IMPLEMENTATION OF A HARDWARE ALGORITHM FOR INTEGER DIVISION

By

RACHANA ERRA

Bachelor of Technology in Electronics and
Communication Engineering
Kakatiya Institute of Technology and Science
Kakatiya University
Warangal, Telangana, India
2017

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
August, 2019

IMPLEMENTATION OF A HARDWARE ALGORITHM FOR INTEGER DIVISION

Thesis Approved:

Dr. James E. Stine, Jr.

Thesis Adviser

Dr. Sabit Ekin

Dr. Weili Zhang

ACKNOWLEDGMENTS

I am indebted to my advisor, Professor Dr. James E. Stine, Jr. of the School of Electrical and Computer Engineering at Oklahoma State University, Stillwater, USA. The door to Dr. Stine's office was always open whenever I ran into a trouble spot or had a question about my research or writing. I am humbled to place on record my deep sense of reverence and gratitude for all the encouragement, support and for all the help he has had for me during my career at the America's Brightest Orange, the Oklahoma State University, Stillwater, USA. Words surely fail to express my thanks to him. May his tribe increase!

Besides my thesis Advisor, I would like to thank Professor Dr. Sabit Ekin of the School of Electrical and Computer Engineering at Oklahoma State University, Stillwater, USA for serving as my committee member. He has helped me with all his value added suggestions in shaping the thesis to make it appropriate in every aspect of its constructive outcome indeed. I really appreciate his moral boost which propels the work to run at speed with ease.

I take this opportunity to place my sincere thanks on record to Professor Dr. Weili Zhang of the School of Electrical and Computer Engineering at Oklahoma State University, Stillwater, USA for serving as my committee member and who is one among the distinguished professors for helping me with for the useful comments, remarks and engagement through the learning process of this masters thesis which, in fact, prompted me to come up with zeal and vigour in the completion of this work.

Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

My sincere thanks go to Dr. Keith Teague of School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, USA whose professional experience and academic discipline are of great aid to me during the course of my thesis work and I thank him for the soft corner he holds for me in responding to my queries when in need.

Much lauded and sought after Library of Oklahoma State University, Stillwater is a repository of rich collection of editions in print and electronic means which meant a great deal for me to depend on while exploring the saga of potential for excellence in my pursuit of my course work and the thesis work. I truly owe a lot to the Library.

Dreams come true with the blessings of virtuous parents like those of mine Dr. E Ram Bhaskar Raju and Sreelakshmi and the best wishes of my only tender sibling Namratha. No words to thank my Granny who always wished to see me study overseas and come up in flying colours!

Finally, I thank the great Oklahoma State University, Stillwater, USA for accommodating me with the matchless experience of education while studying in the School of Electrical and Computer Engineering in my Masters in word and spirit. Long Live OSU.

Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: RACHANA ERRA

Date of Degree: August, 2019

Title of Study: IMPLEMENTATION OF A HARDWARE ALGORITHM FOR INTEGER DIVISION

Major Field: ELECTRICAL ENGINEERING

Abstract:

A hardware algorithm for integer division proposed by Naofumi Takagi, Shunsuke Kadowaki and Kazuyoshi Takagi is implemented using the hardware-descriptive language called Verilog. The hardware algorithm is based on digit-recurrence nonrestoring division. Each partial remainder is represented as an SD2 integer as a pair of its sign and absolute value. Quotient digit is obtained from the sign of every partial remainder with the most significant digit first. Quotient is obtained as an ordinary binary number and remainder as an SD2 integer. Remainder is converted from SD2 to binary using the concept of a carry lookahead adder. Combinational design for 4-bit and 8-bit integers that uses radix-2 signed-digit representation of numbers is implemented. It is simulated and tested using Mentor Graphics Corporation (MGC)® ModelSim™. RTL synthesis is performed and parameters like area, total cells, timing and power are calculated.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Fundamental arithmetic operations such as addition, subtraction, multiplication and division play a vital role in computers, digital signal processors and application specific integrated circuits. Division in particular is used in many scientific and commercial programs, however, many implementations fail to be implemented optimally [1]. Many algorithms have been proposed to implement specialized arithmetic circuits to make certain specific problems be solved quickly. But most of them have been hampered due to large chip area and clocking complexities. Hardware realization of the algorithm involves challenges like meeting the minimal clock period, rapidly normalising given data and controlling the operation of sequence of different modules [2] such that no racing problem occurs [3]. To develop a hardware algorithm suitable for VLSI implementation, the circuit should have a regular structure and also perform high-speed computation [2]. This thesis concentrates on implementing a division algorithm that addresses the above challenges.

Division in general, is often implemented with repeated subtraction. The quotient is subsequently determined based on the number of subtractions. Integer division of two numbers yields an integer quotient and an integer remainder [4]. Division algorithms for hardware implementation can be classified into digit-recurrence, functional iteration and table-based methods [5]. Digit recurrence is further classified into restoring and non-restoring division algorithms [3].

The algorithm implemented in this thesis is based on digit-recurrence non-restoring division. Combinational implementation of this algorithm yields a regular structure [4]. Moreover, combinational circuits do not rely on clocks and do not consume more power like that of sequential circuits. Also, the number representation plays an important role in the computation speed of the algorithm. As division circuits are built on subtractors, their execution time is very high. So, to simplify subtraction, Signed-Digit(SD) representation of numbers is used. It helps perform addition and subtraction in a constant time independent of the word length of the operands [2].

This property of signed-digit numbers helps in achieving high-speed computation. This thesis uses a combinational circuit with radix-2 signed-digit (SD2) representation of numbers to implement a digit-recurrence non-restoring division algorithm. Every partial remainder in the process is represented as a radix-2 signed-digit number [4]. A quotient bit is obtained in every iteration based on the sign of the partial remainder. The remainder is initially obtained as an SD2 integer which needs to be converted to ordinary binary number. Unlike earlier division algorithms, this does not require the normalisation step. Hence an area-consuming leading one(or zero) barrel shifter is not required [4].

