Energy Efficient Low-Latency Signed Multiplier for FPGA Based Hardware Accelerators

Smt. Y. Bhargavi[1], SD.Shahin[2], Y.Deedepya[3], S.Upendra[4], SK.Samreen[5]

[1]Assistant Professor, [2,3,4,5]Student

Electronics and Communication Engineering

N.B.K.R Institute of Science And Technology, Andhra Pradesh , India

Abstract:

In many applications, including multimedia processing and artificial neural networks, multiplication is one of the most commonly used mathematical operations. The multiplier is one of the main causes of energy consumption, critical path latency and resource utilization in such applications. In field programmable gate array (FPGA) based designs, these effects become even more apparent. However, most state-of-the-art designs are for ASIC-based systems. Additionally, the few FPGA-based designs in use today are primarily limited to unsigned integers and require additional circuitry to accept signed operations. This paper provides an area-optimized, low-latency and energy-efficient architecture for an exact signed multiplier to overcome these limitations in FPGA-based applications using signed integers. Our devices reduce area, latency, and energy by up to 40, 43.0, and 70, respectively, compared to Vivado's region-optimized IP factor. An open source library containing RTL implementations of our concepts is available at https://cfaed.tu-dresden.de/pd-downloads.

Keywords: Multiplier, FPGA, Hardware Accelerators, Verilog ,Xilinx Radix-4…..

I.I NTRODUCTION

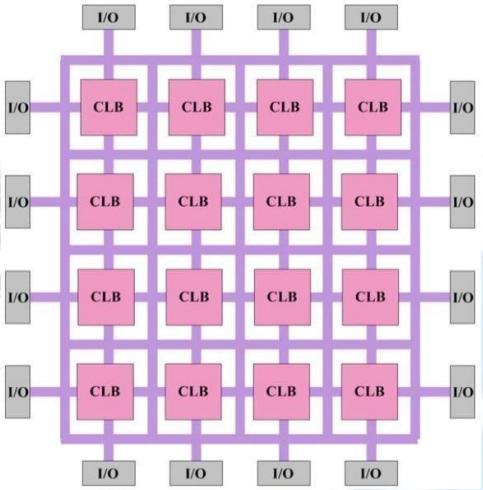
The overall efficiency, resource usage and power consumption of such applications are directly affected by the chosen multiplier architecture and its implementation. Synthesis tools use lookup tables instead of DSP blocks as multipliers for these low-precision integers. It is more useful to have a small-area, efficient and energy-saving LUT-based multiplier option in addition to DSP blocks. For Xilinx FPGAs, we presented inefficient radix-4 multiplier implementations using Booth technology. In their later work, underpowered multipliers were implemented using the multiplier algorithms developed by Booth and Baugh-Wooley. Such methods work well only for modestly large bit width coefficients; higher bit width multipliers require more FPGA resources. Using default Xilinx Vivado synthesis settings, the Virtex-7 FPGA precision 88 multiplier log-based implementation uses 71 LUTs.

II. FPGA

Field Programmable Gate Arrays (FPGAs) are highly flexible, reprogrammable logic devices that leverage advanced CMOS manufacturing technologies, similar to other industry-leading processors and processor peripherals.Like processors and peripherals, FPGAs are fully user programmable.For FPGAs, the program is called a configuration bit stream, which defines the FPGA's

A field-programmable gate array (FPGA)is an integrated circuit designed to be configured by the customer or designer after manufacturing hence "field- programmable". The FPGA configuration is generallyspecified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that anASIC could perform.

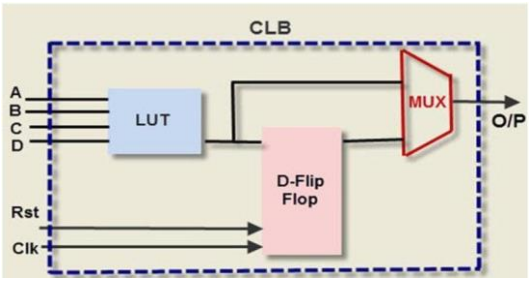
Block Diagram of FPGA



- **FPGA Logic Block**

The logic block contains a MUX (multiplexer), a D-flip-flop and a LUT. LUT implements logical combination functions; A MUX is used for the selection logic and a D-flip-flop stores the output of the LUT. The basic building block of an FPGA is a lookup table-based function generator. The number of LUT inputs varies between 3, 4, 6 and even 8 after the experiments. We now have adaptive LUTs that provide two outputs per LUT with two generator implementations. The Xilinx Virtex-7 is the most popular FPGA that includes a look-up table (LUT) and a flip-flop connected to a MUX. as discussed above. A current FPGA consists of approximately hundreds or thousands of configurable logic blocks. To configure the FPGA, Model sim and Xilinx ISE software are used to generateand develop bitstream files.

