ISSN:0975-9646

FPGA Realization of Radix-4 Booth Multiplication Algorithm for High Speed Arithmetic Logics

K. Babulu,G.Parasuram
Department of Electronics and Communication Engineering,
Jawaharlal Nehru Technological University, Kakinada (JNTUK), Kakinada, India

Abstract: **A new architecture, namely, Multiplier-and-accumulator (MAC) based Radix-4 Booth Multiplication Algorithm for high-speed arithmetic logics have been proposed and implemented on Xilinx FPGA device. By combining multiplication with accumulation and devising a hybrid type adder the performance was improved. The modified booth encoder will reduce the number of partial products generated by a factor of 2. Fast multipliers are essential parts of digital signal processing systems. The speed of multiply operation is of great importance in digital signal processing as well as in the general purpose processors. The number to be added is the multiplicand, the number of times that it is added is the multiplier, and the result is the product. Each step of addition generates a partial product.**

Key Words: **- VLSI, FPGA, Carry Select Adder (CSA), Carry Look Ahead Adder (CLA), ASM**

I. INTRODUCTION

With the recent rapid advances in multimedia and communication systems, real-time signal processing like audio signal processing, video/image processing, or large-Capacity data processing are increasingly being demanded. The multiplier and multiplier-and-accumulator (MAC) [1] are the essential elements of the digital signal processing such as filtering, convolution, and inner products. Most digital signal processing methods use nonlinear functions such as discrete cosine transform (DCT) [2] or discrete wavelet transform (DWT) [3]. Because they are basically accomplished by repetitive application of multiplication and addition, the speed of the multiplication and addition arithmetic's determines the execution speed and performance of the entire calculation.

The article concentrates starting from the basic multiplier fundamentals, the general multiplication types and survey on various types of multipliers. Fast arithmetic requires fast circuits. Fast circuits require small size, to minimize the delay effects of wires. Small size implies a single chip implementation, to minimize wire delays, and to make it possible to implement these fast circuits as part of a larger single chip system to minimize input/output delays.

At this junction, we discuss about a Modified Booth Encoding Radix-4 [9, 10] 8-bit Multiplier. Booth multiplication allows for smaller, faster multiplication circuits through encoding the signed numbers to 2's complement, which is also a standard technique used in chip design, and provides significant improvements by reducing the number of partial product to half over "long multiplication" techniques. This paper reveals and demonstrate an extendable system architecture for 8-bit Radix-4 Booth algorithm [4][5]. As part of that the main blocks of Booth Encoder i.e., Partial Product Generator and Hybrid adder are presented in this algorithm.

II. BASIC BINARY MULTIPLIER

Multiplier circuits are found in virtually every computer, cellular telephone, and digital audio/video equipment. In fact, essentially any digital device used to handle speech, stereo, image, graphics, and multimedia content contains one or more multiplier circuits. The multiplier circuits are usually integrated within microprocessor, media co-processor, and digital signal processor chips. These multipliers are used to perform a wide range of functions such as address generation, Discrete Cosine Transformations (DCT), Fast Fourier Transforms (FFT), multiply-accumulate, etc. As such, multipliers play a critical role in processing audio, graphics, video, and multimedia data.

A multiplying circuit is able to perform a multiplication of n-bits X n-bits at a high speed by increasing the speed of the forming process of the partial products so that the delay time may be inhibited from increasing for a large n, and which can inhibit the chip size becoming large. Multiplication is more complicated than addition, being implemented by shifting as well as addition. Because of the partial products involved in most multiplication algorithms, more time and more circuit area is required to compute, allocate, and sum the partial products to obtain the multiplication result. Fig.1 shows the flow chart for basic binary multiplier.

