Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 03, March-2023

FPGA Implementation of Single Cycle Signed Multiplier using Booth Recoding Algorithm

Piyush Pati
Electronics and Instrumentation Engineering
Odisha University of Technology and Research,
Bhubaneswar, Odisha, India

***Abstract*—The focus of this paper is on the implementation of a single cycle signed multiplier through use of the booth recoding algorithm on an FPGA. By utilizing fewer partial products, this implementation offers benefits such as reduced delay, power consumption, and usage of hardware resources. Additionally, this signed multiplier is capable of performing multiplication of both signed and unsigned numbers. The paper presents a comparative analysis of the 32-bit multiplier's performance in terms of power consumption and FPGA hardware resource utilization. The proposed 32-bit multiplier is designed using Verilog HDL and implemented through Xilinx Vivado 2022.2 software for Xilinx Virtex-7 FPGA.**

***Keywords*—Booth recoding; Signed Multiplier; FPGA; Verilog HDL.**

I. INTRODUCTION

The process for binary multiplication is analogous to that of the decimal system. The regulations governing binary multiplication are illustrated in the following table.

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Table 1:1-bit binary multiplication

In binary arithmetic, when multiplying two numbers, each bit of the multiplier is multiplied by the multiplicand one at a time, similar to how it's done in the decimal system. The resulting partial products are arranged such that the least significant bit (LSB) is positioned under the corresponding bit in the multiplier.

Once the partial products have been calculated through binary multiplication, they are summed together to form the final product.

It is important to keep in mind that any multiplication by zero would result in all bits of the partial product being zero, which can be skipped in intermediate steps.

Furthermore, when multiplied by 1, the bits of the multiplicand remain unaltered, but they are shifted one bit position to the left. Performing intermediate sums of partial products simplifies the multiplication process of binary numbers.

Binary multiplication offers several benefits. It involves adding the multiplicand to itself, after a suitable shift based on the multiplier, which simplifies the process into a series of shifting and adding steps. These steps should be repeated until the MSB of the multiplier has been shifted, and the final addition has been performed.

The paper is organized as follows. Section II presents the design of booth recoding multiplier. Section III contains the sub-modules of implemented booth recoding multiplier. Section IV describes partial product addition unit. Experimental results and comparison are given in Section V. Finally, Section VI concludes the paper.

II. DESIGN OF BOOTH RECODING MULTIPLIER

A. 32x32 bit multiplier(Heading 2)

The Booth recoding multiplier requires two 32-bit inputs representing the multiplier and multiplicand, respectively. The output of the multiplier is 64-bit. Additionally, two control signals are included to specify whether the multiplier and multiplicand are signed or unsigned.

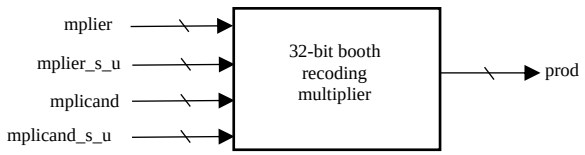

Figure 1: 32-bit booth recoding multiplier top module

Signal Name	Width	Source	Description
mplier	32	input	Top module multiplier input
mplier_s_u	1	input	1= multiplier is signed, 0 =multiplier is unsigned
mplicand	32	input	Top module multiplicand input
mplicand_s_u	1	input	1 = multiplier is signed, 0 =multiplier is unsigned
prod	64	output	Output from the multiplier block

Table 2: Signal Description of top module

B. Mathematical Representation of Signed Number

2's complement representation of A

$$= -2^{31}a_{31} + 2^{30}a_{30} + (2-1)^{29}a_{29} + 2^{28}a_{28} + (2-1)^{27}a_{27} + 2^{26}a_{26} +$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 03, March-2023

$$(2-1)^{25}a_{25} + \ldots\ldots\ldots\ldots + 2^2a_2 + (2-1)^1a_1 + 2^0a_0 + 2^0a_{-1}$$

