	288.35	700.68uW	29.30	361.41	700.68uW	68.37	502.33
Erase Power	12.51uW	4.03	135.53	12.51uW	12.09	179.62	12.51uW	28.22	283.49
Write Power	10.08mW	3.99		10.08mW	11.98		10.08mW	27.96	

To study the impact of the power consumed during the power-on condition, an inverter with an approximated load capacitance C_L of 4.44 pF was used. Any device that allows switching could be used to model the dynamic power of the device; however, the inverter was selected due to its simplified modeling. To this device the power enable PMOS device was added in series. The circuit diagram is shown in Figure 8-7.

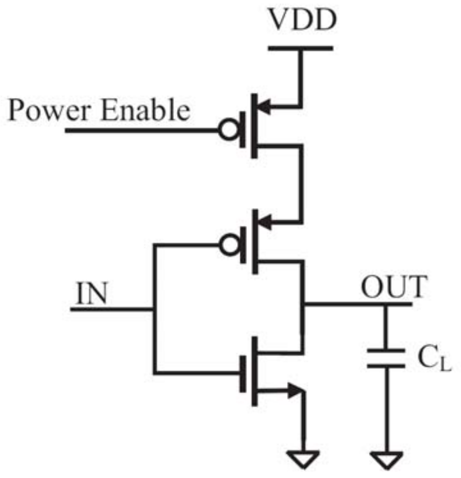


Figure 8-7: Inverter with power gate.

The power-up time and in particular the delay has a significant impact on the power consumed by the memory. For example, with a power up time, t_P = 100 us and an ideal ramp up of the power enable inputs, t_D = 0, the average power consumption is approximately 25.5 nW.

However, as the power enable delay increases linearly, the power consumption increases exponentially to reach 25.4 uW for t_D = 40 us based on the simulations as indicated in Figure 8-3. t_D was varied between 26us and 40us to study the peak currents in more detail. The results are shown in Figure 8-8. For delays exceeding 30 us, there are big spikes initially in supply current because the power enable transistor is turned on causing the internal memory block node capacitances to draw current from the supply. With delays lower than 30 us the power enable transistors do not turn on very much and draw a much lower current.

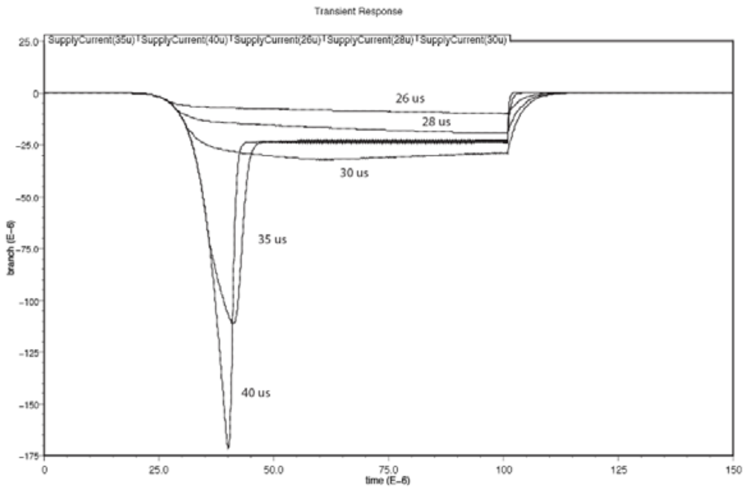


Figure 8-8: Power Enable Ramp-up simulation

9.0 CONCLUSION

For this thesis, I have completed the placement and routing of the SuperCISC reconfigurable hardware fabric including the ALU, MUX and the FINALMUX Stripe in the IBM 0.13um CMOS technology using Cadence SoC Encounter. The post place and route SDF and parasitic SPEF file have been generated from the design to be annotated in the power analysis flow using Prime Power. The post parasitic annotated power analysis numbers have been generated for a variety of benchmarks.

A CMOS thyristor based delay element with a programmable feature has been implemented in the IBM 0.13um CMOS process. The improved delay element consumes very little on-state and sub-threshold leakage power. The design has been characterized for use as a standard cell in an ASIC design flow as well as in the SuperCISC design flow to minimize the glitching power consumption.

An EEPROM design using a 0.35um, 20V technology has been implemented to show that power gating can be used to increase the memory size with a minimum power overhead for use in the SuperCISC architecture. The macromodel of the FLOTOX transistor has been implemented in HSPICE. The design shows that the power gated memory block consumes very little static power and hence can be used to increase the size of the memory.

APPENDIX A

PHYSICAL DESIGN CAD TOOL FORMATS

As place and route is an automated CAD tool process, there are some significant industry standard file formats and standards. A basic understanding of these file formats is essential to work with the state-of-the-art CAD tools.

The important file formats are

(i) LEF (Library Exchange Format)

(ii) DEF (Design Exchange Format)

(iii) Cadence Timing Library Format (*.tlf)

(iv) Synopsys Liberty Format (*.lib)

(v) Standard Delay Format (SDF)

(vi) Standard Parasitic Extraction Format (SPEF)

Library Exchange Format (LEF)

LEF is an ASCII data format standard from Cadence and is used to define the elements of the technology as well the standard cells being used for the design. From a technology perspective, LEF contains descriptions on the various routing layers, design rules for routing, via definitions, metal layer resistance and capacitance. For defining the standard cell library, LEF contains the abstract layout of the cells, cell dimensions, layout of pins and pin capacitances. However, LEF does not contain information about the internal netlist of the cells [32].

Design Exchange Format (DEF)

DEF is also an ASCII data format from Cadence for representing the physical layout. The DEF data represents the netlist and the circuit layout information. DEF is used in conjunction with LEF to represent the complete physical layout of the integrated circuit being designed [33]

Cadence Timing Library Format (TLF)

TLF is an ASCII representation of the timing and power parameters associated with any standard cell in a particular technology [34]. The TLF file contains timing models and data to calculate I/O path delays, timing check values and interconnect delays [34]. A TLF file is used as an input to the Cadence tools.

Synopsys Liberty File Format (.lib)

Synopsys Liberty Format is an industry standard way to represent information about the standard cell library for a particular technology. The .lib file contains descriptions of the standard cell's input and output pins, logical function of every cell, timing information specific to cell delays, rise and fall transitions times at the outputs, energy information specific to the energy consumed when the input makes a rise transition or a fall transition and cell's leakage power. In addition to that, the file also contains environmental statistics such as certain process information, operating temperature and supply voltage variations [35].

Standard Delay Format (SDF)

Standard Delay Format (SDF) files store timing information generated by the EDA tools for a particular design. The data in the SDF is represented in a tool-independent way and can describe module path delays, device delays, interconnect and port delays. It can also include timing constraints such as set-up, hold and skew. The SDF file can be used at various stages in the design process. An SDF generated after synthesis represents the pre-payout timing information in the design while an SDF generated after placement and routing contains post-layout timing information.

The SDF file is typically generated by a timing calculator based on the design netlist and pre or post layout information. The SDF data that is generated is used by the annotator which uses the SDF information for performing various timing and power analysis [36].

Standard Parasitic Extraction Format (SPEF)

The SPEF is an IEEE Standard for representing the parasitic information in the design in an ASCII format. The SPEF specification is a part of the 1481-1999 IEEE Standard for Integrated Circuit(IC) Power and Delay Calculation System standard [37]. The SPEF contains parasitic resistance, capacitance and inductance of wires on the chip. The SPEF information can be used by the delay calculation engine to generate timing data or can be used by an analysis tool to estimate power after the design has been annotated with the parasitic information.

