ENGINEERING IN ADVANCED RESEARCH SCIENCE AND TECHNOLOGY

ISSN 2278-2566
Vol.02, Issue.03
June -2018
Pages: 467-475

EFFICIENT AND ENHANCED LOW DENSE REIABLE MULTIPLIER DESIGN

1. A.N.LAKSHMI, 2.P.PRASANNA KUMAR

1. PG Student, Dept. of ECE, Kakinada Institute Of Technological Sciences, Ramachandrapuram , A.P
2. Assistant Professor, Dept. of ECE, Kakinada Institute Of Technological Sciences, Ramachandrapuram , A.P

ABSTRACT:

This project presents a design methodology for high-speed Booth encoded parallel multiplier. For partial product generation, we propose a new modified Booth encoding (MBE) scheme to improve the performance of traditional MBE schemes. An an optimization for binary radix-16 (modified) Booth recoded multipliers to reduce the maximum height of the partial product columns to $\lfloor n/4 \rfloor$ for n = 64-bit unsigned operands is presented in this concept. This is in contrast to the conventional maximum height of $\lfloor (n + 1)/4 \rfloor$. Therefore, a reduction of one unit in the maximum height is achieved. This reduction may add flexibility during the design of the pipelinedmultiplier to meet the design goals, it may allow further optimizations of the partial product array reduction stage in terms of area/delay/power and/or may allow additional addends to be included in the partial product array without increasing the delay. Further, this project is enhanced by using modified square root carry select adder to further reduce timing constraints.

KEYWORDS: Modified Booth Encoding, Radix-16, Pipeline, Multiplier, Enhanced, Carry select adder, Binary Excess Converter.

INTRODUCTION:

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.

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 waspropose 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 wereproposed 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 effectwas proposed in [20] and [21]. 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.

EXISTING TECHNIQUE:

RADIX-16 BOOTH MULTIPLIER: describe briefly the architecture of the basic radix-16 Booth multiplier (see [17] for instance). For sake of simplicity, but without loss of generality, we consider unsigned operands with $n = 64$. Let us denote with X the multiplicand operand with bit components x_i ($i = 0$ to $n - 1$, with the least-significant bit, LSB, at position 0) and with Y the multiplier operand and bit components y_i. The first step is the recoding of the multiplier operand [8]: groups of four bits with relative values in the set $\{0, 1, \ldots, 14, 15\}$ are recoded to digits in the set $\{-8,-7, \ldots, 0, \ldots, 7, 8\}$ (minimally redundant radix-16 digit set to reduce the number of multiples). This recoding is done with the help of a transfer digit t_i and an interim digit w_i [7]. The recoded digit z_i is the sum of the interim and transfer digits $z_i = w_i + t_i$. When the value of the four bits, v_i, is less than 8, the transfer digit is zero and the interim digit $w_i = v_i$. For values of v_i greater than or equal to 8, v_i is transformed into $v_i = 16 - (16 - v_i)$, so that a transfer digit is generated to the next radix-16 digit position (t_i+1) and an interim digit of value $w_i = -(16 - v_i)$ is left. That is $0 \le v_i < 8 : t_i+1 = 0$ $w_i = v_i$ $w_i \in [0, 7]$ $8 \le v_i \le 15 : t_i+1 = 1$ $w_i = -(16 - v_i)$ $w_i \in [-8,-1]$. The transfer digit corresponds to the most-significant bit (MSB) of the four-bit group, since this bit determines if the radix-16 digit is greater than or equal to 8. The final logical step is to add the interim digits and the transfer digits (0 or 1) from the radix-16 digit position to the right. Since the transfer digit is either 1 or 0, the addition of the interim digit and the transfer digit results in a final digit in the set $\{-8,-7, \ldots, 0, \ldots, 7, 8\}$. Due to a possible transfer digit from the most significant radix-16 digit, the number of resultant radix-16 recoded digits is $\lfloor (n + 1)/4 \rfloor$. Therefore, for $n = 64$ the number of recoded digits (and the number of partial products) is 17. Note that the most significant digit is 0 or 1 because it is in fact just a transfer digit. After recoding, the partial products are generated by digit multiplication of the recoded digits times the multiplicand X.

