DIO: 30.0485/ijearst.06.01.3478 https://ijearst.co.in/

ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

ISSN 2386-7876
Vol.01, Issue.03
Mar-2026
Pages: 700-711

EFFICIENT IMPLEMENTATION OF SIGNED MULTIPLIERS ON FPGAS

[1]V V KRISHNA, [2]K HEMESH, [3]J YOGENDRA REDDY, [4]KOTLA DEVENDRANATH BABU, [5]MUDAVAT RAMESH NAIK, [6]MURAM VENKATA SISINDRA REDDY

[1] Guide, Dept of ECE, ABR College of Engineering and Technology, Kanigiri, A.P.
[2,3,4,5,6]B. Tech, Dept of ECE, ABR College of Engineering and Technology, Kanigiri, A.P.

ABSTRACT: This project presents a simple but effective strategy to implement signed binary multipliers on ay hardware with low power consumption. We present the twin-precision technique for integer multipliers. The twin-precision technique can reduce the power dissipation by adapting a multiplier to the bitwidth of the operands being computed. The technique also enables an increased computational throughput, by allowing several narrow-width operations to be computed in parallel. We describe how to apply the twin-precision technique also to signed multiplier schemes, such as Baugh–Wooley algorithm.
Keywords: Baugh–Wooley, Twin Precision, Latency, Density.

INTRODUCTION: Digital signal processing algorithms, embedded systems typically requires a large number of multiplication operations to be performed quickly and repetitively on a set of data. In ALU also multiplication is the important operation, which consumes more power when compare with all arithmetic operations and takes high computational time. During the last decade of integrated electronic design ever more functionality has been integrated onto the same chip, paving the way for having a whole system on a single chip. The strive for ever more functionality increases the demands on circuit designers that have to provide the foundation for all this functionality. The desire for increased functionality and an associated capability to adapt to changing requirements, has led to the design of reconfigurable architectures. With an increased interest and use of reconfigurable architectures there is a need for example and reconfigurable computational units that can meet the demands of high speed, high throughput, low power, and area efficiency. Multiplications are complex to implement and they continue to give designers headaches when trying to efficiently implement multipliers in hardware. Multipliers are therefore interesting to study, when investigating how to design flexible and reconfigurable computational units. In this thesis the results from investigations on flexible multipliers are presented. Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets – high speed, low power consumption, regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed, low power and compact VLSI implementation.

The common multiplication method is "add and shift" algorithm. In parallel multipliers number of partial products to be added is the main parameter that

determines the performance of the multiplier. To reduce the number of partial products to be added, Modified Booth algorithm is one of the most popular algorithms. To achieve speed improvements Wallace Tree algorithm can be used to reduce the number of sequential adding stages. Further by combining both Modified Booth algorithm and Wallace Tree technique we can see advantage of both algorithms in one multiplier. However with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced speed, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing. On the other hand "serial-parallel" multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this lecture we introduce the multiplication algorithms and architecture and compare them in terms of speed, area, power and combination of these metrics. BINARY multipliers are a widely used building block element in the design of microprocessors and embedded systems, and therefore, they are an important target for implementation optimization [1]–[6]. Current implementations of binary multiplication follow the steps of [7]: 1) recoding of the multiplier in digits in a certain number system; 2) digit multiplication of each digit by the multiplicand, resulting in a certain number of partial products; 3) reduction of the partial product array to two operands using multioperand addition techniques; and 4) carry-propagate addition of the two operands to obtain the final result. The recoding type is a key issue, since it determines the number of partial products. The usual recoding process recodes a binary operand into a signed-digit operand with digits in a minimally redundant digit set [7], [8]. Specifically, for radix-r (r = 2m), the binary operand is composed of nonredundant radix-r digits (by just making groups of m bits), and these are recoded from the set {0, 1, . . . , r − 1} to the set {−r/2, . . . ,−1, 0, 1, . . ., r/2} to reduce the complexity of digit multiplications. For n-bit operands, a total of _n/m_ partial products are generated for two's complement representation, and _(n + 1)/m_ for unsigned representation. Radix-4 modified Booth is a widely used recoding method, that recodes a binary operand into radix-4 signed digits in the set {−2, −1, 0, 1, 2}. This is a popular recoding since the digit multiplication step to generate the partial products only requires simple shifts and complementation. The resulting number of partial products is about n/2.Higher radix signed recoding is less popular because the generation of the partial products requires odd multiples of the multiplicand which can not be achieved by means of simple shifts, but require carry-propagate additions. For instance, for radix-16 signed digit recoding [9] the digit set is {−8,−7, . . . , 0, . . . , 7, 8}, so that some odd multiples of the multiplicand have to be generated. Specifically, it is required to generate ×3, ×5, and ×7 multiples (×6 is obtained by simple shift of ×3). The generation of each of these odd multiplies requires a two term addition or subtraction, yielding a total of three carry-propagate additions. However, the advantage of the high radix is that the number of partial products is further reduced. For instance, for radix-16 and n-bit operands, about n/4 partial products are generated. Although less popular than radix-4, there exist industrial instances of radix-8 [10]–[16]. and radix-16 multipliers [17] in microprocessors implementations. The choice of these radices is related to area/delay/power optimization of pipelined multipliers (or fused multiplier adder as in the case of a Intel Itanium microprocessor [17]), for balancing delay between stages and/or reduce the number of pipelining flip-flops. A further consideration is that carry-propagate adders are today highly energy-delay optimized, while partial

