A Two-Speed, Radix-4, Serial–Parallel Multiplier

Abstract:

In this paper, we present a two-speed, radix-4, serial-parallel multiplier for accelerating applications such as digital filters, artificial neural networks, and other machine learning algorithms. Our multiplier is a variant of the serial–parallel (SP) modified radix-4 Booth multiplier that adds only the nonzero Booth encodings and skips over the zero operations, making the latency dependent on the multiplier value. Two sub circuits with different critical paths are utilized so that throughput and latency are improved for a subset of multiplier values. The multiplier is evaluated on an Intel Cyclone V field-programmable gate array against standard parallel–parallel and SP multipliers across four different process–voltage–temperature corners. We show that for bit widths of 32 and 64, our optimizations can result in a 1.42×–3.36× improvement over the standard parallel Booth multiplier in terms of area–time depending on the input set.

Enhancement of this project :

- To Design a Two Speed Radix-4 Serial Parallel Booth Multiplier using SQRT Carry select Adder at 4-bit, 8-bit, 16-bit, 32-bit, 64-bit and 128-bit. Finally compared this all the parameters in terms of area, delay and power.

Proposed Title ;

FPGA Implementation of 128-Bit Two Speed Radix-4 Serial Parallel Booth Multiplier using SQRT Carry Select Adder

Proposed Abstract:

In a recent all digital signal processing and machine learning applications will have priority one is multiplication its most important to dictate the area, delay, power and improve overall performance with parallel implementations. In this case these number of multiplication will have number of arithmetic additions and subtractions it will take more logic sizes with critical paths and more power consumptions. Due to resolve this problem here, this proposed work will present optimization of radix-4 multiplication circuits has been extensive using modified booth algorithm with SQRT carry select adder to improve overall performance and reduce logic sizes with critical paths compared to Wallace tree and

DADDA Multiplier. Finally this work will present in Verilog HDL, and synthesize in Xilinx FPGA and proved comparisons terms of area, delay and power.

Existing System:

MULTIPLICATION is arguably the most important primitive for digital signal processing (DSP) and machine learning (ML) applications, dictating the area, delay, and overall performance of parallel implementations. The work on the optimization of multiplication circuits has been extensive, however, the modified Booth algorithm at higher radixes in combination with Wallace or Dadda tree has generally been accepted as the highest performing implementation for general problems. In digital circuits, multiplication is generally performed in one of three ways: 1) parallel–parallel; 2) serial–parallel (SP); and 3) serial–serial. Using the modified Booth algorithm, we explore an SP two-speed multiplier (TSM) that conditionally adds the nonzero encoded parts of the multiplication and skips over the zero encoded sections.

In DSP and ML implementations, reduced precision representations are often used to improve the performance of a design, striving for the smallest possible bit width to achieve a desired computational accuracy. Precision is usually fixed at design time, and hence, any changes in the requirements necessitate that further modification involves redesign of the implementation. In cases where a smaller bit width would be sufficient, the design runs at a lower efficiency since unnecessary computation is undertaken. To mitigate this, mixed-precision algorithms attempt to use a lower bit width some portion of time, and a large bit width when necessary. These are normally implemented with two data paths operating at different precisions.

This paper introduces a dynamic control structure to remove parts of the computation completely during runtime. This is done using a modified serial Booth multiplier, which skips over encoded all-zero or all-one computations, independent of location. The multiplier takes all bits of both operands in parallel and is designed to be a primitive block which is easily incorporated into existing DSPs, CPUs, and GPUs. For certain input sets, the multiplier achieves considerable improvements in computational performance. A key element of the multiplier is that sparsity within the input set and the internal binary representation both lead to performance improvements. The multiplier was tested using field-programmable gate array (FPGA) technology, accounting for four different process–voltage–temperature (PVT) corners. The main contributions of this paper are as follows.

1) The first serial modified Booth multiplier where the datapath is divided into two subcircuits, each operating with a different critical path.

2) Demonstrations of how this multiplier takes advantage of particular bit-patterns to perform less work; this results in reduced latency, increased throughput, and superior area–time performance than conventional multipliers.

3) A model for estimating the performance of the multiplier and evaluation of the utility of the proposed multiplier via an FPGA implementation.

Carry Select Adder :

The conventional CSLA consists of one sum and carry generation unit and sum and carry selection unit as shown in Fig. 4. Sum and carry generation unit can be composed of two ripple carry adders one with carry input zero and other with carry input one as shown in fig.5. Where n is the adder bit width. An n-bit RCA can be composed using half-sum generator (HSG), half-carry generator (HCG), full-sum generator (FSG), full-carry generator (FCG) (shown in fig.6). The RAC-I and RAC-2 generates n-bit sum (S0 and S1) and carry out (C0out and Clout) corresponds to input-carry (Cin=0 and Cin=1) respectively.