Block Diagram of Logic Block



III. PROPOSED APPROACH Two's complement coefficients are important for many applications. In thispaper, we present a method to reduce themaximum height of the partial producttable generated by the radix-4 ModifiedBooth Encoded multiplier by one rowwithout increasing the delay of thepartial product generation step. Thisreduction can allow faster packaging ofsome product groups and standardarrangements. This technique is ofparticular interest for all multiplierdesigns, but especially for short widthtwo's complement multipliers for high-performance embedded cores. Theproposed method is general and can beextended to encodings with a larger radius and square and m_n rectangular coefficients of any size. We evaluate theproposed approach against some otherpossible solutions; results based onrough theoretical analysis and logicalsynthesis showed its effectiveness in terms of both range and delay.

We provide region-optimized, low- latency, and energy-efficient exact signed multipliers using the principles of Booth's radix-4 multiplication algorithm.In a multiplier based on bit encoding (BE), the sign (MSB) of the multiplier and the associated BE value determine the correct sign of the partial input. The required sign extensions (SE) are listed in Tables1(a) and 1(b) for each possible combination of BE values and multiplierMSB.

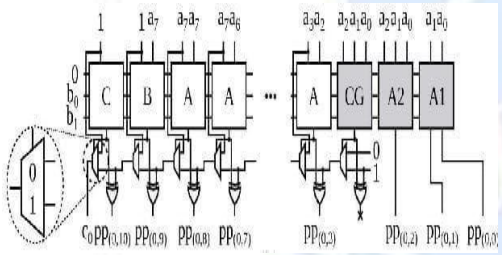
Table 1(a) : Booth Encoding

Inputs			Encoding	Output		
b_{m+1}	b_m	b_{m-1}	BE	s	c	z
0	0	0	0	0	0	1
0	0	1	1	0	0	0
0	1	0	1	0	0	0
0	1	1	2	1	0	0
1	0	0	$\overline{2}$	1	1	0
1	0	1	$\overline{1}$	0	1	0
1	1	0	$\overline{1}$	0	1	0
1	1	1	0	0	0	1

Table 1(b): Sign Extension

BE	MSB Multiplicand	SE
0	0	0
0	1	0
1	0	0
1	1	1
2	0	0
2	1	1
2	0	1
2	1	0
1	0	1
1	1	0

Block Diagram of Proposed Approach

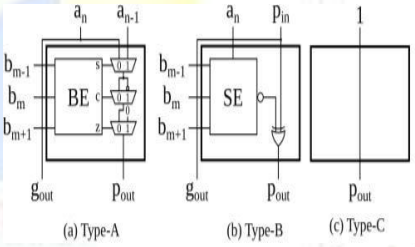


- **Generation of Partial Products**

The lut type-a configuration is used to implement the booth's encoding (a). The initial mux (controlled by the s signal) determines whether an or an-1 should be transmitted for partial product production based on the value of be.

Lastly, depending on the value of the z signal, the third mux can make the partial product zero. This data is sent as a carry propagate signal "P_{out}" To the connected carry chain. The input signal serves as the carry chain's "Gout" Signal generator. The bm 1, bm and bm-1 (from the multiplier), a (the msb of the multiplicand), and pin are the four inputs to the lut type-b. On the first row of partial products, the pin signal is fixed at 1, but for all subsequent rows, it is fixed at 0. The lut calculates the se signal, applies the xor operation to it, and sends the outcome as the carry propagate signal's "Pout" To the connected carry chain. The pin signal directly supplies the carry generate signal (gout). The right sign information from one partial product row is transferred to the next partial product row using lut type-c.displays the first row of partial products for an 8x8 multiplier using luts of types a, b, and c. The required input carry is calculated using the rightmost type-a lut in each partial product row. When representing a partial product in 2's complement format, this input carry is used. A total of four partial product rows will be produced for an 88 multiplier.

Partial Product Generation



- **Reduction in Critical Path Delay**

The transmission chain of each partial product row of the factor NxM is N 4 bits long. The transmission chain length can be reduced to N1 bits to increase the critical path delay of the multiplier.

