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

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

FPGA Realization of Single-Cycle, 32-Bit Booth Recoding Signed Multiplier Enhanced by High-Speed Compressors

Prity Mishra
Electronics and Telecommunication Engineering
International Institute of Information Technology,
Bhubaneshwar, Odisha, India

***Abstract*—A multiplier is one of the integral and frequently used parts of every digital system. In this paper, a 32x32-bit signed multiplier has been implemented using the booth recoding algorithm and high-speed compressors. This in turn facilitates hardware reduction as partial product rows are significantly reduced, thus improving efficiency in terms of speed and power consumption. An analytical comparison of this multiplier with the conventional multiplier has also been provided. This booth recoding multiplier has been designed using the SYSTEM VERILOG HDL and FPGA realization has been achieved through Xilinx Kintex-7 FPGA in Xilinx Vivado 2021.2 software.**

***Keywords*—Booth Recoding; 32x32 bit signed multiplier; Agile compressors; Parallel addition; Verilog HDL; FPGA.**

I. INTRODUCTION

Low-power VLSI circuits have emerged as critical criteria for designing energy-efficient electronic systems tailored for high-performing portable devices. Across various domains such as microprocessors, image and video processing, digital filters, error correction and coding, neural networks, and machine learning, multipliers play an integral role in enhancing computational efficiency. This paper presents the implementation of a 32x32-bit signed multiplier using the Booth recoding algorithm and high-speed agile compressors to improve processing speed. The inputs can be either signed or unsigned.

Conventional multiplication relies on the partial products method, where each bit of the multiplier is multiplied with the multiplicand to form partial product rows. Subsequently, these rows are left-shifted and summed to produce the final result.

A similarity is observed in case of multiplication of two binary numbers. This process is simplified as it involves only two digits—1 and 0—whereby multiplication with ‘1’ results in a direct copy and same with ‘0’ is discarded. However, for numbers with a high number of bits, processing time increases, leading to elevated time complexity and hardware demands.

In this paper, the focus will be on overcoming those major drawbacks with the proposed multiplier design. The paper has been divided into subsequent sections as follows. Section II provides a comprehensive overview of the traditional booth recoding algorithm. Section III goes into detail explaining the operation of the high-speed, agile compressors. The comparative analysis and the experimental results have been included in Section IV, followed by the concluding remarks in Section V.

II. CONVENTIONAL BOOTH RECODING ALGORITHM

A. Abstract block diagram of top module of the multiplier and the signal description

For booth recoding algorithm-based multiplier, two 32-bit inputs are required to represent the multiplicand and multiplier. In addition to that, two control signals are also fed to indicate whether the multiplier and multiplicand are signed or unsigned binary values. The output is a 64-bit value.

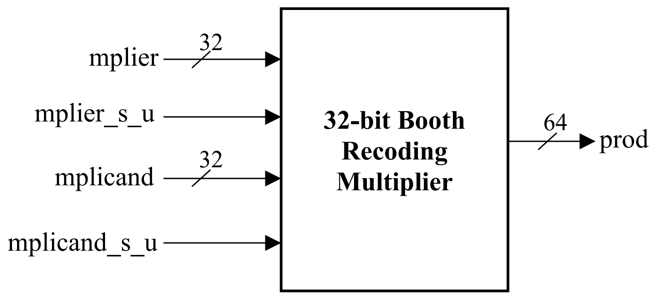


Table 1:Signal Description of Top Module

Figure 1:32-bit Booth Recoding Multiplier

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→multiplicand is signed 0→multiplicand is unsigned
prod	64	output	Output of the multiplier block

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

B. Mathematical representation of signed and unsigned numbers

Signed number:

2s complement representation of A:

$$
\begin{aligned}
&=-2^{31}a_{31}+2^{30}a_{30}+(2-1)2^{29}a_{29}\\
&\quad+2^{28}a_{28}+(2-1)2^{27}a_{27}\\
&\quad+2^{26}a_{26}+(2-1)2^{25}a_{25}\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad+2^{2}a_{2}+(2-1)2^{1}a_{1}\\
&\quad+2^{0}a_{0}+2^{0}a_{-1} \qquad \text{where } a_{-1}\equiv 0
\end{aligned}
$$

$$
\begin{aligned}
&=-2^{31}a_{31}+2^{30}a_{30}+2^{30}a_{29}\\
&\quad-2^{29}a_{29}+2^{28}a_{28}+2^{28}a_{27}\\
&\quad-2^{27}a_{27}+2^{26}a_{26}+2^{26}a_{25}\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad-2^{3}a_{3}+2^{2}a_{2}+2^{2}a_{1}\\
&\quad-2^{1}a_{1}+2^{0}a_{0}+2^{0}a_{-1} \qquad \text{where } a_{-1}\equiv 0
\end{aligned}
$$

$$
\begin{aligned}
&=2^{30}(-2\,a_{31}+a_{30}+a_{29})\\
&\quad+2^{28}(-2\,a_{29}+a_{28}+a_{27})\\
&\quad+2^{26}(-2\,a_{27}+a_{26}+a_{25})\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad+2^{2}(-2\,a_{3}+a_{2}+a_{1})\\
&\quad+2^{0}(-2\,a_{1}+a_{0}+a_{-1}) \qquad \text{where } a_{-1}\equiv 0
\end{aligned}
$$

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

$$=\textstyle\sum_{i=0}^{15} 2^{2i} f_{2i}$$

$$\text{where } f_{2i}=-2\,a_{2i+1}+a_{2i}+a_{2i-1} \quad \ldots\ldots\ldots\ldots(1)$$

Unsigned number:

2s complement representation of A:

$$
\begin{aligned}
&=-2^{32}a_{32}+2^{31}a_{31}+2^{31}a_{30}\\
&\quad-2^{30}a_{30}+2^{29}a_{29}+2^{29}a_{28}\\
&\quad-2^{28}a_{28}+2^{27}a_{27}+2^{27}a_{26}\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad-2^{2}a_{2}+2^{1}a_{1}+2^{1}a_{0}\\
&\quad-2^{0}a_{0}+2^{-1}a_{-1}+2^{-1}a_{-2} \quad \text{where } a_{-1}=a_{-2}\equiv 0
\end{aligned}
$$

$$
\begin{aligned}
&=2^{31}(-2\,a_{32}+a_{31}+a_{30})\\
&\quad+2^{29}(-2\,a_{30}+a_{29}+a_{28})\\
&\quad+2^{27}(-2\,a_{28}+a_{27}+a_{26})\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&\quad+2^{1}(-2\,a_{2}+a_{1}+a_{0})\\
&\quad+2^{-1}(-2\,a_{0}+a_{-1}+a_{-2}) \quad \text{where } a_{-1}=a_{-2}\equiv 0
\end{aligned}
$$

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

$$=\textstyle\sum_{i=0}^{15} 2^{2i-1} f_{2i-1}$$

$$\text{where } \; f_{2i-1}=-2\,a_{2i}+a_{2i-1}+a_{2i-2} \qquad \ldots\ldots\ldots\ldots(2)$$

When equations (1) and (2) are compared, it is evident that preprocessing of integer A can be done before recoding the basic signed and unsigned integer.

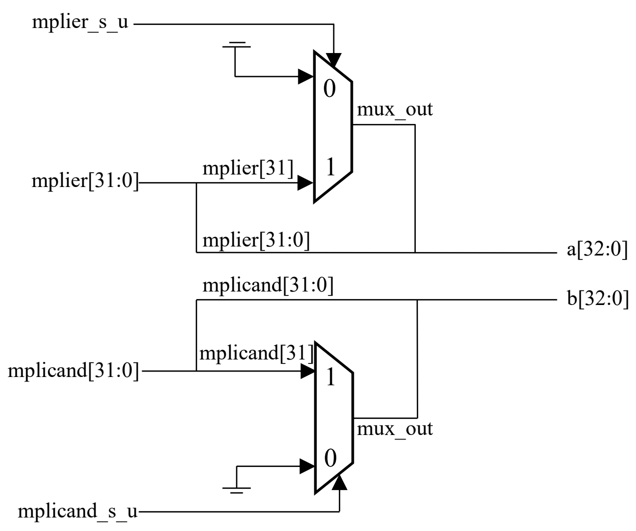
Table 2:Truth Table for F-Block Values

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

C. Sub-modules of the Booth Recoding Multiplier

1) 33rd bit extender unit

Here, a signed multiplier is being implemented to performed unsigned operations. Thus, in order to cover the complete range of unsigned multiplication, 1 extra bit is being extended at MSB in the next step. The 33rd bit for unsigned and signed number is



zero and sign-extended respectively as shown in Fig.2 below.