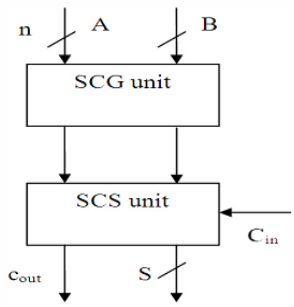


Figure 1 : structure of conventional CSLA

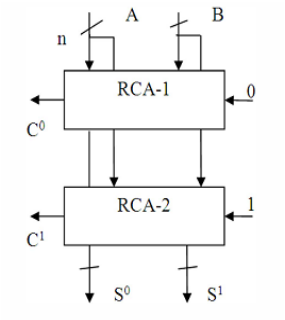


Figure 2 : Structure of SCG unit

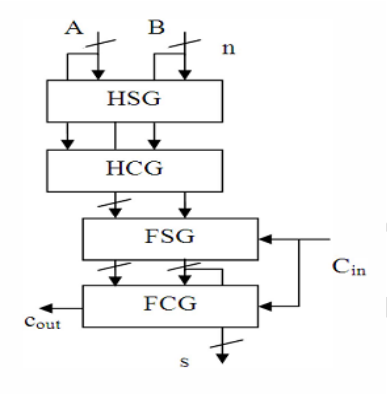


Figure 3 : Structure of Ripple Carry Adder

1.1.1 Logic expression of BEC-based CSLA:

BEC-based CSLA consists of binary to excess-l converter in the place of RCA-2 in conventional CSLA as shown in Fig.7. The RCA is same as that of RAC-l in the conventional CSLA; it calculates n-bit sum { s^0_1 (i)} and carry-out {C^0_{out}} corresponds to C_{in} = 0. Inputs to the BEC unit is {s^0_1 (i), CO out} and output is (n+ 1)-bit excess-l code.

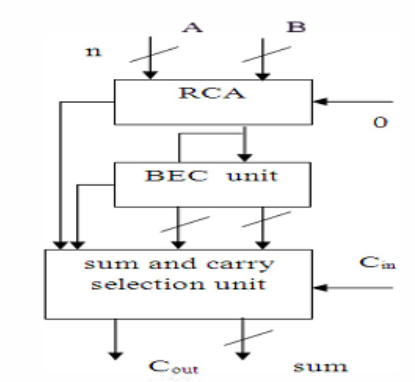


Figure 4 : Structure of BEC-based CSLA

Disadvantages:

- More logic Size and power consumptions
- Number of critical path will take more
- More number iterations

Proposed System:

In a recent all digital signal processing and machine learning applications will have priority one is multiplication its most important to dictate the area, delay, power and improve overall performance with parallel implementations. In this case these number of multiplication will have number of arithmetic additions and subtractions it will take more logic sizes with critical paths and more power consumptions. Due to resolve this problem here, this proposed work will present optimization of radix-4 multiplication circuits has been extensive using modified booth algorithm with SQRT carry select adder to improve overall performance and reduce logic sizes with critical paths compared to Wallace tree and DADDA Multiplier. Finally this work will present in Verilog HDL, and synthesize in Xilinx FPGA and proved comparisons terms of area, delay and power.

Multiplication

Multiplication is a critical primitive that often dictates the performance of large DSP applications. Sze et al. Noted that the majority of hardware optimizations for ML isfocused on reducing the cost of the multiply and accumulate operations. Hence, careful construction of the compute unit, with a focus on

multiplication, leads to the largest performance impact. This section presents an algorithm for the multiplication of unsigned integers followed by its extension to signed integers.

Let x and y be the multiplicand and the multiplier, represented by n digit vectors X and Y in a radix-r conventional number system. The multiplication operation produces p = x × y, where p is represented by the 2n digit vector P. Multiplication is described as

$$p = x\sum_{i=0}^{n-1} Y_i r^i = \sum_{i=0}^{n-1} r^i x Y_i. \qquad (1)$$

Equation (1) can be implemented by first computing the n xri Yi terms followed by the summation. Computation of the ith term involves a i-position left shift of X and the multiplication of a single radix-r digit Yi . This single radix-r digit multiplication is a scaling factor of the ith digit in the digit vector set. In the case of radix-2, this is either 0 or 1. Performing the computation, in this manner, lends itself to a combinational or parallel multiplication unit.

The same computation can be expressed recursively

$$\begin{aligned} p[0] &= 0 \\ p[j+1] &= r^{-1}(p[j] + r^n x Y_j) \quad j = 0, 1, \ldots, n-1 \\ p &= p[n]. \end{aligned} \qquad (2)$$

Expanding this recurrence results in product p[n] = x × y in n steps. Each time step j consists of a multiplication of x by a radix-r digit, Yj , similar to (1). This is followed by a digit left shift and accumulated with the result from the previous time step p[j]. The recurrence is finished with a one digit right shift. It is expressed, in this manner, to ensure that the multiplication can proceed from the least significant digit of the multiplier y, while retaining the same position with respect to the multiplicand x. An example is given in Fig. 1.

$n = 4$	$x = 13$ (X = 1101) $y = 9$ (X = 1001)
p[0]	0000
2^0xY_0	1101
p[1]	11101
2^1xY_1	0000
p[2]	111101
2^2xY_2	0000
p[3]	1111101
-2^3xY_3	0011
p[4]	01110101 = 117

Figure 5: Unsigned two's complement multiplication p = x × y, where x is the multiplicand, y is the multiplier, and X and Y are their respective n = 4 digit vectors in the radix-2 conventional number system.