Critical Path Delay Reduction

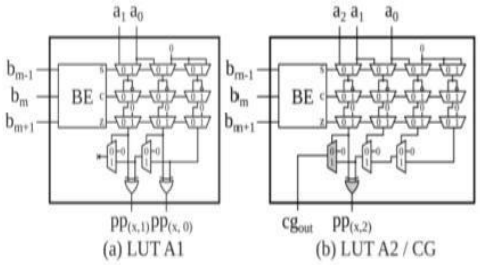


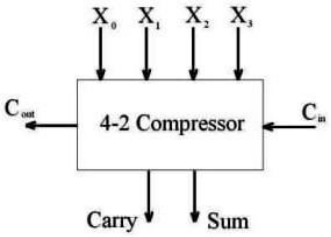
Figure shows a delay-optimized implementation of our unique critical path (b). Each partial input row has two coefficient bits required for the partial input terms pp(x, 0) and pp(x, 1). One 6-input LUT "A1" can realize both partial product expressions. As in the previous example, each pp(x, 2) of the partial output row can be realized independently using another 6-input LUT called "A2". A separate 6-input LUT called "CG" can be used to determine the correct input transfer for each partial product line. The internal configurations of LUT types A1, A2 and CG are shown in Figure 4. Only the output signals pp(x, 2) and cgout are different for LUT types A2 and CG. LUT type A2 uses only the pp(x, 2) signal, while LUT type CG uses only the cgout signal. with the factor NxM (N 3) x (M/2) - 1 LUT is required to produce byproducts.

- **Final Summation**

Binary adders, ternary adders and 4:2 compressors can be used to subtract the generated partial inputs to calculate the final product. A 4:2 compressor can turn two output lines into four partial lines. Our results show that the use of ternary components can reduce overall resource usage. But they are slower than binary adders on the critical path. Thus, the fractional yields obtained in our study are reduced by using 4:2 compressors and binary viewers. We implemented them using 6-input LUTs and associated transport circuits.

Block diagram of 4:2 compressor

-



IV. MODIFIED BOOTH'S ALGORITHM FOR RADIX-4

One solution to implementing fast multipliers is to increase parallelism, which helps reduce the number of subsequent computation steps. The original version of Booth's algorithm (Radix-2) had two shortcomings. They are:

1) When designing parallel multipliers, the number of operations of addition-subtraction and operations of transfer changes and becomes difficult.
2) The algorithm becomes inefficient when there is an isolated

Booth's algorithm for scanning three-bit strings is given below:

a) If necessary, extend the sign bit by 1 position to ensure that n pairs.
b) Attach the 0 multiplier to the right side of the LSB.

Depending on the value of each vector, each partial product is 0, M, -M, 2M or -2M. Negative values of B are obtained by 2's complement, and this article uses CLA (Carry-look-ahead) adders. Multiplication of M is done by shifting M to the left by one bit. Thus, in any case, when designing an n-bit parallel multiplier, only n/2 partial inputs are produced.

$$Z_n = -2xB_{m+1} + B_m + B_{m+1}$$

Where B is the Multiplier

Radix-4 Encoding

Xn	X_{n+1}	X_{n-1}	Recoded bits	Operations performed
0	0	0	0	0
0	0	1	+1	+M
0	1	0	+1	+M
0	1	1	+2	+2M
1	0	0	-2	-2M
1	0	1	-1	-1M
1	1	0	-1	-1M
1	1	1	0	0

Example

Consider example for radix 4:

Multiplicand 1 0 0 0 0 0 0 1

Multiplier 0 1 1 1 1 1 1 0 0

+2 0 0 -2

0000000011111110

00000000000000

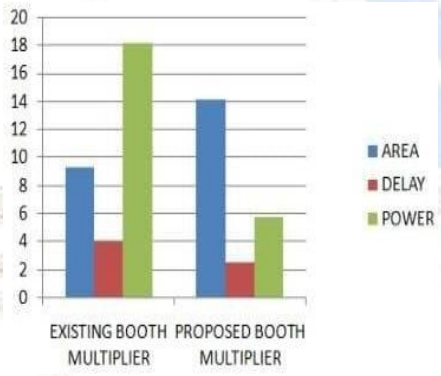
000000000000

1100000010

Product 1100000101111110

V. PERFORMANCE EVALUATION

Comparison Graph Between Existing And Proposed Multiplier



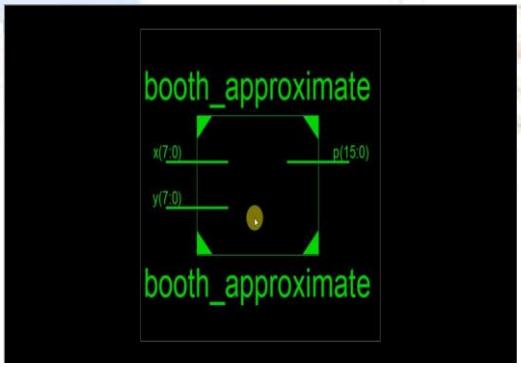
Performance Of Multiplier Based On Radix's