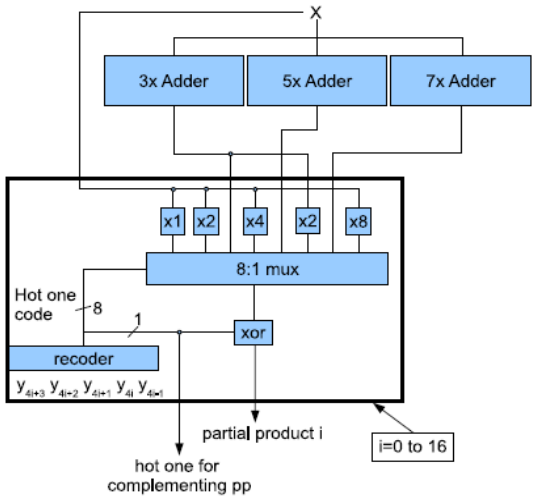


Fig. 1. Partial product generation. For the set of digits $\{-8,-7, \ldots, 0, \ldots, 7, 8\}$, the multiples 1X, 2X, 4X,

and 8X are easy to compute, since they are obtained by simple logic shifts. The negative versions of these multiples are obtained by bit inversion and addition of a 1 in the corresponding position in the bit array of the partial products. The generation of 3X, 5X, and 7X (odd multiples) requires carry-propagate adders (the negative versions of these multiples are obtained as before). Finally, 6X is obtained by a simple one bit left shift of 3X. Fig. 1 illustrates a possible implementation of the partial product generation. Five bits of the multiplier Y are used to obtain the recoded digit (four bits of one digit and one bit of the previous digit to determine the transfer digit to be added). The resultant digit is obtained as a one-hot code to directly drive a 8 to 1 multiplexer with an implicit zero output (output equal to zero when all the control signals of the multiplexer are zero). The recoding requires the implementation of simple logic equations that are not in the critical path due to the generation in parallel of the odd multiples (carry-propagate addition). The XOR at the output of the multiplexer is for bit complementation (part of the computation of the two's complement when the multiplier digit is negative). Fig. 2(a) illustrates part of the resultant bit array for $n = 64$ after the simplification of the sign extension [7]. In general, each partial product has $n + 4$ bits including the sign in two's complement representation. The extra four bits are required to host a digit multiplication by up to 8 and a sign bit due to the possible multiplication by negative multiplier digits. Since the partial products are left-shifted four bit positions with respect to each other, a costly sign extension would be necessary. However, the sign extension is simplified by concatenation of some bits to each partial product (S is the sign bit of the partial product and C is S complemented): $CSSS$ for the first partial product and $111C$ for the rest of partial products (except the partial product at the bottom that is non negative since the corresponding multiplier digit is 0 or 1). The bits denoted by b in Fig. 2 corresponds to the logic 1 that is added for the two's complement for negative partial products. After the generation of the partial product bit array, the reduction (multioperand addition) from a maximum height of 17 (for $n = 64$) to 2 is performed. The methods for multioperand addition are well known, with a common solution consisting of using 3 to 2 bit reduction with full adders (or 3:2 carry-save adders) or 4 to 2 bit reduction with 4:2 carry-save adders. The delay and design effort of this stage are highly dependent on the maximum height of the bit array. It is recognized that reduction arrays of 4:2 carry-save adders may lead to more regular layouts [16]. For instance, with a maximum height of 16, a total of 3 levels of 4:2 carry-save adders would be necessary. A maximum height of 17 leads to different approaches that may increase the delay and/or require to use arrays of 3:2 carry-save adders interconnected to minimize delay [20]. After the reduction to two operands, a carry-propagate addition is performed. This addition may take advantage of the specific signal arrival times from the partial product reduction step. To reduce the maximum height of the partial product bit array we perform a short carry-propagate addition in parallel to the regular partial product generation. This short addition reduces the maximum height by one row and it is faster than the regular partial product generation. Fig. 2(b) shows the elements of the bit array to be added by the short adder. Fig. 2(c) shows the resulting partial product bit array after the short addition. Comparing both figures, we observe that the maximum height is reduced from 17 to 16 for $n = 64$. Fig. 3 shows the specific elements of the bit array (boxes) to be added by the short carry-propagate addition. In this figure, pi,j corresponds to the bit j of partial product i, $s0$ is the sign bit of partial product 0, $c0 = \text{NOT}(s0)$, bi is the bit for the two's complement of partial product i, and zi is the ith bit of the result of the short addition. The selection of these specific bits to be added is justified by the fact that, in this way, the short addition delay is hidden from the critical path that corresponds to a regular partial product generation (this will be shown in Section IV). We perform the computation in two concurrent parts A and B as indicated in Fig. 3. The elements of the part A are generated faster than the elements of part B. Specifically the elements of part A are obtained from: • the sign of the first partial product: this is directly obtained from bit $y3$ since there is no transfer digit from a previous radix-16 digit; • bits 3 to 7 of partial product 16: the recoded digit for partial product 16 can only be 0 or 1, since it is just a transfer digit. Therefore the bits of this partial product are generated by a simple AND operation of the bits of the multiplicand X and bit $y63$ (that generates the transfer from the previous digit). Therefore, we decided to implement part A as a speculative addition, by computing two results, a result with carry-in = 0 and a result with carry-in = 1. This can be computed efficiently with a compound adder [7].

