A Comparison of Layout Implementations of Pipelined and Non-Pipelined Signed Radix-4 Array Multiplier and Modified Booth Multiplier Architectures

Leonardo L. de Oliveira[1], Cristiano Santos[2], Daniel Ferrão[2], Eduardo Costa[3], José Monteiro[4], João Baptista Martins[1], Sergio Bampi[2], Ricardo Reis[2]

[1] Federal University of Santa Maria, PPGEE – GMICRO, Av. Roraima 1000, Camobi, 97105-900 Santa Maria – RS, Brazil, leonardo@mail.ufsm.br, batista@inf.ufsm.br
WWW home page: http://www.ufsm.br/gmicro

[2] Federal University of Rio Grande do Sul, PPGC – GME, Av. Bento Gonçalves, 9500, Agronomia, 91501-970 Porto Alegre – RS, Brazil, {clsantos,dlferrao,bampi,reis}@inf.ufrgs.br
WWW home page: http://www.inf.ufrgs.br/gme

[3] Catolic University of Pelotas, Rua Félix da Cunha 412, 96010-000 Pelotas – RS, Brazil, ecosta@atlas.ucpel.tche.br
WWW home page: http://www.ucpel.tche.br

[4] INESC-ID/IST, Rua Alves Redol 9, 1000-029 Lisboa – Portugal, jcm@inesc-id.pt
WWW home page: http://www.inesc-id.pt

Abstract. This paper presents performance comparisons between two multipliers architectures. The first architecture consists of a pure array multiplier that was modified to handle the sign bits in 2's complement and uses a radix-4 encoding to reduce the partial product lines. The second architecture implemented was the widely used Modified Booth multiplier. We describe a design methodology to physically implement these architectures in a pipelined and non-pipelined form, obtaining area, power consumption and delay results. Up to now only results at the logic level were presented in previous work. The performance of pipelined array architecture is compared with the pipelined Modified Booth. We compare the physical implementations in terms of area, power and delay. The results show that the new pipelined array multiplier can be significantly more efficient, with close to 16% power savings and 55% power savings when considering non-pipelined architectures.

De Oliveira, L.L., Santos, C., Ferrão, D., Costa, E., Monteiro, J., Martins, J.B., Bampi, S., Reis, R., 2007, in IFIP International Federation for Information Processing, Volume 240, VLSI-SoC: From Systems to Silicon, eds. Reis, R., Osseiran, A., Pfleiderer, H-J., (Boston: Springer), pp. 25–39.

1 Introduction

Multiplier modules are common to many DSP applications. The fastest types of multipliers are parallel multipliers. Among these, the Wallace multiplier [18] is among the fastest. However, they do not have such a regular structure as the conventional array [11] or Booth [13] multipliers. Hence, when layout regularity, high-performance and low power are primary concerns, Booth multipliers tend to be the primary choice [2], [7], [9], [13], [16].

In this paper, we present layout implementations for both the Modified Booth multiplier and the new array multiplier in non-pipelined and pipelined versions. The pipelined version of the radix-4 architecture was implemented in order to reduce both the critical path and useless signal transitions that are propagated through the array. This array architecture is extended for radix 2^m encoding, which leads to a reduction of the number of partial lines, enabling a significant improvement in performance and power consumption.

We synthesize the multipliers by using an automatic synthesis tool, named TROPIC [15]. In order to compare the Modified Booth and the array architectures, both using radix-4, the ELDO – a spice simulator, part of the Mentor Graphics environment, was used. The results show that the new array multiplier is significantly more efficient, saving more than 50% in power consumption. This result is very close to the results reported in [4], obtained at the logic level using a switch-level simulator and 16% power savings considering pipelined versions.

The power reduction presented by the new array multiplier is mainly due to the lower logic depth, which has a big impact in the amount of glitching in the circuit. We should stress further that, in contrast to the architecture presented in [4], rasing the radix for the Booth architecture is a difficult task, thus not being able to leverage from the potential savings of higher radices.

This paper is organized as follows. In the next section we give an overview of relevant work related to our work. In section 3 we present a 2's complement binary multiplication. After that, Section 4 briefly describes the radix-4 array multiplier. The Modified Booth multiplier and their pipelined forms are described in Section 5. Section 6 describes the design methodology and how area, power and delay results are obtained. Comparisons between the radix-4 array multiplier architecture and the Modified Booth, for both switch level and electrical level are presented in Section 7. Finally, in Section 9 we conclude this paper, discussing the main contributions and future work.

2 Related Work

A substantial amount of research work has been put into developing efficient architectures for multipliers given their widespread use and complexity. Schemes such a bisection, Baugh Wooley and Hwang [9] propose the implementation of a 2's complement architecture, using repetitive modules with uniform interconnection patterns. However, an efficient VLSI realization is more difficult due to the irregular tree-array form used. The same non-regularity aspect is observed in [13], where a scheme of a multiplexer–based multiplier is presented. In [11] an improvement of

this technique is observed where the architecture has a more rectangular layout than [13].