Equation (1) can be extended to the signed, two's complement system through the incorporation of a sign bit for the multiplier y

$$y = -Y_{n-1}2^{n-1} + \sum_{0}^{n-2} Y_i 2^i \qquad (3)$$

and substituting it into (1). The new expression is given by

$$p = \sum_{i=0}^{n-2} x Y_i r^i - x Y_{n-1} 2^{n-1}. \qquad (4)$$

The negation of x (−x) is performed by flipping all of the bits [bf (1101) = 0010] then adding a single bit in the least significant position (0010 + 1 = 0011).

Multiplier Optimizations

There has been a rich history of ingenious optimizations for the efficient hardware implementation of multiplication, with the multitude of conventional techniques being reviewed in computer arithmetic textbooks. In particular, the signed Booth algorithm was proposed, and the commonly used modified Booth algorithm, presented in Section II, in 1961 .

Recent work has focused on static reordering of the computation or new layouts for the multiplication hardware on FPGAs. Rashidi et al. Proposed a low-power and lowcost shift/add multiplexer-based signed Booth multiplier for a Xilinx Spartan-3 FPGA. The authors used low-power structures, mainly a multiplexer-based Booth encoder with signed shifter blocks and a multiplexer-based Manchester adder. At 50 MHz, the design consumes 58 mW with a total latency of 160 nsec. Devi et al. Focused on a fully combinatorial multiplier design which used custom carry select adders to reduce power consumption by 3.82% and 30% compared to standard ripple carry and carry select adders, respectively. Two contributions were made: a multistage partitioning approach which reduces the overall gate count, and a splitting clock method to reduce the power of the final accumulation. Our work is orthogonal to both works as the same optimizations and structures could be used with our TSM.

Kalivas et al. Described a new bit serial–serial multiplier capable of operating at 100% efficiency. During standard bit serial–serial computation, zero bits are added into the input pipeline between successive inputs words to allow time for the most-significant bits of the product to be produced. Kalivas et al. Removed these bits by adding an additional shift register connected to a secondary output which allows for the most-significant bits of the previous product to be produced while the least significant bits (LSBs) of the current product are produced. This paper differs from our own in two important areas; first, our multiplier is an SP multiplier using the radix-4 Booth algorithm. Second, our multiplier can operate at >100% efficiency since computation is effectively skipped, completing the multiplication in a faster than expected time.

Other work has focused on specialized multiplication structures for the Galois field multiplication. Ten different multiplier alternatives are explored and compared to a reference architecture. The different strategies for combining integer and the Galois field multiplication show area savingsup to 20% with only a marginal increase in delay and an increase in power consumption of 25%.

ncrease in power consumption of 25%. Furthermore, Rashidi proposed a modified retiming serial multiplier for finite impulse response (FIR) digital filters based on ring topologies. The work involved additional logic which allowed for modification of the scheduling of the FIR filter computation, allowing the number of compute cycles to be reduced from 32 to 24. To further improve the performance of the FIR filter computation, the author proposed a high-speed logarithmic carry look ahead adder to work in combination with a carry save adder.

While the TSM is suited for ML and applications with high degrees of sparsity, it differs from the previous research in that the multiplier performs standard signed multiplication and can be used in any application. Our contribution is a new control structure for performing multiplication that dynamically avoids unnecessary computation.

Reduced Precision Multiplication for Neural Networks

The most comparable work to this multiplier is the parallel– serial, or shift-add, multiplier. As described in (2), the product p is iteratively calculated by examining individual bits of X each cycle and accumulating a scaled Y.

Recent work in a bit and digit serial multiplication for FPGAs has focused on online arithmetic and efficient mapping of the algorithms to the FPGA architecture. Shi et al. Analyzed the effect of

overclocking radix-2 online arithmetic implementations and quantified the error introduced by timing violations. They found a significant reduction in error for DSP-based applications compared with conventional arithmetic approaches. Zhao et al. Presented a method for achieving arbitrary precision operations utilizing the on-chip block RAMs to store intermediate values.

In the domain of neural networks, Judd et al. Presented a bit-serial approach for reduced precision computation. They showed a 1.3×–4.5× performance improvement over classical approaches as their arithmetic units only perform the necessary computation for the particular bit width.

Radix-4 Booth Multiplication

This section reviews the radix-4 Booth algorithm, an extension to the parallel–serial multiplier. This computes x × y where x and y are the n bit two's complement numbers (the multiplicand and multiplier respectively); producing a 2n two's complement value in the product p. The multiplication algorithm considers multiple digits of Y at a time and is computed in N partitions where

$$N = \left\lfloor \frac{n+2}{2} \right\rfloor. \qquad (5)$$

An equation describing the computation is given by

$$p = (Y_1 + Y_0)x + \sum_{i=1}^{N} 2^{2i-1}(Y_{2i+1} + Y_{2i} - 2Y_{2i-1})x. \quad (6)$$

Y denotes the length-N digit vector of the multiplier y. The radix-4 Booth algorithm considers three digits of the multiplier Y at a time to create an encoding e given by

$$e_i = Y_{2i+1} + Y_{2i} - 2Y_{2i-1} \qquad (7)$$

where i denotes the ith digit. As illustrated in Table I, apart from Yi+2Yi+1Yi = 000 and Yi+2Yi+1Yi = 111 which results in a 0, the multiplicand is scaled by either 1, 2, −2, or −1 depending on the encoding.