PROPOSED TECHNIQUE:

DESIGN OF SQUARE ROOT CSLA

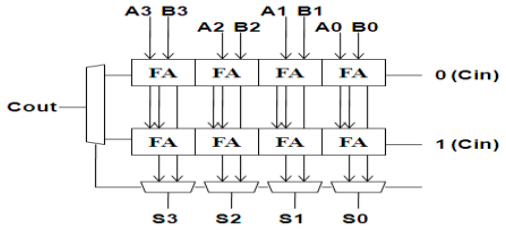


Fig: Basic building block of CSLA

Block diagram of 16-bit Carry Select adder:

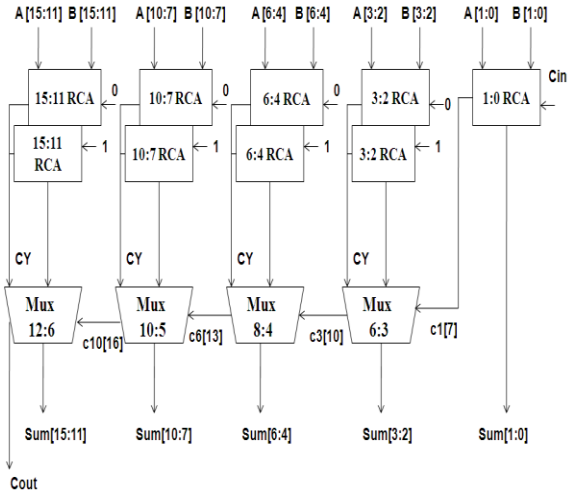


Fig: Existing system (Regular 16-bit Carry select adder)

The block diagram of the regular 16-bit square root CSLA is shown in the figure 3.2. This adder is a variable sized adder.

The carryselect adder generally consists of two ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known.

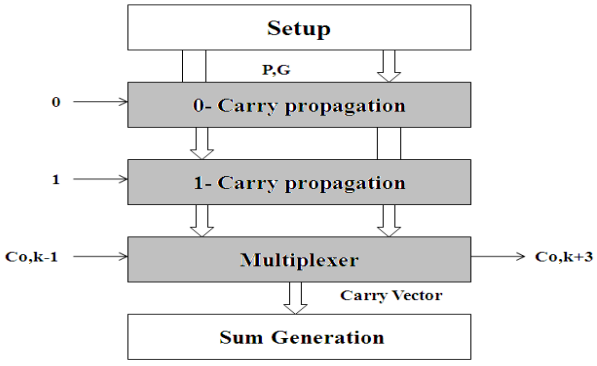


Fig: -bit carry select adder module topology

Seeing at the figure, the hardware overhead of the carry select adder is restricted to an additional carry path and a multiplexer and equals about 80% with respect to ripple carry adder. A full carry select adder is now constructed by chaining equal number of adder stages.

