View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Exeley Inc.

International Journal of Advanced Network, Monitoring and Controls Volume 04, No.02, 2019

Radix-8 Design Alternatives of Fast Two Operands Interleaved Multiplication with Enhanced Architecture

With FPGA implementation & synthesize of 64-bit Wallace Tree CSA based Radix-8 Booth Multiplier

Mohammad M. Asad
King Faisal University, Department of Electrical Engineering, Ahsa 31982, Saudi Arabia
e-mail: asadmosab@gmail.com

Qasem Abu Al-Haija
Department of Computer Information and Systems Engineering
Tennessee State University, Nashville, USA
e-mail: qabualha@my.tnstate.edu

Ibrahim Marouf
King Faisal University, Department of Electrical Engineering, Ahsa 31982, Saudi Arabia
e-mail: i.marouf@outlook.com

***Abstract*—In this paper, we proposed different comparable reconfigurable hardware implementations for the radix-8 fast two operands multiplier coprocessor using Karatsuba method and Booth recording method by employing carry save (CSA) and kogge stone adders (KSA) on Wallace tree organization. The proposed designs utilized ALTERA Cyclone IV FPGA family with target chip device EP4CGX – 22CF19C7 along with Quartus II simulation package. Also, the proposed designs were synthesized and benchmarked in terms of the maximum operational frequency, the total path delay, the total design area and the total thermal power dissipation. The experimental results revealed that the best multiplication architecture was belonging to Wallace Tree CSA based Radix-8 Booth multiplier (WCBM) which recorded: critical path delay of 14.103 ns, maximum operational frequency of 90.83 MHz, hardware design area (number of logic elements) of 14249 LEs, and total thermal power dissipation estimated as 217.56 mW. Consequently, WCBM method can be efficiently employed to enhance the speed of computation for many multiplication based applications such embedded system designs for public key cryptography.**

Keywords-Cryptography; Computer Arithmetic; FPGA Design; Hardware Synthesis; Kogge-Stone Adder (KSA); Radix-8 Booth Recording; Karatsuba Multiplier; Wallace Tree

I. INTRODUCTION

Recently, the vast promotion in the field of information and communication technology (ICT) such as grid and fog computing has increased the inclination of having secret data sharing over the existing non-secure communication networks. This encouraged the researches to propose different solutions to ensure the safe access and store of private and sensitive data by employing different cryptographic algorithms especially the public key algorithms [1] which proved robust security resistance against most of the attacks and security halls. Public key cryptography is significantly based on the use of number theory and digital arithmetic algorithms.

Indeed, wide range of public key cryptographic systems were developed and embedded using hardware modules due to its better performance and security. This increased the demand on the embedded and System-on Chip (SoC) [2] technologies employing several computers aided (CAD) tools along with the configurable hardware processing units such as field programmable gate array (FPGA) and application specific integrated circuits (ASIC). Therefore, considerable number of embedded coprocessors design were used to replace software based (i.e. programming based) solutions of different applications such as image processors, cryptographic processors, digital filters, low power application such as [3] and others. The major part of designing such processors significantly encompasses the use computer arithmetic techniques in the underlying layers of processing.

Computer arithmetic [4] or digital arithmetic is the science that combines mathematics with computer engineering and deals with representing integers and real values in digital systems and efficient algorithms

DOI: 10.21307/ijanmc-2019-043

for manipulating such numbers by means hardware circuitry and software routines. Arithmetic operations on pairs of numbers x and y include addition ($s = x + y$), subtraction ($d = x - y$), multiplication ($p = x \times y$), and division ($q = x / y$). Subtraction and division can be viewed as operations that undo the effects of addition and multiplication, respectively. Multiplication operation is considered as a core operation that affect the performance of any embedded system. Therefore, the use of fast multiplier units will result in enhancements in the overall performance of the system. Recently, several solutions were proposed for multiplication algorithms while few of them were efficient [5].

A multiplication algorithm [6] is method to find the product of two numbers, i.e. $P = X \times Y$. Multiplication is an essential building block for several digital processors as it requires a considerable amount of processing time and hardware resources. Depending on the size of the numbers, different algorithms are in use. Elementary-school grade algorithm was multiplying each number digit by digit producing partial sum with complexity of $O(n^2)$. [5]For larger numbers, more efficient algorithms are needed. For example, let a and b integers to be multiplied with $n - \text{bit}$ equal to 1k bits, thus $1{,}000{,}000$ $\text{single} - \text{digit}$ multiplications. However, more efficient and practical multiplication algorithms will be discussed in the following subsections.

In this paper, we report on several fast alternative designs for Radix-8 based multiplier unit including: Radix-8 CSA Based Booth Multiplier, CSA Based Radix-8 Booth, Wallace Tree Karatsuba Multiplier, CSA Based Radix-8 Booth, KSA Based Karatsuba Multiplier, CSA Based Radix-8 Booth, With Comparator Karatsuba Multiplier, Sequential 64-Bit CSA Based Radix-8 Booth Multiplier, 64-bit Wallace Tree CSA based Radix-8 Booth multiplier (WCBM). The remaining of this paper is organized as follows: Section 2m discusses the core components of efficient multiplier design. Section 3,provides the proposed design alternatives of Radix-8 based multiplier, Section 4, presents the synthesizing results and analysis, and, finally, Section 5 concludes the paper.

II. CORE DESIGN COMPONENTS-REVIEW

Two operands-multiplication is a substantial arithmetic operation since it plays a major role in the design of many embedded and digital signal processors [7]. Therefore, the efficient design and implementation of a fast multiplier unit is on demand. In this paper, we propose a competitive reconfigurable multiplier design using scalable and efficient modules. Thus, the following subsections reviews the core design components for the proposed multiplier implementation unit.

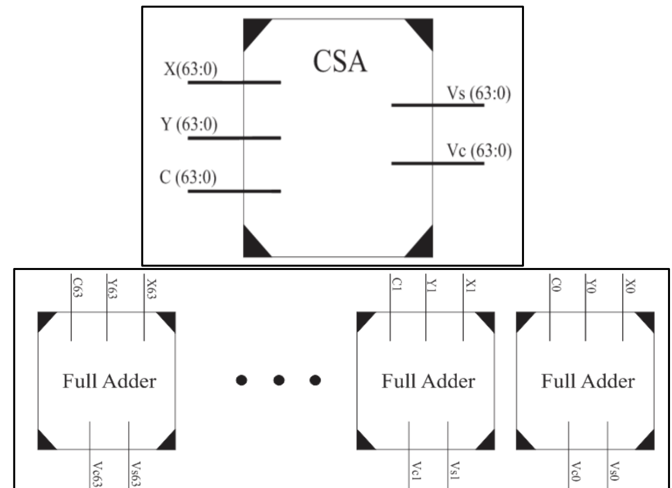


Figure 1. Carry save Adder: (a) Top View Design (b) Internal Architecture

A. *Carry save Adder (CSA)*

CSA [4] is a fast-redundant adder with constant carry path delay regardless of the number of operands' bits. It produces the result as two-dimensional vectors: sum vector (or the partial sum) and carry vector (or partial carry). The advantage of CSA is that the speed is constant regardless the number of bits. However, its area increases linearly with the number of bits. The top view of the CSA unit along with its internal logic design architecture are provided in Fig.1 below.

