IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197 Volume 1, Issue 6 (Mar. – Apr. 2013), PP 19-26
www.iosrjournals.org

An Efficient Two's Complement Multiplier With FPGA Implementation

Kolathur Lakshmipathy [1], , Voruganti Santhosh Kumar [2], Dr Vangala padmaja[3]
1(Department of ECE, VNR VJIET, Hyderabad, India)
2 (Department of ECE, AEC, Kodad, A.P, India)
3 (Department of ECE, VNR VJIET, Hyderabad, India)

Abstract: *The performance of multiplication is crucial for multimedia applications such as 3D graphics and signal processing systems which depend on extensive numbers of multiplications. Previously reported multiplication algorithms mainly focus on rapidly reducing the partial products rows down to final sums and carries used for the final accumulation. These techniques mostly rely on circuit optimization and minimization of the critical paths.*

In this paper, an algorithm to achieve fast multiplication in two's complement representation is presented. Indeed, our approach focuses on reducing the number of partial product rows. In turn, this directly influences the speed of the multiplication, even before applying partial products reduction techniques. Fewer partial products rows are produced, thereby lowering the overall operation time. This results in a true diamond-shape for the partial product tree which is more efficient in terms of implementation.
Keywords:— *MBE, PPR, PPRG, FPGA.*

I. INTRODUCTION

The performance of 3D graphics and signal processing systems strongly depends on the performance of multiplications because these applications need to support highly multiplication intensive operations. Therefore, there has been much work on advanced multiplication algorithms and designs [1, 22, 3, 23, 18, 14, 13, 6, 7, 16, 20, 24, 12].

There are three major steps to any multiplication. In the first step, the partial products are generated. In the second step, the partial products are reduced to one row of final sums and one row of carries. In the third step, the final sums and carries are added to generate the result. Most of the above mentioned approaches employ the Modified Booth Encoding (MBE) approach [6, 7, 13, 24, 4] for the first step because of its ability to cut the number of partial products rows in half. They then select some variation of any one of partial products reduction schemes such as the Wallace trees [22, 6] or the compressor trees [16, 13, 18, 14] in the second step to steeply reduce the number of partial product rows to the final two (sums and carries). In the third step, they use some kind of advanced adder approach such as carry-lookahead or carry-select adders [5, 17, 11] to add the final two rows, resulting in the final product. The main focus of recent multiplier papers [7, 16, 20, 24,4, 12] has been on rapidly reducing the partial product rows by using some kind of circuit optimization and identifying the critical paths and signal races. In other words, the goals have been to optimize the second step of the multiplication described above.

However, in this paper, we will focus on the first step which consists in forming the partial product array and we will strive to design a multiplication algorithm which will produce fewer partial product rows. By having fewer partial product rows, the reduction tree can be smaller in size and faster in speed. It should also be noted that 8 or16-bit words are the most commonly used word sizes in the kernels of most multimedia applications [19] and that the implementation of our overall algorithm is particularly well suited to such word sizes. In the next section, the conventional multiplication method is described in detail with an emphasis on its weaknesses. In section iii, a step-by-step procedure to prevent the adverse effects of some conventional multiplication algorithms is presented. In section iv, the effectiveness and usage of our method is presented by showing a detailed evaluation.

II. Multiplication Algorithms

There is no doubt that MBE is efficient when it comes to reducing the partial products. However, it is important to note that there are two unavoidable consequences when using MBE: sign extension prevention and negative encoding. The combination of these two unavoidable consequences results in the formation of one additional partial product row and of course, this additional partial product row requires more hardware but more importantly time

A. Modified Booth Encoding and the Overhead of Negative Encodings

This grouping of the multiplier bits of MBE is shown in Figure 1 and it is based on a window size of 3 bits and astride of 2. The multiplier (Y) is segmented into groups of three bits ($y2i+1$, $y2i$, $y2i-1$) and each such group of bits is associated with its own partial product row by using Table 1 [15]. In this grouping, $y-1 = 0$.By applying this encoding, the number of partial product rows to be accumulated is reduced from n to $n/2$. For example, for an 8 × 8 multiplication, a multiplier without MBE will generate eight partial product rows (because there is one partial product row for each bit of the multiplier).However, with MBE, only $n/2$ (= 4) partial products rows are generated as shown in the example of Figure 2.However, there are actually $n/2 + 1$ partial product rows anther than $n/2$, because of the last *neg* signal (*neg*3 in Figure2).