product reductions trees suffer the increasingly serious problems related to a complex wiring and glitching due to unbalanced signal paths. It is recognized in the literature that a radix-8 recoding leads to lower power multipliers compared to radix-4 recoding at the cost of higher latency (as a combinational block, without considering pipelining) [4], [18]. Moreover, although the radix-16 multiplier requires the generation of more odd multiples and has a more complex wiring for the generation of partial products [4], a recent microprocessor design [17] considered it to be the best choice for low power (under the specific constraints for this microprocessor). In [1] and [2], some optimizations for radix-4 two's complement multipliers were introduced. Although for n-bit operands, a total of _n/2_ partial products are generated, the resulting maximum height of the partial product array is _n/2_+ 1 elements to be added (in just one of the columns). This extra height by a single-bit row is due to the +1 introduced in the bit array to make the two's complement of the most significant partial product (when the recoded most significant digit of the multiplier is negative).

LITERATURE SURVEY: A traditional method to reduce the aging effects is overdesign which includes techniques like guard-banding ad gate oversizing. This approach can be area and power inefficient [8]. To avoid this problem, an NBTI-aware technology mapping technique was proposed in [7] which guarantee the performance of the circuit during its lifetime. Another technique was an NBTI- aware sleep transistor in [3] which improve the lifetime stability of the power gated circuits under considerations. A joint logic restructuring and pin reordering method in [6] is based on detecting functional symmetries and transistor stacking effects. This approach is an NBTI optimization method that considered path sensitization. Dynamic voltage scaling and bogy-biasing techniques were proposed in [4] and [5] to reduce power or extend circuit life. These techniques require circuit modification or do not provide optimization of specific circuits. Every gate in any VLSI circuit has its own delay which reduces the performance of the chip. Traditional circuits use critical pathdelayas the overall circuit clock cycle in order to perform correctly. However, in many worst-case designs, the probability that the critical pathdelay is activated is low. In such cases, the strategy of minimizing the worst-case conditions may lead to inefficient designs. Fornoncritical path, using the critical path delay as the overall cycle period will result in significant timing waste. Hence, the variable latencydesign was proposed to reduce the timing waste of traditional circuits. A short path activation function algorithm was proposed in [16] to improve the accuracy of the hold logic and to optimize the performance of the variable-latency circuit. An instruction scheduling algorithm was proposed in [17] to schedule the operations on nonuniform latency functional units and improve the performance of Very Long Instruction Word processors. In [18], a variable-latency pipelined multiplier architecture with a Booth algorithm was proposed. In [19], process-variation tolerant architecture for arithmetic units was proposed, where the effect of process-variation is considered to increase the circuit yield. In addition, the critical paths are divided into two shorter paths that could be unequal and the clock cycle is set to the delay of the longer one. These research designs were able to reduce the timing waste of traditional circuits to improve performance, but they did not consider the aging effect and could not adjust themselves during the runtime. A variable-latency adder design that considers the aging effect was proposed in [20]. Chen et al (2003) presented low-power 2's complement multipliers by minimizing the switching activities of partial