This encoding ei is used to calculate a partial product Partial Producti by calculating

$$Partial Product_i = e_i x = (Y_{2i+1} + Y_{2i} - 2Y_{2i-1})x. \quad (8)$$

This Partial Product is aligned using a left shift (22i−1) and the summation is performed to calculate the final result p. Since the Y−1 digit is nonexistent, the 0th partial product Partial Product0 = (Y1 +Y0)x.

A serial (sequential) version of the multiplication is performed by computing each partial product in N cycles

$$
\begin{aligned}
p[0] &= 2^{n-2}(Y_1 + Y_0)x \\
p[j+1] &= 2^{-2}(p[j] + 2^n(Y_{2j+1} + Y_{2j} - 2Y_{2j-1})x), \\
&\qquad\qquad j = 1, \ldots, N-1 \quad (9) \\
p &= p[N].
\end{aligned}
$$

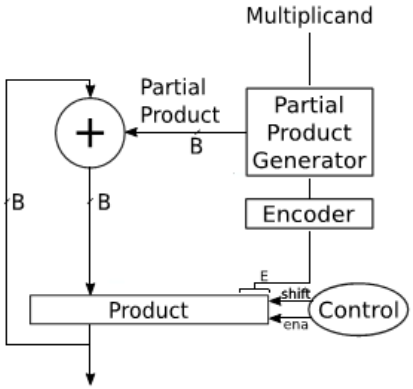


Figure 6: n bit serial multiplier.

Fig. 2. Two optimizations are performed to allow for better hardware utilization. First, the product p is assigned the multiplier y (p = y), this removes the need to store y in a separate register and utilizes the n LSBs of the p register. Consequently, as the product p is shifted right (p = sra(p, 2)), the next encoding ei can be calculated from the three LSBs of p. The second optimization removes the realignment left shift of the partial product (2n) by accumulating the Partial Product to the n most-significant bits of the product p (P[2∗ B −1 : B]+ = Partial Product).

Two-Speed Multiplier

TSM which is an extension to the serial Booth multiplication algorithm and implementation. The key change is to partition the circuit into two paths; each having critical paths, τ and Kτ , respectively (see Fig. 3). The multiplier is clocked at a frequency of (1/τ), where the Kτ region is a fully combinatorial circuit with a delay of Kτ .

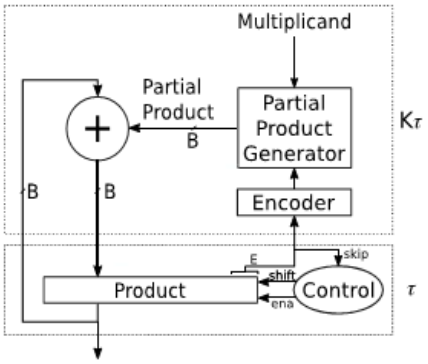


Figure 7: n bit TSM.

As illustrated in Algorithm 2, before performing the addition, the encoding, e (the three LSBs of the product) is examined and a decision is made between two cases: 1) the encoding and Partial Product are zero and 0x, respectively, and 2) the encoding is nonzero. These two cases can be distinguished by generating

$$\text{skip} = \begin{cases} 1, & \text{if } P[2:0] \in \{000, 111\} \\ 0, & \text{otherwise.} \end{cases} \qquad (10)$$

When skip = 1 only the right shift and cycle counter accumulate need to be performed, with a critical path of τ . In the case of a nonzero encoding (skip = 0), the circuit is clocked K^- times at τ . This ensures sufficient propagation time within the adder and partial product generator, allowing the product register to honor its timing constraints. Hence, the total time T taken by the multiplier can be expressed as (11), where N is defined by (5), and O is the number of nonzero encodings in the multiplier's Y digit vector

$$T(O) = (N - O)\tau + O\bar{K}\tau. \qquad (11)$$

The time taken to perform the multiplication is dependent on the encoding of the bits within the multiplier y. The upper and lower bound for the total execution time occurs when O = N and O = 0, respectively. From (11), the max and min are

$$N\tau \le T \le N\bar{K}\tau. \qquad (12)$$

The input that results in the minimum execution time is when y = 0. In this case, all bits within the multiplier are 0, and every three LSB encoding results in a 0x scaling and O = 0. There are a few input combinations that result in the worst case, O = N. One case would be a number of alternating 0 and 1, i.e., 1010101..10101..10101. In this case, each encoding results in a nonzero Partial Product.