The techniques described above have been applied to conventional array multipliers whose operation is performed bit by bit and some times the regularity of the multipliers is not preserved. More regular and suitable multiplier designs based on the Booth recoding technique have been proposed [7][2][16]. The main purpose of these designs is to increase the performance of the circuit by the reduction of the number of partial products. In the Modified Booth algorithm approximately half of the partial products that need added is used.

Although the Booth algorithm provides simplicity, it is sometimes difficult to design higher radices due to the complexity to pre-compute an increasing number of multiples of the multiplicand within the multiplier unit. In [7][16] high performance multipliers based on higher radices are proposed. However, these circuits have little regularity and no power savings are reported. Research work that directly targets power reduction by using higher radices for the Booth algorithm is presented in [2][10]. Area and power improvements are reported with a highly optimized encoding scheme ate the circuit level. At this level of abstraction some other works have applied complementary pass-transistor logic in their design in order to improve the Booth encoder and full adder circuits [9][13][14].

In our work, the improvement in power has the same principal source as the Booth architecture, the reduction of the partial product terms, while keeping the regularity of an array multiplier. We show that our architecture can be more naturally extended for higher radices using less logic levels and hence presenting much less spurious transitions. We present layout implementation of pipelined and non-pipelined versions of our multipliers.

3 Array Multipliers

In this section we describe how we derive the 2's complement binary multiplication. Consider two operands W-bits wide, $A = \sum_{i=0}^{W-1} a_i 2^i$ and $B = \sum_{j=0}^{W-1} b_j 2^j$. We have that

$$A \times B = \sum_{j=0}^{W-1} A \cdot b_j 2^j \tag{1}$$

where in turn,

$$A \cdot b_j = \sum_{i=0}^{W-1} b_j \cdot a_i 2^i \tag{2}$$

A conventional array multiplier [3] translates this expression directly to hardware, where we have the *W* partial product rows from Equation 1, each made of *W* bit level products as in Equation 2, which can be arranged in a simply, very regular, array structure. Each bit product is simply an AND gate.

The conventional array multiplier is only applicable to unsigned operands. We are able to show that exactly the same architecture can be used on signed operands in 2's complement with very little changes.

2's complement is the most used encoding for signed operands. The most significant bit, a_{W-1}, is the sign bit. If the number A is positive, its representation is the same as for an unsigned number, simply A. If the number is negative, it is represented as $2^W - A$.

Conversely, the value of the operand can be computed as follows:

$$A = \begin{cases} A & , \quad a_{W-1} = 0 \\ A - 2^W & , \quad a_{W-1} = 1 \end{cases} \tag{3}$$

We make the following observation that enables us simplify our architecture. Let us define $A' = \sum_{i=0}^{W-2} a_i 2^i$, an unsigned value. For positive numbers, $a_{W-1} = 0$, hence the value represented by A is A'. For negative numbers, $a_{W-1} = 1$, hence this value is $A - 2^W = (2^{W-1} + A') - 2^W = A' - 2^{W-1}$. Then equation 3 becomes:

$$A = \begin{cases} A' & , \quad a_{W-1} = 0 \\ A' - 2^{W-1} & , \quad a_{W-1} = 1 \end{cases} \tag{4}$$

or simply $A = A' - a_{W-1} 2^{W-1}$.

What Equation 4 tell us is that the multiplication of two operands in 2's complement can be performed as an unsigned multiplication for $(W-1)^2$ of the bit products. Let us consider the 4 possible scenarios for $A \times B$:

$$\begin{aligned} &A > 0, B > 0: \quad A' \times B' \\ &A > 0, B < 0: \quad A' \times B' - A' 2^{W-1} \\ &A < 0, B > 0: \quad A' \times B' - \sum_{j=0}^{W-1} b_j 2^{W-1+j} \\ &A < 0, B < 0: \quad A' \times B' - A' 2^{W-1} - \sum_{j=0}^{W-1} b_j 2^{W-1+j} \end{aligned} \tag{5}$$

which can be reduced to

$$A \times B = A' \times B' - b_{W-1} A' 2^{W-1} - a_{W-1} \sum_{j=0}^{W-1} b_j 2^{W-1+j} \tag{6}$$

The form of Equation 6 highlights:

- from the first term, that the *W-1* least significant bits A and B can be treated exactly as an unsigned array multiplier;
- from the second term, that the last row of the multiplier is either non-existent (B>0) or a subtracter of A' shifted by W-1 bits (B<0);
- from the third term, that, at each partial product line, the most significant bit is either 0 (A>0) or -1 (A<0).