products using the radix-4 Booth algorithm. Before computation for two input data, the one with a smaller effective dynamic range is processed to generate Booth codes, thereby increasing the probability that the partial products become zero. By employing the dynamic-range determination unit to control input data paths, the multiplier with a column-based adder tree of compressors or counters is designed. To further reduce power consumption, the two multipliers based on row-based and hybrid-based adder trees are realized with operations on effective dynamic ranges of input data. Functional blocks of these two multipliers can preserve their previous input states for non effective dynamic data ranges and thus, reduce the number of their switching operations. It illustrates the proposed multipliers exhibiting low-power dissipation, the theoretical analyzes of switching activities of partial products 27 are derived. The proposed 16 /spl times/ 16-bit multiplier with the columnbased adder tree conserves more than 31.2%, 19.1%, and 33.0% of power consumed by the conventional multiplier. Furthermore, the proposed multipliers with row-based, hybrid-based adder trees reduce power consumption by over 35.3%, 25.3% and 39.6%, and 33.4%, 24.9% and 36.9%, respectively. Oscal et al (2003) presented low-power 2's complement multipliers by minimizing the switching activities of partial products using the radix-4 Booth algorithm. Before computation for two input data, the one with a smaller effective dynamic range is processed to generate Booth codes, thereby increasing the probability that the partial products become zero. By employing the dynamic-range determination unit to control input data paths, the multiplier with a column-based adder tree of compressors or counters is designed. To further reduce power consumption, the two multipliers based on row-based and hybrid-based adder trees are realized with operations on effective dynamic ranges of input data. Functional blocks of these two multipliers can preserve their previous input states for non effective dynamic data ranges and thus, reduce the number of their switching operations. To illustrate the proposed multipliers exhibiting low-power dissipation, the theoretical analyzes of switching activities of partial products are derived. The proposed 16 /spl times/ 16-bit multiplier with the column-based adder tree conserves more than 31.2%, 19.1%, and 33.0% of power consumed by the conventional multiplier, in applications of the ADPCM audio, G.723.1 speech, and wavelet-based image coders, respectively. Furthermore, the proposed multipliers with row-based, hybrid-based adder trees reduce power consumption by over 35.3%, 25.3% and 39.6%, and 33.4%, 24.9% and 36.9%, respectively. When considering product factors of hardware areas, critical delays and power consumption, the proposed multipliers can outperform the conventional multipliers.

PROPOSED METHOD:

TWIN PRECISION: During the last decade of integrated electronic design ever more functionality has been integrated onto the same chip, paving the way for having a whole system on a single chip. The strive for ever more functionality increases the demands on circuit designers that have to provide the foundation for all this functionality. The desire for increased functionality and an associated capability to adapt to changing requirements, has led to the design of reconfigurable architectures. With an increased interest and use of reconfigurable architectures there is a need for exited and reconfigurable computational units that can meet the demands of high speed, high throughput, low power, and area efficiency. Multiplications are complex to implement and they continue to give designers headaches when trying to efficiently

implement multipliers in hardware. Multipliers are therefore interesting to study, when investigating how to design flexible and reconfigurable computational units

APPLYING TWIN PRECISION TO GENERAL MULTIPLIER: For a first analysis of the twin-precision technique, the discussion will be based on an illustration of an unsigned binary multiplication. In an unsigned binary multiplication each bit of one of the operands, called the multiplier, is multiplied with the second operand, called multiplicand that way one row of partial products is generated. Each row of partial products is shifted according to the position of the bit of the multiplier, forming what is commonly called the partial-product array. Finally, partial products that are in the same column are summed together, forming the final result. An illustration of an 8-bit multiplication is shown in below figure

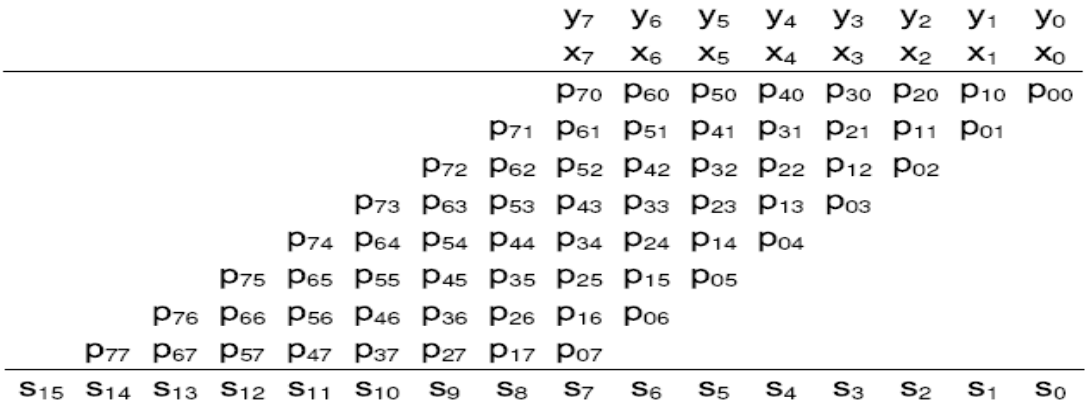


Figure 1.1: Illustration of an unsigned 8-bit multiplication

Let us look at what happens when the precision of the operands is smaller than the multiplier we intend to use. In this case, the most significant bits of the operands will only contain zeros, thus large parts of the partial-product array will consist of zeros. Further, the summation of the most significant part of the Partial-product array and the most significant bits of the final result will only consist of zeros.