Figure 2:33rd-bit extender unit block diagram

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

2) Preprocessing/F block formation unit

- In this unit, a "0" is added at the least significant bit (LSB) of the "a"(Here, a is the 32-bit input to pre-processing block which is also the output of the 33rd bit extender unit)
- Following that, blocks of 3 bits each are made. The most significant bit (MSB) of the previous block is made the least significant bit (LSB) of current one.
- As a 33-bit number "a" is in consideration, so after addition of extra bit of "0" in the least significant bit (LSB) position, we have a shortage of one bit needed to form the F-block.
- To counter this, a bit is again added to the most significant bit (MSB), which is the same as the 33rd bit, irrespective of whether the number is signed or unsigned.

Table 3:Values after extension

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

Total width of the number after pre-processing becomes 34-bits. Pre-processing is just a simple rewiring process.

Table 4:Grouping for F-Block

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	$\{a_{32}, a_{32}, a_{31}\}$

3) Partial product generation unit

There will be 16 rows of partial products of the 16 F-blocks that

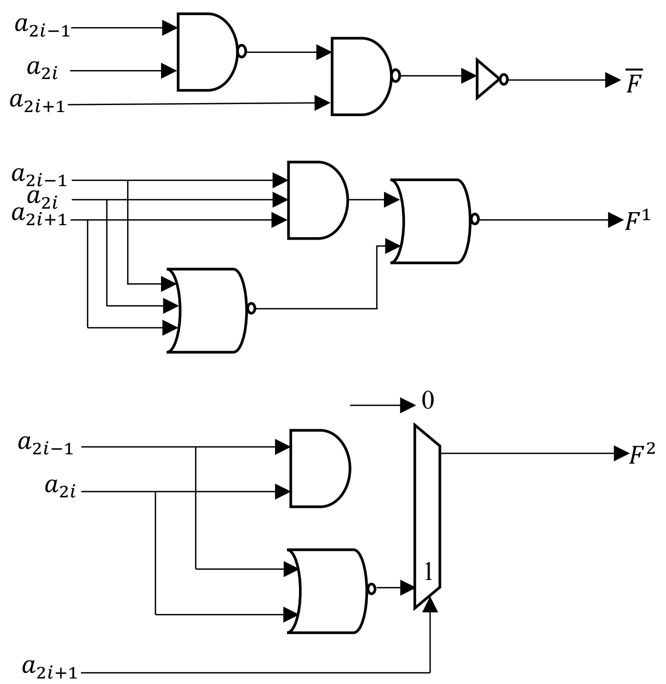


Figure 3:Partial Product Generator

have been generated. Using Table 4, $\overline{F}$, F^1 and F^2 have been obtained and will be used to do the necessary bit manipulation to get the required partial products. The hardware realization for $\overline{F}$, F^1 and F^2 is given above(Fig.3).

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

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

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

All the F-blocks need to be processed individually to find the respective $\overline{F}$, F^1 and F^2.

Using booth recoding algorithm, our partial products will be reduced. There will only be 17 rows of partial product to be added. For example, to generate ROW_0 partial product, we will need to operate $\overline{F0}$, $F0^1$ and $F0^2$ in "b"(33-bit multiplicand; output of preprocessing unit).Similarly, other rows are operated upon to get the respective partial product rows.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