The critical path is shaded in gray color. From inspection of the circuit, we can derive the first order model of the worst case propagation delay of the module written as,

$$T = t_{setup} + P \times t_{carry} + (2N)^{1/2} \times t_{mux} + t_{sum}$$

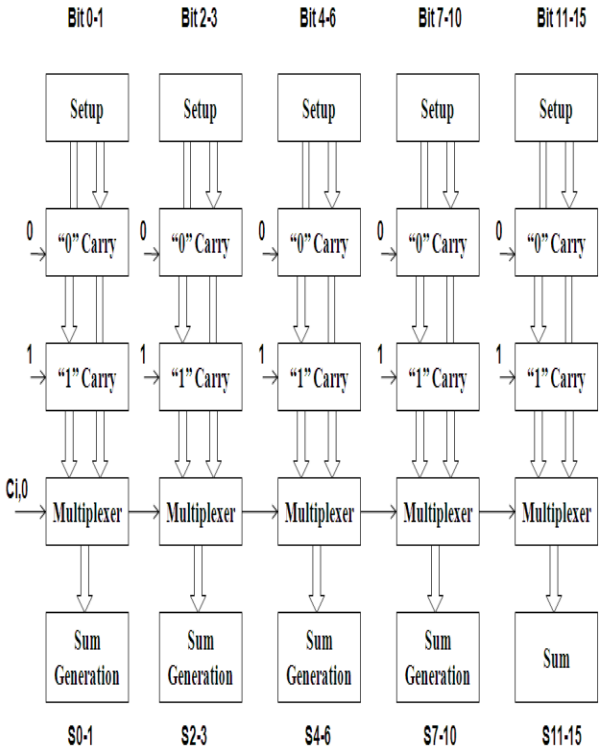
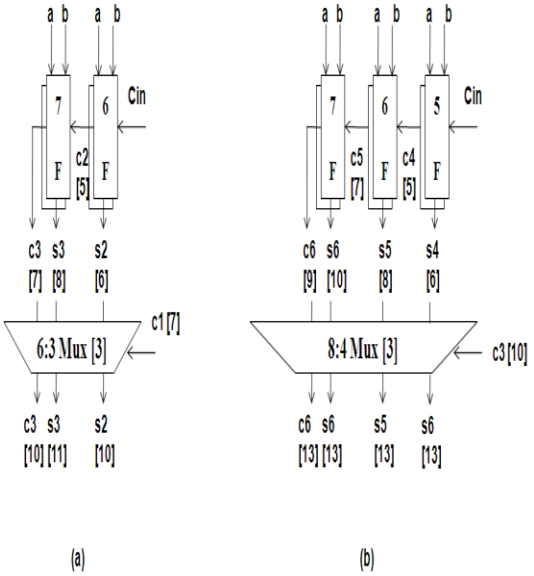


Fig: Delay propagation of 16-bit CSLA

The design procedure and the delay propagation of the 16-bit square root CSLA can be best explained from the figure 3.4. As from the figure, it can be seen that the model consists of 5 groups of different size. The addition process is carried out by considering the carry Cin=0 and Cin=1 and then generating the actual sum and carry using the actual carry from the previous stage is accomplished.