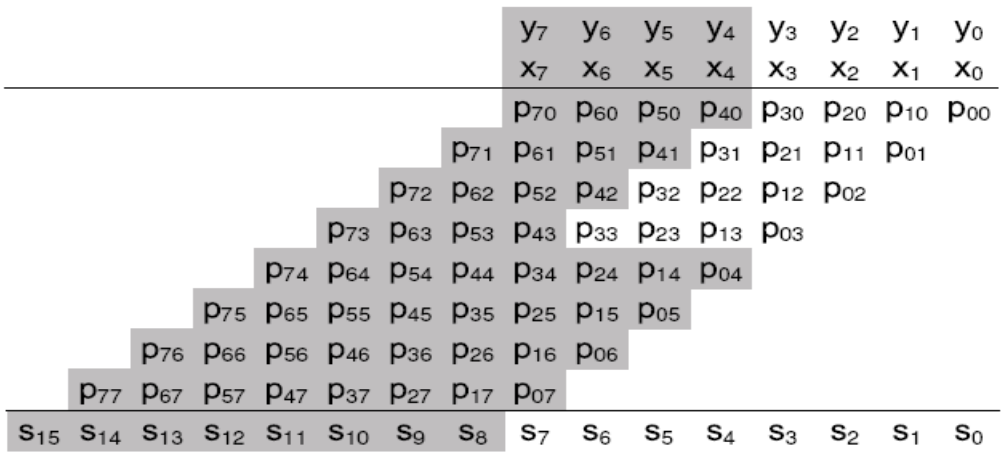


Figure 1.2: Illustration of an unsigned 8-bit multiplication, where the precision of the operands is smaller than the precision of the multiplication. Unused bits of operands and product, as well as unused partial products, are shown in gray.

Fig. 1.2 shows that large parts of the partial products are only containing zeros and are, thus, not contributing with any useful information for the final result.

If we are only looking at the upper half of the operands, the partial products generated from these bits are the ones shown in black in Fig. 1.3. By setting the other partial products to zero, it is then possible to perform two multiplications within the same partial-product array, without changing the way the summation of the partial-product array is done. How the partial products, shown in gray, can be set to zero will be investigated in the implementation section later on.

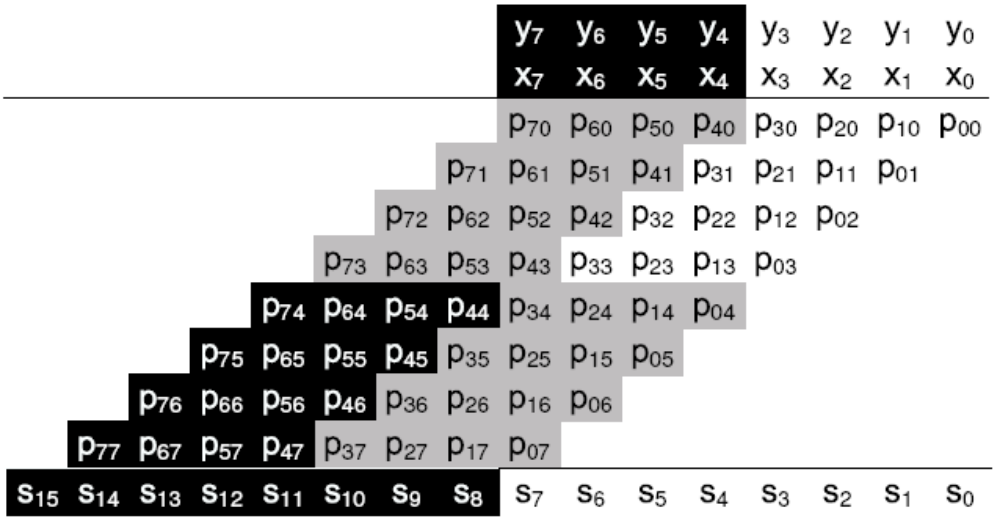


Figure 1.3: Illustration of an unsigned 8-bit multiplication, where a 4-bit multiplication, shown in white, is computed in parallel with a second 4-bit multiplication, shown in black.