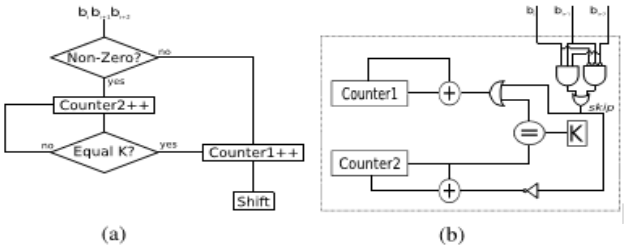


Figure 8: (a) Controller flowchart. (b) Control circuit.

Control

As shown in Fig. 4(a) and (b), the control circuit consists mainly of: one log2(N) accumulator, one log2(K^-) accumulator, three gates to identify the nonzero encodings, and a comparator. Counter2 is responsible for counting the number of cycles needed for the addition without violating anytiming constraints, i.e., K^- . When the encoding is nonzero, Counter2 is incremented. Counter1 accumulates the number of encodings that have been processed. As shown in Section III, the number of cycles needed to complete a single multiplication is N, therefore, the accumulator and Counter1 needs to be log2(N) bits wide. Counter1 is incremented when the comparator condition has been met, Counter2 = K^- , or a zero encoding is encountered. When Counter.

The control needs to distinguish between the zero and nonzero encodings. It contains a three-gate circuit, performing (10); taking in the three LSBs of the multiplier y. Two cases of zero encoding exist. The three gates are designed to identify these nonzero encodings; an inverter is connected to the accumulator of Counter2, incrementing, in these cases.

Bit Representation	Action	Time	PartialProduct
1111010001000	skip	τ	$0x \times 2^0$
1111010010	add	$\tau + \bar{K}\tau$	$1x \times 2^2$
1111010	skip	$2\tau + \bar{K}\tau$	$0x \times 2^4$
11110	add	$2\tau + 2\bar{K}\tau$	$1x \times 2^6$
1110	add	$2\tau + 3\bar{K}\tau$	$-1x \times 2^8$
111	skip	$3\tau + 3\bar{K}\tau$	$0x \times 2^{10}$

Figure 9: Control example.

Fig. 5 provides an example of the control operating in the multiplier and the time taken to perform the multiplication. Each cycle, the three LSBs of the multiplier y are examined and an action is generated based on their encoding. Since 000 results in a 0x partial product, the first action is a "skip" and only the right shift is performed in τ time. The next threebit encoding, 010, is examined and results in a 1x partial product. This generates the "add" action in which Counter2 is accumulated to K^- and the product register is held constant. After K^- τ time, the value stored in the register has had enough time to propagate through the adder and the result is latched in the product register without causing timing violations. The multiplier continues operating in this fashion until all bits of y have been processed and the final result produced. In Fig. 5, the total time is 3τ + 3K^- τ since there are three "skips" and three "adds."

Set Analysis and Average Delay

Given an input set D of length l and a function f (y) [given by (13)] that calculates the number of nonzero encodings for a given multiplier y, the probability distribution p of encountering a particular encoding can be calculated by Algorithm 3

$$f(y) = \neg(Y_1 \oplus Y_0) + \sum_{i=1}^{N} (\neg(Y_{2i+1} \oplus Y_{2i}) \wedge \neg(Y_{2i} \oplus Y_{2i-1})) \quad (13)$$

where ¬, ⊕, and ∧ are the logical "NOT," "XOR," and "AND" symbols, respectively.

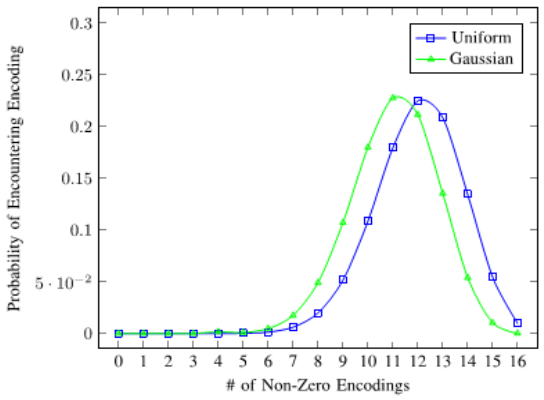


Figure 10: p(i) 32-bit distribution.

Fig. 6 shows the Gaussian and uniform encoding probability distribution for 32 bits. There are significantly less numbers in the lower, nonzero encoding region compared with the higher, nonzero encoding region, resulting in increased computation time. However, as discussed in Section V, for other workloads, the distributions can shift and change depending on the problem and optimization techniques used.

Using the probability p, the average delay of the multiplier can be calculated using the following equation:

$$\mathbb{T} = \frac{1}{N}\sum_{i=0}^{N} p(i)T(i) \qquad (14)$$

where T (i) is calculated using (11) and p(i) denotes the probability of encountering an encoded number with i nonzeros.

Timing

During standard timing analysis, the Kτ path would cause a timing violation for the circuit operating at frequency (1/τ). There are two ways to address this issue. The first involves a standard “place and route” of each individual multiplier as it is instantiated in the design. An additional timing constraint is included to address the otherwise violated Kτ path, allowing timing driven synthesis and placement to achieve the best possible layout. The second option is to create a reference post—“place and route” block—that is used whenever the multiplier is instantiated. This ensures each multiplier has the same performance and is placed in exactly the same configuration.

