12-2007

Design of ALU and Cache Memory for an 8 bit ALU

Pravin chander Chandran
Clemson University, pravinc@clemson.edu

DESIGN OF ALU AND CACHE MEMORY FOR AN 8 BIT MICROPROCESSOR

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Pravin Chander Chandran
December 2007

Accepted by:
Dr. Kelvin Poole, Committee Chair
Dr. William Harrell
Dr. Michael Bridgwood

ABSTRACT

The design of an ALU and a Cache memory for use in a high performance processor was examined in this thesis. Advanced architectures employing increased parallelism were analyzed to minimize the number of execution cycles needed for 8 bit integer arithmetic operations. In addition to the arithmetic unit, an optimized SRAM memory cell was designed to be used as cache memory and as fast Look Up Table.

The ALU consists of stand alone units for bit parallel computation of basic integer arithmetic operations. Addition and subtraction were performed using Kogge Stone parallel prefix hardware operating at 330MHz. A high performance multiplier was built using Radix 4 Modified Booth Encoder (MBE) and a Wallace Tree summation array. The multiplier requires single clock cycle for 8 bit integer multiplication and operates at a maximum frequency of 100MHz. Multiplicative division hardware was built for executing both integer division and square root. The division hardware computes 8-bit division and square root in 4 clock cycles. Multiplier forms the basic building block of all these functional units, making high level of resource sharing feasible with this architecture. The optimal operating frequency for the arithmetic unit is 70MHz.

A 6T CMOS SRAM cell measuring 90 μm^2 was designed using minimum size transistors. The layout allows for horizontal overlap resulting in effective area of 76 μm^2 for an 8x8 array. By substituting equivalent bit line capacitance of P4 L1 Cache, the memory was simulated to have a read time of 3.27ns.

An optimized set of test vectors were identified to enable high fault coverage without the need for any additional test circuitry. Sixteen test cases were identified that would toggle all the nodes and provide all possible inputs to the sub units of the multiplier. A correlation based semi automatic method was investigated to facilitate test case identification for large multipliers. This method of testability eliminates performance and area overhead associated with conventional testability hardware.

Bottom up design methodology was employed for the design. The performance and area metrics are presented along with estimated power consumption. A set of Monte Carlo analysis was carried out to ensure the dependability of the design under process variations as well as fluctuations in operating conditions. The arithmetic unit was found to require a total die area of $2mm^2$ (approx.) in 0.35 micron process.

DEDICATION

To My Beloved Parents

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Kelvin Poole for his technical guidance, support and patience throughout the progress of this thesis. I would like to thank Dr. William R Harrell and Dr. Michael Bridgwood for serving on my committee.

I would like to thank my committee members and other faculty members for their contributions and support in enriching my educational experience. Finally, I would like to thank my family for their constant support and encouragement in all my endeavors.

TABLE OF CONTENTS

Page

Table of Contents (Continued) Page

LIST OF TABLES

LIST OF FIGURES

List of Figures (Continued)

List of Figures (Continued)

Figure Page

CHAPTER ONE

OVERVIEW

Due to wide spread use of microprocessors and signal processors, implementation of high performance arithmetic hardware has always remained an attractive design problem. Arithmetic and Logic Unit (ALU) is the workhorse of a microprocessors and determines the speed of operation of the processor. All modern processors include stand alone hardware for computation of basic arithmetic operations. In addition to fast arithmetic hardware, processors are also equipped with on-chip memory (cache) to achieve significant performance improvement by avoiding delay due to data access from main memory. The key objective of this work is to address the design of the above mentioned functional blocks. The design goals are summarized as follows.

a. Implementation of a high performance arithmetic hardware with minimum possible clock cycles capable of computing square, square root and inverse in addition to basic arithmetic operations.
b. Implementation of a fast data rate cache.
c. Investigation of testing method to eliminate the need for additional hardware for testability while ensuring high fault coverage.

A brief description of components needed to build a complete processor is presented in the following section. The functional blocks of a processor include,

1. ALU and Floating Point Unit
2. Data and Instruction Cache

3. Control Unit

4. Registers, Flags, Stack, Queues, Buffers

5. Bus circuitry

A decoder and control circuitry are needed to generate control signals needed for execution of instructions. Registers and Flags are needed within processor to store intermediate values and status after various ALU operations. Bus circuitry involving data, address and control bus is needed for proper integration of different units.

In addition to these common elements, all superscalar processors contain specialized circuits called Branch Predictors to prevent branch penalty in a pipeline implementation. Also special purpose buffers like Translation Look Aside Buffer (TLB) and Branch Target Buffer (BTB) etc are used to increase the performance of processor. TLB is used by Memory Management Unit (MMU) as a look up table to convert virtual address into physical address. BTB is used in association with Branch Predictor to cache information corresponding to conditional path determined by branch predictor. These buffers have fast access rate requirements which can satisfactorily be met by SRAM memory designed in this thesis.

Arithmetic and Logic Unit

Design of ALU was undertaken in this thesis in the context of high performance and testability. Architectures with high degree of parallelism were explored for design of high speed arithmetic unit. For simplicity, functional units were designed with 8 bit

capability. Due to architectural parallelism, increase in operand size would only require replication of hardware parallel to existing circuitry.

The ALU has stand alone hardware for performing basic integer arithmetic operations and is capable of computing square, square root and inverse as well. A logic unit performing 8 bit logic operations was built using logic cells available in the IC cell library and was found to have a high operating frequency close to 1GHz.

The multiplier is the most critical functional unit in the ALU. Popular techniques for improving the speed of a multiplier include reduction of the number of partial product rows, fast reduction of partial product rows and final summation of result using a fast adder. Wallace tree multiplier, a column compression multiplier found in many processors including IBM PowerPC [1], was implemented in this thesis. Modified Booth Encoding and Kogge stone adder were used to enhance the performance of the Wallace multiplier.

In the past, emphasis was mostly on development of high performance adder and multiplier. This resulted in poor performance of division unit and forced programmers to develop algorithms free of these operations. To increase the efficiency of a processor it is required that the latency for division is close to thrice the latency of multiplication. In recent processors, division takes 4-10 times the time needed for multiplication. Many modern processors take advantage of their fast multiplier to achieve high division speed using multiplicative division techniques. In most of the recent implementations, division and square root share functionality with FP multiplier to avoid large area and cost

involved in having a stand alone unit. Though resource sharing is beneficial, it sometimes hits the performance of arithmetic unit by loading the multiplier heavily.

In this thesis, Newton Raphson multiplicative division algorithm was used to implement a combined division and square root hardware. NR algorithm is based on functional recurrence and uses iterative refinement to obtain the result. An initial approximation is generally used to increase the rate of convergence. Popular implementation of NR division unit includes floating point division units of IBM RS6000, IBM Power PC and Power2 processors etc. A small look up table was used to decrease the number of computational cycles to four cycles.

Square root was also implemented using Newton Raphson algorithm. For processor optimization, the desired latency for square root is about 9.1 times the division latency. The hardware implemented in this thesis is capable of calculating integer square root in 4 clock cycles and inverse in 3 clock cycles. Squaring is performed by the multiplier in a single cycle clock cycle.

Cache Memory

Memory access is a frequent operation in any program. More than one memory access per instruction is needed to fetch instruction and to fetch/store data. However there has been an increasing gap between speed of memory and that of logic devices as technology scales. Modern processors tend to bridge the gap between logic performance and memory latency by use of large cache on chip. Cache stores most frequently used locations of memory and enable much faster access. The access time for main memory is

typically 100 cycles while access time for L1 cache is only 1-2 cycles. Generally a SRAM cell is used for L1 cache and DRAM is used for L2 cache and main memory. In modern processors, memory occupies as much as three quarters of die area.

In this work, SRAM memory cell was designed for optimum area and speed. In large SRAM arrays, only one cell from each column is connected to bit line at a time depending on the signal from column decoder. Other cells in the column though isolated from the bit line, contribute a significant capacitance to it. Hence the delay characteristics of a large memory array can be studied by adding appropriate capacitance to bit line of a single cell.

To simulate the performance of a large cache, it was assumed that the SRAM cell designed in this thesis is used to build L1 Cache of the P4 processor. To determine the attainable performance, number of rows in the cache of Pentium processor was first determined. By substituting the equivalent bit line capacitance, characteristics of the cache was simulated. To obtain high access rate, a differential pair sense amplifier was designed.

A row of 8 SRAM cells was built for use as Look up table for division. Due to large cycle time and small bit line capacitance, timing constraints weren't aggressive for this 8 bit SRAM unit. Hence sense amplifier for this row was substituted with an inverter. Since a single SRAM row was used, column decoder and timing circuitry were not needed for this work. The SRAM row can be seen to meet the timing requirements for registers and buffers needed for the processor and can be used to build these components as well.

The remaining portion of this thesis is structured as follows. Chapter II-IV addresses the design and development of components for arithmetic unit. Chapter V presents the design of sub units of SRAM memory. Power estimation and worst case delay analysis of the designed circuitry is described in Chapter VI. The following chapter begins with design of hardware for high speed addition.

CHAPTER TWO

ADDITION AND SUBTRACTION

Addition is the most commonly used arithmetic operation and hence the performance of an ALU is greatly dependent on the performance of its adder. A variety of choices exists for addition depending on speed and area requirements.

Background

A Ripple Carry Adder (RCA) consisting of cascaded full adders is the simplest adder available with smallest area and largest delay. Numerous techniques have been proposed so far to enhance addition speed by optimizing carry propagation chain. A Carry Skip Adder offers data dependent performance improvement by featuring a Carry-Bypass path in addition to carry ripple path. Carry is by-passed over a fixed block of adders if group propagate signal for that block is high [2]. A Variable Carry Skip adders consisting of adder blocks of variable size have been shown to improve speed [3] and reduce power dissipation [4]. The worst-case delay for a Carry Skip adder is the same as that of a Ripple Carry Adder which makes it unsuitable for high performance applications. In case of Carry Select Adder, [5] results corresponding to both possible values of carry-in are pre computed and appropriate result is later selected based on carry rippled from previous stage.

Look Ahead adder is one of the fastest adders available, characterized by absence of any ripple mechanisms. Carry Look Ahead adder [CLA] computes carry-in for each bit using a Look Ahead circuitry [6]. The complexity of look-ahead circuitry increases with

operand size due to fan-in requirements, making the adder unsuitable for large operand sizes. Prefix tree architectures with controlled fan out are commonly used for wide operand look-ahead addition.

A class of adders based on Ling's algorithm target alternate implementations of the carry equation [7]. Instead of propagating the actual carry, propagation of pseudo carry is investigated in these algorithms. Naffziger adder [8], a popular variation of Ling's algorithm, found its implementation in Itanium 2 and HP PA RISC 64 bit processors. Alternate adders like NMOS based Manchester Chain Carry Skip adders are exemplary of circuit level techniques to improve addition speed [9, 10].

Hybrid adders exploit the advantages of more than one addition scheme and are proven to outperform each of the constituent adders [11, 12]. For large operand sizes, parallel prefix adders are believed to have a superior performance compared to other addition techniques. A brief description of the available parallel prefix adders is presented in the following section.

Parallel Prefix Adders (PPA)

With increasing clock frequencies, simple CLA adders cannot meet the timing requirements for use in wide data paths. Dramatic improvement in CLA performance for large operands is obtained by use of prefix tree architectures. Popular PP architectures include Brent-Kung [13], Kogge-Stone [14], Ladner Fischer [15], Han Carlson [16] and Sklansky [17]. All these architectures differ only in the way they handle the trade off between logic depth and hardware size.

Of the available PPAs, Brent and Kung adder has the maximum logic depth and minimum area. Ladner-Fischer adder has the minimum number of stages with maximum fan out for all stages. A Kogge Stone has the maximum number of cells with minimum fan out. Han Carlson adder is a hybrid of the Ladner Fischer and the Kogge Stone adder with reduced wiring and area compared to Kogge Stone adder. A summary of characteristics of different parallel prefix adders is given below.

ADDER	LOGIC LEVEL	FAN OUT	CELLS
Brent – Kung	$2\log_2 N - 1$	2	$2N$
Sklansky	$\log_2 N$	$N/2 + 1$	$0.5N \log_2 N$
Kogge Stone	$\log_2 N$	2	$N \log_2 N$
Han-Carlson	$\log_2 N + 1$	2	$\frac{N}{2} * \log_2 N + N - 1$
Ladner-Fischer	$\log_2 N$	$N/2$	$0.5 * N * \log_2 N$

Table 2-1: Comparison of Parallel Prefix Adders.

Kogge Stone tree is the most preferred topology for high performance data paths. Based on logical effort calculations, speed of a Kogge Stone adder has been reported to exceed the performance of any other PPA adders for bit sizes upto 128 bits [18]. Work on

high radix implementation of parallel prefix adders has shown further improvement in performance through reduction of both logic depth as well as cell count [19].

Addition and Subtraction

A Kogge Stone adder was implemented in this thesis for addition and subtraction. Kogge Stone adder has a fan out of 2 and completely eliminates the fan out problem associated with large carry look ahead adders. The equations for prefix computation for the KS adder as well as the PG diagram are presented below. The notation for sub units of PG diagram can be found in Appendix 1.

Generate and Propagate:

$$g_i = A_i.B_i$$
$$p_i = A_i \oplus B_i$$

Sum Generation:

$$S_i = p_i \oplus c_i$$

Look Ahead Carry Generation

$$C_1 = g_1$$
$$C_i = g_i + p_i C_{i-1}$$
$$C_{out} = C_{n+1}$$

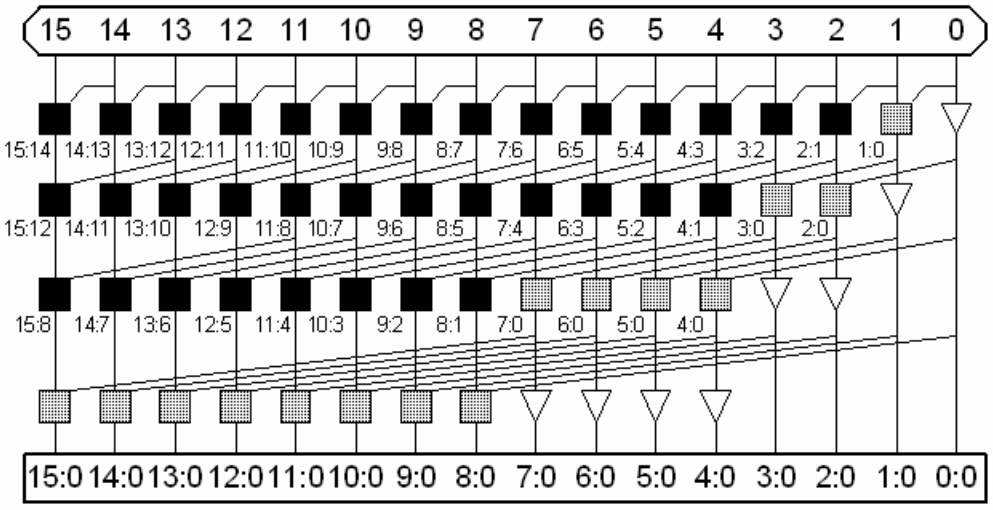


Figure 2-1: PG Diagram of a 16 bit Kogge Stone Adder.

Parallel prefix adders have a staged architecture as can be seen with Kogge Stone adder above. Presence of multiple stages facilitates pipelining which makes these adders a preferred choice for current pipelined implementations.

Subtraction

Hardware for addition and subtraction was implemented as a combined ADD/ SUB unit. Subtraction is generally performed using two's complement addition. Two's complement of a number is obtained by negation of the operand followed by an increment-by-1. To invert the operand, an XOR gate was used as a conditional NOT gate with one of its input serving as control input (CTRL). The XOR gate inverts the other input if the CTRL input is 'High'. The CTRL signal is applied to C_{in} to perform the increment needed to complete two's complement calculation. When CTRL signal is 'Low', the unit performs addition.

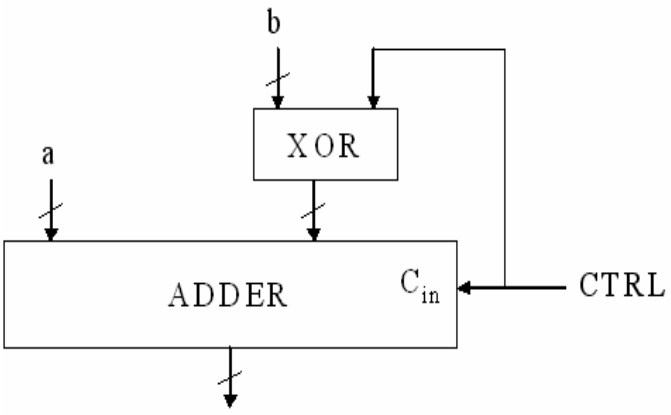


Figure 2-2: Block Diagram for Combined ADD/SUB Unit.

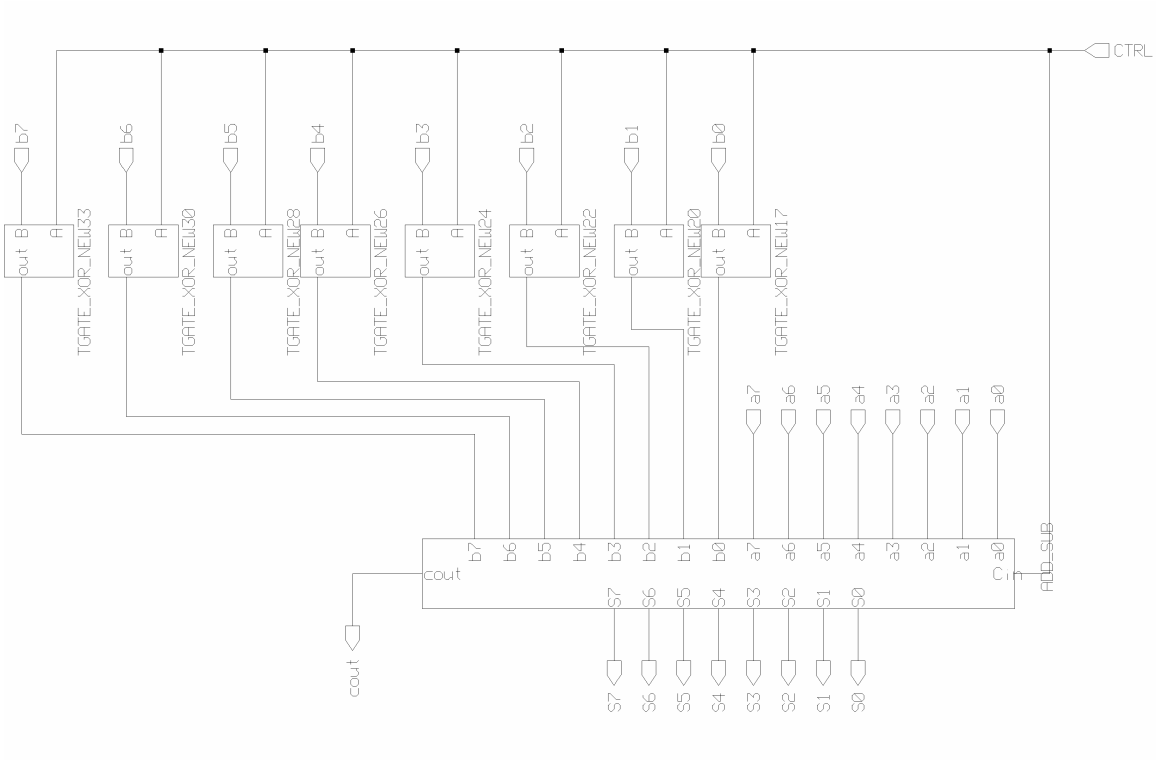


Figure 2-3: Schematic of ADD/SUB Unit.

Testing

The hardware for combined addition and subtraction was tested using a worst case input. Worst case input for an adder is one in which carry due to LSB affects the result of all other bits till MSB. Since adder is used for subtraction as well, worst case for subtraction occurs if the carry propagates from LSB to MSB during addition of inverted bit. The ADD/SUB unit was tested with the following input.

A –11111111	A + B: Sum = 11111110 ; Carry = 1
B –11111111	A – B: 00000000

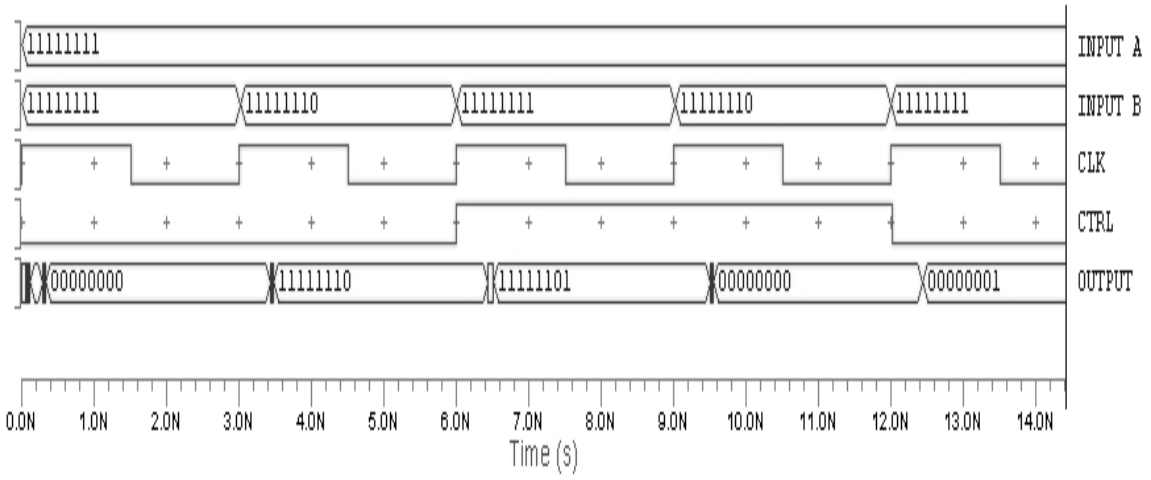


Figure 2-4: Output of ADD/SUB Unit

The maximum speed of operation of the unit was found to be 333 MHz for a worst case input.

Adder for Multiplier

In addition to the adder performing arithmetic additions, an ALU also needs adders for its multiplier. The following adders were needed for the multiplier that was built for this thesis.

1. Carry Save
2. Fast Adder for final sum

Carry save adders are often called 3:2 compressors and are employed in partial product reduction. Carry Save adder produces partial sum and carry outputs. The carry is then shifted and added with the input at next bit position. Carry Save adders are generally used in multiplication hardware for fast reduction of partial products rows.

Fast Adder for Multiplier

A variety of choices exists for design of final adder for the multiplier. For moderate operand sizes, a small CLA is generally configured as an adder chain and used as Block CLA adder. Carry Select adder and hybrid adders are also sometimes used for final addition in a multiplier. This thesis analyzed the performance improvement obtained by use of a parallel prefix adder over Block CLA and Carry Select adders for different operand sizes. This data can be used to select appropriate adders depending on bit size and performance requirements.

To draw a comparison between these adders, 4, 8, 16, and 32 bit adders were laid out and their delay, power and area were determined. The result of the study is presented below.

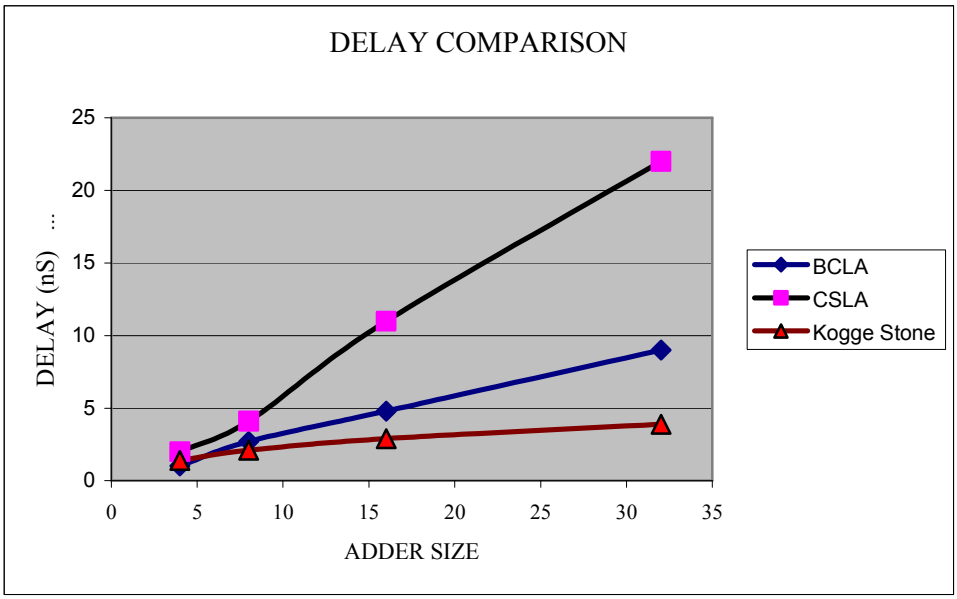


Figure 2-5: Delay Comparison of Adders.

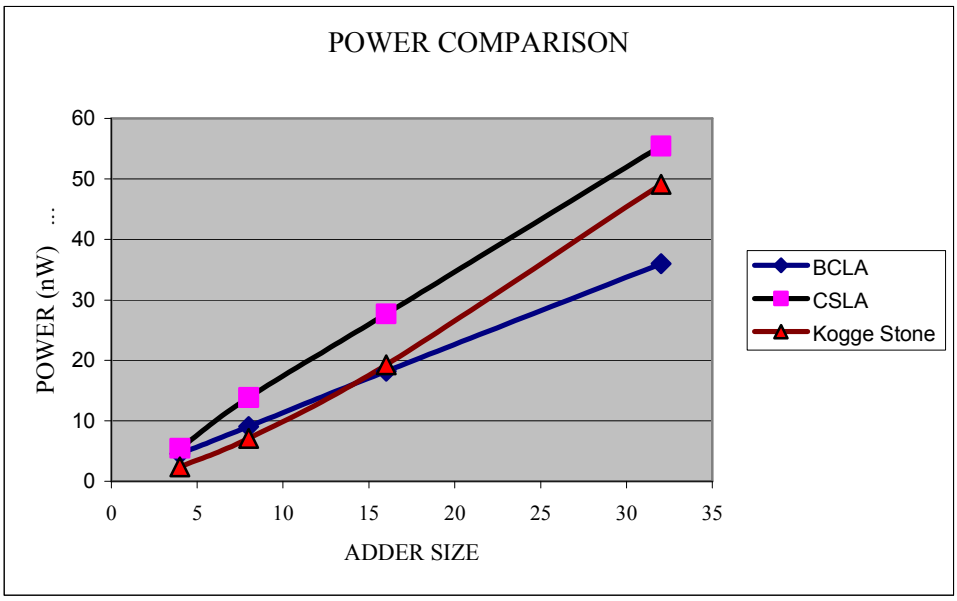


Figure 2-6: Power Comparison of Adders.