APPENDIX B

HSPICE POST PROCESSOR SCRIPT

The Perl (Practical Extraction and Report Language) script that was used automate the delay element standard cell characterization is shown below.

```
#!/usr/bin/perl -w


#Enter the name of the spice netliost that needs to be processed
#Outputs two files:measurement.out contains the raw current measurement output
#energy.out contains the actual energy measurements


if(!$ARGV[0]) {

  print "Usage  Enter the spice netlist to be processed>\n";
  exit(1);

}

#this is based on CLKBUFX12TS template
@loadarray = ("9.48f","24.648f","56.88f","121.344f","250.272f","506.232f","1023.84f");
@trarray = ("0.028e-9","0.044e-9","0.076e-9","0.138e-9","0.264e-9","0.516e-9","1.02e-9");
#@loadarray = ("121.344f");
#@trarray = ("1.02e-9");


open IN, $ARGV[0];
$i=0;
```

```
while(<>)
  {
  $original_spice_array[$i] = $_;
  $modified_spice_array[$i] = $_;
  $i++;
 }

close (IN);

 open(MEASHANDLE,">measurement.out");
 open(ENERGYHANDLE,">energy.out");
 print MEASHANDLE "CAPACITIVE_LOAD TRANSTION_TIME RISE_POWER1 RISE_POWER2
RISE_POWER3 RISE_POWER4 RISE_POWER5 RISE_POWER6 RISE_POWER7
RISE_POWER8 RISE_POWER9 RISE_POWER10 RISE_POWER11 RISE_POWER12
FALL_POWER1 FALL_POWER2 FALL_POWER3 FALL_POWER4 FALL_POWER5 FALL_POWER6
FALL_POWER7 FALL_POWER8 FALL_POWER9 FALL_POWER10 FALL_POWER11 FALL_POWER12
RISE_TR1 RISE_TR2 RISE_TR3 RISE_TR4 RISE_TR5 RISE_TR6 RISE_TR7 RISE_TR8 RISE_TR9
RISE_TR10 RISE_TR11 RISE_TR12 FALL_TR1 FALL_TR2 FALL_TR3 FALL_TR4 FALL_TR5 FALL_TR6
FALL_TR7 FALL_TR8 FALL_TR9 FALL_TR10 FALL_TR11 FALL_TR12 CELL_RISE_DELAY1
CELL_RISE_DELAY2 CELL_RISE_DELAY3 CELL_RISE_DELAY4 CELL_RISE_DELAY5
CELL_RISE_DELAY6 CELL_RISE_DELAY7 CELL_RISE_DELAY8 CELL_RISE_DELAY9
CELL_RISE_DELAY10 CELL_RISE_DELAY11 CELL_RISE_DELAY12 CELL_FALL_DELAY_1
CELL_FALL_DELAY2 CELL_FALL_DELAY3 CELL_FALL_DELAY4 CELL_FALL_DELAY5
CELL_FALL_DELAY6 CELL_FALL_DELAY7 CELL_FALL_DELAY8 CELL_FALL_DELAY9
CELL_FALL_DELAY10 CELL_FALL_DELAY11 CELL_FALL_DELAY12 ILEAK_HI1 ILEAK_HI2
ILEAK_HI3 ILEAK_HI4 ILEAK_HI5 ILEAK_HI6 ILEAK_HI7 ILEAK_HI8 ILEAK_HI9 ILEAK_HI10
ILEAK_HI11 ILEAK_HI12 ILEAK_LO1 ILEAK_LO2 ILEAK_LO3 LEAK_LO4 ILEAK_LO5 ILEAK_LO6
ILEAK_LO7 ILEAK_LO8 ILEAK_LO9 ILEAK_LO10 ILEAK_LO11 ILEAK_LO12
REFERENCE_CURRENT\n";

 print ENERGYHANDLE "CAPACITIVE_LOAD TRANSTION_TIME RISE_POWER1 RISE_POWER2
RISE_POWER3 RISE_POWER4 RISE_POWER5 RISE_POWER6 RISE_POWER7
RISE_POWER8 RISE_POWER9 RISE_POWER10 RISE_POWER11 RISE_POWER12
FALL_POWER1 FALL_POWER2 FALL_POWER3 FALL_POWER4 FALL_POWER5 FALL_POWER6
FALL_POWER7 FALL_POWER8 FALL_POWER9 FALL_POWER10 FALL_POWER11 FALL_POWER12
RISE_TR1 RISE_TR2 RISE_TR3 RISE_TR4 RISE_TR5 RISE_TR6 RISE_TR7 RISE_TR8 RISE_TR9
RISE_TR10 RISE_TR11 RISE_TR12 FALL_TR1 FALL_TR2 FALL_TR3 FALL_TR4 FALL_TR5 FALL_TR6
FALL_TR7 FALL_TR8 FALL_TR9 FALL_TR10 FALL_TR11 FALL_TR12 CELL_RISE_DELAY1
CELL_RISE_DELAY2 CELL_RISE_DELAY3 CELL_RISE_DELAY4 CELL_RISE_DELAY5
CELL_RISE_DELAY6 CELL_RISE_DELAY7 CELL_RISE_DELAY8 CELL_RISE_DELAY9
CELL_RISE_DELAY10 CELL_RISE_DELAY11 CELL_RISE_DELAY12 CELL_FALL_DELAY_1
CELL_FALL_DELAY2 CELL_FALL_DELAY3 CELL_FALL_DELAY4 CELL_FALL_DELAY5
CELL_FALL_DELAY6 CELL_FALL_DELAY7 CELL_FALL_DELAY8 CELL_FALL_DELAY9
CELL_FALL_DELAY10 CELL_FALL_DELAY11 CELL_FALL_DELAY12 ILEAK_HI1 ILEAK_HI2
ILEAK_HI3 ILEAK_HI4 ILEAK_HI5 ILEAK_HI6 ILEAK_HI7 ILEAK_HI8 ILEAK_HI9 ILEAK_HI10
ILEAK_HI11 ILEAK_HI12 ILEAK_LO1 ILEAK_LO2 ILEAK_LO3 LEAK_LO4 ILEAK_LO5 ILEAK_LO6
ILEAK_LO7 ILEAK_LO8 ILEAK_LO9 ILEAK_LO10 ILEAK_LO11 ILEAK_LO12 BUF_RISE_DELAY1
BUF_RISE_DELAY2 BUF_RISE_DELAY3 BUF_RISE_DELAY4 BUF_RISE_DELAY5 BUF_RISE_DELAY6
BUF_RISE_DELAY7 BUF_RISE_DELAY8 BUF_RISE_DELAY9 BUF_RISE_DELAY10
BUF_RISE_DELAY11 BUF_RISE_DELAY12 BUF_FALL_DELAY1 BUF_FALL_DELAY2
BUF_FALL_DELAY3 BUF_FALL_DELAY4 BUF_FALL_DELAY5 BUF_FALL_DELAY6
BUF_FALL_DELAY7 BUF_FALL_DELAY8 BUF_FALL_DELAY9 BUF_FALL_DELAY10
BUF_FALL_DELAY11 BUF_FALL_DELAY12 \n";
```

```
$k=-1;
$j=-1;
$i=-1;

foreach $load (@loadarray)
  {


  print "HELLO WORLD1\n";
   $k++;
   $j=-1;
   $i=-1;
   foreach $searchstring (@modified_spice_array) {
       $i++;
       @cap = grep(/cload/,$searchstring);
       if ($#cap == -1) {

         }
     else {
         @temp1 = split(/\s/,$searchstring);
         $temp1[3] = $loadarray[$k];
         $temp2 = join(" ",@temp1);
         $modified_spice_array[$i] = $temp2."\n";

       }
     }
     foreach $tr (@trarray)
      {

        $j++;
        $i=-1;
         print "CAPACITANCE : $load  ";
        print "TRANSIENT TIME : $tr\n";

     $PW = 50e-9;
     $PERIOD = 100e-9;

      $ONTIME = $PW + $tr;
      $OFFTIME = $PERIOD - $ONTIME;


      $OFFSET1 = 10e-6;
      $OFFSET2 = 11e-6;
      $OFFSET3 = 12e-6;
      $OFFSET4 = 13e-6;
      $OFFSET5 = 14e-6;
      $OFFSET6 = 15e-6;
      $OFFSET7 = 16e-6;
      $OFFSET8 = 17e-6;
      $OFFSET9 = 18e-6;
      $OFFSET10 = 19e-6;
      $OFFSET11 = 20e-6;
      $OFFSET12 = 21e-6;
```

```
$SETTLING_TIME = 10e-9;
$DELAY_VALUE = 4e-9;
$RISE_DUR = $DELAY_VALUE + $SETTLING_TIME;
$FALL_DUR = $SETTLING_TIME;

$RISE_START_1 = $OFFSET1;
$RISE_STOP_1 = $OFFSET1 + $RISE_DUR;

$RISE_START_2 = $OFFSET2;
$RISE_STOP_2 = $OFFSET2 + $RISE_DUR;
$RISE_START_3 = $OFFSET3;
$RISE_STOP_3 = $OFFSET3 + $RISE_DUR;
$RISE_START_4 = $OFFSET4;
$RISE_STOP_4 = $OFFSET4 + $RISE_DUR;
$RISE_START_5 = $OFFSET5;
$RISE_STOP_5 = $OFFSET5 + $RISE_DUR;
$RISE_START_6 = $OFFSET6;
$RISE_STOP_6 = $OFFSET6 + $RISE_DUR;
$RISE_START_7 = $OFFSET7;
$RISE_STOP_7 = $OFFSET7 + $RISE_DUR;
$RISE_START_8 = $OFFSET8;
$RISE_STOP_8 = $OFFSET8 + $RISE_DUR;
$RISE_START_9 = $OFFSET9;
$RISE_STOP_9 = $OFFSET9 + $RISE_DUR;
$RISE_START_10 = $OFFSET10;
$RISE_STOP_10 = $OFFSET10 + $RISE_DUR;
$RISE_START_11 = $OFFSET11;
$RISE_STOP_11 = $OFFSET11 + $RISE_DUR;
 $RISE_START_12 = $OFFSET12;
$RISE_STOP_12 = $OFFSET12 + $RISE_DUR;



$FALL_START_1 = $OFFSET1 + $ONTIME;
$FALL_STOP_1 = $FALL_START_1 + $FALL_DUR;

$FALL_START_2 = $OFFSET2 + $ONTIME;
$FALL_STOP_2 = $FALL_START_2 + $FALL_DUR;

$FALL_START_3 = $OFFSET3 + $ONTIME;
$FALL_STOP_3 = $FALL_START_3 + $FALL_DUR;

$FALL_START_4 = $OFFSET4 + $ONTIME;
$FALL_STOP_4 = $FALL_START_4 + $FALL_DUR;

$FALL_START_5 = $OFFSET5 + $ONTIME;
$FALL_STOP_5 = $FALL_START_5 + $FALL_DUR;

$FALL_START_6 = $OFFSET6 + $ONTIME;
$FALL_STOP_6 = $FALL_START_6 + $FALL_DUR;

$FALL_START_7 = $OFFSET7 + $ONTIME;
$FALL_STOP_7 = $FALL_START_7 + $FALL_DUR;
```

```
     $FALL_START_8 = $OFFSET8 + $ONTIME;
     $FALL_STOP_8 = $FALL_START_8 + $FALL_DUR;

     $FALL_START_9 = $OFFSET9 + $ONTIME;
     $FALL_STOP_9 = $FALL_START_9 + $FALL_DUR;

     $FALL_START_10 = $OFFSET10 + $ONTIME;
     $FALL_STOP_10 = $FALL_START_10 + $FALL_DUR;


     $FALL_START_11 = $OFFSET11 + $ONTIME;
     $FALL_STOP_11 = $FALL_START_11 + $FALL_DUR;

     $FALL_START_12 = $OFFSET12 + $ONTIME;
     $FALL_STOP_12 = $FALL_START_12 + $FALL_DUR;


#######Replacing the V1 pulse source with the proper rise and fall transition time
  foreach $searchstring (@modified_spice_array) {
         $i++;
     @risetime = grep(/v7/,$searchstring);
      if ($#risetime == -1) {
       $modified_spice_array[$i] = $searchstring;
       }
    else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[7] = $trarray[$j];
       $temp1[8] = $trarray[$j];
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2."\n";
       print $modified_spice_array[$i];
       print "\n";
      }

    @rise_power_one = grep(/rise_power1\s/,$searchstring);
      if ($#rise_power_one == -1) {

       }
    else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$RISE_START_1";
       $temp1[6] = "TO="."$RISE_STOP_1"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
      }

    @rise_power_two = grep(/rise_power2/,$searchstring);
      if ($#rise_power_two == -1) {

       }
  else {
     @temp1 = split(/\s/,$searchstring);
     $temp1[5] = "FROM="."$RISE_START_2";
     $temp1[6] = "TO="."$RISE_STOP_2"."\n";
     $temp2 = join(" ",@temp1);
     $modified_spice_array[$i] = $temp2;
     print $modified_spice_array[$i];
     print "\n";
    }


@rise_power_3 = grep(/rise_power3/,$searchstring);
    if ($#rise_power_3 == -1) {

     }
  else {
     @temp1 = split(/\s/,$searchstring);
     $temp1[5] = "FROM="."$RISE_START_3";
     $temp1[6] = "TO="."$RISE_STOP_3"."\n";
     $temp2 = join(" ",@temp1);
     $modified_spice_array[$i] = $temp2;
     print $modified_spice_array[$i];
     print "\n";
    }

@rise_power_4 = grep(/rise_power4/,$searchstring);
    if ($#rise_power_4 == -1) {
    # $modified_spice_array[$i] = $searchstring;
     }
  else {
     @temp1 = split(/\s/,$searchstring);
     $temp1[5] = "FROM="."$RISE_START_4";
     $temp1[6] = "TO="."$RISE_STOP_4"."\n";
     $temp2 = join(" ",@temp1);
     $modified_spice_array[$i] = $temp2;
     print $modified_spice_array[$i];
     print "\n";
    }

@rise_power_5 = grep(/rise_power5/,$searchstring);
    if ($#rise_power_5 == -1) {
    # $modified_spice_array[$i] = $searchstring;
     }
  else {
     @temp1 = split(/\s/,$searchstring);
     $temp1[5] = "FROM="."$RISE_START_5";
     $temp1[6] = "TO="."$RISE_STOP_5"."\n";
     $temp2 = join(" ",@temp1);
     $modified_spice_array[$i] = $temp2;
     print $modified_spice_array[$i];
     print "\n";
    }
@rise_power_6 = grep(/rise_power6/,$searchstring);
```

```
    if ($#rise_power_6 == -1) {
    # $modified_spice_array[$i] = $searchstring;
     }
  else {
      @temp1 = split(/\s/,$searchstring);
      $temp1[5] = "FROM="."$RISE_START_6";
      $temp1[6] = "TO="."$RISE_STOP_6"."\n";
      $temp2 = join(" ",@temp1);
      $modified_spice_array[$i] = $temp2;
      print $modified_spice_array[$i];
      print "\n";
    }

@rise_power_7 = grep(/rise_power7/,$searchstring);
    if ($#rise_power_7 == -1) {
    # $modified_spice_array[$i] = $searchstring;
     }
  else {
      @temp1 = split(/\s/,$searchstring);
      $temp1[5] = "FROM="."$RISE_START_7";
      $temp1[6] = "TO="."$RISE_STOP_7"."\n";
      $temp2 = join(" ",@temp1);
      $modified_spice_array[$i] = $temp2;
      print $modified_spice_array[$i];
      print "\n";
    }

@rise_power_8 = grep(/rise_power8/,$searchstring);
    if ($#rise_power_8 == -1) {
    # $modified_spice_array[$i] = $searchstring;
     }
  else {
      @temp1 = split(/\s/,$searchstring);
      $temp1[5] = "FROM="."$RISE_START_8";
      $temp1[6] = "TO="."$RISE_STOP_8"."\n";
      $temp2 = join(" ",@temp1);
      $modified_spice_array[$i] = $temp2;
      print $modified_spice_array[$i];
      print "\n";
    }
@rise_power_9 = grep(/rise_power9/,$searchstring);
    if ($#rise_power_9 == -1) {
    # $modified_spice_array[$i] = $searchstring;
     }
  else {
      @temp1 = split(/\s/,$searchstring);
      $temp1[5] = "FROM="."$RISE_START_9";
      $temp1[6] = "TO="."$RISE_STOP_9"."\n";
      $temp2 = join(" ",@temp1);
      $modified_spice_array[$i] = $temp2;
      print $modified_spice_array[$i];
      print "\n";
    }
  @rise_power_10 = grep(/rise_power10/,$searchstring);
     if ($#rise_power_10 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$RISE_START_10";
       $temp1[6] = "TO="."$RISE_STOP_10"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
     }

  @rise_power_11 = grep(/rise_power11/,$searchstring);
     if ($#rise_power_11 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$RISE_START_11";
       $temp1[6] = "TO="."$RISE_STOP_11"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;

       print $modified_spice_array[$i];
       print "\n";
     }

@rise_power_12 = grep(/rise_power12/,$searchstring);
     if ($#rise_power_12 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$RISE_START_12";
       $temp1[6] = "TO="."$RISE_STOP_12"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;

       print $modified_spice_array[$i];
       print "\n";
     }

  @fall_power_one = grep(/fall_power1\s/,$searchstring);
     if ($#fall_power_one == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_1";
       $temp1[6] = "TO="."$FALL_STOP_1"."\n";
       $temp2 = join(" ",@temp1);
```

```
        $modified_spice_array[$i] = $temp2;
        print $modified_spice_array[$i];
        print "\n";
       }

  @fall_power_two = grep(/fall_power2/,$searchstring);
      if ($#fall_power_two == -1) {
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
        @temp1 = split(/\s/,$searchstring);



        $temp1[5] = "FROM="."$FALL_START_2";
        $temp1[6] = "TO="."$FALL_STOP_2"."\n";
        $temp2 = join(" ",@temp1);
        $modified_spice_array[$i] = $temp2;
        print $modified_spice_array[$i];
        print "\n";
       }


 @fall_power_3 = grep(/fall_power3/,$searchstring);
      if ($#fall_power_3 == -1) {
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
        @temp1 = split(/\s/,$searchstring);
        $temp1[5] = "FROM="."$FALL_START_3";
        $temp1[6] = "TO="."$FALL_STOP_3"."\n";
        $temp2 = join(" ",@temp1);
        $modified_spice_array[$i] = $temp2;
        print $modified_spice_array[$i];
        print "\n";
       }


  @fall_power_4 = grep(/fall_power4/,$searchstring);
      if ($#fall_power_4 == -1) {
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
        @temp1 = split(/\s/,$searchstring);
        $temp1[5] = "FROM="."$FALL_START_4";
        $temp1[6] = "TO="."$FALL_STOP_4"."\n";
        $temp2 = join(" ",@temp1);
        $modified_spice_array[$i] = $temp2;
        print $modified_spice_array[$i];
        print "\n";
       }

 @fall_power_5 = grep(/fall_power5/,$searchstring);
      if ($#fall_power_5 == -1) {
```

```
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_5";
       $temp1[6] = "TO="."$FALL_STOP_5"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
      }

 @fall_power_6 = grep(/fall_power6/,$searchstring);

      if ($#fall_power_6 == -1) {
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_6";
       $temp1[6] = "TO="."$FALL_STOP_6"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
      }

 @fall_power_7 = grep(/fall_power7/,$searchstring);
      if ($#fall_power_7 == -1) {
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_7";
       $temp1[6] = "TO="."$FALL_STOP_7"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
      }
 @fall_power_8 = grep(/fall_power8/,$searchstring);
      if ($#fall_power_8 == -1) {
      # $modified_spice_array[$i] = $searchstring;
       }
    else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_8";
       $temp1[6] = "TO="."$FALL_STOP_8"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
      }
```

```
@fall_power_9 = grep(/fall_power9/,$searchstring);
     if ($#fall_power_9 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_9";
       $temp1[6] = "TO="."$FALL_STOP_9"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;



       print $modified_spice_array[$i];
       print "\n";
     }

@fall_power_10 = grep(/fall_power10/,$searchstring);
     if ($#fall_power_10 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_10";
       $temp1[6] = "TO="."$FALL_STOP_10"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
     }

@fall_power_11 = grep(/fall_power11/,$searchstring);
     if ($#fall_power_11 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_11";
       $temp1[6] = "TO="."$FALL_STOP_11"."\n";
       $temp2 = join(" ",@temp1);
       $modified_spice_array[$i] = $temp2;
       print $modified_spice_array[$i];
       print "\n";
     }

@fall_power_12 = grep(/fall_power12/,$searchstring);
     if ($#fall_power_12 == -1) {
     # $modified_spice_array[$i] = $searchstring;
      }
   else {
       @temp1 = split(/\s/,$searchstring);
       $temp1[5] = "FROM="."$FALL_START_12";
```

```
        $temp1[6] = "TO="."$FALL_STOP_12"."\n";
        $temp2 = join(" ",@temp1);
        $modified_spice_array[$i] = $temp2;
        print $modified_spice_array[$i];
        print "\n";
       }
 } ##end of foreach of mystring
############Replacing the measure command with the proper values so that they measure correctly



   open(OUTFILE,">spice_lr_temp.ckt");
   foreach $printstring (@modified_spice_array)
     {
    print OUTFILE $printstring;
   }
   close OUTFILE;
    print $loadarray[$k];
    print "   ";
    print  $trarray[$j];
    print "  ";
    print  "SIMULATION IN PROGRESS..\n";
    `hspice  spice_lr_temp.ckt`;
   open(MTHANDLE,"spice_lr_temp.mt0");
   $line_count=0;


   print MEASHANDLE $load."F";
   print MEASHANDLE "         ";
   print ENERGYHANDLE $load."F";
   print ENERGYHANDLE "         ";
   print MEASHANDLE ($tr/1e-9)."ns";
   print MEASHANDLE "         ";
   print ENERGYHANDLE ($tr/1e-9)."ns";
   print ENERGYHANDLE "         ";

   while(<MTHANDLE>) {
     $line_count++;
     chop();

 if($line_count >=40 && $line_count <= 45)

     {
           for($count = 0; $count < 4; $count++) {
             /(-*\d+\.\d+e-*\d+)/g;
             $number[$count] = $1;

           }
       $temp10[0] = ($number[0] *-1.5 * $RISE_DUR);
       $temp10[1] = ($number[1] *-1.5 * $RISE_DUR);
       $temp10[2] = ($number[2] *-1.5 * $RISE_DUR);
       $temp10[3]  = ($number[3] *-1.5 * $RISE_DUR);
       $energy_string = join(" ",@temp10);
```

```
        print MEASHANDLE $_;
        print ENERGYHANDLE $energy_string;
        print MEASHANDLE " ";
        print ENERGYHANDLE " ";
              }


 if($line_count >=46 && $line_count <= 51)

      {
              for($count = 0; $count < 4; $count++) {
               /(-*\d+\.\d+e-*\d+)/g;
               $number[$count] = $1;

              }
        $temp10[0] = ($number[0] /1e-12);
        $temp10[1] = ($number[1] /1e-12);
        $temp10[2] = ($number[2] /1e-12);
        $temp10[3]  = ($number[3] /1e-12);
        $energy_string = join(" ",@temp10);

        print MEASHANDLE $_;
        print ENERGYHANDLE $energy_string;
        print MEASHANDLE " ";
        print ENERGYHANDLE " ";
              }

if($line_count >=52 && $line_count <= 54)

      {
              for($count = 0; $count < 4; $count++) {
               /(-*\d+\.\d+e-*\d+)/g;
               $number[$count] = $1;

              }
        $temp10[0] = ($number[0] /1e-9);
        $temp10[1] = ($number[1] /1e-9);
        $temp10[2] = ($number[2] /1e-9);
        $temp10[3]  = ($number[3] /1e-9);
        $energy_string = join(" ",@temp10);

        print MEASHANDLE $_;
        print ENERGYHANDLE $energy_string;
        print MEASHANDLE " ";
        print ENERGYHANDLE " ";
              }
if($line_count >=55 && $line_count <= 57)
      {
              for($count = 0; $count < 4; $count++) {
               /(-*\d+\.\d+e-*\d+)/g;
               $number[$count] = $1;

              }
        $temp10[0] = ($number[0] /1e-12);
        $temp10[1] = ($number[1] /1e-12);
        $temp10[2] = ($number[2] /1e-12);
        $temp10[3]  = ($number[3] /1e-12);
        $energy_string = join(" ",@temp10);

        print MEASHANDLE $_;



        print ENERGYHANDLE $energy_string;
        print MEASHANDLE " ";
        print ENERGYHANDLE " ";
              }

if($line_count >=58 && $line_count <= 63)

      {
            for($count = 0; $count < 4; $count++) {
              /(-*\d+\.\d+e-*\d+)/g;
              $number[$count] = $1;

            }
        $temp10[0] = ($number[0] /-1e-9);
        $temp10[1] = ($number[1] /-1e-9);
        $temp10[2] = ($number[2] /-1e-9);
        $temp10[3]  = ($number[3] /-1e-9);
        $energy_string = join(" ",@temp10);

        print MEASHANDLE $_;
        print ENERGYHANDLE $energy_string;
        print MEASHANDLE " ";
        print ENERGYHANDLE " ";
              }


if($line_count >=64 && $line_count <= 69)

      {
            for($count = 0; $count < 4; $count++) {
              /(-*\d+\.\d+e-*\d+)/g;
              $number[$count] = $1;

            }
        $temp10[0] = ($number[0] /1e-12);
        $temp10[1] = ($number[1] /1e-12);
        $temp10[2] = ($number[2] /1e-12);
        $temp10[3]  = ($number[3] /1e-12);
        $energy_string = join(" ",@temp10);

        print MEASHANDLE $_;
        print ENERGYHANDLE $energy_string;
        print MEASHANDLE " ";
        print ENERGYHANDLE " ";
              }


   }

     print MEASHANDLE "\n";
     print ENERGYHANDLE "\n";
     print "HELLO WORLD\n";



close (MTHANDLE);
} #closing of tr array
} #closing of load array

exit;
```

APPENDIX C

SUPECISC RHF AUTOMATION SCRIPTS

This Appendix contains the scripts for the automation of the BIGFABRIC placement and routing.