where $a_{-1} \equiv 0$

$$= -2^{31}a_{31} + 2^{30}a_{30} + 2^{30}a_{29} - 2^{29}a_{29} + 2^{28}a_{28} + 2^{28}a_{27} - 2^{27}a_{27} + 2^{26}a_{26} + 2^{26}a_{25}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$-2^3a_3 + 2^2a_2 + 2^2a_1 - 2^1a_1 + 2^0a_0 + 2^0a_{-1}$$

where $a_{-1} \equiv 0$

$$= 2^{30}(-2^{31}a_{31} + 2^{30}a_{30} + 2^{30}a_{29}) 2^{28}(-2^{29}a_{29} + 2^{28}a_{28} + 2^{28}a_{27}) 2^{26}(-2^{27}a_{27} + 2^{26}a_{26} + 2^{26}a_{25})$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$2^2(-2^3a_3 + 2^2a_2 + 2^2a_{1)}) 2^0(-2^1a_1 + 2^0a_0 + 2^0a_{-1})$$

where $a_{-1} \equiv 0$

$$= \sum_{i=0}^{15} 2^{2i}(-2a_{2i+1} + a_{2i} + a_{2i-1})$$

$$= \sum_{i=0}^{15} 2^{2i} f_{2i} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

$$where\ f_{2i} = -2a_{2i+1} + a_{2i} + a_{2i-1}$$

C. Mathematical Representation of Unsigned Number

$$= -2^{32}a_{32} + 2^{31}a_{31} + 2^{31}a_{30} - 2^{30}a_{30} + 2^{29}a_{29} + 2^{29}a_{28} - 2^{28}a_{28} + 2^{27}a_{27} + 2^{27}a_{26}$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$-2^2a_2 + 2^1a_1 + 2^1a_0 - 2^0a_0 + 2^{-1}a_{-1} + 2^{-1}a_{-2}$$

Where $a_{-1} = a_{-2} \equiv 0$

$$= 2^{31}(-2a_{32} + a_{31} + a_{30}) + 2^{29}(-2a_{30} + a_{29} + a_{28}) +$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$2^1(-2a_2 + a_1 + a_0) + 2^{-1}(-2a_0 + a_{-1} + a_{-2})$$

Where $a_{-1} = a_{-2} \equiv 0$

$$= \sum_{i=0}^{16} 2^{2i-1}(-2a_{2i} + a_{2i-1} + a_{2i-2})$$

$$= \sum_{i=0}^{16} 2^{2i-1} f_{2i} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

$$where\ f_{2i-1} = -2a_{2i} + a_{2i-1} + a_{2i-2}$$

Comparing equation 1 and 2 it is apparent that one can process the integer A before re-coding basic unsigned and signed integer.

a_{2i+1}	a_{2i}	a_{2i-1}	f_{2i}	$\overline{F}$ (+/−)	F^1 (x1/0)	F^2 (x2/0)
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	2	0	1	1
1	0	0	-2	1	1	1
1	0	1	-1	1	1	0
1	1	0	-1	1	1	0
1	1	1	0	0	0	0

Table 3:Booth recoding truth table

III. SUB-MODULES OF IMPLEMENTED BOOTH RECODING MULTIPLIER

A. 33rd bit extension unit

Our implementation involves using a signed multiplier to perform an unsigned operation. To cover the entire range of unsigned multiplication, we add an extra bit at the MSB. For unsigned numbers, the 33rd bit is zero, while for signed numbers, the 33rd bit is sign-extended.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 03, March-2023

