The Nucleus **50**, No. 4 (2013) 351-358

The Nucleus
A Quarterly Scientific Journal of Pakistan Atomic Energy Commission
NCLEAM, ISSN 0029-5698

HIGH-LEVEL POWER OPTIMIZATION FOR ARRAY MULTIPLIERS

Y. A. DURRANI

Department of Electronic Engineering, University of Engineering and Technology, Taxila, Pakistan

(Received July 09, 2013 and accepted in revised form November 19, 2013)

Multiplication is the basic operation in most arithmetic features in computing systems. Generally multiplier occupies large area, long delay and high power dissipation. Therefore, low power multiplier design has been an important part in very large scale integrated (VLSI) design. Power consumption is directly related to data switching patterns and it is difficult to consider high-level application-specific data characteristics in power optimization. In this paper, we present a feasible method of pipelined array multiplier and evaluated the results by the flexible estimation methods at register transfer level (RTL). The multiplier architecture is for low power and high speed applications. The experimental results indicate that the internal optimization using pipelined technique reduces the power consumption of the circuit considerably.

Keywords : Array multiplier, Pipelining, RTL, Power estimation, Critical path, Power dissipation

1. Introduction

As the scale of integration keeps growing, more and more sophisticated digital systems are being implemented on a VLSI chip. The portable devices not only demand great computation capacity but also consume considerable amount of energy. While performance and area remain to be two major design goals, power consumption has become a critical concern in today's system design. The need of low power VLSI systems arises from two main factors. First, with the steady growth of operating frequency and the processing capacity per chip, large current has to be delivered and the heat due to large power consumption must be removed by proper cooling techniques. Second, the battery life in portable devices is limited.

This paper addresses the high-level optimization technique for low power multipliers. High-level techniques refer to RTL approach that consider multiplication's arithmetic features and input data characteristics. The main research hypothesis of this work is that high-level optimization of multiplier design produces more power-efficient solutions than optimization only at low-levels. Specifically, we consider how to optimize the internal architecture of multipliers and how to control the active multiplier resource to match the external data characteristics. Our primary objective is the power reduction with the smaller area and the minimum delay. By using RTL, it is possible to achieve both power reduction and area/delay reduction, which is the strength of high- level optimization. The trade-off between power, area and delay is also considered in some cases.

Several approaches have been proposed for positive numbers with two's complement form [1-3]. The basic idea was the fast implementation of the addition of the partial products. For this purpose, the Carry Save Addition technique has been extensively used. In this approach, the intermediate results were always in a redundant form of two numbers. Two types of arrays were introduced for the addition of the intermediate results. In the first type, the arrays were iterative with regular interconnection structure that permits multiplication without delay [4, 5]. The second type arrays were used in tree form, permitting higher speed in time, but the irregular form of a tree-array did not permit an efficient VLSI realization [6]. Modern multiplier designs used [4:2] adders to reduce partial products logic delay and regularize the layout. To improve the regularity, the regular structured tree with recurring blocks and rectangular-styled tree was proposed at the expense of more complex interconnects [7, 8]. In Ref. [9], three dimensional minimization algorithm was developed to design the adder of the maximal possible size with optimized signal connections, which further shortened the path by 1~ 2 XOR delays. However, the resulting structure was more complex than [4:2] adder based tree.

* Corresponding author : yaseer.durrani@uettaxila.edu.pk

Several optimization techniques have achieved power reduction at all abstraction levels [10-13]. The techniques at the lowest technology level and the highest architecture/system level were generally more efficient than techniques at middle levels. In low power, technology-level optimization affects three important factors such as: the load capacitance (C_L), the supply voltage (V_{dd}) and the clock frequency (f_{clk}). All three factors are very effective for RTL optimization.

Recently, we have presented power macromodels for intellectual property (IP) macro-blocks and the IP-based digital systems [14-16]. In this paper, we continue our previous research developing a feasible method of pipelined array multiplier and evaluated the results by the flexible estimation methods at RTL. The proposed multiplier architecture is for low power and high speed applications.