The algorithm presented in this thesis has been implemented for 4-bit and 8-bit numbers in Verilog. It is simulated and tested using Mentor Graphics Corporation (MGC)® ModelSim™ for all four variants possible in integer division i.e. positive/positive, negative/positive, positive/negative and negative/negative. RTL synthesis for both 4-bit and 8-bit implementations is performed. Area, total cells required, computation time and power are calculated.

This thesis is organised as follows. Chapter 2 presents an outline of the number

representation used in this thesis. Chapter 3 presents division algorithms and implementation. Chapter 4 presents the results. Chapter 5 presents the conclusion and future work.

CHAPTER II

NUMBER REPRESENTATION

Number representation plays an important role in deciding the speed of a hardware algorithm. The algorithm implemented in this thesis uses the radix-2 signed-digit representation.

2.1 Signed-Digit Representation

It is a numeral system that handles more than one representation for a number. It uses a digit set $D_{<r,a>} = \{-a, \ldots, -1, 0, 1, \ldots, a\}$, where r is the radix and a is the largest digit of the set [6]. A radix-2 representation uses the digit set {-1, 0, 1} with an n-digit redundant binary number being A=$[a_{n-1}a_{n-2}\ldots a_0]$ bearing a value $\sum_{i=0}^{n-1} 2^i \cdot a_i$ [4]. It is a number system in which each digit has its own sign regardless of the sign of the number represented. For example, number 2 can be represented as [0010], [00$\overline{1}$0] and [1$\overline{1}\overline{1}$0] where $\overline{1}$ represents -1. Each signed-digit is represented as a 2-bit binary number in Verilog with MSB indicating the sign and LSB indicating the value.

This number system is more advantageous for arithmetic operations than for logical operations. It helps perform carry-free addition and borrow-free subtraction. Adding or subtracting arbitrary long numbers is performed in a constant time [7]. Also, negative numbers are written in the same way as positive numbers without 2's-complement representation which also makes negation simple [7]. In the case of division, signed-digit representation helps simplify the choices for quotient by allow-

ing more digits to be used per iteration.

2.1.1 Borrow Propagation Free Subtraction

Signed-Digit numbers support carry/borrow free addition/subtraction. In the hardware algorithm that is implemented, subtraction of an ordinary binary integer from an SD2 integer results in an SD2 integer [4]. All of this happens in two different steps. The first step produces a borrow bit and an intermediate bit and they belong to the digit set {0, 1}. The final result is produced in the second step that belongs to the digit set {-1, 0, 1}. The subtraction takes place as shown in the Tables 2.1 and 2.2

		minuend digit		
		-1	0	1
subtrahend bit	0	1,1	0,0	0,1
	1	1,0	1,1	0,0

b_{i+1},e_i

Table 2.1: First Stage of Subtraction

		e_i	
		0	1
b_i	0	0	1
	1	-1	0

Table 2.2: Second Stage of Subtraction

2.1.2 Radix-2 Signed-Digit to Binary Conversion

An n-digit signed number taken as A=$[a_{n-1}a_{n-2}\ldots a_0]$ where a_i belongs to the digit set {-1, 0, 1} can be converted into an equivalent binary number in the following way [2]:

equivalent binary number = A^- - A^+

where A^- is the n-bit unsigned number obtained from the negative digits of A having

positives and zeros as 0 and negatives as 1 and A^+ is the n-bit unsigned number obtained from the positive digits of A having negatives and zeros as 0 and positives as 1.

An SD2 integer [$1\bar{1}\ \bar{1}0$] can be converted into equivalent binary as [1000]-[0110] = [0010]. But for an unsigned SD2 integer, the equivalent binary is the number itself. This process has been implemented using a carry-look-ahead adder in this thesis.

2.2 Sign and Absolute Value

Every partial remainder produced in the process of division is expressed as an SD2 integer. It is expressed as a pair of its sign and absolute value. The sign of the partial remainder is used to calculate the quotient bit of that iteration and the absolute value is used to calculate the next partial remainder.

2.2.1 Sign

The sign of the most significant non-zero digit is the sign of an SD2 integer [4]. It is calculated as shown below.

for $i = n - 1$ *downto* 0 **do** ▷ n bits of the partial remainder
  if (sign of previous bit = 0 and the value of the current bit = -1) **then**
    sign of partial remainder till the particular bit = -1
  else if (sign of previous bit = 0 and the value of current bit = 1) **then**
    sign of partial remainder till this particular bit = 1
  else
    sign of partial remainder till the particular bit = sign of previous bit
  end if
end for

2.2.2 Absolute Value

For a negative number, its negation gives the absolute value which is also an SD2 integer. In other cases when the number is positive or zero, it is the number itself. It is calculated as shown below.

if (sign of partial remainder bit = -1 and value of partial remainder bit $\neq 0$) **then**

absolute value = negation of the partial remainder bit

else

absolute value = partial remainder bit itself

end if

CHAPTER III

DIVISION ALGORITHM AND ITS IMPLEMENTATION

Division is about finding how many times a number can be subtracted from the other without yielding a negative number. The divisor is subtracted from the dividend as long as a negative remainder is not obtained. If obtained, the divisor is added back as a correction factor. The quotient is calculated as #subtraction $-$ 1 [8]. For dividend x, divisor y, quotient q and remainder r $x = q \cdot y + r$ [5]. The example below shows 1347 divided by 2.

$$
\begin{array}{r}
673 \\
2\overline{)1347} \\
\underline{12} \\
14 \\
\underline{14} \\
07 \\
\underline{06} \\
1
\end{array}
$$

This is called digit recurrence method of division. It is further classified as restoring and non-restoring. Digit recurrence algorithms generate a quotient bit in a step-by-step process with the most significant digit first. Quotient bit is inferred from the sign of the difference of dividend and divisor. Divided in each iteration is the partial remainder that is obtained. Divisor in each iteration is quotient times times the actual divisor multiplied by its place value. A positive dividend and a positive divisor result in a positive quotient and a positive remainder. A negative dividend and a positive divisor result in a negative quotient and a negative remainder. A positive dividend and negative divisor result in a negative quotient and a positive remainder. A negative dividend and a negative divisor result in a positive quotient and a negative remainder.

