ISSN:0975-9646

An Optimized Design for Parallel MAC based on Radix-4 MBA

R.M.N.M.Varaprasad , M.Satyanarayana

Dept. of ECE, MVGR College of Engineering,
Andhra Pradesh, India

***Abstract*— In this paper a novel architecture of multiplier and accumulator (MAC) for high speed arithmetic is presented. The architecture adopts radix-4 modified booth algorithm (MBA) and hybrid carry save adder, in which the accumulator that has the largest delay in MAC was merged into Carry save adder (CSA) block. The performance of final adder block, which determines critical path of the architecture, is improved by reducing number of input bits of the final adder itself. Moreover the design accumulates the intermediate results in the type of sum and carry bits instead of the output of the final adder, which made it possible to optimize the pipeline scheme. Using this architecture the overall performance can be elevated twice that of previous architectures. The proposed design was coded in verilog HDL and simulated using Xilinx ISE tool. FPGA Spartan 3E starter kit was used for implementation of design.**

***Keywords*— Carry look ahead adder (CLA), Carry save adder (CSA), Multiplier and accumulator (MAC), Modified booth algorithm (MBA), Partial product.**

I. INTRODUCTION

Multiplication can be considered as a series of repeated addition operations. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. The multiplication operation is generally performed by multiplying each term in multiplier with whole multiplicand, thus generating a partial product and final summing all the partial products to obtain the result. This repeated method is rather slow that it is almost always replaced by an algorithm.

It is possible to decompose multipliers into two steps. The first step is dedicated to the generation of partial products, and the second one collects and adds them. The speed of the multiplication and addition determines the execution speed and performance of the entire calculation. Many of the Digital signal processing (DSP) applications are accomplished by repetitive multiplication and addition operations. Therefore multiplier-and-accumulator (MAC) unit is the essential element of the digital signal processor. In order to increase the speed of a multiplier, the number of the partial products generated must be reduced. If N-bit data are multiplied, the number of the generated partial products is proportional to N, thus the execution time. The accumulation operation has the largest delay in MAC. Therefore in-order to enhance performance of MAC, an architecture that uses modified Booth algorithm and hybrid carry save adder is proposed.

This paper is organized as follows. In Section II, a simple introduction of MAC will be given, and the architecture for the proposed design will be described in Section III. In Section IV, Simulation result will be analyzed. Finally, the conclusion will be given in Section V.

II. MAC UNIT

In this section, a brief description of MAC unit and its operation is introduced. In general, MAC unit consists of multiplier and an adder. Multiplier performs multiplication operation between multiplicand and multiplier where as adder adds the multiplier result to the contents of accumulator. This process of multiplication and accumulation continues to operate until generation of final result, that itself stored in the accumulator. The number of clock cycles required for the operation depends on the number of input bits fed to the MAC and the speed of the operation depends on the number of partial products generated during the operation.

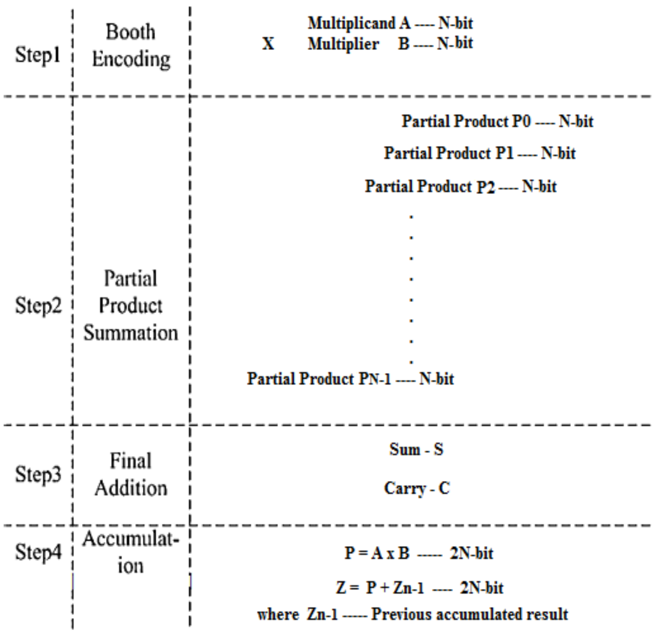

Step1
Booth Encoding
X
Multiplicand A ---- N-bit
Multiplier B ---- N-bit
Step2
Partial Product Summation
Partial Product P0 ---- N-bit
Partial Product P1 ---- N-bit
Partial Product P2 ---- N-bit
Partial Product PN-1 ---- N-bit
Step3
Final Addition
Sum - S
Carry - C
Step4
Accumulat-ion
P = A x B ---- 2N-bit
Z = P + Zn-1 ---- 2N-bit
where Zn-1 ---- Previous accumulated result