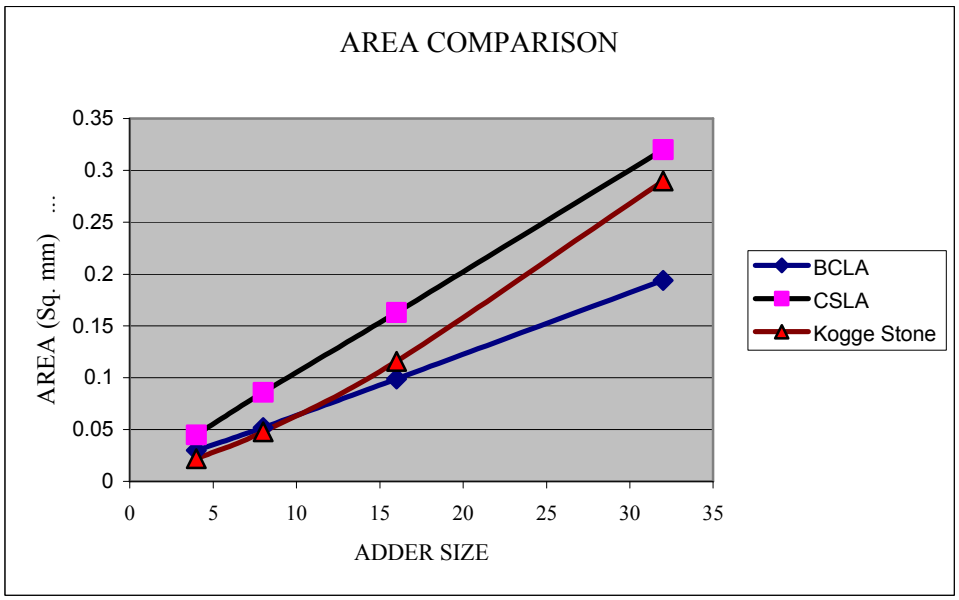


Figure 2-7: Area Comparison of Adders.

From the above plots, it can be seen that the Kogge Stone adder gives a much higher speed performance than the Block CLA and Carry Select adders, as expected. However power consumption of the Kogge Stone adder is also higher. Performance of a Carry Select adder can be seen to worsen with increase in operand sizes. Delay associated with long operand sizes could be due the remnant ripple mechanism in these adders. The rate of increase in delay with operand size was found to be much lower for a Kogge Stone adder making it an ideal adder for large operands.

CHAPTER THREE

MULTIPLICATION

Multiplication is a frequently used operation in applications like speech and image processing. Hence the speed of the multiplier determines the performance of a processor used for these applications. Recursive addition based multipliers like Shift and Add multiplier are simple and occupy small area but need multiple clock cycles to produce a data dependent result. In this thesis, a fast multiplier that performs multiplication in single cycle was built.

Background

A sequential multiplication method like serial Shift-Add takes as many cycles as the word size for multiplication [20, 21]. Reduction in number of cycles is achieved by using high radix multiplication methods that operate on more than one bit at a time [22]. Higher radix multipliers reduce the computational time at the cost of circuit complexity and can be used with architectures involving increased parallelism to achieve a single cycle multiplication.

Among parallel multipliers available, array multipliers are the easiest to design and have a regular structure that facilitates easy implementation. However array multipliers are associated with large delay and increased power consumption [23, 24]. Tree Multipliers are faster than array multiplier though they employ the same number of adders. Baugh Wooley [25, 26], Pezaris [27], Wallace tree [28, 29] or any variations of

these multiplication techniques are commonly considered for fast multiplication. Lack of regularity is the major bottleneck in implementation of tree multipliers.

Baugh Wooley multiplier adjusts partial products to maximize regularity of multiplication array. Though compact, it doesn't employ any partial product reduction scheme and hence suffers from latency of combining large partial product rows. Wallace Tree multiplier is believed to achieve superior performance over other multipliers. Dadda Multiplier [30] is a modification of Wallace tree multiplier that uses less number of adders and has a little improvement in speed. A Wallace multiplier however is comparatively easier to implement using a full custom layout than a Dadda multiplier. Hence a Wallace tree multiplier is by far the common choice when high performance is targeted. [31, 32]. A Wallace tree combines partial product bits as early as possible and reduces the depth of adder chain. Further there are fewer transitions at nodes, and fewer glitches, which result in low power consumption. Methods to improve the regularity of a Wallace tree had been of interest to some researchers [33, 40].

A Wallace tree multiplier was designed in this thesis. In addition, a minimal set of test vectors with high fault coverage were identified to ensure testability of the multiplier. The description of the multiplier architecture and the testability method are presented in the following sections.

Integer Multiplier Architecture

The integer multiplier was built to perform unsigned integer multiplication using 2's complement binary representation. General hardware for fast parallel multiplication consist of modules for,

1. Encoding
2. Partial Product reduction
3. Final Summation

Generation of fewer partial product rows greatly improves the speed of multiplication. Partial products are then compressed into a sum and a carry for each bit position using an adder tree and finally summed using a high-speed adder.

Encoding

Minimization of partial product rows is a critical step in improving the speed of multiplication. Reduction in row count is generally achieved by processing operands using some coding scheme. Gray code has the characteristic of low switching activity and is used in some multipliers [34]. Due to prevalent use of binary number system, binary encoding is however predominant in most implementations. Binary encoding techniques include Canonical (Booth), Differentiating, Non Restoring, and Modified Booth recoding. Majority of the latest works in high-speed multiplication employ Modified Booth encoding [35, 36, 37].

A simple booth coding scheme involves replacement of consecutive ones by a 1 in MSB+1 and -1 in LSB.

$$2^N + 2^{N-1} + 2^{N-2} \ldots\ldots 2^n = 2^{N+1} - 2^n$$

A modified version of booth encoder proposed by Macsorley [38] is widely used for encoding multiplicands. The algorithm can be implemented at different radices. A radix 2^N booth encoder reduces the partial product rows by N. A Radix-8 encoder, for example, produces only a third of original partial product rows [39]. Higher radix implementations however increase circuit complexity limiting the use of radix to 4 in most cases. The truth table of a Radix 4 booth encoder is given below. Booth selector is a multiplexer to select an operand based on control signal.

M_{i+1}	M_i	M_{i-1}	MBE OUTPUT
0	0	0	0
0	0	1	$\times 1$
0	1	0	$\times 1$
0	1	1	$\times 2$
1	0	0	$\times -2$
1	0	1	$\times -1$
1	1	0	$\times -1$
1	1	1	0

Table 3-1: Truth Table of Modified Booth Encoder

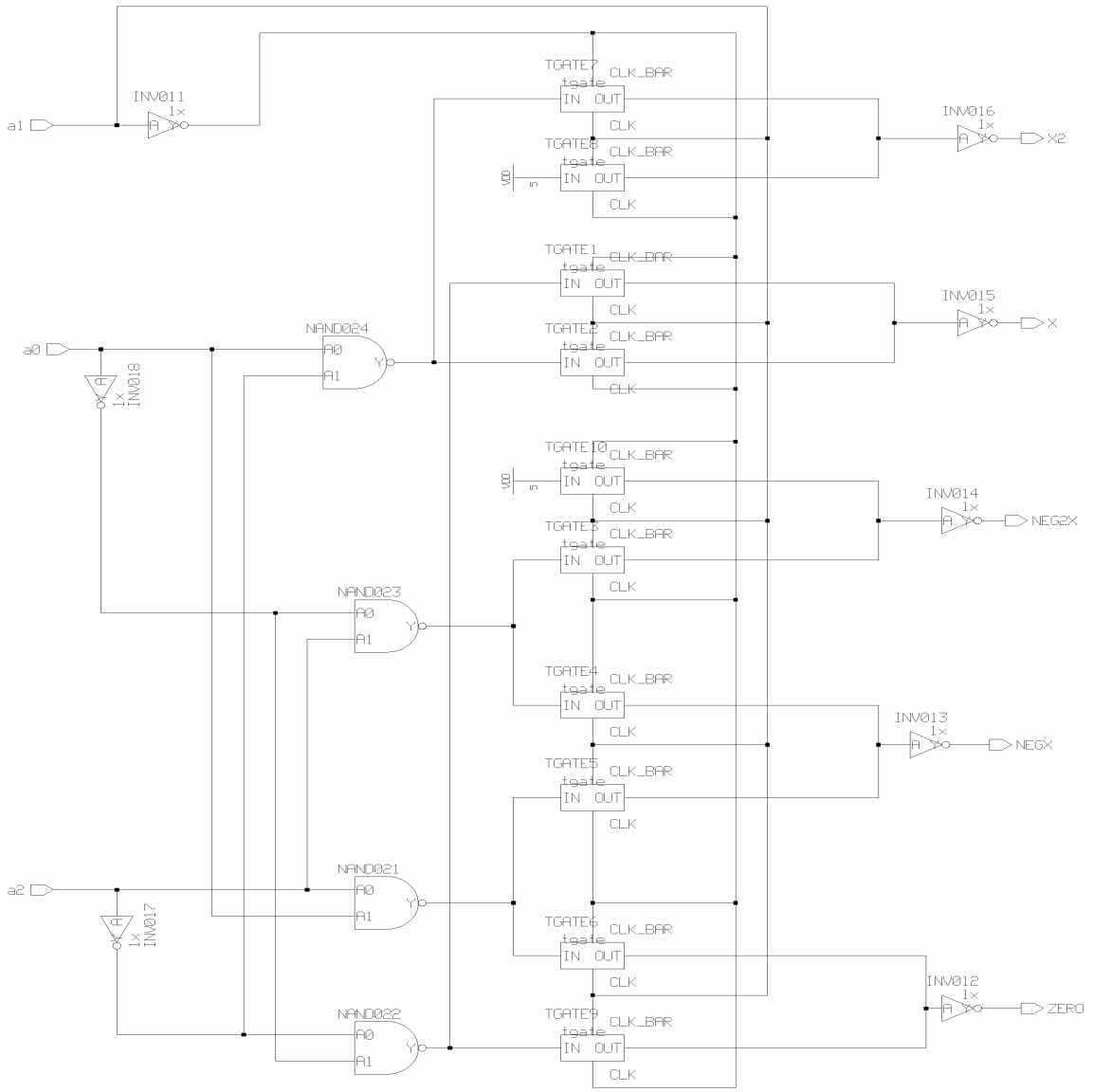


Figure 3-1: Schematics of Modified Booth Encoder using T-Gate

For 8-bit multiplication, Radix 4 MBE produces four partial product rows.The number of partial product rows is thus only half the original number. In reality a MBE

produces N+ 1 row of partial products due to use of negative encoding. Methods to reduce the additional row have been investigated with good success though [40].

Negative Product Generation

An MBE produces positive as well as negative partial products. Generation of negative multiples requires two's complement determination. -2Y term for example, can be generated using a hardwire shift and two's complementation. Two's complement involves inversion of the number followed by addition of a '1'. To complete two's complement, a number of fast adders, proportional to partial product rows are needed. To avoid this, partial product is only inverted and 1 is added to LSB if the recoding signal is negative

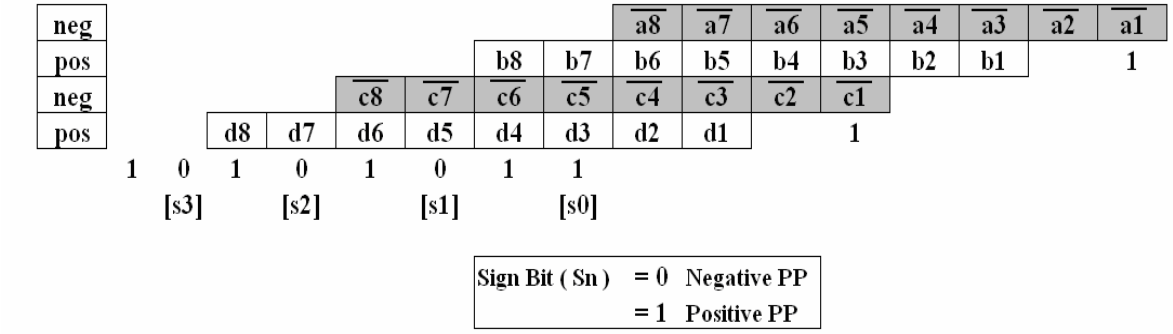


Figure 3-2: Generation of Negative Partial Product Rows.

Sign Extension

In case of signed numbers, the MSB bit represents the sign of the number. The partial product is negative if it contains a '1' in its MSB and is positive otherwise. In case of negative partial products this sign bit has to be extended till the bit position 16 as shown below.

Neg	1	1	1	1	1	1	1	1	a8	a7	a6	a5	a4	a3	a2	a1
Pos	0	0	0	0	0	0	b8	b7	b6	b5	b4	b3	b2	b1		
Neg	1	1	1	1	c8	c7	c6	c5	c4	c3	c2	c1				
Pos	0	0	d8	d7	d6	d5	d4	d3	d2	d1						

Figure 3-3: Simple Sign Extension

The MSB bit will be heavily loaded if the sign bit were to be literally extended till MSB as shown above. In addition the number of computations needed and power consumption also increases in case of such an extension scheme. Instead the result of addition is generally pre computed (10101011 in Figure 3-2) and added to partial product row at the appropriate position. The combination of '10101011' and sign bits [S0-S3] is used for sign extension and negative product generation needed for negative partial products.

For unsigned multiplication, an additional row of multiplicand has to be added to partial product row.

Partial Product Reduction

A Wallace tree increases the speed of partial product reduction by virtue of its increased parallelism. Carry save adders are used for constructing adder tree and configuration of carry save adders determines power and performance of reduction scheme. A 3:2, 4:2 or 5:2 CSA can be employed for PP reduction. Logic depth decreases with the size of CSA while the power consumption increases [36]. To minimize power, a 3:2 CSA was used in this work. Further improvement in performance and power can be achieved by path delay matching.

Wallace Tree Construction

Bakalis et al. [41] suggested some rules for construction of Wallace tree. These rules were found to reduce unwanted transition at nodes resulting in performance improvement and power reduction. The key items to be considered in constructing a Wallace tree are summarized as follows.

a. The partial product bits (PP) are to be grouped in triplets and fed to a full adder at the first level of each tree. Partial product bits greater than 3 in a particular bit position need to be summed at the next level along with carry from the preceding bit position.
b. Half adders should be used only at the terminal level and the number of HA per tree should be at most one.
c. The sign extension bits are summed either at the last or the previous to last level of each tree to avoid unnecessary transitions in the tree.

The dot diagram and tree structure that follows describe the Wallace tree reduction logic.

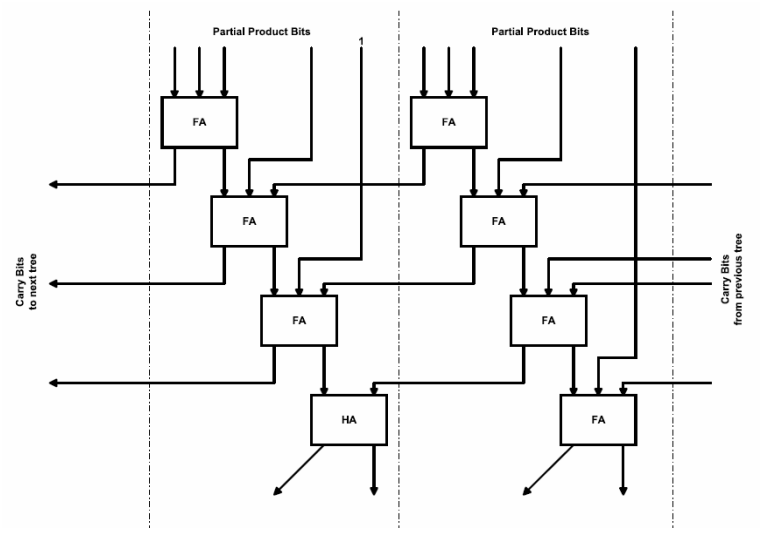


Figure 3-4: Wallace Tree Construction diagram [41]

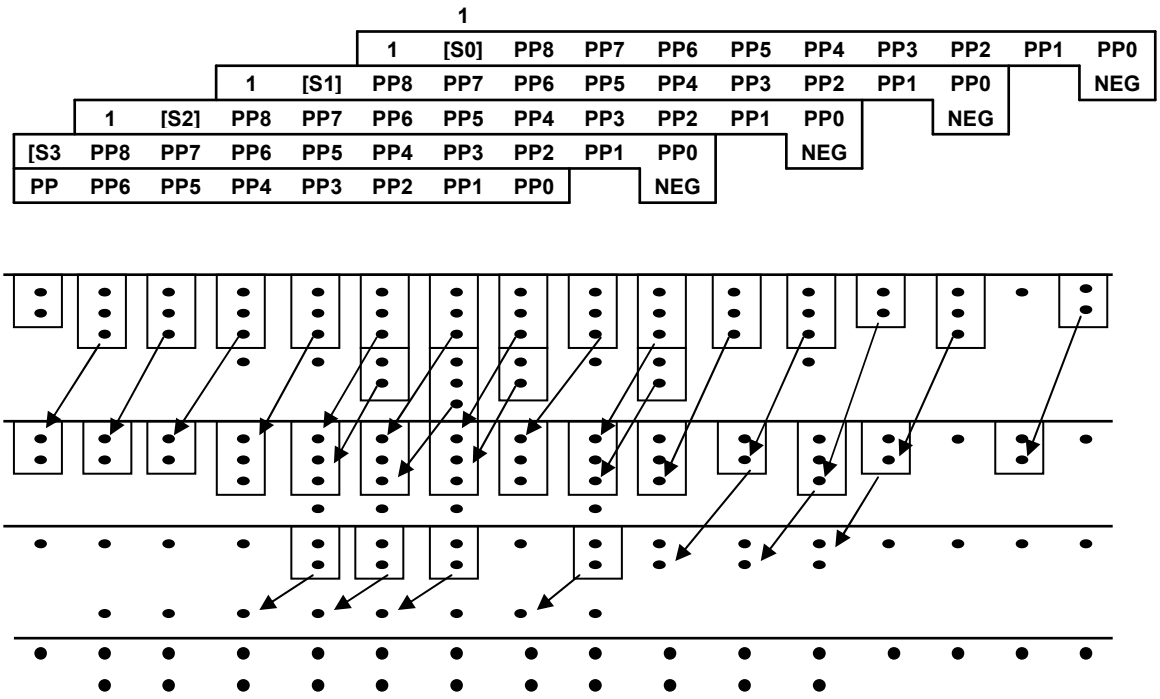


Figure 3-5: Partial Product reduction using Wallace Tree

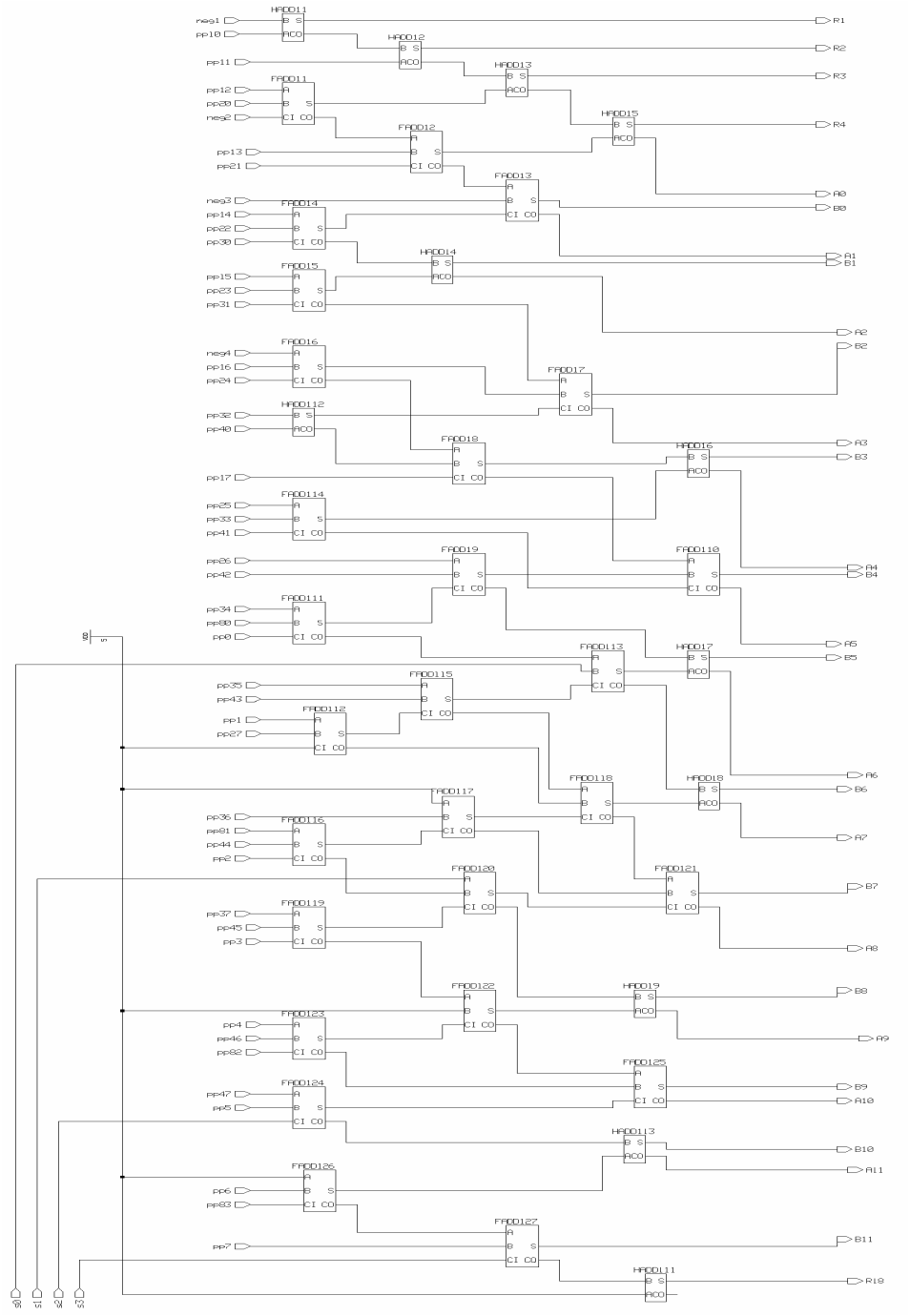


Figure 3-6: Schematic of Wallace Tree Adder.

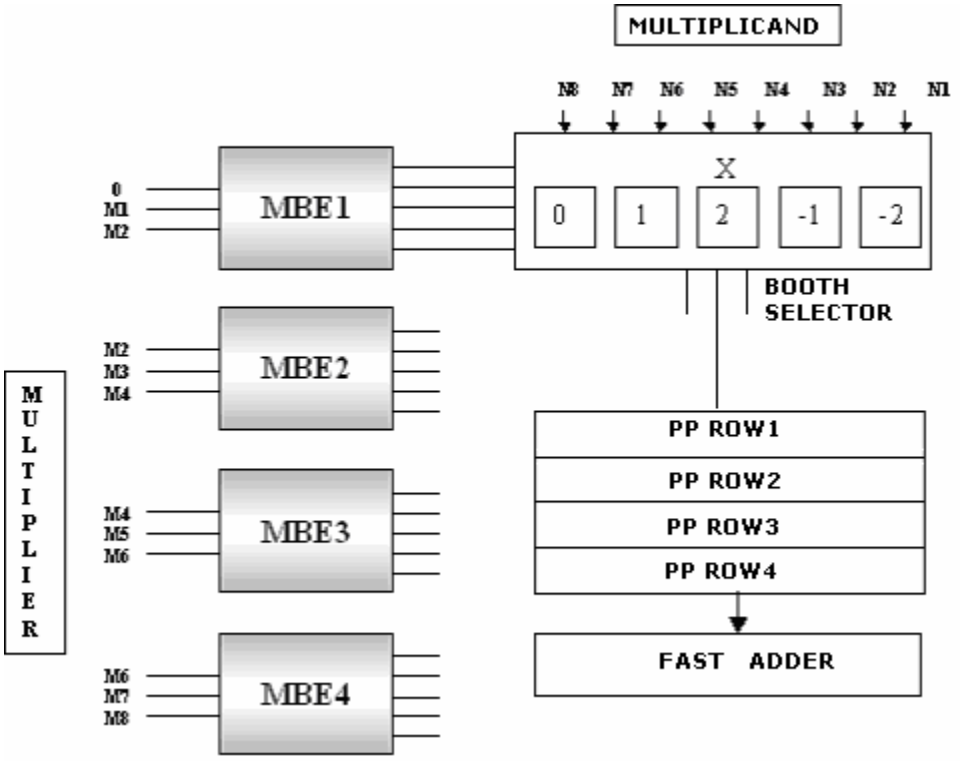


Figure 3-7: Block Diagram of Integer Multiplier

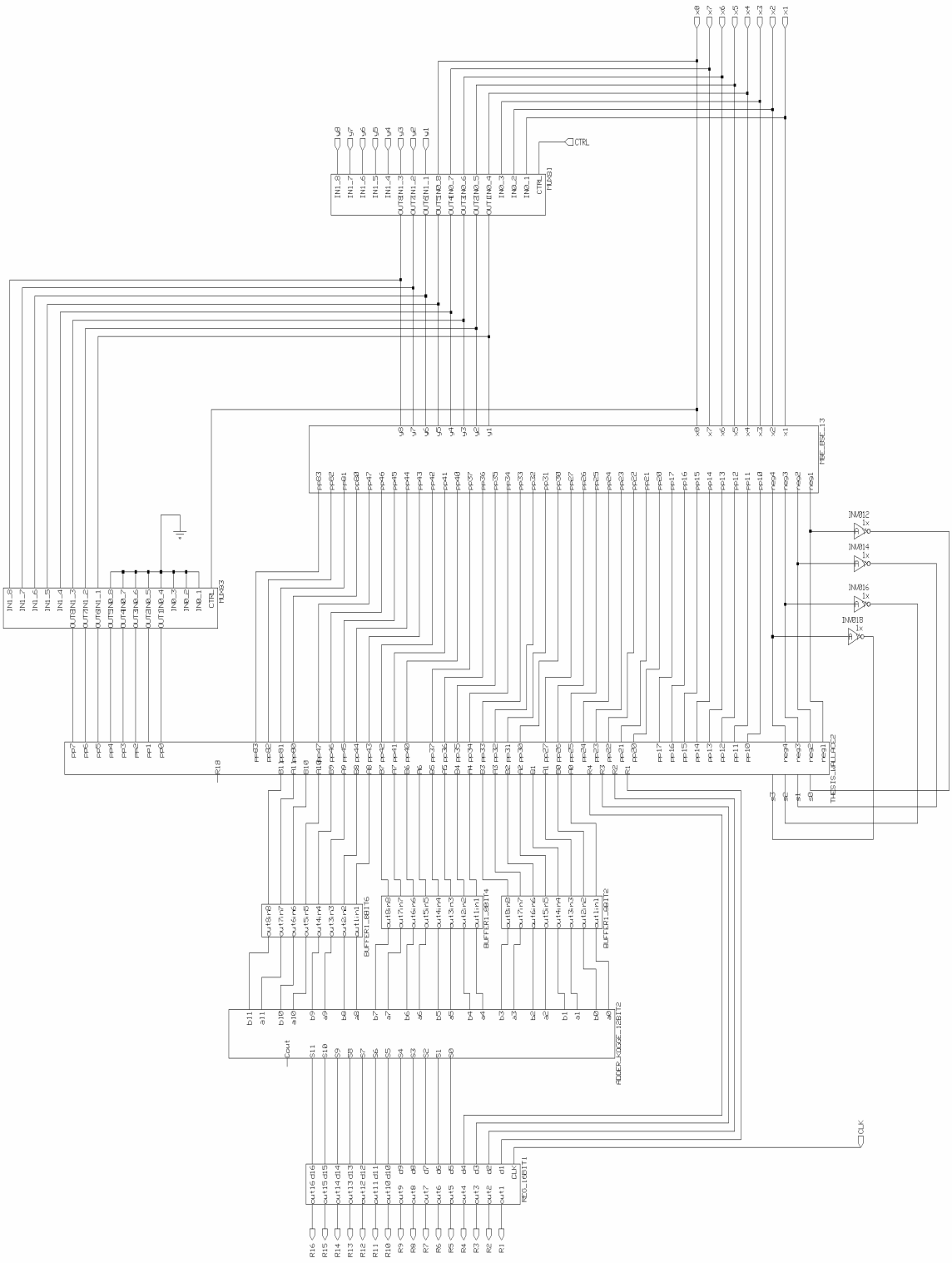


Figure 3-8: Schematic of a Multiplication and Squaring Hardware

Testability

The objectives of testability are high fault coverage, ‘at speed’ testing, minimum area overhead and minimum number of test vectors. Boundary Scan using TAP (Test Access Port) greatly improves observability and controllability and is implemented in most of the modern processors. Scan methods however result in a large area over ahead as well as increased testing time. Proper choice of test cases is still needed to ensure functionality of the design in addition to scan hardware. BIST schemes are popular for testing memory and are sometimes employed for logic circuitry as well. BIST schemes generate test vectors automatically but require increased hardware for test vector generation and compaction. The additional hardware for testability in turn affects the speed of the system under test as well. In this thesis, an optimal set of test vectors were identified to make the multiplier fully testable without the need for any additional circuitry. 16 test cases were identified that would be sufficient to verify the functionality of the multiplier circuit.