Proposed Carry Select Adder:

The proposed CSLA adder is a variable length CSLA adder and based on this proposed CSLA we are creating the 256 bit sqrt- CSLA structure. This sqrt CSLA adder is reducing the delay of the architecture. The Proposed CSLA is design with variable length inputs, so it’s flexible to different application. The proposed CSLA structure is as shown in Fig.8. It is composed of one half-sum generation (HSG) unit, one full sum generation (FSG) unit, one carry-generation (CG) unit, and one carry-selection (CS) unit. The CG unit composed of two units namely CGO and CG 1 Corresponding to input-carry '0' and '1', respectively. Input to the HSG unit is two n-bit operands A and B and outputs are half-sum (HS) word So and half-carry (HC) word Co of width n-bit each. CG unit receives both So and Co from HSG unit and gives two n-bit full-carry words co, and c1, corresponds to carry-input '0' and '1'

,respectively. The carry selection unit selects final carry based on the Cin from two anticipated carry words C^0_1 and c^1_1 If Cin = 0 then it selects CO I; otherwise it selects c^1_1 Cout is the MSB of c obtained from CS unit and remaining (n-l) LSBs of CS unit are XORed with (n-l) MSBs of half-sum (so) in the FSG unit to obtain final-sum.

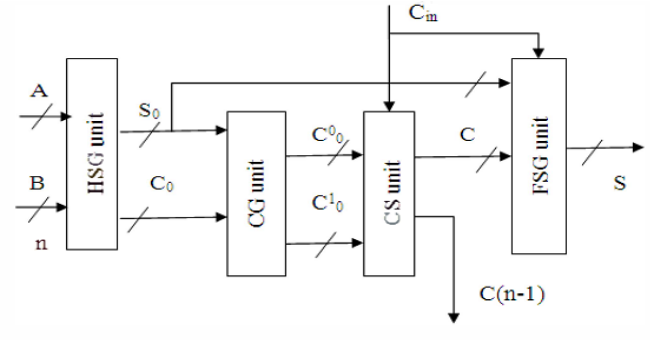


Figure 11: proposed CSLA structure

Where,

HSG - Half-sum generator

S_0 (i) = A (i) XOR B (i)

HCG -Half-carry generator

C_0 (i) = A (i) AND B (i)

FSG -Full-sum generator

S (0) = S0 (0) XOR Cin S (i) = S0 (i) XOR C (i-I)

FCG -Full-carry generator

C^0_1 (i) = C01 (i-l) AND s0 (i) OR c0 (i) for (C^0_1 (0) = 0)

C^1_1 (i) = C11 (i-l) AND s0 (i) OR c0 (i) for (C^1_1 (O) = 1)

Cout = c (n-l)

The SQRT CSLA architecture is shows in fig 9. For the figure we are obtain the proposed CSLA adder used in different length of inputs.