Fig. 1 Basic steps in MAC operation

Multiplication and accumulation operation can be divided into four operational steps as shown in Fig.1. The first is Booth encoding in which partial products are generated from the multiplicand *A* and the multiplier *B* by applying algorithms. Since speed of operation depends on number of Partial products generated, booth encoding should be capable of reducing partial product count effectively. The second is partial product summation which includes addition of all partial products. The next steps include the final addition and accumulation operations, which includes the process of accumulating the multiplied results. Multiplication and accumulation operation is done by multiplying the inputs, multiplier *B* and the multiplicand *A*. The obtained multiplication result *P* is added to the previous accumulation content *Zn-1* as the accumulation step. Final result *Z* of the operation will be stored in accumulator. Hardware architecture of MAC is shown in Fig. 2.

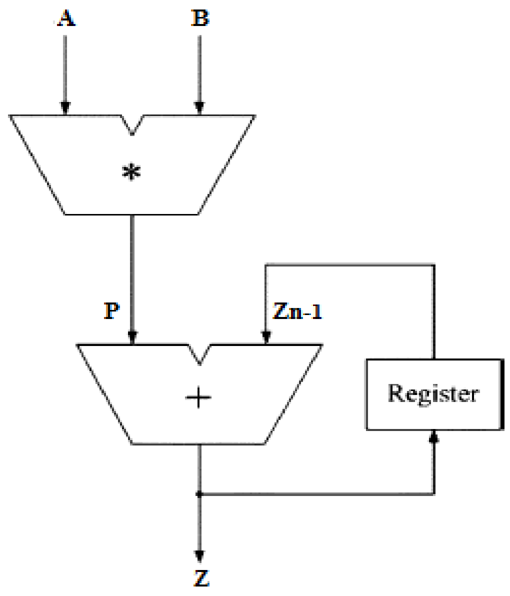


Fig. 2 General Hardware architecture

A. *Representing in terms of Equations*

The N-bit 2's complement binary number can be expressed as

$$A = -2^{N-1}a_{N-1} + \sum_{i=0}^{N-2} a_i 2^i, \quad a_i \in 0,1. \qquad (1)$$

Eq. (1) can written as

$$A = \sum_{i=0}^{N/2-1} d_i 4_i \qquad (2)$$

where

$$d_i = -2a_{2i+1} + a_{2i} + a_{2i-1} \qquad (3)$$

Similarly

$$B = \sum_{i=0}^{N/2-1} d_i 4_i \qquad (4)$$

where

$$d_i = -2b_{2i+1} + b_{2i} + b_{2i-1} \qquad (5)$$

Using above equations multiplication operation can be expressed as

$$P = A \times B = \sum_{i=0}^{N/2-1} d_i 2^{2i} B \qquad (6)$$

Therefore multiplication – accumulation result can be expressed as

$$Z = A \times B + Z_{n-1} = \sum_{i=0}^{N/2-1} d_i 2^{2i} B + \sum_{j=0}^{2N-1} z_j 2^j$$

$$¥\, i, j € 0 \text{ to } N \qquad (7)$$

III. Proposed Design

In this section, brief description of proposed design will be discussed. The proposed design uses modified booth algorithm for booth encoding. If two N-bit numbers are multiplied and accumulated, the result generated is of 2N-bit number and the critical path is determined by the accumulation operation. Therefore the accumulator which has the largest delay limits the performance of MAC. Even though pipeline scheme is applied, the delay of the last accumulator affects the performance of the MAC.

Therefore performance of MAC is improved by eliminating the accumulator itself and combining it with the CSA function. The critical path of the architecture which depends on accumulator is now determined by the final adder in the multiplier. In order to improve the performance of the final adder the number of input bits fed to it should be reduced. To reduce this number of input bits, the multiple partial products are compressed into a sum and a carry by CSA.