Architecture of 16-bit square root CSLA:



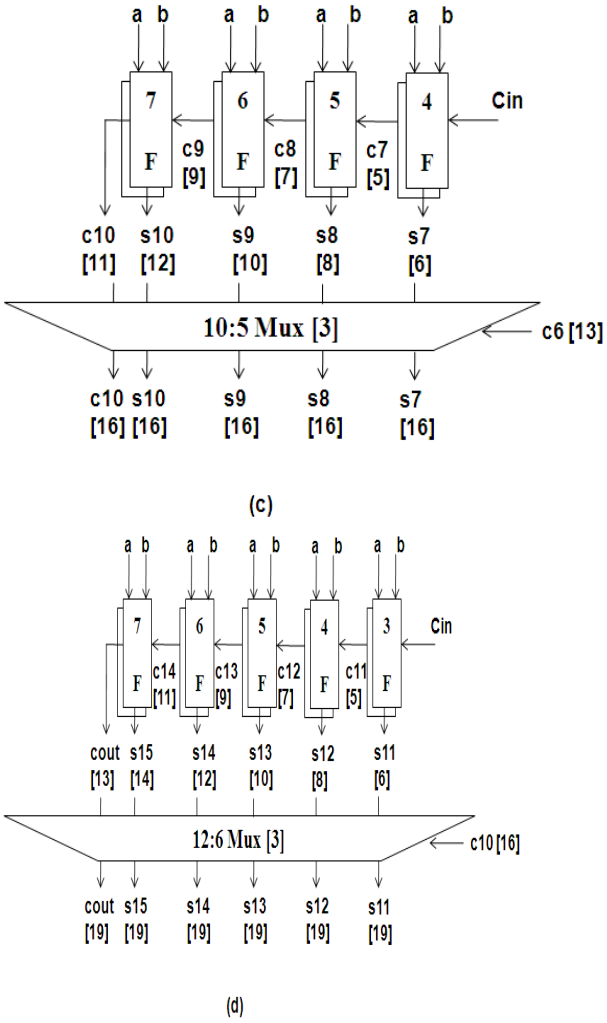


Fig: Delay and area evaluation of regular SQRT CSLA: (a) group2, (b) group3, (c) group4 and (d) group5. F is a Full Adder.

This 16-bit square root CSLA consists of five groups where each group is of variable size. The 16-bit value data is divided as 2-bit, 2-bit, 3-bit, 4-bit, 5-bit groups. The first group consists of 2-bit ripple carry adder. The actual input carry is applied to this adder. The ripple carry adder receives the carry and performs the 2 2-bit addition (a[1:0], b[1:0]).

Delay and Area Evaluation Methodology of the basic adder blocks:

The AND, OR and Inverter (AOI) implementation of an XOR gate is shown in the figure 3.6. The gates between the dotted lines are performing the operations in parallel and the numeric representation of each gate indicates the delay contributed by that gate.

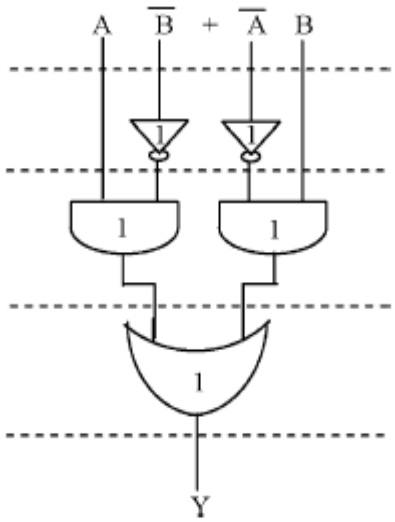
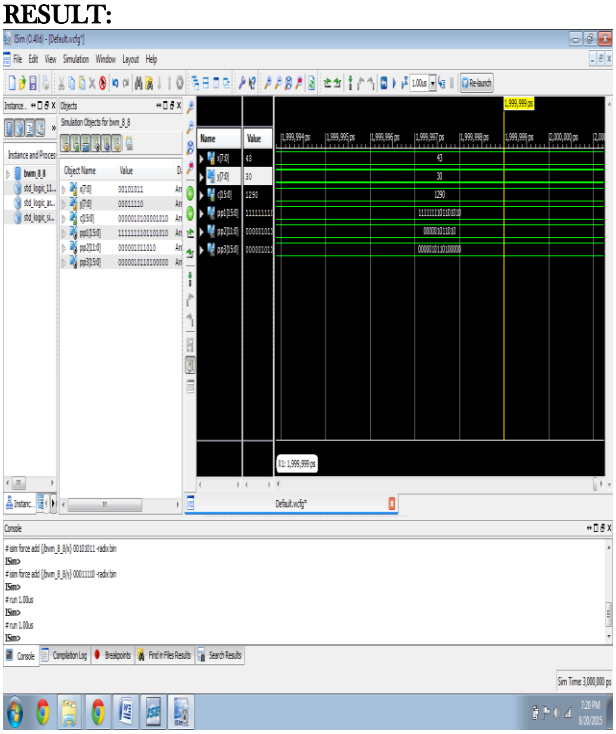