Consider now $A' = \sum_{i=0}^{\frac{W}{m}-2} a_i 2^{i \cdot m}$, where a_i is a m-bit digit. For positive numbers, the value represented by A is A' as before. For negative numbers, this

value is $A - 2^W = a_{\frac{W}{m}-1}2^{W-m} + A' - 2^W = A' - a_{\frac{W}{m}-1}2^{W-m}$, since $a_{\frac{W}{m}-1}2^{W-m} - 2^W$ is the 2's complement of $a_{\frac{W}{m}-1}2^{W-m}$. Then we have:

$$A = \begin{cases} A' & , \quad a_{W-1} = 0 \\ A' - a_{\frac{W}{m}-1}2^{W-m} & , \quad a_{W-1} = 1 \end{cases} \tag{7}$$

or simply

$$A = A' - a_{W-1}a_{\frac{W}{m}-1}2^{W-m} \tag{8}$$

Using analogous observations as made for the binary case, from Equation 8 we can write:

$$A \times B = A' \times B' - A'b_{W-1}b_{\frac{W}{m}-1}2^{W-m} - a_{W-1}a_{\frac{W}{m}-1}\sum_{j=0}^{\frac{W}{m}-1} b_j 2^{W-m+j} \tag{9}$$

4 Radix-2^m Array Multiplier

In this section, we summarize the methodology of [5] for the generation of regular structures for arithmetic operators using signed radix-2^m representation and extend it into a pipelined version [6].

For the operation of a radix-2^m multiplication, the operands are split into groups of *m* bits. Each of these groups can be seen as representing a digit in a radix-2^m. Hence, the radix-2^m multiplier architecture follows the basic multiplication operation of numbers represented in radix-2^m. The radix-2^m operation in 2's complement representation is given by Equation 10.

$$R \times Y = R' \times Y' - R'y_{W-1}y_{\frac{W}{m}-1}2^{W-m} - r_{W-1}r_{\frac{W}{m}-1}\sum_{j=0}^{\frac{W}{m}-1} y_j 2^{W-m+j} \tag{10}$$

where R and Y are two operands W-bits wide; r_{W-1} is the most significant bit (is the sign bit); and $R' = \sum_{i=0}^{W-2} r_i 2^i$.

This operation is illustrated in Fig. 1. For the *W-m* least significant bits of the operands unsigned multiplication can be used. The partial product modules at the left and bottom of the array need to be different to handle the sign of the operands.

For this architecture, three types of modules are needed, as shown in Fig. 2. Type I are the unsigned modules. Type II modules handle the *m*-bit partial product of an unsigned value with a 2's complement value. Finally, Type III are modules that operate on two signed values. Only one Type III module is required for any type of multiplier, whereas $2\frac{W}{m}$ - 2 Type II modules and $(\frac{W}{m} - 1)^2$ Type I modules are needed. Fig. 6 shows an example of an 8-bit wide 2's pipelined complement radix-4 array multiplier.

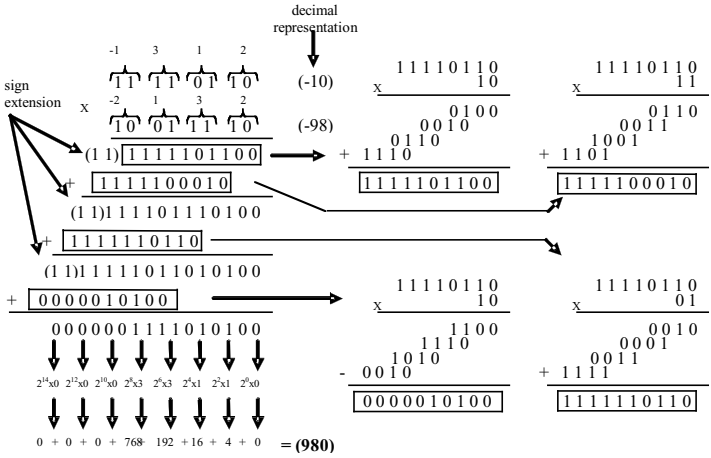

Fig. 1. Example of a 2's complement 8-bit wide radix-4 multiplication

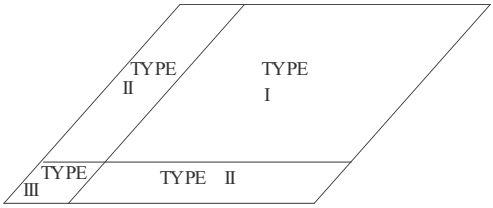

Fig. 2. General structure for a 2's complement radix-2m multiplier

We present a summarized example for W=8 bit wide operands using radix-4 (m=2) in Fig. 3.