The *neg* signals (*neg*0, *neg*1, *neg*2, and *neg*3) are needed because MBE may generate a negative encoding ((-1) times the multiplicand or (-2) times the multiplicand).Consequently, one additional carry save adding stage is needed to perform the reduction. This is the overhead of implementing the negative encodings of MBE.

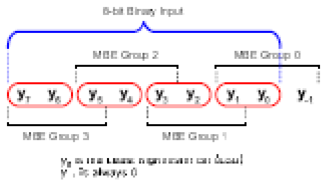

Figure 1: Multiplier bits grouping according to modified booth encoding for 8-bit input

Table 1: Modified Booth Encoding (radix-4)

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Generated partial products
0	0	0	0×X
0	0	1	1×X
0	1	0	1×X
0	1	1	2×X
1	0	0	(-2)×X
1	0	1	(-1)×X
1	1	0	(-1)×X
1	1	1	0×X

B. Sign Extension and its Prevention

In signed multiplication, the sign bit of a partial product row would have to be extended all the way to the MSB position which would require the sign bit to drive that many output loads (each bit position until the MSB should have the same value as the sign). This makes the partial product rows unequal in length as shown in Figure : the first row spans 16 bits (*pp*00 to the leftmost *pp*80), the second row 14 bits (*pp*01 to the leftmost *pp*81), the third row 12bits (*pp*02 to the leftmost *pp*82), and the fourth row 10 bits(*pp*03 to the leftmost *pp*83). The sign extension prevention method shown in figure3.and arrives a newly formed partial product rows as in figure4[10]where the sign extension has been removed. We use this structure as the basis structure for our multiplier architecture.

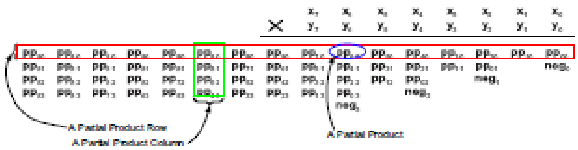

Figure 2. The Array of Partial Products for Signed Multiplication with MBE

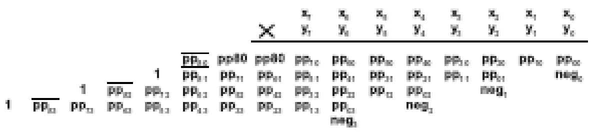

Figure. 3: Application of sign extension prevention measure on the partial product array of 8×8 multiplier

C. One Additional Partial Product Row

However, there is still the problem of having the last *neg* forming one additional partial product row (*neg*3 in Figure3) and this causes another carry save adder delay in order to generate the sums and carries before the final accumulation. This is because in any case, one more partial product means one additional 3-2 reduction. For example, Figure 4(a)[10] shows a 8-input reduction (16 bit × 16 bit multiplication for our architecture) using 4-2 compressors. If we have to reduce 9 inputs (16 bit × 16 bit multiplication for the conventional architecture), one additional carry save adder is required as in Figure 4(b)[10].

.

III . Stopping the Extra Partial Product Row

Therefore, our aim is to remove the last *neg* signal. This would prevent the extra partial product row, and thus save the time of one additional carry save adding stage and the hardware required for the additional carry save adding. We noticed that if we could somehow produce the two's complement of the multiplicand while the other partial products were produced, there would be no need for the last *neg* because this *neg* signal would have already been applied when generating the two's complement of the multiplicand.
Therefore, we "only" need to find a faster method to calculate the two's complement of a binary number.

A . A Quick Method to Find Two's Complements

Our method is an extension of well-known algorithm that two's complementation complements all the bits after the rightmost "1" in the word but keeps the other bits as they are. The two's complement of a binary number 0010102(1010) is 1101102 (−1010). For this number, the right most "1" happens in bit position 1 (the check mark position in Figure 4).