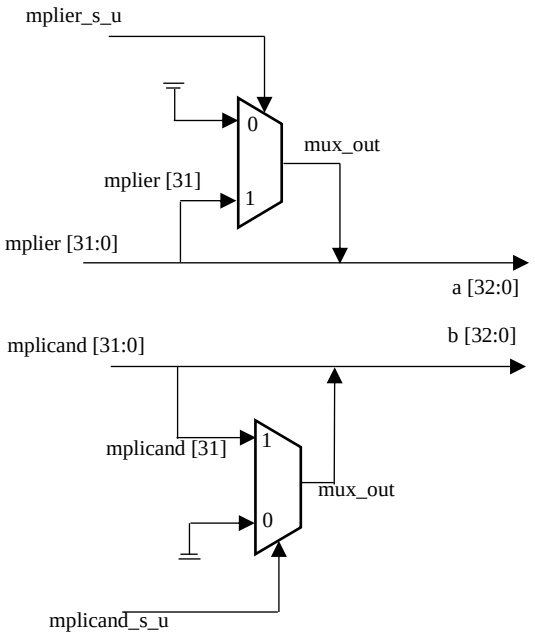

Figure 2: 33rd bit extension unit block diagram

B. *Preprocessing/F block formation unit*

- During the pre-processing phase, we will append a "0" to the least significant bit (LSB) of "a", which is a 32-bit input to the pre-processing block, and serves as the output of the 33rd bit extension unit.
- Following that, we need to create blocks consisting of three bits each, where the least significant bit (LSB) of the current block becomes the most significant bit (MSB) of the previous block.
- Since we are forming blocks in the 33-bit number "a", adding a "0" to the least significant bit results in a shortage of one bit required to form the F block.
- To address this situation, we will add a number to the most significant bit (MSB) that is the same as the 33rd bit, regardless of whether we are generating signed or unsigned partial products.

Extra bit added for block formation	Extended 33-bit number (output of 33rd bit extension unit)	0 added in LSB
a [32]	a [32:0]	0

Table 4:Extended 34-bit multiplier

After preprocessing total width of the number is 34-bit.

Note: preprocessing of number "a" is nothing but just rewiring.

F0	$\{a_1, a_0, 0\}$
F2	$\{a_3, a_2, a_1\}$
F4	$\{a_5, a_4, a_3\}$
F6	$\{a_7, a_6, a_5\}$
F8	$\{a_9, a_8, a_7\}$
F10	$\{a_{11}, a_{10}, a_9\}$
F12	$\{a_{13}, a_{12}, a_{11}\}$
F14	$\{a_{15}, a_{14}, a_{13}\}$
F16	$\{a_{17}, a_{16}, a_{15}\}$
F18	$\{a_{19}, a_{18}, a_{17}\}$
F20	$\{a_{21}, a_{20}, a_{19}\}$
F22	$\{a_{23}, a_{22}, a_{21}\}$
F24	$\{a_{25}, a_{24}, a_{23}\}$
F26	$\{a_{27}, a_{26}, a_{25}\}$
F28	$\{a_{29}, a_{28}, a_{27}\}$
F30	$\{a_{31}, a_{30}, a_{29}\}$
F32	$\{a_{32}, a_{32}, a_{31}\}$

Table 5: F Block Formation

C. *Partial product generation*

- There will be 16 rows of partial products for each of the 16 "F" blocks. Table 3 was used to generate $\bar{F}, F^1$ and F^2 which are then used for the required bit manipulation to generate the necessary partial products. The hardware implementation for $\bar{F}, F^1$ and F^2 is provided below.

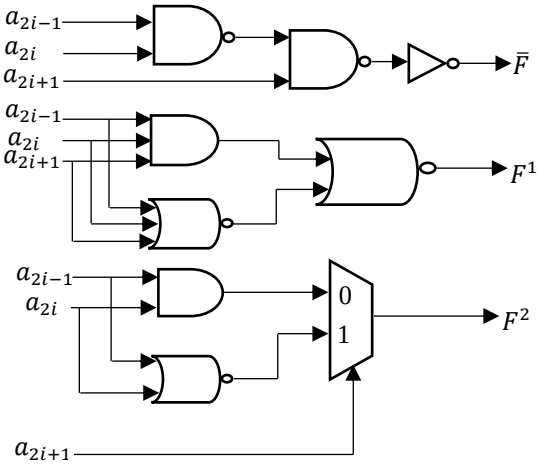

Figure 3: Hardware realization for $\bar{F}, F^1$ and F^2

Note: 1. $\bar{F}$ is high when f_{2i} is -ve, low otherwise.