The rest of this paper is organized as follows. In Section 2 we give the background for the array multiplier architecture. In Section 3, we discuss the power macromodeling for the multiplier. Power estimation is evaluated in Section 4. Section 5 summarizes our work.

2. Array Multiplier Methodology

The multiplication process may be viewed to consist of the following two steps: the evaluation of partial products and the accumulation of shifted partial products [12, 13]. Binary multiplication consists of following basic operations:

$$0 \times 0 = 0, \quad 1 \times 1 = 1, \quad 0 \times 1 = 0, \quad 1 \times 0 = 0$$

If we multiply two bits *p* and *q*, then logical AND operation produces the same result as shown in figure 1.

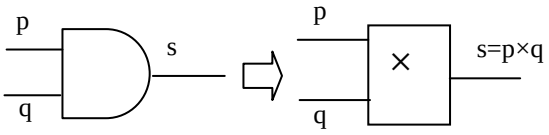

Figure 1. Bit-level multiplier

An array multiplier accepts the multiplier and multiplicand and uses an array of cells to calculate the bit products, $p_j.q_k$ individually in a parallel manner. Figure 2 illustrates a symbol for the high-level view and the multiplication of two 4-bit words. The product bits s_{jk} are generated by combining multiplicand and multiplier bits using an AND gate, $s_{jk} = p_j q_k$, where p_j are bits of the multiplier and q_k are bits of the multiplicand. A row of cells adds 0 to the partial product if the corresponding bit of the multiplier is 0 and adds the multiplicand if it is 1. The structure of the array causes the multiplicand to shift to the left corresponds the weight of the multiplier bit.

For *n*-bit words, each bit q_i multiplies the multiplicand p on a bit-by-bit basis. The product term from least significant bit $q_0$is aligned to the multiplicand, while the next term is shifted one column left. The array builds until every bit of the multiplier is used. The product bits s_i are obtained by summing each of the *i-th* columns, accounting for a carry the *(i-1)-th* the column. A simple expression in "(1)":

$$s_i = \sum_{i=j+k} p_j q_k + r_{i-1} \quad (1)$$

Where $r_{i-1} = 0 \; for \; (i-1) \leq 0.$

For the multiplication of two *n*-bit words, the algorithm for the product can be expressed in "(2)":

$$s_{i+1} = (s_i + pq_i 2^n)2^{-1} \quad (2)$$

With $s_n = s$, the final result such that $s_0 = 0$. The factor $(s_i + pq_i 2^n)$ gives the addition while 2^{-1} accounts for a right shift. The factor of 2^n multiplying *p* is used to compensate for the 2^{-n} introduced by the right shift at the end of the calculation.

An array multiplier accepts the multiplier and multiplicand and uses an array of cells to calculate the bit products $p_j.q_k$ individually in a parallel manner. The bit product $p_j.q_k$ is add it to other contributions in column $i = (j + k)$. This produces the sum for each product bit in "(2)".

An equivalent description of the operation is obtained in base-10 values :

$$P = \sum_{j=0}^{n-1} p_j 2^j \quad and \quad Q = \sum_{k=0}^{n-1} q_k 2^k \quad (3)$$

Then forming the product

$$S = PQ = \left(\sum_{j=0}^{n-1} p_j 2^j\right)\left(\sum_{k=0}^{n-1} q_k 2^k\right) = \sum_{i=0}^{n-1}.\sum_{j=0}^{n-1} p_j q_k 2^{j+k} \quad (4)$$

We see in "(4)" the terms $p_j.q_k$ provide the bit value and $2^{j+k}$the weighting. This scheme calculates the bit products $p_j.q_k$ using AND gates. The product bits are formed using adders in each column. The adders are arranged in a carry-save chain by noting the carry-out bits are fed to the