C.1 BIGFABRIC INITIALIZATION SCRIPT

This Appendix contains the TCL scripts for "ibm_bigfabric_init.tcl". It is used to initialize the top-level chip with its die size and power ring details.

```
setImportMode -syncRelativePath 1
#loadConfig bigfabric.conf 1
set CORE_TO_LEFT  16.8
set CORE_TO_RIGHT 16.8
set CORE_TO_TOP 16.8
set CORE_TO_BOTTOM 16.8

set MODULE_WIDTH 500.8
set NUMBER_OF_MODULES_PER_STRIPE 20
set MODULE_LEFT_DISTANCE 3.6
set MODULE_RIGHT_DISTANCE 3.6

set INTER_MODULE_DISTANCE_MIN 7.2
set INTER_MODULE_DISTANCE_MAX 22.4

set POWER_RING_TOTAL_LEFT 16.8
set POWER_RING_TOTAL_RIGHT 16.8
set VAL [expr "$NUMBER_OF_MODULES_PER_STRIPE % 2"]
if {$VAL ==1} {
   set DIE_WIDTH [expr {($MODULE_WIDTH *$NUMBER_OF_MODULES_PER_STRIPE) +
((($NUMBER_OF_MODULES_PER_STRIPE-1)/2)*$INTER_MODULE_DISTANCE_MIN) +
(((($NUMBER_OF_MODULES_PER_STRIPE-1)/2)-1)*$INTER_MODULE_DISTANCE_MAX) +
$MODULE_LEFT_DISTANCE + $MODULE_RIGHT_DISTANCE + $POWER_RING_TOTAL_LEFT +
$POWER_RING_TOTAL_RIGHT} ]
} else {
   set DIE_WIDTH [expr {($MODULE_WIDTH *$NUMBER_OF_MODULES_PER_STRIPE) +
((($NUMBER_OF_MODULES_PER_STRIPE)/2)*$INTER_MODULE_DISTANCE_MIN) +
(((($NUMBER_OF_MODULES_PER_STRIPE)/2)-1)*$INTER_MODULE_DISTANCE_MAX) +
$MODULE_LEFT_DISTANCE + $MODULE_RIGHT_DISTANCE + $POWER_RING_TOTAL_LEFT +
$POWER_RING_TOTAL_RIGHT}]
}

set POWER_RING_WIDTH_TOP 7.2
set POWER_RING_WIDTH_BOTTOM 7.2
set POWER_RING_WIDTH_LEFT 7.2
set POWER_RING_WIDTH_RIGHT 7.2

set POWER_RING_SPACING_TOP 0.8
set POWER_RING_SPACING_BOTTOM 0.8
set POWER_RING_SPACING_LEFT 0.8
set POWER_RING_SPACING_RIGHT 0.8

#this power_ring_offset_left is the distance between the IO and the power ring and the distance between the power
ring and core
set POWER_RING_OFFSET_TOP 0.8
set POWER_RING_OFFSET_BOTTOM 0.8
set POWER_RING_OFFSET_LEFT 0.8
set POWER_RING_OFFSET_RIGHT 0.8

#The 0.8 is distance between alu and mux
set INTER_STRIPE_SPACING 7.2
set FINAL_MUX_STRIPE_ROUTING_OFFSET 40
set DIE_HEIGHT_ALU_STRIPE 285.6
set DIE_HEIGHT_MUX_STRIPE 103.2
set DIE_HEIGHT_FINAL_MUX_STRIPE 69.6
set CHIP_POWER_OFFSET 16.8
set ALU_MUX_TOTAL_HEIGHT [expr {$DIE_HEIGHT_ALU_STRIPE + $DIE_HEIGHT_MUX_STRIPE}]

#20.8 for additional mux height
set DIE_HEIGHT [expr {($ALU_MUX_TOTAL_HEIGHT*17) + ($INTER_STRIPE_SPACING*34) +
$DIE_HEIGHT_ALU_STRIPE + $INTER_STRIPE_SPACING + $DIE_HEIGHT_FINAL_MUX_STRIPE + (2 *
$CHIP_POWER_OFFSET) + $FINAL_MUX_STRIPE_ROUTING_OFFSET + 2.4 + 14.4 + 7.2 + 2.0}]

puts "DH $DIE_HEIGHT"

floorplan -d $DIE_WIDTH $DIE_HEIGHT $CORE_TO_LEFT $CORE_TO_BOTTOM $CORE_TO_RIGHT
$CORE_TO_TOP -fplanOrigin l
globalNetConnect vdd -type pgpin -pin VDD -instanceBasename * -all -override -verbose
globalNetConnect gnd -type pgpin -pin VSS -instanceBasename * -all -override -verbose

addRing -nets {gnd vdd} -type core_rings -follow core -center 0 -offset_top $POWER_RING_OFFSET_TOP -
offset_bottom $POWER_RING_OFFSET_BOTTOM -offset_left $POWER_RING_OFFSET_LEFT -offset_right
```

```
$POWER_RING_OFFSET_RIGHT -layer_top M3 -layer_bottom M3 -layer_left M2 -layer_right M2 -width_top
$POWER_RING_WIDTH_TOP -width_bottom $POWER_RING_WIDTH_BOTTOM -width_left
$POWER_RING_WIDTH_LEFT -width_right $POWER_RING_WIDTH_RIGHT  -snap_wire_center_to_grid
Grid -spacing_top $POWER_RING_SPACING_TOP -spacing_bottom $POWER_RING_SPACING_BOTTOM -
spacing_left $POWER_RING_SPACING_LEFT -spacing_right $POWER_RING_SPACING_RIGHT
```

C.2 FLOORPLANNING SCRIPT

```
set LEFT_IO_TO_CORE      16.8
set RIGHT_IO_TO_CORE     16.8
set TOP_IO_TO_CORE       16.8
set BOTTOM_IO_TO_CORE    16.8

set LEFT_CORE_TO_MODULE 0.0
set RIGHT_CORE_TO_MODULE 0.0
set TOP_CORE_TO_MODULE 0.0
set BOTTOM_CORE_TO_MODULE 0.0
set INTER_STRIPE_SPACING 7.2

set FINAL_MUX_STRIPE_ROUTING_OFFSET 40
set DIE_HEIGHT_ALU_STRIPE 213.6
set DIE_HEIGHT_MUX_STRIPE 103.2
set DIE_HEIGHT_FINAL_MUX_STRIPE 69.6
set CHIP_POWER_OFFSET 16.8
set POWER_RING_TOTAL_LEFT 16.8
set POWER_RING_TOTAL_RIGHT 16.8
set ALU_MUX_TOTAL_HEIGHT [expr {$DIE_HEIGHT_ALU_STRIPE + $DIE_HEIGHT_MUX_STRIPE}]

set MODULE_WIDTH 500.8
set MODULE_LEFT_DISTANCE 3.6
set MODULE_RIGHT_DISTANCE 3.6
set INTER_MODULE_DISTANCE_MIN 7.2
set INTER_MODULE_DISTANCE_MAX 22.4


set NUMBER_OF_MODULES_PER_STRIPE 20

#The additional numbers that are added to the die height are to make the core height a multiple of the standard cell
height
set DIE_HEIGHT [expr {($ALU_MUX_TOTAL_HEIGHT*17) + ($INTER_STRIPE_SPACING*34) +
($DIE_HEIGHT_ALU_STRIPE)+ $INTER_STRIPE_SPACING + $DIE_HEIGHT_FINAL_MUX_STRIPE +
(2*$CHIP_POWER_OFFSET) + $FINAL_MUX_STRIPE_ROUTING_OFFSET + 2.4 + 14.4 + 7.2 + 2.0}]

set VAL [expr "$NUMBER_OF_MODULES_PER_STRIPE % 2"]
if {$VAL ==1} {
    set DIE_WIDTH_STRIPE [expr {($MODULE_WIDTH *$NUMBER_OF_MODULES_PER_STRIPE) +
((($NUMBER_OF_MODULES_PER_STRIPE-1)/2)*$INTER_MODULE_DISTANCE_MIN) +
(((($NUMBER_OF_MODULES_PER_STRIPE-1)/2)-1)*$INTER_MODULE_DISTANCE_MAX) +
$MODULE_LEFT_DISTANCE + $MODULE_RIGHT_DISTANCE + $POWER_RING_TOTAL_LEFT +
$POWER_RING_TOTAL_RIGHT} ]
} else {
    set DIE_WIDTH_STRIPE [expr {($MODULE_WIDTH *$NUMBER_OF_MODULES_PER_STRIPE) +
((($NUMBER_OF_MODULES_PER_STRIPE-1)/2)*$INTER_MODULE_DISTANCE_MIN) +
(((($NUMBER_OF_MODULES_PER_STRIPE)/2)-1)*$INTER_MODULE_DISTANCE_MAX) +
$MODULE_LEFT_DISTANCE + $MODULE_RIGHT_DISTANCE + $POWER_RING_TOTAL_LEFT +
$POWER_RING_TOTAL_RIGHT}]
}


set WIDTH_ALU_STRIPE            $DIE_WIDTH_STRIPE
set WIDTH_MUX_STRIPE            $DIE_WIDTH_STRIPE
set FINAL_MUX_STRIPE_STRING "I18_m"

set CURR_MODULE 0
set CUR_X  [expr {$LEFT_IO_TO_CORE + $LEFT_CORE_TO_MODULE}]
set NEXT_X [expr {$WIDTH_ALU_STRIPE + $LEFT_IO_TO_CORE +$LEFT_CORE_TO_MODULE}]

set NEXT_Y  [expr {$DIE_HEIGHT - $TOP_IO_TO_CORE - $TOP_CORE_TO_MODULE - 7.2}]
set CUR_Y   [expr {$DIE_HEIGHT - $TOP_IO_TO_CORE - $TOP_CORE_TO_MODULE - 7.2}]

#Placement of Final Mux stripe
set CUR_STR $FINAL_MUX_STRIPE_STRING
set CUR_Y [expr "$CUR_Y - $DIE_HEIGHT_FINAL_MUX_STRIPE"]

setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y
puts "setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y"

set CUR_Y [expr "$CUR_Y  -$INTER_STRIPE_SPACING - $FINAL_MUX_STRIPE_ROUTING_OFFSET"]
set NEXT_Y [expr "$NEXT_Y - $INTER_STRIPE_SPACING - $DIE_HEIGHT_FINAL_MUX_STRIPE -
$FINAL_MUX_STRIPE_ROUTING_OFFSET"]

for {set i 1} {$i<18} {incr i} {
        puts "HELLO WORLD $i"
#Placement of ALU Stripe
        set CURR_MODULE [expr "$CURR_MODULE+1"]
        set CUR_Y [expr "$CUR_Y  - $DIE_HEIGHT_ALU_STRIPE"]
        set CUR_STR "I"
        append CUR_STR $CURR_MODULE
        append CUR_STR "_s"
        setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y
        puts "setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y"
#Placement of Mux stripe
        set CUR_Y [expr "$CUR_Y - $INTER_STRIPE_SPACING - $DIE_HEIGHT_MUX_STRIPE"]
        set NEXT_Y [expr "$NEXT_Y -$INTER_STRIPE_SPACING - $DIE_HEIGHT_ALU_STRIPE"]
        set CUR_STR "I"
        append CUR_STR $CURR_MODULE
        append CUR_STR "_m"
        setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y
        puts "setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y"
        set NEXT_Y [expr {$NEXT_Y - $INTER_STRIPE_SPACING -$DIE_HEIGHT_MUX_STRIPE}]
        set CUR_Y [expr {$CUR_Y - $INTER_STRIPE_SPACING}]
```

```
}

 #Placement of last ALU Stripe
        puts "HELLO WORLD $i"
        set CURR_MODULE [expr "$CURR_MODULE+1"]
        set CUR_Y [expr "$CUR_Y  - $DIE_HEIGHT_ALU_STRIPE"]
        set CUR_STR "I"
        append CUR_STR $CURR_MODULE
        append CUR_STR "_s"
        setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y
        puts "setObjFPlanBox Module $CUR_STR $CUR_X $CUR_Y $NEXT_X $NEXT_Y"
```