2. F^1 is high for $f_{2i} \neq 0$,low otherwise.

3. F^2 is high for $f_{2i} = \pm 2$,low otherwise.

- We need to process all the F block in this hardware to get corresponding $\bar{F}, F^1$ and F^2. For example, if we process F0 block we can get the $\overline{F0}, F0^1$ and $F0^2$. Similarly, we can get the values for $\overline{F2}, F2^1$ and $F2^2$------$\overline{F32}, F32^1$ and $F32^2$by processing corresponding F blocks.
- As we are using booth recoding algorithm, so our partial product will reduce. There will be 17 rows of partial

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 03, March-2023

product which we need to add. For generating ROW#0 partial product we need to $\overline{F0}, F0^1$ and $F0^2$ in b (33-bit multiplicand, output of preprocessing unit). Similarly for ROW#2 we need to use $\overline{F2}, F2^1$ and $F2^2$ on "b" and so on for other ROW'S

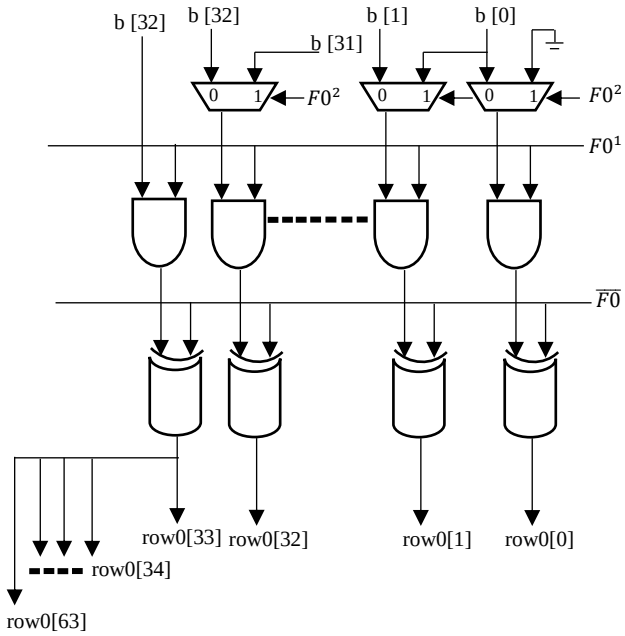

Figure 4: ROW#0 calculation using $\overline{F0}, F0^1$ and $F0^2$

Similar structure can be used for calculation of other ROW's.

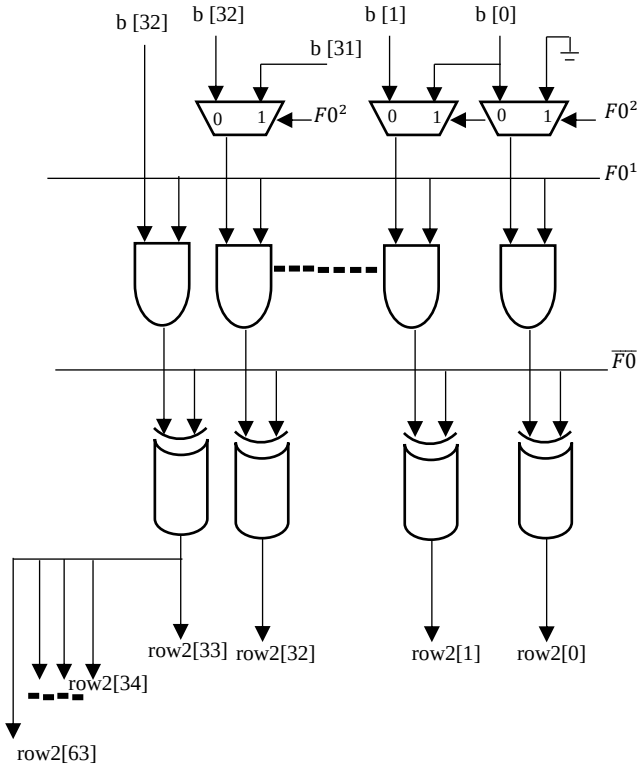

Figure 5: ROW#2 calculation using $\overline{F2}, F2^1$ and $F2^2$

IV. PARTIAL PRODUCT ADDITION UNIT

The process in this unit involves the addition of all partial products, namely ROW#0, ROW#2, ..., ROW#32. To achieve this, we use full adders and half adders. The carries are propagated diagonally to the left and downward to achieve superior speed. However, when processing the last row, ROW#32, there is no row below it, and so we propagate the carries horizontally instead.