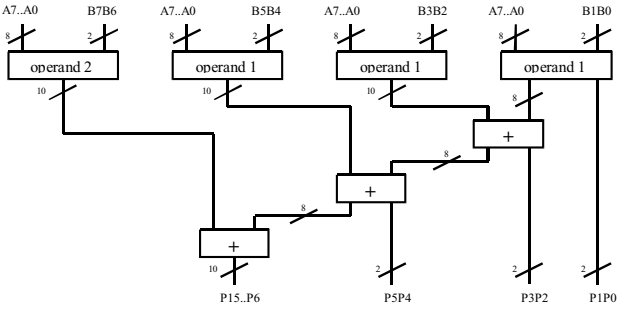

Fig. 3. 8-bit wide 2's complement Binary array multiplier m=2

Figure 4 and Figure 5 show the structure of operands 1 and 2, their inputs and outputs and nearest connections between them and the blocks of adders. In additional they show the sign extension that has been used in operands 1 and 2.

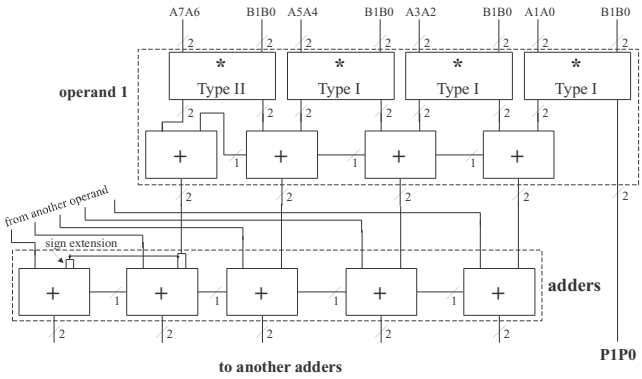

Fig. 4. Operand 1 and connections to first line of adders showing the sign extension

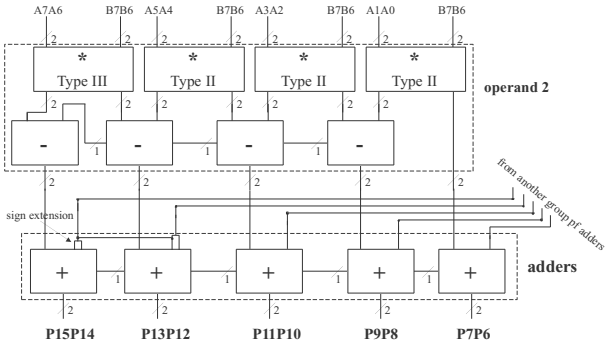

Fig. 5. Operand 2 and connections to third line of adders showing the sign extension

4.1 Pipelined Array Multiplier

Glitches are unwanted switching activities that occur before a signal settles to its intended value. Each clock edge changes the inputs to the combinatorial logic between registers and every node has a different delay from different inputs, which change their state several times before settling down. Glitches on a node are dependent on the logic depth to that node, i.e. the number of logic gates from the node to the primary inputs (or sequential elements). The deeper and wider the logic behind a node, the more it glitches. These glitches can be reduced by reducing the depth of logic levels

The regularity of this array architecture makes it suitable for the application of other power reducing techniques. A pipelined version was constructed in order to reduce the critical path and useless signal transitions that are propagated through the array. The doted lines in Fig. 6 show the pipelined version of the radix-4 array multiplier for 8-bit operands. As can be observed, the advantage of the layered structure of the array was taken into account and two layers of registers were introduced. Thus, 3 clock cycles are necessary to perform the computation considering 8-bit architectures.