Multiplier Size	Area [LUTs]	Latency (ns)	Area [LUTs]	Latency (ns)
	Ca		Cc	
4×4	12	5.846	12	5.846
8×8	57	7.746	56	6.946
16×16	245	10.765	240	7.613

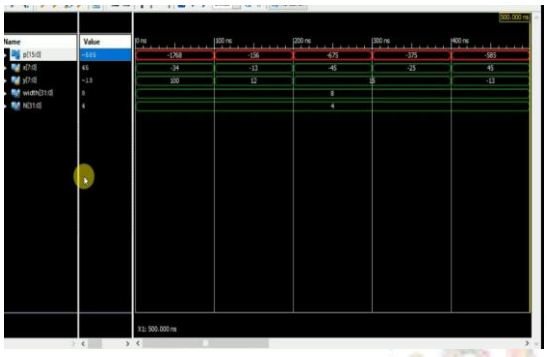
We Also Used Our Multiplier For The Reasoning Phase Of The Neural Network. Ann Inference Accuracy Is 96.67% For 10,000 Photos Using 8-Bit Fixed-Point Numbers And An 8x8 Multiplier. Using a 64-Bit Number And a Multiplier, The Original Accuracy Is 97.09%. The Predicted Lut Savings Would Be 17.5% If The Network Were Implemented On An Fpga Using The Exact Multipliers We Proposed Instead Of The Multiplier Optimized For The Vivado Region. Implementation Of Artificial Neural Network Inference In Fpga: We Use a Neural Network To Estimate Multiplier Performance. The Purpose Of This Experiment Is To Find Out If The Advantages Of Our Multiplier Apply To Other Fpga Architectures And The General System Layout. The Target Fpga Is The Xilinx Zynq Ultrascale Xczu3eg From The Ultra96 Evaluation Platform

VI. SCHEMATIC DIAGRAMS

Block Diagram Of Radix-4 Booth Multiplier



Simulation Result



VII. CONCLUSION

For FPGA-based systems, this study provided a revolutionary area-optimized, low-latency and energy-efficient signed multiplier design. The advantages of coefficients in neural network applications were also assessed. RTL models of our ideas are available as an open source library at https://cfaed.tudresden.de/pd-downloads.

VIII. REFERENCES

[1] Xilinx. 2018. 7 Series DSP48E1 Slice, UG479.

[2] S. Ullah, et al., "Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators," in DAC 2018.

[3] I. Kuon etal., "Measuring the gap between FPGAs and ASICs," in IEEE TCADICS 2007.

[4] Xilinx. 2015. LogiCORE IP Multiplier v12.0, PG108.

[5] A. D. Booth, "A Signed Binary Multiplication Technique," in the Quarterly Journal of Mechanics and Applied Mathematics 1951.

[6] C. R. Baugh etal., "A two's complement parallel array multiplication algorithm," in IEEE TC, vol. 100, no. 12, 1973.

[5] M. Kumm, et al., "An eff i cient softcore multiplier architecture for Xilinx FPGAs," in Computer Arithmetic (ARITH), 2015.

[6] H. Parandeh-Afshar etal., "Measuring and reducing the performance gap between embedded and soft multipliers on FPGAs," in FPL, 2011.

[7] Xilinx. 2016. 7 Series FPGAs Conf i gurable Logic Block, UG474.

[8] H. Parandeh-Afshar, et al., "Exploiting fast carry-chains of FPGAs for designing compressor trees," in FPL, 2009.

[9] A. Kakacak etal., "Fast multiplier generator for FPGAs with LUT based partial product generation and column/row compression," in Integr. VLSI J. 2017.

[10] M. Kumm etal., "Resource Optimal Design of Large Multipliers for FPGAs," in ARITH 2017.

[11] V. Mrazek etal., "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in DATE 2017.

[12] Rajendra Katti, "A Modified Booth Algorithm for High Radix Fixed point Multiplication", Very Large Scale Integration (VLSI) Systems, IEEE Transactions, vol. 2, pp.:
522-524, Dec. 1994.