3.1 Restoring Division

The conventional restoring division is performed as a series of shifts and subtractions [8]. Quotient is determined from the sign of the partial remainder. If the partial remainder is negative, divisor is added back which is called restoring. The algorithm is as follows[9]:

1: Initialize the registers i.e. Q=dividend, M=divisor, A=0, N= number of bits in dividend.

2: Shift contents of A and Q by one unit.

3: Perform A=A-M.

4: If the result is positive, LSB of Q is set to 1. Else, LSB of Q is set to 0 and the contents of register A are restored.

5: Decrement the value of N by one.

6: Repeat steps 2 through 5 until N=0.

7: Q = quotient and A = remainder

The following example is shown for two input operands Divisor=0010, Dividend=0101

N	M	A	Q	Comments
4	0010	0000	0101	initialize registers
4	0010	0000	101	shift left AQ
4	0010	-0010	101	subtract M from A
4	0010	1110	101	A is -ve, restore A in next step,Q[0]=0
3	0010	0000	1010	decrement n, restore and Q[0]
3	0010	0001	010	shift left AQ
3	0010	-0010	010	subtract M
3	0010	1111	010	A is -ve, restore and Q[0]=0 in next step
2	0010	0001	0100	decrement n, restore and Q[0]
2	0010	0010	100	shift left AQ

2	0010	-0010	100	subtract M
2	0010	0000	100	A is +ve, Q[0]=1 in next step
1	0010	0000	1001	decrement n, restore and Q[0]
1	0010	0001	001	shift AQ
1	0010	-0010	001	subtract M
1	0010	1111	001	A is -ve, restore A and Q[0]=0 in next step
0	0010	0001	0010	A=remainder, Q=quotient

3.2 Non-restoring Division

It is an improved division algorithm where restoring does not take place. The operation after shifting is either addition or subtraction depending on the partial remainder [8].The algorithm is as follows [10]:

1: Initialize the registers i.e. Q=dividend, M=divisor, A=0, N= number of bits in the dividend.

2: Check the sign of contents in register A.

3: If positive, shift left AQ and perform A=A-M. If negative, shift left AQ and perform A=A+M.

4: Check the sign of contents of register A.

5: If positive, LSB of Q is set to 1. If negative, LSB of Q is set to 0.

6: Decrement the value of N by one.

7: Repeat steps 2 through 6 until N = 0.

8: Check the sign of contents of A. If negative, perform A=A+M.

9: Q = quotient and A = remainder

The following example is shown for two input operands Divisor=0010, Dividend=0101:

N	M	A	Q	Comments
4	0010	0000	0101	A +ve
4	0010	0000	101	shift AQ
4	0010	-0010	101	A=A-M
3	0010	1110	1010	A -ve, Q[0] = 0, decrement N
3	0010	1101	010	shift AQ
3	0010	+0010	010	A = A+M
2	0010	1111	0100	A -ve, Q[0] = 0, decrement N
2	0010	1110	100	Shift AQ
2	0010	+0010	100	A = A+M
1	0010	0000	1001	A +ve, Q[0]= 1, decrement N,
1	0010	0001	001	A +ve, Shift AQ
1	0010	-0010	001	A = A - M
0	0010	1111	0010	A -ve, Q[0]= 0, decrement N
0	0010	+0010	0010	A -ve, A = A + M
0	0010	0001	0010	A = remainder, Q = quotient

Non-restoring method is quicker and better than the restoring method because it takes only one decision per quotient and addition/subtraction per quotient bit [11]. Also, the sign of the partial remainder is allowed to be either positive/negative in non-restoring. It can be expressed in terms of signed digit representation which facilitates hardware implementation.

3.3 Radix-2 Signed-Digit Non-restoring Integer Division

The radix-2 signed-digit non-restoring division algorithm on which the original hardware algorithm that is implemented is as shown below [4].

Divisor: Y Dividend: X

Let $D = |Y|$ and $R_n = X$

for $j \leftarrow n-1$ downto 0 **do**

if $R_{j+1}(partialremainder) = 0$ **then**

$Q(quotient) = [q_{n-1}q_{n-2}\ldots q_{j+1}0\ldots 0]$

$R = 0$

if $Y(divisor) < 0$ **then**

$Z(finalquotient) = -Q$

else

$Z = Q$

end if

end if

if $(R_{j+1}(partialremainder) < 0)$ **then**

$q_j = -1$

else

$q_j = 1$

end if

$R_j = R_{j+1} - q_j \cdot 2^j \cdot D$ ▷ calculating next partial remainder

end for

$Q = [q_{n-1}q_{n-2}\ldots q_0]$

if $(X(divisor) > 0$ and R_0(final partial remainder)$< 0)$ **then**

R(remainder) $= R_0 + D$

$Q = Q - 1$

else if $(X < 0$ and $R_0 > 0)$ **then**

R(remainder) $= R_0 - D$

$Q = Q + 1$

else

$R = R_0$

end if

3.4 Hardware Algorithm

It is the hardware algorithm implemented in this thesis. It is based on digit-recurrence non-restoring division algorithm [4]. It uses radix-2 signed-digit representation to represent partial remainders. Each partial remainder is represented as a pair of its sign and absolute value where the absolute value is an SD2 integer. Absolute value of the partial remainder can be calculated simultaneously with sign detection. It can also be started even before the sign detection of its preceding partial remainder. Quotient digit in each iteration is obtained from the sign of the corresponding partial remainder [4]. It is obtained as an ordinary binary number. Remainder is obtained as an SD2 integer. Conversion from SD2 to binary is required. Quotient adjustment and remainder selection are done based on sign of the divisor and dividend.