C.3 ALU STRIPE PIN ASSIGNMENT SCRIPT

```
set CLEARANCE 0.0
set NUMBER_OF_MODULES 20

#Set input pins
set WIDTH_ALU 500.8

#set DIE_HEIGHT_ALU_STRIPE 213.6
set DIE_HEIGHT_ALU_STRIPE 285.6

set DIE_WIDTH_ALU_STRIPE 10330.0
set PIN_BANK_SPACING [expr {$WIDTH_ALU}]
#16.8 for the power ring
set CHIP_OFFSET  [expr {16.8 + 3.6}]

#10.4 to accomodate the spacing
set INTER_ALU_DISTANCE_MAX 22.4
set INTER_ALU_DISTANCE_MIN 7.2
set INPUT_PIN_SPACING 1.6
set OUTPUT_PIN_SPACING 1.6

#its taken care such that the pin position after offset + 23.92 + 5.2 is aligned on a 0.4um pin grid
set INP1_OFFSET 40.0
set INP2_OFFSET 290.0
set DOUT_OFFSET 340.0


set RIGHT_OFFSET_TOP 0
set RIGHT_OFFSET_BOTTOM 0
```

```
#************************PLACING inp1 ***********************#
#Placing Pins inp1
set BANK_NO [expr {$NUMBER_OF_MODULES -1}]
set Y_LOC $DIE_HEIGHT_ALU_STRIPE
#The subtraction of the input pin spacing is necessary
set X_LOC [expr {$INP1_OFFSET + $CHIP_OFFSET - $INPUT_PIN_SPACING}]
set FLAG 0
set PINS_PER_ALU_COUNT 1
set PIN_COUNT [expr {($NUMBER_OF_MODULES*32)-1}]
for {set i $PIN_COUNT} {$i > -1} {decr i 1} {
   if {$PINS_PER_ALU_COUNT == 33} {
   if {$FLAG == 0} {
   set X_LOC [expr {($X_LOC - (31*$INPUT_PIN_SPACING)) + $PIN_BANK_SPACING +
$INTER_ALU_DISTANCE_MIN}]
   set FLAG 1
   } else {
       set X_LOC [expr {($X_LOC - (31*$INPUT_PIN_SPACING)) + $PIN_BANK_SPACING +
$INTER_ALU_DISTANCE_MAX}]
   set FLAG 0
   }
   set PINS_PER_ALU_COUNT 2
 } else {
   set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
   incr PINS_PER_ALU_COUNT 1
 }
set PIN_NAME "inp1\\\[$i\\\]"
puts "$PIN_NAME $X_LOC $Y_LOC"
preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
}

#*************PLACING  inp2 pins and sel bus pins***********************#
#Placing Pins inp2
set PIN_COUNT [expr {($NUMBER_OF_MODULES*32)-1}]
set Y_LOC $DIE_HEIGHT_ALU_STRIPE
set X_LOC [expr {$INP2_OFFSET + $CHIP_OFFSET -$INPUT_PIN_SPACING}]
set FLAG 0
set PINS_PER_ALU_COUNT 1
set BANK_COUNT [expr {$NUMBER_OF_MODULES -1}]
for {set i $PIN_COUNT} {$i > -1} {decr i 1} {
if {$PINS_PER_ALU_COUNT == 33} {
  set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
  set PIN_NAME "sel_bus\\\[$BANK_COUNT\\\]"
  puts "$PIN_NAME"
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
 decr BANK_COUNT 1
   if {$FLAG == 0} {
   set X_LOC [expr {($X_LOC - (32*$INPUT_PIN_SPACING)) + $PIN_BANK_SPACING +
$INTER_ALU_DISTANCE_MIN}]
   set FLAG 1
   } else {
   set X_LOC [expr {($X_LOC - (32*$INPUT_PIN_SPACING)) + $PIN_BANK_SPACING +
$INTER_ALU_DISTANCE_MAX}]
   set FLAG 0
   }
  set PINS_PER_ALU_COUNT 2
 } else {
  set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
  incr PINS_PER_ALU_COUNT 1
 }
set PIN_NAME "inp2\\\[$i\\\]"
puts "$PIN_NAME"
preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
}

#****************************Placing of sel bus 0  pins**********************
##The following section of code places the sel_bus[0]
 set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
set PIN_NAME "sel_bus\\\[0\\\]"
puts "$PIN_NAME"
preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2


#***************************PLACING  dout pins***********************#
#Placing Pins dout
set PIN_COUNT [expr {($NUMBER_OF_MODULES*32)-1}]
set Y_LOC 0
set X_LOC [expr {$DOUT_OFFSET +$CHIP_OFFSET -$OUTPUT_PIN_SPACING}]
set FLAG 0
set PINS_PER_ALU_COUNT 1
set BANK_COUNT [expr {$NUMBER_OF_MODULES -1}]

for {set i $PIN_COUNT} {$i > -1} {decr i 1} {
if {$PINS_PER_ALU_COUNT == 33} {

  set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
  set PIN_NAME "l_dout_bus\\\[$BANK_COUNT\\\]"
  puts "$PIN_NAME"
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
  decr BANK_COUNT 1
   if {$FLAG == 0} {
   set X_LOC [expr {($X_LOC - (32*$OUTPUT_PIN_SPACING)) + $PIN_BANK_SPACING +
$INTER_ALU_DISTANCE_MIN}]
   set FLAG 1
   } else {
       set X_LOC [expr {($X_LOC - (32*$OUTPUT_PIN_SPACING)) + $PIN_BANK_SPACING +
$INTER_ALU_DISTANCE_MAX}]
   set FLAG 0
   }
  set PINS_PER_ALU_COUNT 2
 } else {
  set X_LOC [expr {$X_LOC + $OUTPUT_PIN_SPACING}]
  incr PINS_PER_ALU_COUNT 1
 }
set PIN_NAME "dout_bus\\\[$i\\\]"
puts "$PIN_NAME"
preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
}

##The following section of code places the l_dout_bus[0]
 set X_LOC [expr {$X_LOC + $OUTPUT_PIN_SPACING}]
set PIN_NAME "l_dout_bus\\\[0\\\]"
puts "$PIN_NAME"
```

```
preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
#***************************************************************************

##Assignment of control pins
set X_LOC 0
set HOR_FEEDTHROUGH_OFFSET 3.6
# The Y location is set so that the pin is assigned near the top
set Y_LOC [expr {$DIE_HEIGHT_ALU_STRIPE - 16.8 - $HOR_FEEDTHROUGH_OFFSET}]
set VERT_PIN_SPACING 0.4
set SET_TO_SET_OFFSET 0.8
set  LAST_MODULE_NUMBER [expr {$NUMBER_OF_MODULES-1}]

puts $X_LOC
puts $Y_LOC

for {set i $LAST_MODULE_NUMBER} {$i > -1} {decr i 1} {

  set PIN_NAME "lo_hi_bus\\\[$i\\\]"
  set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
  puts "$PIN_NAME"
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5

  set OPSEL_CNT [expr ((($i+1)*5)-1)]
  set PIN_NAME "op_sel_bus\\\[$OPSEL_CNT\\\]"
  puts $PIN_NAME
  set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5

  decr OPSEL_CNT 1
  set PIN_NAME "op_sel_bus\\\[$OPSEL_CNT\\\]"
  puts $PIN_NAME
  set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5

  decr OPSEL_CNT 1
  set PIN_NAME "op_sel_bus\\\[$OPSEL_CNT\\\]"
  puts $PIN_NAME
  set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5

  decr OPSEL_CNT 1
  set PIN_NAME "op_sel_bus\\\[$OPSEL_CNT\\\]"
  puts $PIN_NAME
  set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5

  decr OPSEL_CNT 1
  set PIN_NAME "op_sel_bus\\\[$OPSEL_CNT\\\]"
  puts $PIN_NAME
  set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
  preassignPin stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5

  set Y_LOC [expr {$Y_LOC - $SET_TO_SET_OFFSET}]
}
```

C.4 MUX STRIPE PIN ASSIGNMENT SCRIPT

```
set NUMBER_OF_MODULES 20
set CHIP_OFFSET [expr {16.8 + 3.6}]


#Set input pins
set WIDTH_MUX 500.8

set HEIGHT_MUX_STRIPE 103.2
set WIDTH_MUX_STRIPE 10330.0

set LEFT_OFFSET_TOP 340.00
set LEFT_OFFSET_BOTTOM 0
set RIGHT_OFFSET_TOP 0
set RIGHT_OFFSET_BOTTOM 0

#INPUT PINS ARE DOUT
#OUTPUT PINS ARE INTEROUT1 and INTEROUT2
set INPUT_OFFSET $LEFT_OFFSET_TOP
#set INTEROUT1_OFFSET 78
#set INTEROUT2_OFFSET 328

set INTEROUT1_OFFSET 40
set INTEROUT2_OFFSET 290

set DOUT_BANK_SPACING [expr {$WIDTH_MUX}]
set INTEROUT1_BANK_SPACING [expr {$WIDTH_MUX}]
set INTEROUT2_BANK_SPACING [expr {$WIDTH_MUX}]
#The 10.4 is added to compensate for the distance between the mux and the power stripe
set INTER_MUX_DISTANCE_MAX 22.4
set INTER_MUX_DISTANCE_MIN 7.2
set INPUT_PIN_SPACING 1.6
set OUTPUT_PIN_SPACING 1.6

#****************************PLACING interout1************************
set PIN_COUNT [expr {($NUMBER_OF_MODULES *32)-1}]
set Y_LOC 0
set FLAG 0
set X_LOC [expr {$INTEROUT1_OFFSET + $CHIP_OFFSET -$OUTPUT_PIN_SPACING}]
set PINS_PER_BANK 1
set BANK_NO 19

for {set i $PIN_COUNT} {$i > -1} {decr i 1} {
if {$PINS_PER_BANK == 33} {
   if {$FLAG ==0} {
   set X_LOC [expr {($X_LOC -(31*$OUTPUT_PIN_SPACING)) + $INTEROUT1_BANK_SPACING +
$INTER_MUX_DISTANCE_MIN}]
   set FLAG 1
   } else {
```

```
   set X_LOC [expr {($X_LOC -(31*$OUTPUT_PIN_SPACING)) + $INTEROUT1_BANK_SPACING +
$INTER_MUX_DISTANCE_MAX}]
   set FLAG 0
   }
   set PINS_PER_BANK 2
  } else {
  set X_LOC [expr {$X_LOC + $OUTPUT_PIN_SPACING}]
  incr PINS_PER_BANK 1
}

set PIN_NAME "interout1\\\[$i\\\]"
puts $PIN_NAME
preassignPin mux_instance $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
}

#****************************PLACING interout2 and interout3 pins******
set PIN_COUNT [expr {(($NUMBER_OF_MODULES *32)-1)}]
set Y_LOC 0
set X_LOC [expr {$INTEROUT2_OFFSET +$CHIP_OFFSET -$OUTPUT_PIN_SPACING}]
set PINS_PER_BANK 1
set BANK_NO 19
set FLAG 0
for {set i $PIN_COUNT} {$i > -1} {decr i 1} {
if {$PINS_PER_BANK == 33} {
    set TEMP_X_LOC [expr {$X_LOC + $OUTPUT_PIN_SPACING}]
    set PIN_NAME "interout3\\\[$BANK_NO\\\]"
    preassignPin mux_instance $PIN_NAME -loc $TEMP_X_LOC $Y_LOC -layer 2
    if {$FLAG ==0} {
     set X_LOC [expr ($X_LOC - (31*$OUTPUT_PIN_SPACING)) + $INTEROUT2_BANK_SPACING +
$INTER_MUX_DISTANCE_MIN]
            set FLAG 1
   } else {
    set X_LOC [expr ($X_LOC - (31*$OUTPUT_PIN_SPACING)) + $INTEROUT2_BANK_SPACING +
$INTER_MUX_DISTANCE_MAX]
    set FLAG 0
   }
  set PINS_PER_BANK 2
  decr BANK_NO 1
 } else {
  set X_LOC [expr {$X_LOC + $OUTPUT_PIN_SPACING}]
  incr PINS_PER_BANK 1
 }

set PIN_NAME "interout2\\\[$i\\\]"
puts "$PIN_NAME"
preassignPin mux_instance $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
}
 set TEMP_X_LOC [expr {$X_LOC + $OUTPUT_PIN_SPACING}]
 set PIN_NAME "interout3\\\[$BANK_NO\\\]"
 preassignPin mux_instance $PIN_NAME -loc $TEMP_X_LOC $Y_LOC -layer 2


#****************************PLACING dout_bus_big1************************#
set X_LOC $INPUT_OFFSET
set Y_LOC $HEIGHT_MUX_STRIPE
set PIN_COUNT [expr {($NUMBER_OF_MODULES*32)-1}]
set FLAG 0
set PINS_PER_BANK 1
set X_LOC [expr {$LEFT_OFFSET_TOP +$CHIP_OFFSET - $INPUT_PIN_SPACING}]
set BANK_NO 19

for {set i $PIN_COUNT} {$i > -1} {decr i 1} {
if {$PINS_PER_BANK == 33} {
  set TEMP_X_LOC [expr {$X_LOC +$INPUT_PIN_SPACING} ]
  set PIN_NAME "l_dout_bus_big\\\[$BANK_NO\\\]"
  puts $PIN_NAME
  preassignPin mux_instance $PIN_NAME -loc $TEMP_X_LOC $Y_LOC -layer 2
  decr BANK_NO 1
    if {$FLAG == 0} {
   set X_LOC [expr $X_LOC - {31*$INPUT_PIN_SPACING} + $DOUT_BANK_SPACING +
$INTER_MUX_DISTANCE_MIN]
   set FLAG 1
   } else {
  set X_LOC [expr $X_LOC - {31*$INPUT_PIN_SPACING} + $DOUT_BANK_SPACING +
$INTER_MUX_DISTANCE_MAX]
  set FLAG 0
   }
  set PINS_PER_BANK 2
 } else {
  set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
  incr PINS_PER_BANK 1
 }
set PIN_NAME "dout_bus_big\\\[$i\\\]"
puts "$PIN_NAME"
preassignPin mux_instance $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
}
 set X_LOC [expr {$X_LOC + $INPUT_PIN_SPACING}]
set PIN_NAME "l_dout_bus_big\\\[0\\\]"
puts $PIN_NAME
preassignPin mux_instance $PIN_NAME -loc $X_LOC $Y_LOC -layer 2

set X_LOC 0
# The Y location is set so that the pin is assigned near the top
set DIE_HEIGHT 103.2
set CLEARANCE [expr "50.4 + 1.2"]
set Y_LOC [expr {$DIE_HEIGHT - $CLEARANCE}]
set VERT_PIN_SPACING 0.4
set NUMBER_OF_MODULES 20

set X_LOC 0
set COUNT [expr {(($NUMBER_OF_MODULES *6)-1)}]
for {set i $COUNT} {$i > -1} {decr i 1} {
    set PIN_NAME "sel_mux_bus_big\\\[$i\\\]"
    puts "$PIN_NAME"
    preassignPin mux_instance $PIN_NAME -loc $X_LOC $Y_LOC -layer 5
    set Y_LOC [expr $Y_LOC - $VERT_PIN_SPACING]
}
```

C.5 FINAL FUX STRIPE PIN ASSIGNMENT SCRIPT

```
#Module related details
set NUMBER_OF_MODULES 20
set NUMBER_OF_FINALMUXES 20

set FIRST_INPUT_NAME "dout_bus_big6"
set SECOND_INPUT_NAME "dout_bus_big10"
set THIRD_INPUT_NAME "dout_bus_big14"
set FOURTH_INPUT_NAME "dout_bus_big18"

#Set input pins
set FINALMUX_WIDTH 500.8
#set FINALMUX_HEIGHT 14.4
set FINALMUX_STRIPE_HEIGHT 69.6
set FINALMUX_STRIPE_WIDTH 10330.0

set INTER_MODULE_SPACING_MIN 7.2
set INTER_MODULE_SPACING_MAX 22.4

set Y_LOC 0
set HOR_PIN_SPACING_TOP 2.0
set HOR_PIN_SPACING_BOTTOM 2.0
set VERT_PIN_SPACING 0.4

#*PLACING dout_bus_big6 dout_bus_big10 dout_bus_big14 dout_bus_big18*******#

set START_POSITION 24.0
for {set MOD_NUMBER 20} {$MOD_NUMBER >0} {decr MOD_NUMBER 1} {
  set X_LOC $START_POSITION
  set START_PIN_NUMBER [expr "$MOD_NUMBER*32"]
  set START_PIN_NUMBER [expr "$START_PIN_NUMBER -1"]

for {set i 0} {$i < 32} {incr i 1} {
   set PIN_NUMBER [expr "$START_PIN_NUMBER - $i"]
   set PIN_NAME "$FIRST_INPUT_NAME\\\[$PIN_NUMBER\\\]"
   puts "$PIN_NAME"
   preassignPin final_mux_stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
   set X_LOC [expr "$X_LOC + $HOR_PIN_SPACING_TOP"]
 }

set X_LOC [expr "$X_LOC + 7.2"]
for {set i 0} {$i < 32} {incr i 1} {
   set PIN_NUMBER [expr "$START_PIN_NUMBER - $i"]
   set PIN_NAME "$SECOND_INPUT_NAME\\\[$PIN_NUMBER\\\]"
   puts "$PIN_NAME"
   preassignPin final_mux_stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
   set X_LOC [expr "$X_LOC + $HOR_PIN_SPACING_TOP"]
 }
set X_LOC [expr "$X_LOC + 7.2"]
for {set i 0} {$i < 32} {incr i 1} {
   set PIN_NUMBER [expr "$START_PIN_NUMBER - $i"]
   set PIN_NAME "$THIRD_INPUT_NAME\\\[$PIN_NUMBER\\\]"
   puts "$PIN_NAME"
   preassignPin final_mux_stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
   set X_LOC [expr "$X_LOC + $HOR_PIN_SPACING_TOP"]
}
set X_LOC [expr "$X_LOC + 7.2"]
for {set i 0} {$i < 32} {incr i 1} {
   set PIN_NUMBER [expr "$START_PIN_NUMBER - $i"]
   set PIN_NAME "$FOURTH_INPUT_NAME\\\[$PIN_NUMBER\\\]"
   puts "$PIN_NAME"
   preassignPin final_mux_stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
   set X_LOC [expr "$X_LOC + $HOR_PIN_SPACING_TOP"]
 }

set val [expr "$MOD_NUMBER %2"]
if {$val == 0} {
   set START_POSITION [expr "$START_POSITION + $INTER_MODULE_SPACING_MIN +
$FINALMUX_WIDTH"]
   puts "MIN_SPACING\n"
} else {
   set START_POSITION [expr "$START_POSITION + $INTER_MODULE_SPACING_MAX +
$FINALMUX_WIDTH"]
   puts "MAX_SPACING\n"
}

}
#End of outer for loop

set START_POSITION 400
set Y_LOC $FINALMUX_STRIPE_HEIGHT
for {set MOD_NUMBER $NUMBER_OF_FINALMUXES} {$MOD_NUMBER >0} {decr MOD_NUMBER 1} {
   set X_LOC $START_POSITION
   set START_PIN_NUMBER [expr "$MOD_NUMBER*32"]
   set START_PIN_NUMBER [expr "$START_PIN_NUMBER -1"]

for {set i 0} {$i < 32} {incr i 1} {
   set PIN_NUMBER [expr "$START_PIN_NUMBER - $i"]
   set PIN_NAME "final_output\\\[$PIN_NUMBER\\\]"
   puts "$PIN_NAME"
   preassignPin final_mux_stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 2
   set X_LOC [expr "$X_LOC + $HOR_PIN_SPACING_BOTTOM"]
 }
set val [expr "$MOD_NUMBER %2"]
if {$val == 0} {
   set START_POSITION [expr "$START_POSITION + $INTER_MODULE_SPACING_MIN +
$FINALMUX_WIDTH"]
} else {
   set START_POSITION [expr "$START_POSITION + $INTER_MODULE_SPACING_MAX
+$FINALMUX_WIDTH"]

}
}
```

```
set X_LOC 0
set Y_LOC 41.2
for {set i 39} {$i > -1} {decr i 1} {
   set PIN_NAME "final_sel_bus\\\[$i\\\]"
   puts "$PIN_NAME"
   preassignPin final_mux_stripe $PIN_NAME -loc $X_LOC $Y_LOC -layer 5
   set Y_LOC [expr "$Y_LOC - $VERT_PIN_SPACING"]
}
```