From the large set of inputs available for the 8 bit multiplier, an optimum set capable of generating all possible input combinations to the sub units were identified. The identified test vectors can generate all possible inputs to the adders of Wallace tree as well as to all the booth encoders. Since all the critical nodes of the circuit are toggled, these cases would be sufficient to test the functionality of the circuit. Booth selector and the final adder however are not targeted by the test cases. It is assumed that faults in these circuit blocks, if any, would be observable for any valid input.

Initially optimal set of test cases was manually identified by populating partial product rows based on values desired at adder inputs. The test cases manually identified are presented below. Appendix II shows the output obtained at adders of Wallace tree for the following test cases. It can be seen from the output that all full and half adders receive all possible inputs. Booth encoders can also be seen to receive all inputs needed for complete functional testing.

TEST CASE ID	MULTIPLICAND	MULTIPLIER
TC1	00000000	00000000
TC2	10101011	10010010
TC3	10101011	01101001
TC4	01010101	110111011
TC5	10101011	10000000
TC6	10101011	11011101
TC7	11111110	111X1101
TC8	01110111	10010000
TC9	10001001	1110101
TC10	1101101	01000010
TC11	00100010	10101101
TC12	10100100	10111111
TC13	01010101	00110111
TC14	11000101	01011101
TC15	10101011	11100111
TC16	10011011	11110100

Table 3-2: Test Cases for the Multiplier

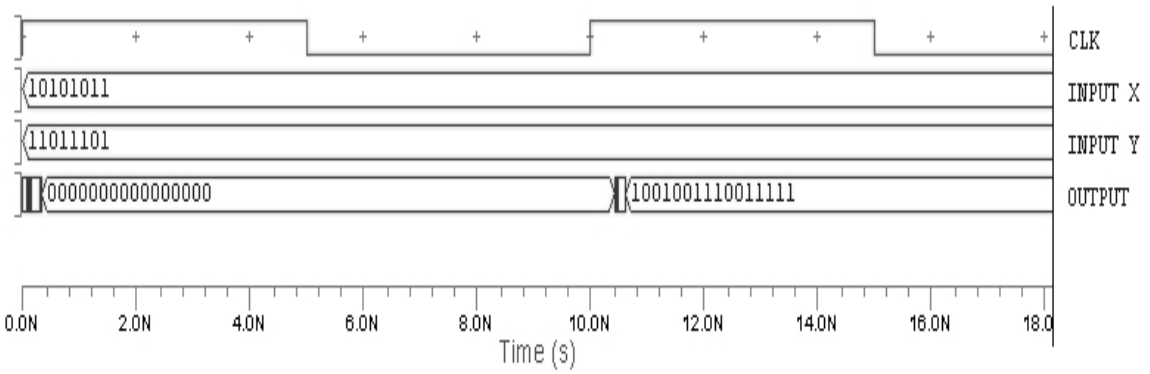


Figure 3-9: Multiplier Output

$X=171_d$; $Y = 187_d$; Result = 31977_d

The major drawback of this manual procedure is that the complexity of identification increases with bit size. A hand analysis like this would not be feasible for multipliers operating on large word lengths. A statistical method was investigated to assist human analyst with the identification task.

An attempt was made to determine test cases for adders of Wallace tree in a semi automatic fashion. The multiplier was constructed in MATLAB and input at 3:2 adders corresponding to all possible multiplier inputs was captured in a table. Adder inputs were grouped as a vector and similarity between different vectors was used as an index to discriminate between cases. An adder has 8 possible inputs and the vectors were sorted based on maximum occurrence of each input and written into separate tables. SET 1

(Table 3-3) was created by grouping one case from each table and was found to have fault coverage of 71.3 %.

To improve the fault coverage, top 25 vectors were picked from each sorted table and compared against each other. A measure of dissimilarity between the vectors was computed to improve fault coverage. SET 2 consists of one dissimilar pair for each input and sets 3 & 4 were identified by repeating the procedure with different pairs.

TEST ID	NO. OF TEST CASES	FAULT COVERAGE %
SET 1	8	71.3 %
SET 2	16	88.6 %
SET 3	16	94 %
SET 4	16	91.33 %

Table 3-3: Semi Automatic Test Case Identification

Fault coverage with the later procedure (SET 2-4) seems satisfactory for preliminary test case generation. Each set can then be refined to further improve fault coverage.

In summary, a fast multiplier with an operating frequency of 100MHz was built in this work. Also a method to generate test vectors was investigated to improve the testability of the multiplier. Performance of many other operations like division, square root and inverse are dependent on the multiplier. The following chapter describes the

design of division hardware using this multiplier. The worst case delay and the power consumption of the multiplier are examined in Chapter VI.

CHAPTER FOUR

DIVISON AND OTHER FUNCTIONS

Division and square root are increasingly becoming performance bottlenecks in realization of a high performance processor. Though the number of occurrences of division in a program is very small, the total processor time spent on division is the same as that spent for addition and subtraction. Since the operating frequency of ALU is determined by its slowest functional unit, fast division hardware is essential to improve the overall performance of the ALU.

Digit recurrence and Functional Recurrence are two major classes of division algorithms. Digit recurrence methods use subtraction to obtain quotient and divisor resulting in a linear dependence of latency on bit size. The most popular among digit recurrence algorithm is the SRT division [42, 43] which found its implementation in many commercial processors. Higher radix architecture improves the speed of division by retiring larger number of bits per cycle. High radix SRT hardware reduces the number of cycles needed but increases circuit complexity and latency. Radix-4 SRT division was used to perform integer division in early Intel Pentium Processors [44].

For high speed division, variations of functional recurrence methods are widely used. Multiplicative Division and Divisor Reciprocation are the common functional iteration methods found in many floating point hardware. Functional recurrence algorithms use iterative refinement to improve the precision of the result. The precision of the result doubles with each iteration. An initial approximation is often used to reduce the number of iterations needed to arrive at the final result. Newton Raphson and

Goldschimdt algorithm are the commonly used algorithms for multiplicative division. Newton-Raphson method [45, 46] involves two dependent multiplications that need to be performed sequentially, followed by a subtraction. Hence the time per iteration is large. A major advantage of this method is that the algorithm is self correcting and error in iteration doesn't affect the result of next iteration.

Goldschmidt method is a variation of NR method without a dependent multiplication [47, 48]. Since numerator and denominator operations are independent of each other, Goldschmidt's algorithm remains an ideal choice for pipelined implementations. Goldschmidt algorithm was used in IBM S/390, AMD K7 etc. Goldschmidt algorithm, though faster, is not self correcting. Error in any iteration can cause the result to drift from the actual value necessitating the need for error correction hardware.

Both these algorithms use an initial inverse approximation of divisor and refine the result using a series of iterations. While digit recurrence methods produce quotient and reminder, functional recurrence algorithms requires calculation of reminder from the quotient using an additional cycle. Though functional recurrence methods result in better performance, digit recurrence still retains its popularity. In this thesis Self Correcting Newton Raphson algorithm was used to build the division hardware. The following section explains the basics of NR algorithm applied to division.

Newton Raphson Algorithm

Newton Raphson algorithm applied to quotient determination is illustrated below. Consider the problem of determining the quotient of a division a/b. If x represent the inverse of the divisor,

$$x = 1/b$$
$$f(x) = bx - 1 = 0$$

The first iteration in Newton Raphson yields,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$
$$x_{i+1} = x_i - \frac{b - 1/x_i}{1/x^2{}_i}$$
$$x_{i+1} = x_i(2 - bx_i)$$

Quotient is determined using Newton Raphson iterative algorithm using the following steps.

a. Reciprocal Approximation

b. Divisor Reciprocal determination using NR iteration

$$[\, x_{i+1} = x_i * (2 - b * x_i) \,]$$

c. Multiplication of divisor reciprocal with dividend to get the quotient.

Each NR refinement step has two dependent multiplications and a subtraction. By reordering one of the dependent multiplications to next iteration, the dependency was resolved to achieve speed improvement at the cost of one additional cycle.

Fast division hardware often makes use of a pre computed initial approximation to increase the speed of division. A Look Up table is often used to store a low precision inverse approximation of the divisor. The number of iterations needed to determine the inverse accurate to n bits depends on the initial approximation used. The number of cycles reduces with bit accuracy of inverse approximation. However the size of Look Up Table increases quadratically with increase in approximation accuracy. A small look up table is easily manageable and hence is often preferable.

Normalization

To ensure that the quotient bit is obtained in estimated clock cycles, the divisor is normalized such that none of the leading bit positions are zeros. Initial approximation corresponding to most significant bits of normalized divisor is obtained from the Look up Table.

The number of clock cycles needed for division is dependent on initial approximation used. With 8 bit approximation, the output can be obtained with a single multiplication requiring two clock cycles. However the size of the look up table needed for this is 256 (2^8) rows. When a 4 bit approximation is used, additional clock cycle is needed while the look up table reduces to 16 rows. A proper choice of the table size can be made depending on area and clock cycle requirements.

Since the MSB after normalization is always 1, it can be omitted and the next four bits can be used to index the table. This will reduce the size of look up table to 8 rows for 4 bit accuracy. However instead of reducing the table size, bit accuracy of the result was

increased in this work. Since hardware to support rounding was not implemented, it was decided to increase the accuracy of the inverse approximation by one more bit. Four bits after the MSB of normalized divisor is used to address the table. This would ensure that the quotient is obtained in two iterations under all circumstances irrespective of truncation and approximation errors.

A T-Gate based Barrel Shifter was used in association with a priority detector to normalize the divisor. Priority detector circuit determines the position of first non zero in the divisor and the shifter moves the first non zero bit to MSB flushing away the leading zeros.

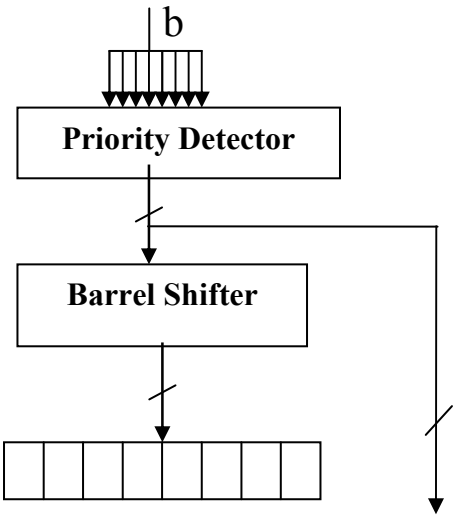


Figure 4-1: Block Diagram of Divisor Normalization Unit

The priority detector outputs the bit position of the input corresponding to first occurrence of '1'.The logic equation and truth table for priority detector is given by,

Logic Equation:

$P8 = B8$

$Pi = B_i.\overline{B_{i-1}}$

INPUT	OUTPUT
1XXX XXXX	10000000
01XXXXXX	01000000
001X XXXX	00100000
0001XXXX	00010000
00001XXX	00001000
000001XX	00001000
0000001X	0000 0010
00000001	00000001

Table 4-1: Truth Table for priority detector

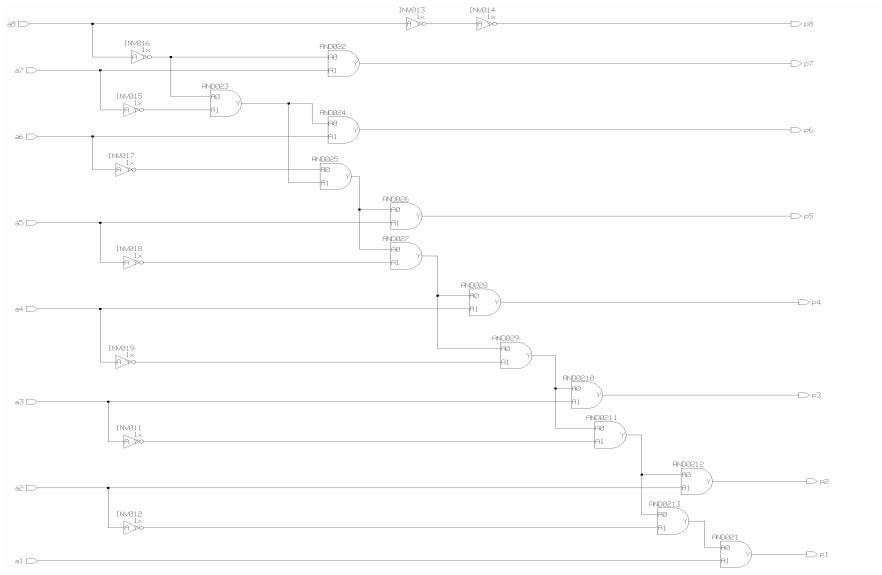


Figure 4-2: Schematic of Priority Detector.

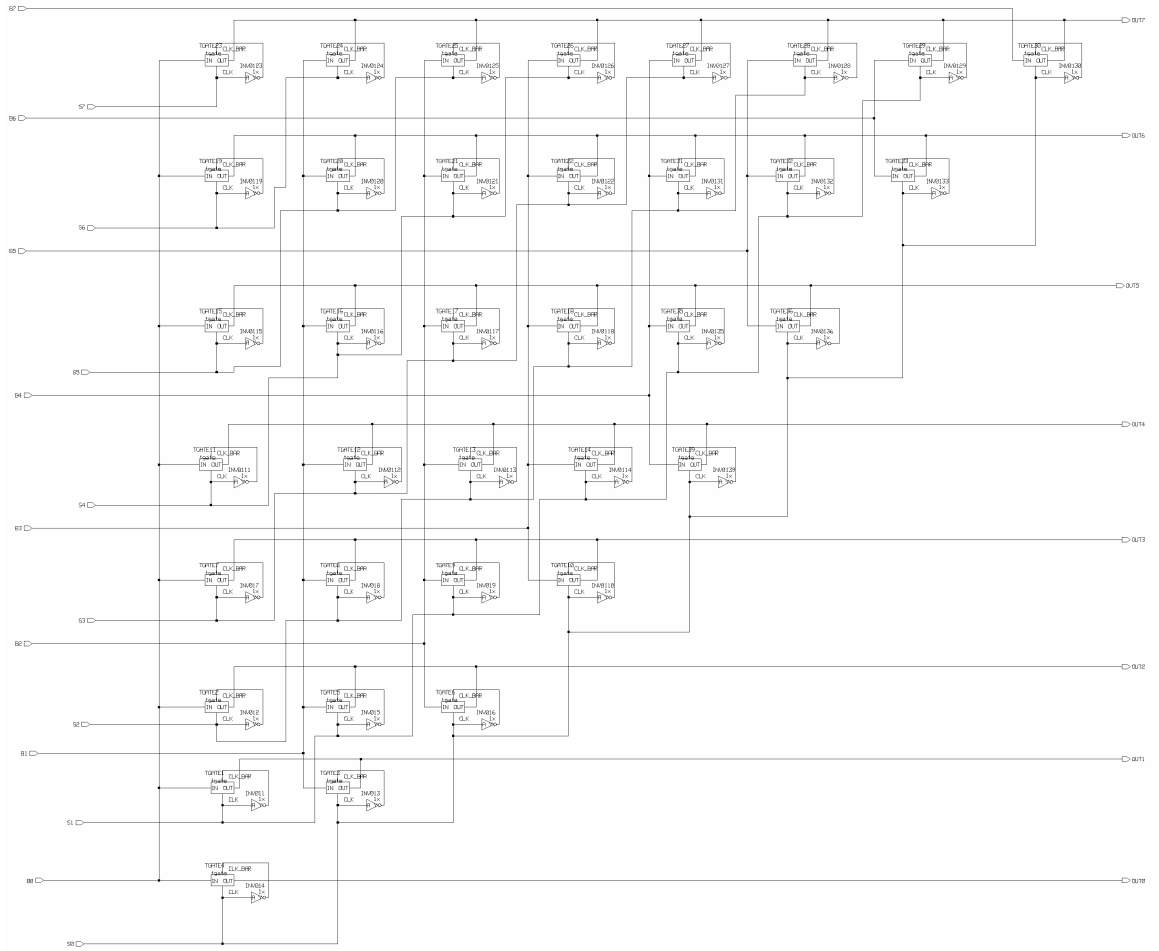


Figure 4-3: T Gate based Barrel Shifter for Divisor Normalization.

Hardware to facilitate mantissa computation for inverse operation was built along with normalization circuitry. The circuit estimates the mantissa based on size of the divisor. The details of the computation are explained under Square root computation.

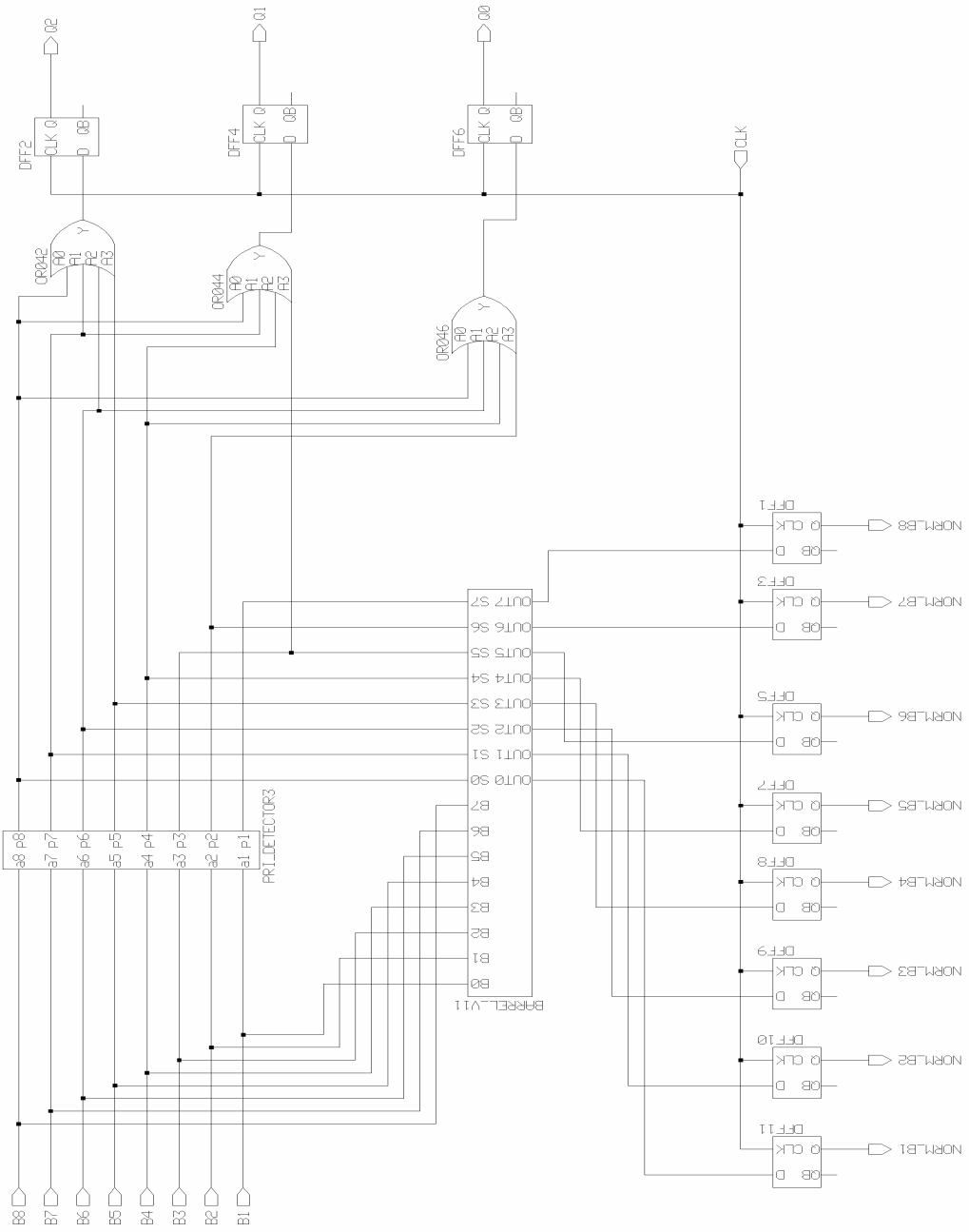


Figure 4-4: Combined Normalization and Mantissa Computation Circuit.

Initial Approximation

The approximation is pre computed using an expanded version of the Newton Raphson algorithm and stored in a table.

$$x_i = (2 - b_n) * (1 + (b_n - 1)^2) * (1 + (b_n - 1)^4$$

where, b_n - Normalized Denominator

MSB	INITIAL APPROX.
10000	1.0000000
10001	1.1110000
10010	1.1100011
10011	1.1010111
10100	1.1001100
10101	1.1000011
10110	1.0111010
10111	1.0110010
11000	1.0101010
11001	1.0100011
11010	1.0011101
11011	1.0010111
11100	1.0010010
11101	1.0001101
11110	1.0001000
11111	1.0000100

Table 4-2: Initial Approximation Look Up Table

Generally a ROM Look Up table is used to store the value. In this thesis a single row of SRAM was used instead of the look up table. An array of 16x8 SRAM cells along with column decoder can be used to build a fully functional look up table for the actual hardware. The design of the SRAM cell is described in chapter 5. Address decoder and timing design were ignored in this work for the purpose of simplicity. The inverse approximation corresponding to the divisor is stored in the SRAM before the start of division.

For an 8 bit division, the quotient is obtained in 3 cycles. Since the rate of convergence is quadratic, a 64 bit division can be performed in 6 cycles using this architecture. Increase in size of LUT can further decrease the clock cycles needed. If the look up table is to be avoided, 3 additional cycles would be needed to compute the 4 bit initial error approximation. However the normalization cycle would no longer be required making the effective increase in clock cycles to be 2. Increase in clock cycle would increase power consumption as well. Hence the use of a look up table is justifiable from performance and power perspective in modern high speed low power hardware.

NR Iterations

Result of first iteration has a bit precision of 5 MSB bits corresponding to normalized divisor. Second iteration gives a result with 10 bit precision which guarantees the desired accuracy. Since the quotient corresponds to normalized denominator, the result was again shifted using the barrel shifter.

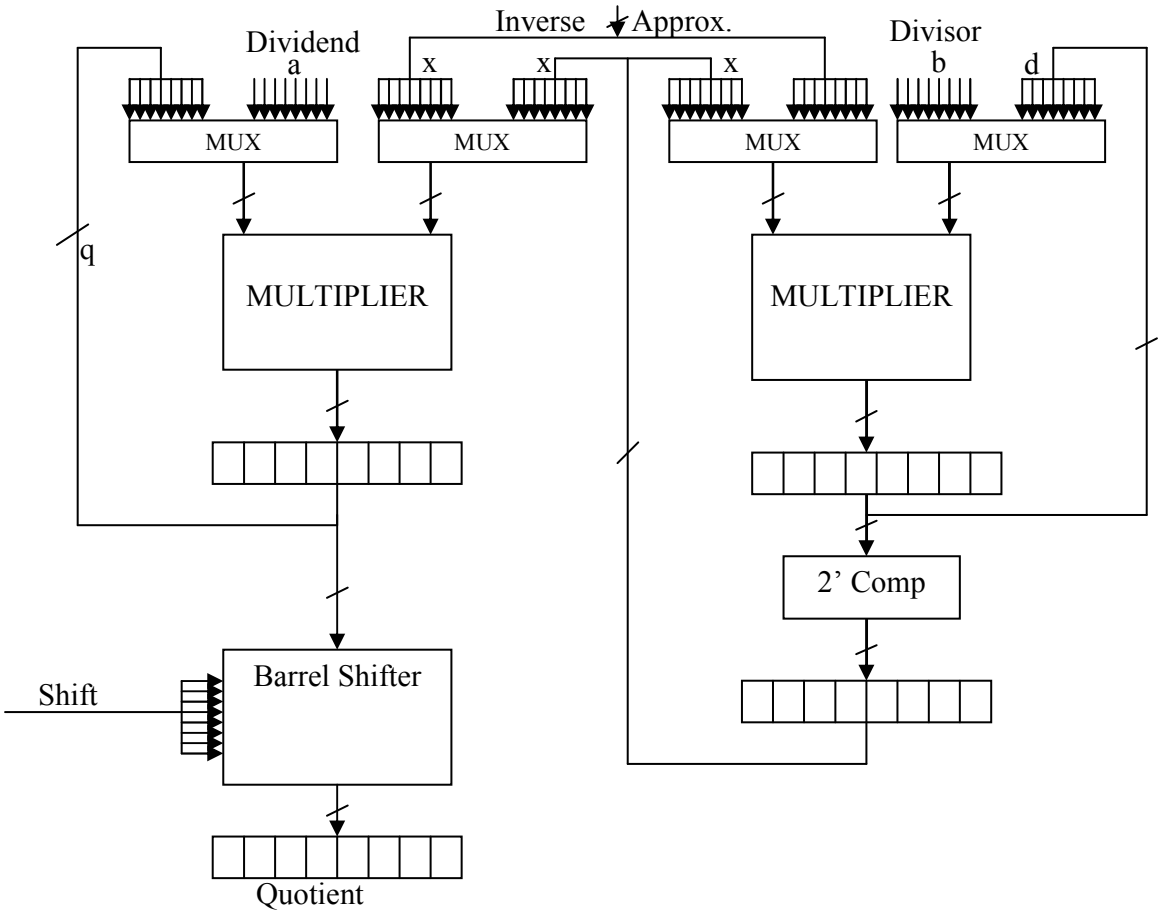


Figure 4-5: Block Diagram of Division Hardware for NR Iteration

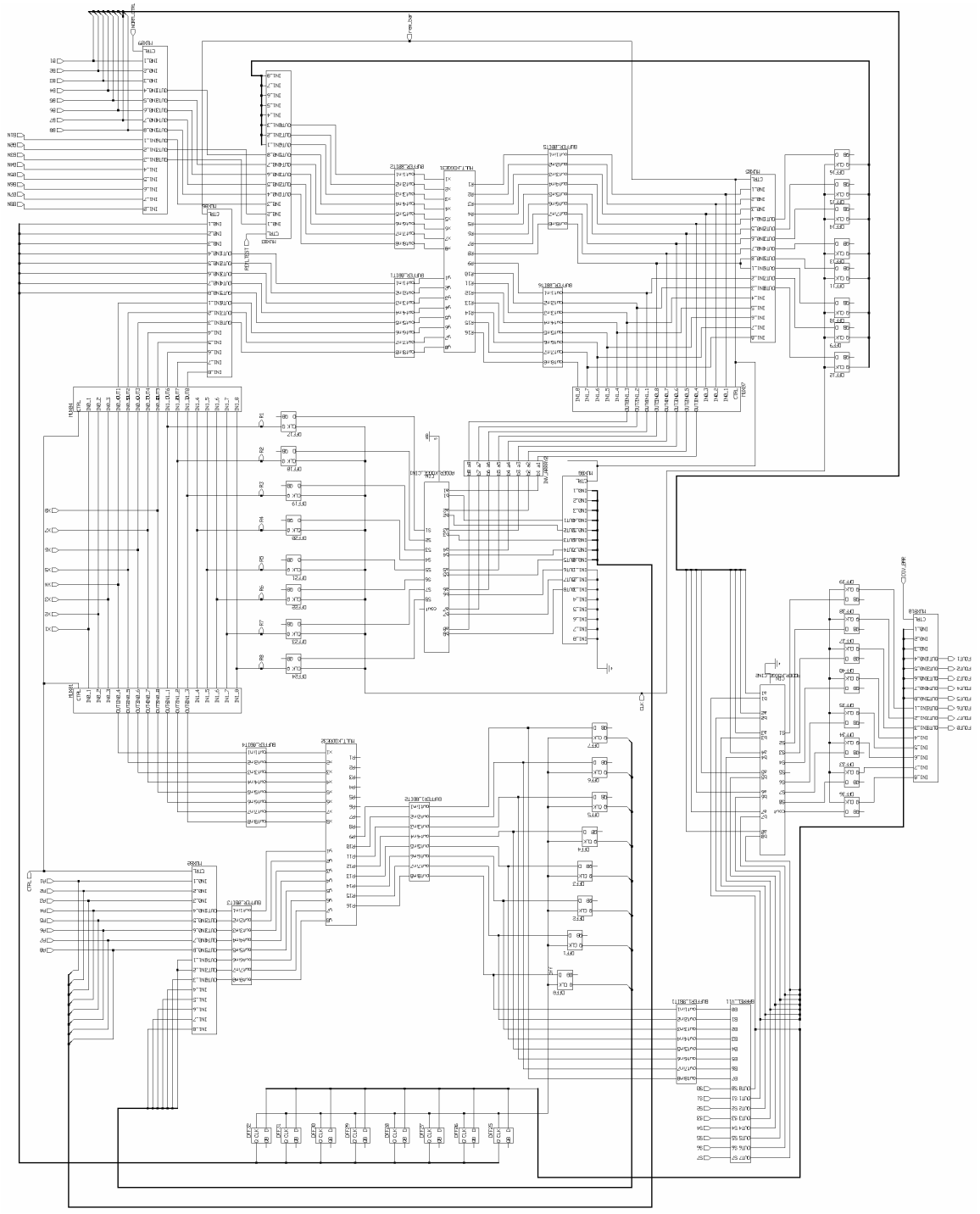


Figure 4-6: Schematic of Combined Division and Square Root Hardware

Functional Testing

Since the multiplier along with shift registers forms the basic building blocks of the divider, it can be assumed that faults in other units, if any, are observable for any given input. Hence functional testing of division hardware was performed using a worst case input.