Note: Horizontal carry propagation is feasible because the augend is unoccupied, given the absence of a row beneath it.

It is worth noting that if the value of f_{2i} is negative, we must compute the 2's complement of the number "b". However, the structure depicted in figures 4 and 5 merely perform XOR operations. Additionally, we need to add 1 to the LSB position, aligned with the binary values of the respective ROWs. To address this, we introduce an extra ROW, known as ROW#-1, which will facilitate the addition of 1 to the aligned LSB position for the relevant ROWs.

ROW#-1= {32'b0, $\overline{F32}$, 0 , $\overline{F30}$, 0 , $\overline{F28}$, 0 , $\overline{F26}$, 0 ,$\overline{F24}$, 0 ,$\overline{F22}$, 0 ,$\overline{F20}$, 0 , $\overline{F18}$, 0 , $\overline{F16}$, 0 , $\overline{F14}$, 0 , $\overline{F12}$, 0 ,$\overline{F10}$, 0 ,$\overline{F8}$, 0 ,$\overline{F6}$, 0 , $\overline{F4}$, 0 , $\overline{F2}$, 0 , $\overline{F0}$}

A. Addition Stages

During the initial stage of addition, we will add ROW#-1, ROW#0, and ROW#2 (shifted left by 2 positions). In the subsequent stage, we will add the sum obtained from the previous stage, ROW#4 (shifted left by 4 positions), and the carry generated by the first stage (shifted left by 1 position). This procedure is repeated in a similar manner for the succeeding stages of addition.

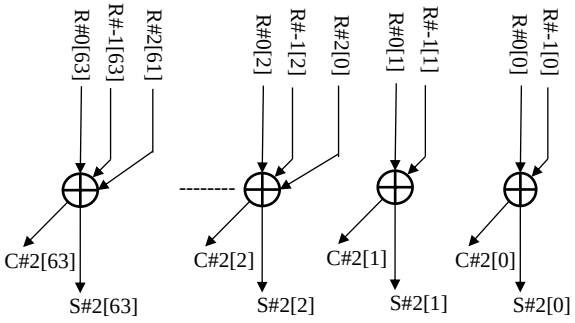

Figure 6 : First stage addition block

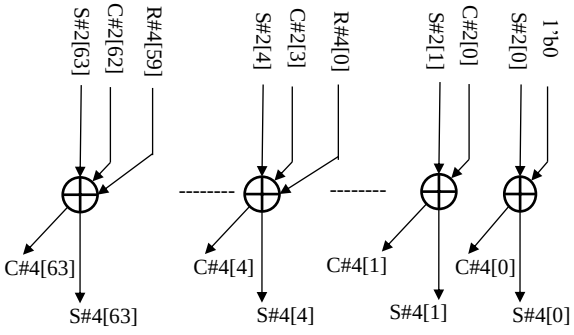

Figure 7: Second stage addition block

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 12 Issue 03, March-2023

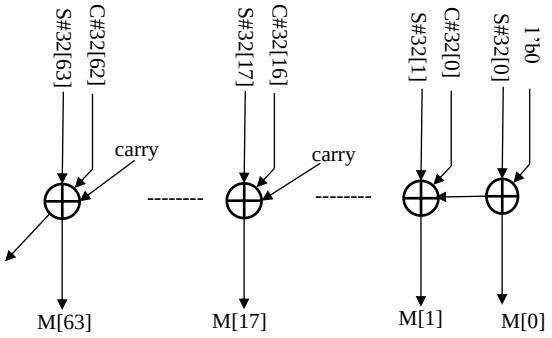

Figure 8: Final Stage addition block

V. RESULTS AND COMPARISON

A test bench program has been developed to verify the results of the implemented single-cycle signed multiplier using the Booth recoding algorithm. The code was written and executed using Xilinx Vivado 2022.2. The synthesized design was targeted towards the Virtex-7 FPGA, specifically the Device-XC7V585T with Package-FFG1157 and speed-1. Table-6 presents a comparison between the implemented single-cycle signed multiplier and the 32-bit complex Vedic multiplier [2].