APPENDIX D

Table D 1: Characterization data for 4ns delay element using 160fF buffer

Load Cap	Transitio n Time	Rise Energ y (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028ns	0.2195	0.2374	69.16	58.55	3.49	0.35	3.68	4.34	93.28	117.93
9.48fF	0.044ns	0.2216	0.2369	59.36	58.27	3.52	0.35	1.73	4.71	106.4	117.92
9.48fF	0.076ns	0.2201	0.2375	66.71	58.76	3.51	0.36	3.64	3.58	96.40	118.12
9.48fF	0.138ns	0.2222	0.2372	61.01	58.09	3.55	0.38	2.68	4.56	103.1	117.62
9.48fF	0.264ns	0.2242	0.2376	59.19	57.35	3.65	0.40	3.59	3.60	106.6	117.63
9.48fF	0.516ns	0.2282	0.2386	59.17	58.69	3.84	0.45	2.72	5.12	106.4	117.94
9.48fF	1.02ns	0.2277	0.2388	58.22	56.99	3.95	0.50	3.24	5.60	107.5	117.48
24.648fF	0.028ns	0.2538	0.2382	95.10	88.23	3.52	0.37	2.97	4.05	118.3	140.56
24.648fF	0.044ns	0.2561	0.2381	91.36	88.60	3.54	0.38	4.10	4.32	128.3	140.18
24.648fF	0.076ns	0.2544	0.2377	94.78	88.63	3.54	0.38	2.86	3.71	119.2	140.58
24.648fF	0.138ns	0.2568	0.2377	91.05	87.38	3.57	0.40	2.62	5.04	122.1	139.61
24.648fF	0.264ns	0.2582	0.2377	90.79	88.07	3.65	0.42	3.65	4.75	125.4	139.73
24.648fF	0.516ns	0.2621	0.2391	89.81	88.04	3.85	0.47	2.56	4.97	127.4	140.29
24.648fF	1.02ns	0.2620	0.2413	88.92	86.75	3.96	0.53	3.34	5.29	128.8	139.86
56.88fF	0.028ns	0.3250	0.2387	161.86	148.88	3.56	0.41	3.75	5.28	156.2	179.15
56.88fF	0.044ns	0.3263	0.2385	161.37	148.56	3.57	0.41	2.58	3.92	158.8	179.22
56.88fF	0.076ns	0.3260	0.2381	162.18	147.89	3.57	0.42	3.05	4.15	157.0	178.94
56.88fF	0.138ns	0.3280	0.2384	160.58	148.30	3.60	0.44	3.22	4.25	158.1	178.73
56.88fF	0.264ns	0.3302	0.2384	159.45	148.38	3.68	0.46	2.56	4.54	163.7	178.59
56.88fF	0.516ns	0.3342	0.2394	158.90	148.24	3.88	0.51	3.82	4.14	165.6	179.18
56.88fF	1.02ns	0.3346	0.2423	158.73	147.33	4.02	0.58	3.42	5.42	166.9	178.38
121.344fF	0.028ns	0.4668	0.2390	308.81	273.18	3.63	0.48	3.08	3.59	227.4	249.81
121.344fF	0.044ns	0.4682	0.2388	307.38	272.62	3.64	0.49	2.58	3.92	234.1	249.84
121.344fF	0.076ns	0.4692	0.2386	307.02	272.88	3.65	0.49	2.56	4.18	230.0	249.73
121.344fF	0.138ns	0.4687	0.2387	307.97	272.67	3.67	0.51	3.89	4.26	225.7	249.58
121.344fF	0.264ns	0.4723	0.2388	306.78	273.29	3.75	0.54	3.13	4.06	234.9	249.49
121.344fF	0.516ns	0.4773	0.2399	306.61	273.36	3.97	0.58	3.91	1.81	239.4	249.77
121.344fF	1.02ns	0.4778	0.2420	307.18	273.04	4.09	0.64	3.76	5.41	239.4	249.49
250.272fF	0.028ns	0.7558	0.2392	612.40	537.60	3.77	0.62	3.82	4.76	368.1	388.60
250.272fF	0.044ns	0.7595	0.2394	612.53	532.37	3.79	0.62	2.99	4.22	382.2	389.11
250.272fF	0.076ns	0.7576	0.2392	609.36	534.35	3.79	0.63	2.86	3.89	374.2	389.00
250.272fF	0.138ns	0.7582	0.2392	608.67	535.33	3.82	0.65	2.14	3.10	369.7	388.39
250.272fF	0.264ns	0.7628	0.2390	607.29	535.17	3.89	0.67	3.81	4.63	382.2	388.44
250.272fF	0.516ns	0.7663	0.2404	608.51	535.72	4.04	0.72	3.55	4.55	384.1	389.19
250.272fF	1.02ns	0.7659	0.2419	607.96	535.25	4.22	0.77	3.19	4.58	385.5	388.70
506.232fF	0.028ns	1.3316	0.2396	1218.50	1046.00	4.07	0.90	3.09	1.69	668.1	664.70
506.232fF	0.044ns	1.3347	0.2396	1208.83	1045.00	4.08	0.90	2.55	4.55	668.1	665.21

Table D 1 (Continued)

506.232fF	0.076ns	1.3337	0.2393	1211.17	1045.50	4.09	0.91	2.84	4.91	668.9	665.24
506.232fF	0.138ns	1.3348	0.2396	1209.58	1045.50	4.12	0.92	3.61	4.14	666.2	664.85
506.232fF	0.264ns	1.3348	0.2391	1211.08	1045.17	4.18	0.95	3.38	4.79	669.2	664.88
506.232fF	0.516ns	1.3406	0.2406	1211.50	1045.67	4.44	1.00	3.54	4.01	673.0	665.36
506.232fF	1.02ns	1.3390	0.2422	1210.08	1045.25	4.50	1.05	3.52	4.71	674.8	664.82
1023.84fF	0.028ns	2.5018	0.2396	2430.17	2082.08	4.65	1.46	3.01	3.84	1247.	1223.00
1023.84fF	0.044ns	2.5043	0.2392	2429.67	2081.25	4.66	1.46	2.65	5.30	1256.	1222.92
1023.84fF	0.076ns	2.5018	0.2390	2429.67	2082.50	4.67	1.46	1.75	5.20	1253.	1222.67
1023.84fF	0.138ns	2.5053	0.2396	2428.92	2080.17	4.70	1.48	2.31	3.47	1257.	1222.25
1023.84fF	0.264ns	2.5067	0.2395	2428.50	2082.00	4.77	1.51	1.79	4.65	1258.	1222.50
1023.84fF	0.516ns	2.5125	0.2405	2427.58	2082.58	5.03	1.56	2.76	5.37	1259.	1223.00
1023.84fF	1.02ns	2.5111	0.2418	2427.17	2082.17	5.08	1.60	3.60	5.12	1258.	1222.42

Table D 2: Characterization data for 4ns delay element using 640fF buffer

Load Cap	**Transition Time**	**Rise Energy (pJ)**	**Fall Energy (pJ)**	**Rise Time (ps)**	**Fall Time (ps)**	**Rise Cell Delay (ns)**	**Fall Cell Delay (ns)**	**Ileak-hi (nW)**	**Ileak-lo (nW)**	**Bufer rise delay (ps)**	**Buffer fall delay (ps)**
9.48fF	0.028ns	0.2813	0.3213	30.26	34.41	3.53	0.43	11.88	-9.74	129.2	200.60
9.48fF	0.044ns	0.2840	0.3209	29.73	34.53	3.56	0.44	-2.05	-5.75	146.3	200.27
9.48fF	0.076ns	0.2821	0.3215	30.54	35.17	3.55	0.44	0.49	-1.07	136.4	200.55
9.48fF	0.138ns	0.2809	0.3219	35.06	34.62	3.58	0.46	-1.88	18.79	131.3	200.28
9.48fF	0.264ns	0.2848	0.3226	29.82	35.47	3.66	0.49	2.23	-1.16	141.6	200.53
9.48fF	0.516ns	0.2854	0.3228	29.62	35.24	3.63	0.53	-3.93	-5.19	144.2	200.61
9.48fF	1.02ns	0.2890	0.3252	29.89	35.39	3.97	0.60	-6.22	1.87	146.0	200.85
24.648fF	0.028ns	0.3155	0.3237	39.63	41.82	3.54	0.44	2.53	-3.75	140.3	206.64
24.648fF	0.044ns	0.3186	0.3231	39.36	41.76	3.57	0.44	8.17	2.80	153.8	206.84
24.648fF	0.076ns	0.3172	0.3227	39.49	41.90	3.56	0.45	5.67	-6.56	142.9	207.17
24.648fF	0.138ns	0.3178	0.3224	40.05	41.82	3.60	0.46	-12.86	5.74	147.1	207.20
24.648fF	0.264ns	0.3195	0.3234	39.05	42.28	3.66	0.49	-8.81	-0.56	150.7	207.12
24.648fF	0.516ns	0.3231	0.3241	39.21	41.86	3.85	0.54	-0.17	5.45	153.0	207.10
24.648fF	1.02ns	0.3239	0.3265	39.14	42.11	4.00	0.60	-1.96	-5.27	153.6	207.04
56.88fF	0.028ns	0.3903	0.3237	60.47	54.86	3.56	0.45	-1.67	5.99	155.3	218.12
56.88fF	0.044ns	0.3914	0.3236	60.53	54.94	3.58	0.45	-3.13	9.28	165.1	217.83
56.88fF	0.076ns	0.3897	0.3244	60.46	54.79	3.57	0.46	-0.66	-0.40	151.0	218.10
56.88fF	0.138ns	0.3896	0.3245	61.21	54.99	3.60	0.48	-0.40	2.59	154.3	218.07
56.88fF	0.264ns	0.3919	0.3248	59.95	55.30	3.69	0.50	-3.63	9.15	162.5	218.01
56.88fF	0.516ns	0.3955	0.3253	59.94	54.88	3.86	0.55	0.75	-2.27	164.4	218.32
56.88fF	1.02ns	0.3972	0.3288	60.42	55.22	4.00	0.62	3.30	5.30	165.7	218.75
121.344fF	0.028ns	0.5344	0.3260	105.23	81.83	3.58	0.47	11.56	-	177.4	236.29
121.344fF	0.044ns	0.5350	0.3257	105.25	82.29	3.59	0.47	4.66	8.90	184.4	236.31
121.344fF	0.076ns	0.5356	0.3259	104.93	82.13	3.59	0.48	-4.60	-3.37	170.2	236.27
121.344fF	0.138ns	0.5336	0.3263	105.15	81.41	3.62	0.49	-1.19	5.12	179.2	236.43
121.344fF	0.264ns	0.5367	0.3260	104.96	81.52	3.71	0.52	8.65	5.26	184.5	236.65
121.344fF	0.516ns	0.5407	0.3273	104.52	82.20	3.93	0.57	-1.08	-3.38	186.7	236.33
121.344fF	1.02ns	0.5409	0.3286	104.33	81.85	4.00	0.63	-1.40	2.79	188.3	236.93
250.272fF	0.028ns	0.8217	0.3275	194.67	135.10	3.62	0.50	1.53	4.53	217.2	268.10

Table D 2 (Continued)

250.272fF	0.044ns	0.8259	0.3275	194.51	134.80	3.64	0.50	-3.54	1.66	229.8	267.88
250.272fF	0.076ns	0.8204	0.3272	194.90	134.73	3.64	0.51	3.95	4.67	220.8	267.97
250.272fF	0.138ns	0.8216	0.3275	194.58	134.52	3.68	0.53	6.76	-1.76	228.0	268.30
250.272fF	0.264ns	0.8243	0.3275	194.69	134.15	3.74	0.55	-9.42	7.49	227.0	268.49
250.272fF	0.516ns	0.8276	0.3280	194.62	134.19	3.93	0.60	-2.26	-4.49	233.2	268.21
250.272fF	1.02ns	0.8291	0.3301	194.78	133.72	4.07	0.66	6.37	5.34	233.6	268.85
506.232fF	0.028ns	1.4053	0.3293	379.60	239.82	3.71	0.56	-2.37	6.62	303.7	327.97
506.232fF	0.044ns	1.3999	0.3291	377.55	240.20	3.72	0.56	1.77	4.69	316.0	327.43
506.232fF	0.076ns	1.4026	0.3290	377.93	240.24	3.72	0.57	-1.72	2.08	308.3	327.67
506.232fF	0.138ns	1.3937	0.3286	377.64	240.90	3.76	0.59	-3.68	-6.67	311.1	328.15
506.232fF	0.264ns	1.4009	0.3289	378.52	240.51	3.84	0.62	-6.02	-1.61	317.8	328.34
506.232fF	0.516ns	1.4059	0.3299	378.99	240.73	4.08	0.66	-4.98	-3.57	320.3	327.98
506.232fF	1.02ns	1.4071	0.3323	378.47	240.50	4.17	0.73	4.19	-0.34	320.8	328.84
1023.84fF	0.028ns	2.5534	0.3293	748.43	459.73	3.89	0.68	4.06	7.19	487.4	447.34
1023.84fF	0.044ns	2.5505	0.3299	749.67	460.08	3.90	0.68	2.93	-2.98	496.5	447.23
1023.84fF	0.076ns	2.5492	0.3297	747.78	461.27	3.90	0.69	0.24	6.73	484.8	447.48
1023.84fF	0.138ns	2.5536	0.3295	745.13	461.70	3.95	0.71	3.71	0.51	498.9	448.00
1023.84fF	0.264ns	2.5520	0.3293	747.48	464.20	4.01	0.73	-6.59	-5.48	495.1	447.80
1023.84fF	0.516ns	2.5529	0.3307	747.93	462.15	4.21	0.78	-3.08	-3.85	497.7	447.71
1023.84fF	1.02ns	2.5534	0.3337	747.02	463.94	4.33	0.84	1.77	-0.03	498.5	448.39

Table D 3: Characterization data for 4ns delay element using 80fF buffer

Load Cap	Transition Time	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028ns	0.2136	0.2274	80.76	61.17	3.48	0.33	5.58	6.82	78.37	98.03
9.48fF	0.044ns	0.2151	0.2273	72.26	60.74	3.50	0.33	5.58	6.83	90.49	97.95
9.48fF	0.076ns	0.2145	0.2270	76.79	61.08	3.50	0.34	5.59	6.85	82.55	98.21
9.48fF	0.138ns	0.2147	0.2274	100.77	60.64	3.52	0.36	5.60	6.88	73.75	98.21
9.48fF	0.264ns	0.2162	0.2273	71.33	60.54	3.60	0.38	5.64	6.92	89.81	98.19
9.48fF	0.516ns	0.2200	0.2279	67.96	60.81	3.75	0.43	5.71	7.03	94.54	98.26
9.48fF	1.02ns	0.2208	0.2313	66.83	59.69	3.94	0.50	5.83	7.25	95.95	98.13
24.648fF	0.028ns	0.2469	0.2278	140.13	114.85	3.51	0.36	5.58	6.82	115.1	131.39
24.648fF	0.044ns	0.2484	0.2276	136.86	114.67	3.52	0.37	5.58	6.83	119.7	131.29
24.648fF	0.076ns	0.2484	0.2277	139.58	114.90	3.54	0.37	5.59	6.84	116.7	131.34
24.648fF	0.138ns	0.2490	0.2276	136.73	115.02	3.56	0.39	5.60	6.87	119.6	131.33
24.648fF	0.264ns	0.2519	0.2279	131.86	114.93	3.65	0.42	5.64	6.92	128.3	131.38
24.648fF	0.516ns	0.2537	0.2287	132.10	114.93	3.80	0.47	5.71	7.03	127.0	131.54
24.648fF	1.02ns	0.2555	0.2307	132.04	114.78	3.96	0.52	5.82	7.24	129.6	131.67
56.88fF	0.028ns	0.3191	0.2279	277.96	234.58	3.59	0.43	5.59	6.82	183.7	196.98
56.88fF	0.044ns	0.3202	0.2277	278.51	233.56	3.61	0.43	5.59	6.83	196.0	197.08
56.88fF	0.076ns	0.3203	0.2277	277.08	233.68	3.61	0.44	5.60	6.84	193.5	197.02
56.88fF	0.138ns	0.3210	0.2275	277.79	234.01	3.64	0.45	5.60	6.87	192.1	197.16
56.88fF	0.264ns	0.3230	0.2279	275.56	233.93	3.72	0.48	5.65	6.92	196.8	197.21
56.88fF	0.516ns	0.3272	0.2284	275.23	233.27	3.93	0.53	5.73	7.03	199.0	197.18
56.88fF	1.02ns	0.3266	0.2305	275.30	233.07	4.04	0.59	5.83	7.25	199.9	197.28
121.344fF	0.028ns	0.4647	0.2276	569.58	477.78	3.74	0.56	5.59	6.82	331.5	327.34
121.344fF	0.044ns	0.4651	0.2279	568.98	478.53	3.74	0.56	5.59	6.83	336.6	327.41