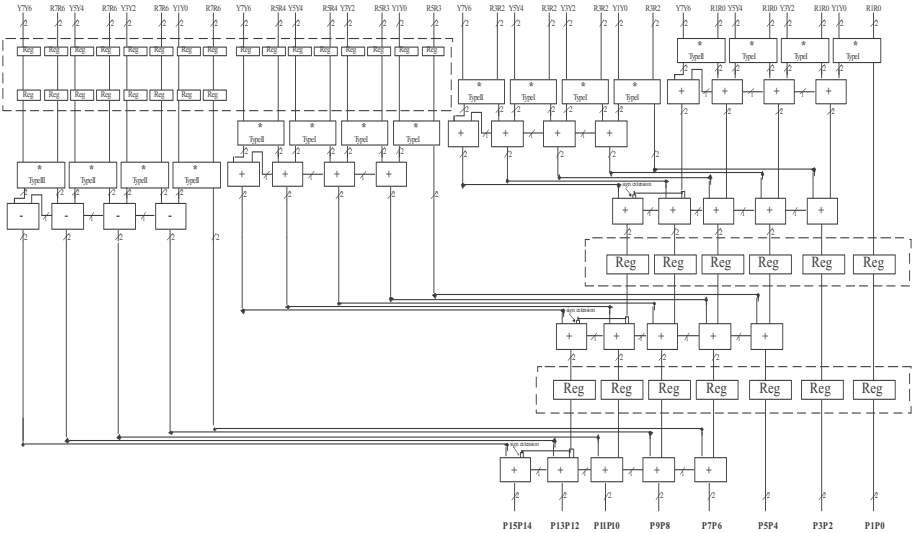

Fig. 6. Example of an 8-bit wide 2's complement radix-4 array multiplier

5 Modified Booth multiplier

The radix-4 Booth's algorithm (also called Modified Booth) has been presented in [5]. In this architecture it is possible to reduce the number of partial products by encoding the two's complement multiplier. In the circuit the control signals (0, +Y, +2Y, -Y and -2Y) are generated from the multiplier operand Y for each 3-bit group, as shown in the example of Fig. 7, for an 8-bit wide operation. A multiplexer produces the partial product according to the encoded control signal.

Common to both architectures is that, at each step of the algorithm, two bits are processed. However, the basic Booth cells are not simple adders as in the array multiplier, but must perform addition-subtraction-no operation and controlled left-shift of the bits of the multiplicand. Fig. 8, shows an example of an 8-bit modified Booth architecture.

5.1 Pipelined Modified Booth Multiplier

A pipelined Modified Booth by introducing registers along the layers of the array was implemented in and it is presented in Fig. 8. As it can be observed in this figure, there are two layers of registers along the array as in the binary array multiplier with m=2. Again, 3 clock cycles are required to compute the final result in the 8-bit architecture and six cycles to the 16-bit one. Moreover, common to both architectures is that the registers are inserted at the output of the adders which are responsible for adding the partial product terms. However, in the Booth multiplier it is also necessary to introduce registers in the output of the encoders to perform the correct operation of each clock cycle as shown in Fig. 8.

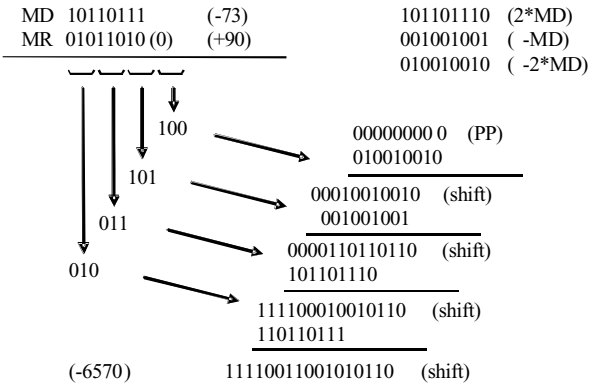

Fig. 7. Example of an 8-bit multiplication with Modified Booth algorithm

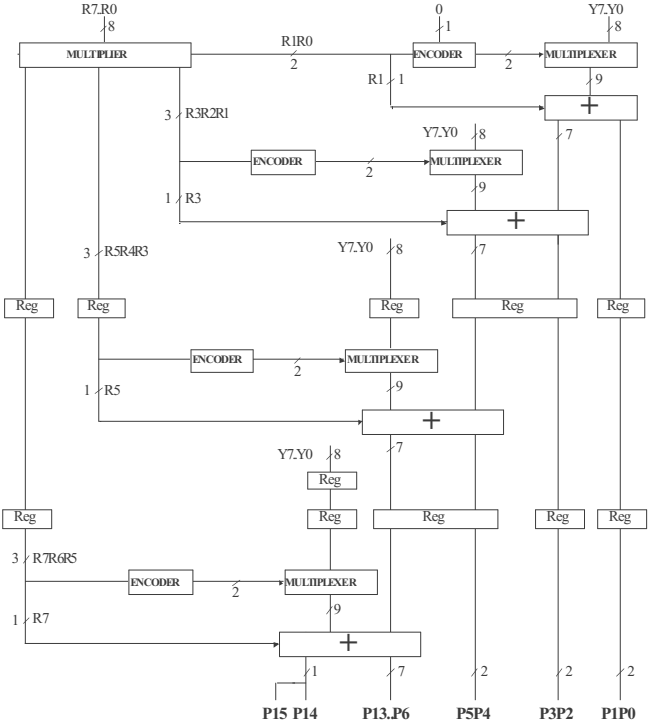

Fig. 8. 8-bit pipelined modified Booth architecture

6 Design Methodology

Fig. 9 shows the design flow used in the physical implementation of the multipliers. Two methodologies are presented: our methodology (black), and the methodology used in [7] and [8] with the SIS environment (gray). The multipliers were originally described in BLIF (Berkeley Logic Interchange Format). Thus, these BLIF files are used as input of the design flow, as can be observed in Fig. 9.

In [5] and [6], the performance of the multipliers was evaluated only in a logic level. The SIS [17] tool was used to synthesize and estimate area and delay of the multipliers while power consumption was estimated using the switch-level simulator SLS [8].