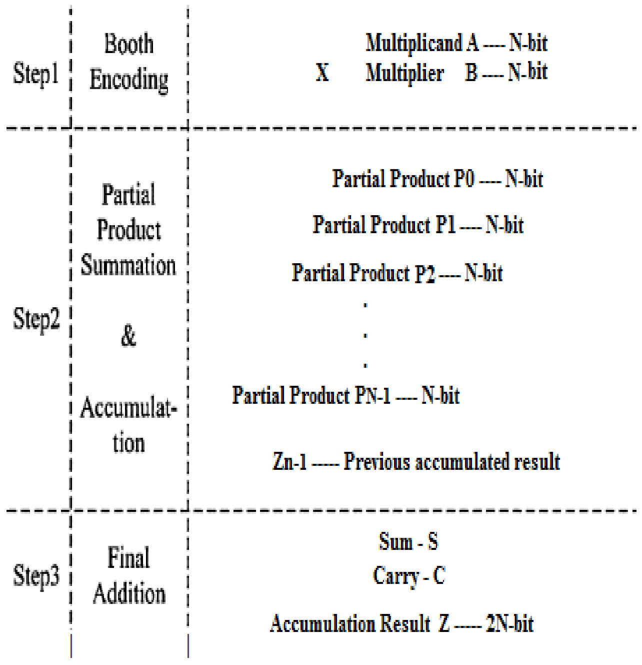

Step1
Booth Encoding
Multiplicand A — N-bit
X Multiplier B — N-bit
Step2
Partial Product Summation & Accumulat-tion
Partial Product P0 — N-bit
Partial Product P1 — N-bit
Partial Product P2 — N-bit
Partial Product PN-1 — N-bit
Zn-1 — Previous accumulated result
Step3
Final Addition
Sum - S
Carry - C
Accumulation Result Z — 2N-bit

Fig. 3 Proposed MAC operation

The MAC process steps presented in the previous section are rearranged, as shown in Fig.3, in which the MAC operation is organized into three steps. In this figure, the accumulation step has been merged into the process of adding the partial products and the final addition process in step 3 is not always run. Since accumulation is carried out using the result from step 2 instead of that from step 3

A. *Radix-4 MBA*

The algorithm used here is Modified Booth's algorithm (MBA) which approximately twice as fast as Booth's algorithm. The modified Booth algorithm reduces the number of partial products by half in the first step, thus enhances performance of the design. Radix-4 Modified Booth Algorithm is used for the proposed design, since it offers more ease of implementation for higher order bits. The algorithm involves shift and complement operations with only one final addition operation. In order to multiply *A* by *B* using the MBA, the algorithm starts from grouping Multiplier *B* by three bits (with one bit overlapping in each pair) and encoding into partial product scale factors {-2, -1, 0, 1, 2}. The recoding table for the algorithm is shown in Table 1. Each row of table indicates a partial product scale factor and an operation to be performed on multiplicand *A* to generate partial product. For example '0XA' indicates multiplication of multiplicand *A* with zero (simply replacing with zeros), '1XA' indicates shift operation of multiplicand *A* and '2XA' indicates double shift operations of multiplicand *A*, where as negation indicates shift operation to be performed on 2's complement of the multiplicand *A* .

TABLE I
Radix-4 Recoding table

X_{i+1}	X_i	X_{i-1}	**Action**
0	0	0	0 × A
0	0	1	1 × A
0	1	0	1 × A
0	1	1	2 × A
1	0	0	-2 × A
1	0	1	-1 × A
1	1	0	-1 × A
1	1	1	0 × A

B. *CSA and CLA*

The idea behind using CSA is to reduce delay further. The concept of CSA is to add three numbers together, x + y + z, and convert it into 2 numbers c + s such that x + y + z = c + s, and do this in O(1) time. The reason why addition cannot be performed in O(1) time is because the carry information must be propagated. In CSA, carry information can be passed directly, until the very last step, unlike, normal addition, where three numbers are aligned and then preceded column by column addition. The three digits in a row are added, and any overflow goes into the next column. The number of bits of sums and carries to be transferred to the final adder is reduced by adding the lower bits of sums and carries in advance within the range in which the overall performance will not be degraded.