Table D 3 (Continued)

121.344fF	0.076ns	0.4634	0.2277	570.15	478.51	3.74	0.57	5.59	6.84	324.6	327.37
121.344fF	0.138ns	0.4653	0.2279	570.78	479.59	3.78	0.59	5.61	6.87	329.2	327.38
121.344fF	0.264ns	0.4681	0.2280	569.47	478.42	3.88	0.61	5.65	6.93	340.2	327.48
121.344fF	0.516ns	0.4706	0.2288	569.72	478.53	4.02	0.66	5.72	7.03	340.2	327.55
121.344fF	1.02ns	0.4711	0.2317	569.38	478.92	4.19	0.72	5.83	7.25	340.3	327.63
250.272fF	0.028ns	0.7527	0.2281	1158.67	968.94	4.01	0.82	5.58	6.82	610.8	587.51
250.272fF	0.044ns	0.7533	0.2281	1163.17	967.63	4.03	0.82	5.58	6.83	619.9	587.74
250.272fF	0.076ns	0.7546	0.2282	1160.92	970.78	4.04	0.83	5.60	6.84	617.6	587.50
250.272fF	0.138ns	0.7545	0.2283	1159.83	966.67	4.06	0.85	5.61	6.87	613.0	588.03
250.272fF	0.264ns	0.7566	0.2282	1162.17	966.90	4.15	0.87	5.64	6.92	622.6	587.78
250.272fF	0.516ns	0.7561	0.2292	1162.08	969.48	4.16	0.92	5.68	7.03	622.4	587.80
250.272fF	1.02ns	0.7592	0.2318	1161.50	969.47	4.48	0.98	5.83	7.25	626.2	587.73
506.232fF	0.028ns	1.3317	0.2281	2338.08	1920.17	4.57	1.34	5.59	6.82	1169.	1104.00
506.232fF	0.044ns	1.3328	0.2281	2333.50	1920.58	4.59	1.34	5.59	6.83	1183.	1104.08
506.232fF	0.076ns	1.3325	0.2279	2336.33	1920.67	4.59	1.35	5.59	6.84	1178.	1104.08
506.232fF	0.138ns	1.3319	0.2277	2337.08	1920.83	4.61	1.36	5.60	6.87	1168.	1103.83
506.232fF	0.264ns	1.3358	0.2283	2334.08	1921.08	4.71	1.39	5.64	6.92	1183.	1104.25
506.232fF	0.516ns	1.3369	0.2292	2333.58	1921.00	4.77	1.44	5.69	7.03	1185.	1104.25
506.232fF	1.02ns	1.3392	0.2303	2334.67	1920.92	5.01	1.49	5.82	7.25	1188.	1104.50
1023.84fF	0.028ns	2.4808	0.2277	4689.75	3877.83	5.71	2.38	8.42	6.82	2308.	2148.00
1023.84fF	0.044ns	2.4838	0.2279	4689.25	3878.67	5.73	2.38	8.47	6.83	2323.	2148.00
1023.84fF	0.076ns	2.4826	0.2281	4691.83	3878.25	5.73	2.39	8.37	6.84	2315.	2148.00
1023.84fF	0.138ns	2.4829	0.2281	4689.50	3878.42	5.76	2.41	8.50	6.87	2314.	2148.00
1023.84fF	0.264ns	2.4843	0.2280	4689.92	3877.50	5.83	2.43	8.63	6.92	2314.	2147.67
1023.84fF	0.516ns	2.4868	0.2292	4688.08	3878.50	5.96	2.48	9.01	7.03	2323.	2148.00
1023.84fF	1.02ns	2.4882	0.2310	4690.25	3878.75	6.17	2.53	9.54	7.25	2324.	2148.25

Table D 4: Characterization data for a 5ns delay element using 160fF buffer

Load Cap	Transition Time	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028ns	0.23	0.25	59.23	59.36	3.87	0.35	3.26	2.39	108.37	118.36
9.48fF	0.044ns	0.22	0.25	59.86	57.95	3.87	0.35	3.74	2.44	109.08	117.81
9.48fF	0.076ns	0.22	0.25	60.05	58.50	3.88	0.36	3.36	3.65	109.17	117.97
9.48fF	0.138ns	0.23	0.25	60.34	57.57	3.90	0.37	3.79	2.50	109.90	117.20
9.48fF	0.264ns	0.23	0.25	59.08	57.79	3.97	0.40	3.37	3.10	109.52	117.29
9.48fF	0.516ns	0.23	0.25	59.36	58.03	4.15	0.45	3.51	4.04	106.16	117.77
9.48fF	1.02ns	0.23	0.25	59.27	57.04	4.29	0.51	3.79	2.94	107.90	117.37
24.648fF	0.028ns	0.26	0.25	90.75	88.62	3.89	0.37	2.86	3.41	131.33	140.70
24.648fF	0.044ns	0.26	0.25	90.88	88.53	3.88	0.38	2.46	4.48	129.69	140.21
24.648fF	0.076ns	0.26	0.25	89.49	88.65	3.90	0.38	3.70	3.86	130.59	140.51
24.648fF	0.138ns	0.26	0.25	90.08	87.81	3.93	0.40	3.18	3.27	130.03	139.44
24.648fF	0.264ns	0.26	0.25	90.57	88.22	3.95	0.43	3.23	3.09	130.45	139.43
24.648fF	0.516ns	0.26	0.25	89.55	88.12	4.20	0.47	3.66	3.75	128.98	140.19
24.648fF	1.02ns	0.26	0.26	89.74	86.62	4.31	0.53	3.69	3.08	129.90	139.41

Table D 4 (Continued)

56.88fF	0.028ns	0.33	0.25	159.88	149.10	3.93	0.41	2.88	4.32	168.75	179.20
56.88fF	0.044ns	0.33	0.25	159.91	147.87	3.93	0.41	3.22	3.18	168.00	179.07
56.88fF	0.076ns	0.33	0.25	159.32	147.64	3.94	0.42	3.76	3.38	168.33	178.99
56.88fF	0.138ns	0.33	0.25	159.23	148.40	3.96	0.44	3.76	2.30	166.82	178.65
56.88fF	0.264ns	0.33	0.25	159.53	147.94	4.02	0.46	3.71	3.58	169.19	178.65
56.88fF	0.516ns	0.33	0.25	159.68	148.08	4.19	0.51	3.56	4.15	166.50	179.10
56.88fF	1.02ns	0.34	0.26	159.44	147.66	4.36	0.57	3.75	3.94	168.18	178.48
121.344fF	0.028ns	0.47	0.25	307.47	272.90	4.00	0.48	3.93	2.56	241.13	249.93
121.344fF	0.044ns	0.47	0.25	306.30	274.03	4.00	0.49	2.94	2.93	240.75	249.88
121.344fF	0.076ns	0.47	0.25	306.63	273.38	4.01	0.49	4.49	3.17	239.78	249.88
121.344fF	0.138ns	0.47	0.25	306.69	272.43	4.04	0.51	2.83	3.70	241.12	249.73
121.344fF	0.264ns	0.47	0.25	306.68	273.36	4.11	0.53	2.56	3.43	240.83	249.31
121.344fF	0.516ns	0.48	0.25	306.30	273.43	4.37	0.58	3.40	2.76	239.41	249.80
121.344fF	1.02ns	0.48	0.26	306.95	273.23	4.43	0.64	3.87	3.55	240.51	249.33
250.272fF	0.028ns	0.76	0.25	608.93	531.50	4.15	0.62	3.50	1.53	387.13	389.20
250.272fF	0.044ns	0.76	0.25	608.67	533.90	4.14	0.62	3.64	3.80	386.77	389.28
250.272fF	0.076ns	0.76	0.25	608.92	536.78	4.15	0.63	2.88	4.05	382.37	388.67
250.272fF	0.138ns	0.76	0.25	609.58	530.93	4.18	0.65	2.86	2.49	384.24	388.43
250.272fF	0.264ns	0.77	0.25	610.36	534.90	4.26	0.67	3.15	2.95	387.13	388.64
250.272fF	0.516ns	0.77	0.26	608.04	535.37	4.50	0.72	3.93	3.94	383.12	389.13
250.272fF	1.02ns	0.77	0.26	610.47	535.43	4.59	0.78	3.64	2.93	384.12	388.51
506.232fF	0.028ns	1.34	0.25	1210.5	1043.92	4.44	0.90	2.56	4.88	676.08	665.62
506.232fF	0.044ns	1.34	0.25	1210.6	1044.50	4.43	0.90	3.93	3.09	676.48	665.33
506.232fF	0.076ns	1.34	0.25	1210.5	1043.83	4.45	0.91	3.95	3.38	676.08	664.82
506.232fF	0.138ns	1.34	0.25	1210.5	1044.92	4.47	0.92	4.02	2.97	673.22	664.58
506.232fF	0.264ns	1.34	0.25	1211.7	1044.42	4.52	0.95	4.18	3.62	675.90	664.98
506.232fF	0.516ns	1.34	0.25	1209.4	1044.50	4.67	1.00	3.67	2.84	672.18	665.31
506.232fF	1.02ns	1.34	0.26	1213.4	1045.08	4.86	1.06	3.22	3.41	675.93	664.83
1023.84fF	0.028ns	2.51	0.25	2429.5	2081.17	5.02	1.45	2.74	3.32	1261.50	1222.8
1023.84fF	0.044ns	2.51	0.25	2429.6	2080.75	5.02	1.46	2.00	2.18	1262.00	1223.0
1023.84fF	0.076ns	2.51	0.25	2430.7	2080.33	5.03	1.46	3.67	2.58	1260.92	1222.4
1023.84fF	0.138ns	2.51	0.25	2429.4	2082.67	5.06	1.48	3.99	3.75	1256.00	1222.4
1023.84fF	0.264ns	2.51	0.25	2430.5	2082.00	5.13	1.51	3.23	4.82	1260.17	1222.3
1023.84fF	0.516ns	2.51	0.26	2429.1	2081.00	5.32	1.56	3.10	2.85	1257.92	1222.9
1023.84fF	1.02ns	2.51	0.26	2432.0	2082.42	5.46	1.62	3.41	3.99	1261.67	1222.0

Table D 5: Characterization data for a 5ns delay element using 640fF buffer

Load Cap (fF)	Tran sition Time (ns)	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48	0.028	0.2886	0.3393	29.81	34.43	4.59	0.43	-15.7	-8.73	148.47	200.50
9.48	0.044	0.2885	0.3387	30.13	34.55	4.59	0.44	-2.66	-5.00	145.73	200.22
9.48	0.076	0.2893	0.3390	29.58	35.00	4.61	0.44	-12.8	7.29	148.19	200.56
9.48	0.138	0.2885	0.3404	29.97	35.00	4.63	0.46	-6.43	4.32	144.72	200.18
9.48	0.264	0.2884	0.3413	31.94	36.14	4.69	0.49	-6.77	-21.60	133.34	200.86

Table D 5 (Continued)

9.48	0.516	0.2939	0.3408	29.83	34.67	4.95	0.53	-5.29	1.57	147.64	200.31
9.48	1.02	0.2938	0.3452	29.88	35.35	5.03	0.60	-5.49	-5.21	148.13	200.88
24.648	0.028	0.3235	0.3417	39.33	41.74	4.60	0.44	-8.45	-0.82	155.18	206.90
24.648	0.044	0.3233	0.3404	38.96	41.61	4.60	0.44	-12.1	0.13	154.88	206.93
24.648	0.076	0.3230	0.3408	40.38	41.75	4.61	0.45	-13.1	-5.17	151.83	206.78
24.648	0.138	0.3222	0.3415	40.21	42.09	4.63	0.46	-5.70	-12.34	151.09	206.65
24.648	0.264	0.3224	0.3413	40.33	41.89	4.70	0.49	-4.91	-7.46	145.94	207.22
24.648	0.516	0.3279	0.3417	39.26	41.82	4.93	0.54	3.66	-7.93	154.28	207.00
24.648	1.02	0.3281	0.3449	39.29	41.76	5.02	0.60	-2.78	-1.80	153.38	207.33
56.88	0.028	0.3964	0.3424	60.80	55.10	4.61	0.45	-7.42	5.25	167.23	217.82
56.88	0.044	0.3972	0.3417	60.50	54.93	4.61	0.45	-4.94	-5.68	166.81	217.82
56.88	0.076	0.3963	0.3419	60.56	55.07	4.62	0.46	-2.64	-3.91	165.14	217.97
56.88	0.138	0.3963	0.3428	60.96	55.02	4.65	0.48	-15.0	-5.98	164.36	217.87
56.88	0.264	0.3977	0.3430	60.39	55.05	4.71	0.51	-3.47	-6.93	164.93	218.13
56.88	0.516	0.3997	0.3433	59.98	54.98	4.89	0.55	3.38	-3.84	163.20	217.99
56.88	1.02	0.4007	0.3467	60.49	55.35	5.02	0.61	3.83	5.19	166.02	218.59
121.344	0.028	0.5399	0.3443	105.23	81.73	4.63	0.47	-11.3	6.21	187.80	236.30
121.344	0.044	0.5396	0.3444	104.67	82.13	4.63	0.47	-17.2	-11.51	186.85	236.33
121.344	0.076	0.5391	0.3441	104.99	81.96	4.64	0.48	-13.4	-13.86	183.08	236.34
121.344	0.138	0.5390	0.3448	104.96	80.94	4.66	0.49	-2.02	-5.91	172.88	236.45
121.344	0.264	0.5401	0.3448	104.80	81.64	4.74	0.52	-1.05	-3.30	187.15	236.63
121.344	0.516	0.5454	0.3452	104.67	81.93	4.98	0.57	2.08	-6.67	188.36	236.48
121.344	1.02	0.5454	0.3481	104.68	81.83	5.09	0.63	0.72	-4.29	188.97	236.88
250.272	0.028	0.8276	0.3460	194.12	134.27	4.68	0.50	-11.9	11.91	234.44	268.52
250.272	0.044	0.8308	0.3457	194.65	134.81	4.68	0.50	-4.01	11.89	234.55	267.86
250.272	0.076	0.8276	0.3454	194.53	134.89	4.69	0.51	-1.51	5.25	234.05	267.96
250.272	0.138	0.8256	0.3455	194.44	133.60	4.72	0.53	-9.59	6.57	233.83	268.67
250.272	0.264	0.8278	0.3459	194.33	133.98	4.79	0.55	-11.6	-0.66	229.23	268.74
250.272	0.516	0.8319	0.3469	194.60	133.47	5.01	0.60	-6.88	0.53	233.09	268.36
250.272	1.02	0.8296	0.3495	194.43	133.78	5.08	0.67	-3.59	0.60	230.58	269.14
506.232	0.028	1.4012	0.3476	376.16	240.08	4.77	0.56	-7.19	13.20	322.46	327.68
506.232	0.044	1.4062	0.3474	380.70	240.13	4.76	0.56	-6.15	-5.21	321.98	327.41
506.232	0.076	1.4055	0.3473	379.37	240.53	4.78	0.57	0.25	-5.37	319.94	327.72
506.232	0.138	1.4012	0.3473	378.05	240.49	4.80	0.59	-14.6	-4.80	318.33	328.17
506.232	0.264	1.3999	0.3475	377.08	240.42	4.88	0.62	-7.72	-3.29	317.21	328.31
506.232	0.516	1.4098	0.3484	378.50	240.62	5.16	0.66	-9.73	-2.86	322.02	327.87
506.232	1.02	1.4078	0.3515	378.19	240.06	5.18	0.73	-0.78	-3.96	318.03	328.67
1023.84	0.028	2.5498	0.3480	748.90	460.93	4.94	0.68	-2.76	-21.33	500.31	447.31
1023.84	0.044	2.5539	0.3482	745.98	460.11	4.94	0.68	-7.32	-0.81	493.03	447.24
1023.84	0.076	2.5517	0.3480	749.88	460.94	4.96	0.69	-8.80	-0.50	500.14	447.44
1023.84	0.138	2.5506	0.3475	748.22	462.73	4.98	0.71	-3.33	3.21	498.84	447.78
1023.84	0.264	2.5508	0.3483	748.08	463.57	5.05	0.73	-3.92	-2.06	499.23	447.90
1023.84	0.516	2.5584	0.3491	749.33	461.40	5.32	0.78	-5.24	-2.58	499.68	447.61
1023.84	1.02	2.5583	0.3533	748.41	464.42	5.36	0.85	5.20	-3.65	495.97	448.08