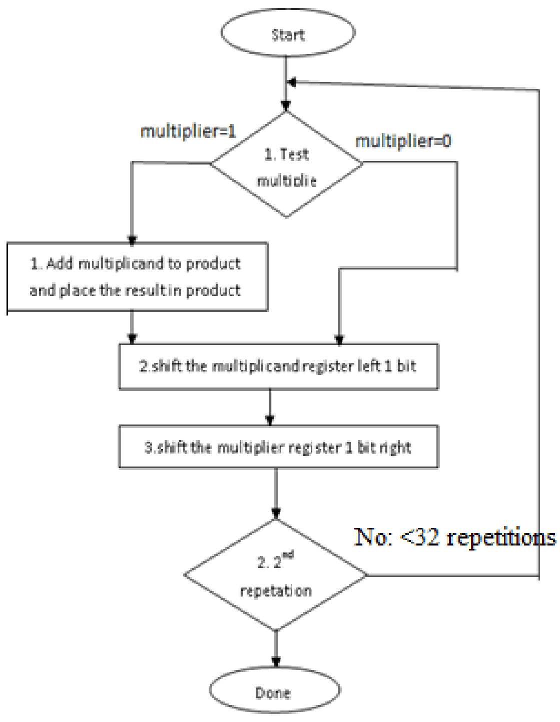
Start
multiplier=1
1. Test multiplie
multiplier=0
1. Add multiplicand to product and place the result in product
2.shift the multiplicand register left 1 bit
3.shift the multiplier register 1 bit right
No: <32 repetitions
2. 2nd repetation
Done

Fig.1. Flow Chart for Basic Binary Multiplier

The methods all reduce to two basic steps-create a group of partial products, then add them up to produce the final product. Different ways of adding the partial products were mentioned, but little was said about how to generate the partial products to be summed. A recoding scheme introduced by Booth reduces the number of partial products by about a factor of two.

III. ADDERS FOR MULTIPLICATION

Fast carry propagate adders are important to high performance multiplier design in two ways. First, an efficient and fast adder is needed to make any "hard" multiples that are needed in partial product generation. Second, after the partial products have been summed in a redundant form, a carry propagate adder is needed to produce the final non redundant product.

A. Ripple Adder

It is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a C_{in} which is the C_{out} of the previous adder. This kind of adder is a ripple carry adder, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder in some cases.

The layout of a ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit. Each full adder requires three levels of logic.

B. Carry Look-Ahead Adder (CLA)

The concept behind the CLA is to get rid of the rippling carry present in a conventional adder design. The rippling of carry produces unnecessary delay in the circuit. Carry look-ahead logic uses the concepts of generating and propagating carries. Although in the context of a carry look ahead adder, it is most natural to think of generating and propagating in the context of binary addition, the concepts can be used more generally than this. In the descriptions below, the word digit can be replaced by bit when referring to binary addition.

C. Carry Select Adder (CSA)

The carry select adder generally consists of two ripple carry adders and a multiplexer. Adding two k-bit numbers with a carry select adder is done with two k/2 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. The number of bits in each carry select block can be uniform, or variable.

In the uniform case, the optimal delay occurs for a block size of square root of K. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added, since that will yield an equal number of MUX delays.

D. Hybrid Adder

Hybrid Adder [11, 12] is a combination of any two adders. It is used in high speed applications. The proposed hybrid adder consists of two carry look ahead adders and a multiplexer. Adding two n-bit numbers with a hybrid adder is done with two adders (therefore two carry look ahead 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. The propagation delay is less for hybrid adder and at the same time it occupies larger area compared to the other adders.

IV. DESIGN APPROACH

This section focus on the design approach for Radix-2 and Radix-4 Booth multipliers by considering the necessary specifications and made in the form of state diagrams and ASM charts for develop the relevant source code in VHDL. The presented Figures elaborate the logics required for necessary operations.

A. Booth Multiplication Algorithm for Radix-2

It will encode the multiplicand based on multiplier bits. In Radix -2 we will compare 2 bits at a time with overlapping technique. Grouping starts from the LSB, and the first block only uses one bit of the multiplier and assumes a zero for the second bit.