In this work, we have implemented the CSA adder using VHDL code for different bit sizes ranges from 8-bits through 64-bits [8]. The synthesize results of total delay in (ns) and area in Logic Elements (LEs) were analyzed and reported in [8] and they are illustrated in Fig.2. These results were generated using Quartus II 14.1 software [9], simulated for Altera Cyclone IV DE2 − 115 model [10] and they highly conform theoretical evaluation of CSA operation since the delay time is almost equal for all bits. However, the area is almost double for each number of bits. Also, the timing estimation of 64 − bits CSA was generated via TimeQuest Time Analyzer tool provided in the Quartus II CAD package. Accordingly, the critical path delay is 3.529 ns which is data arrival time while the data delay is only 2.866 ns which provide a frequency of 349 MHz.Finally, to verify the performance of CSA, we have compared it with the well-known Carry LockAhead Adder (CLA) in terms of area and delay. CLA is a carry propagation

adder (CPA) with logarithmic relation between the carry propagation delay and the number of bits in operands.

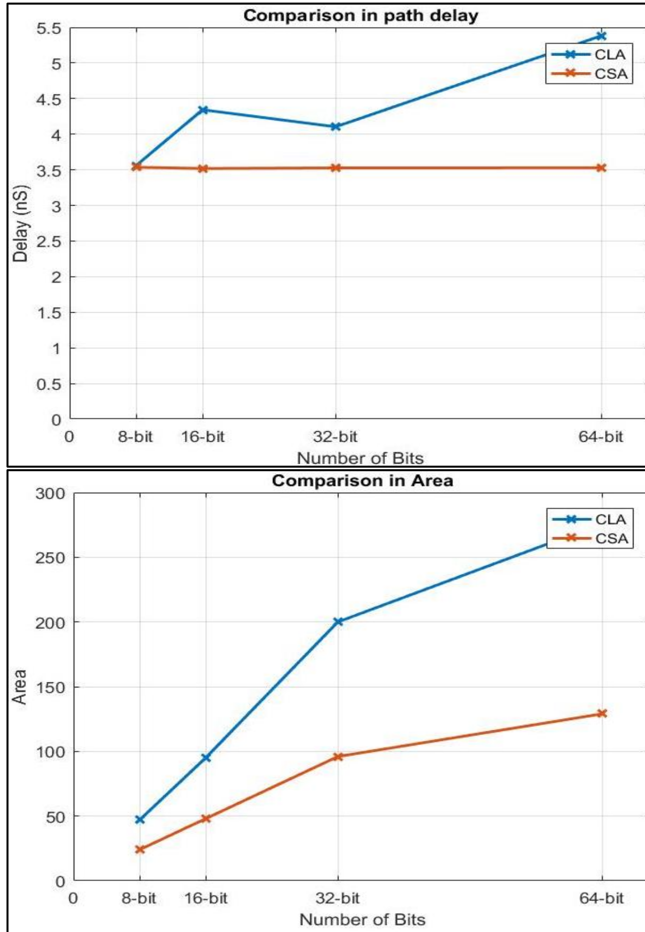


Figure 2. Delay-Area analysis of CSA vs CLA implementations (8–64 bit)

The simulation results of both CSA and CLA is provided in Fig.2 shows that CSA is superior in both Area and speed. It has almost a constant time delay and relatively less area than CLA. Whereas CLA time delay increases as the number of bit increases but not much as the area size.

B. Kogge-Stone Adder (KSA)

KSA is a fast two operands parallel prefix adder (PPAs) [11] that executes addition on parallelized manner. PPAs are just like CLA but with an enhancement in the carry propagation stage (called the middle stage). There are five different variations of PPAs namely: Ladner-Fischer Adder (LFA), Brent-Kung Adder (BKA), Kogge-Stone Adder (KSA), Hans-Carlson Adder (HCA), and Sklansky Adder (SkA). These adders differ by the tree structure design to optimize certain aspects such as, performance, power, area size, and fan in/out.

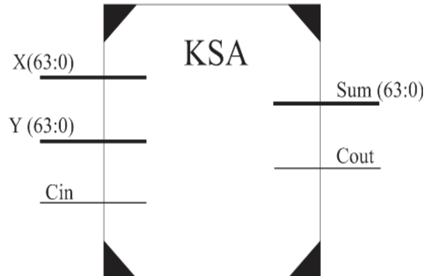
To verify the performance of all PPAs, we have implemented them on FPGA and the experimental results [6] showed that KSA utilizes larger area size to achieve higher performance comparing among all other five PPAs. Thus, we decided to consider KSA as our basic carry propagation adder (CPA) to finalize the redundant results and to build up many other units that are in-need for conventional adder. In short, the simulation results of [6] showed that KSA leading the other adders as it has the smallest time delay with only 4.504. This result is very useful and conforms the theatrical modeling of KSA which has the least number of logic levels. Like all PPAs, KSA functionality consists of three computational stages as illustrated in Fig.3, as follows:

- ***Pre-processing stage*:** The computation of generate and propagate of each bit from A and B are done in this step. These signals are given by the logic equations: $Pi = Ai \text{ xor } Bi$ and $Gi = Ai \text{ and } Bi$
- ***Carry generation network*:** PPA differentiates from each other by the connections of the network. It computes carries of each bit by using generate and propagate signals from previous block. In this block two blocks are defined group generation and propagation (GGP), in addition to group generation only (GGO), as shown in Fig.3. Logic blocks used for the calculation of generate and propagate bits can be describe by the following logic equations: $P_{out} = P_{in1} \cdot P_{in2}$ and $G_{out} = G_{in1} \,||\, (P_{in1} \cdot G_{in2})$, Where the generation group have only logic equation for carry generation: $G_{out} = G_{in2} || (P_{in2} \cdot G_{in1})$.
- ***Post processing (Calculating the Sum):*** This is the last step and is common to all adders of this family (carry look ahead). It involves computation of sum bits. Sum bits are computed by the logic given in: $S_i = P_i \oplus G_{i-1}$. The top view and the internal logic circuit is provided in the Fig.3.

C. Fast Multi-Operands Addition

Addition operation is not commonly used to add two operands only, instead, it is more involved with multiplication and inner product computations [12]. The use of regular two operands adders leads to intermediate results before getting the last answer which affect the performance or the time delay of a system. Therefore, Multi-operand adders are manly studied to reduce this problem. Wallace and Dadda trees [13] are considered as two variations of high-performance multi-operands addition. Fig.4. shows the dot notation to represent the digit positions or

alignments (instead of using the value which is quite useful) for the use of Multi-operand addition in multiplication and inner-product computation.



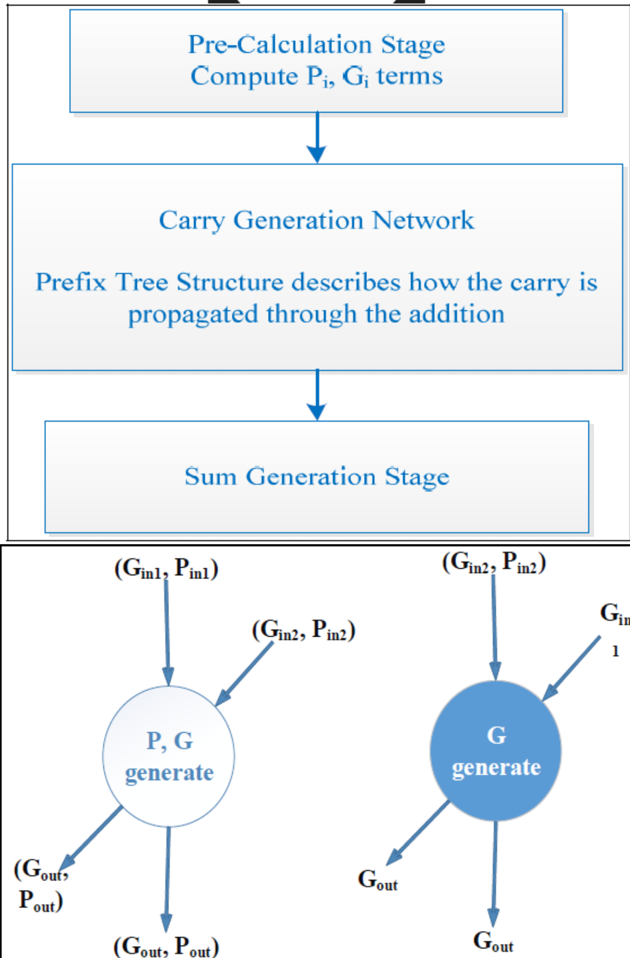


Figure 3. Kogge Stone Adder: (a) Top View Design of KSA (c) KSA Stages (c) Group generation and propagation

In this work, we have adopted a CSA based Wallace tree since it confirmed better operands organization to improve the total addition delay [8]. We have implemented two CSA Wallace Trees: 10-operands addition and 22-operands addition. The structure logic diagram of 10 operands is given in Fig.5. It's clearly seen that the Wallace tree unit is designed behaviorally (FSM is generated).