To make the radix-2 signed-digit non-restoring algorithm suitable for hardware implementation, a series of variables $\hat{R}_j$s is introduced as partial remainder. $\hat{R}_j$ is calculated as

$$\hat{R}_j = sign(R_{j+1}).R_j$$

and its absolute value is given [4] by $|\hat{R}_j| = |R_j|$

Sign of $\hat{R}_j$ and R_j are calculated [4] as

$$sign(\hat{R}_j) = \begin{cases} sign(R_{j+1}) \cdot sign(R_j), & \text{if } R_j \neq 0 \\ +1, & \text{if } R_j = 0 \end{cases}$$

$$sign(R_j) = \begin{cases} sign(R_{j+1}) \cdot sign(\hat{R}_j), & \text{if } R_j \neq 0 \\ +1, & \text{if } R_j = 0 \end{cases}$$

Partial remainder is calculated using $\hat{R}_j = |\hat{R_{j+1}}| - 2^j \cdot D$ [4].It is represented as an $n + j$ digit SD2 representation and a pseudo-overflow correction is applied to the subtraction result [4]. Only the upper n digits of $\hat{R}_j$ are calculated while the least significant digits are taken from the corresponding bits of dividend with the sign based on the sign(R_j). $|\hat{R_{j+1}}|$ is calculated from $\hat{R_{j+1}}$ using the sign and absolute value calculation algorithm presented in chapter 2. The computation is continued till $j = 0$ even if $R_{j+1} = 0$ to have a regular structure. So, the quotient adjustment differs from what is considered in radix-2 non-restoring division algorithm [4]. In the final iteration, $\hat{R_0}^*$ is calculated as $|\hat{R_0}^*| - D$, because all the dividend bits will be used up by then.

The division algorithm implemented in this thesis for a 4-bit integer is as follows [4].

Divisor Y = $[y_3y_2y_1y_0]$ Dividend X = $[x_3x_2x_1x_0]$

For an n-bit, Divisor Y = $[y_{n-1} \ldots y_1y_0]$ Dividend X = $[x_{n-1} \ldots x_1x_0]$ taking $D = Y$ and $\hat{R}_n = X$

Step 1: Divisor sign determination

if $y_{n-1} = 0$ **then** ▷ divisor is positive

$D = Y$

else

$D = \overline{Y} + 1$ ▷ 2's complement

end if

Step 2: Dividend sign determination

if $x_{n-1} = 1$ **then** ▷ dividend is negative

$sign(R_n) = -1$

else

$sign(R_n) = +1$

end if

Step 3: Calculating partial remainder and its sign and then calculating quotient bit

for $j \leftarrow n-1$ downto 0 **do**

Step 3.1: Partial remainder

$\hat{R_j} = |\hat{R_{j+1}}|$ - $2^j \cdot D$

$\hat{R_j}$ = absolute($\hat{R_j}$) ▷ calculated using the algorithm presented in chapter 2

Step 3.2: Determining the sign of partial remainder

if $sign(\hat{R_j}') = -1$ or (sign($\hat{R_j}'$) = 0 and sign(R_{j+1})= -1) **then**

$sign(R_j) = -sign(R_{j+1})$

else

$sign(R_j) = sign(R_{j+1})$

end if

Step 3.3 : Determining the quotient bit

if $sign(R_{j+1}) = -1$ **then**

if $y_{n-1} = 0$ **then**

$p_{j+1} = 0$

else

$p_{j+1} = 1$

end if

else

if $y_{n-1} = 0$ **then**

$p_{j+1} = 1$

else

$p_{j+1} = 0$

end if

end if

end for

P(initial quotient) = $[p_{n-1}p_{n-2}\ldots p_1 1]$

Step 4: Quotient adjustment and remainder selection

if $x_{n-1} = 0$ **then**

if $sign(R_0)$ = -1 **then**

$R = |\hat{R_0}^*|$

if $y_{n-1} = 0$ **then**

$Z = P - 1$

else

$Z = P + 1$

end if

else

$R = |\hat{R_0}|$

$Z = P$

end if

else

if sign($\hat{R_0}^*$) = 0 **then**

R = - $|\hat{R_0}^*|$

if $y_{n-1} = 0$ **then**

$Z = P - 1$

else

$Z = P + 1$

end if

else if (sign($\hat{R_0}$)) $\neq$ 0 and sign(R_0) = +1) **then**

R = - $|\hat{R_0}^*|$

if $y_{n-1} = 0$ **then**

$Z = P + 1$

else

$Z = P - 1$

end if

else

R = -$|\hat{R}_0|$

$Z = P$

end if

end if

3.5 Combinational Implementation

The combinational implementation of the above algorithm yields the following regularly structured design which is quite feasible for implementation on a VLSI chip. The design has been shown for a 4-bit integer in Figure 3.1. It has 12 cells namely SUB, NEG, MUX, CMP, ABS, OVF, DVDSGN, PORTSIZE, SGN, ADJ, PADJ and NEGCONV. The blocks PORT SIZE and DVD SGN, not given the original reference paper [4] have been added for a clear presentation of the design. The design shows exactly what has been implemented in HDL. The function of each cell is as described below.

SUB: It performs the first stage of subtraction as shown in Table 2.1. The minuend is a signed number and the subtrahend is an ordinary binary producing an unsigned borrow and intermediate difference.

ABS: It performs the second stage of subtraction as shown in Table 2.2 thereby producing the partial remainder. It also calculates the absolute value of the partial remainder based on sign of $\hat{R}_j$ which is also calculated by ABS cell. It follows the

sign and absolute value calculation algorithm presented in Chapter 2.

OVF: It compensates the pseudo overflow correction. An overflow is generated because the partial remainder bits are shifted to the left to accommodate the LSB obtained from dividend. This cell receives 3 inputs from the previous OVF cell, ABS cell and SUB cell that contribute to the next partial remainder. It calculates the difference of the values of ABS and SUB cells and gives as input to its adjacent ABS cell. The input from OVF cell is sent to next OVF cell.

SGN: It calculates the sign of R_j from sign of R_{j+1} and sign of the partial remainder $\hat{R}_j$ and determines the quotient bit.

NEG: It determines the sign of LSB (obtained from the dividend) of the minuend for the next stage of subtraction, based on the sign of R_j calculated by the SGN cell.

CMP: It has two functions- One is to determine the sign the remaining bits of the divisor based on its MSB. These divisor bits serve as subtrahend in each stage of subtraction. Second function of CMP is to determine the sign of the quotient bit based on MSB of the divisor. The quotient is obtained from SGN cell.

DVDSGN: This cell is not shown in the original design given in the base paper. It is added to determine the sign of the dividend using its MSB and converting it into SD2 number which is given to SGN cell to determine the sign of the first partial remainder.