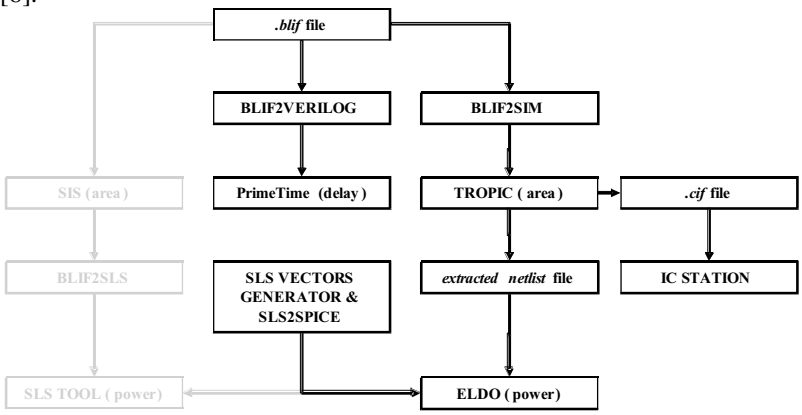

Fig. 9. Design tools for synthesis and performance estimation

In this work, the TROPIC tool was used for the physical synthesis of the implemented multipliers. This tool uses a spice like format (*sim*) as input and performs a library-free automatic layout generation of the circuit regarding the design rules of the target technology. TROPIC gives the total area occupied by the layout and the number of transistors of the synthesized circuits. Before the layout synthesis of the circuits, it is necessary to set the size of the transistors and the number of rows. This last parameter is useful to set the aspect ratio width/height.

Since the TROPIC tool generates the widely used *cif* format, the resulting circuit layout can be visualized with Mentor Graphics IC Station tool. Fig. 10 shows the layout for the 8-bit array multiplier, which was generated automatically by TROPIC tool. Once the *cif* file is generated, an electrical extraction can be performed using the TROPIC tool.

The extracted SPICE netlists were simulated using the ELDO electrical simulator in order to obtain power estimation at the back-annotated electrical level. This simulator is part of the Mentor Graphics environment for power estimation. The same set of input vectors used in [4] and [5] for power estimation was converted from SLS to SPICE format and then used for transient analysis.

The timing analysis tool PrimeTime [12] was used to estimate the critical delay of the circuits. PrimeTime is able to perform both static and functional timing analysis. Static timing analysis (STA) is the standard approach used for delay estimation in the current designs complexity. The main issue of this approach is that logic information about the cells of the circuit is not considered during the critical delay search. At the same time that this issue makes the delay estimation faster, it can make STA suffers from the false path syndrome. In order to avoid this false path syndrome, the designer must report all timing exceptions of the circuit to the STA tool, and it can be a very hard task.

Another way to avoid false paths during delay estimation is using functional timing analysis (FTA). FTA performs the critical delay search taking into account information about the logic cells of the circuit. So, paths that can not propagate a

transition are not considered and the critical delay will be the delay of the longest sensitizable path. Primetime uses the Exact Floating Mode sensitization criterion during the critical path search. This sensitization criterion considers both logic and timing information of the cells during the path sensitization.