Table 1: Radix-2 Booth Encoding Table

Block	**Partial Product**
00	0
01	1*Multiplicand
10	-1*Multiplicand
11	0

The functional operation of booth encoder is tabulated in Table 1. There are two inputs for booth encoder one is multiplicand and the other is 2 bits from multiplier, based on these two inputs it will encode the multiplicand.

State diagram

The state diagram of the Radix-2 Booth multiplier is shown in Fig.2. Here we have four different types of states. For 00, 11 states we can perform multiplication of multiplicand with zero. For 01 state, we can multiply multiplicand with one whereas for 10 state, we can multiply multiplicand with -1.

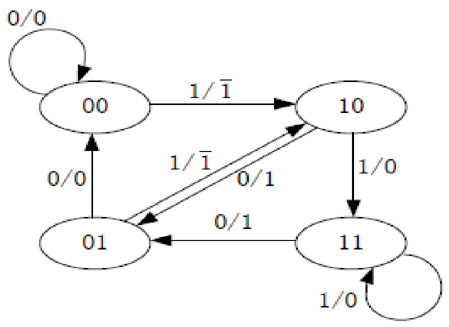

Fig.2. State diagram for Radix-2 Multiplier

ASM chart

The Fig.3 shows the ASM chart for Radix-2 booth multiplier. It represents conventional procedure for various operations required with respect to state of machine. Here we generate the partial products by Radix-2 booth encoder. By using this technique we can reduce the partial products generation and the computation time delay is less than ordinary multiplication.

B. Booth Multiplication Algorithm for Radix-4

One of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm (Radix-2) had two drawbacks. They are: (i) the number of add subtract operations and the number of shift operations become variable and become inconvenient in designing parallel multipliers. (ii) The algorithm becomes inefficient when there are isolated 1's. These problems are overcome by using modified Radix-4 Booth multiplication algorithm. The design approach of Radix-4 algorithm is described with the pictorial views of state diagram and ASM chart.