Assume, for now, that there is a way of setting unwanted partial products to zero, then it suddenly becomes possible to partition the multiplier into two smaller multipliers that can compute multiplications in parallel. In the above illustrations the two smaller multiplications have been chosen such that they are of equal size. This is not necessary for the technique to work. Any size of the two smaller multiplications can be chosen, as long as the precision of the two smaller multiplications together are equal or smaller than the full precision (NFULL) of the multiplication, To be able to distinguish between the two smaller multiplications, they are referred to as the multiplication in the east Significant Part (LSP) of the partial-product array with size NLSP , shown in white, and the multiplication in the Most Significant part (MSP) with size MSP , shown in black.

FIRST IMPLEMENTATION: The basic operation of generating a partial product is that of a 1-bit multiplication using a 2-input AND gate, where one of the input

signals is one bit of the multiplier and the second input signal is one bit of the multiplicand. The summation of the partial products can be done in many different ways, but for this investigation we are only interested in parallel multipliers that are based on 3:2 full adders1. For this First implementation an array of adders will be used because of its close resemblance to the previously used illustration of a multiplication. In the previous section we assumed that there is a way of setting unwanted partial products to zero. This is easily accomplished by changing the 2-input AND gate to a 3-input AND gate, where the extra input can be used for a control signal. Of course, only the AND gates of the partial products that has to be set to zero need to be changed to a 3-input version. During normal operation when a full-precision multiplication is executed the control signal is set to high,

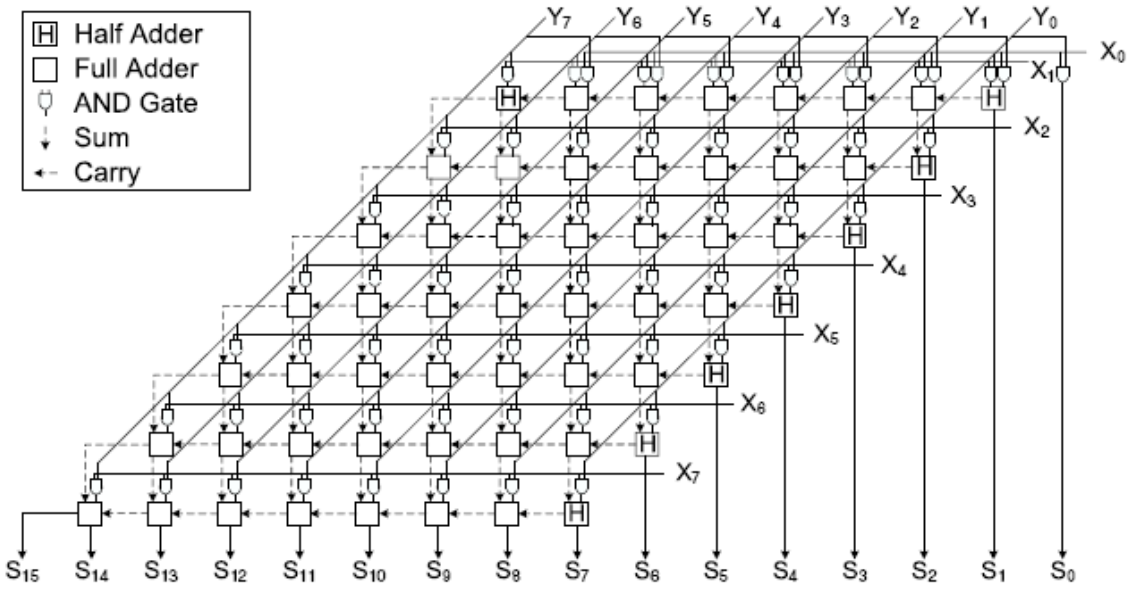


Figure 1.4: Block diagram of an unsigned 8-bit array multiplier.

thus all partial products are generated as normal and the array of adders will sum them together and create the final result. When the control signal is set to low the unwanted partial products will become zero. Since the summation of the partial products is not overlapping, there is no need to modify the array of adders. The array of adders will produce the result of the two multiplications in the upper and lower part of the final output. The block diagram of an 8-bit twin-precision array multiplier capable of computing two 4-bit multiplications is shown in Fig. 1.5. The two multiplications have been colored in white and black to visualize what part of the adder array is used for what multiplication. More flexibility might be wanted, like the possibility to compute a single low-precision multiplication or two parallel low-precision multiplications, within the same multiplier. This can be done by changing the 2-input AND gates for the partial product generation of the low-precision multiplication as well. In the array multiplier in Fig. 1.5, the AND gates for the 4-bit MSP multiplication, shown in black, can be changed to 3-input AND gates to which a second control signal can be added. Assuming the multiplier is divided into two equal parts, this modification makes it possible to either compute an N-bit, a single N=2-bit or two concurrent N=2-bit multiplications.