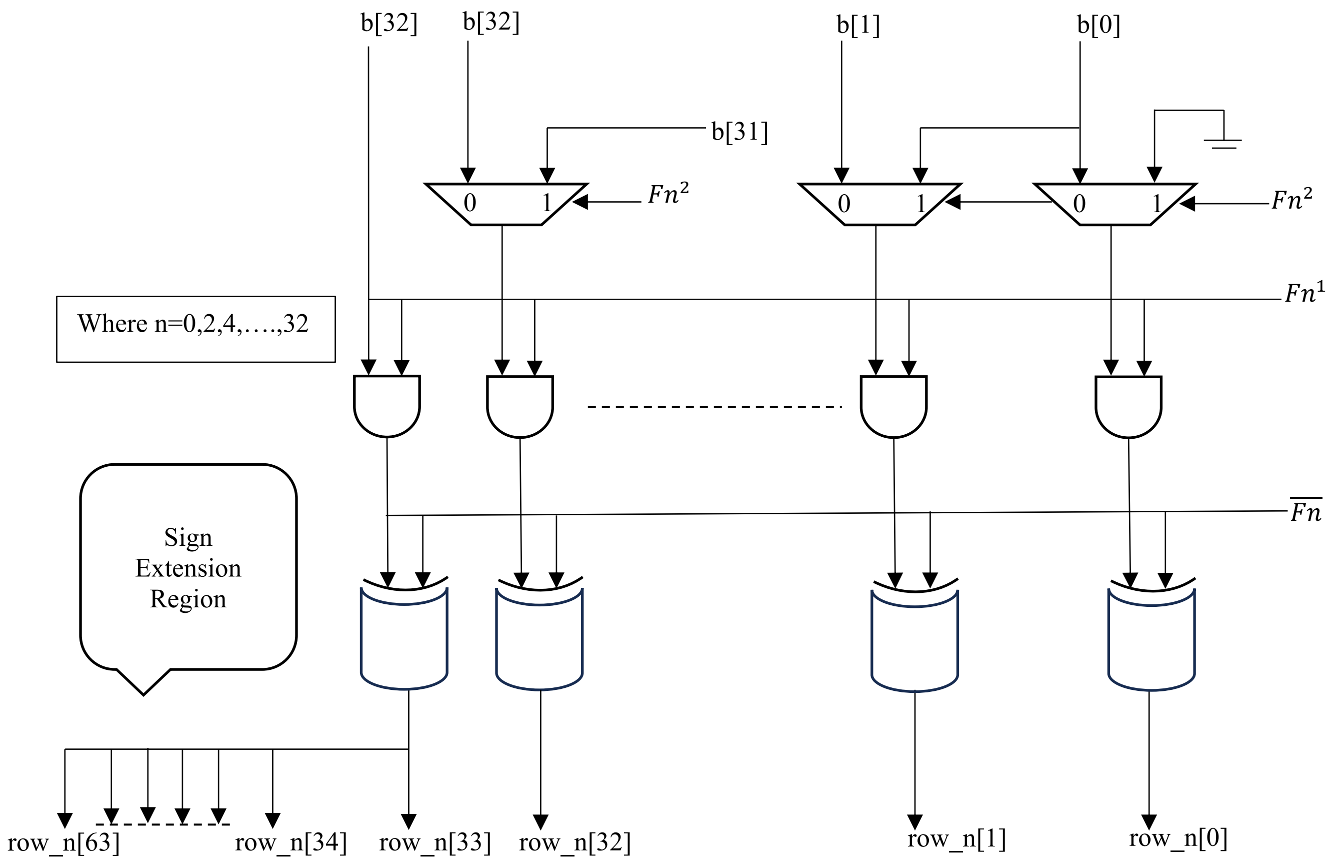


Figure 4:Row#n calculation using Fn^2, Fn^1 and $\overline{\mathrm{Fn}}$

4) Partial product addition unit

This unit undertakes the summation of all partial products, denoted as ROW#0 through ROW#32. Carries are efficiently propagated diagonally leftward and downward to enhance computational speed. However, during the processing of the final row, ROW#32, which lacks a row beneath it, horizontal carry propagation becomes necessary.

It is imperative to acknowledge that horizontal carry propagation is viable due to the unoccupied augend, as evidenced by the absence of a subsequent row. Notably, in cases where the value of f2i is negative, the computation necessitates the derivation of the 2's complement of the number "b". Despite the depiction in Fig.4, the operational scope is confined to XOR operations. Furthermore, the addition of 1 to the Least Significant Bit (LSB) position, in alignment with the binary values of the respective ROWs, is indispensable.