Table D 6: Characterization data for a 5ns delay element using 80fF buffer

Load Cap	Transition Time	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028ns	0.2206	0.238	68.94	61.31	4.54	0.33	6.05	4.45	97.43	98.22
9.48fF	0.044ns	0.2206	0.238	68.57	61.28	4.54	0.33	6.06	4.46	96.74	97.93
9.48fF	0.076ns	0.2208	0.238	69.47	61.48	4.55	0.34	6.07	4.47	97.22	98.17
9.48fF	0.138ns	0.2211	0.238	70.36	60.95	4.58	0.36	6.08	4.48	96.81	98.33
9.48fF	0.264ns	0.2217	0.238	76.31	60.43	4.65	0.38	6.13	4.52	88.50	98.28
9.48fF	0.516ns	0.2262	0.239	69.70	60.90	4.89	0.43	6.24	4.60	96.77	98.09
9.48fF	1.02ns	0.2249	0.242	73.17	59.91	4.97	0.49	6.35	4.75	92.86	98.14
24.648fF	0.028ns	0.2548	0.239	131.43	114.68	4.57	0.36	6.06	4.45	131.53	131.29
24.648fF	0.044ns	0.2547	0.239	132.40	114.50	4.57	0.37	6.05	4.46	131.00	131.30
24.648fF	0.076ns	0.2549	0.239	132.38	114.65	4.59	0.37	6.07	4.47	130.96	131.28
24.648fF	0.138ns	0.2548	0.238	132.60	115.10	4.61	0.39	6.08	4.49	131.04	131.40
24.648fF	0.264ns	0.2558	0.238	134.48	115.03	4.69	0.42	6.12	4.52	128.31	131.45
24.648fF	0.516ns	0.2603	0.240	132.51	114.73	4.94	0.47	6.24	4.59	131.62	131.49
24.648fF	1.02ns	0.2590	0.242	133.95	114.57	4.98	0.53	6.34	4.74	128.96	131.61
56.88fF	0.028ns	0.3261	0.239	277.58	234.95	4.65	0.43	6.05	4.45	202.09	196.84
56.88fF	0.044ns	0.3256	0.238	274.20	233.50	4.64	0.43	6.06	4.46	201.70	196.80
56.88fF	0.076ns	0.3261	0.238	276.89	234.27	4.66	0.44	6.07	4.47	201.91	196.92
56.88fF	0.138ns	0.3257	0.239	276.87	233.86	4.68	0.45	6.08	4.49	201.35	196.94
56.88fF	0.264ns	0.3274	0.239	276.95	233.38	4.76	0.48	6.12	4.52	199.87	197.03
56.88fF	0.516ns	0.3292	0.240	276.01	234.00	4.89	0.53	6.20	4.59	201.83	197.24
56.88fF	1.02ns	0.3306	0.243	276.64	233.56	5.05	0.60	6.33	4.75	200.28	197.28
121.344fF	0.028ns	0.4707	0.239	569.23	475.87	4.78	0.56	6.05	4.45	343.62	327.31
121.344fF	0.044ns	0.4707	0.239	569.63	476.98	4.79	0.56	6.05	4.46	343.40	327.33
121.344fF	0.076ns	0.4708	0.239	569.89	477.38	4.80	0.57	6.07	4.47	343.26	327.37
121.344fF	0.138ns	0.4709	0.239	570.13	479.33	4.83	0.58	6.09	4.49	341.08	327.35
121.344fF	0.264ns	0.4714	0.239	570.23	479.12	4.87	0.61	6.12	4.52	341.97	327.25
121.344fF	0.516ns	0.4768	0.240	569.89	478.24	5.17	0.66	6.25	4.59	342.46	327.58
121.344fF	1.02ns	0.4754	0.242	569.37	479.17	5.20	0.72	6.35	4.75	339.33	327.46
250.272fF	0.028ns	0.7620	0.239	1160.0	963.83	5.07	0.82	6.06	4.45	627.22	587.48
250.272fF	0.044ns	0.7584	0.239	1165.6	963.63	5.07	0.82	6.05	4.46	627.33	587.53
250.272fF	0.076ns	0.7594	0.239	1161.9	962.83	5.07	0.83	6.07	4.47	613.89	587.56
250.272fF	0.138ns	0.7600	0.239	1162.1	963.16	5.11	0.85	6.09	4.48	623.44	587.85
250.272fF	0.264ns	0.7621	0.239	1162.0	963.18	5.17	0.87	6.12	4.52	626.70	587.81
250.272fF	0.516ns	0.7663	0.240	1162.2	962.93	5.40	0.92	6.24	4.59	627.32	587.61
250.272fF	1.02ns	0.7656	0.242	1161.5	963.05	5.50	0.98	6.35	4.74	627.06	587.58
506.232fF	0.028ns	1.3398	0.239	2336.8	1921.50	5.63	1.34	6.06	4.45	1188.25	1103.9
506.232fF	0.044ns	1.3391	0.239	2337.0	1922.08	5.63	1.34	6.06	4.46	1188.25	1104.0
506.232fF	0.076ns	1.3387	0.239	2335.0	1921.00	5.64	1.35	6.07	4.47	1185.17	1104.0
506.232fF	0.138ns	1.3391	0.239	2337.5	1921.50	5.67	1.36	6.08	4.48	1184.58	1104.0
506.232fF	0.264ns	1.3404	0.239	2335.1	1921.42	5.74	1.39	6.12	4.52	1186.83	1104.0
506.232fF	0.516ns	1.3446	0.239	2332.4	1921.00	5.95	1.44	6.23	4.59	1188.67	1104.0
506.232fF	1.02ns	1.3438	0.242	2333.5	1920.92	6.02	1.49	6.34	4.74	1183.92	1104.0
1023.84fF	0.028ns	2.4885	0.239	4686.3	3881.92	6.77	2.38	9.96	4.45	2325.83	2147.9
1023.84fF	0.044ns	2.4885	0.238	4694.6	3882.00	6.77	2.38	9.97	4.46	2325.00	2148.0
1023.84fF	0.076ns	2.4885	0.239	4688.0	3882.25	6.78	2.39	9.97	4.47	2324.92	2148.0
1023.84fF	0.138ns	2.4885	0.239	4686.0	3879.00	6.81	2.41	10.07	4.48	2325.00	2148.0
1023.84fF	0.264ns	2.4900	0.239	4691.3	3882.00	6.86	2.43	10.19	4.52	2325.33	2147.8
1023.84fF	0.516ns	2.4934	0.240	4693.2	3882.25	7.06	2.48	10.65	4.59	2323.58	2148.0
1023.84fF	1.02ns	2.4921	0.242	4694.7	3881.25	7.17	2.54	11.22	4.75	2323.83	2148.0

Table D 7: Characterization data for a 7ns delay element using 160fF buffer

Load Cap	Transiti on Time	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Dela y (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufe r rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028ns	0.2385	0.2797	69.53	59.16	6.49	0.35	5.25	1.12	105.0	118.20
9.48fF	0.044ns	0.2384	0.2797	67.32	59.11	6.49	0.35	3.14	0.47	106.1	118.08
9.48fF	0.076ns	0.2390	0.2792	68.57	58.53	6.49	0.36	3.60	1.10	104.3	118.01
9.48fF	0.138ns	0.2395	0.2788	66.04	57.79	6.52	0.37	4.73	1.59	106.1	117.49
9.48fF	0.264ns	0.2396	0.2789	70.52	57.62	6.56	0.40	4.31	1.31	102.0	117.13
9.48fF	0.516ns	0.2438	0.2807	66.68	58.06	6.79	0.45	4.21	0.71	104.1	117.57
9.48fF	1.02ns	0.2433	0.2828	65.90	57.25	6.89	0.50	4.14	1.63	107.0	117.76
24.648fF	0.028ns	0.2717	0.2806	101.88	88.55	6.50	0.37	5.02	2.08	122.7	140.63
24.648fF	0.044ns	0.2718	0.2802	98.04	88.57	6.51	0.38	4.36	1.58	128.1	140.17
24.648fF	0.076ns	0.2716	0.2799	101.26	88.49	6.51	0.38	4.54	1.38	124.0	140.28
24.648fF	0.138ns	0.2726	0.2795	97.82	87.45	6.53	0.40	4.18	1.44	123.6	139.45
24.648fF	0.264ns	0.2738	0.2802	95.53	87.96	6.61	0.43	3.46	0.85	127.6	139.50
24.648fF	0.516ns	0.2779	0.2815	95.08	87.63	6.85	0.47	3.76	0.80	127.6	139.87
24.648fF	1.02ns	0.2767	0.2834	96.32	86.98	6.89	0.53	4.15	1.34	124.5	139.93
56.88fF	0.028ns	0.3429	0.2812	165.73	148.2	6.55	0.41	4.90	0.78	160.4	179.03
56.88fF	0.044ns	0.3442	0.2811	163.13	148.0	6.54	0.41	4.00	1.11	165.9	179.08
56.88fF	0.076ns	0.3414	0.2809	170.67	148.4	6.54	0.42	4.10	1.76	154.1	179.05
56.88fF	0.138ns	0.3436	0.2805	164.20	148.4	6.57	0.44	3.86	1.16	162.7	178.67
56.88fF	0.264ns	0.3454	0.2808	161.91	147.8	6.64	0.47	4.18	1.41	166.6	178.37
56.88fF	0.516ns	0.3488	0.2820	163.24	148.7	6.85	0.51	4.45	1.86	164.9	178.96
56.88fF	1.02ns	0.3504	0.2829	160.38	147.8	6.95	0.56	3.95	1.34	169.1	178.65
121.344f	0.028ns	0.4875	0.2814	308.79	272.5	6.62	0.48	3.70	1.26	234.6	249.86
121.344f	0.044ns	0.4858	0.2814	308.00	273.2	6.61	0.49	4.48	1.16	237.4	249.86
121.344f	0.076ns	0.4863	0.2811	310.15	273.3	6.62	0.49	3.94	1.65	232.6	249.82
121.344f	0.138ns	0.4877	0.2811	308.90	272.0	6.64	0.51	3.56	1.40	237.2	249.25
121.344f	0.264ns	0.4882	0.2811	308.49	272.9	6.69	0.53	4.69	1.43	238.9	249.24
121.344f	0.516ns	0.4917	0.2828	307.83	272.6	6.85	0.59	3.56	1.15	239.9	249.61
121.344f	1.02ns	0.4914	0.2846	308.61	273.3	7.01	0.64	4.32	0.90	236.3	249.33
250.272f	0.028ns	0.7756	0.2819	608.58	532.9	6.77	0.62	4.03	1.82	380.0	389.22
250.272f	0.044ns	0.7761	0.2819	608.01	531.8	6.76	0.62	4.25	1.43	385.7	389.16
250.272f	0.076ns	0.7741	0.2816	609.25	535.3	6.76	0.63	4.44	1.21	375.1	389.19
250.272f	0.138ns	0.7766	0.2816	606.99	533.0	6.79	0.65	4.21	0.85	383.5	388.51
250.272f	0.264ns	0.7772	0.2817	611.08	537.1	6.85	0.68	4.37	1.68	382.8	388.64
250.272f	0.516ns	0.7779	0.2830	610.42	534.2	6.87	0.72	4.76	0.94	381.0	388.78
250.272f	1.02ns	0.7813	0.2863	610.01	534.7	7.17	0.79	5.40	1.29	383.3	388.75
506.232f	0.028ns	1.3477	0.2822	1211.9	1040.	7.05	0.90	3.99	0.88	670.6	665.31
506.232f	0.044ns	1.3446	0.2823	1208.0	1041.	7.05	0.90	2.30	-1.55	675.4	665.10
506.232f	0.076ns	1.3493	0.2816	1210.0	1041.	7.06	0.91	5.34	1.08	671.4	665.13
506.232f	0.138ns	1.3516	0.2815	1209.9	1042.	7.08	0.92	4.53	1.19	673.6	664.23
506.232f	0.264ns	1.3530	0.2818	1211.5	1042.	7.15	0.95	4.70	1.28	672.6	664.83
506.232f	0.516ns	1.3518	0.2836	1211.2	1040.	7.16	1.00	4.39	1.03	673.2	665.24
506.232f	1.02ns	1.3531	0.2862	1211.1	1041.	7.46	1.06	4.70	1.66	669.2	664.76
1023.84f	0.028ns	2.5186	0.2822	2429.2	2081.	7.64	1.45	3.96	1.17	1251.	1222.75
1023.84f	0.044ns	2.5208	0.2822	2428.5	2080.	7.64	1.46	3.21	0.32	1262.	1223.00
1023.84f	0.076ns	2.5180	0.2817	2428.0	2081.	7.64	1.47	4.73	1.04	1252.	1222.75
1023.84f	0.138ns	2.5207	0.2816	2428.6	2081.	7.66	1.48	3.95	1.11	1258.	1222.25
1023.84f	0.264ns	2.5224	0.2815	2428.4	2081.	7.73	1.51	4.82	1.09	1259.	1222.08
1023.84f	0.516ns	2.5261	0.2836	2425.6	2081.	7.87	1.56	3.74	0.62	1263.	1222.42
1023.84f	1.02ns	2.5269	0.2848	2428.5	2080.	8.06	1.62	5.48	1.01	1258.	1221.92