A 2-bit CLA is used to add the lower bits in the CSA. A carry-look ahead adder improves speed by reducing the amount of time required to determine carry bits. Generally adders such as, ripple carry adder ,the carry bit is calculated alongside the sum bit, and each bit must wait until the previous carry has been calculated to begin calculating its own result and carry bits. The carry-look ahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger value bits.

C. *Hardware Architecture*

The hardware architecture of the proposed design is shown in Fig. 4. The N –bit MAC inputs, *A* and *B*, are converted into an (N+1) -bit partial products by passing through the Booth encoder. At most (N/2+1) partial products are generated. In the CSA and accumulator, accumulation is carried out along with the addition of the partial products. As a result, N -bit Sum *S*, Carry *C* and *Z* [N-1: 0] bits are generated.

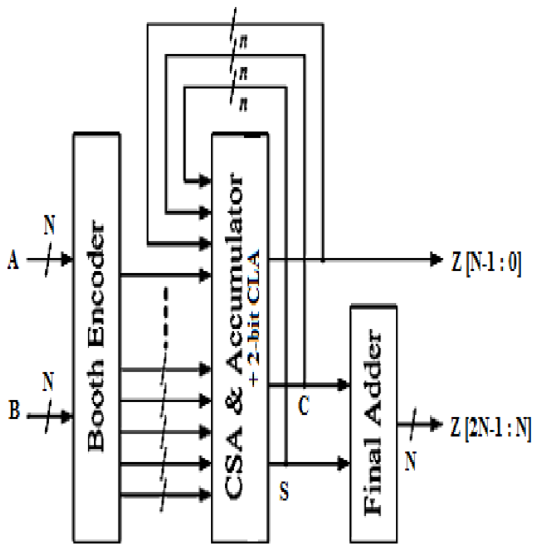


Fig.4 Hardware architecture for the Proposed MAC

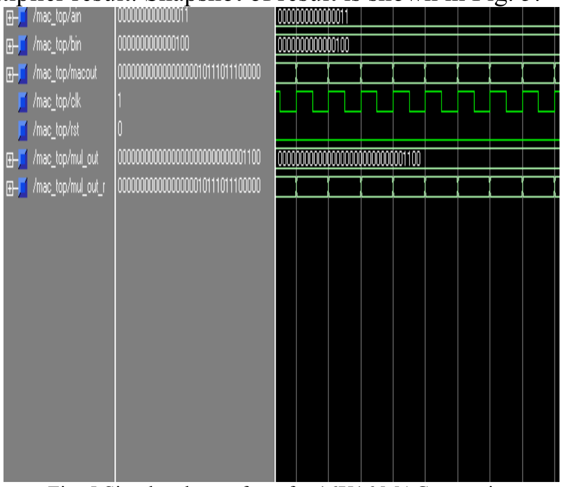
These values are fed back and used for the next accumulation. The final result consists of higher order bits *Z* [2N-1: N] that are generated by adding Sum *S* and Carry *C* in the final adder and lower order bits *Z* [N-1: 0] that are already generated. This way of accumulating the sum and carry bits from the CSA instead of the output bits from the final adder, in the manner that the sum and carry bits from the CSA in the previous cycle are inputted to CSA, increases the output rate when pipelining is applied. Due to this feedback of both sum and carry, the number of inputs to CSA increases, compared to the standard design steps in Fig.1.

D. *FPGA*

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer. The FPGA configuration is generally specified using a hardware description language (HDL), can be used to implement any logical function and has the ability to update the functionality, partial re-configuration of the design and involves low non-recurring engineering cost. The most common FPGA architecture consists of an array of programmable logic components called logic blocks, I/O pads, and a hierarchy of reconfigurable interconnects that allow the blocks to be wired together. Logic blocks can be configured to perform complex combinational functions. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs/LABs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