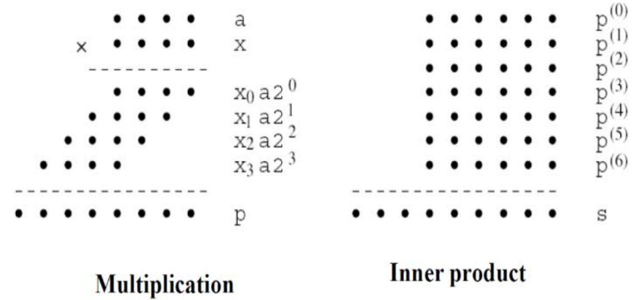


Figure 4. Dot notation of Multi-operand addition for multiplication and inner-product computation

D. *Karatsuba Multiplier*

To enhance the performance of multiplication for large operands (i.e. 1024-bit size), a re-organization process can be adopted for the multiplication operands to utilize the maximum possible parallelism to enhance the multiplication time. Karatsuba algorithm [14] is pipelined multiplication process used mainly to construct the high precision multipliers form multiple small precision multiplier blocks by exploiting the maximum available parallelism between the multiplication blocks. The basic idea of Karatsuba algorithm is illustrated in fig.6 and Karatsuba algorithm can be defined as follows:

Let x, y be integers and B is the base (Radix_2) and $m < n$ where n: the number of digits, then:

1) *Re-write x and y as follows:*
$$x = x_1 2^m + x_0 and y = y_1 2^m + y_0$$
2) *Calculate Product $p = xy$ as follows:*
$$p = z_2 b^2 + z_1 b + z_0, where:$$
$$z_2 = x_1 y_1 , \quad z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0 , \quad z_0 = x_0 y_0 , \quad b = B^m$$

A more efficient implementation of Karatsuba multiplication can be accomplished as:

$$xy = (b^2 + b)z_2 - b(x_1 - x_0)(y_1 - y_0) + (b + 1)z_0$$

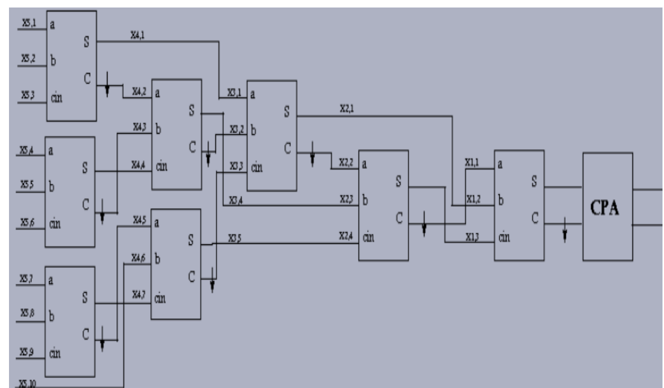


Figure 5. Multi-operand addition for 10 operands.

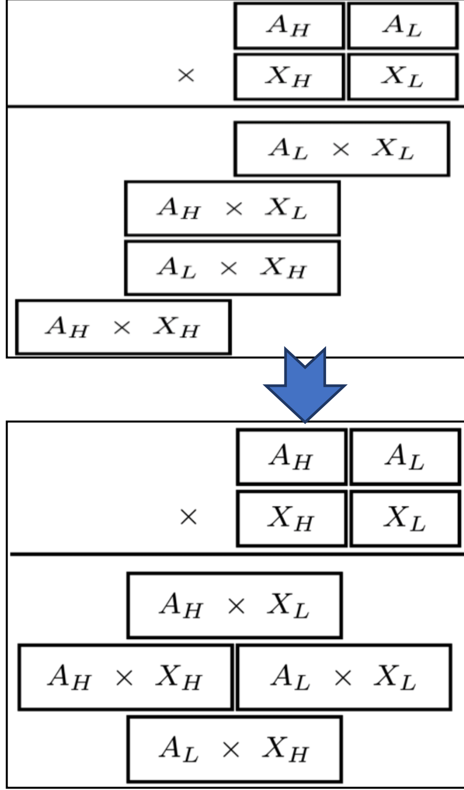


Figure 6. Aligning Partial Products.

E. Magnitude Comparator

The magnitude (or digital) comparator is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number. Like that in binary addition, the efficient comparator can be implemented using G (generate) and P (propagate) signal for comparison. Basically, the comparator involves two 2-bits: $A_1A_0 \& B_1B_0$ can be realized by:

$$B_{Big} = \overline{A_1}B_1 + \overline{(A_1 \oplus B_1)}.(\overline{A_0}B_0) \quad (1)$$

$$\text{EQ} = \overline{(A_1 \oplus B_1)}.\overline{(A_0 \oplus B_0)} \quad (2)$$

For A<B, “BBig, EQ” is “1,0”. For A=B, “BBig, EQ” is “0,1”. Hence, for A>B, “BBig, EQ” is “0,0”. Where BBig is defined as output A less than B (A_LT_B).Comparing Eq. (1) and (2) with carry signal (3):

$$C_{out} = AB + (A \oplus B).C_{in} = G + P.C_{in} \quad (3)$$

Where A & B are binary inputs Cin is carry input, Cout is carry output, and G & P are generate & propagate signals, respectively. Now, after comparing equations (1) & (3), we got:

$$G_1 = \overline{A_1}B_1, \quad EQ_1 = \overline{(A_1 \oplus B_1)}, \quad C_{in} = \overline{A_0}B_0 \quad (4)$$

Cin can be considered as G0. For this, encoding equation is given as:

$$G_{[i]} = \overline{A_{[i]}}B_{[i]} \quad (5)$$

$$EQ_{[i]} = \overline{(A_{[i]} \oplus B_{[i]})} \quad (6)$$

Substituting the two values from equations (5) & (6) in (1) & (2) results in:

$$B_{Big[2j+1:2j]} = G_{[2j+1]} + EQ_{[2j+1]}.G_{[2j]} \quad (7)$$

$$EQ_{[2j+1:2j]} = EQ_{[2j+1]}.EQ_{[2j]} \quad (8)$$

G&P signals can be further combined to form group G & P signals. For instance, for 64-bit comparator, B_{Big}&EQ can be computed as:

$$B_{Big[63:0]} = G_{63} + \sum_{k=0}^{62}\left(G_k \cdot \prod_{m=k+1}^{63} EQ_m\right) \quad (9)$$

$$EQ_{[63:0]} = \prod_{m=0}^{63} EQ_m \quad (10)$$

Fig 7. Shows the complete design of an 8-bit comparator as an example of this techniques where: i= 0…7, j = 0…3.