Multiplier and the shifter constitute the major building blocks of the division hardware. Barrel shifter has a shift independent delay. Hence worst case delay of the divider occurs for worst case at multiplier. The bit positions 7,9-12 of the multiplier has longest logical depth and worst case for a multiplier occurs if the operands are large enough to toggle carry save adders in these positions. A suitable input to perform this was selected to test the divider.

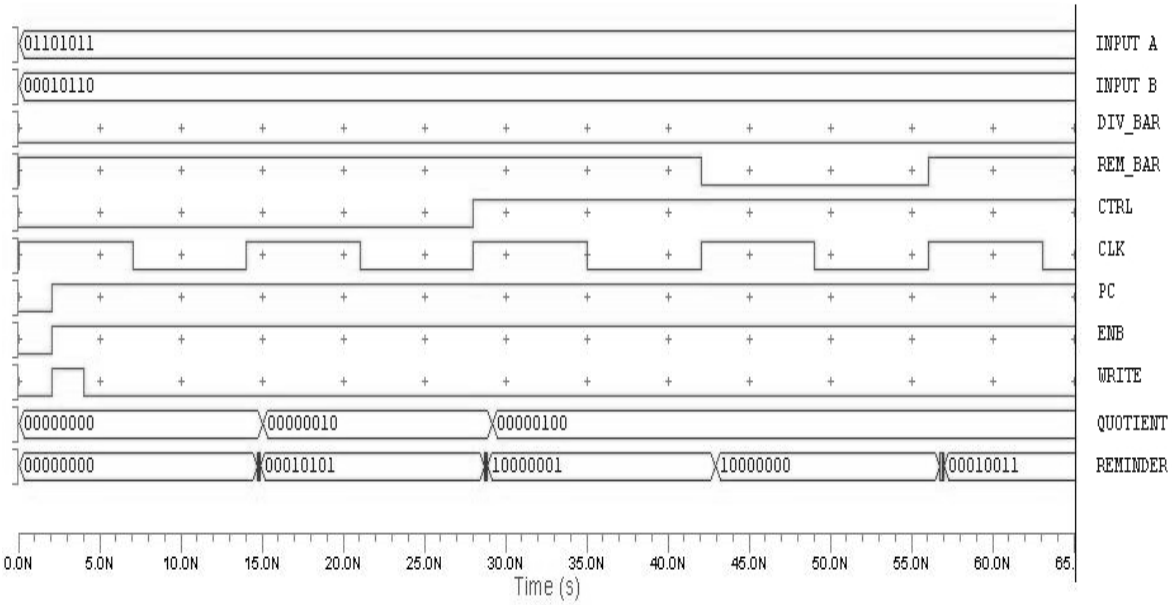


Figure 4-7: Output of Division Unit

A = 107_d; B = 22_d; Quotient =4 ; Reminder = 19

Since the division hardware encounters additional delay of an 8 bit addition when used for determining square root, the maximum frequency of operation of the division unit was fixed to be 70MHz.

Square Root

Square root is found to occur 10 times less frequently in a program than division. Due to large area and cost involved, a stand alone unit for square rooting is not justifiable. However a slow square root may still bring down the overall efficiency of the processor. In many of the modern processors FP division unit is used to compute square root as well.

In this thesis, Newton Raphson Iterative Square root algorithm was used to compute integer square root. The NR formula for square root computation is given by,

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{N}{x_i} \right)$$

Divide by 2, needed for this can be achieved by the use of a hard wire shift by one position. Fast Kogge Stone adder was used again to compute the sum. An initial approximation was obtained from a LUT to increase the speed of computation.

The proposed hardware for square root approximation consists of a priority encoder to determine the size of the operand accurate to MSB bit. The encoder output is used to address the Look up table to obtain square root approximation. Based on square root approximation, inverse approximation for division is obtained from a different table.

Use of a look up table results in reducing the number of iterations needed as in case of division.

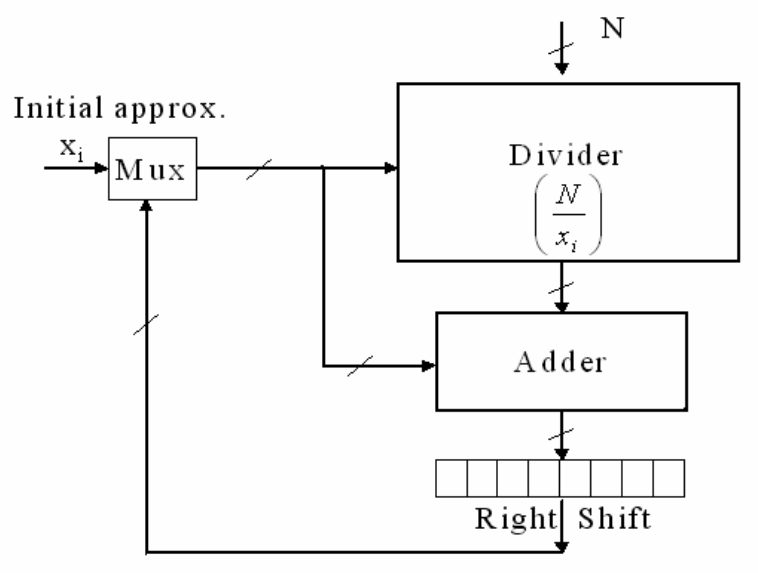
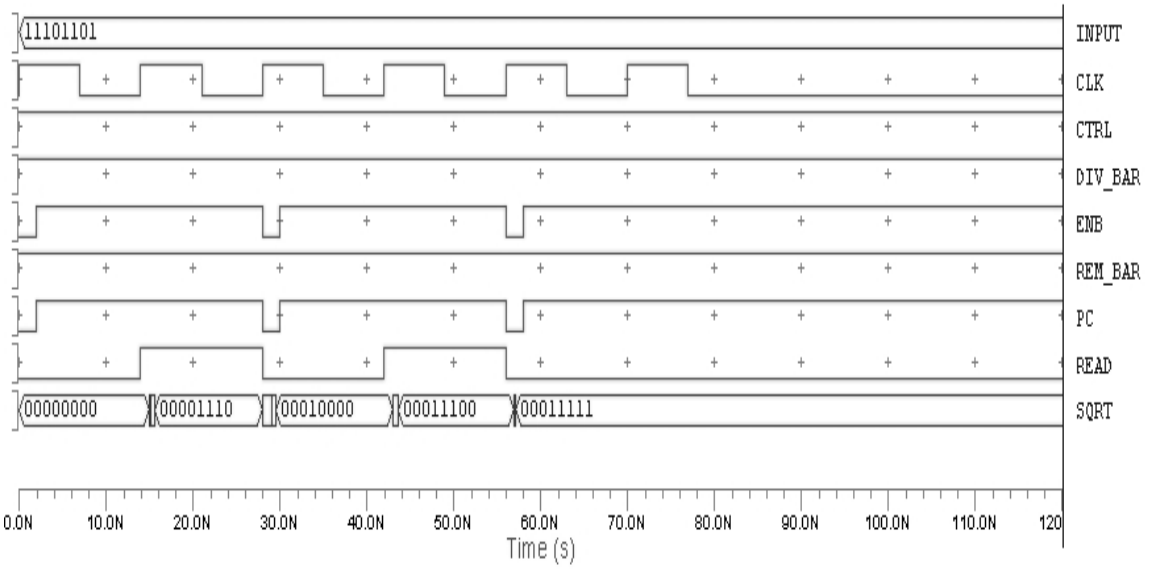


Figure 4-8: Block Diagram for Square Root computation

OPERAND	PRI. ENC. O/P	SQRT APPROX.
1X	001	1
1XX	010	10
1XXX	011	11
1XXXX	100	100
1XXXXX	101	101
1XXXXXX	110	1000
1XXXXXXX	111	1011

Table 4-3: Square Root Look Up Table

The division hardware needs an additional SRAM table to complete the circuitry needed for square root computation. Though square root approximation requires additional hardware, it doesn't result in any performance overhead on the division circuit. The value of square root approximation was directly supplied in this implementation. The cycle time for division is 14ns, allowing another memory access in normalization cycle with inclusion of appropriate hardware. Since the maximum output is 4 bit, square root computation requires two more NR iterations resulting in 4 clock cycles.



*LSB needs to be ignored to account for 'divide by two'.

Figure 4-9: Output of Square Root Computation

INPUT = 237 ; SQRT = 15

Square

If booth encoding is not used, multiplier architecture can be modified to implement an exclusive squaring hardware with only half the partial product rows as the multiplier. However when an encoder is employed both multiplication and squaring generates same number of partial product rows. Hence multiplier was made to handle squaring as well.

Based on CTRL signal the hardware can be operated as a multiplier or a squaring unit. When CTRL is 'High', the hardware performs X*Y else it determines X^2 .

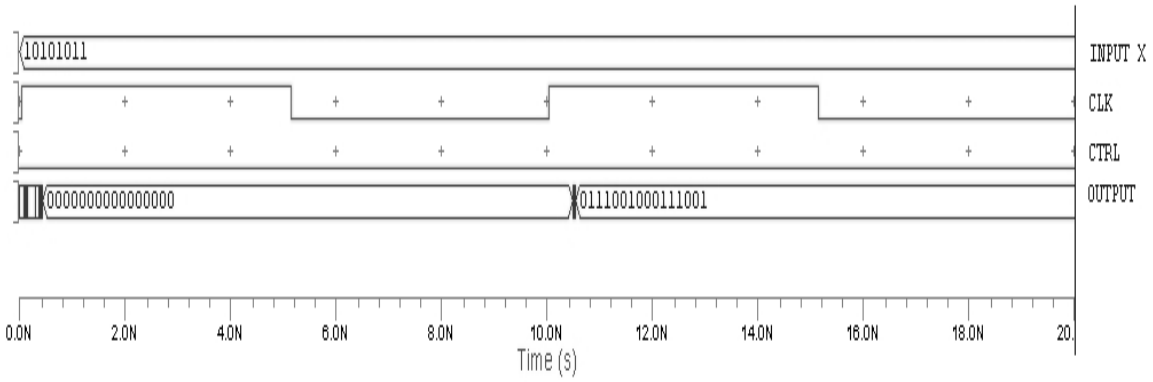


Figure 4-10: Output of Square computation.

X- 171_d; X^2=29241

Inverse

Division hardware was used to implement inverse as well. To facilitate inverse computation, an encoder circuitry was built in the division hardware along with normalization circuitry. The encoder was used for computation of mantissa involving negative power of final result based on the original divisor. The delay obtained for

inverse is essentially same as that for a division. Since inverse doesn't involve calculation of a reminder, only 3 cycles are needed to compute inverse. An 8 bit exponential result and a 3 bit mantissa are produced by the hardware during inverse computation. Final 8 bit result is obtained by replacing the 3 least significant bits of the exponential term with encoder output. Thus result of inverse is a 5 bit exponential term and a 3 bit mantissa carrying a negative power.

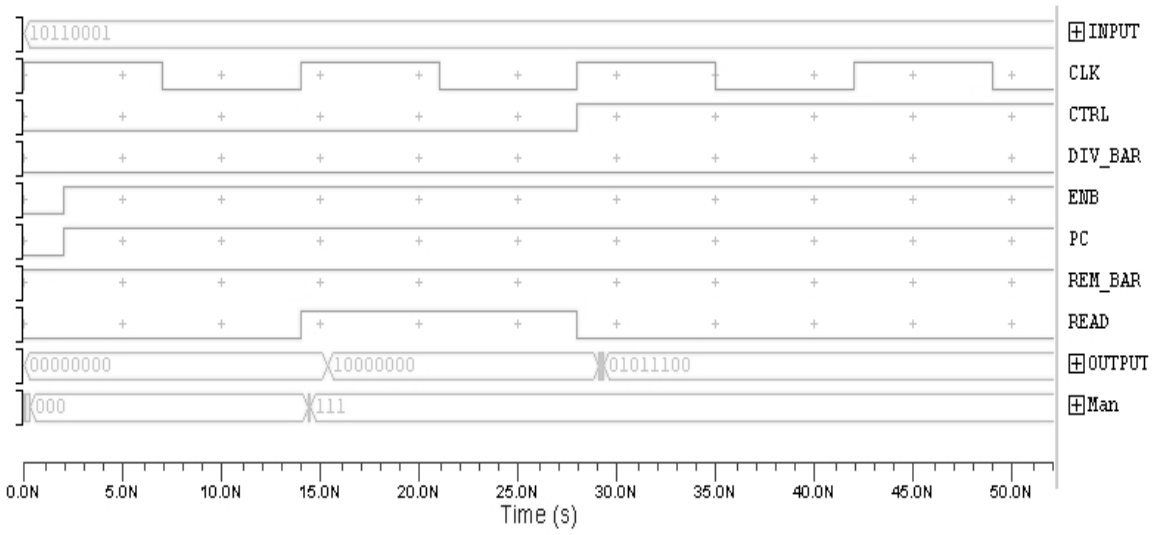


Figure 4-11: Output of Inverse Computation.

Y =177 ; Inverse = $\frac{23}{2^5} * 2^{-7} = 0.0056$

This section concludes the work on arithmetic unit. Since the slowest unit determines the overall operating speed, the optimum operating frequency of the processor is 70MHz. In many modern processors, the adder is operated at twice the clock frequency to improve performance. Adder designed in this work supports such performance

enhancement and can be operated at higher speed. Worst case performance analysis as well as power estimation for the arithmetic unit is presented in Chapter VI. The following chapter addresses the design of SRAM cell and the peripheral circuits needed to handle R/W memory requirements of the process.

CHAPTER FIVE

MEMORY DESIGN

SRAMs (Static Random Access Memory) are high speed, low power random access memories with high noise margin and full compatibility with current logic process. Despite its higher cost, SRAM is the most popular choice for high speed memory. In addition to use as cache in high performance microprocessors, SRAM memory cells area also used as data buffers and as fast Look-Up-Tables (RAMDAC - Random Access Memory Digital-to-Analog Converter).

SRAM ORGANIZATION – Single Column

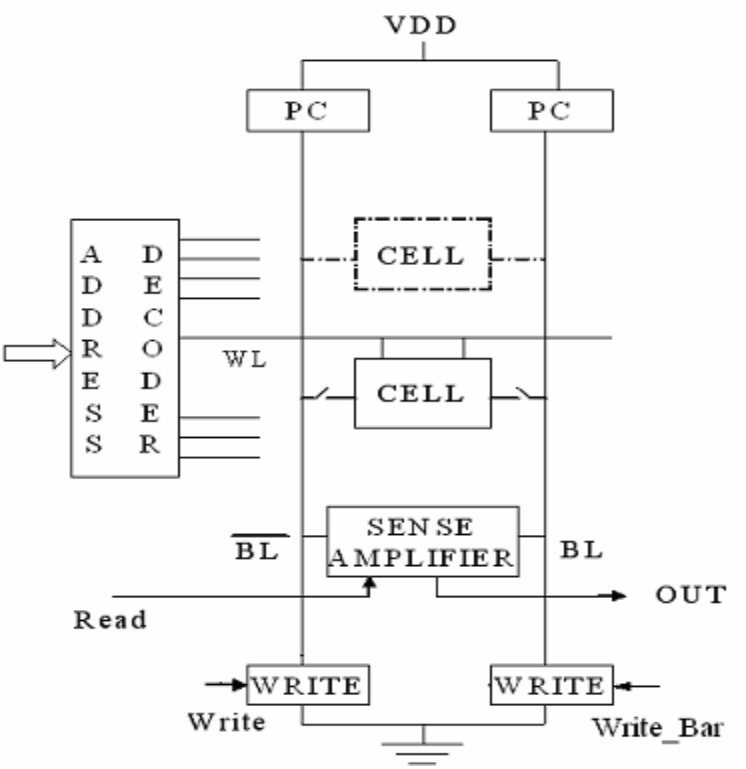


Figure 5-1: Organization of Single Column of SRAM Memory Array

A single column of SRAM array is shown in Fig 5.1. The key components are,

1. SRAM Cell
2. Column Pull up (or) Pre Charge Circuitry
3. Write Circuitry
4. Read Circuitry

A 6T SRAM cell consumes low power and has a high static noise margin when compared to other topologies like 4T and 4TLL. Hence the 6T cell topology is more popular than other SRAM configurations despite its comparatively bigger size. Address decoders are used to select appropriate word line (WL) based on memory address. Column Pull up consist of PMOS transistors that pre charge the bit line to V_{dd} before each read. Sense amplifier amplifies the reduced voltage swing across the bit lines into a full logic level signal. Write circuitry consists of NMOS transistors to discharge appropriate bit line capacitance and set the cell voltage at the desired node to zero.

6 T SRAM Design

A 6T SRAM cell consists of a positive feedback amplifier formed by two cross coupled inverters. The cell has two stable states and preserves one of its stable states as long as the power supply is present. Regeneration towards one of the stable states will occur if the gain of the inverters is greater than one. Since the two inverters complement each other, the logic level is maintained by the cell as long as power supply is on and voltage level at the nodes is not externally disturbed. In addition to the inverter pair, the cell consists of two n type access transistors that connect the data to the bit line. The

access transistors are enabled by row select signal which connect appropriate cell to bit line for read and write operations.

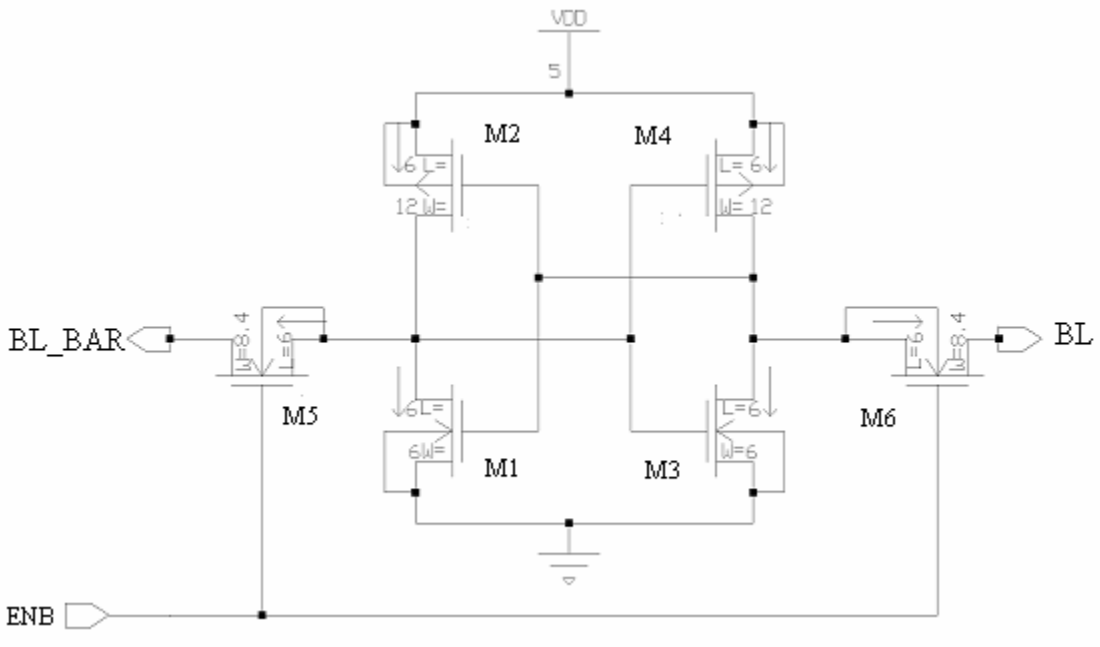


Figure 5-2: 6-T SRAM Memory Cell

To write data, the bit lines are pre charged to the value to be stored and the access transistors are enabled. The new logic state is set by forcing the bit line voltage on the internal nodes for sufficient interval of time. Faster write can be achieved by increasing the size of write transistor.

To read data, both the bit lines are initially pre charged to a high value. When the access transistors are enabled, the side with stored ‘0’ discharges the bit line voltage. Direct read of the memory cell is time consuming since it requires discharge of large bit line capacitance by the cell. To increase the speed of operation, the cell is allowed to discharge a small voltage and a sense amplifier is used to convert the small differential

voltage to full logic level output. The pre charge transistors can be sized according to bit line capacitance and the frequency of operation.

Design

Area and speed optimization remain the key objective in SRAM cell design. Since a 6T SRAM cell is symmetric, design involves sizing of only three transistors. For design simplicity, long channel MOS equations were used for cell sizing.

Data Read

For read operation, bit lines are initially pre charged to V_{dd} and the access transistors corresponding to the desired cell are enabled by asserting the appropriate word line. A differential voltage develops across the bit line which is later sensed to determine the value stored in the cell. Assuming that BL_BAR stores a low value, the transistors M1 and M5 will be ON and will begin to discharge the bit line capacitance. The discharge current increases the node voltage Q_bar. Logic level on Q should not be disturbed during the read to avoid read upset. Hence the maximum allowable ripple at node voltage at Q_bar is less than the threshold voltage of M3.

The relationship between the transistor sizes and change in node voltage at Q_bar can be used to determine the device dimensions. A plot of maximum value of ripple voltage at the node as a function of Cell Ratio was used to determine safe operating conditions for the cell.

Transistor M5 is in saturation region and M1 is in linear region. The fact that same current flows through both the devices is used to determine the size-voltage relation.

$$k_{n,M5}\left((V_{DD}-\Delta V-V_{Tn})V_{DSATn}-\frac{V^2{}_{DSATn}}{2}\right)=k_{n,M1}\left((V_{DD}-V_{Tn})\Delta V-\frac{\Delta V^2}{2}\right)$$

$$\Delta V=\frac{V_{DSATn}+CR(V_{DD}-V_{Tn})-\sqrt{V^2{}_{DSATn}(1+CR)+CR^2(V_{DD}-V_{Tn})^2}}{CR}$$

For long channel devices, saturation voltage is given by $V_{dsat}=(V_{DD}-V_{Tn})$,

$$\Rightarrow \Delta V=(V_{DD}-V_{Tn})\left(\frac{1+CR-\sqrt{(1+CR)+CR^2}}{CR}\right) \quad \text{Where, } CR=\frac{S_1}{S_5}$$

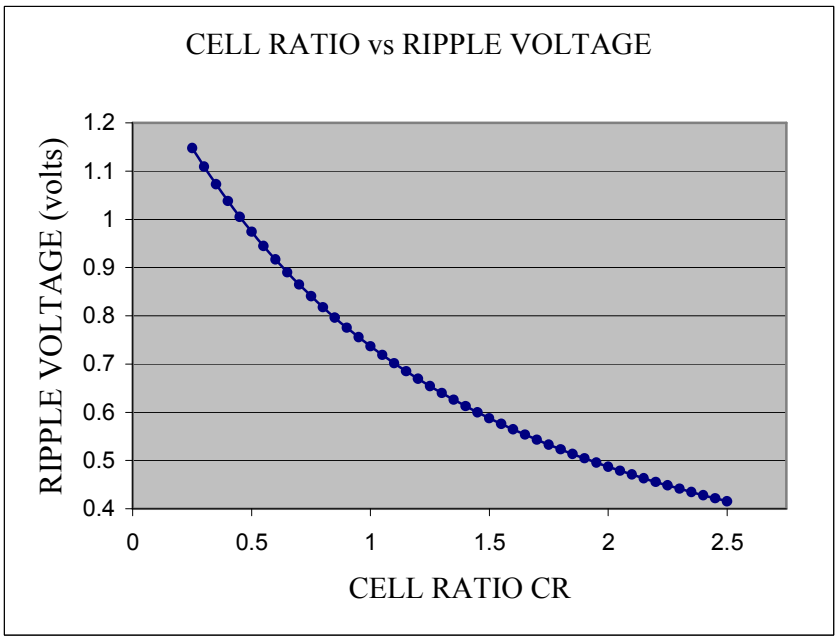


Figure 5-3: Ripple Voltage Vs Cell Ratio

Data Write

To design the cell for proper write operation, let us assume that Zero has to be stored in Q. To store zero, the bit line BL needs to be pulled down to force the cell to a new logic state. This requires that the pass transistor M6 be stronger than the pull up M4 to force the internal node to the new value. Pull down ratio was determined in similar manner as CR by analyzing the current through M4 and M6.

$$k_{n,M6}\left((V_{DD}-V_{Tn})V_Q-\frac{V^2_Q}{2}\right)=k_{p,M4}\left((V_{DD}-|V_{Tp}|)V_{DSAT_p}-\frac{V^2_{DSATp}}{2}\right)$$

$$V_Q=V_{DD}-V_{Tn}-\sqrt{(V_{DD}-V_{Tn})^2-2\frac{\mu_p}{\mu_n}*PR*\left(\left(V_{DD}-|V_{Tp}|\right)V_{DSATp}-\frac{V^2_{DSATp}}{2}\right)}$$

$$PR=\frac{S_4}{S_6}$$

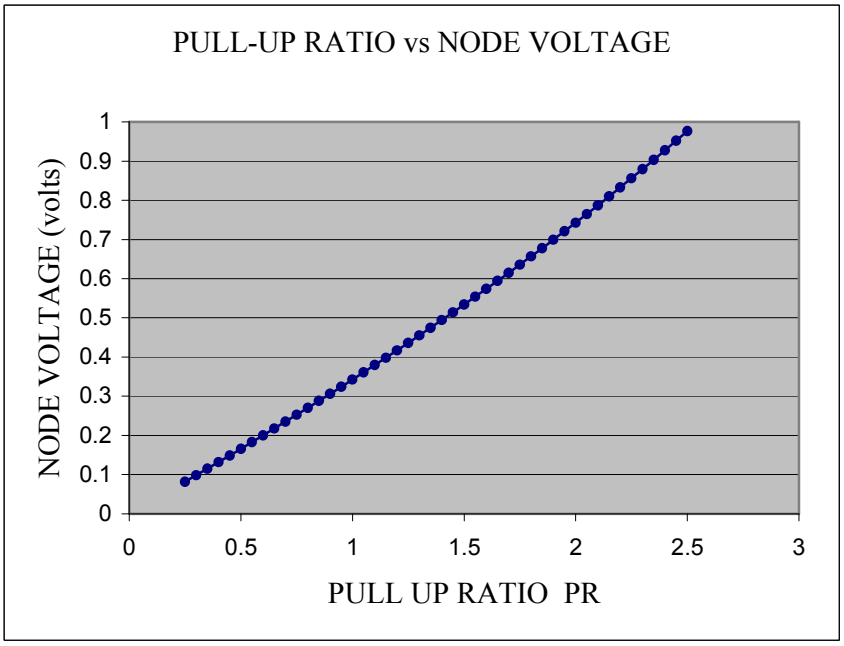


Figure 5-4: Node Voltage Vs Pull Up Ratio

Sizing and Performance

From Figure 5-3, Cell Ratio (CR) needs to be greater than 1.7. This can be achieved by either increasing the size of pull down transistors or decreasing the size of access transistors, while keeping the size of the other transistor minimum. Decreasing the size of access transistor increases the load on word line and is not generally preferred. Increasing the size of pull down devices increases the storage capacity of the cell while minimizing the load on word line.