PORTSIZE: This cell is also not shown in base paper. It converts the MSB of the divisor into an SD2 number to be given as LSB of the minuend to obtain the first

partial remainder.

ADJ: This cell generates control signals for quotient adjustment (given to PADJ) and remainder selection (given to MUX).

PADJ: It produces the final quotient based on the control signal produced by ADJ.

NEGCONV: It converts an SD2 number to a 2's complement binary number. The conversion uses the concept of a carry lookahead adder[12].

MUX: It outputs remainder bits by choosing between $\hat{R_0}$ and $\hat{R_0}^*$ based on the control signal by the ADJ cell.

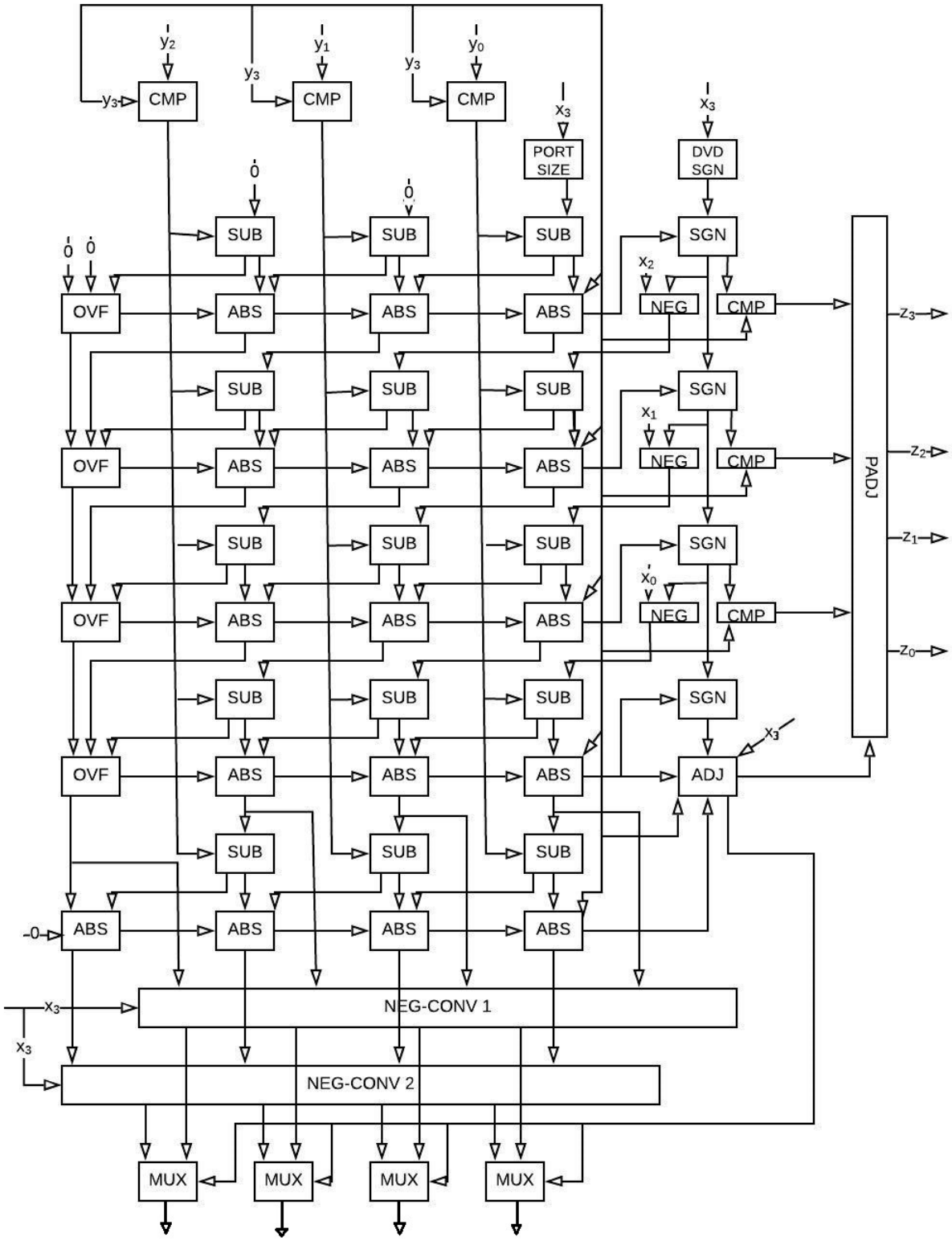

Figure 3.1: Combinational Implementation

CHAPTER IV

RESULTS

Realizing a design into real hardware involves two phases namely the front end and the back end. Front end includes determining the design specifications, architectural design, functional verification and power analysis, logic and test synthesis. The back end phase involves floor planning and CTS, layout design, placement and routing, physical verification and tapeout [13]. This summarizes VLSI design flow. This thesis realises the front end phase for the discussed division hardware algorithm.

The divider has been implemented in Verilog. Simulation and testing is performed using MGC ® ModelSim™. The code is tested for various 4-bit and 8-bit inputs. The entire process includes determining how the given algorithm works, understanding and modifying the given design to make it suitable for designing HDL, implement using a HDL, create a testbench with a couple of test vectors, simulate and finally compare the simulated results with the known results.

A sample of the manual division for 4-bit is shown here considering dividend and divisor as 5 and 2 respectively with all four variants: 5/2 (positive/positive) , -5/2 (negative/positive) , -5/-2 (negative/negative) and 5/-2 (positive/negative). Negative numbers are shown in 2s complement representation.