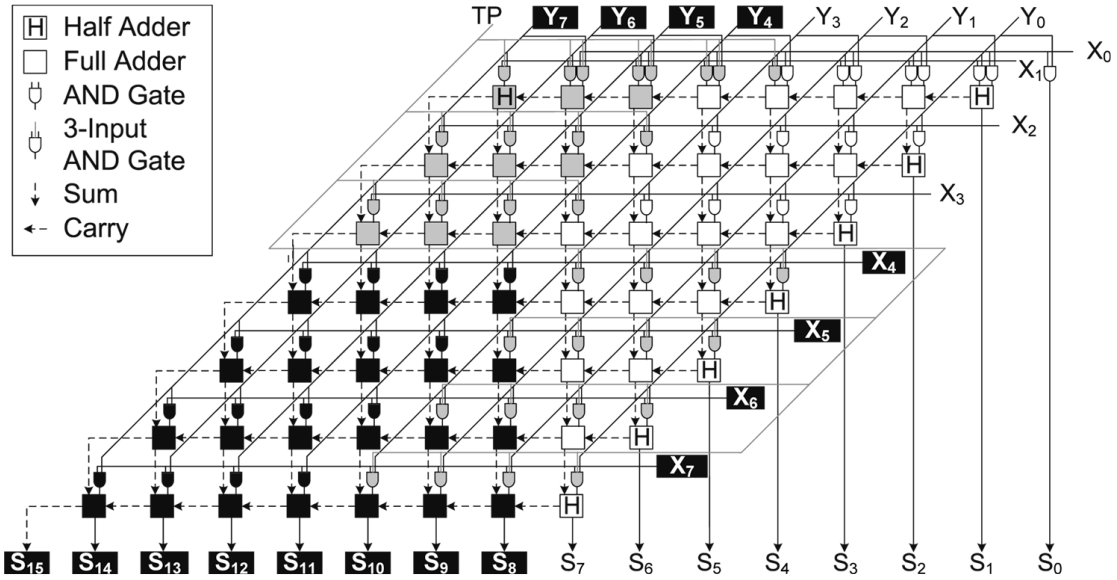


Fig.1. 5 Block diagram of an unsigned 8-bit twin-precision array multiplier. The TP signal is used for controlling if one full-precision multiplication should be computed (TP= high) or two 4-bit multiplications should be computed in parallel (TP=Low)

BAUGH WOOLEY MULTIPLICATION:

Adjusts partial products to maximize regularity of multiplication array.

Moves partial products with negative signs to the last steps; also adds negation of partial products rather than subtracts.

Baugh-Wooley Multiplier

This technique has been developed in order to design regular multipliers, suited for 2's-complement numbers.

Let us consider 2 numbers A and B :

$$A = (a_{n-1} \dots a_0) = -a_{n-1} \cdot 2^{n-1} + \sum_{0}^{n-2} a_i \cdot 2^i$$

$$B = (b_{n-1} \dots b_0) = -b_{n-1} \cdot 2^{n-1} + \sum_{0}^{n-2} b_i \cdot 2^i \quad (64),\ (65)$$

The product A.B is given by the following equation :

$$A \cdot B = a_{n-1} \cdot b_{n-1} \cdot 2^{2n-2} + \sum_{0}^{n-2} \sum_{0}^{n-2} a_i \cdot b_i \cdot 2^{i+j} - a_{n-1} \sum_{0}^{n-2} b_i \cdot 2^{n+i-1} - b_{n-1} \sum_{0}^{n-2} a_i \cdot 2^{n+i-1}$$

(66)

We see that subtractor cells must be used. In order to use only adder cells, the negative terms may be rewritten as :

$$-a_{n-1} \sum_{0}^{n-2} b_i \cdot 2^{i+n-1} = a_{n-1} \cdot \left(-2^{2n-2} + 2^{n-1} + \sum_{0}^{n-2} \overline{b_i} \cdot 2^{i+n-1} \right) \quad (67)$$