The Pull Up Ratio (PR) needs to be less than 1.5 to ensure proper Write. Hence size of both the transistors is kept at the minimum value permissible by layout rules.

TRANSISTOR	L	W	CR	PR
M1, M3	0.4	1.6	2	1
M5,M6	0.5	1		
M2,M4	0.5	1		

Table 5-1: 6T Transistor Sizing.

Sense Amplifier

A sense amplifier converts the reduced signal swing in the form of a differential voltage across the bit lines to a full logic level voltage at its output. A differential sense amplifier offers high read speed and is generally used in SRAM memories. However only SRAM cells generate differential output and alternative amplification methods need to be used for other types of memory like DRAM, ROM, and EEPROMs. A Latch type sense amplifier, a common choice for DRAM memories, was also implemented in this thesis along with high speed differential sense amplifier.

Typical Size of Cache Memory in some commercial processors is tabulated below. The performance of the SRAM cache was analyzed later by determining read time for an array equivalent of a P4 L1 cache.

PROCESSOR	CACHE	SIZE	TYPE	# LINES
Pentium 4	L1 (64)	8KB	4 Way SA	1024
	L2(128)	512KB	8 Way SA	32768
Athlon 64FX	L1(64)	64KB	2 Way SA	8192
	L2(64)	1 MB	16 Way SA	128000

Table 5-2: Cache Memory in Pentium 4 and Athlon Microprocessors.

Sense Amplifier for Division Look Up Table

Look up needed for division hardware is taken care of by a single row of SRAM cell. The bit line capacitance of a single SRAM row is 2.5fF, which can quickly be discharged by the cell itself. An inverter attached to BL_BAR was used to read the data stored in the cell since a sense amplifier is not needed to read the cell.

The SRAM cell was found to have an access time of 0.75ns even without the use of a sense amplifier. Addition of bit line capacitance would increase the access time and require use of a sense amplifier.

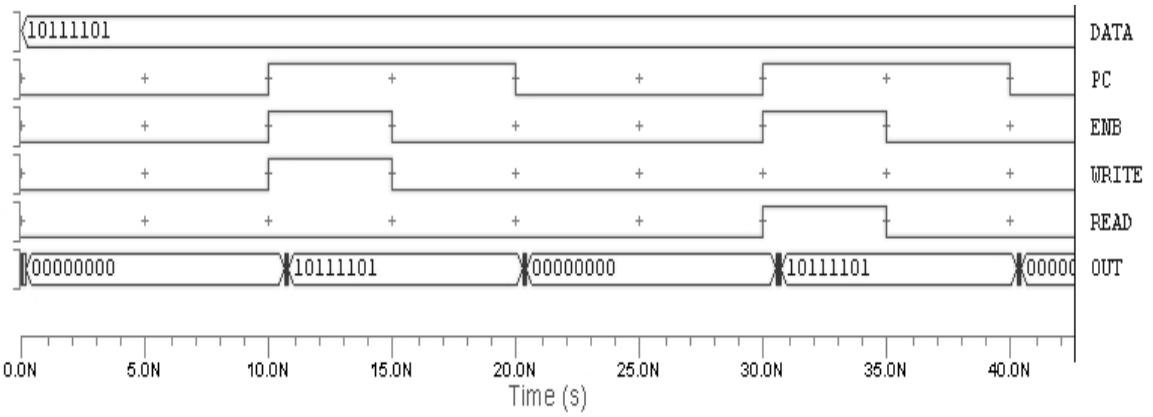


Figure 5-5: Output of single SRAM row.

Differential Sense Amplifier

A differential sense amplifier was built using a NMOS differential pair and a current mirror load. A current mirror load enables conversion of a differential signal to a single ended output with high common mode rejection ratio. The amplifier was designed for a gain of 15.

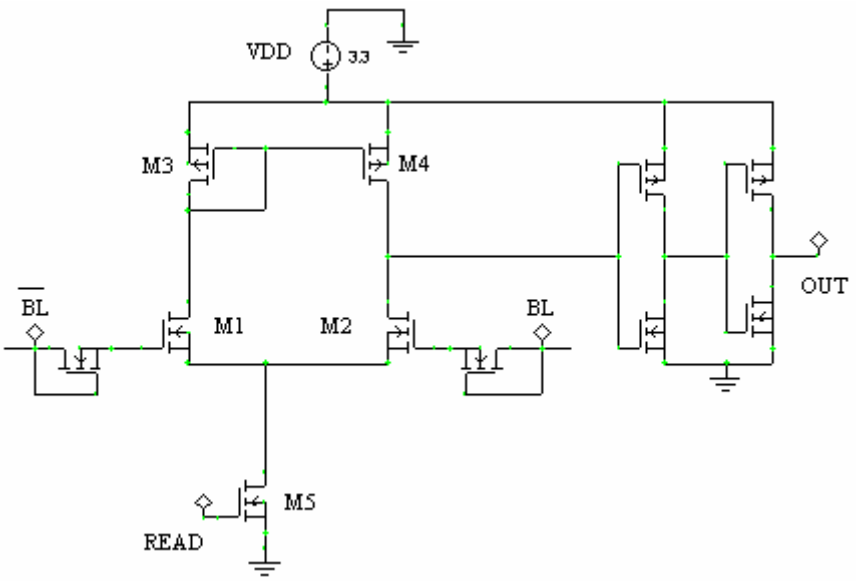


Figure 5-6: Schematic of CMOS Differential Sense Amplifier with Inverter

Output of the sense amplifier is generally connected to a driver to prevent loading of the amplifier. In this design, single ended output from the sense amplifier was connected to an inverter. Since inverter acts as a regeneration circuit, no constraints were imposed on the output swing during design of the amplifier.

$$A_{Sense} = -g_{m1}(r_{O2} \| r_{O4})$$

$$g_{m1} = \sqrt{\left(k_n * I_5 * W_1 / L_1\right)}$$

The above equation indicates that the gain of the amplifier can be increased by either increasing r_o of load transistors or by increasing the transconductance of the differential pair. The transistors are operated in the saturation region and r_o tends to be a high value in saturation mode of operation. Transconductance of the differential pair can

be increased by increasing both W/L and bias current. Care is needed when designing a high gain amplifier since increase in I_5 decreases r_o. Since gain required for this design is small, this was not of any concern for this design.

Gain

$$A_{Sense} = \sqrt{\frac{2 * K_n * \left(\frac{W}{L}\right)_1}{I_1 * (\lambda_p + \lambda_n)^2}}$$

Slew Rate and settling time

$$SR = I_5 / C_L$$

Common mode input range is another important design parameter for the sense amplifier.

$$IC\_\min = V_{DS5(sat)} + V_{GS1} = V_{DS5} + \sqrt{\frac{I_5}{K_n * S_1}} + V_{tn}$$

$$IC\_\max = V_{DD} - V_{DS3} = V_{DD} - (V_{GS3} - V_{tn})$$

Design

A MATLAB program was used to design the differential amplifier. Since the common mode range and output swing of the amplifier is not critical, attempt was made to minimize the size of transistors while keeping these values at an acceptable range. Plugging in the values used in the program, the size of transistors is derived below.

$I_5 = 100^{-6}$ A

$$\left(\frac{W}{L}\right)_1 = \frac{A^2_{Sense} * I_1 * (\lambda_p + \lambda_n)^2}{2 * K_n}$$

$$\left(\frac{W}{L}\right)_1 = \frac{15^2 * 50 * 10^{-6} * (0.04 + 0.06)^2}{2 * 90 * 10^{-6}} = 1.11 \sim 1$$

Designing S5 and S3 for Input Common Mode Range,

$$V_{DS5} = (V_{IC\_\min} - V_{GS1}) = V_{IC\_\min} - (V_{DS1} + V_{tn})$$

$$V_{DS1} = \sqrt{\frac{I_5}{K_n * S_1}}$$

$$\left(\frac{W}{L}\right)_5 = \frac{2 * I_5}{K_n * (V_{IC\_\min} - V_{DS1} - V_{tn})^2}$$

$$\left(\frac{W}{L}\right)_5 = \frac{2 * 100 * 10^{-6}}{90 * 10^{-6} * (0.5 - 1 - 0.55)^2} = 2$$

S3 depends on maximum common mode input and can be calculated using the following relation,

$$V_{GS3} = V_{DD} - V_{IC\_\max} + V_{tn}$$

$$V_{GS3} = 3.3 - 2.8 + 0.55 = 1.05$$

$$\left(\frac{W}{L}\right)_3 = \frac{I_5}{K_p * (V_{GS3} - |V_{tp}|)^2} = \frac{100 * 10^{-6}}{35 * 10^{-6} * (1.1 - 0.68)^2} = 12.9$$

Power Consumption:

$$Power = I_5 * V_{DD} = 100 * 10^{-6} * 3.3 = 0.33mW$$

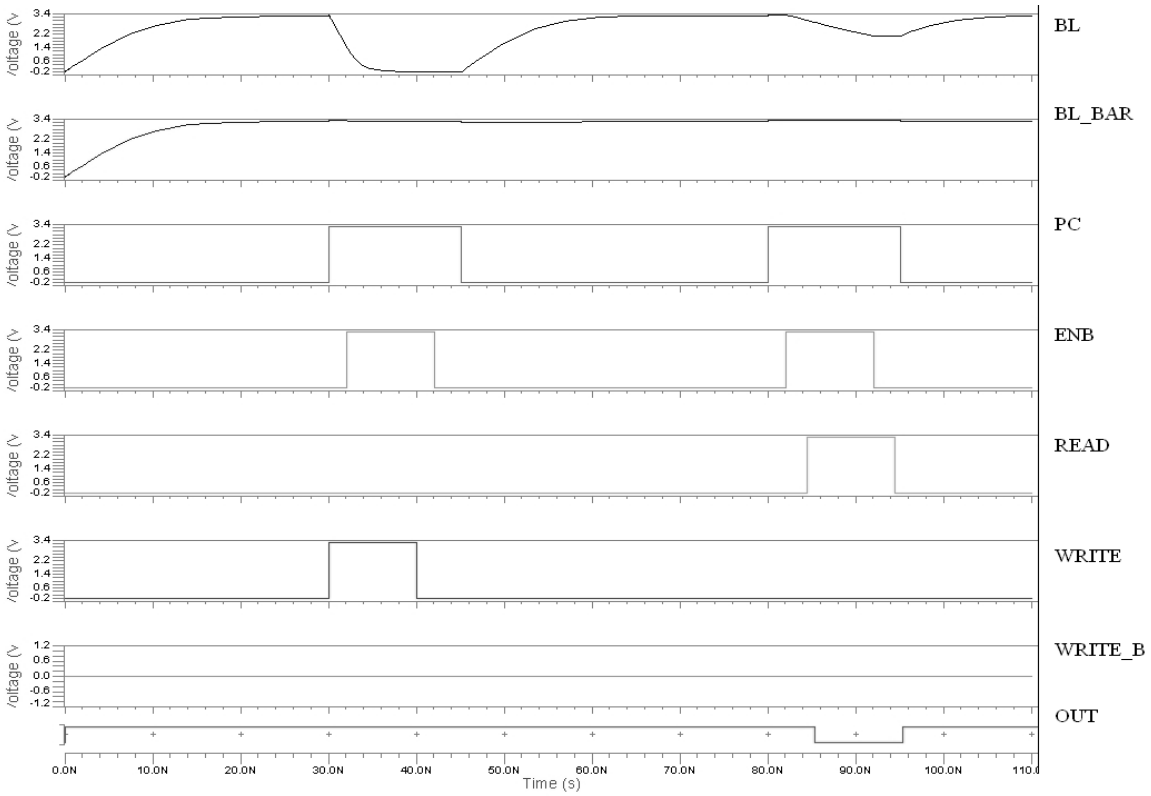


Figure 5-7: Cache Output for Write and Read of '0'.

The access time for various bit line capacitances was determined and plotted to characterize the performance of SRAM and the differential sense amplifier. As bit line capacitance increases, the time needed for the cell to discharge bit line by 300mV increases, resulting in increased access time. The read time of the sense amplifier was fairly constant around 0.9ns. The access time can be seen to exhibit a linear dependence with bit line capacitance. For a bit line capacitance of 2.56pF, which corresponds to equivalent bit line capacitance of a P4 L1 cache, the access time was found to be 3.27ns.

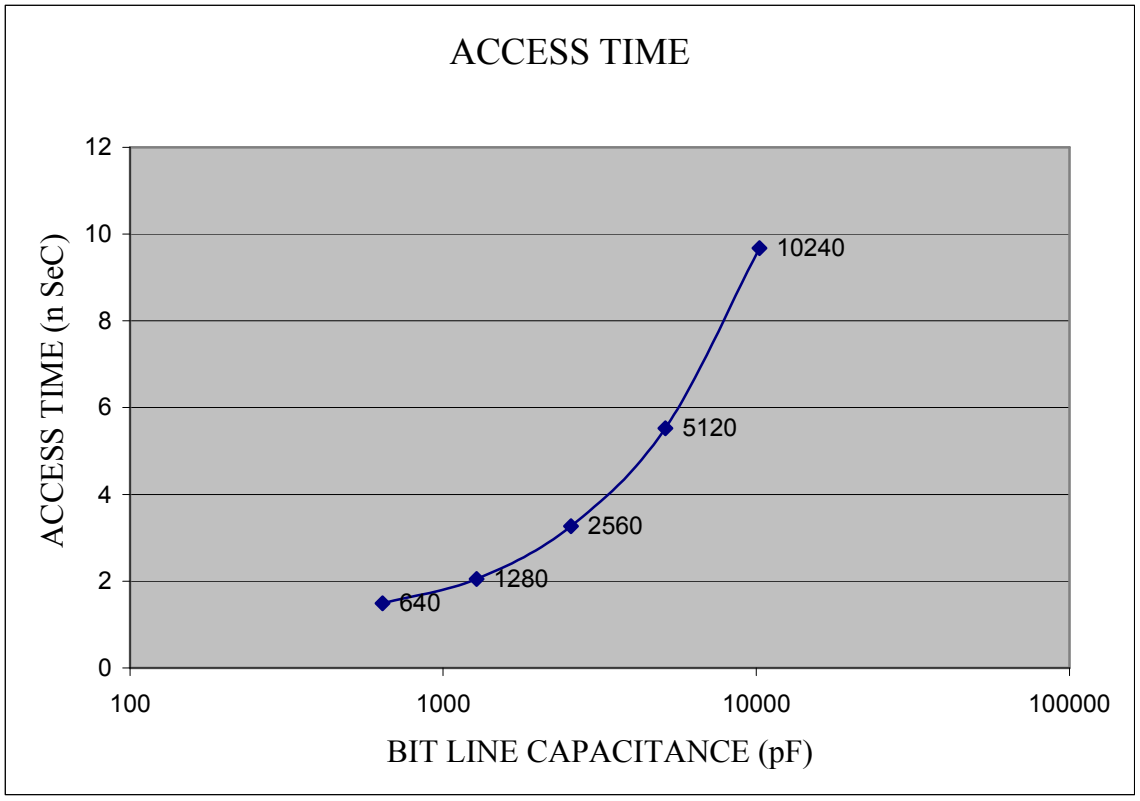


Figure 5-8: Bit Line Capacitance Vs Access Time

Though bit line capacitance was calculated and substituted, the word line capacitance was ignored in this analysis. Since word line is made of POLY, large word line contributes to a larger RC affecting the transient performance. In actual implementation of a cache, the word line capacitance needs to be taken in to account and the drive strength of decoder circuit should be improved accordingly.

Latch Type Sense Amplifier

Latch type sense amplifier can be used with different types of memories while the differential pair is useful only with an SRAM. The time needed for read is much higher than that with a differential sense amplifier due to the need to fully discharge the bit line capacitance. Hence with this sense amplifier, Set Up time and Read Time are both dependent on bit line capacitance.

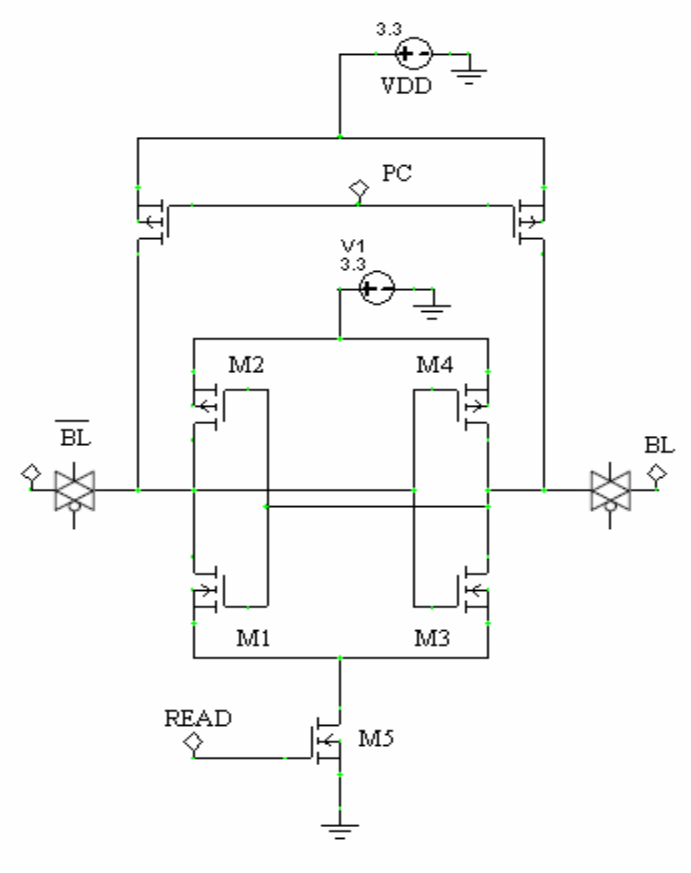


Figure 5-9: Schematic of Latch Type Sense Amplifier.

A Latch type sense amplifier consists of two cross coupled amplifiers, similar to the SRAM cell, and a clocked pull-up and pull-down. The sense amplifier is generally

forced to an unstable state by pre charging both the transistors to $V_{dd}/2$ prior to read and then allowed to sense the differential voltage. The side with lower voltage drops to zero regenerative amplification while the other bit line stays at high value.

This thesis implemented a variation of common sense amplifier [Howe et. al]. In this method, the sense amplifier is held in an unstable state, by driving both the inverters by V_{dd}. Since both the inverters are driven by V_{dd}, a p-channel gating transistor is not needed for this design. The bit line with lower voltage gets discharged when a differential voltage is applied to the sense amplifier.

Design

The Latch based sense amplifier was designed to have a sense time of 2ns for a 1pf bit line capacitance. Sense time of the cross coupled inverter is given by,

$$t_{sense} = \frac{C_{Bit}}{g_{m_n}} * \ln\left(\frac{\Delta V_{OUT}}{\Delta V_{IN}}\right)$$

Therefore to achieve a desired sense time, the transconductance of the NMOS transistors need to be,

$$g_{m_n} = \frac{C_{Bit}}{t_{sense}} * \ln\left(\frac{\Delta V_{OUT}}{\Delta V_{IN}}\right)$$

For a sense time of 1ns and a drop in voltage to 50% of the initial value, the sizes of the pull down can be calculated as,

$$g_{m_n} = \frac{1pf}{2*10^{-9}} * \ln\left(\frac{3.3-1.65}{0.3}\right)$$

$$g_{m_n} = 0.85 * 10^{-3}$$

$$g_{m_n} = \left(\frac{W}{L}\right)_3 * K_n * (V_{DD} - 0.3V - V_{Tn})$$

$$g_{m_n} = \left(\frac{W}{L}\right)_3 * 90 * 10^{-6} * (3.3 - 0.3 - V_{Tn})$$

$$\left(\frac{W}{L}\right)_3 = \frac{g_{m_n}}{90 * 10^{-6} * (3.3 - 0.3 - V_{Tn})} = \frac{0.85 * 10^{-3}}{90 * 10^{-6} * (3.3 - 0.3 - 0.55)} = 3.85$$

PMOS transistors M2 and M4 were sized twice the NMOS to obtain mid point voltage equal to $V_{dd}/2$

To determine the size of NMOS gating transistors, the current through pull down was determined and equated to current through the gating transistor.

Current through pull down transistor M1 is given by,

$$I_{D1} = \frac{1}{2}\left(\frac{W}{L}\right)_3 * \mu_n * C_{ox} * (V_{DD} - V_{DS5} - V_{Tn})^2$$

$$I_{D1} = \frac{1}{2} * 4 * 90 * 10^{-6} * (3.3 - 0.3 - 0.55)^2$$

$$I_{D1} = 1.1mA$$

Since same current flows through M1 and M5,

$$I_{D5} = I_{D1} = \left(\frac{W}{L}\right) * \mu_n * C_{ox} * (V_{DD} - V_{Tn} - \frac{V_{DS5}}{2}) * V_{DS5}$$

$$\left(\frac{W}{L}\right)_5 = \frac{I_{D5}}{K_n * (V_{DD} - V_{Tn} - \frac{V_{DS5}}{2}) V_{DS5}}$$

$$= \frac{I_{D1}}{K_n * (V_{DD} - V_{Tn} - \frac{V_{DS5}}{2}) V_{DS5}}$$

$$= \frac{1.1 * 10^{-3}}{90 * 10^{-6} * (3.3 - 0.55 - 0.15) * 0.3} = 15$$

$$\left(\frac{W}{L}\right)_5 = 15$$

TRANSISTOR	W/L
M3 / M1	4
M2 / M4	8
M5	15

Table 5-3: Sizing of Latch Type Sense Amplifier.

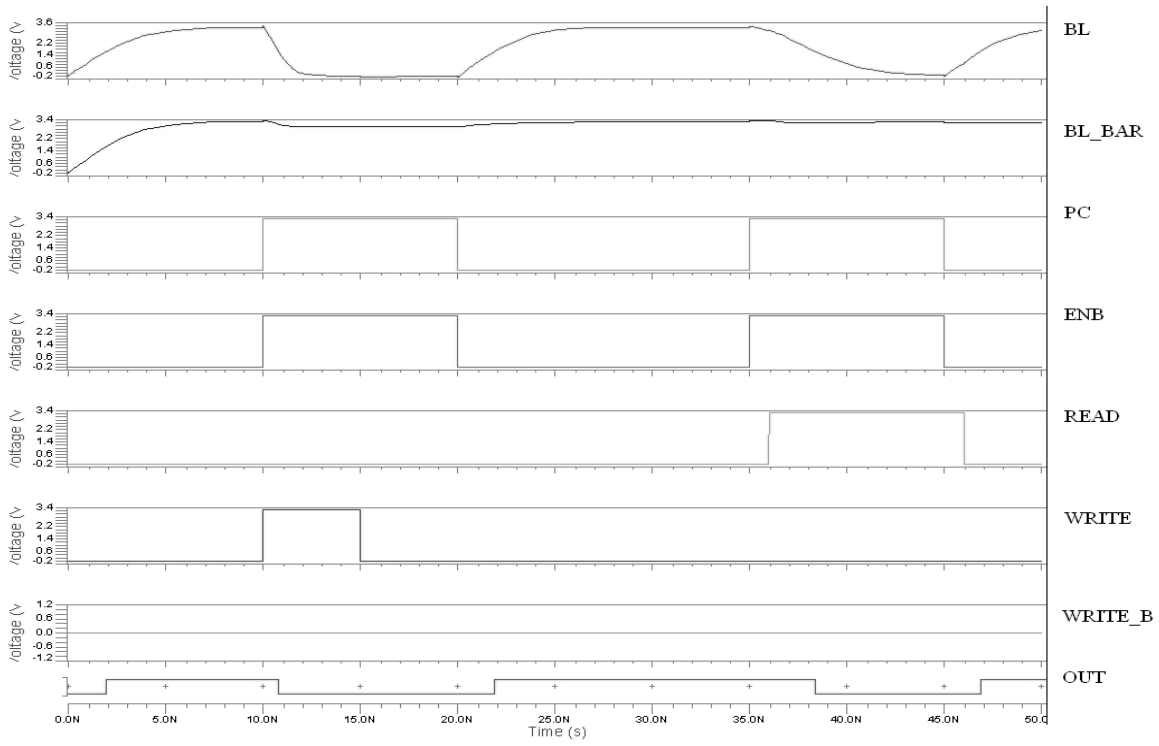


Table 5-10: Latch Type Sense Amplifier Output.

The designed amplifier was found to have a set up time of 2.23ns and a read time of 0.97ns. Latch type sense amplifiers generally require more area than a differential pair to attain comparable performance. Most of the available SRAM memories employ variations of a differential pair to attain their characteristic higher performance.

Write Circuitry

Write circuitry consists of two NMOS transistors driven by WRITE and WRITE_BAR. Based on the data to be written, one of the transistors is switched ON while the other transistor remains OFF. The size of transistors depends on the bit line capacitance and the discharge time. Write circuitry is not critical in SRAM design since it can be sized for any desired frequency of operation independent of other components.

The following example illustrates design of NMOS transistors for write circuitry to discharge a bit line capacitance of 2pF in 2ns.

Current needed to discharge bit line capacitance is given by,

$$I_{Write} = \frac{C_{Bit} * \Delta V}{t_{\Delta V}} = \frac{2pF * 3.3}{2 * 10^{-9}} = 3.3mA$$

$$\Rightarrow \left(\frac{W}{L}\right)_{write} = \frac{I_{D\_Write}}{0.5 * K_n * C_{ox} * (V_{DD} - V_{Tn})^2} = \frac{3.3 * 10^{-3}}{0.5 * 90 * 10^{-6} * (3.3 - 0.55)^2} = 10$$

$$\left(\frac{W}{L}\right)_{write} = 10$$

This chapter addressed the design of an SRAM cell and analyzed a couple of sensing options. Based on performance requirements, all these options generally find a place in a processor. A single row of SRAM, similar to division LUT, can be employed as fast registers and buffers. Latch type sense amplifier can be used as sense amplifiers for DRAM memories as well as for moderate sized SRAM arrays. Differential pair sense amplifier will be employed in large sized cache. Thus requirements for fast static RW memory in a processor can be taken care of by units designed in this work.