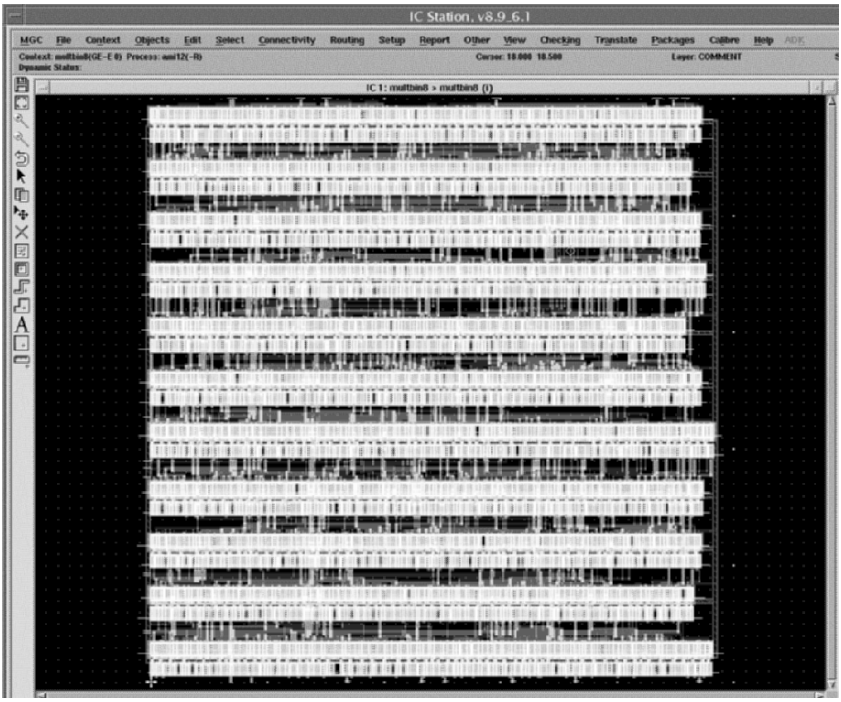

Fig. 10. Layout of an 8-bit array multiplier generated automatically by TROPIC

7 Performance Comparisons

In this section, we present area, delay and power results for the 16-bit multipliers after layout generation. The circuits were implemented using HCMOS 0.25μm technology and the same transistor size (WP=5μm and WN=3μm). Area results were obtained using the TROPIC layout generation tool and are presented both in terms of total area and in terms of number of transistors. Power consumption was estimated through electrical simulation using ELDO simulator and applying a random pattern signal with 100 input vectors. Power results are presented in terms of average power consumption. PrimeTime was used to perform static and functional timing analysis and both delay results are presented. We have not applied yet any transistor-level techniques which can further improve the efficiency of booth architectures.

7.1 Pipelined and Non-Pipelined Results

Table 1 presents area results for 16-bit radix-4 Booth and the new array multiplier proposed in [6], both implemented in layout level.

Table 1. Area results for 16-bit parallel multipliers

	Parameter	Array	Booth	Diff(%)
non-pipelined	Number of transistors	12484	10064	-19.4
	Total area (mm^2)	0.2872	0.2172	-24.4
pipelined	Number of transistors	23014	21220	-7.8
	Total area (mm^2)	0.4829	0.4608	-4.6

As it can be observed in Table 1, the array multiplier presents the highest area and number of transistors. This occurs due to the fact that the partial product lines operate on group of *m* bits and the basic multiplier elements, which compose the modules for the product terms, are slightly more complex. The introduction of registers along the layers of the arrays increases the area of both architectures when compared to the non-pipelined architectures as shown in Table 1. Although the array multiplier presents the highest area value, this architecture can be slightly more efficient in terms of delay result as presented in Table 2. This is due to the lower logic depth presented by our proposed architecture.

Table 2. Delay results for 16-bit parallel multipliers

		Array	Booth	Diff (%)
pipelined	FTA	9.80ns	10.59ns	+8.06
	STA	9.86ns	10.61ns	+7.60
non- pipelined	FTA	17.75ns	18.97ns	+6.87
	STA	18.26ns	19.59ns	+7.28

Fig. 1 and Fig. 8 show that while in the pipelined array multiplier the critical path is given by a *m*=2 multiplier module and 2 full adders, in the pipelined Modified Booth, the critical path includes the encoder, an operand circuit composed by a multiplexer and a full adder. These circuits produce a large number of interconnections and a longer delay per row. Thus, the array multiplier presents less delay values than the Modified Booth even in the pipelined version as shown in Table 2.

As observed in [1], the major sources of power dissipation in multipliers are spurious transitions and logic races that flow through the circuit. Thus, the significantly less amount of spurious transitions in the new array multiplier justifies the gain in power when compared against the Booth multiplier as shown in Table 3. Moreover, the new array multiplier presents less logic depth due to the more balanced paths to the basic blocks that compose the array architecture. This contributes for improvement in power reduction because of the less generation of useless transitions. Our architecture is more efficient in reducing glitching and hence reducing power, as the results in Table 3 demonstrate. It is also apparent that our 6-stage pipelining for the 16-bit multiplier is not optimum, as the power increase demonstrates for the pipelined version of both multiplier architectures. It is also apparent that our architecture is more power efficient for a smaller number of pipeline stages, when compared to the Modified Booth. All power results are for the same pipeline frequency (50MHz).

This occurs because in the pipelined approach glitching is reduced significantly. This reduction will have a greater impact in the case where the glitching was higher. However, the reduced logic depth and delay presented by our architecture still makes it significantly more efficient, as shown in Table 3.

Table 3. Power dissipation for 16-bit parallel multipliers at Vdd=2.5V and freq=50MHz

	Array (*mW*)	Booth (*mW*)	Diff (%)
pipelined	14.76	17.12	+16.0
non-pipelined	10.76	16.75	+55.7

7.2 Comparison between Electrical and Logic Results

Table 4 shows area, delay and power percentage changes between the pipelined and non-pipelined array and Modified Booth multipliers. The estimates at the logic level and after layout correlate well for power. Area estimates at the logic level is just the number of literals coming from logic synthesis (SIS environment). Delay at the logic level was also estimated in SIS environment by using *mcnc* library. The relative power estimations are fairly close as shown in Table 4. In the logic level power results were obtained by using a random pattern input signal with 10,000 input vectors. The larger number of glitches generated in the Modified Booth makes this architecture more power consuming in both pipelined and non-pipelined version, which is captured with the SLS simulator. This validates the results reported in [5] and [6] at gate level design.