III. PROPOSED MULTIPLIER DESIGN ALTERNATIVES

Fundamentally, multiplication operation (along with fast addition) is a significant unit in almost all cryptographic coprocessors. For instance, in the design of SSC Crypto-processor[15], the multiplication primarily used to compute the square parameter(p^2), the public key (p^2q) and the modulus (pq). Also, in the design of RSA Crypto-processor, the multiplier is used to compute the modulus (p. q) and the Euler function $\emptyset(n) = (p-1).(q-1)$ [16]. One more example, is the need for fast multipliers at several computation stages of ECC cryptosystem [17]. Indeed, wide range of methods have been proposed to address the efficient design of fast two operands arithmetic

multiplier. In this paper, we have spent an extensive time to design an efficient multiplier by trying several variations of different multiplier design specifications. The first design was the implementation of Radix-8 Booth Encoding Multiplier. Then, we tried many variations to employ this multiplier with different design methods. In the next subsections, we provide six design alternatives of the proposed multiplier to come up with the most cost-effective multiplier design. We finally report on the final implemented design.

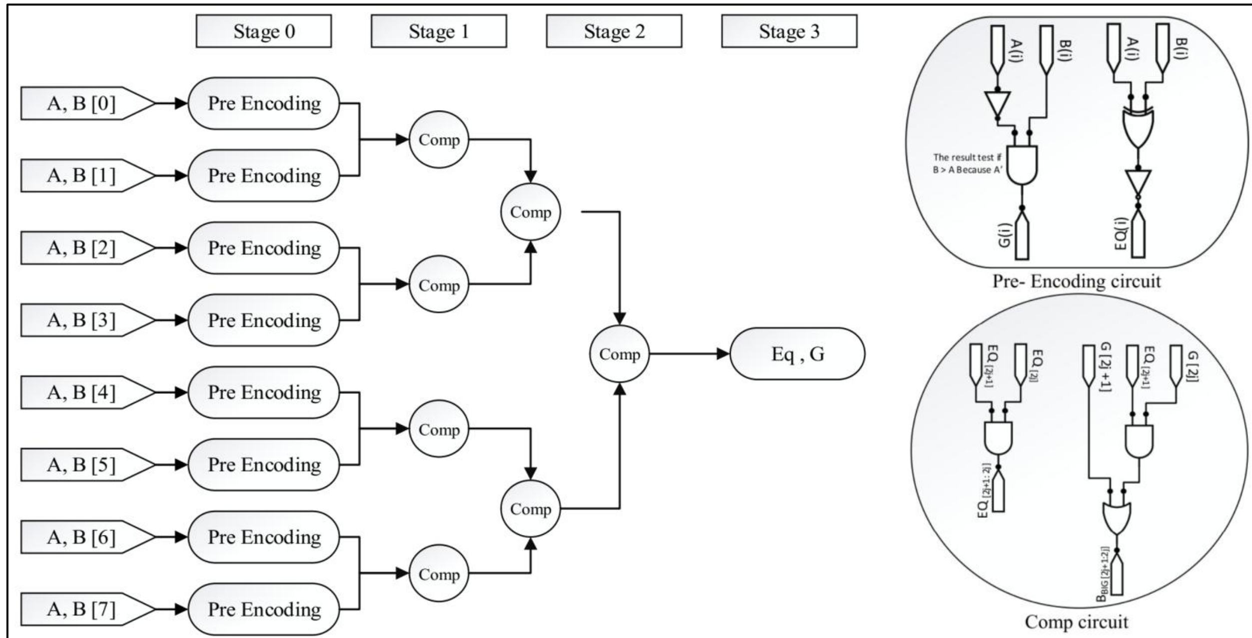


Figure 7. The complete design of8- Bit Comparatorincluding Pre- Encoding circuit and Comp circuit

A. *Radix-8 CSA Based Booth Multiplier*

Unlike Binary radix booth encoder, Radix-8 booth encodes each group of three bits as shown in table 1. The encoding technique uses shift operation to produce 2A and 4A while 3A is equal to 2A+A. The logic diagram of implementing CSA based Radix-8 booth multiplier is shown in Fig. 8. The use of CSA provides very powerful performance with limited area cost. The partial products for radix-2 is n (where n is the number of operand bits). However, for radix 8 the number of partial products is only n/3.

TABLE I. RADIX-8 BOOTH ENCODING.

Inputs (bits of M-bit multiplier)				**Partial Product**
x_{i+2}	x_{i+1}	x_i	x_{i-1}	PPR_i
0	0	0	0	0
0	0	0	1	A
0	0	1	0	A
0	0	1	1	2A
0	1	0	0	2A
0	1	0	1	3A
0	1	1	0	3A
0	1	1	1	4A
1	0	0	0	-4A
1	0	0	1	-3A
1	0	1	0	-3A
1	0	1	1	-2A
1	1	0	0	-2A
1	1	0	1	-A
1	1	1	0	-A
1	1	1	1	0

As can be seen from fig.8, the multiplier accepts two 32-bit operands (Aand X) and stores the operand (A) in a shift register to select the group bits used in encoding whereas the operand (X) processed with the

booth encoder. The output of encoding stage is added via the sequential CSA adder and the result is provided in a redundant representation (vector sum and vector carry).

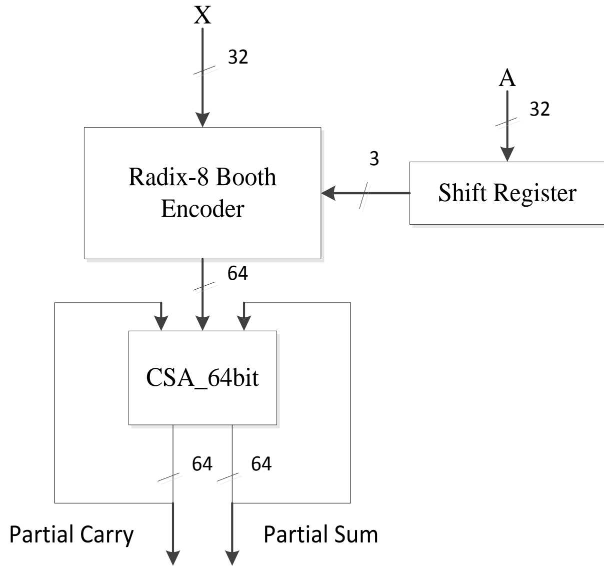


Figure 8. Design of Radix-8 Booth 32-bit multiplier

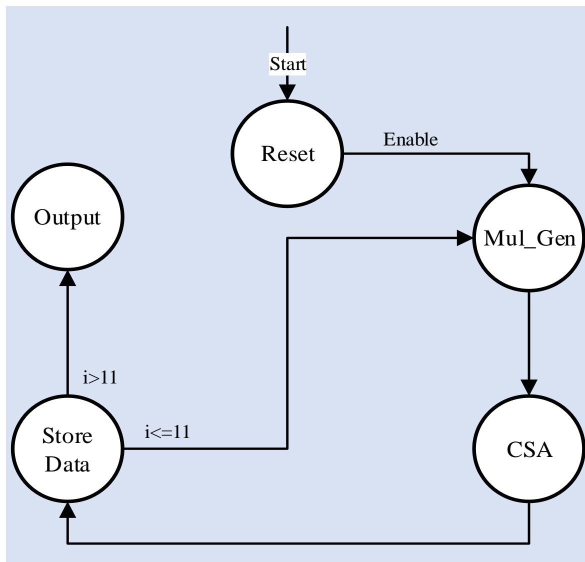


Figure 9. State machine diagram for 32-bit Booth multiplier.