Table D 8: Characterization data for a 7ns delay element using 640fF buffer

Load Cap	Transition Time	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028n	0.297	0.381	33.7	34.96	6.52	0.43	4.04	-0.71	137.59	200.4
9.48fF	0.044n	0.297	0.382	32.1	35.85	6.52	0.44	6.56	-6.0	142.68	200.9
9.48fF	0.076n	0.298	0.380	34.1	34.39	6.52	0.44	5.79	-5.46	132.90	200.3
9.48fF	0.138n	0.299	0.382	30.3	34.71	6.55	0.46	4.72	3.07	144.93	200.2
9.48fF	0.264n	0.300	0.382	31.0	35.53	6.61	0.49	8.16	2.63	144.93	200.3
9.48fF	0.516n	0.304	0.384	30.0	35.18	6.72	0.54	-4.15	-1.39	149.14	200.1
9.48fF	1.02ns	0.304	0.385	31.1	35.35	6.93	0.60	-1.73	-0.51	143.45	200.8
24.648	0.028n	0.330	0.383	40.5	41.65	6.53	0.44	10.75	-4.94	147.00	206.9
24.648	0.044n	0.331	0.383	42.0	41.73	6.52	0.44	5.57	-5.20	147.01	207.2
24.648	0.076n	0.331	0.383	41.4	41.72	6.53	0.45	9.57	-4.65	148.06	206.7
24.648	0.138n	0.333	0.383	43.5	41.96	6.55	0.46	-4.76	-1.64	142.13	206.8
24.648	0.264n	0.333	0.383	41.8	41.65	6.61	0.49	1.07	0.67	147.73	206.8
24.648	0.516n	0.338	0.384	39.6	41.65	6.81	0.54	-6.16	-5.13	153.18	206.9
24.648	1.02ns	0.338	0.387	41.7	42.09	6.96	0.60	3.38	8.55	148.87	207.1
56.88f	0.028n	0.400	0.384	61.2	54.93	6.54	0.45	14.48	-9.49	156.03	217.8
56.88f	0.044n	0.403	0.385	59.8	54.93	6.54	0.45	-0.92	-8.58	163.58	218.4
56.88f	0.076n	0.402	0.384	61.7	54.86	6.54	0.46	9.20	0.79	156.53	217.9
56.88f	0.138n	0.405	0.385	60.6	54.88	6.57	0.48	4.22	-5.24	163.06	218.0
56.88f	0.264n	0.404	0.385	62.0	55.08	6.61	0.50	-1.86	-1.57	159.64	217.9
56.88f	0.516n	0.410	0.386	60.0	55.17	6.81	0.55	5.39	-7.96	166.03	218.0
56.88f	1.02ns	0.409	0.388	62.2	55.31	6.95	0.60	-4.27	-6.16	158.41	218.2
121.34	0.028n	0.546	0.386	104.	81.83	6.57	0.47	15.26	3.92	180.01	236.1
121.34	0.044n	0.549	0.386	104.	82.34	6.56	0.47	7.82	0.96	186.88	236.4
121.34	0.076n	0.548	0.386	105.	82.47	6.57	0.48	13.28	-1.05	183.53	236.3
121.34	0.138n	0.548	0.387	105.	81.50	6.59	0.49	10.74	-1.84	180.00	236.6
121.34	0.264n	0.550	0.387	105.	82.11	6.66	0.52	6.75	-7.26	182.24	236.3
121.34	0.516n	0.554	0.388	105.	81.82	6.80	0.57	0.13	-2.41	184.44	236.3
121.34	1.02ns	0.555	0.391	104.	81.76	6.99	0.63	-9.23	-1.66	183.71	236.7
250.27	0.028n	0.834	0.389	195.	135.12	6.61	0.50	16.71	4.98	225.71	267.8
250.27	0.044n	0.834	0.389	194.	134.66	6.61	0.50	25.62	-3.13	226.98	268.1
250.27	0.076n	0.832	0.388	195.	134.29	6.62	0.51	1.03	-4.98	223.88	268.1
250.27	0.138n	0.842	0.388	193.	133.72	6.64	0.53	-2.06	0.91	232.29	268.3
250.27	0.264n	0.837	0.388	195.	133.78	6.71	0.55	4.19	-5.20	227.11	268.6
250.27	0.516n	0.840	0.390	195.	134.30	6.81	0.60	-7.38	-6.56	232.28	268.2
250.27	1.02ns	0.841	0.392	195.	133.34	6.99	0.66	-2.45	-4.23	225.52	269.1
506.23	0.028n	1.408	0.390	376.	240.39	6.70	0.56	-1.74	-3.92	317.38	327.7
506.23	0.044n	1.410	0.390	380.	240.22	6.69	0.56	13.89	-9.41	311.65	328.0
506.23	0.076n	1.404	0.389	377.	240.38	6.69	0.57	9.35	-4.53	309.46	327.8
506.23	0.138n	1.412	0.390	377.	240.38	6.73	0.58	-0.56	-6.94	318.08	327.9
506.23	0.264n	1.405	0.390	376.	240.15	6.76	0.62	-2.97	2.35	313.60	328.4
506.23	0.516n	1.413	0.391	379.	240.37	6.88	0.66	-0.24	-4.48	318.85	327.5
506.23	1.02ns	1.415	0.394	377.	240.25	7.10	0.72	-0.52	2.28	319.32	328.7
1023.8	0.028n	2.558	0.391	748.	462.08	6.87	0.68	1.27	10.06	496.41	447.3
1023.8	0.044n	2.559	0.392	748.	461.23	6.87	0.68	7.92	1.94	493.83	447.7
1023.8	0.076n	2.558	0.390	747.	462.27	6.87	0.69	15.13	-3.86	485.15	447.3
1023.8	0.138n	2.562	0.391	746.	462.17	6.90	0.71	2.31	1.69	491.92	447.5
1023.8	0.264n	2.560	0.391	749.	463.63	6.95	0.73	8.59	-7.02	490.86	447.7
1023.8	0.516n	2.567	0.392	748.	462.49	7.19	0.78	-7.89	-1.96	499.97	447.4
1023.8	1.02ns	2.570	0.395	748.	463.05	7.28	0.84	-3.09	-2.17	491.78	448.2

Table D 9: Characterization data for a 7ns delay element using 80fF buffer

Load Cap	Transition Time	Rise Energy (pJ)	Fall Energy (pJ)	Rise Time (ps)	Fall Time (ps)	Rise Cell Delay (ns)	Fall Cell Delay (ns)	Ileak-hi (nW)	Ileak-lo (nW)	Bufer rise delay (ps)	Buffer fall delay (ps)
9.48fF	0.028n	0.231	0.267	82.9	61.24	6.47	0.33	6.76	3.27	86.79	98.21
9.48fF	0.044n	0.231	0.267	82.4	60.87	6.47	0.33	6.77	3.28	90.31	98.08
9.48fF	0.076n	0.231	0.267	78.5	61.07	6.48	0.34	6.76	3.28	91.38	98.11
9.48fF	0.138n	0.232	0.266	74.9	60.80	6.50	0.36	6.78	3.30	93.74	98.35
9.48fF	0.264n	0.232	0.266	83.8	60.45	6.54	0.38	6.83	3.33	89.87	98.40
9.48fF	0.516n	0.237	0.268	71.9	60.73	6.79	0.43	6.95	3.38	95.90	98.09
9.48fF	1.02ns	0.237	0.271	76.8	59.80	6.88	0.50	7.09	3.50	92.48	98.18
24.648	0.028n	0.265	0.268	138.	114.71	6.51	0.36	6.76	3.27	124.75	131.2
24.648	0.044n	0.265	0.267	140.	115.08	6.50	0.37	6.76	3.28	122.26	131.2
24.648	0.076n	0.265	0.267	138.	114.88	6.51	0.37	6.78	3.28	124.06	131.3
24.648	0.138n	0.265	0.267	137.	115.15	6.53	0.39	6.78	3.30	126.17	131.4
24.648	0.264n	0.266	0.267	135.	114.84	6.59	0.42	6.83	3.33	128.71	131.2
24.648	0.516n	0.269	0.268	133.	114.57	6.70	0.47	6.91	3.38	131.48	131.3
24.648	1.02ns	0.269	0.271	141.	114.64	6.91	0.52	7.09	3.49	122.41	131.6
56.88f	0.028n	0.337	0.268	278.	234.88	6.58	0.43	6.77	3.28	194.48	196.8
56.88f	0.044n	0.337	0.267	277.	232.97	6.58	0.43	6.77	3.28	199.43	197.0
56.88f	0.076n	0.335	0.267	284.	233.78	6.57	0.44	6.77	3.28	181.33	196.8
56.88f	0.138n	0.337	0.268	278.	234.43	6.61	0.45	6.78	3.30	200.74	196.9
56.88f	0.264n	0.338	0.267	279.	233.28	6.66	0.48	6.83	3.33	192.92	196.9
56.88f	0.516n	0.339	0.268	281.	233.76	6.75	0.53	6.91	3.38	189.88	197.1
56.88f	1.02ns	0.341	0.271	278.	233.42	6.98	0.58	7.10	3.50	194.62	197.2
121.34	0.028n	0.481	0.268	570.	476.85	6.72	0.56	6.76	3.27	336.06	327.4
121.34	0.044n	0.481	0.267	571.	478.83	6.72	0.56	6.76	3.28	339.22	327.3
121.34	0.076n	0.481	0.267	570.	478.05	6.72	0.57	6.77	3.28	333.25	327.4
121.34	0.138n	0.482	0.267	570.	479.48	6.75	0.58	6.79	3.30	340.05	327.4
121.34	0.264n	0.482	0.267	570.	479.00	6.80	0.61	6.82	3.33	338.16	327.1
121.34	0.516n	0.484	0.269	570.	477.28	6.87	0.66	6.90	3.38	339.06	327.3
121.34	1.02ns	0.486	0.271	570.	478.64	7.13	0.71	7.09	3.50	340.58	327.7
250.27	0.028n	0.770	0.268	1163	961.40	7.01	0.82	6.77	3.27	621.43	587.5
250.27	0.044n	0.767	0.267	1162	964.02	6.99	0.82	6.76	3.28	608.58	587.8
250.27	0.076n	0.770	0.267	1159	962.09	7.00	0.83	6.77	3.28	614.24	587.4
250.27	0.138n	0.770	0.267	1162	964.53	7.03	0.84	6.79	3.30	621.66	587.9
250.27	0.264n	0.770	0.267	1162	963.18	7.08	0.87	6.83	3.33	612.73	587.7
250.27	0.516n	0.777	0.269	1163	961.67	7.37	0.92	6.98	3.38	627.83	587.7
250.27	1.02ns	0.774	0.273	1162	962.10	7.38	0.99	7.08	3.50	617.60	587.6
506.23	0.028n	1.349	0.268	2334	1921.3	7.57	1.34	6.76	3.27	1182.8	1104.
506.23	0.044n	1.349	0.267	2336	1922.8	7.57	1.34	6.76	3.28	1186.0	1104.
506.23	0.076n	1.348	0.267	2334	1922.0	7.56	1.35	6.77	3.28	1177.0	1103.
506.23	0.138n	1.350	0.267	2335	1922.3	7.59	1.36	6.78	3.30	1185.0	1104.
506.23	0.264n	1.351	0.268	2333	1922.4	7.66	1.39	6.83	3.33	1184.9	1103.
506.23	0.516n	1.353	0.269	2334	1921.8	7.81	1.44	6.94	3.38	1179.3	1104.
506.23	1.02ns	1.354	0.270	2333	1921.8	7.97	1.49	7.09	3.50	1181.9	1104.
1023.8	0.028n	2.498	0.268	4686	3877.9	8.70	2.38	12.78	3.27	2319.5	2147.
1023.8	0.044n	2.498	0.267	4683	3879.5	8.70	2.38	12.69	3.28	2317.0	2148.
1023.8	0.076n	2.497	0.268	4684	3882.2	8.70	2.39	12.76	3.28	2314.5	2147.
1023.8	0.138n	2.499	0.267	4695	3881.5	8.73	2.40	12.98	3.30	2322.5	2147.
1023.8	0.264n	2.499	0.268	4693	3878.0	8.78	2.43	13.22	3.32	2321.4	2148.
1023.8	0.516n	2.504	0.269	4691	3876.2	9.01	2.48	13.84	3.38	2324.0	2148.
1023.8	1.02ns	2.503	0.272	4687	3882.2	9.11	2.53	14.44	3.49	2319.0	2148.

BIBLIOGRAPHY

[1] http://www.us.design-reuse.com/articles/9010/fpga-s-vs-asic-s.html

[2] A.K.Jones, R.Hoare, D.Kusic. G.Mehta, J.Fazekas and J.Foster, Reducing Power while increasing performance with SuperCISC, ACM Transactions on Embedded Computing Systems (TECS), 5(3): 658-686, 2006

[3] Gayatri Mehta, Justin Stander, Mustafa Baz, Brady Hunsaker and Alex K. Jones, IEEE, 2007, Interconnect Customization for a coarse-grained reconfigurable fabric

[4] R. Hoare, A. K. Jones, D. Kusic, J. Fazekas, J. Foster, S. Tung, and M. McCloud. Rapid VLIW processor customization for signal processing applications using combinational hardware functions. EURASIP Journal on Applied Signal Processing, 2006:Article ID 46472, 23 pages, 2006.

[5] Prime Power Manual

[6] S. G. Narendra and A. Chandrakasan, Leakage in Nanometer CMOS Technologies, Springer, 2006

[7] Jan M. Rabaey and Anantha Chandrakasan and Borivoje Nikolic, Digital Integrated Circuits: A Design Perspective, Prentice Hall, 2003, 2nd edition.

[8] Zhan Guo, A Short Introducttion to ASIC Design Flow in AMIS Library, Nov 2003

[9] SoC Encounter Text Command Reference, Cadence

[10] SoC Encounter User Guide, Cadence

[11] An Analysis of the Robustness of CMOS Delay Elements, Srivathsan Krishnamohan and Nihar Mahapatra, GLSVLSI April, 2005 .

[12] Yuk-Wah Pang et al. An asynchronous cell library for self-timed system designs. IEICE Transactions on Information and Systems, pages 296-305, March 1997.

[13] M.F. Aburdene, J. Zheng, and R.J.Kozick., New recursive VLSI architectures for forward and inverse discrete cosine transform. Proceedings of SPIE – The International Society for Optical Engineering, 1996

[14] A.W.Buchwald, K.W.Martin and A.K.Oki. A 6GHz integrated phase-locked loop using AlGaAs/GaAs heterojunction bipolar transistors. IEEE Journal of Solid-State Circuits, pages 1752-1762, 1992.

[15] Comparison and Analysis of Delay Elements, Nihar R.Mahapatra, Alwin Tareen and Sriram V.Garimella, Proc. IEEE Computer Society Annual Workshop on VLSI (WVLSI 2000), pp. 81-86.

[16] An Empirical and analaytical comparison of delay elements and a new delay element design, Nihar R. Mahapatra, Sriram V Garimella, and Alwin Tareen,

[17] A Low-Voltage, Low-Power CMOS delay element, Gyudong Kim, Min-Kyu Kim, Byoung-Soo Chang and Wonchan Kim, IEEE Journal of Solid state circuits, Vol 31, July 1996

[18] A Low –Power Thyristor-Based CMOS Programmable delay element, Junmou Zhang et.al., IEEE,2004

[19] ST Microelectronics Application Note on EEPROMS

[20] J.M. Portal, L.Forli and D.Nee, Floating_gate EEPROM Cell Model based on MOS Model 9, IEEE,2002

[21] Macromodel Development for a FLOTOX EEPROM, Kenneth V. Noren and Ming Meng, IEEE, 1998.

[22] J.M.Daga et.al, Design Techniques for EEPROMs embedded in portable systems on chips, IEEE Design and Test of Computers, 2003.

[23] Tommaso Zerilli, Maurizio Gaibotti, Generator Circuit for Voltage Ramps and Corresponding Voltage Generation Method, US Patent, No: US 6,650,153 B2, Nov, 2003.

[24] L. Dong-Sheng, Z.Xue-Cheng, Z.Fan and D.Min, "New Design of EEPROM Memory for RFID Tag IC," IEEE Circuits and Devices Magazine, 2006

[25] J.M.Daga et.al., A 40ns random access time low voltage 2Mbits EEPROM memory for embedded applications, International Workshop on Memory Technology, Design and Testing, IEEE, 2003

[26] M.Combe, J-M Daga, S. Ricard and M.Merandat, EEPROM Architecture and Programming Protocol, US Patent, Patent No US 6859391 B1, Feb 2005.

[27] Henry A. Om'mani, Power Multiplexer and Switch with Adjustable well bias for gate breakdown and well protection, US 7,005,911 B1, Feb, 2006

[28] A.Turier, L.B. Ammar and A.Amara, "An accurate Power and Timing Modeling Technique Applied to a Low Power ROM Compiler", PATMOS, pp. 181-190, 1998

[29] L.Yang and J. S. Yuan, Design of a new CMOS output buffer with low switching noise, Proc. of the International Conference on Microelectronics (ICM)},pages 131-134,2003

[30] Jinn-Shyan Wang, Po-Hui Yang and Wayne Tseng, Low-Power Embedded SRAM Macros with Current-Mode Read/Write Operations, ACM, 1998

[31] R.Morales-Ramos, J.A.Montiel_nielson, R.Berenguer and A.Garcia-Alonso, "Voltage Sensor for Supply Capacitor in Passive UHF RFID Transponders", Proc. Of 9th EUROMICRO Conference on Digital System Design, 2006.

[32] Standard Cell Library/Library Exchange Format, www.csee.umbc.edu/~cpatel2/links/414/slides/ lect03_LEF.pdf

[33] Wikipedia, http://en.wikipedia.org/wiki/Design_Exchange_Format

[34] Timing Library Format Reference

[35] Introduction to Cell Characterization, SimuCad

[36] Standard Delay Format Specification, Version 3.0

[37] www.wikipedia.org/wiki/SPEF

[38] http://www.vlsibank.com/sessionspage.asp?titl_id=13830