To address this requirement, an additional ROW, denoted as ROW#-1, is introduced. This supplementary row serves the purpose of facilitating the addition of 1 to the aligned LSB position for the relevant ROWs, ensuring computational accuracy and integrity within the system architecture.

Conventional approaches in multiplier design have often relied on the utilization of half and full adders. However, our approach diverges from tradition as we opt for the integration of high-speed agile compressors within our multiplier architecture. Critical path in conventional addition is longer compared to compressors. We use this fact and our strategic choice of utilising compressors over traditional adders aims to further augment computational speed and efficiency, thereby enhancing the overall performance of the multiplier.

III. COMPRESSORS

Binary multiplication involves the execution of numerous partial product additions. In conventional multiplier architectures, full adders are conventionally employed for this purpose. However, full adders can only accommodate the addition of up to three inputs simultaneously. Consequently, when adding all partial products, a substantial number of full adders are necessitated.

To mitigate the requirement for a vast number of adders in this operational scheme, we introduce high-speed compressors. These compressors are developed based on the principle of binary counter properties. Among the various compressor designs, the 5-3 compressor stands out, as it allows the addition of up to five inputs simultaneously, yielding a three-bit output. This innovation serves to streamline the computational process and optimize resource utilization within the multiplier architecture. Table-5 showcases the counter property of 5-3 compressor.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

Table 5:Counter-Property of a 5:3 Compressor

Input Condition	Out3	Out2	Out1	Decimal value
All inputs are zero	0	0	0	0
Any one input is one	0	0	1	1
Any two input are one	0	1	0	2
Any three input are one	0	1	1	3
Any four input are one	1	0	0	4
Any five input are one	1	0	1	5

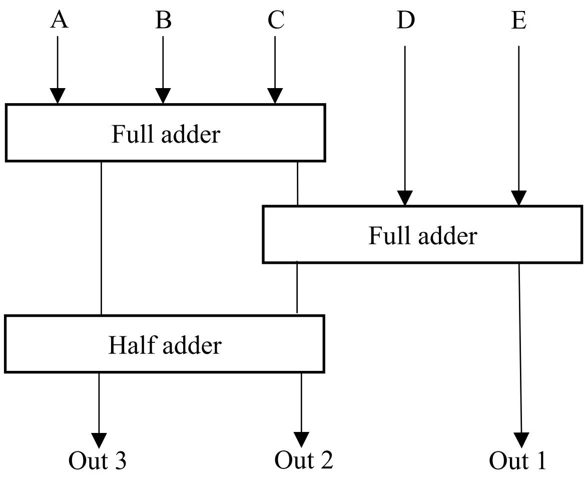


Figure 6:Structure of a 5:3 Compressor

Different compressors:-their design and sub-units

Some basic different compressors' architecture are designed and implemented by using the same logic. Their block diagrams are as follows