Positive/Positive:
$Dividend(X) = 5$ $Divisor(Y) = 2$ $Remainder(R) = 1$ $Quotient(Z) = 2$

X = $[x_3x_2x_1x_0]$ Y = $[y_3y_2y_1y_0]$
X = [0101] Y= [0010]
D = [010] $\text{sign}(\hat{R_j}')$ $\text{sign}(R_j)$ p_{j+1} ▷ determine divisor

$\hat{R_4}$ 0101 +1 ▷ sign of dividend

$|\hat{R_4}'|$ 0000 ▷ load register with 0s, LSB = $x_3 * sign(R_j)$
$-2^3 \cdot D$ -010 ▷ subtract to calculate the next partial remainder
0010
-0100
$\hat{R_3}'$ $0\bar{1}10$ −1 −1 0 ▷ sign and quotient bit calculation

$|\hat{R_3}'|$ $1\bar{1}0\bar{1}$ ▷ absolute value, shift left, LSB= $x_2 * sign(R_j)$from previous step
$-2^2 \cdot D$ -010 ▷ subtract to calculate the next partial remainder
1111
-1110
$\hat{R_2}'$ 0001 +1 −1 0 ▷ sign and quotient bit calculation

$|\hat{R_2}'|$ 0010 ▷ absolute value, shift left, LSB= $x_1 * sign(R_j)$from previous step
$-2^1 \cdot D$ -010 ▷ subtract to calculate the next partial remainder
0000
-0000
$\hat{R_1}'$ 0000 0 +1 +1 ▷ sign and quotient bit calculation

$|\hat{R_1}'|$ 0001 ▷ absolute value, shift left, LSB= $x_0 * sign(R_j)$from previous step
$-2^0 \cdot D$ -010 ▷ subtract to calculate the next partial remainder
0011
-0100
$\hat{R_0}$ $0\bar{1}11$ −1 −1 ▷ sign calculation

$|\hat{R_0}|$ $01\bar{1}\bar{1}$ ▷ absolute value
$-D$ -010 ▷ subtract to calculate the next partial remainder
0101 Quotient Adjustment
-0110 P=[0011]
$\hat{R_0}^*$ $00\bar{1}1$ −1 X > 0 and Y > 0
therefore Z = P-1 = [0010]
$|\hat{R_0}^*|$ $001\bar{1}$ Remainder Selection
X > 0 and $\text{sign}(R_0)$ = -1
therefore R = $|\hat{R_0}^*|$ =[0001]

Negative/Positive:
$Dividend(X) = -5$ $Divisor(Y) = 2$ $Remainder(R) = -1$ $Quotient(Z) = -2$

X = [1011] Y= [0010]
D = [010]

		sign($\hat{R_j}$`)	signR_j	p_{j+1}	
$\hat{R_4}$	1011		-1		
$\lvert\hat{R_4}`\rvert$	0001				
$-2^3 \cdot D$	-010				
	0011				
	-0100				
$\hat{R_3}`$	$0\bar{1}11$	-1	$+1$	$+1$	
$\lvert\hat{R_3}`\rvert$	$1\bar{1}\bar{1}0$				
$-2^2 \cdot D$	-010				
	1100				
	-1100				
$\hat{R_2}`$	0000	0	$+1$	$+1$	
$\lvert\hat{R_2}`\rvert$	0001				
$-2^1 \cdot D$	-010				
	0011				
	-0100				
$\hat{R_1}`$	$0\bar{1}11$	-1	-1	0	P=[1101]
					X < 0 and Y > 0
$\lvert\hat{R_1}`\rvert$	$1\bar{1}\bar{1}\bar{1}$				therefore Z = P+1 = [1110]
$-2^0 \cdot D$	-010				
	1101				X < 0
	-1110				sign($\hat{R_0}$)$\neq$ 0 and sign(R_0) = +1
$\hat{R_0}$	$00\bar{1}1$	-1	$+1$		therefore R = -$\lvert\hat{R_0}^*\rvert$ =[1111]
$\lvert\hat{R_0}\rvert$	$001\bar{1}$				
$-D$	-010				
	0001				
	-0010				
$\hat{R_0}^*$	$00\bar{1}1$	-1			
$\lvert\hat{R_0}^*\rvert$	$001\bar{1}$				

Negative/Negative:

$Dividend(X) = -5$ $Divisor(Y) = -2$ $Remainder(R) = -1$ $Quotient(Z) = 2$

X = [1011] Y= [1110]

D = [001]+1		sign($\hat{R_j}$`)	signR_j	p_{j+1}	
$\hat{R}_4$	1011		-1		
$\lvert\hat{R_4}\text{`}\rvert$	0001				
$-2^3 \cdot D$	-001				
	0000				
	-0001				
$\hat{R_3}$`	000$\bar{1}$	-1	$+1$	0	
$\lvert\hat{R_3}\text{`}\rvert$	0010				
$-2^2 \cdot D$	-001				
	0011				
	-0011				
$\hat{R_2}$`	0000	0	$+1$	0	
$\lvert\hat{R_2}\text{`}\rvert$	0001				
$-2^1 \cdot D$	-001				
	0000				
	-0001				
$\hat{R_1}$`	000$\bar{1}$	-1	-1	$+1$	P=[0011]
					X < 0 and Y < 0
$\lvert\hat{R_1}\text{`}\rvert$	001$\bar{1}$				therefore Z = P-1 = [0010]
$-2^0 \cdot D$	-001				
	0010				X < 0
	-0011				sign($\hat{R_0}$)$\neq 0$ and sign(R_0) $= +1$
$\hat{R_0}$	000$\bar{1}$	-1	$+1$		therefore R = -$\lvert\hat{R_0}^*\rvert$ =[1111]
$\lvert\hat{R_0}\rvert$	0001				
$-D$	-001				
	0000				
	-0001				
$\hat{R_0}^*$	000$\bar{1}$	-1			
$\lvert\hat{R_0}^*\rvert$	0001				

Positive/Negative:

$Dividend(X) = 5$ $Divisor(Y) = -2$ $Remainder(R) = 1$ $Quotient(Z) = -2$

X = [0101] Y= [1110]