Also, Fig.9 illustrates the FSM diagram of 32-bit booth multiplier. It starts with Reset_State, where all signals and variables are cleared (i.e. reset). Next state is Mul_Gen, where encoding is occurred. After that, the generated vector is added to the previous results of CSA state. Fourth, results are stored in Store_State and moves back to Mul_Gen state in loop until all the bits are selected and encoded. Finally, the output results are provided in Output state.Note that in Radix-8 encoding the number of generated partial product vectors are computed by dividing the number of bits over 3, since each three bits are selected and used for encoding.

B. CSA Based Radix-8 Booth, Wallace Tree Karatsuba Multiplier

In this method, we combine the benefits of the bit reduction of radix 8 booth along with the parallelism of CSA based Wallace tree as well as the pipelining process of Karatsuba multiplication. Thus, this design achieved minimum path delay and minimized area (i.e. the best performance). However, redundancy in this design produced one critical problem regarding the middle carry at the edges of blocks that affects the results. Fig.10 illustrates the flow diagram for this design. Here, we first designed a 64-bit Karatsuba Multiplier using a 32-bit CSA based radix-8 Booth for partial products calculation (as for our target design and since we are implementing 64-bit multiplier; m was chosen to be 32 bits (half size)). First, the entered two operands are divided into halves$(x1, x2, y1, y2)$. Next, they are fed into the Booth multiplier to compute the partial products as given in Karatsuba formula. Since the results are redundant and we have 5 partial products according to Karatsuba:

$$b^2z_2 + bz_2 - b(x_1 - x_0)(y_1 - y_0) + bz_0 + z_0.$$

Thus, 10 partial products are generated. In the final stage, a CSA based Wallace tree was implemented to be used for adding the resulted partial products. Final result is represented redundantly as vector sum and vector carry. This design achieves minimum path delay with limited area.

However, redundancy in this design produces one critical problem that affects the results. As a rule-of-thumb, if we multiply two $\mathrm{N-bit}$ numbers (i.e. p and q), the multiplication result will be increased to $2\mathrm{N-bit}$. However, this is not the case when using redundant systems since the result is stored as two $2\mathrm{N-bit}$ vectors and adding the two vectors to we tend to obtain the conventional product might result in $2\mathrm{N}+1$ bits. This additional bit brings up a new problem in the preliminary design. Now, this problem can be solved by discarding the last carry when converting back to conventional representation. However, in Karatsuba algorithm the numbers are split into 32-bit (original size is 64). The result must be 128-bit, but in Karatsuba case will be 10 partial product vectors of 64-bit shifted in such a way that adding those vectors will result in 128-bit. Thus, discarding all the generated carry when converting back to conventional system leads to error since only the carry generated of adding the two vectors corresponding to

the same variable (or the same partial product in this case) needs to be discarded. Other generated carries must be considered. Fig.11. demonstrate this problem graphically.

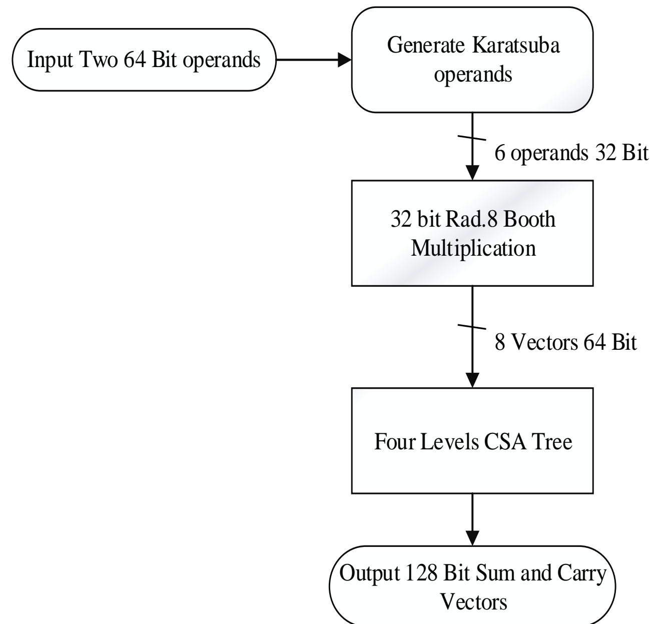


Figure 10. Design of 64-bit CSA Based Radix-8 Booth, Wallace Tree Karatsuba multiplier.

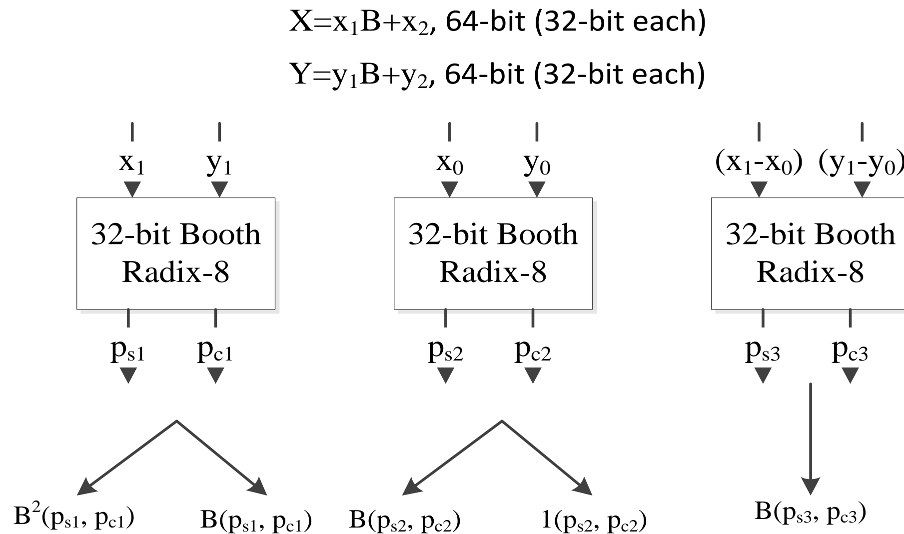


Figure 11. Graphical approaches to demonstrate the carry error (the mid-carry problem), here we have two cases:Case I- ps1+ pc1 = might result in carry, result = 65-bit (wrong). Carry must be discarded and Case II- ps1+ ps2 = might result in carry, result = 65-bit (correct). Carry must be considered.

Eventually, the mid-carry problem was solved by either using 64-bit CSA Based Radix-8 Booth, KSA Based Karatsuba multiplier or using 64-bit CSA Based Radix-8 Booth, with comparator Karatsuba multiplier. However, both solutions have added more overhead to design cost; therefore, this solution has been excluded. Both solutions are discussed in the following subsections.

1) CSA Based Radix-8 Booth, KSA Based Karatsuba Multiplier.

Since the carry to be eliminated is the generated one from booth multiplier, a first thought is to exchange the CSA adder with KSA adder to convert back the two vectors into one 64-bit number and discard any generated carry. All the 8 vectors are reduced into five 64-bit vectors in parallel. This stage helps to eliminate the false carry without the need to do any further examination. KSA is a fast adder, thus this design maintains its high performance utilizing more logic elements. The logic diagram of the design is shown in Fig.12.

2) CSA Based Radix-8 Booth, With Comparator Karatsuba Multiplier.

Another noticeable design option can solve the mid-carry problem is to use a 64-bit comparator to test if the two vectors will generate a carry if yes, then do the correction step before input the 10 vectors to CSA Tree. After Booth multiplication stage, connect the vector sum and vector carry that may produce carry error to the inputs of 64-Bit comparator unit, then perform correction if needed. Finally, all vectors added using CSA tree. The complete solution is depicted in fig.13.