By this way, A.B becomes :

$$A.B = a_{n-1}.b_{n-1}.2^{2n-2} + \sum_{0}^{n-2}\sum_{0}^{n-2} a_i.b_j.2^{i+j}$$

$$+ b_{n-1}\left[-2^{2n-2} + 2^{n-1} + \sum_{0}^{n-2} \overline{a_i}.2^{i+n-1}\right]$$

$$+ a_{n-1}\left[-2^{2n-2} + 2^{n-1} + \sum_{0}^{n-2} \overline{b_i}.2^{i+n-1}\right] \quad (68)$$

The final equation is :

$$A.B = -2^{2n-1} + \left(\overline{a_{n-1}} + \overline{b_{n-1}} + a_{n-1}.b_{n-1}\right).2^{2n-2}$$

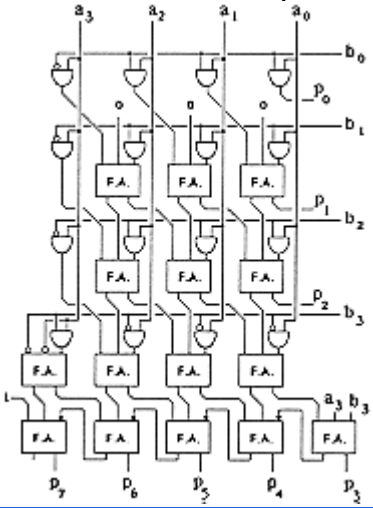
$$+ \sum_{0}^{n-2}\sum_{0}^{n-2} a_i.b_j.2^{i+j} + \left(a_{n-1} + b_{n-1}\right).2^{n-1}$$

$$+ \sum_{0}^{n-2} b_{n-1}.\overline{a_i}.2^{i+n-1} + \sum_{0}^{n-2} a_{n-1}.\overline{b_i}.2^{i+n-1} \quad (69)$$

because :

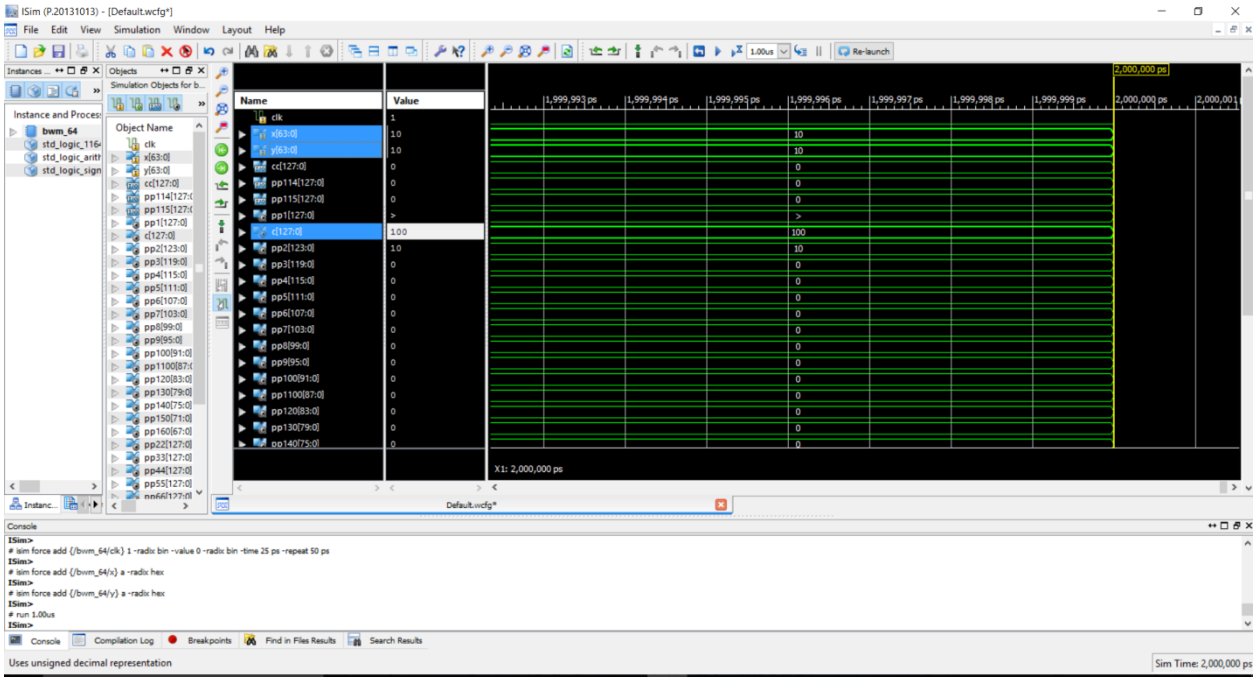
$$-\left(b_{n-1} + a_{n-1}\right).2^{2n-2} = -2^{2n-1} + \left(\overline{a_{n-1}} + \overline{b_{n-1}}\right).2^{2n-2} \quad (70)$$

A and B are n-bits operands, so their product is a 2n-bits number. Consequently, the most significant weight is 2n-1, and the first term -22n-1 is taken into account by adding a 1 in the most significant cell of the multiplier.