D = [001]+1		sign($\hat{R_j}$`)	signR_j	p_{j+1}	
$\hat{R}_4$	0101		+1		
$\lvert\hat{R_4}`\rvert$	0000				
$-2^3 \cdot D$	-001				
	0001				
	-0011				
$\hat{R_3}`$	$00\bar{1}0$	-1	-1	$+1$	
$\lvert\hat{R_3}`\rvert$	$010\bar{1}$				
$-2^2 \cdot D$	-001				
	0100				
	-0011				
$\hat{R_2}`$	$01\bar{1}\bar{1}$	$+1$	-1	$+1$	
$\lvert\hat{R_2}`\rvert$	$1\bar{1}\bar{1}0$				
$-2^1 \cdot D$	-001				
	1111				
	-1111				
$\hat{R_1}`$	0000	0	+1	0	P=[1101]
					X > 0 and Y < 0
$\lvert\hat{R_1}`\rvert$	0001				therefore Z = P+1 = [1110]
$-2^0 \cdot D$	-001				
	0000				X > 0
	-0001				sign(R_0) = -1
$\hat{R_0}$	$000\bar{1}$	-1	-1		therefore R = $\lvert\hat{R_0}^*\rvert$ =[0001]
$\lvert\hat{R_0}\rvert$	0001				
$-D$	-001				
	0000				
	-0001				
$\hat{R_0}^*$	$000\bar{1}$	-1			
$\lvert\hat{R_0}^*\rvert$	0001				

Simulation results are shown in Figures 4.1 and 4.2 for 4-bit and 8-bit respectively. The waveforms show division computed for 20 input vectors and the results tally with the theoretical results. The signals shown in the waveform are divisor, dividend, quotient and remainder along with few internal signals like sign of the divisor, control signal for quotient and remainder, overflow in each iteration and so forth.

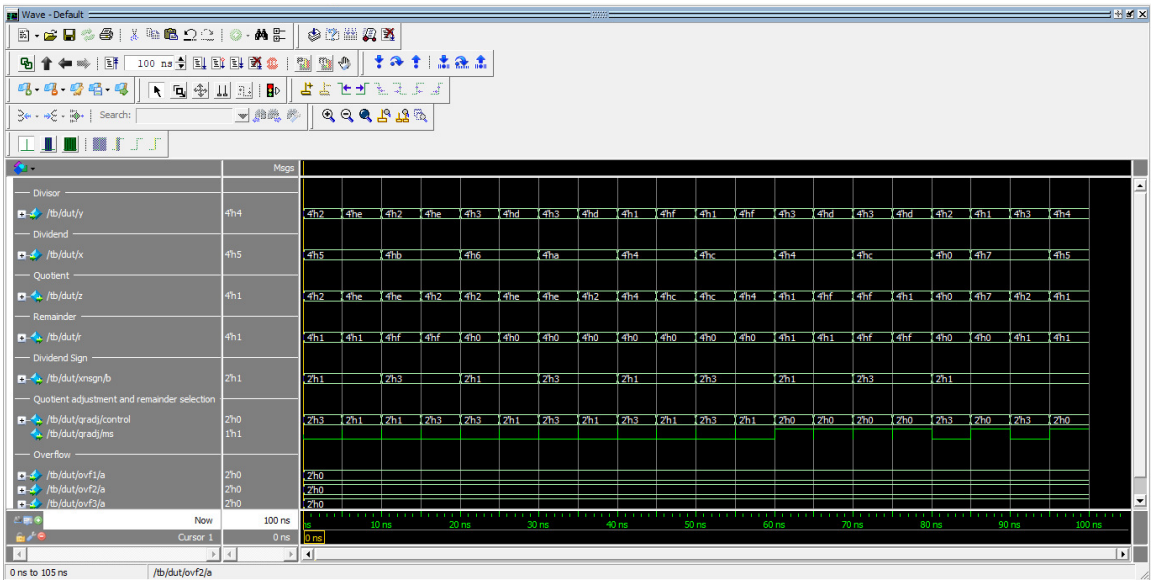

Figure 4.1: Waveform generated in MGC ® ModelSim™ for 4-bit implementation

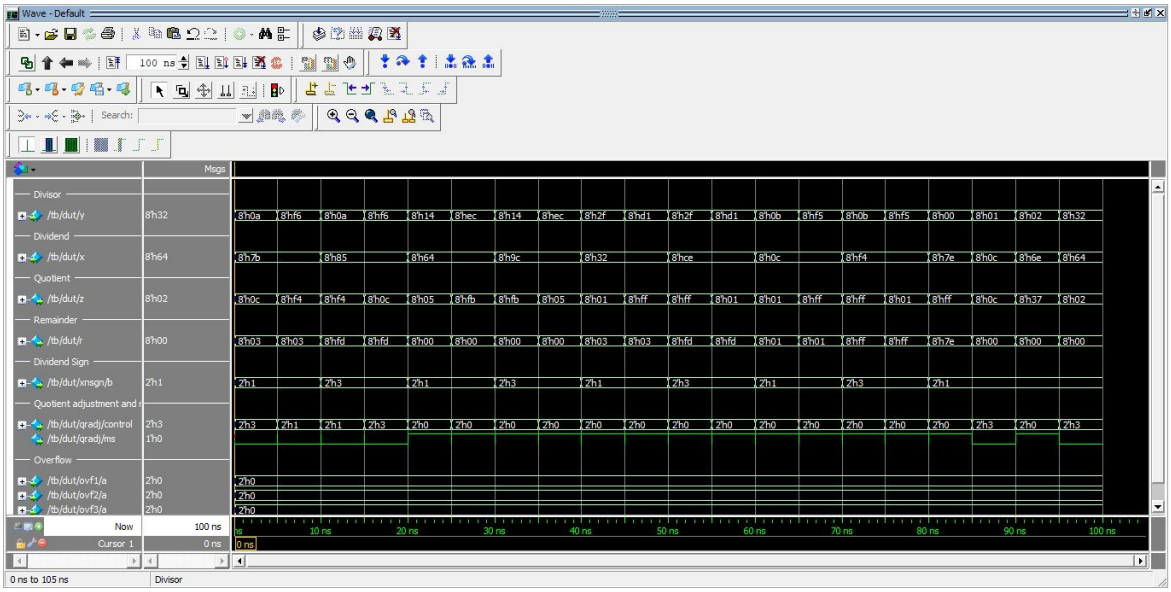

Figure 4.2: Waveform generated in MGC ® ModelSim™ for 8-bit implementation

4.1 Synthesis

After the architectural design is obtained and digitally implemented using a HDL, simulation and functional verification is done to test if the RTL design matches the specifications, the design is expressed in the form of basic circuit elements likes gates and transistors which is called gate level netlist. This is called synthesis. It is performed using a synthesis tool. The tool uses a standard cell library, constraints and the RTL code to produce the gate-level netlist. Parameters like the total cell area, number of cells used, computation time and power are calculated for both 4-bit and 8-bit which are shown in Table 4.1. This thesis uses 0.5m Scalable CMOS (SCMOS) standard-cells for synthesis.