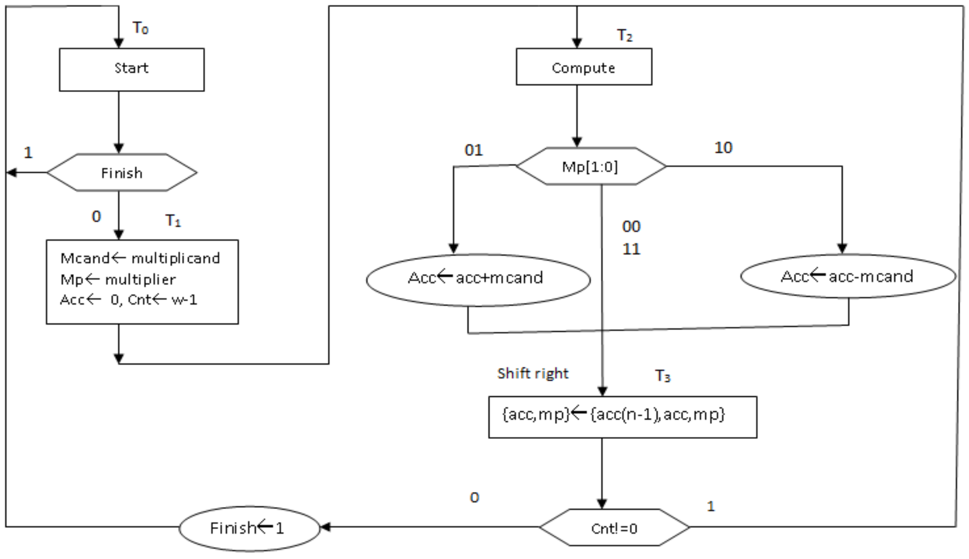

Fig.3. ASM chart for Radix-2 Booth Multiplier

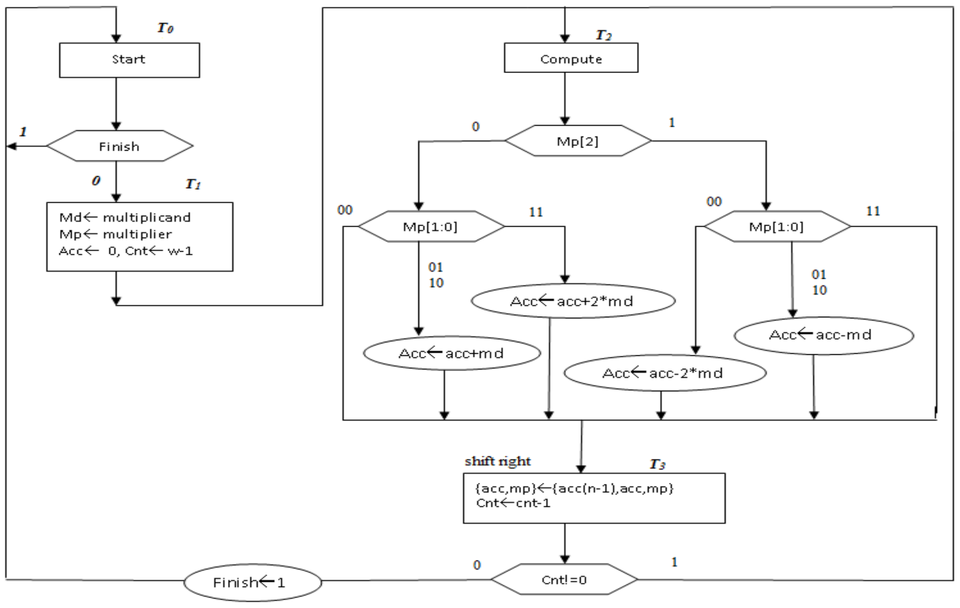

Fig.5. ASM chart for Radix-4 Booth Multiplier

Table 2: Radix-4 Booth Encoding Table

Block	Partial Product
000	0
001	1*multiplicand
010	1*multiplicand
011	2*multiplicand
011	2*multiplicand
100	-2*multiplicand
101	-1*multiplicand
110	-1*multiplicand
111	0

This algorithm scans strings of three bits as follows:
1) Extend the sign bit 1 position if necessary to ensure that n is even.
2) Append a 0 to the right of the LSB of the multiplier.
3) According to the value of each vector, each Partial Product will bhe 0, +y, -y, +2y or -2y.
Radix-4 booth encoder performs the process of encoding the multiplicand based on multiplier bits. It will compare 3 bits at a time with overlapping technique. Grouping starts from the LSB, and the first block only uses two bits of the multiplier and assumes a zero for the third bit. The functional operation of Radix-4 booth encoder is shown in the Table 2.
The state diagram of the Radix-4 Booth multiplier is shown in Fig.4. It consists of eight different types of states and during these states we can obtain the outcomes, which are multiplication of multiplicand with 0,-1 and -2 consecutively. The pictorial view of the state diagram presents various logics to perform the Radix-4 Booth multiplication in different states as per the adopting encoding technique.

ASM chart

The ASM chart for Radix-4 booth multiplier is as shown in Fig.5. This represents the conventional flow of operations that are required for Radix-4 booth multiplier in various states. Here we can generate the partial products by Radix-4 booth encoder. By using this technique we can further reduce the partial products generation and the computation time delay, which is less than that of Radix-2 multiplication.