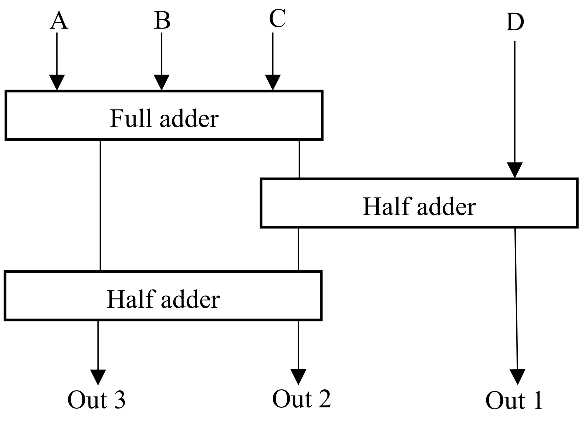


Figure 5:Structure of a 4:3 compressor

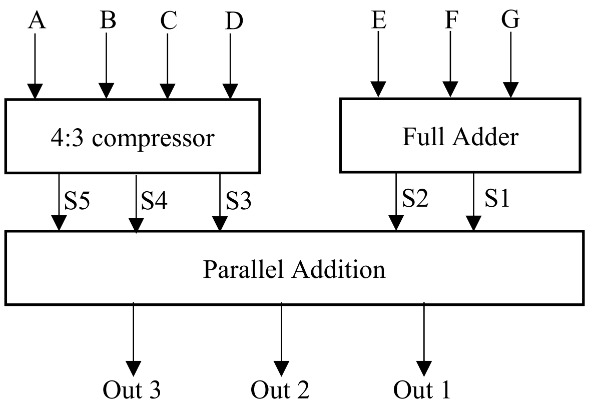


Figure 7:Structure of a 7:3 Compressor

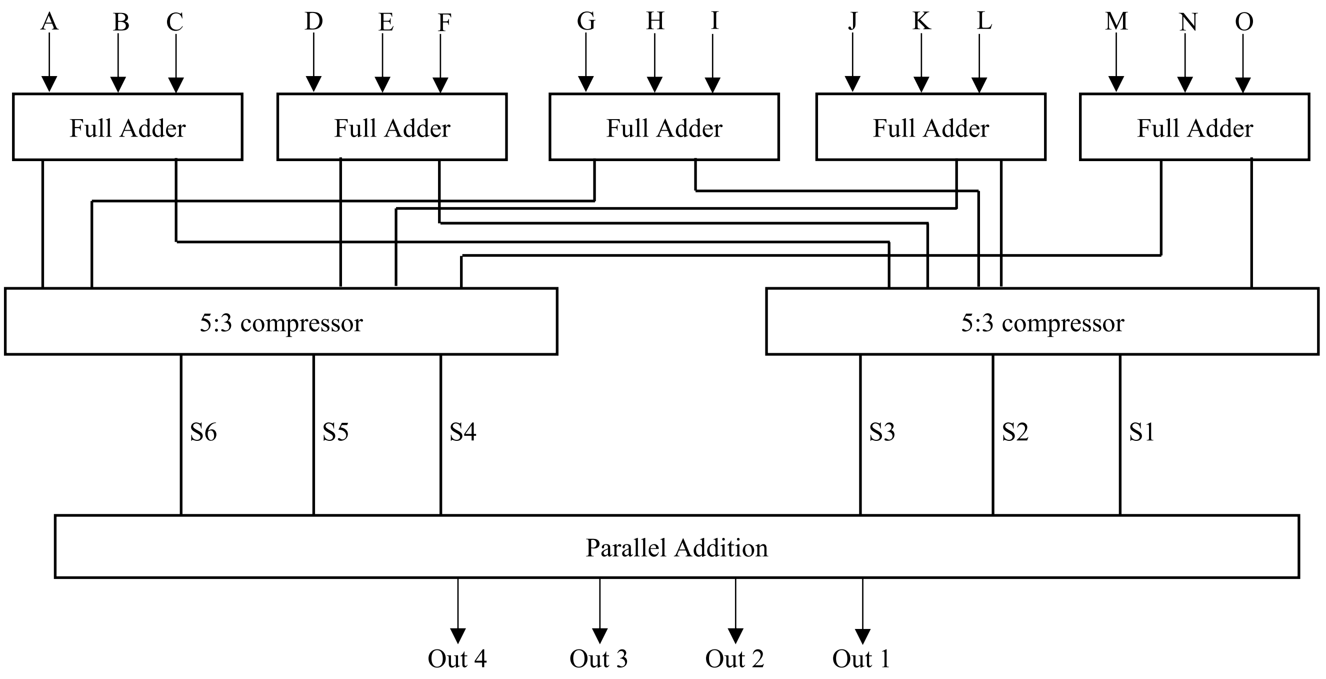


Figure 8:Structure of a 15:4 Compressor

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

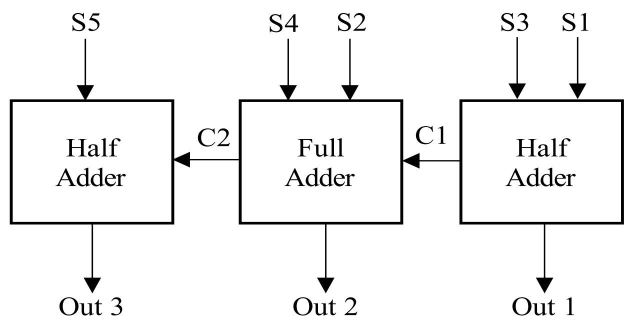


Figure 9:Parallel Addition Unit for 7:3 compressor

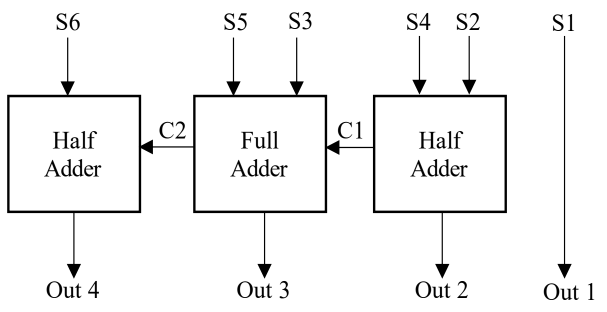


Figure 10:Parallel Addition Unit for15:4 compressor

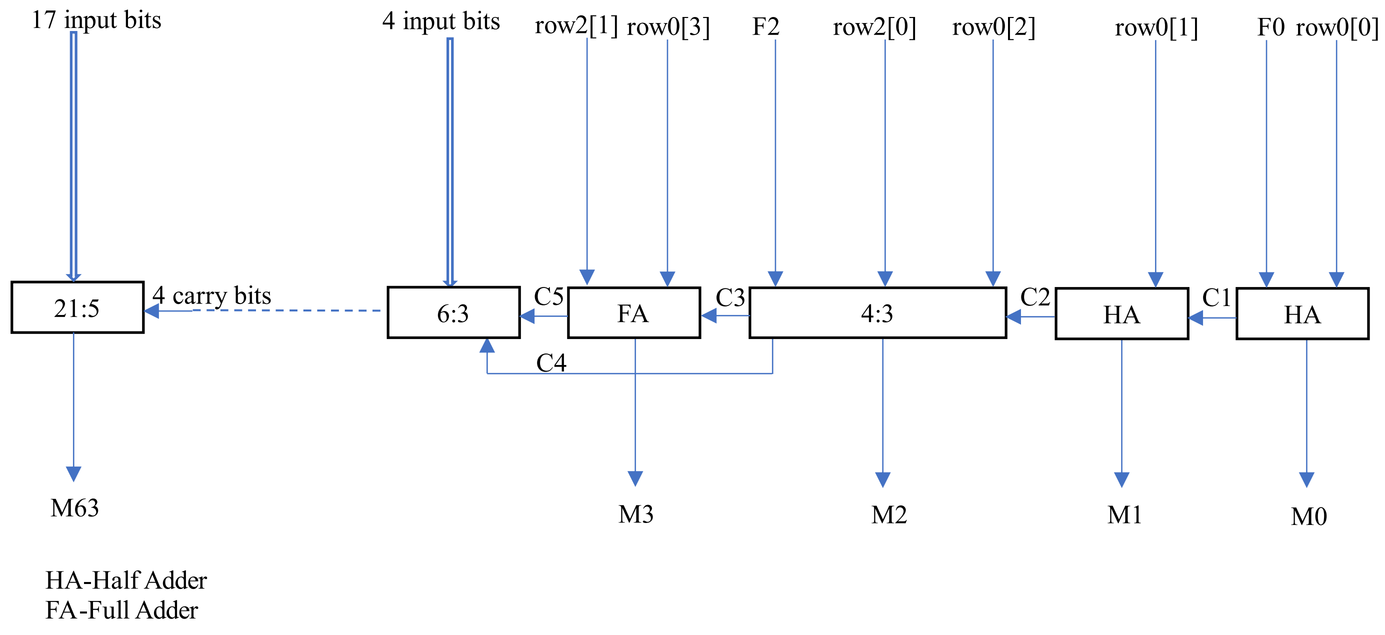


Figure 11:Partial Product Addition Using High-speed Agile Compressor

IV. RESULTS AND COMPARISON

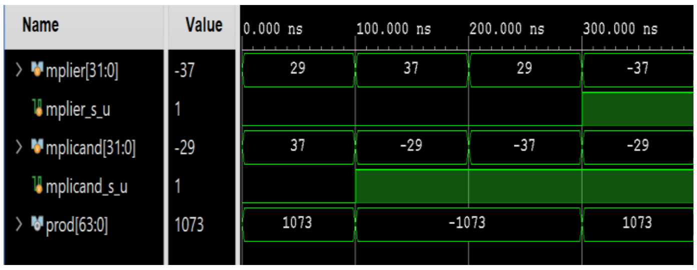
The implemented single cycled, 32-bit signed multiplier using booth recoding algorithm and high speed compressors has been successfully verified through a written testbench script. The code was written and executed on the Xilinx Vivado 2021.2 platform. The synthesized design was targeted towards the Kintex-7 FPGA, specifically the Device- XC7K70T