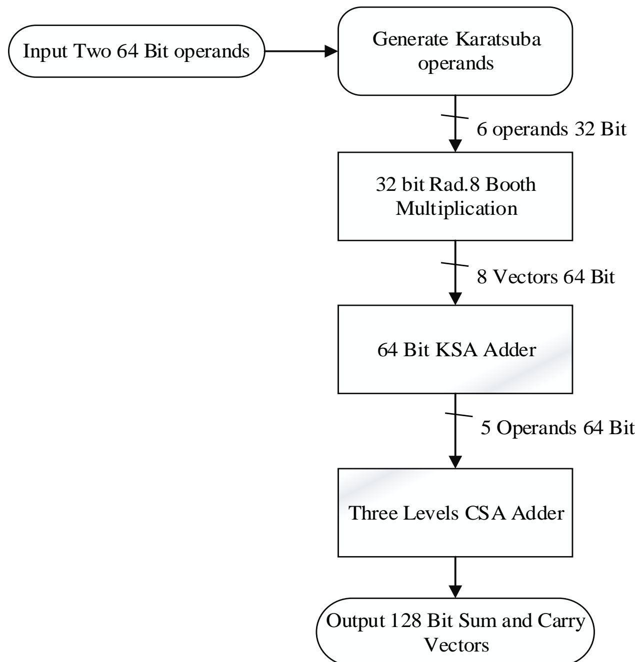


Figure 12. Design of 64-bit: 64-bit CSA Based Radix-8 Booth, KSA Based Karatsuba multiplier.

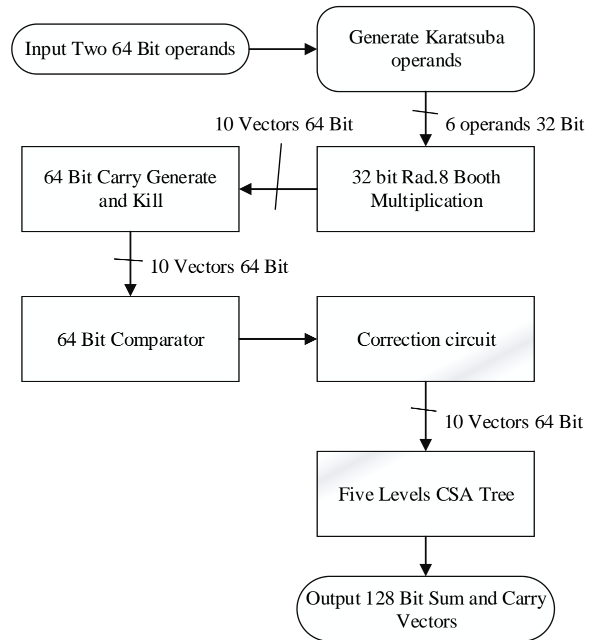


Figure 13. Karatsuba multiplication based on CSA and comparator.

Note that the 64-bit comparator can be built with 8 stages in total recording a total delay of 13 level gate delay and area of 317 gates (like the design of 8-bit comparator discussed in section.2.5). To predict whether the carry will be generated or not, then we need to generate 64-Bit G (generate) and K (kill) vectors. Thus, we have three cases which might happen as follows:

- Case I: when $G(i) = K(i) = 0$. The carry is propagated. Here we need to define the first carry state before $propagate$. If the state is $kill$, then the vector does not need any correction. But, if the state is a $generate$ state, then we need to subtract one from the highest bit (MSB) of any vector to prevent the carry to $propagate$.
- Case II:when$G(i) = 0 \, and \, K(i) = 1$. Here we have a$Kill$state, so that no need to correction.
- Case III:when $G(i) = 1 \, and \, K(i) = 0$ .Here is a$generate$state and a correction is needed. If this happed at highest bit (MSB), then it needs to subtract 2 ones. But if it after some $propagate$, then this is Case I.

To define the first case, we have used a comparator to compare the two vectors G and K as the comparator results:

- $G > k$: Generate state happened first or it is the first state after propagation
- $K > G$: kill state happened first or it is the first state after propagation

- $G = K$: All states are propagating states, no need for correction because we do not have input carry

3) Comparisons between Design II & Design III

We investigated both proposed design alternatives of Karatsuba based multiplication theoretically in terms of critical path delay (using gate delay unit) and the area of the multiplier (how many gates used in the implementation). The results are shown in table 2 below.

TABLE II. COMPARISON BETWEEN DESIGN II & DESIGN III.

Design Solutions #	Delay (gate delay)	% Optimization	Area (# of gates)	% Optimization
Solution I: using KSA Adder.	23	+15%	6130	
Solution II: using Comparator unit.	27		3712	+50%

C. Sequential 64-Bit CSA Based Radix-8 Booth Multiplier

This design is accomplished by expanding the 32-bit booth to 64-bit. The two modules (i.e. 64-bit and 32-bit Booth) differ only in the number of generated partial products. Since radix-8 is used, 22 partial products are generated in the new module instead of 11 while other logic components remained the same. Fig.14 shows the logic diagram of new 64-bit implementation. This design was implemented and was simulated on Altera FPGA Kit recording a path delay of 10.701 ns for one loop and since the program runs 22 times(i.e. 22 partial products), thus the total path delay is 235.422 nS. Also, this multiplier requires 3330 logic elements (LEs).

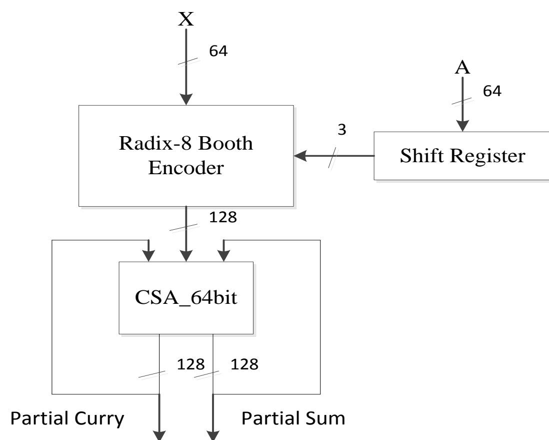


Figure 14. Design of CSA based Radix-8 Booth 64-bit multiplier.

IV. SYNTHESIZE RESULTS AND ANALYSIS

To speed up the performance of sequential 64-Bit CSA Based Radix-8 Booth Multiplier, we parallelized the addition of partial products produced in the same level by using Wallace CSA tree instead of sequential CSA to exploit the maximum possible parallelism between the partial products to gain in speed and enhance the design performance. That's it, we end up with implementing a 64-bit Wallace Tree CSA based Radix-8 Booth multiplier (WCBM). The block diagram for the proposed design is shown in Fig.15. (a). The comparison with the other design alternatives showed that Wallace Tree CSA Based Radix-8 Booth Multiplier (WCBM) has decreased the total delay and increased the operational speed for the multiplication operation. Also, the design is modified to increase the frequency be dividing the program to three main states.

The top view of our implemented WCBM unit is given in Fig.15. (b). It's clearly seen that WCBM unit is triggered by CLK signal along with enable line. The generated number can be obtained from the output portliness "sum" which is 128 bits. Besides the unit encompasses three control input signals (enable, reset, clk) and two control output signals (Ack and Ready). Moreover, the finite state machine (FSM) diagram for the implemented WCBM is shown in Fig.15. (c). FSM consists of three main phases: Partial product generation (Initially, 22 partial products are generated by using radix-8 Booth encoding), Wallace tree phase (these partial products are added by using 7 levels Wallace Tree CSA based) and KSA phase (because the result is redundant, KSA is used in the last phase to convert it to conventional result). Finally, Fig.16. Illustrates a sample numerical example of the proposed WCBM that is generated from Quartus II simulation tool.