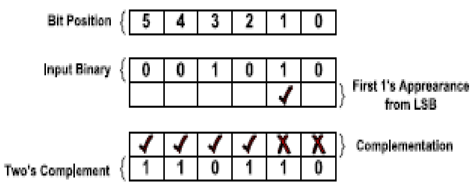

Figure 4:Two Complement Conversion Example

Therefore, values in bit positions 2 to5 can simply be complemented while values in bit positions0 and 1 are kept as they were. Therefore, two's complementation now comes down to finding the conversion signals that are used for selectively complementing some of the input bits. If the conversion signal at any position is "0" (the crosses in Figure 4), then the value is kept as it is and if the conversion signal is "1"(the checks in Figure 4), then the value is complemented. The conversion signals after the rightmost "1" are always1. They are 0 otherwise. Once a lower order bit has been detected to be a "1," the conversion signals for the higher order bits to the left of that bit position should all be "1."However, this searching for the rightmost "1" could as time consuming as rippling a carry through to the MSB since the previous bits information must be transferred to the MSB. Therefore, we must find a method to expedite this detection of the rightmost "1."As we will see, this search for the rightmost "1" can be achieved in logarithmic time using a binary search tree-like structure. We first find the conversion signals for a 2-bitgroup by grouping two consecutive bits (the grouping always starts from the LSB) from the input and finds the conversion signals in each group as shown in Figure 6(a)[10]. Then we find the conversion signals for a 4-bit group (formed by two consecutive 2-bit groups). Then we find the conversion

Signals for a 8-bit group (formed by two consecutive 4-bitgroup). This divide-and-conquer approach is pursued until the whole input has been covered. When grouping two 2*n*-bits groups, the leftmost conversion signals from the right group contain the accumulative information of its group about whether a "1" ever appeared in any bit position of its group, so that a conversion signal should force all the conversion signals from the left group all the way to the "1" if it is itself is a "1." For instance, as shown in Figure 6(b)[10], if *CS*1 (the leftmost conversion signal from the right group) = "1," the conversion signals from the left group (*CS*2 and *CS*3) should be forced to a"1," regardless of their previous values. If *CS*1 = "0," nothing happens to the conversion signals from the left group. This variable control is shown with a dashed arrow. Likewise,*CS*5 may affect conversion signals *CS*6 and *CS*7.The same goes for *CS*3' which may affect the conversion signals (*CS*7', *CS*6', *CS*5', and *CS*4').The inputs to the 2-bit group are bits from the original binary number. However, the inputs to the next level groups are conversion signals from the previous level. For instance, the inputs to the 4-bit group are the conversion signals generated from two 2-bit groups. Therefore, from the second level (4-bit grouping) on, the conversion signals are scanned in order to find the rightmost "1." One possible implementation of our algorithm is shown in Figure7 (a). Figure 7(b)[10] shows another version of the design using NAND, NOR, and inverter. Once we have the complete conversion signals, these signals are shifted left 1 bit and EXOR-ed with the input to create the two's complement of the input. One complete example of two's complementation of "00101000_2"is shown in Figure 8[10]. Our approach is more general and shows better adaptability to any word size.

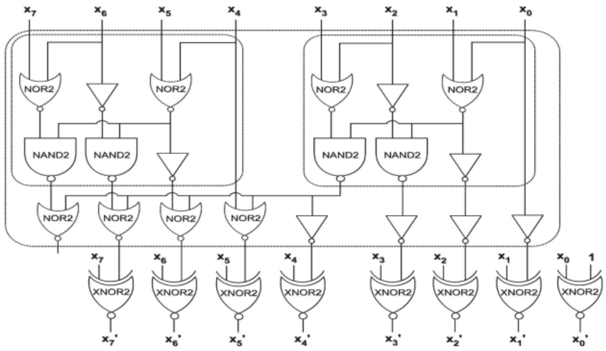

Figure 5: Two's complement computation

A. Putting it all together

By applying the method we just described for two's complementation, the last partial product row (in Figure 3) is correctly generated without the last *neg*(*neg*3 in Figure 3).Now, the multiplication can have a smaller critical path. This avoids having to include one extra carry saving adding stage. It also reduces the time to find the product and saves the hardware corresponding to the carry saving adding stage. Forming a truly parallelogram-shaped partial product array after removing the last *neg* requires undergoing the following steps:

Step a: Replace the last partial product row and *neg*3 in Figure 3 with signals *s*9 ~*s*0 as shown in Figure 6.
Step b: Replace the second to the last partial product row as in Figure 6.