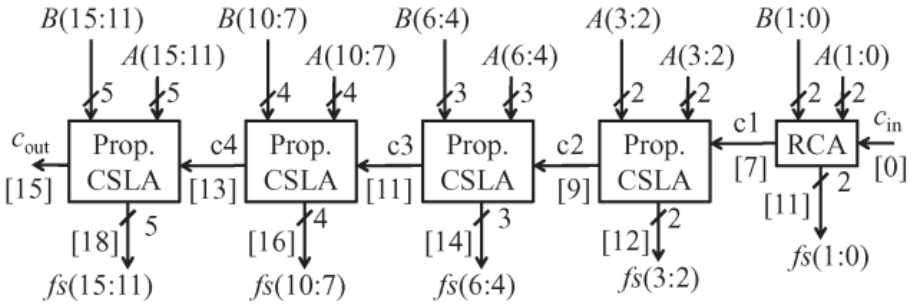
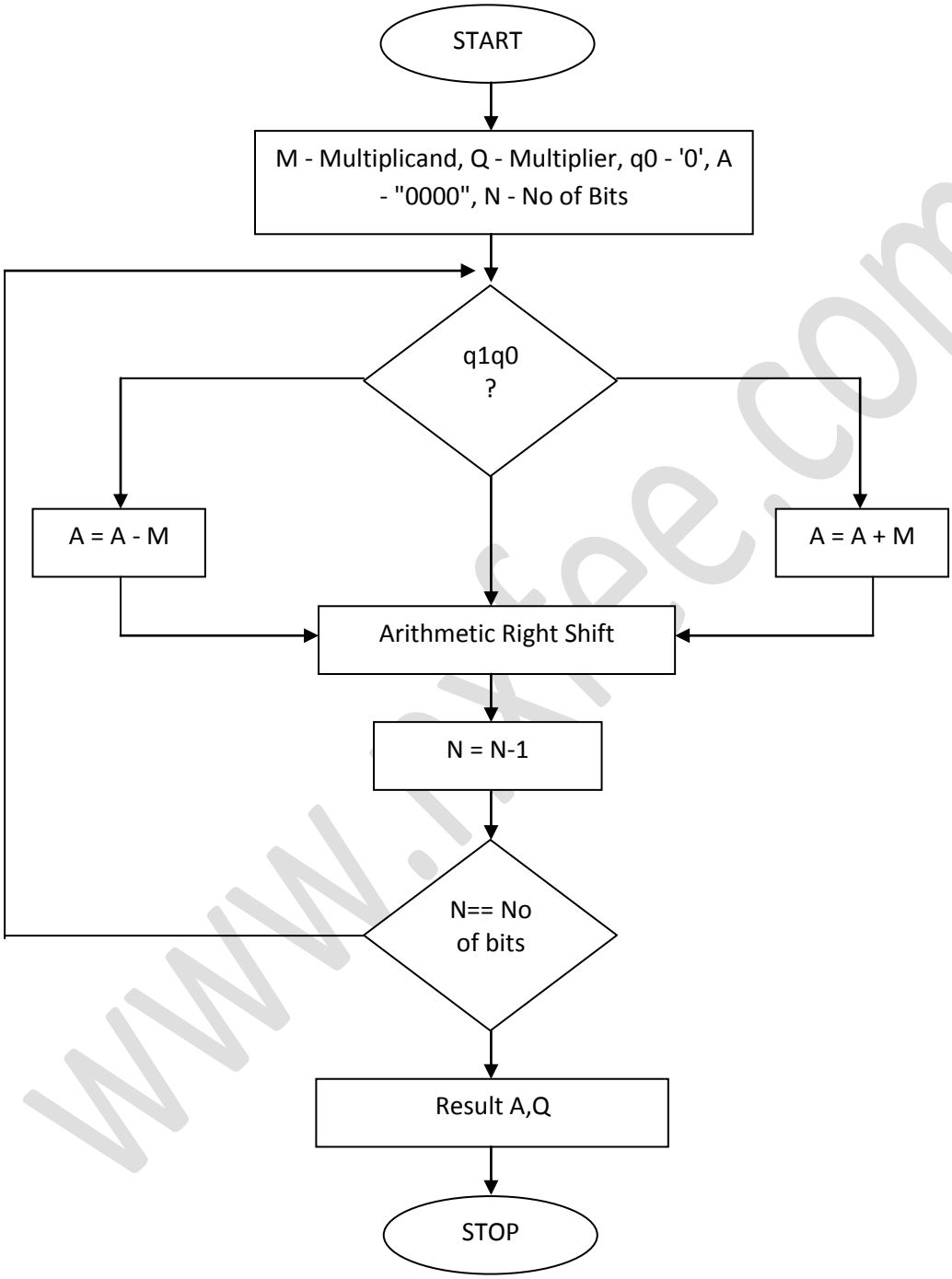


Figure 12: Proposed SQRT-CSLA for n = 16. All intermediate and output signals are labelled with delay (shown in square brackets).

Flow Chart for Booth Algorithm:



Advantages:

- Less logic Size and power consumptions
- Less critical path
- Less number iterations

Literature Survey:

- "An efficient SQRT architecture of Carry Select Adder design by Common Boolean logic", In the design of Integrated Circuits, area occupancy plays a vital role because of increasing the necessity of portable systems. Carry Select Adder (CSLA) is one of the fastest adders used in many data-processing processors to perform fast arithmetic functions. In this paper, an area-efficient carry select adder by sharing the common Boolean logic term (CBL) is proposed. After logic simplification and sharing partial circuit, only one XOR gate and one inverter gate in each summation operation as well as one AND gate and one inverter gate in each carry-out operation are needed. Through the multiplexer, the correct output is selected according to the logic states of the carry in signal. Based on this modification a new architecture has been developed and compared with the regular and modified Square-root CSLA (SQRT CSLA) architecture. The modified architecture has been developed using Binary to Excess-1 converter (BEC). The proposed architecture has reduced area and delay as compared with the regular SQRT CSLA architecture. The result analysis shows that the proposed SQRT CSLA structure is better than the regular SQRT CSLA.
- "Arithmetic optimization using carry-save-adders", Carry-save-adder (CSA) is the most often used type of operation In Implementing a fast computation of arithmetics of register-transfer level design in industry. This paper establishes a relationship between the properties of arithmetic computations and several optimizing transformations using CSAs to derive consistently better qualities of results than those of manual implementations. In particular, we introduce two important concepts, operation-duplication and operation-split, which are the main driving techniques of our algorithm for achieving an extensive utilization of CSAs. Experimental results from a set of typical arithmetic computations found in industry designs indicate that automating CSA optimization with our algorithm produces designs with significantly faster timing and less area.