State diagram

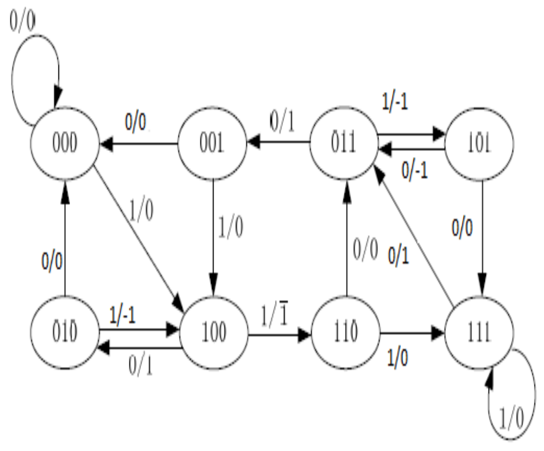

Fig.4. State diagram of Radix-4 Booth Multiplier

V. SIMULATION RESULTS

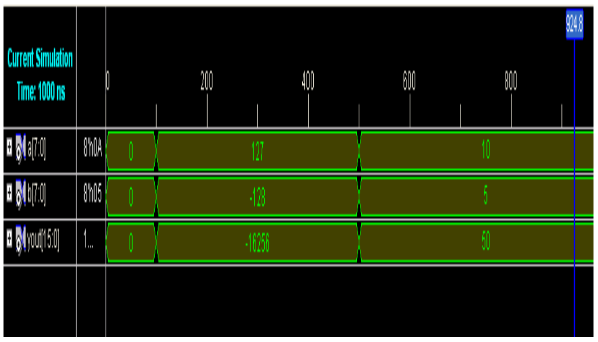

Fig.6. Simulation Results of Radix-2 Booth multiplier

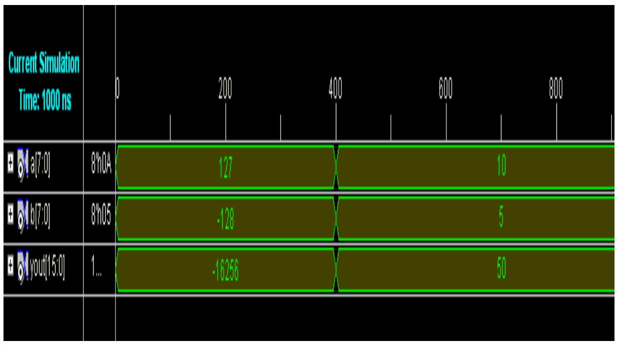

Fig.7. Simulation Results of Radix-4 Booth multiplier

Fig.6.shows the simulation result of Radix-2 Booth multiplier in which they are two binary inputs, multiplicand and multiplier. If both binary numbers are positive then it will go directly to booth encoding. If any one of operands is negative it will take two's complement and then it performs booth encoding. Initial consider two-two bits from multiplier as one zero bit and other bit as lowest bit of multiplicand. During the next cycle it takes two-two bits from multiplicand in overlap manner. Perform the same until the process completed. At last the addition of partial products is done by tree type hybrid adder.
Fig.7.shows the simulation result of Radix-4 Booth multiplier in which they are two binary inputs as input, one is multiplicand and another one is multiplier. If both binary numbers are positive then it will perform booth encoding. If any one of operands is negative it will take two's complement and then it will do booth encoding. Consider three-three bits from multiplier, initially take one bit zero and other bits from lowest bits of multiplicand. For next operation consider three-three bits from multiplicand in overlap manner. At end of operation an addition of partial products can be carried out by tree type hybrid adder. The required simulation has been carried out by using Model Simulator and the functional verification performed.

VI. FPGA REALIZATION

The designed system is targeted onto Xilinx xc2vpx70-7-ff1704 FPGA device belonging to virtex2p family with a speed grade of –7. The logical routing can be observed from the obtained Place and route result from the FPGA Editor option in Xilinx synthesizer. It is observed that about 40% area for the targeted FPGA is covered for the implementation of this System. The CLB's are connected in cascade manner to obtain the functionality for the designed system.

A. Synthesis Report

The synthesis result for the proposed algorithm is presented:

Macro Statistics# Registers: 49

# Multiplexers	: 25
# Tristates	: 74
# Adders/Subtractors	: 618
# Multipliers	: 29
# Comparators	: 128

Design Statistics

# IOs	: 26

Cell Usage :

# BELS	: 181

Minimum period: 5.220ns (Maximum Frequency: 191.571MHz)

From the result it is observed that logical counts of 181 Basic Element Logic (BEL) is required for the realization of DST processor. The real time Maximum operating frequency obtained is 191.571 MHz and this operation frequency is considerably higher than the current sample frequency and makes it more suitable for real time current analysis.