with Package- FBG484 and speed-1. Table-6 presents a comparison between the implemented single-cycle 32-bit signed multiplier and the 32-bit complex Vedic multiplier [2].

Table 6:A Comparison with The Vedic Multiplier

Parameters	32-bit Vedic Multiplier	Implemented 32-bit Signed Multiplier	Improvement
Slice LUTs	7874	1978	74.88%
Bonded IOBs	258	130	49.61%

Table 7:FPGA Hardware Utilization

Parameter		Utilization
Bonded IOB		130
Slice	SLICEL	310
	SLICEM	243
LUT as Logic	Using o6 output only	1713
	Using o5 and o6	264



```
Name              Value   0.000 ns   100.000 ns   200.000 ns   300.000 ns
mplier[31:0]      -37     29         37           29           -37
mplier_s_u        1
mplicand[31:0]    -29     37         -29          -37          -29
mplicand_s_u      1
prod[63:0]        1073    1073       -1073                     1073
```

Figure 12:Behavioural Simulation Results

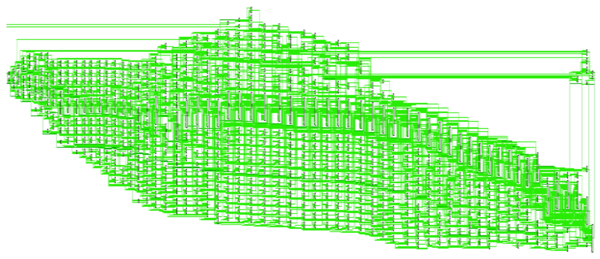


Figure 13:RTL Schematic of implemented multiplier

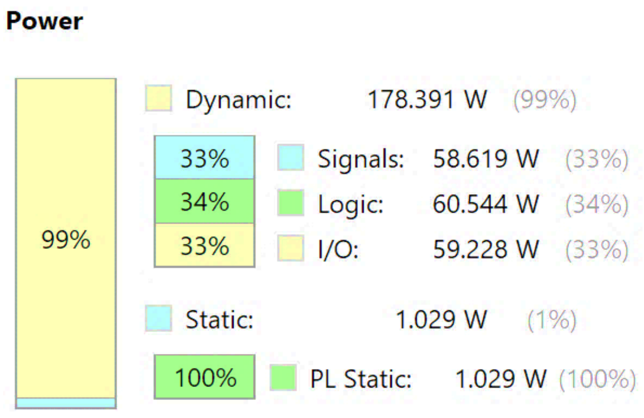


Figure 14:On-Chip Power

V. CONCLUSION

The architecture of the multiplier proposed by us in this paper can multiply two signed as well as unsigned 32-bit integers using booth recoding algorithm and high speed agile compressors. The booth recoding algorithm significantly reduces the number of partial product rows which in turn improve the speed as compared to conventional multipliers and the high speed compressors have a much shorter critical path as compared to traditional adders. The implemented multiplier's performance has been examined 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 74.88% 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] Shubhajit Roy Chowdhury, Aritra Banerjee, Aniruddha Roy, Hiranmay Saha,”Design, Simulation and Testing of a High Speed Low Power 15-4 Compressor for High Speed Multiplication Applications”, First International Conference on Emerging Trends in Engineering and Technology, 2008.

[5] M. Morris Mano, “Digital Design”,3rd edition, Prentice Hall,2002..