- “Optimal Allocation of Carry-Save-Adders in Arithmetic Optimization”, Carry-save-adder(CSA) is one of the most widely used schemes for fast arithmetic in industry. This paper provides a solution to the problem of finding an optimal-timing allocation of CSAs. Specifically, we present a polynomial time algorithm which finds an optimal-timing CSA allocation for a given arithmetic expression. In addition, we extend our result for CSA allocation to the problem of optimizing arithmetic expressions across the boundary of design hierarchy by introducing a new concept, called auxiliary ports. Our algorithm can be used to carry out the CSA allocation step optimally and automatically, and this can be done within the context of a standard HDL synthesis environment.
- Approximate computing has recently emerged as a promising approach to energy-efficient design of digital systems. Approximate computing relies on the ability of many systems and applications to tolerate some loss of quality or optimality in the computed result. By relaxing the need for fully precise or completely deterministic operations, approximate computing techniques allow substantially improved energy efficiency. This paper reviews recent progress in the area, including design of approximate arithmetic blocks, pertinent error and quality measures, and algorithm-level techniques for approximate computing.
- Current microprocessors employ a global timing reference to synchronize data transfer. A synchronous system must know the maximum time needed to compute a function, but a circuit usually finishes computation earlier than the worst-case delay. The system nevertheless waits for the maximum time bound to guarantee a correct result. As a first step in achieving variable pipeline delays based on data values, approximation circuits can increase clock frequency by reducing the number of cycles a function requires. Instead of implementing the complete logic function, a simplified circuit mimics it using rough calculations to predict results. The results are correct most of the time, and simulations show improvements in overall performance in spite of the overhead needed to recover from mistakes.
- The occurrence of errors are inevitable in modern VLSI technology and to overcome all possible errors is an expensive task. It not only consumes a lot of power but degrades the speed performance. By adopting an emerging concept in VLSI design and test-error-tolerance (ET), we managed to develop a novel error-tolerant adder which we named the Type II (ETAII). The circuit to some extent is able to ease the strict restriction on accuracy to achieve tremendous improvements in both the power consumption and speed performance. When compared to its conventional counterparts, the proposed ETAII is able to achieve more than 60% improvement

in the power-delay product (PDP). The proposed ETAII is an enhancement of our earlier design, the ETAI, which has problem adding small number inputs.

- The conventional digital hardware computational blocks with different structures are designed to compute the precise results of the assigned calculations. The main contribution of our proposed Bio-inspired Imprecise Computational blocks (BICs) is that they are designed to provide an applicable estimation of the result instead of its precise value at a lower cost. These novel structures are more efficient in terms of area, speed, and power consumption with respect to their precise rivals. Complete descriptions of sample BIC adder and multiplier structures as well as their error behaviors and synthesis results are introduced in this paper. It is then shown that these BIC structures can be exploited to efficiently implement a three-layer face recognition neural network and the hardware defuzzification block of a fuzzy processor.

References:

[1] A. D. Booth, "A signed binary multiplication technique," Quart. J. Mech. Appl. Math., vol. 4, no. 2, pp. 236–240, 1951.

[2] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs. New York, NY, USA: Oxford Univ. Press, 2000.

[3] M. D. Ercegovac and T. Lang, Digital Arithmetic (Morgan Kaufmann Series in Computer Architecture and Design). San Mateo, CA, USA: Morgan Kaufmann, 2004.

[4] B. Dinesh, V. Venkateshwaran, P. Kavinmalar, and M. Kathirvelu, "Comparison of regular and tree based multiplier architectures with modified booth encoding for 4 bits on layout level using 45 nm technology," in Proc. Int. Conf. Green Comput. Commun. Elect. Eng., Mar. 2014, pp. 1–6.

[5] O. L. MacSorley, "High-speed arithmetic in binary computers," Proc. IRE, vol. PROC-49, no. 1, pp. 67–91, Jan. 1961.

[6] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication," IEEE Trans. Comput., vol. C-24, no. 10, pp. 1014–1015, Oct. 1975, doi: 10.1109/T-C.1975.224114.

[7] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture, Oct. 2016, pp. 1–12.

[8] G. C. T. Chow, W. Luk, and P. H. W. Leong, “A mixed precision methodology for mathematical optimisation,” in Proc. IEEE 20th Int. Symp. Field-Program. Custom Comput. Mach., Apr./May 2012, pp. 33–36.

[9] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. Leong, and D. B. Thomas, “A mixed precision Monte Carlo methodology for reconfigurable accelerator systems,” in Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays, 2012, pp. 57–66.

[10] S. J. Schmidt and D. Boland, “Dynamic bitwidth assignment for efficient dot products,” in Proc. Int. Conf. Field Program. Log. Appl., Sep. 2017, pp. 1–8.

[11] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, “Hardware for machine learning: Challenges and opportunities,” CoRR, vol. abs/1612.07625, Dec. 2016. [Online]. Available: https://arxiv.org/abs/1612.07625

[12] B. Rashidi, S. M. Sayedi, and R. R. Farashahi, “Design of a low-power and low-cost Booth-shift/add multiplexer-based multiplier,” in Proc. Iranian Conf. Elect. Eng. (ICEE), May 2014, pp. 14–19.

[13] P. Devi, G. P. Singh, and B. Singh, “Low power optimized array multiplier with reduced area,” in High Performance Architecture and Grid Computing, A. Mantri, S. Nandi, G. Kumar, and S. Kumar, Eds. Berlin, Germany: Springer, 2011, pp. 224–232.