	4-bit	8-bit
AREA(mm^2)	0.046134	0.439299
TOTAL CELLS	203	1,809
COMPUTATION TIME(ns)	9.957	10.694
POWER		
Internal Power(mW)	2.303970	28.059448
Switching Power(mW)	3.016685	29.666616
Leakage Power(nW)	13.018942	128.886765
Total Power(mW)	5.320667	57.726223

Table 4.1: Synthesis results for 4-bit and 8-bit implementation targeted at 100MHz

Total power is the sum of internal, switching and leakage powers. Switching power is dynamic while internal and leakage are static. The next step is to create a layout using layout synthesis tools. It is created by converting each logic component (cells, macros, gates, transistors) into a geometric representation (specific shapes in multiple layers), which perform the intended logic function of the corresponding component [14]. After the layout is verified it is sent for fabrication.

CHAPTER V

CONCLUSION AND FUTURE WORK

Combinational implementation of a hardware algorithm for integer division is presented in this thesis. A 4-bit and 8-bit implementation of the design has been performed in Verilog HDL. It is simulated and tested using MGC ® ModelSim™. RTL synthesis is performed and parameters like area, total cells, timing and power are computed. Algorithm is based on digit-recurrence and non-restoring division that uses radix-2 signed-digit representation. Each partial remainder is represented as an SD2 integer with a sign and absolute value. Quotient is obtained from each partial reminder as an ordinary binary integer. Remainder requires conversion from SD2 to binary. Four variants of integer division i.e. positive/positive, negative/positive, negative/negative and positive/negative are shown. Results prove that division has been carried out correctly. The advantage of this algorithm over many existing algorithms is that it takes very less computation time which can be stated from the number representation used. It also uses less power as it is a combinational implementation and not sequential. Also, non-restoring algorithm produces a quicker output when compared to the restoring algorithm because it skips the restoring step after subtraction.

Future work is to integrate the design into hardware i.e. perform the back end phase of a VLSI design flow. Also, improving the logic of cells like NEG-CONV (SD2 to binary conversion) for faster computation is a prospect.

REFERENCES

[1] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Norwell, MA, USA: Kluwer Academic Publishers, 1994.

[2] N. Takagi, "Studies on hardware algorithms for arithmetic operations with a redundant binary representation," Jan 1988. [Online]. Available: http://hdl.handle.net/2433/154053

[3] C.-C. Wang, C.-J. Huang, and G.-C. Lin, "Cell-based implementation of radix-4/2 64b dividend 32b divisor signed integer divider using the compass cell library," *IEEE Proceedings - Computers and Digital Techniques*, vol. 147, no. 2, pp. 109–115, Mar 2000.

[4] N. Takagi, S. Kadowaki, and K. Takagii, "A hardware algorithm for integer division," *17th IEEE Symposium on Computer Arithmetic (ARITH'05)*, June 2005.

[5] N. Boullis and A. Tisserand, "On digit-recurrence division algorithms for self-timed circuits," *Proceedings of SPIE - The International Society for Optical Engineering*, Nov 2001.

[6] "Redundant arithmetic," http://people.ece.umn.edu/users/parhi/SLIDES/chap14.pdf.

[7] "Redundant numeral systems," http://arogozhnikov.github.io/2013/08/10/redundant-numeral-systems.html.

[8] M. Lu, *Arithmetic and Logic in Computer Systems*. Wiley-Interscience, 2004.

[9] "Restoring division algorithm for unsigned integer," https://www.geeksforgeeks.org/restoring-division-algorithm-unsigned-integer.

[10] "Non-restoring division for unsigned integer," https://www.geeksforgeeks.org/non-restoring-division-unsigned-integer.

[11] "Division algorithm," https://en.wikipedia.org/wiki/Division_algorithm.

[12] S. Yen, C. Laih, C. Chen, and J. Lee, "An efficient redundant-binary number to binary number converter," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 109–112, Jan 1992.

[13] "SoC verification using System Verilog," https://www.udemy.com/soc-verification-systemverilog/learn/lecture/942636?start=270#overview.

[14] "VLSI design flow," http://vlsibyjim.blogspot.com/2015/03/vlsi-design-flow.html.

[15] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 389–400, Sep 1961.

[16] J. E. Stine, *Digital Computer Arithmetic Datapath Design Using Verilog HDL*. Norwell, Massachusetts, USA: Kluwer Academic Publishers, 2004.

[17] H. Dawid and H. Meyr, "The differential cordic algorithm: Constant scale factor redundant implementation without correcting iterations," *IEEE Transactions on Computers*, vol. 45, no. 3, pp. 307–318, March 1996.

[18] "Division, computer arithmetic," https://www.ece.ucsb.edu/~parhami/pres_folder/f31-book-arith-pres-pt4.pdf.

[19] "Sequential algorithms for multiplication and division," http://euler.ecs.umass.edu/arith/parts-pdf/Part3-seq.pdf.

VITA

Rachana Erra

Candidate for the Degree of

Master of Science

Thesis: IMPLEMENTATION OF A HARDWARE ALGORITHM FOR INTEGER DIVISION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Hanamkonda, Telangana, India on July 20, 1996.

Education:

Received B.Tech. degree from Kakatiya Institute of Technology and Science, Kakatiya University, Warangal, Telangana, India, 2017, in Electronics and Communication Engineering

Completed the requirements for the degree of Master of Science with a major in Electrical Engineering at Oklahoma State University, Stillwater, Oklahoma in August, 2019.

Experience:

Graduate Teaching Assistant - OSU
Digital Integrated Circuit Design- Fall 2017, Fall 2018
Computer Architecture- Spring 2018, Spring 2019
Fundamentals of Electric Circuits- Fall 2018, Spring 2019