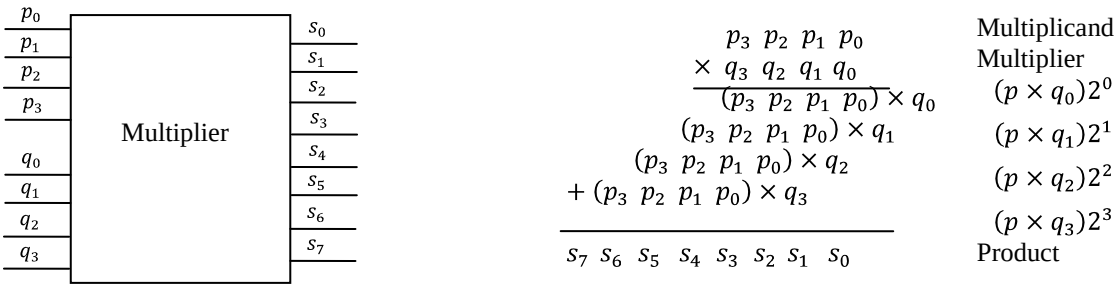

Figure 2. Multiplication of two 4-bit words.

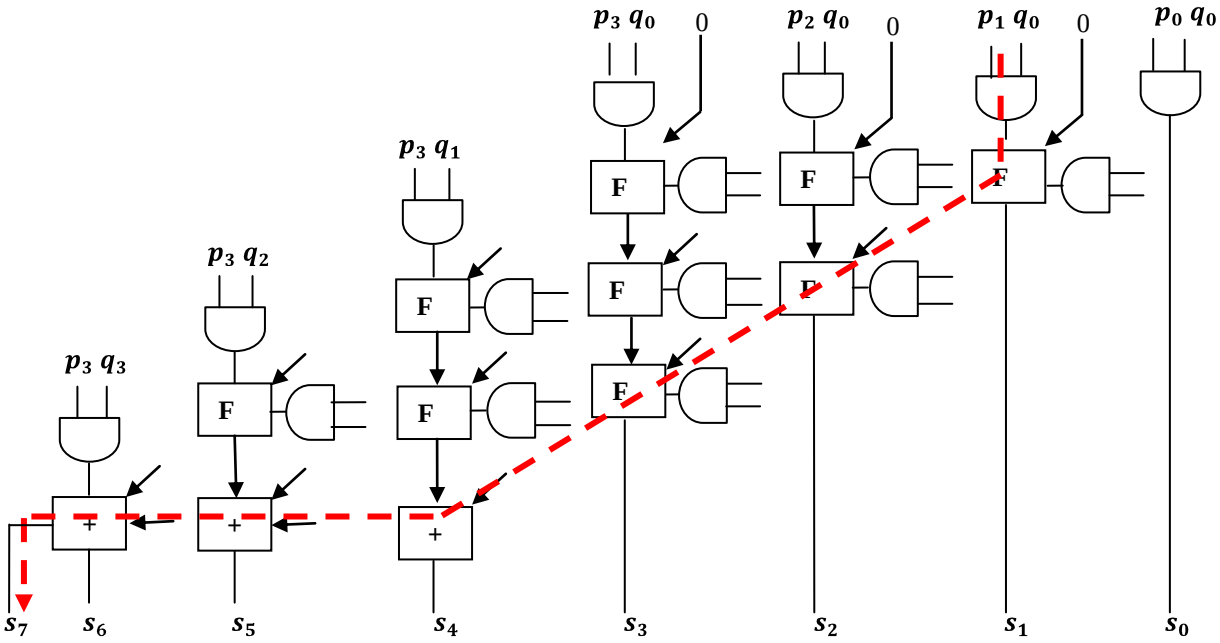

Figure 3. Structure array multiplier showing critical path.

next available adder in the column. The array multiplier accepts all the input bits simultaneously. The longest delay in the calculation of the product bits depends on the speed of the adders. The carry-chain in s_7 that originates from the carry bits from the s_1 column and propagates through the $s_2 - s_6$ quantities.