IV. RESULTS AND DISCUSSION

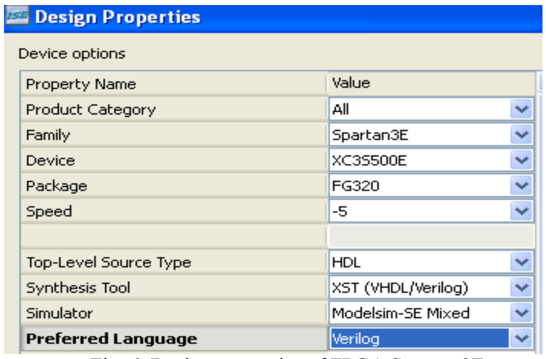
The proposed architecture is defined in verilog HDL and simulated using Xilinx ISE tool. Values are taken in a 16-bit multiplicand (a_{in}) and multiplier (b_{in}) operands. A 32 –bit MAC out operand is defined which displays the result. A 32-bit Mul-out operand is also defined which displays multiplier result. Snapshot of result is shown in Fig. 5.



```
/mac_top/ain        0000000000000011
/mac_top/bin        0000000000000100
/mac_top/macout     00000000000000001011101110000
/mac_top/clk        1
/mac_top/rst        0
/mac_top/mul_out    00000000000000000000000000001100
/mac_top/mul_out_r  00000000000000001011101110000
```

Fig. 5 Simulated waveform for 16X16 MAC operation

The Code is synthesized using Xilinx XST tool and implemented using FPGA Spartan 3E starter kit. The device properties are shown in Fig. 6. The Design summary and Performance summary is as shown Table 2 and Table 3 respectively. Xilinx X-power tool is used for approximate power estimation and analysis Table 4 and Table 5 gives approximate power analysis summary.



Design Properties

Device options

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-5
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE Mixed
Preferred Language	Verilog

Fig. 6. Design properties of FPGA Spartan 3E

TABLE II
Device Utilization summary

Logic utilization	Used	Available	Utilization (%)
Number of slice flip flops	42	3,840	1%
Number of 4 input LUTs	40	3,840	1%
Number of occupied Slices	30	1,920	1%
Number of Slices containing only related logic	30	30	100%
Number of Slices containing unrelated logic	0	30	0%
Total Number of 4 input LUTs	42	3,840	1%
Number used as logic	40	…..	…..
Number used as a route-thru	2	…..	…..
Clocks	1	……	……
Number of bonded IOBs	5	173	2%
Number of BUFG MUXs	1	8	12%
Average Fanout of Non-Clock Nets	3.26	…..	......

TABLE III
Performance summary

Final Timing Score:	0 (Setup: 0, Hold: 0)
Routing Results:	All Signals Completely Routed
Timing Constraints:	All Constraints Met

TABLE IV
Power and Temperature analysis

Parameter	Value
Total quiescent power	0.04098(w)
Total Dynamic power	0.00000(w)
Total power	0.04098(w)
Junction temperature	26.3^{o}C
Effective ThetaJA (^{o}C/w)	30.9
Max Ambient (^{o}C)	83.7

TABLE V
Supply voltage summary

Parameter	Power (w)	Voltage	Range	Iccq (A)
Vcc int	0.01223	1.200	1.140 to 1.260	0.01020
Vcc aux	0.02500	2.500	2.375 to 2.625	0.01000
Vcco 25	0.00375	2.500	2.375 to 2.625	0.00150

V. CONCLUSION

A 16X16 multiplier-accumulator (MAC) is presented in this work. Radix-4 Modified Booth multiplier circuit is used for MAC architecture. Compared to other circuits, this architecture has the highest operational speed and less hardware count. By removing the independent accumulation process that has the largest delay and merging it to the compression process of the partial products, the overall MAC performance has been improved almost twice as much as in the previous work.

REFERENCES

[1] Cooper A. R., "Parallel architecture modified Booth multiplier," *Proc.Inst. Electr. Eng. G*, vol. 135, pp. 125–128, 1988.

[2] Fadavi-Ardekani.J, "MXN Booth encoded multiplier generator using optimized Wallace trees," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 1, no. 2, pp. 120–125, Jun. 1993.

[3] Rajendra.K "A modified booth algorithm for high radix fixed point multiplication."*IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 2, no. 4, pp. 522–524, Dec. 1994.

[4] Shanbag.N.R and Juneja.P, "Parallel implementation of a 4X4-bit multiplier using modified Booth's algorithm," *IEEE J. Solid State Circuits*, vol. 23, no. 4, pp. 1010–1013, Aug. 1988.

[5] Wallace.C.S, "A suggestion for a fast multiplier," *IEEE Trans. Electron Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.

[6] Young-Ho Seo, Dong-Wook Kim "A New VLSI Architecture of Parallel Multiplier–Accumulator Based on Radix-2 Modified Booth Algorithm", *IEEE transactions on VLSI* systems, Vol 28 (2010).