Parameter	32-bit Vedic Multiplier	Implemented 32-bit Signed Multiplier	Improvement
Slice LUTs	7874	1305	83.43%
Bonded IOBs	258	130	49.61%

Table 6: Comparison with Vedic multiplier

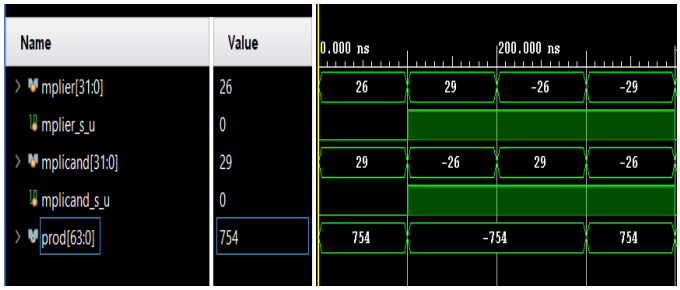

Figure 9 : Simulation Waveform

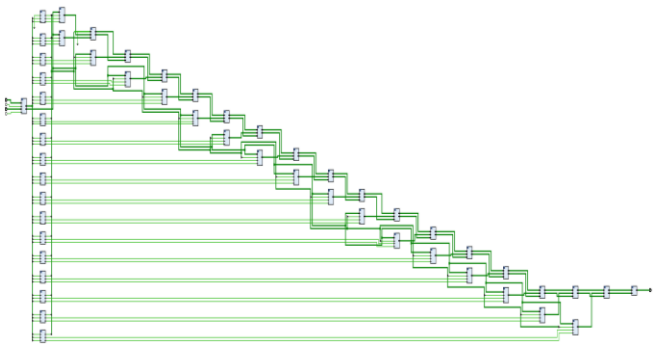

Figure 10 : RTL schematic of Implemented multiplier

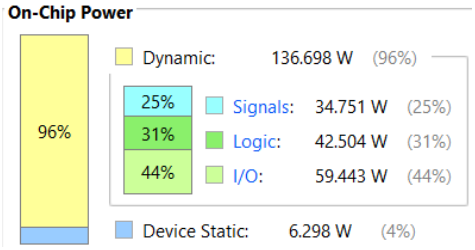

Figure 11 : On Chip Power

Parameter		Utilization
Bonded IOB		130
Slice	SLICEL	223
	SLICEM	144
LUT as Logic	using o5 output only	0
	using o6 output only	919
	using o5 and 06	386

Table 7:FPGA hardware utilization

VI. CONCLUSION

The architecture we propose in this paper can multiply two 32-bit signed numbers using the Booth recoding algorithm. Compared to conventional multipliers, our multiplier algorithm requires only half of the partial product addition. We examine the implemented multiplier's performance using various parameters, including power and hardware utilization. Compared to the 32-bit complex Vedic multiplier [2], the slice LUTs and bonded IOBs utilizations are significantly better, with improvements of 83.43% and 49.61%, respectively.

REFERENCES

[1] Saha, P., Banerjee, A., Bhattacharyya, P., and Dandapat, A. (2011, January). "High speed ASIC design of complex multiplier using vedicmathematics". In Students' Technology Symposium (TechSym), 2011 IEEE (pp. 237-241). IEEE.

[2] Ankush Nikam, Swati Salunke, Sweta Bhurse. "Design and Implementation of 32bit Complex Multiplier using Vedic Algorithm" IJERT ,2015 March,Vol 4.

[3] A. Dandapat, S. Ghosal, P. Sarkar, D. Mukhopadhyay, "A 1.2-ns16×16-Bit Binary Multiplier Using High Speed Compressors", International Journal of Electrical and Electronics Engineering, 2010.

[4] M. Morris Mano, "Digital Design",3rd edition, Prentice Hall,2002.