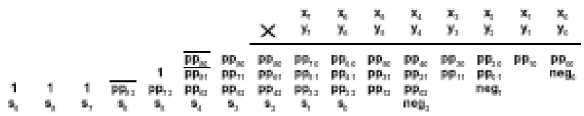

Figure 6: Replacing the last row and the Last *neg* with signals *s*9 -*s*0

Step c: Finally, the MSB of the last row can be complemented (*s*9) and the "1" directly above it can be removed as shown in Figure 7.

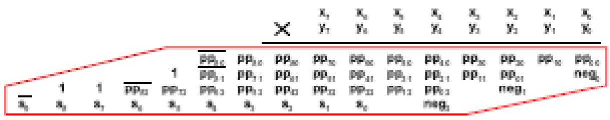

Figure 7: Partial Products After Removing the last *neg*

As can be seen, the critical path column with *n* /2 +1 elements (6*th* bit position of Figure 3 (*n*−2)) now have only *n/*2 elements as shown in Figure 7(the *neg*3 is no longer there). This directly improves the speed of the multiplication. The multiplier architecture to generate the partial products is shown in Figure. The only Difference between our architecture and the conventional multiplier architectures is that for the last partial product row, our architecture has no partial product generation but partial product selection with a two's complement unit. The 3-5 decoder select the correct value from 5 possible inputs (2×X, 1×X, 0, -1×X,-2×X) which are either coming from the two's complement logic or the multiplicand itself and input into the row of 5-1selectors. Unlike the other rows which use PPRG (Partial Product Row Generator), the two's complement logic does not have to wait for MBE to finish. Two's complementation is performed in parallel with MBE and the 3-5 decoders.

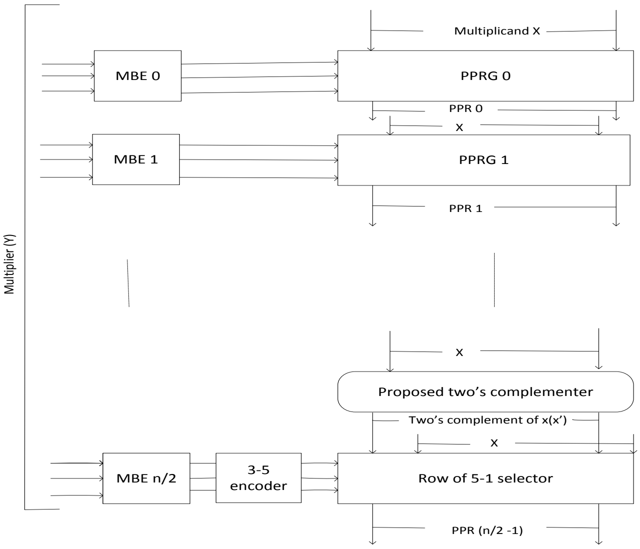

Figure 8. Proposed Multiplier Architecture

IV . Performance Evaluation and Results Discussion

The performance of our multiplier architecture clearly depends on the speed of the two's complementation step. If we can generate the last partial product row of our multiplier architecture within the exact time that the other partial product rows are generated, the performance will be improved as we have predicted because of the removal of the additional partial product row. Therefore, in this section, we evaluate the performance of our two's complement logic by comparing it to the delay of generating other partial products. Then, we investigate the overall impact (in terms of speed) of using our multiplier architecture as compared to previous methods.

The main tools required for this project is MODELSIM 6.4, XILINX 10.1i.By Using these tools we perform simulation and synthesis and get the simulation results and synthesis reports from a two's complement multiplier ppg module, and compare the ppg generation results with our method listed in Table 2.