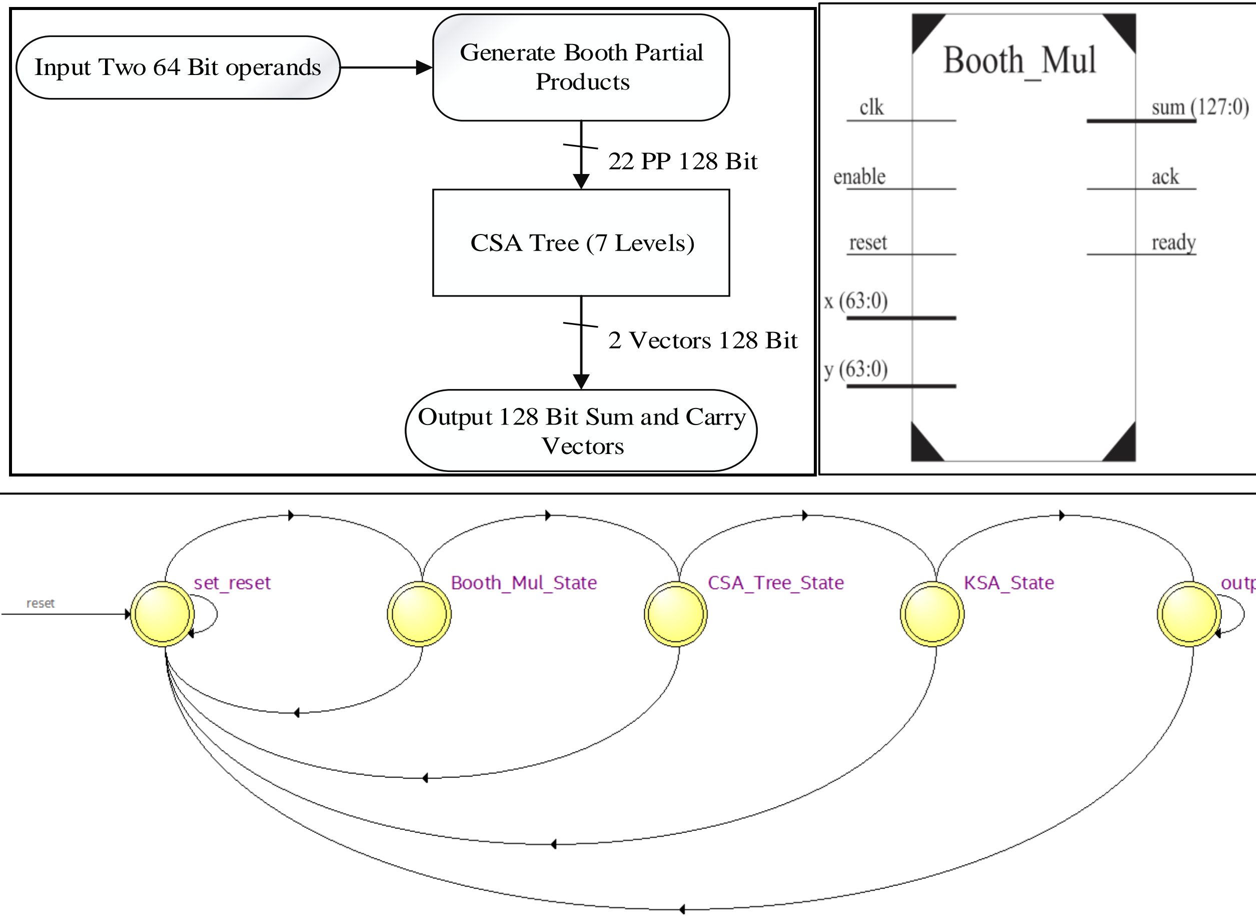


Figure 15. (a) Design Architecture of WCBM (a) Top Level DiagramWCBM (C) FSM Diagram for WCBM.

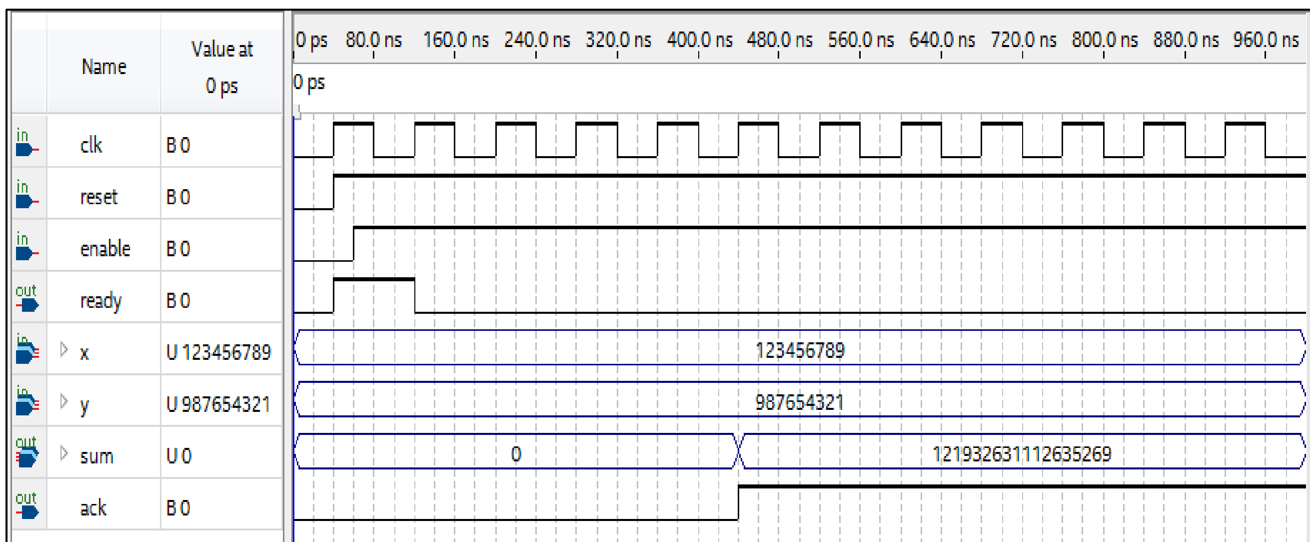


Figure 16. Sample run example of WCBM process of two 64-bit numbers

The proposed multiplier implementation has been synthesized using Altera Cyclone EP4CGX-22CF19C7 FPGA kit to analyze several design factors such as design area, the total delay of the multiplication unit and the thermal power consumption of FPGA implementation. We have evaluated the performance of the 64-bit Wallace Tree CSA based Radix-8 Booth multiplier WCBM module for different data path sizes. Timing analysis of the critical clock cycle for the implemented WCBM is illustrated in Fig.17. It can be seen from the graph that the critical path delay is 14.103 ns in which 3.094 ns for the clock delay and 11.009 ns for the data delay. This give a maximum frequency for the circuit of 90.83 MHz.In addition, the area of the design has recorded a constant number of logic elements (i.e. 14249 LEs) with the total thermal power dissipation estimated by using PowerPlay Power analyzer tool of Quartus II software of 217.56 mW.