Table 4. Comparison between parallel multipliers in electrical and logic simulations

	pipelined		non-pipelined	
Parameter	Logic Level	Electrical Level	Logic Level	Electrical Level
Area (n. of transistors)	-14.4%	-7.8%	-20.2%	-19.4%
Delay (ns)	+15.2%	+8.06%	+1.1%	+6.87%
Power (*mW*)	+18.7%	+16.0%	+54.0%	+55.7%

8 Conclusions

We have described the layout implementation of a new array multiplier and Modified Booth multiplier both in pipelined and non-pipelined versions operating in 2's complement numbers using radix-2^m encoding. We have presented results that show significant improvement in power consumption in the new pipelined and non-pipelined array multiplier. We have compared the new array and Modified Booth multipliers simulated both at the logic and electrical levels. The results showed that the relative values at the two levels of abstraction are similar when we compare the

power consumption of the multipliers. As future work we hope to be able to prototype these architectures in order to experimentally validate these results.

9 Acknowledgments

The support of CNPq, PDI-TI-CTINFO, FAPERGS and FCT is gratefully acknowledged.

10 References

[1] Callaway, T.; Swartzlander, E. Optimizing multipliers for WSI. In Fifth Annual IEEE International Conference on Wafer Scale Integration, pages 85-94, 1993.

[2] Cherkauer, B; Friedman, E. A Hybrid Radix-4/Radix-8 Low Power, High Speed Multiplier Architecture for Wide Bit Widths. In IEEE International Symposium on Circuits and Systems, volume 4, pages 53–56, 1996.

[3] Wang, Y.; Jiang, Y.; Sha, E. On Area-Efficient Low Power Array Multipliers. In the 8th IEEE International Conference on Electronics, Circuits and Systems, pages 1429-1432, 2001

[4] Costa E. da; Monteiro J., and S. Bampi. A New Architecture for 2's Complement Gray Encoded Array Multiplier. In Proceedings Symposium on Integrated Circuits and Systems, pages 14-19, 2002.

[5] Costa, E., Monteiro, J., Bampi, S. A New Architecture for Signed Radix-2^m Pure Array Multiplier. IEEE ICCD, September 2002.

[6] Costa, E., Bampi, S., Monteiro, J. A New Pipelined Array Architecture for Signed Multiplication. 16th SBCCI, September 2003.

[7] Gallagher, W. and Swartzlander, E. High Radix Both Multipliers Using Reduced Area Adder Trees. In Twenty-Eighth Asilomar Conference on Signals, Systems and Computers, volume I, pages 545-549, 1994.

[8] Genderen, A. J. SLS: An Efficient Switch-Level Timing Simulator Using Min-Max Voltage Waveforms. Proceedings of VLSI Conference, pages 79-88, 1989.

[9] Goto, G.; et al. A 4.1-ns Compact 54 x 54-b Multiplier Utilizing Sign-Select Booth Encoders. IEEE Journal of Solid-State Circuits, 32:1676-1682, 1997.

[10] Goldovsky and et al. Design and Implementation of a 16 by 16 Low Power Two's Complement Multiplier. In IEEE International Symposium on Circuits and Systems, volume 5, pages 345-348, 2000.

[11] Hwang, K. Computer Arithmetic - Principles, Architecture and Design. John Wiley & Sons, 1979.

[12] Synopsys PrimeTime Design Reference Manual, 2004.

[13] Khater, I.; Bellaouar, A.; Elmasry, M. Circuit Techniques for CMOS Low-Power, High-Performance Multipliers. IEEE Journal of Solid-State Circuits, 31:1535-1546, 1996.

[14] Yano, K. and et al. A 3.8-ns CMOS 16 x 16-b Multiplier Using Complementary Pass Transistor Logic. Journal of solid-State Circuits, 25:388-395, 1990.

[15] Moraes, F. A Virtual CMOS Library Approach for Fast Layout Synthesis. In: IFIP TC10 WG10.5 International Conference on Very Large Scale Integration, 10, pages 415-426, 1999.

[16] Seidel, P., Mcfearin, L. and Matula, D. Binary Multiplication Radix-32 and Radix-256. In 15th Symposium on Computer Arithmetic, pages 23–32, 2001.

[17] Sentovich, E. and et al. SIS: A System for Sequential Circuit Synthesis. Technical report, University of California at Berkeley, UCB/ERL – Memorandum nº M92/41, 1992.

[18] Wallace, C. A Suggestion for a Fast Multiplier. IEEE Transactions on Electronic Computers, 13:14–17, 1964.