]Figure-1.6: shows a 4-bits Baugh-Wooley multiplier

RESULTS:



ADVANTAGES

1. Reduces the number of partial products using techniques like Booth encoding
2. Improves computation speed with parallel processing structures
3. Efficient handling of signed numbers (2's complement representation)
4. Lower power consumption compared to conventional multipliers
5. Optimized area utilization in VLSI implementations
6. Suitable for high-speed arithmetic operations
7. Scalable for larger bit-width operations (16-bit, 32-bit, 64-bit)

APPLICATIONS

1. Digital Signal Processing (DSP) systems
2. Image and video processing
3. Microprocessors and microcontrollers
4. Communication systems (modulation, filtering)
5. Cryptographic applications
6. Artificial Intelligence and Machine Learning accelerators
7. Embedded systems and real-time computing
8. Graphics processing units (GPUs)

Conclusion: The efficient implementation of signed multipliers plays a crucial role in improving the overall performance of digital systems. By adopting advanced algorithms such as Twin precision and Baugh-wooley algorithms, significant improvements in speed, area, and power consumption can be achieved. The use of sign handling techniques and reduction of partial products contributes to faster computation and reduced hardware complexity. Overall, the proposed or studied multiplier designs demonstrate better efficiency compared to conventional array multipliers, making them suitable for high-performance and low-power applications.

FUTURE SCOPE

Implementation using advanced technologies like FinFET and GAAFET
Integration with low-power design techniques such as clock gating and power gating
Exploration of approximate multipliers for AI/ML applications
Use of emerging technologies like Quantum-dot Cellular Automata (QCA)
Optimization using machine learning-based hardware design techniques
Implementation in FPGA and ASIC for real-time applications
Development of hybrid multiplier architectures combining multiple algorithms
Focus on fault-tolerant and error-resilient multiplier designs

REFERENCES:

1. C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
2. L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349–356, 1965.
   A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236–240, 1951.
3. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," Proceedings of the IRE, vol. 49, no. 1, pp. 67–91, Jan. 1961.
4. B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, Oxford University Press, 2000.
5. N. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed., Pearson, 2011.
6. K. Hwang, Computer Arithmetic: Principles, Architecture, and Design, Wiley, 1979.
7. R. Zimmermann and W. Fichtner, "Low-Power Logic Styles: CMOS Versus Pass-Transistor Logic," IEEE Journal of Solid-State Circuits, vol. 32, no. 7, pp. 1079–1090, Jul. 1997.
8. S. Knowles, "A Family of Adders," Proceedings of the IEEE Symposium on Computer Arithmetic, pp. 277–281, 1999.
9. J. M. Rabaey, Digital Integrated Circuits, Prentice Hall, 2003.
10. P. K. Meher et al., "50 Years of Booth Algorithm," IEEE Transactions on Circuits and Systems, vol. 58, no. 8, pp. 1942–1951, Aug. 2011.
11. M. D. Ercegovac and T. Lang, Digital Arithmetic, Morgan Kaufmann, 2004.
12. Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," ACM Transactions on Algorithms, vol. 3, no. 2, 2007.
13. H. Eriksson et al., "Comparison of CMOS and Pass Transistor Logic," IEEE Journal of Solid-State Circuits, 1997.
14. S. Venkatesan et al., "High-Speed Wallace Tree Multiplier," IEEE Conference on VLSI Design, 2007.
15. Bellaouar and M. Elmasry, Low-Power Digital VLSI Design, Springer, 1995.
16. J. E. Stine and M. J. Schulte, "The Symmetric Truncated Multiplier," IEEE Transactions on Computers, 2001.
17. D. Harris, "A Taxonomy of Parallel Prefix Networks," Asilomar Conference on Signals, 2003.
18. S. Borkar, "Design Challenges of Technology Scaling," IEEE Micro, 1999.
19. K. Roy et al., "Leakage Current Mechanisms and Reduction Techniques," Proceedings of the IEEE, 2003.