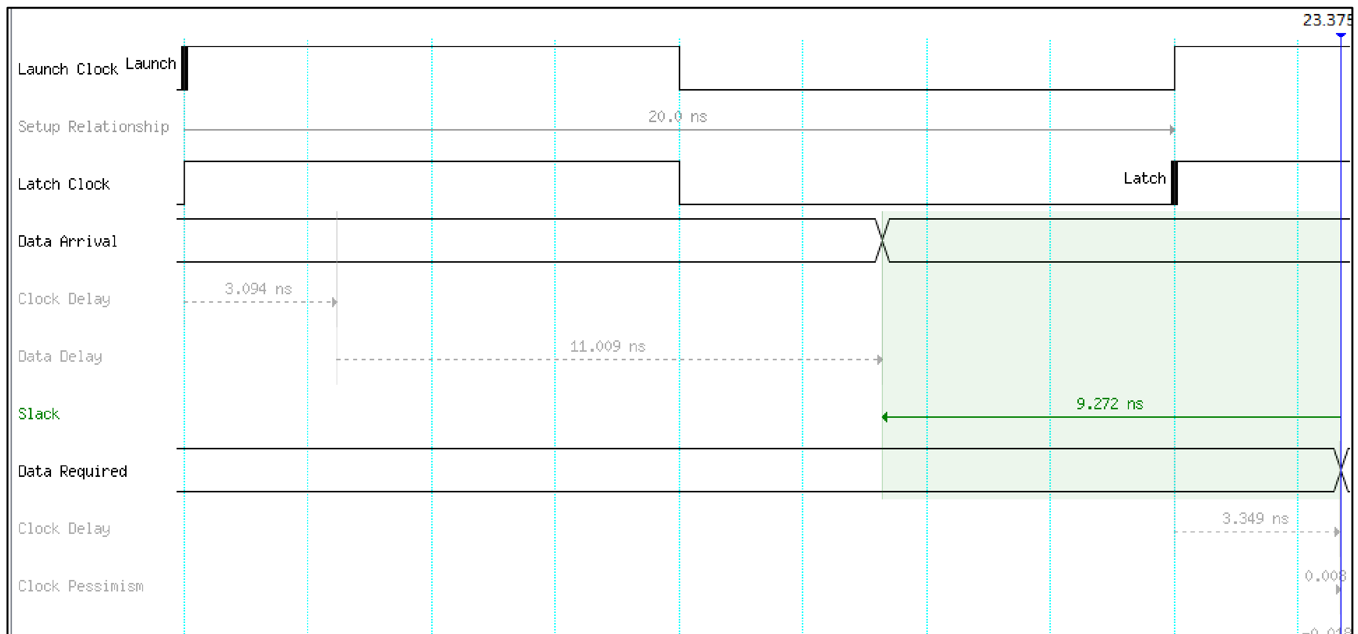


Figure 17. Waveform sample of the proposed WCBM data delay

V. CONCLUSIONS AND REMARKS

Multiplication operation is a core operation that domineer the performance of several public cryptographic algorithms such as RSA and SSC. In this paper, we have thoroughly discussed several design alternatives of radix-8 based multiplier unit by employing the Karatsuba method and Booth recording method with carry save and Kogge stone adders on Wallace tree organization. The proposed designs were evaluated in terms of many aspects including: maximum frequency and critical path delay, design area, and the total FPGA power consumption. The proposed hardware cryptosystem design is conducted using Altera Cyclone FPGA design technology along with the help of CAD package of Altera such as Quartus II and Modelsim 10.1. To sum up, we have successfully implemented and synthesized the Wallace Tree CSA Based Radix-8 Booth Multiplier (WCBM) module via the target FPGA technology for 64-bits. The synthesizer results showed an attractive results in terms of several design factors that can improve the computation performance for many multiplication based applications.

REFERENCES

[1] A.J. Menezes, P.C. Van Oorschot and S.A. Vanstone. (1996). Handbook of Applied Cryptography", CRC Press, Boca Raton, Florida.

[2] K. Javeed, X. Wang and M. Wang, 'Serial and Parallel Interleaved Modular Multipliers on FPGA Platform', IEEE 25th International Conference on Field Programmable Logic and Applications (FPL), 2015 https://doi.org/10.1109/FPL.2015.7293986

[3] D. J Greaves, System on Chip Design and Modelling, University of Cambridge, Computer Laboratory, Lecture Notes, 2011. http://www.cl.cam.ac.uk/teaching/1011/SysOnChip/socdam-notes1011.pdf.

[4] M. D. Ercegovac and T. Lang, (2004) 'Digital Arithmetic', Morgan Kaufmann Publishers, Elsevier, vol.1, p.p.51-136. http://www.sciencedirect.com/science/book/9781558607989

[5] Qasem Abu Al-Haija, Sharifah M. S. Ahmad, "Fast Radix-2 Sequential Multiplier Using Kintex-7 FPGA Chip Family", The Open Cybernetics & Systemics Journal, Bentham Open, Vol. 12, 2018.

[6] Mohammed Mosab Asad, Ibrahim Marouf, Qasem Abu Al-Haija, "Review Of Fast Multiplication Algorithms For Embedded Systems Design", International Journal Of Scientific & Technology Research Volume 6, Issue 08, 2017.

[7] Heath, Steve (2003). Embedded systems design. EDN series for design engineers (2 ed.). Newnes. p. 2. ISBN 978-0-7506-5546-0. An embedded system is a microprocessor based system that is built to control a function or a range of functions.

[8] I. Marouf, M. M. Asad, A. Bakhuraibah and Q. A. Al-Haija, "Cost analysis study of variable parallel prefix adders using altera cyclone IV FPGA kit," 2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA), Ras Al Khaimah, 2017, pp. 1-4.doi: 10.1109/ICECTA.2017.8252011

[9] Altera Co., “Introduction to Quartus II Software: Ver 10.0”, Intel Quartus II MNL-01055-1.0, 2012.

[10] Altera Corporation, “Cyclone IV Device Handbook”, Vol. 1, CYIV-5V1-2.2, https://www.altera.com/, 2012.

[11] S. Butchibabu, S. Kishore Bab (2014). Design and Implementation of Efficient Parallel Prefix Adders on FPGA, International Journal of Engineering Research & Technology, Vol. 3 Issue No.7.

[12] B. Parhami, (1999), “Computer Arithmetic: Algorithms and Hardware Designs”, Oxford University Press, Oxford.

[13] D. Purohit, H. Joshi, (2014), ‘Comparative Study and Analysis of Fast Multipliers’, International Journal of Engineering and Technical Research (IJETR), Vol. 2, No.7, 2014.

[14] A. Karatsuba and Y. Ofman, (1963) ‘Multiplication of Multidigit Numbers on Automata’, Soviet Physics, Doklady, p.p.595-596. https://www.researchgate.net/publication/234346907_Multiplication_of_Multidigit_Numbers_on_Automata

[15] Qasem Abu Al-Haija, Mohamad M.Asad, Ibrahim Marouf,"A Systematic Expository Review of Schmidt-Samoa Cryptosystem", International Journal of Mathematical Sciences and Computing(IJMSC), Vol.4, No.2, pp.12-21, 2018.DOI: 10.5815/ijmsc.2018.02.02

[16] Qasem Abu Al-Haija, Mahmoud Smadi, Monther Al-Ja’fari, Abdullah Al-Shua’ibi, "Efficient FPGA implementation of RSA coprocessor using scalable modules", Procedia Computer Science, Elsevier, Vol 34, 2014.

[17] Qasem Abu Al-Haija, Mohammad Alkhatib, Azmi B Jaafar, "Choices on Designing GF(P) Elliptic Curve Coprocessor Benefiting from Mapping Homogeneous Curves in Parallel Multiplications", International Journal on Computer Science and Engineering, Engg Journals Publications, Vol. 3, No. 2, 2011.