B.RTL Views

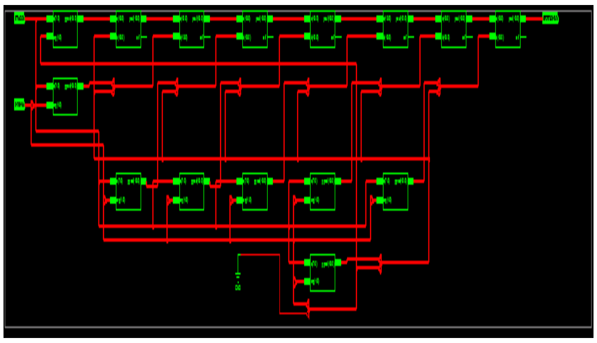
Fig.8. RTL Schematic of Radix-2 Booth multiplier

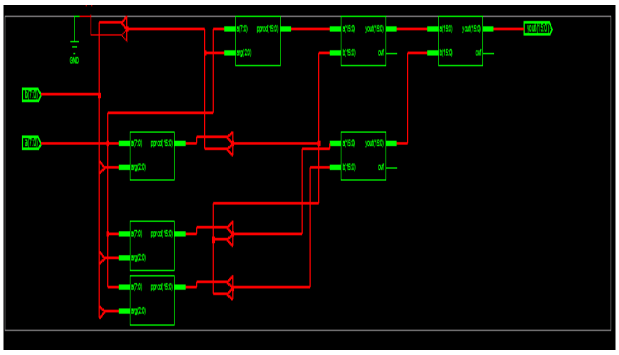
Fig.9. RTL Schematic of Radix-4 Booth multiplier

C. Routing and Placement

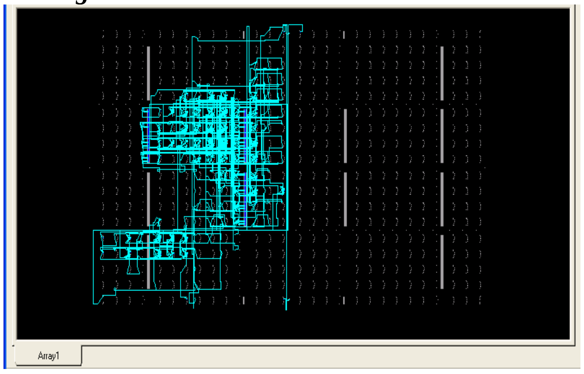
Fig.10. Routing of logical placement in targeted FPGA

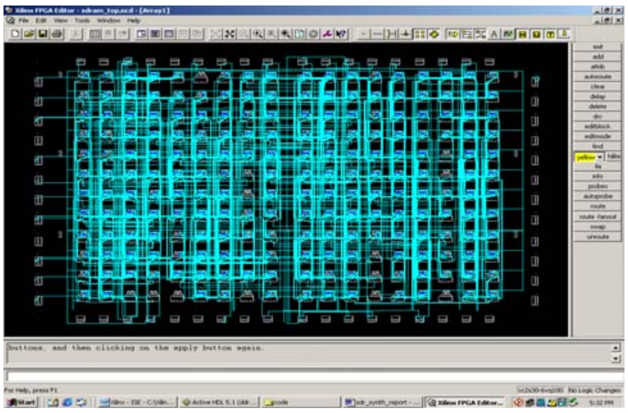
Fig.11. FPGA Placement of the targeted FPGA

D. Implementation Observations

The implementation of proposed Radix-4 Booth Multiplication algorithm is illustrated in various pictorial views obtained during the process of realization i.e., from Fig.8 to Fig.12. Fig.8 & Fig.9 shows the RTL views of existing and proposed algorithms. Routing of logical placement in targeted FPGA is shown in Fig.10 and Fig.11 represent the placement of the targeted logic onto FPGA device.