CHAPTER SIX

PERFORMANCE ANALYSIS AND CONCLUSION

Delay of a circuit is sensitive to variations in variety of parameters like supply voltage, temperature, noise as well as process variations. In this chapter, effect of these variations on performance of the arithmetic unit was studied to ensure that the desired performance is always achieved. Effect of these parametric variations was studied by observing the output of each unit prior to the final register. The delay due to the final register was however taken into account to test for maximum operating frequency of the circuit.

Process Variations

MOS devices often exhibit variations in process parameters. Variations in process parameters can alter the delay of a circuit by affecting drain current. To ensure that the design works under variations in process parameters, Monte Carlo analysis was performed for random variation of following process parameters.

Channel doping concentration (NCH) is an important process parameter that affects threshold voltage as well as the drain current of a gate. Gate oxide thickness (T_{OX}) affects the oxide capacitance and in turn affects drain current and delay. Drain current is also directly proportional to mobility. Change in delay due to 5% variation in the above mentioned parameters was studied in this work.

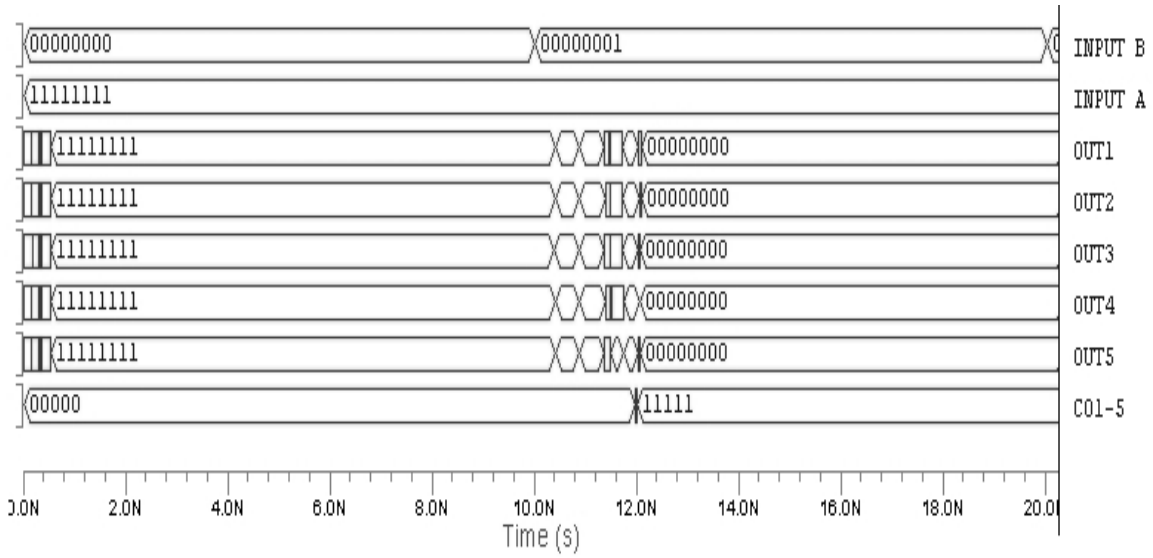


Figure 6-1: Output of Adder under process variations.

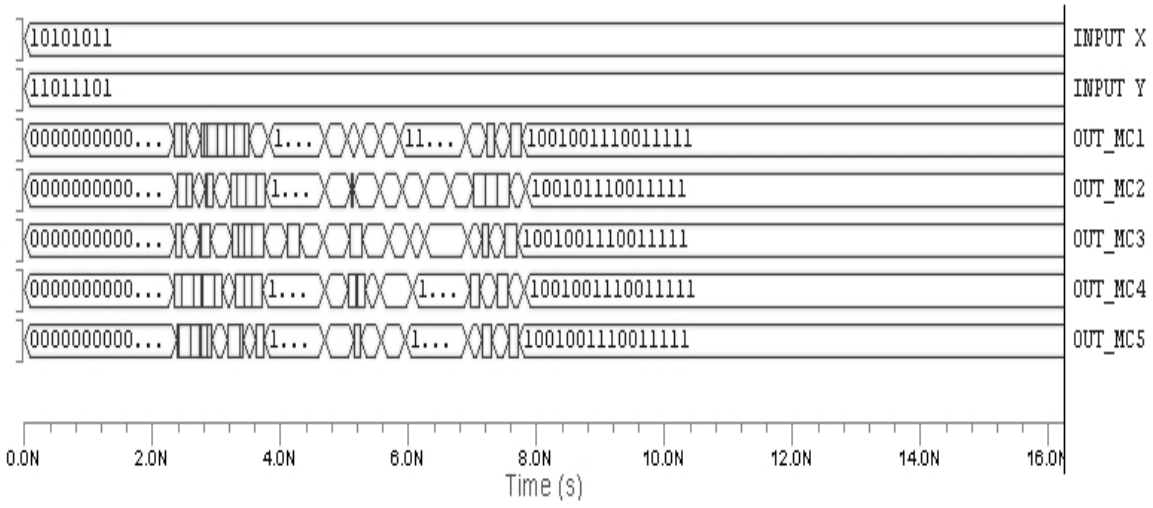


Figure 6-2: Output of Multiplier under process variations.

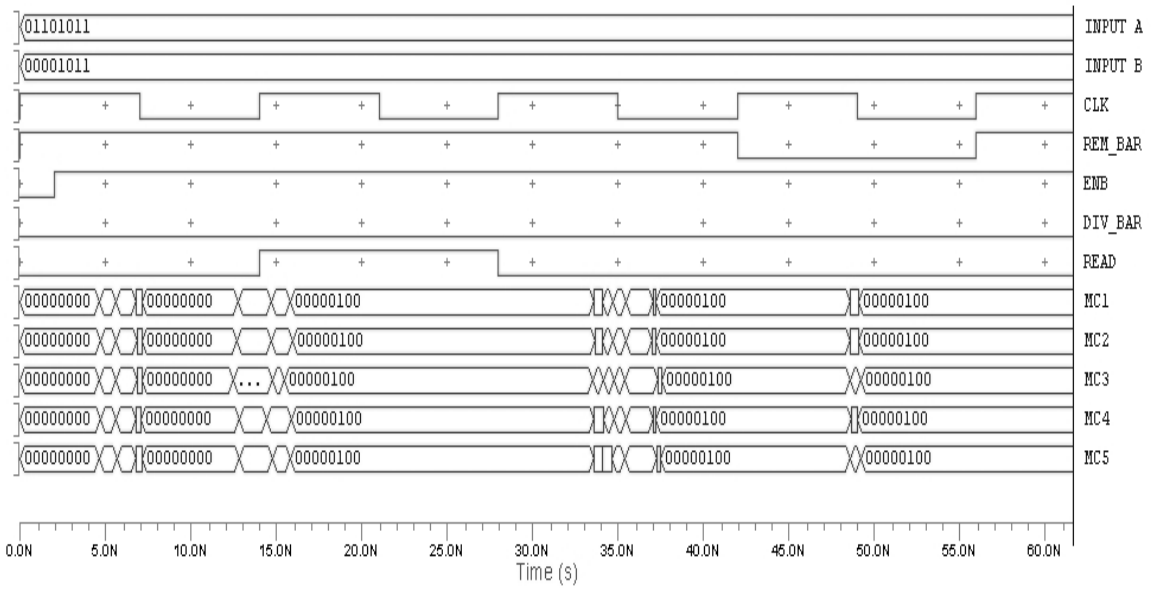


Figure 6-3: Output of Divider under process variations.

The results above confirm that the design meets the operating frequency determined earlier under process variations.

Supply Variation

In large circuits, the supply voltage is also seen to vary from one region to another due to resistance and inductance of the metal lines. Further general power supplies also exhibit fluctuations up to 10%. The delay of a gate increases with decrease in supply voltage. To ensure proper operation under voltage variation, the performance of arithmetic units were tested for power supply variation from 3 Volts to 3.6 Volts

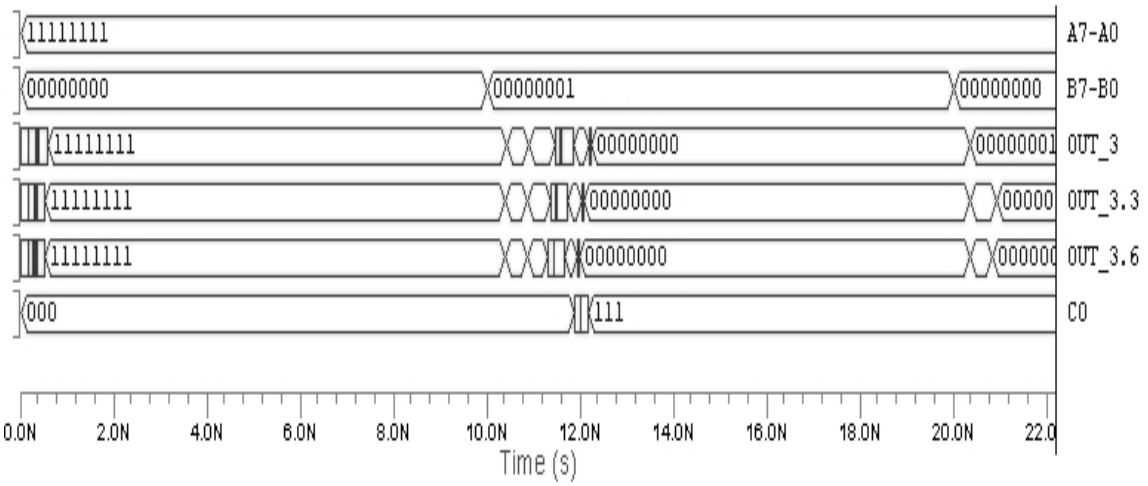


Figure 6-4: Output of Adder under Supply Variations.

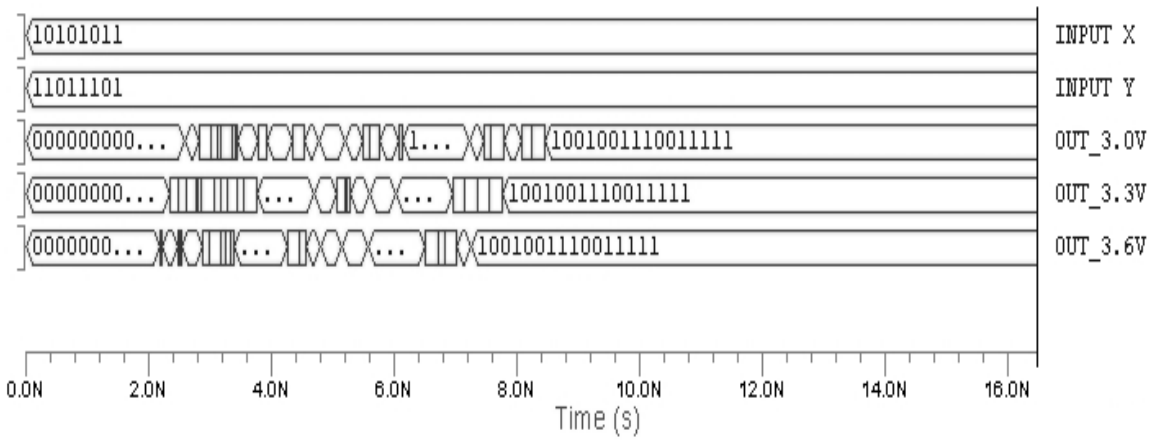


Figure 6-5: Output of Multiplier under Supply Variations.

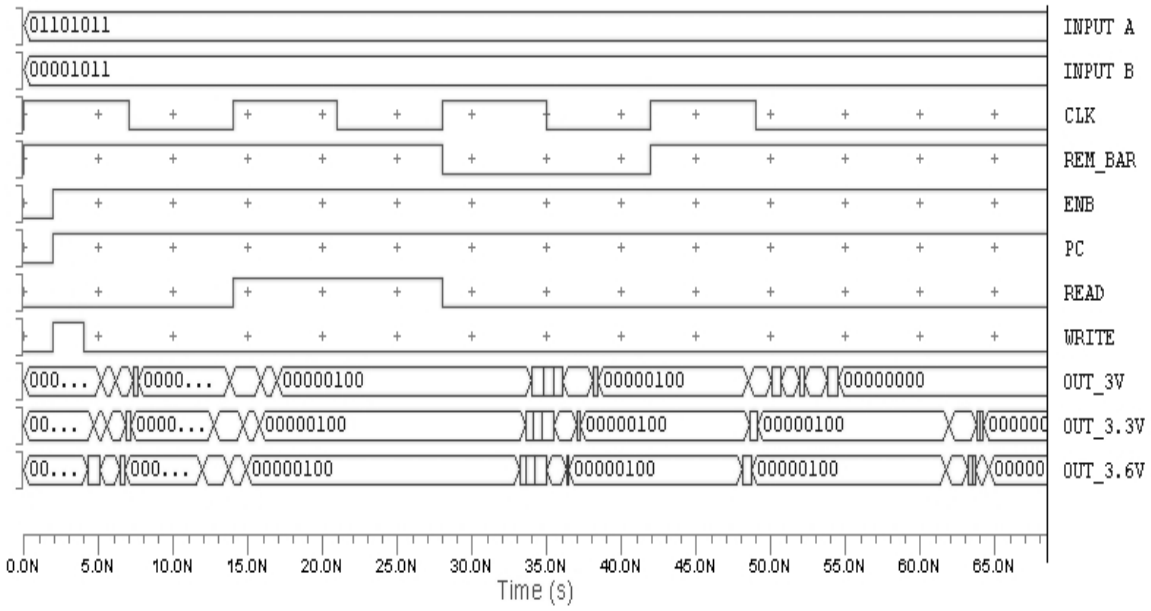


Figure 6-6: Output of Divider under Supply Variations.

Temperature Variations

Temperature represents the single major factor affecting the performance of a digital circuit. Mobility of charge carriers is found to decrease by 40% for 100 K rise in temperature. Temperature dependence of mobility is given by

$$\mu_T = \mu_o \left(\frac{T}{300^o K} \right)$$

Mobility has a direct impact on the drain current. To obtain the worst case delay due to increase in temperature, the circuit was tested over a temperature range of 20 ^oC to 80 ^oC. The response of different sub units to temperature variations is presented below.

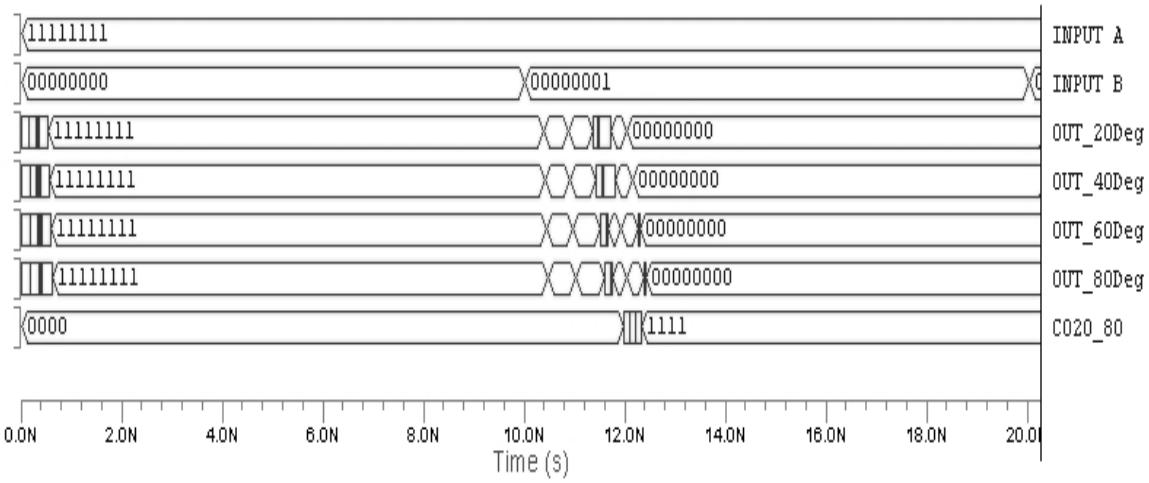


Figure 6-7: Output of Adder under Temperature variation.

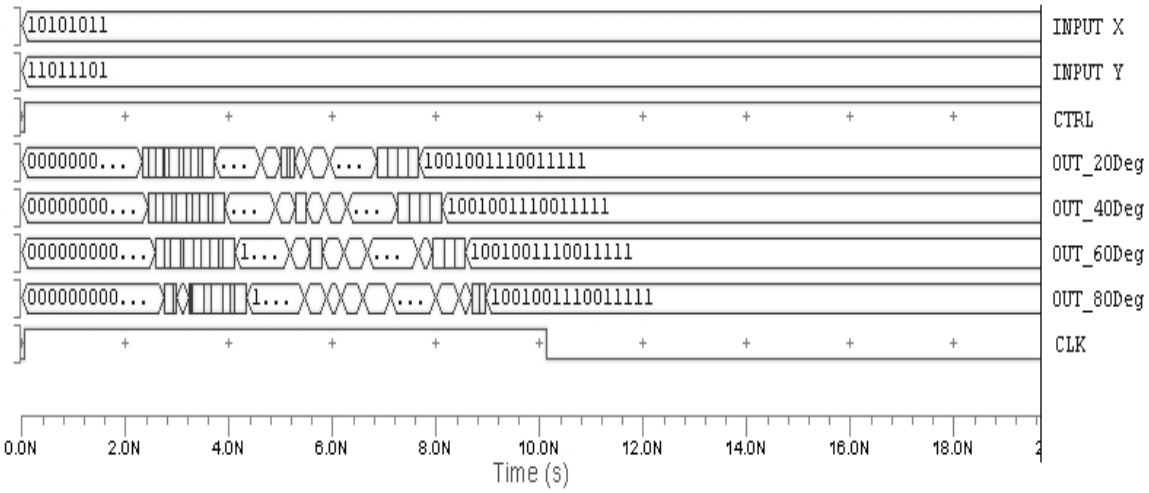


Figure 6-8: Output of Multiplier under Temperature variation.

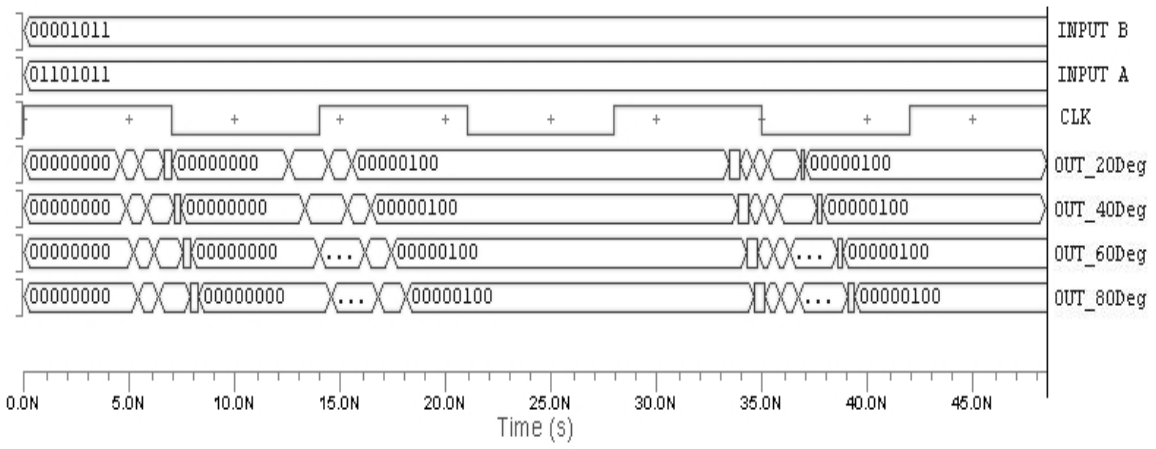


Figure 6-9: Output of Divisor under Temperature variation.

The above analysis was used to determine the worst case delay of these functional units to determine the operating frequency. The results confirm that reliable operation of these devices can be ensured for the frequencies estimated earlier.

Based on these parametric variations, it can be concluded that the components meet the operating speed for operating range. Frequency of the adder, multiplier and the division hardware are 330MHz, 100MHz and 70MHz respectively.

Power

Power consumption is an important factor that must be estimated for assessment of overall performance of a design. Higher the performance, higher is the power consumption of a circuit. In this work dynamic power consumption for basic arithmetic operations was estimated. Dynamic power is the power dissipated by a gate during

switching activity. Since switching activity in a circuit is dependent on input operands, power consumption is often difficult to estimate accurately.

Power consumption of sub units of arithmetic unit was determined using ELDO simulation. When tested with cases used for functional testing, the power consumption of the adder, multiplier and the division hardware were found to be 3.2nW, 2.38mW and 13.68mW respectively.

Summary & Conclusion

In summary, an attempt was made to design arithmetic hardware with minimum possible clock cycles. Leading architectures used in commercial processors were used for implementation of a high performance arithmetic hardware. The hardware performs integer addition, subtraction, multiplication and squaring in a single clock cycle. Division and square rooting need 4 clock cycles. This is comparable to minimum clock cycles needed in commercial processors that typically devote 4-10 times the multiplication time for division. Floating point 8-bit inverse was computed in 3 cycles. The architectures employed supports increase in operand size by duplication of basic building blocks.

A Wallace tree multiplier was built to cater to the requirement for a high speed multiplier. Multiplicative division method was used to build hardware for division and square rooting. Multiplication and division hardware primarily support integer arithmetic operations and can be extended for floating point implementations by inclusion of hardware to normalize operands and add/subtract mantissa depending on the operation.

In addition to design of high performance hardware, this work also proposed a method to test complex designs using optimal set of test vectors. This approach to testability eliminates area and performance overhead associated with conventional hardware-based testability methods while improving the speed of testing as well.

The optimal speed of operation of the arithmetic unit was found to be 70MHz. Pentium II, a 32 bit commercial processor built in a comparable process technology (0.28 micron) had a maximum operating speed of 300MHz size with 14 pipeline stages. Pipelining increases the speed of operation of a processor and a two stage pipelined processor can operate at almost twice the frequency of an un-pipelined processor with exclusion of delay due additional pipeline buffers. Parallel architectures result in logarithmic increase in delay with operand size. The operating frequency of 70MHz for un-pipelined implementation clearly reveals that the speed performance of the arithmetic unit is comparable to commercial implementations.

Also a fast access rate 8KB, 300MHz cache was built in this work using 6T SRAM cells. To compare the performance of this cache with current implementations, the effect of scaling need to be considered. As technology scales, the cache speed increases proportionately due to scaling of bit line capacitance. Speed roughly doubles for scaling by a factor of 2. The speed of operation of the memory hardware, to the first order, would be roughly 5.4 times faster if the same design is implemented in 65nm process technology. This implies a dual cycle L1 access for a hardware operating at 3.2GHz which is comparable to speed performance of P4 processor. This concludes that

the hardware designed in this work is comparable in performance to commercial implementations.

The physical design of the circuit was carried out in Mentor Graphics environment using Autoplace and Route option. The hardware was implementation using a 0.35 micron process and requires an area of approximately $2mm^2$ when the multiplier is shared by the division hardware. Cache size would however determine the final die area if this hardware is used to design a complete microprocessor.

Further Improvements

This work can be extended in the following directions to further improve the overall performance of the hardware. Power reduction through gating schemes can be implemented to turn off idle units and reduce static power. Path delay balancing can be used to reduce transition activity in the sub circuits and in turn reduce delay as well as power consumption. The process of test case identification can be fully automated to make the approach more useful for larger designs.

APPENDICES

Appendix A

KOGGE ADDER – PG NOTATION

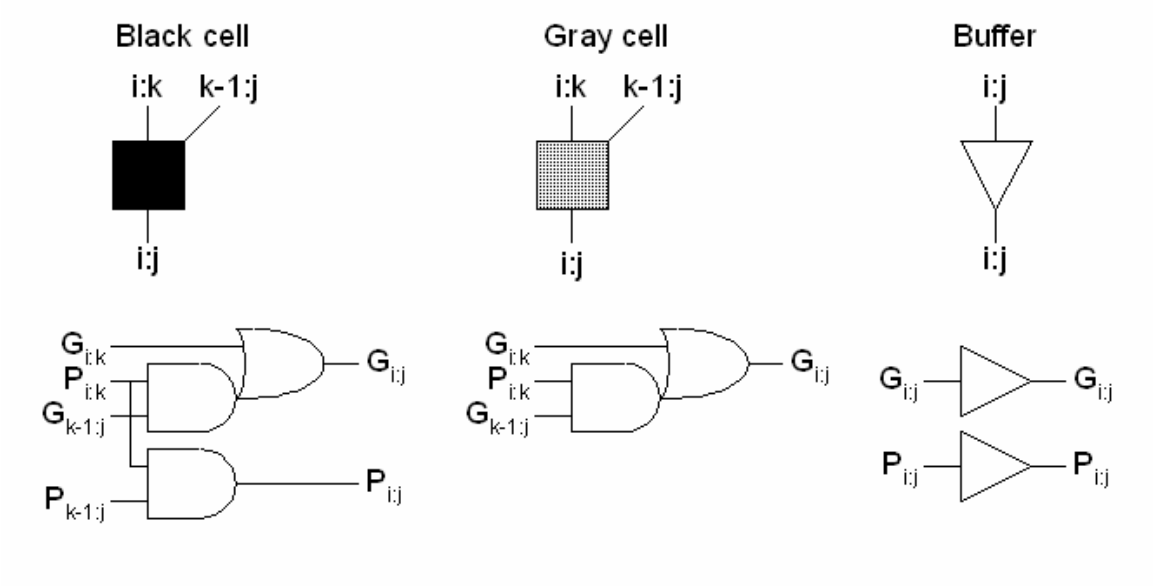
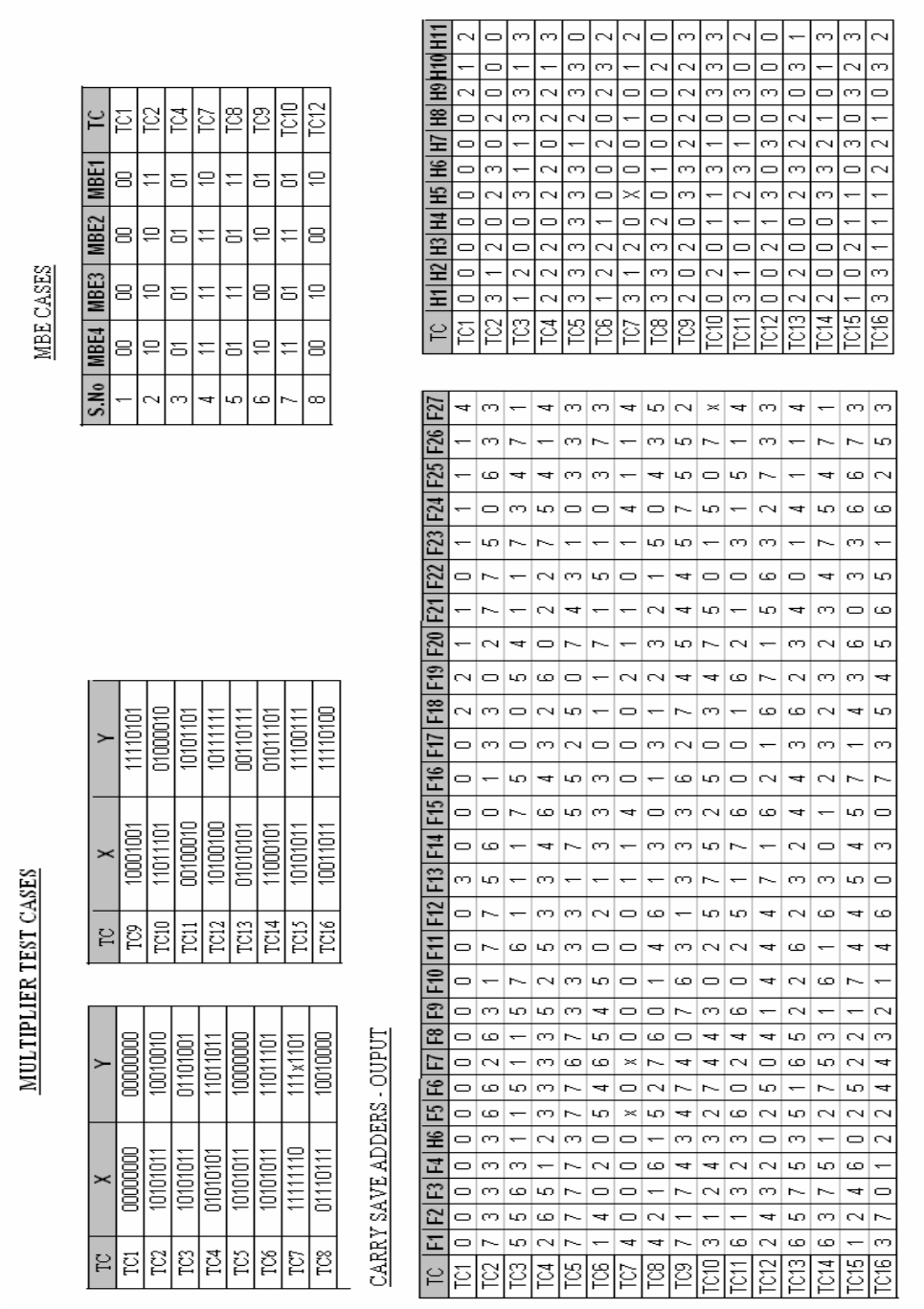


Figure A-1: PG Diagram notation for Kogge Stone Adder [52].