Our proposed multiplier has generated partial product generation with estimated delay of 9.5 ns, 9.5ns, 9.5ns with corresponding 8×8, 16×16, 32×32 bit multipliers respectively. But the actual critical path delay for the partial product generation in proposed multiplier is 9.321ns; this one is obtained from synthesis report of ppg module. The figure shows the generation of partial product in our proposed multiplier. Hence we concluded here that our approach is reducing computation time in our proposed multiplier. The estimated critical path delay is slightly high when compared to actual critical path delay for generation of partial product in our method. This leads to reduce the maximum combinational path delay of our proposed multiplier.

Table 2: Estimated Critical Path Delay for the Partial Product Generation for various multipliers

Estimated Critical Path Delay for the Partial Product Generation (in Nano seconds)			
Technique	8×8	16×16	32×32
Standard multiplier (any row) (Gen MBE,Gen PPs)	9.8	9.8	9.8
Standard multiplier (first row) (Gen MBE +PPs)	4.8	4.8	4.8
Proposed multiplier (Gen PPs +last*neg)*	9.5	9.5	9.5
Two's complement (4× 1 mux +two's complement tree)	11.9	13.3	15.1

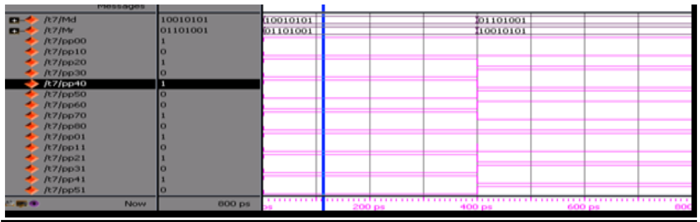

Figure 9 : simulation results for a partial product generation

The developed project is simulated and verified their functionality. Once the functional verification is done, the RTL model is taken to the synthesis process using the Xilinx ISE tool. In synthesis process, the RTL model will be converted to the gate level net list mapped to a specific technology library. Here in this Spartan 3E family, many different devices were available in the Xilinx ISE tool. In order to synthesis this design the device named as "XC3S500E" has been chosen and the package as "FG320" with the device speed such as "-4". There are four Partial product rows km1, km2, km3, km4 are generated. And simulation results for the top module show in figure 10.

We are compared our proposed multiplier with arry multiplier, booth's multiplier and conventional Vedic multiplier. From the table we concluded that our proposed multiplier is an efficient one among all. The Maximum combinational path delays are given table 3.

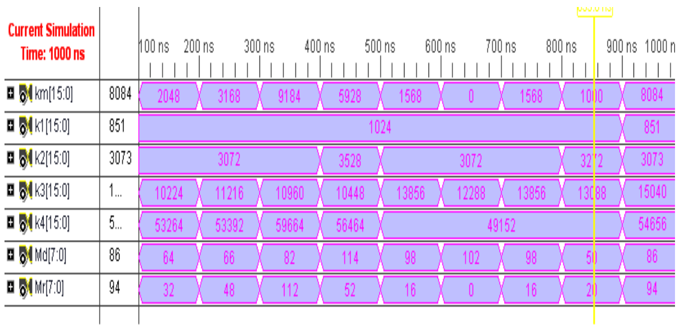

Figure 10: simulation results for top module

Table 3: Comparison of Maximum combinational Path Delay between different multipliers

Maximum Combinational Path Delay for Different Multipliers (in Nano seconds)			
Array Multiplier	Booth's Multiplier	Conventional Vedic Multiplier	Proposed Multiplier
32.01	29.549	23.679	21.995

.The RTL (Register Transfer Logic) can be viewed as black box after synthesize of design is made. It shows the inputs and outputs of the system. By double-clicking on the diagram we can see gates, flip-flops and MUX