VII. PERFORMANCE OF MULTIPLIERS

Table 3: Performance of Multipliers

	Radix-4 booth multiplier with hybrid adder	Radix-2 booth multiplier with hybrid adder
No. of slices	119	166
No. of LUTs	213	300
Path delay	29.198ns	37.881ns

Table 3 is valid for 8 bit x 8 bit multiplier. The table distinguishes the performance of proposed Radix-4 Booth Multiplier with the existing Radix-2 Booth Multiplier. The main advantage of using Radix-4 is it has less propagation delay, i.e speed and at the same time it occupies lesser area than Radix-2.

VIII. FUTURE SCOPE

The algorithm has been implemented using hybrid adder to add the partial products in parallel for the final output. Hybrid adder is a combination of carry look ahead adder and carry select adder. It can be further extended by taking combination of any two adding techniques so that propagation delay is further reduced. For higher inputs Radix 2^n multipliers will give better performance.

IX. CONCLUSION

It is to be concluded that this presentation projects the design approach for modified (Radix-4) Booth's algorithm. Further, we have observed the simulation results of the booth multiplier and booth encoder for radix-2 and radix-4 algorithms. The FPGA realization of the proposed Booth multiplier has been performed using relevant synthesizer. The design flow was discussed with the aid of necessary ASM charts and state diagrams.

REFERENCES

[1] J. J. F. Cavanagh, *Digital Computer Arithmetic*. New York: McGraw-Hill, 1984.

[2] *Information Technology-Coding of Moving Picture and Associated Autio, MPEG-2 Draft International Standard*, ISO/IEC 13818-1, 2, 3,1994.

[3] JPEG *2000 Part I Final1191 Draft*, ISO/IEC JTC1/SC29 WG1.

[4] O. L. MacSorley, "High speed arithmetic in binary computers," Proc.IRE, vol. 49, pp. 67–91, Jan. 1961.

[5]A.D.Booth,"A signed binary multiplication technique," Quart. J.Math., vol. IV, pp. 236–240, 1952.

[6] G. Goto, T. Sato, M. Nakajima, and T. Sukemura, "A 54×54 regular structured tree multiplier," IEEE J. Solid- State Circuits, vol. 27, no. 9,pp. 1229–1236, Sep. 1992.

[7] J. Fadavi-Ardekani, "M×N Booth encoded multipliergenerator using optimizedWallace trees," IEEE Trans.Very Large Scale Integr. (VLSI) Syst., vol. 1, no. 2, pp.120–125, Jun. 1993.

[8] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A.Shimizu, K. Sasaki, and Y. Nakagome, "A 4.4 ns CMOS 54×54 multiplier using

pass-transistor multiplexer," IEEE J. Solid-State Circuits, vol. 30, no.3, pp. 251–257, Mar. 1995.

[9] Kim, K., P. Beerel, "A synchronous matrix-vector multiplier for discrete cosine transform", in international symposium on low power electronics and design, pp. 256-261, July 2000.

[10] Tang, T.Y., C.S. Choy, P.L. Siu and C.F. Chan, "Design of self-timed asynchronous Booth's multiplier", in Proc. Asia South Pacific Design Automation Conf., pp. 15-16, Jan 2000.

[11] Fahmi, M.N., F. Elguibaly, E. Abdel-raheem, and A Tawfik, "Area-time efficient fixed-point multiplier-accumulators for inner-product computation", in Proc. IEEE Int. Conf.Microelectronics, Dhahran, Saudi Arabia, pp. 189-192, Dec, 1999.

[12] Kim, S., C.H. Ziesler, and M.C. thymiou, "Design, verification, and test of a true single-phase 8-bit adiabatic multiplier", in Proc. 19th Conf. Advances Research VLSI, Salt Lake City, UT, pp. 42-58, Mar. 2001.