Appendix B

MULTIPLIER - TESTABILITY



MULTIPLIER TEST CASES

TC	X	Y
TC1	00000000	00000000
TC2	10101011	10010010
TC3	10101011	01101001
TC4	01010101	11011011
TC5	10101011	10000000
TC6	10101011	11011101
TC7	11111110	111x1101
TC8	01110111	10010000

TC	X	Y
TC9	10001001	11110101
TC10	11011101	01000010
TC11	00100010	10101101
TC12	10100100	10111111
TC13	01010101	00110111
TC14	11000101	01011101
TC15	10101011	11100111
TC16	10011011	11110100

MBE CASES

S.No	MBE4	MBE3	MBE2	MBE1	TC
1	00	00	00	00	TC1
2	10	10	10	11	TC2
3	01	01	01	01	TC4
4	11	11	11	10	TC7
5	01	11	01	11	TC8
6	10	00	10	01	TC9
7	11	01	11	01	TC10
8	00	10	00	10	TC12

CARRY SAVE ADDERS - OUPUT

TC	F1	F2	F3	F4	H6	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	F22	F23	F24	F25	F26	F27
TC1	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	2	2	1	1	0	1	1	1	1	4
TC2	7	3	3	3	3	6	6	2	6	3	1	7	7	5	6	0	1	3	3	0	2	7	7	5	0	6	3	3
TC3	5	5	6	3	1	1	5	1	1	5	7	6	1	1	1	7	5	0	0	5	4	1	1	7	3	4	7	1
TC4	2	6	5	1	2	3	3	3	3	5	2	5	3	3	4	6	4	3	2	6	0	2	2	7	5	4	1	4
TC5	7	7	7	7	3	7	7	6	7	3	3	3	3	1	7	5	5	2	5	0	7	4	3	1	0	3	3	3
TC6	1	4	0	2	0	5	4	6	5	4	5	0	2	1	3	3	3	0	1	1	7	1	5	1	0	3	7	3
TC7	4	0	0	0	0	x	0	x	0	0	0	0	0	1	1	4	0	0	0	2	1	1	0	1	4	1	1	4
TC8	4	2	1	6	1	5	2	7	6	0	1	4	6	1	3	0	1	3	1	2	3	2	1	5	0	4	3	5
TC9	7	1	7	4	3	4	7	4	0	7	6	3	1	3	3	3	6	2	7	4	5	4	4	5	7	5	5	2
TC10	3	1	2	4	3	2	7	4	4	3	0	2	5	7	5	2	5	0	3	4	7	5	0	1	5	0	7	x
TC11	6	1	3	2	3	6	0	2	4	6	0	2	5	1	7	6	0	0	1	6	2	1	0	3	1	5	1	4
TC12	2	4	3	2	0	2	5	0	4	1	4	4	4	7	1	6	2	1	6	7	1	5	6	3	2	7	3	3
TC13	6	5	7	5	3	5	1	6	5	2	2	6	2	3	2	4	4	3	6	2	3	4	0	1	4	1	1	4
TC14	6	3	7	5	1	2	7	5	3	1	6	1	6	3	0	1	2	3	2	3	2	3	4	7	5	4	7	1
TC15	1	2	4	6	0	2	5	2	2	1	7	4	4	5	4	5	7	1	4	3	6	0	3	3	6	6	7	3
TC16	3	7	0	1	2	4	4	3	2	1	4	6	0	3	0	7	3	5	4	5	6	5	1	6	2	5	3	

TC	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11
TC1	0	0	0	0	0	0	0	0	2	1	2
TC2	3	1	2	0	2	3	0	2	0	0	0
TC3	1	2	0	0	3	1	1	3	3	1	3
TC4	2	2	2	0	2	2	0	2	2	1	3
TC5	3	3	3	3	3	3	1	2	3	3	0
TC6	1	2	2	1	0	0	2	0	2	3	2
TC7	3	1	2	0	x	0	0	1	0	1	2
TC8	3	3	3	2	0	1	0	0	2	0	0
TC9	2	0	2	0	3	3	2	2	2	2	3
TC10	0	2	0	1	1	3	1	0	3	3	3
TC11	3	1	0	1	2	3	1	0	3	0	2
TC12	0	0	2	1	3	0	3	0	3	0	0
TC13	2	2	0	0	2	3	2	2	0	3	1
TC14	2	0	0	0	3	3	2	1	0	1	3
TC15	1	0	2	1	1	0	3	0	3	2	3
TC16	3	3	1	1	1	2	2	1	0	3	2

Figure B-1: Test Case Identification for 8 bit Wallace Multiplier.

SET 2	
Multiplicand	Multiplier
01101101	11100011
01011000	01110111
10000111	10010010
01101100	10011010
00101011	01100001
10101110	11000010
11010110	11111111
00101000	10010000
10101001	11010010
01101100	10100111
00010111	00110100
11000111	11111110
01000010	11111101
00010111	10010110
01000001	10000001
00010001	10000011

SET 3	
Multiplicand	Multiplier
01100111	11100001
01010100	01110111
10101100	11001101
01011011	10001010
10101011	11000101
00101100	01000011
10011000	11000110
11001110	11110001
10101001	10110001
11001010	11001100
10000011	11111011
11110111	11111001
01010111	10010001
01000010	01111101
01000001	10100100
01010110	10000010

SET 4	
Multiplicand	Multiplier
01101101	11100100
01010100	01110110
10000111	10010010
01101100	10001110
10001110	11001101
01101100	11000010
10011000	11101000
01010110	01111101
10101101	11000001
01101100	10100111
00011011	01000000
11110111	11111001
01010111	10010001
00010010	11110110
01000001	10100100
01010110	10000001

Figure B-2: Semi Automatic Test Case Identification for 8 bit Wallace Multiplier.

Appendix C

SRAM – CAPACITANCE CALCULATIONS

Bit Line Capacitance

Bit line capacitance is generally a major consideration in design of an SRAM array. Overlap capacitance and the drain/source bulk capacitance of the access transistors contribute to a major part of bit line capacitance.

$$C_{BIT\_LINE} = C_{OV} + C_{Diff} \quad \text{.....(1)}$$

Overlap Capacitance

Overlap capacitance of a cell is given by,

$$\frac{C_{BIT-ov}}{Cell} = W * C_{OV}$$

$$C_{OV} = C_{OL} + C_{fringe} \quad \text{.....(2)}$$

Where,

$$C_{OL} = CGDO + CGSO$$

$$C_{fringe} = \frac{2\xi_{OX}}{\pi} \ln\left(1 + \frac{T_{POLY}}{t_{OX}}\right)$$

CGDSO and CGSO are Gate drain and Gate source overlap capacitance per unit gate width and were obtained from model file.

$$CGDO = CGSO = 2.06e^{-10} Fm^{-1} \sim 0.2 fF\,\mu m^{-1}$$

$$C_{OL} = CGDO + CGSO = 0.4 fF.\mu m^{-1} \quad \text{.....(3)}$$

For current process, T_{POLY} is roughly 100 times $t_{ox,}$

$$\Rightarrow C_f = \frac{2\xi_{OX}}{\pi}\ln\left(1+\frac{T_{POLY}}{t_{OX}}\right) = \frac{2*11.9}{3.14}\ln(1+100) = 0.1fF \quad \text{.....(4)}$$

Using (2), (3) and (4), the value of Overlap Capacitance can be determined to be

$$\boxed{\frac{C_{BIT-OV}}{Cell} = 0.5*1 = 0.5f}$$

Diffusion Capacitance

$$\frac{C_{BIT-Diff}}{Cell} = C_{BP} + C_{SW}$$

$$\frac{C_{BIT-Diff}}{Cell} = (W * L_{diff}) * C_{Jn} + (W + 2 * L_{diff}) * C_{JSWn}$$

From Modle File,

$$C_J = 9.79e^{-4}F.m^{-1} = 0.979\,fF.\mu m^{-2}$$

$$C_{JSW} = 3.6e^{-4}F.m^{-1} = 0.36\,fF.\mu m^{-1}$$

$$\Rightarrow \frac{C_{BIT-Diff}}{Cell} = (1*1)*0.979 + (1+2*1)*0.36$$

$$\boxed{\frac{C_{BIT-Diff}}{Cell} = 2fF}$$

Total Bit line capacitance

$C_{BIT-OV} + C_{BIT-Diff} = 0.5+2 = 2.5$fF

Interconnect capacitance is not calculated in this work. For large arrays interconnect capacitance also adds a significant value to the bit line capacitance.

Word Line Time Constant

Word Line time constant is another important capacitance associated with the SRAM cell. For wide SRAM rows, the RC delay due to word line capacitance is an important parameter that needs to be considered to determine cell characteristics as well for decoder design.

$$\tau_{WORD} = R_{WORD} * C_{WORD}$$

Word Line Resistance:

$$R_{WORD} = \frac{\square}{Cell} * \frac{R}{\square} \quad \text{.....(5)}$$

For 8 bit word line,

$$R_{WORD} = 8\frac{\square}{Cell} * \frac{R}{\square}$$

Resistance per Square – $8\,\Omega$ (approx)

$$R_{WORD} = 8*16*8 = 1024\Omega$$

$$\boxed{R_{WORD} = 1K\Omega}$$

Word Line Capacitance:

In case of SRAM cell, one of the access transistors will be off all the time. The actual word line capacitance is given by,

$$C_{WORD} = C_{Access\_On} + C_{Access\_Off}$$

Where,

$$C_{Access-Off} = 2WC_{OV} + WL(C_{OX} \,||\, C_b)$$

$$C_{Access-ON} = 2WC_{OV} + \frac{2}{3}WLC_{OX}$$

To simplify word line capacitance calculation, full gate capacitance is assumed and the sum of capacitances seen looking into the two access transistors was calculated.

$$\frac{C_{WORD}}{Cell} = 2WLC_{OX} \quad \text{.....(6)}$$

Oxide capacitance is calculated as,

$$C_{ox} = \frac{8.854*10^{-14}*3.9}{7.8*10^{-9}*100} F/cm^2$$

$$= \frac{8.854*10^{-14}*3.9}{7.8*10^{-9}*100}*10^{-8} = 4.42\, F/\mu m^2$$

$$\frac{C_{WORD}}{Cell} = 2WLC_{OX} = 2*1*0.5*4.42*10^{-15}$$

$$\frac{C_{WORD}}{Cell} = 4.42\, fF$$

For 8 bit word line,

$$C_{WORD} = 8*\frac{C_{WORD}}{Cell} = 8*4.4\, fF$$

$$\boxed{C_{WORD} = 35.2\, fF}$$

$$\tau_{WORD} = R_{WORD} * C_{WORD}$$

$$= 1K\Omega * 35.42*10^{-15} = 3.5*10^{-11}\,\text{s}$$

RC of word line is a very low value for 8 bit SRAM array and hence can be excluded for sense amplifier design.

Oxide Capacitance Calculation

$$C_{ox} = \frac{\xi_o \xi_{OX}}{T_{OX}}$$

$$C_{ox} = \frac{8.854 * 10^{-14} * 3.9}{7.8 * 10^{-9} * 100} F/cm^2$$

$$= \frac{8.854 * 10^{-14} * 3.9}{7.8 * 10^{-9} * 100} * 10^{-8} F/\mu m^2$$

$$= 4.42 F/\mu m^2$$

Bit Line Capacitance of Pentium4 - L1 Cache

Cache Size = 8KB

Number of Bits / Line = 64 bits =8 Bytes

Cache Lines = Cache Size / Number of Bits = 8KB / 8B = 1K

=1024 Lines.

If SRAM cell designed in this thesis was used to build the L1 Cache of P4, the total bit line capacitance will be 1024 * 2.5fF = 2.56pF

Appendix D

SCHEMATICS

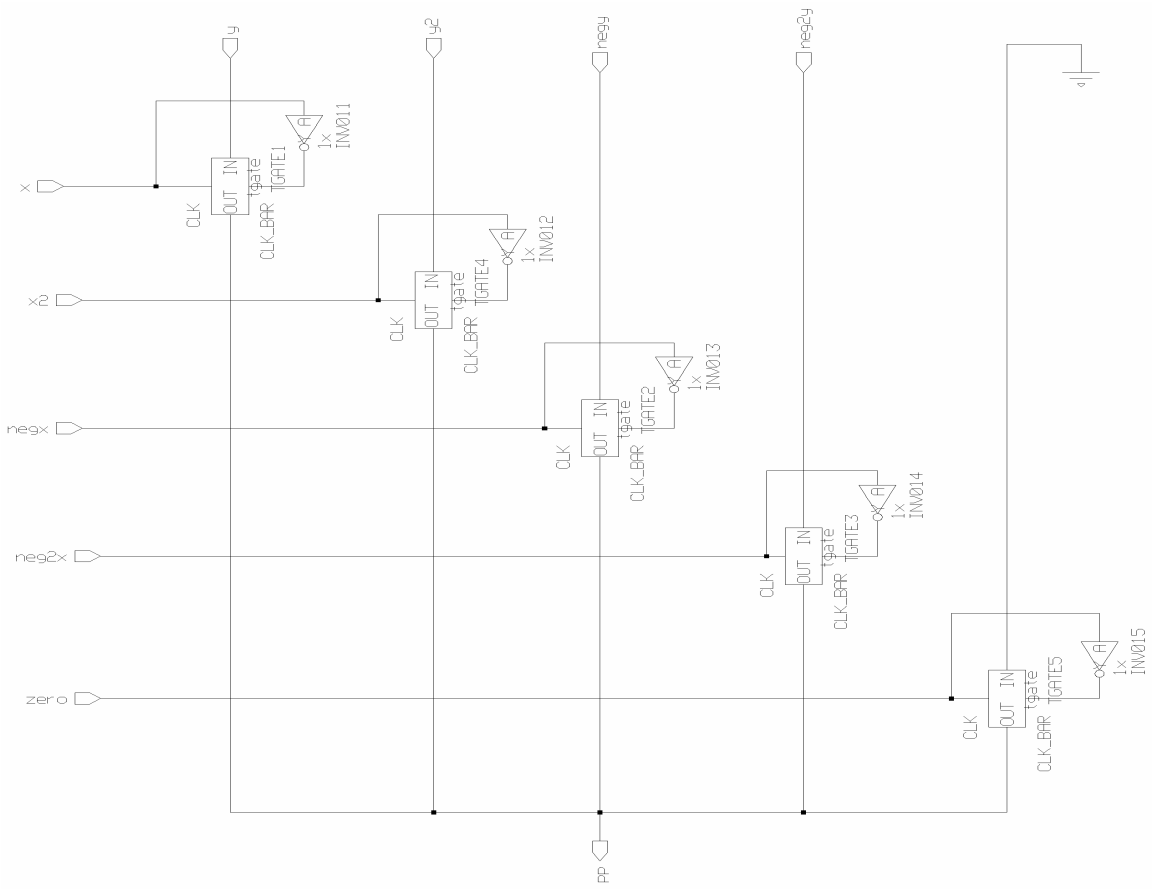


Figure D-1: T-Gate based Booth Selector.

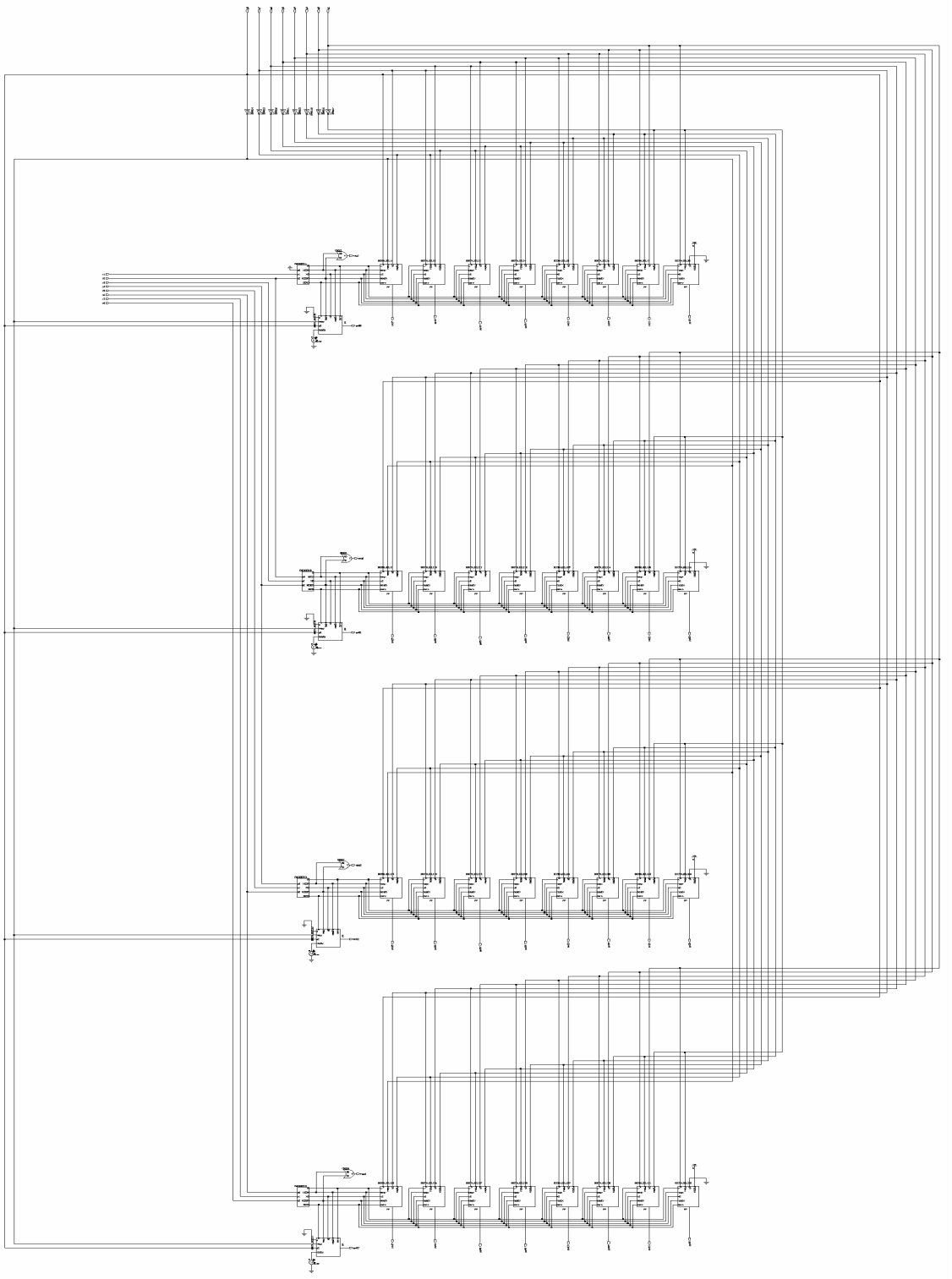


Figure D-2: 8 Bit Booth encoder and selector to generate partial product rows.

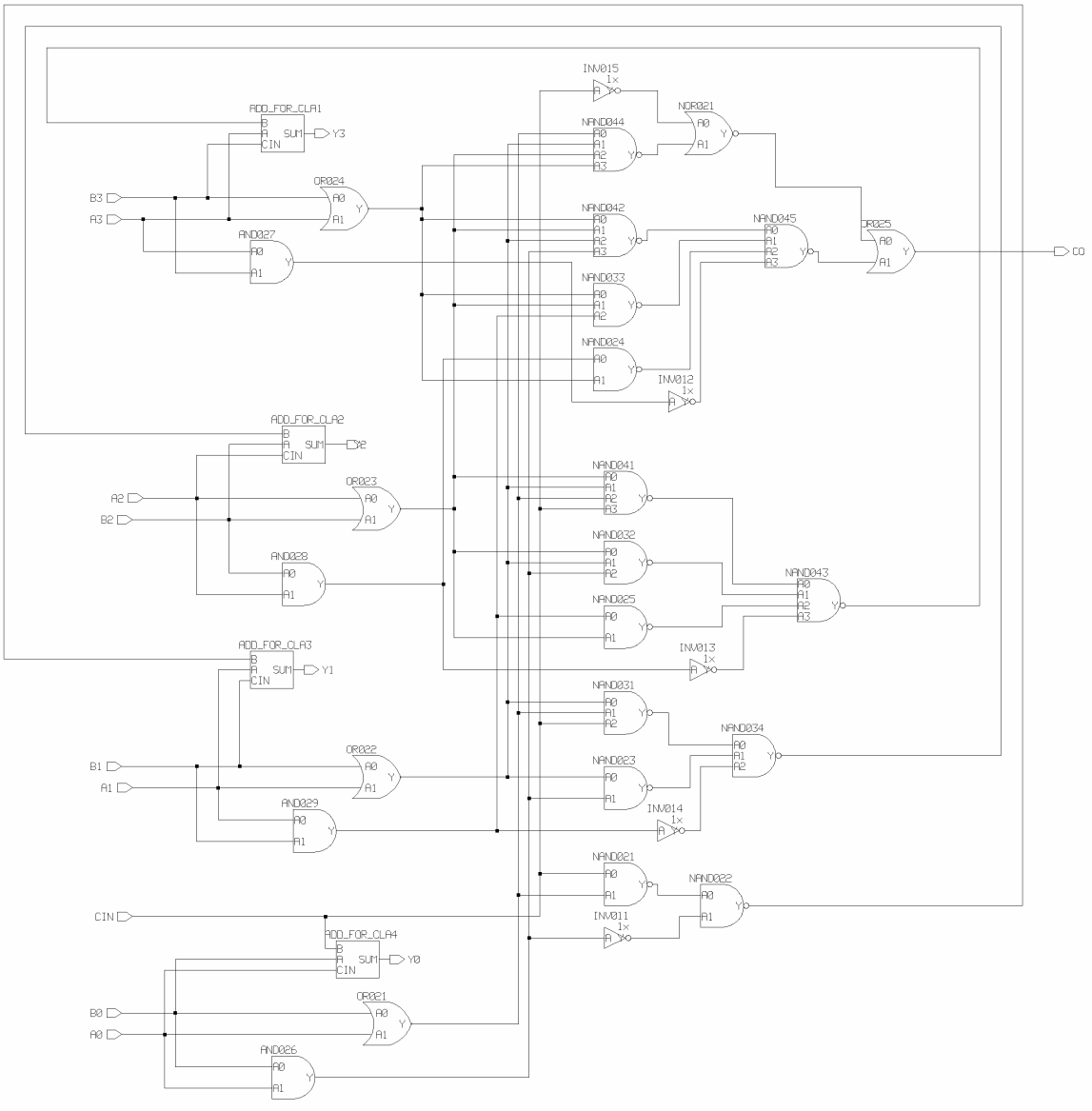


Figure D-3: 4 Bit Carry Look Ahead Adder.

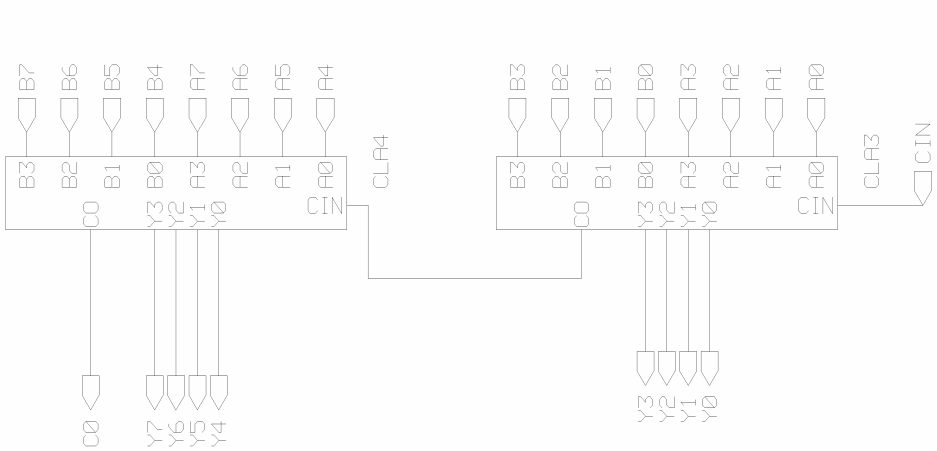


Figure D-4: 8 Bit Block Carry Look Ahead Adder using 4 bit CLA.

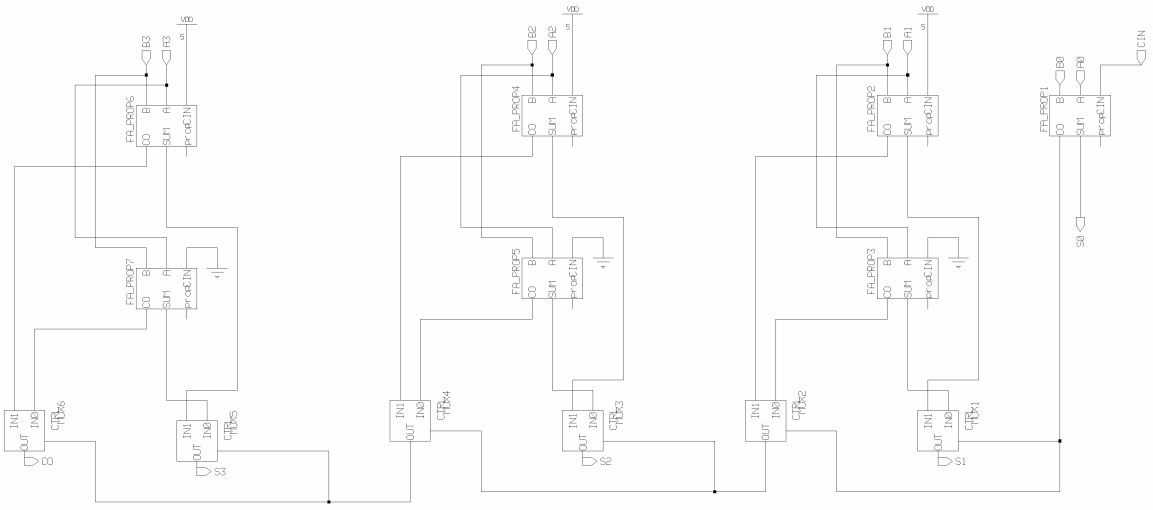


Figure D-5: 4 Bit Carry Select Adder.