Fig : Delay and Area evaluation of an XOR gate

The delay and area evaluation methodology considers all gates to be made up of AND, OR and Inverter, each having delay equal to 1 unit and area equal to 1 unit.

RESULT:



CONCLUSION:

In this paper, a new Radix16 modified booth multiplier architecture along with carry select adder is presented to execute the multiplication-accumulation

operation, which is the key operation, for digital signal processing and multimedia information processing efficiently, was proposed. By removing the independent accumulation process that has the largest delay and merging it to the compression process of the partial products, the overall multiplier performance has been improved almost twice as much as in the previous algorithms.

REFERENCE:

[1] Wen-Chang Yeh and Chein-Wei Jen, "High-speed Booth encoded parallel multiplier design," IEEE Trans. on Computers, vol. 49, isseu 7, pp. 692-701, July 2000.

[2] Jung-Yup Kang and Jean-Luc Gaudiot, "A simple high-speed multiplier design," IEEE Trans. on Computers, vol. 55, issue 10, Oct. pp. 1253-1258, 2006.

[3] Shiann-Rong Kuang, Jiun-Ping Wang and Cang-Yuan Guo, "Modified Booth multipliers with a regular partial product array," IEEE Trans. onCircuit and Systems, vol.56, Issue 5, pp. 404-408, May 2009.

[4] Li-rong Wang, Shyh-Jye Jou and Chung-Len Lee, "A well-structured modified Booth multiplier design," Proc. of IEEE VLSI-DAT, pp. 85-88, April 2008.

[5] A. A. Khatibzadeh, K. Raahemifar and M. Ahmadi, "A 1.8V 1.1GHz Novel Digital Multiplier," Proc. of IEEE CCECE, pp. 686-689, May 2005.

[6] S. Hus, V. Venkatraman, S. Mathew, H. Kaul, M. Anders, S. Dighe, W. Burleson and R. Krishnamurthy, "A 2GHZ 13.6mW 12x9b mutiplier for energy efficient FFT accelerators," Proc. of IEEE ESSCIRC, pp. 199-202, Sept. 2005.

[7] Hwang-Cherng Chow and I-Chyn Wey, "A 3.3V 1GHz high speed pipelined Booth multiplier," Proc. of IEEE ISCAS, vol. 1, pp. 457-460, May 2002.

[8] M. Aguirre-Hernandez and M. Linarse-Aranda, "Energy-efficient high-speed CMOS pipelined multiplier," Proc. of IEEE CCE, pp. 460-464, Nov. 2008.

[9] Yung-chin Liang, Ching-ji Huang and Wei-bin Yang, "A 320-MHz 8bit x 8bit pipelined multiplier in ultra-low supply voltage," Proc. of IEEE A-SSCC, pp. 73-76, Nov. 2008.

[10] S. B. Tatapudi and J. G. Delgado-Frias, "Designing pipelined systems with a clock period approaching pipeline register delay," Proc. of IEEE MWSCAS, vol. 1, pp. 871-874, Aug. 2005.

[11] A. D. Booth, "A signed binary multiplication technique," Quarterly J. Mechanical and Applied Math, vol. 4, pp.236-240, 1951.

[12] M. D. Ercegovac and T. Lang, *Digital Arithmetic*, Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 2003.

1

2