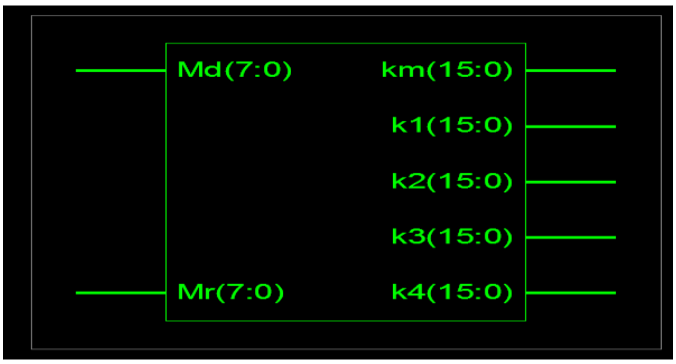

Figure 11 : RTL schematic diagram for test module

The figure shows the technical schematic of top module,

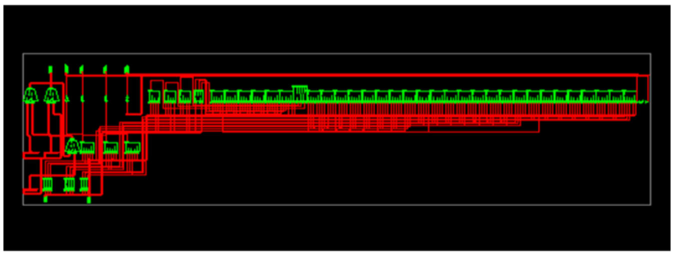

Figure 12 : technology schematic diagram for ppg module

which consists of iob's, lookup tables, functional blocks and flip-flops**.**

Figure 13 : Hardware implementation

The above FPGA implementation shows the multiplication operation, i.e. the multiplier value is 1 & the multiplicand value is 127. Hence the output lights glow from 1 to 7 continuously, which indicates the output value is 127

V . Conclusions

In this project an algorithm is presented to reduce from [*n*/2] +1 to [*n* /2] the number of partial product rows generated during the first step of a multiplication algorithm. By doing so, the structure of the partial product array becomes more regular and easier to implement. Even more importantly, the product is found faster. This can be achieved using less hardware. A detailed and step-by-step approach to prevent the occurrence of the additional row is shown. The proposed multiplication method is particularly efficient when executing the multiplications of the kernels of most common multimedia applications which are based on 8 to 16-bit operands & implemented by using Spartan 3(XC3S400) FPGA.

Compared our approach with a recent proposal with the same aim, considering results using a widely used industrial synthesis tool and concluded that our approach may improve both the performance and area requirements of square multiplier designs. The proposed approach also applies with small modifications to rectangular and to general radix-B Modified Booth Encoding multipliers.

References

[1] M.D. Ercegovac and T. Lang, Digital Arithmetic. Morgan Kaufmann Publishers, 2003.

[2] S.K. Hsu, S.K. Mathew, M.A. Anders, B.R. Zeydel, V.G.Oklobdzija, R.K. Krishnamurthy, and S.Y. Borkar, "A 110GOPS/W 16-Bit Multiplier and Reconfigurable PLA Loop in 90-nm CMOS," IEEE J. Solid State Circuits, vol. 41, no. 1, pp. 256-264, Jan.2006.

[3] H. Kaul, M.A. Anders, S.K. Mathew, S.K. Hsu, A. Agarwal, R.K.Krishnamurthy, and S. Borkar, "A 300 mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45 nm CMOS," IEEE J. Solid State Circuits, vol. 45, no. 1, pp. 95-101, Jan. 2010

[4] M.S. Schmookler, M. Putrino, A. Mather, J. Tyler, H.V. Nguyen, C.Roth, M. Sharma, M.N. Pham, and J. Lent, "A Low-Power, High-Speed Implementation of a PowerPC Microprocessor Vector Extension," Proc. 14th IEEE Symp. Computer Arithmetic, pp. 12-19,1999.

[5] O.L. MacSorley, "High Speed Arithmetic in Binary Computers," Proc. IRE, vol. 49, pp. 67-91, Jan. 1961.

[6] L. Dadda, "Some Schemes for Parallel Multipliers," Alta Frequenza, vol. 34, pp. 349-356, May 1965.

[7] C.S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Trans.Electronic Computers, vol. EC-13, no. 1, pp. 14-17, Feb. 1964.

[8] D.E. Shaw, "Anton: A Specialized Machine for Millisecond-Scale Molecular Dynamics Simulations of Proteins," Proc. 19th IEEE Symp. Computer Arithmetic, p. 3, 2009.