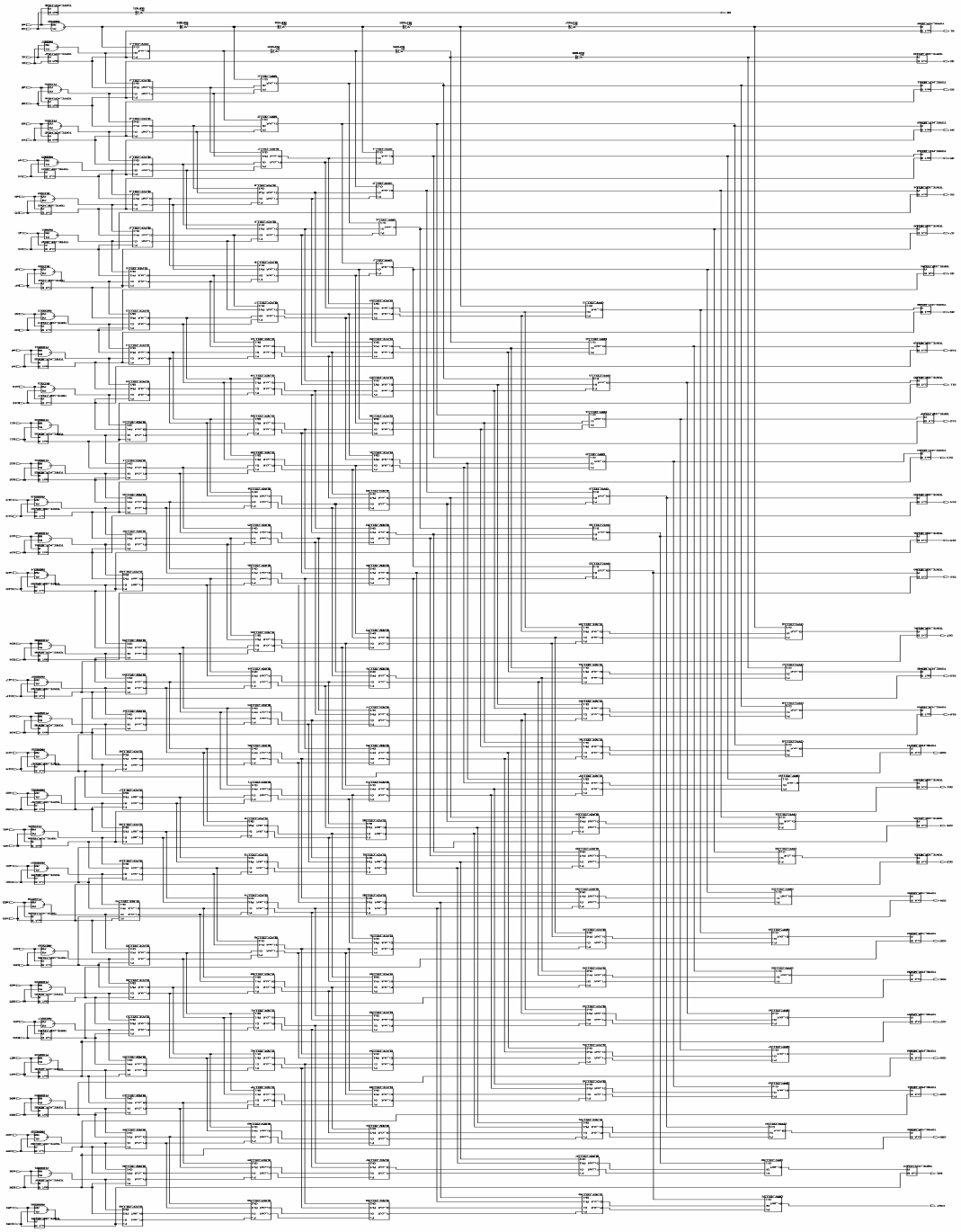


Figure D-6: 32 Bit Kogge Stone Adder Circuit.

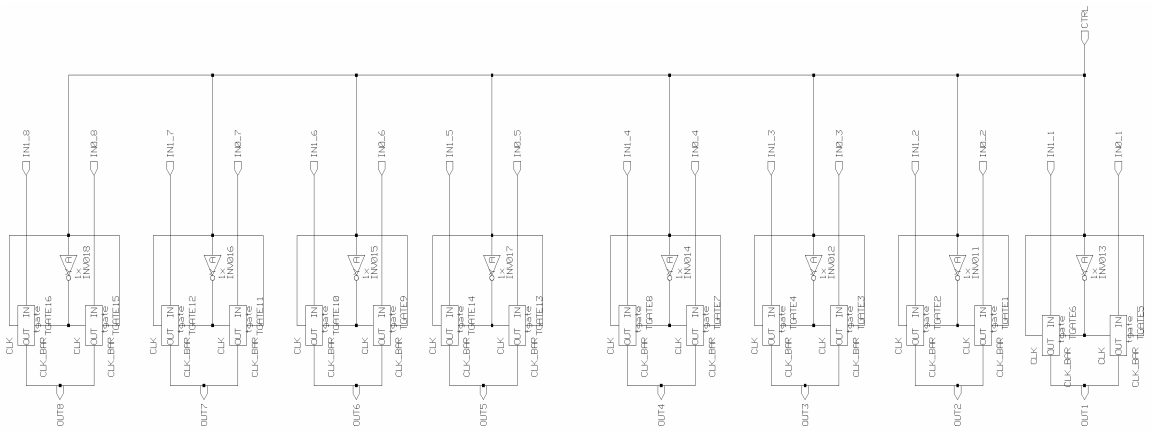


Figure D-7: T Gate based multiplexer.

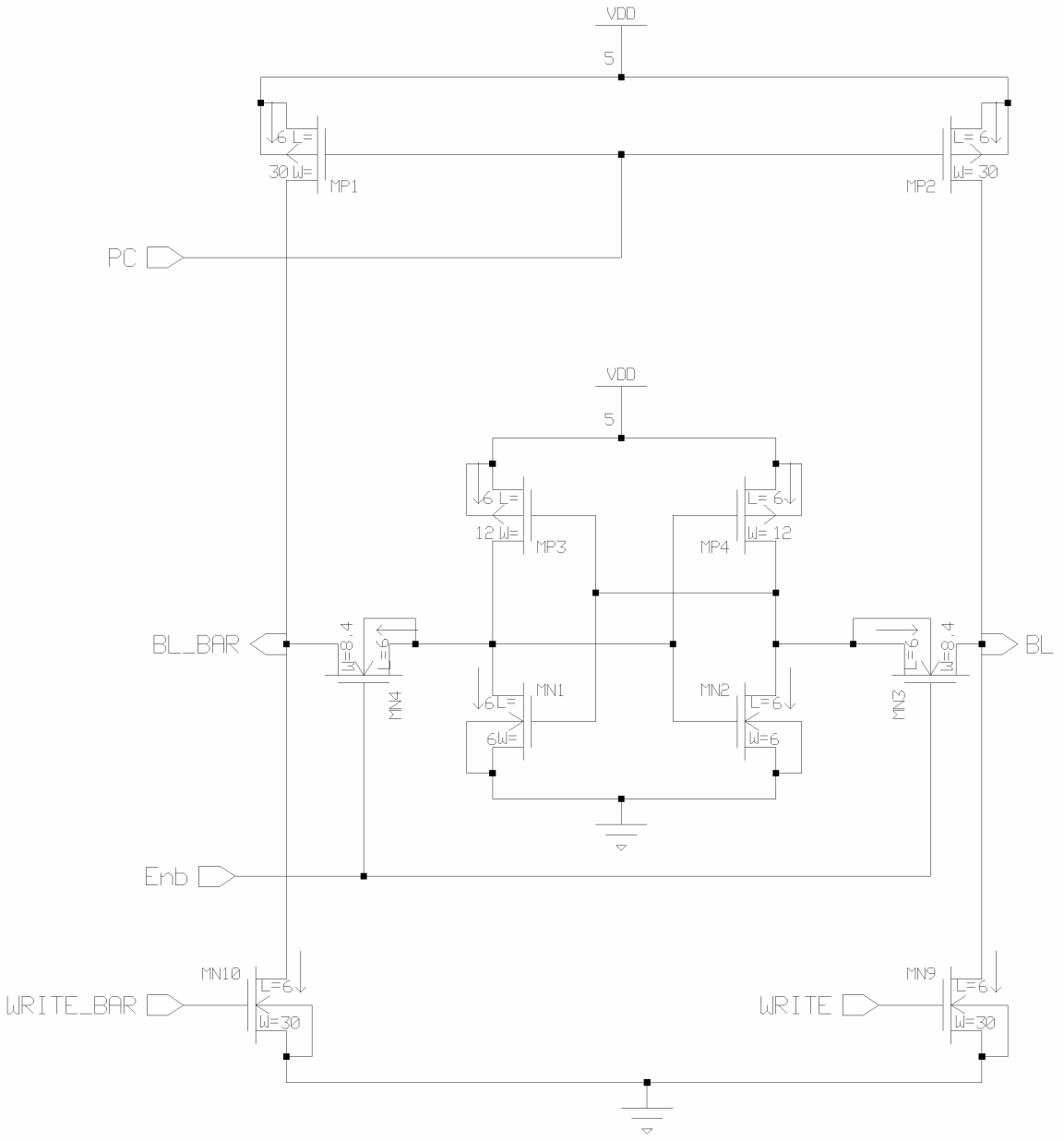


Figure D-8: SRAM Column without Read Circuitry.

Appendix E

LAYOUTS

SRAM CELL

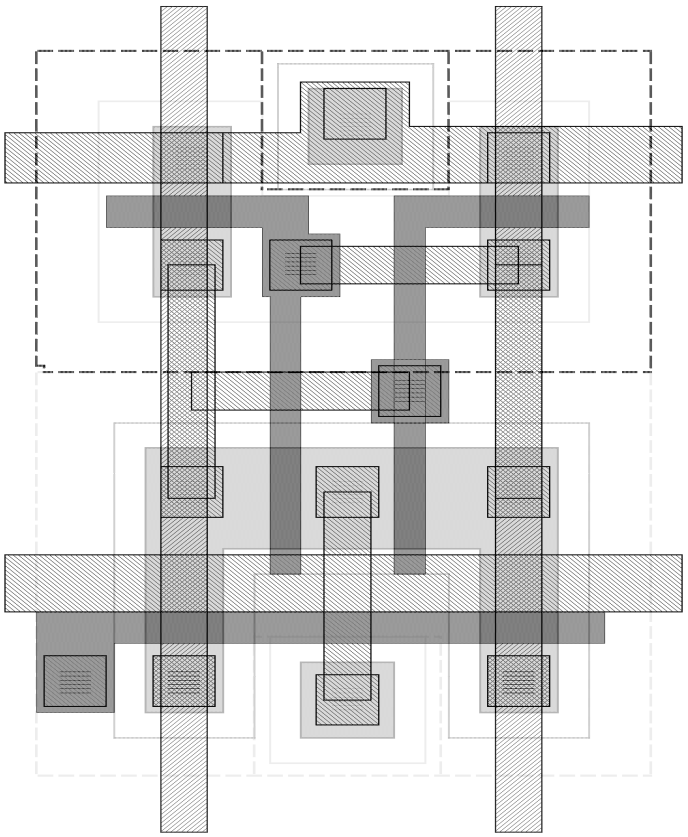


Figure E-1: Layout of SRAM Cell.

Sense Amplifier

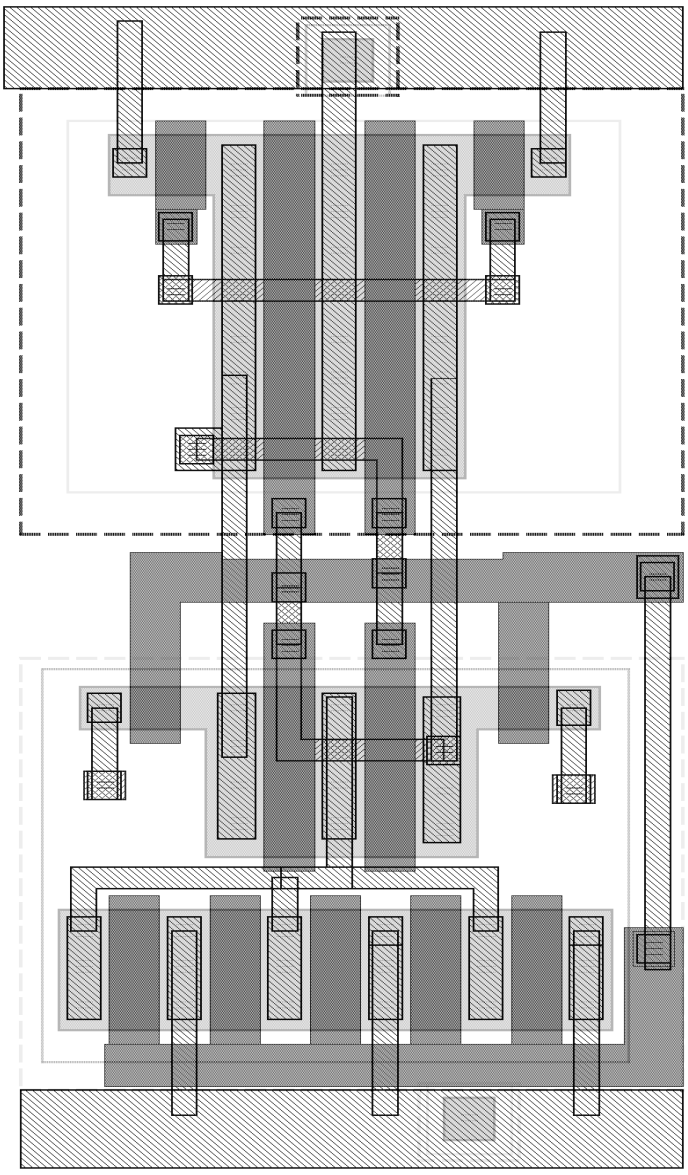


Figure E-2: Latch Type Sense Amplfiier.

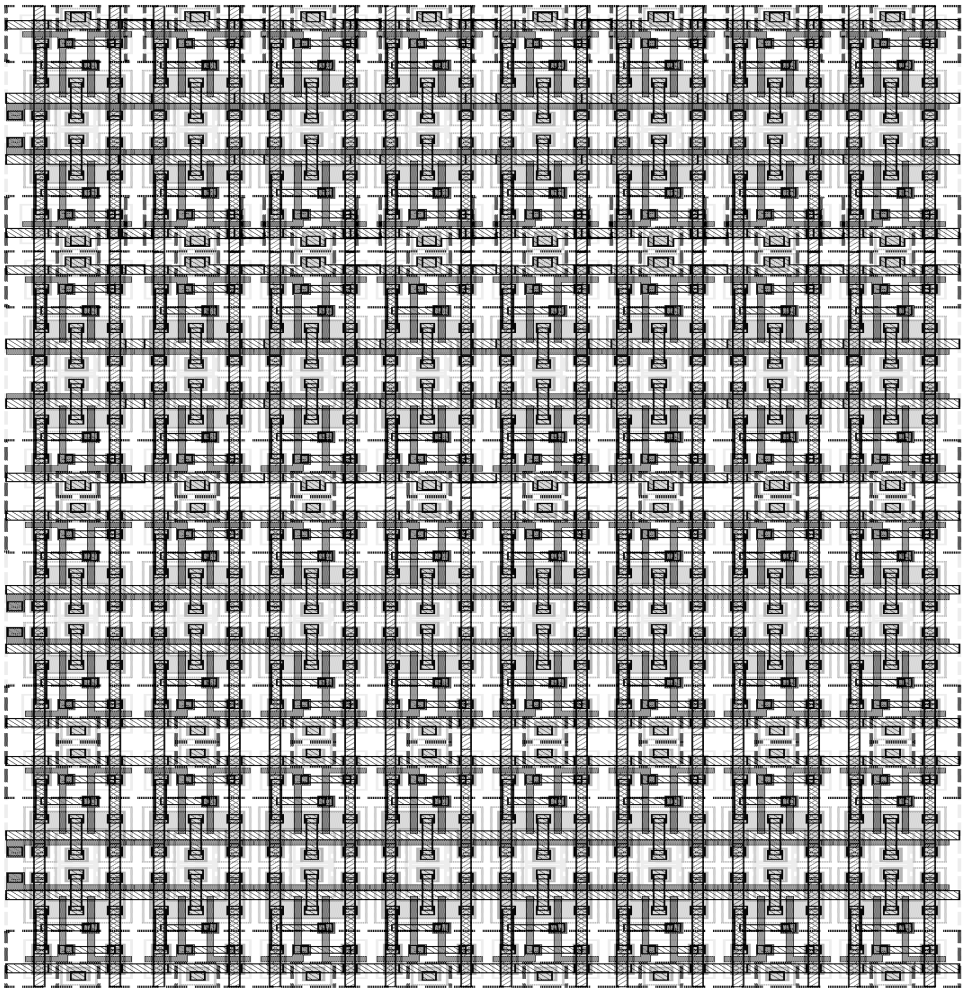


Figure E-3: SRAM cell arranged as 8x8 array.

REFERENCES

[1] Jessani, R.M.; Olson,C.H.,"Floating-point unit of the PowerPC 603e microprocessor" *IBM Journal of Research and Development*, v 40, n 5, Sep, 1996, p 559-566

[2] M. Lehman and N. Burla, "Skip techniques for high-speed carry propagation in binary arithmetic units", *IRE Trans. Electron. Computers*., vol. EC-10, pp. 691-698, 1961

[3] V. G. Oklobdzija and E. R. Barnes, "Some Optimal Schemes For ALU Implementation in VLSI Technology", *Proceedings of the 7th Symposium on Computer Arithmetic ARITH-7,* pp. 2-8, Reprinted in "Computer Arithmetic", E. E. Swartzlander, (editor), Vol. II, pp. 137-142, 1985

[4] M. J. Schulte, K. Chirca, J. Glossner, H. Wang, S. Mamidi, P. Balzola, S. Vassiliadis, "A Low-Power Carry Skip Adder with Fast Saturation", asap, pp. 269-279, *15th IEEE International Conference on Application-Specific Systems, Architectures and Processors* (ASAP'04), 2004

[5] J. Bedrij, "Carry-Select Adder", *IRE Transactions on Electronic Computers*, p. 340-344, 1962

[6] Weinberger and J. L. Smith, "A Logic for High-Speed Addition", *National Bureau of Standards*, Circ. 591, pp. 3-12, 1958.

[7] H. Ling, "High speed binary adder", *IBM J. Res. Develop,.* vol. 25, p. 156, May 1981

[8] S.Naffziger, "A Sub nano second 0.5µm 64 bit Adder Design" , *Digest of Technical Papers*: *1996 IEEE International Solid State Circuits Conference*, 1996, pp. 362-363.

[9] T. Kilburn, D. B. G. Edwards, and D. Aspinall, "Parallel Addition in Digital Computers: A New Fast Carry Circuit", Proceedings of IEE, Volume 106-B, p.464, September 1959.

[10] R. Hashemian, "A fast carry propagation technique for parallel adders", Circuits and Systems, 1990, Proceedings of the 33rd Midwest Symposium, pp 456 - 459 vol.1, Aug. 1990

[11] T.Kilburn, D.B.G. Edwards and D.Aspinall, "A Parallel Arithmetic Unit Using a Saturated Transitor Fast Carry Circuit", Proceedings of the IEE, Pt. B, vol. 107, pp. 573-584, November 1960.

[12] Kwon, E. Swartzlander, K. Nowka, "A fast hybrid carry-look ahead / carry-select adder design", *Proceedings of the 11th Great Lakes Symposium on VLSI*, pp 149-152, 2001

[13] R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders", *IEEE Transaction on Computers*, Vol. C-31, No. 3, p. 260-264, March, 1982.

[14] P.M Kogge and H.S Stone, "A Parallel Algorithm for the Efficient solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. 22, no. 8, pp.786-793, Aug 1973.

[15] R.E. Ladner and M.J Fischer, "Parallel Prefix Computation," *Journal of the Association for Computer Machinery*, vol. 27, no. 4, pp. 831-838, Oct. 1980.

[16] T. Han and D.A Carlson, "Fast Area Efficient VLSI Adders," *in Proceedings of the 8th IEEE Symposium on Computer Arithmetic*, 1987, pp. 49-56.

[17] J. Sklansky, "An Evaluation of Several Two Summand Binary Adders," EC 9, No.2, June 1960, pp. 213-226.

[18] D. Harris and J. Sutherland, "Logical Effort of Carry Propagate Adders," *In Proceedings of the 37th Asilomar conference on Signals, Systems and Computers*, CA, pp 873-878, November 2003.

[19] Gurkaynak, Frank, K. Leblebici, Yusuf, Chaouat, Laurent, McGuinness, Patrick, "Higher Radix Kogge Stone parallel prefix adder architectures," *Proceedings - IEEE International Symposium on Circuits and Systems*, v 5, 2000, p V-609-V-612

[20] Gnanasekaran, R ,"A Fast Serial-Parallel Binary Multiplier", *Computers, IEEE Transactions on*, Volume C-34, Issue 8, pp741 – 744, Aug 1985

[21] Shih-Lien Lu; Kenney, J.,"Design of most-significant-bit-first serial multiplier", *Electronics Letters*,vol. 31, Issue 14, pp.1133 – 1135, Jul 1995

[22] M Core Reference Manual, Motorla Inc., 1998.

[23] Zhijun Huang, Milos D. Ercegovac, "High-Performance Low-Power Left-to-Right Array Multiplier Design," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 272-283, Mar., 2005.

[24] P.C.H Meier, R.A Rutembar and L.R Carley, “Exploring Multiplier Architecture and Layout for Low Power,” *Proc. Of IEEE 1996 Custom Integrated Circuits Conference*, pp. 513-516, 1996

[25] Baugh, Charles R.; Wooley, Bruce A.,”Two’s complement Parallel Array Multiplication algorithm,”*IEEE Transactions on Computers*, v C-22, n 12, p 1045-1047, Dec, 1973.

[26] P.E Blankesnhip, ”Comments on A Two’s Complement Parallel Array Multiplication Algorithms,” *IEEE Transactions on Computers*, vol. C-23, p. 1327,1974

[27] Pezaris. S.D,”40- ns 17- bit by 17- bit array multiplier”, *IEEE Trans Computers*, v C-20, n 4, Apr, 1971, p 442-7

[28] C.S. Wallace, “A Suggestion for a Fast Multiplier,” *IEEE Trans. Computers*, vol. 13, no. 2, pp. 14-17, 1964.

[29] F. Elguibaly, “A Fast Parallel Multiplier-Accumulator Using the Modified Booth Algorithm,” *IEEE Trans. Circuits and Systems*, vol. 47, no. 9, pp. 902-908, 2000.

[30] L. Dadda, “Some Schemes for Parallel Multiplier,” Alta Frequenza, vol. 34, pp. 349-356, 1965.

[31] J. Fadavi-Ardekani, “M x N Booth Encoded Multiplier Generator Using Optimized Wallace Trees,” *IEEE Trans. Very Large Scale Integration*, vol. 1, no. 2, pp. 120-125, 1993.

[32] Farooqui, A.A.; Oklobdzija, V.G., "General data-path organization of a MAC unit for VLSI implementation of DSP processors," *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on* , vol.2, no.pp.260-263 vol.2, 31 May-3 Jun 1998

[33] Rectangular Wallace N. Itoh, Y. Naemura, H. Makino, Y. Nakase, T. Yoshihara, and Y. Horiba, “A 600-MHz 54x54-bit Multiplier with Rectangular-Styled Wallace Tree,” *IEEE J. Solid-State Circuits* 36, No. 2, 249–257,February 2001.

[34] E. Costa, J. Monteiro, and S. Bampi. A New Architecture for 2's Complement Gray Encoded Array Multiplier.” *In Proceedings of the XV Symp. on Integrated Circuits and Systems Design*, pages 14--19, September 2002.

[35] O.L. MacSorley, “High Speed Arithmetic in Binary Computers,” *Proc. IRE*, vol. 49, pp. 67-91, 1961

[36] Jeff Scott, Lea Hwang Lee, Ann Chin, John Arends, bill Moyer, “Designing the M.CoreTM M3 CPU Architecture,” *IEEE International Conference on Computer Design*, Austin, Texas, Oct 10-13,1999.

[37] Alexander Goldovsky, Bimal Patel, Michael Schulte, Ravi Kolagotla, Hosahalli Srinivas, and Geoffrey Burns. Design and implementation of a 16 by 16 low-power two's complement multiplier. *In IEEE International Symposium on Circuits and Systems*, volume 5, pages 345--348, May 2000.

[38] O.L.MacSorley, "High-Speed Arithmetic in Binary Computers," *IRE Proceedings* vol.49, pp.67--91, 1961.

[39] McFearin, L.D.; Matula, D.W,Seidel, P.M, “Binary multiplication radix-32 and radix-256”,*15th IEEE Symposium on Computer Arithmetic*, p 23-32, 2001.

[40] Y.Wang, Y.Jiang and E. Sha, On Area-Efficient Low Power Array Multipliers. *In the 8th IEEE International conference on electronics*, Circuits and Systems, pages 505-508,2001.

[41] D. Bakalis, D. Nikolos, On low power BIST for carry save array multipliers, *in: Proceedings of the 5th International On-Line Testing Workshop*, 1999, pp. 86-90.

[42] S. E. McQuillan, J.V. McCanny, R. Hamill, “New Algorithms and VLSI Architectures for SRT Division and Square Root,” *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, 1993, pp. 80-86.

[43] D. Harris, S. Oberman and M. Horowitz, “SRT Division Architectures and Implementations. *Proc. 13th Symp. Computer Arithmetic*, pp. 18-24, July 1997

[44] T. Coe, P.T.P. Tang, "It Takes Six Ones To Reach a Flaw," arith, p. 140, *12th IEEE Symposium on Computer Arithmetic* (ARITH-12 '95), 1995

[45] Wang, L. Schulte, J. Michael,” Decimal floating-point division using Newton-Raphson iteration”, Proceedings - *15th IEEE International Conference on Applications-Specific Systems, Architectures and Processors*, 2004, p 84-97

[46] P. Montuschi, L. Ciminiera, A. Giustina, "Division unit with Newton-Raphson approximation and digit-by-digit refinement of the quotient", *IEE Proceedings - Computers and Digital Techniques* -- November 1994 -- Volume 141, Issue 6, p. 317-324

[47] R.E. Goldschmidt, "Applications of Division by Convergence," MS thesis, Dept. of Electrical Eng., Massachusetts Inst. of Technology, Cambridge, Mass., June 1964

[48] Peter Markstein, Software Division and Square Root Using Goldschmidt's Algorithms, *Proceedings of the 6th Conference on Real Numbers and Computers*, pp. 146-157, 2004.

OTHER REFERENCES

[49] Digital Integrated Circuits: A Design Perspective, 2nd Ed, Jan M. Rabaey, Anantha Chandrankasan, Borivoje Nikolic, Prentice Hall, 2005.

[50] Microelectronics: An Integrated Approach, Roger T. Howe and Charles G. Sodini, Prentice Hall, 1997

[51] CMOS Digital Integrated Circuits, Analysis and Design, 3rd Ed, Sung-Mo Kang, Yusuf Leblebici, TataMcGraw-Hill 2003.

[52] Adders: Lecture Notes, David Harris, Harvey Mudd College.