[9] J.-Y. Kang and J.-L. Gaudiot, "A Simple High-Speed Multiplier Design," IEEE Trans. Computers, vol. 55, no. 10, pp. 1253-1258, Oct.2006.

[10] J.-Y. Kang and J.-L. Gaudiot, "A Fast and Well-Structured Multiplier," Proc. Euromicro Symp. Digital System Design, pp. 508-515, Sept. 2004.
[11] F. Lamberti, N. Andrikos, E. Antelo, and P. Montuschi,"Speeding-Up Booth Encoded Multipliers by Reducing the Size of Partial Product Array," internal report, http://arith.polito.it/ir_mbe.pdf, pp. 1-14, 2009.
[12] E.M. Schwarz, R.M. Averill III, and L.J. Sigal, "A Radix-8 CMOSS/390 Multiplier," Proc. 13th IEEE Symp. Computer Arithmetic,pp. 2-9, 1997.
[13] W.-C. Yeh and C.-W. Jen, "High-Speed Booth Encoded Parallel Multiplier Design," IEEE Trans. Computers, vol. 49, no. 7, pp. 692-701, July 2000.
[14] Z. Huang and M.D. Ercegovac, "High-Performance Low-Power Left-to-Right Array Multiplier Design," IEEE Trans. Computers,vol. 54, no. 3, pp. 272-283, Mar. 2005.
[15] R. Zimmermann and D.Q. Tran, "Optimized Synthesis of Sum-of-Products," Proc. Conf. Record of the 37th Asilomar Conf. Signals,Systems and Computers, vol. 1, pp. 867-872, 2003.
[16] V.G. Oklobdzija, D. Villeger, and S.S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," IEEE Trans.Computers, vol. 45, no. 3, pp. 294-306, Mar. 1996.
[17] P.F. Stelling, C.U. Martel, V.G. Oklobdzija, and R. Ravi, "Optimal Circuits for Parallel Multipliers," IEEE Trans. Computers, vol. 47, no. 3, pp. 273-285, Mar. 1998.
[18] J.-Y. Kang and J.-L. Gaudiot, "A Logarithmic Time Method for Two's Complementation," Proc. Int'l Conf. Computational Science, pp. 212-219, 2005.
[19] K. Hwang, Computer Arithmetic Principles, Architectures, and Design.Wiley, 1979.
[20] R. Hashemian and C.P. Chen, "A New Parallel Technique for Design of Decrement/Increment and Two's Complement Circuits,"Proc. 34th Midwest Symp. Circuits and Systems, vol. 2,pp. 887-890, 1991.
[21] D. Gajski, Principles of Digital Design. Prentice-Hall, 1997.
[22] STMicroelectronics, "130nm HCMOS9 Cell Library," http://www.st.com/stonline/products/technologies/soc/evol.htm, 2010.

Kolathur Lakshmipathy received the B.Tech. Degree in electronics and Communication engineering from SANA Engineering College, affiliated to Jawaharlal Nehru Technological University Hyderabad , AP, India, in 2007. He received M.Tech Degree in VLSI system design From VNR Vignana Jyothi Institute of Engineering & technology, Bachpally, Hyderabad, India,in 2012. His research interests include Digital design with FPGA and Low Power vlsi.

Voruganti Santhosh Kumar received the B.Tech. Degree in electronics and Communication engineering from SANA Engineering College, affiliated to Jawaharlal Nehru Technological University Hyderabad, AP, India, in 2007.he received M.Tech Degree in VLSI system design From Anurag Engineering college, kodad, AP, India. He is an assistant professor in Dept of ECE in Anurag College of engineering. His research interests include Digital design with FPGA.

Dr. V. Padmaja born in 1968. She received B.E Degree in Electronics and Communications Engineering, M.E Degree in Digital Systems Engineering from O.U in 1991 and 1999 respectively. She received Ph.D from J.N.T.U in 2009. She is an Professor in Dept. of ECE in VNRVJIET, her research of interest includes image processing and Embedded Systems. She has authored more than 5 Research papers in National and International Conferences and Journals