The speed of the multiplier is determined by both architecture and circuit level. The speed can be expressed by the number of the cell delays along the *critical path* on the architecture level of the multiplier. The cell delay, which is normally a delay of an adder, is determined by the design of the circuit of the cell. In terms of power consumption, the array multiplier is more efficient than other types. Basically the array multiplier originates from the multiplication parallelogram. As shown in Figure 3, each stage of the parallel adders should receive some partial product inputs. The carry-out is propagating into the next row. The bold line is the critical path of the multiplier. In a non-pipelined array multiplier, all of the partial products are generated at the same time. For *n*x*n*-bit array multiplier the critical path consists of two parts: vertical and horizontal. Both have the same delay in terms of full adder (FA) and the gate delays. For an *n*-bit array multiplier, the vertical and horizontal delays are both the same as the delay of an *n*-bit full adder. The FA circuit produces the two-bit sum of three one-bit binary numbers. Several of the FA can be reduced to Half Adders (HA). The FA is the most critical circuit in the multiplier, as it ultimately determines the speed and the power dissipation of the array.

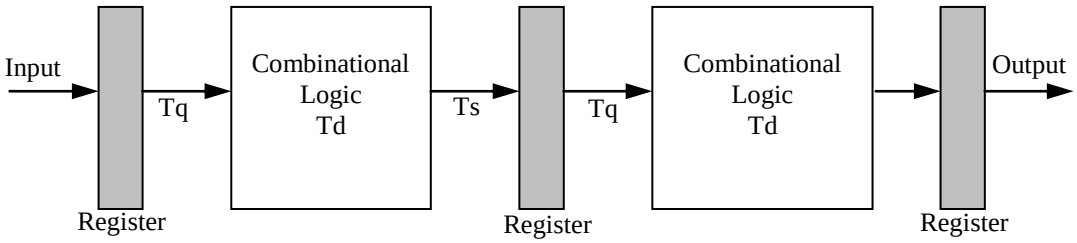

Figure 4. A synchronous pipelined model.

In our experiments, instead of making one straight forward *4*×*4*-bit multiplication having eight bit result, it can carry out the multiplication in two steps: First is the four *2*×*2*-bit multiplications carried out, by creating four partial *4*-bit products. These partial products are added together to create the final *8*-bit product. In our design there are only three different type of non-complex blocks are needed to build up the entire multiplier. These blocks are Carry Propagate Adder (CPA), Carry Save Adder (CSA) and Multiplexer (MUX) as shown in figure 6-a. The four different partial products are created with the four multipliers with one CPA and four MUX blocks. Each MUX block has two bit input data while the CPA takes four bit input data. All four inputs of the multiplexers are four bit wide. The addition and the carry shift are performed with two CSA and one CPA blocks, which permits the addition of the outputs of the right boundary adders. Each CPA and CSA block has six FA. The proposed architecture has the advantage of low power consumption and the high operating speed. Moreover, it occupies small area due to the less number of transistors. This architecture is achieved at the circuit level by minimizing the number of internal node capacitances and reducing the switching activity in the circuit.

3. Pipelined Array Multiplier

Pipelined multipliers are effective in systems where arithmetic throughput is more important than the latency. A linear pipeline processor is a cascade of processing stages which are linearly connected to perform a fixed function over a stream of data flowing from one end to the other. Our basic model of the synchronous pipelining is illustrated in Figure 4. The clocked registers are used to interface between different stages. Upon the arrival of a clock pulse, all registers data transfers simultaneously to the next stage. Successive tasks or operations are initiated one clock per cycle to enter the pipeline. Once the pipeline is filled, one result emerges from the pipeline for an additional cycle. This throughput is sustained only if the successive tasks are independent of each other. The clock cycle τ of a pipeline is determined as: Let τ_i be the time delay of the circuitry in stage S_i, T_s the setup time of the register, and T_q the Clock-to-Q delay. Denote the maximum stage delay as τ_m, then:

$$\tau_m = \max_i(\tau_i)_1^k \tag{5}$$

$$\tau = \tau_m + T_s + T_q \tag{6}$$

The pipeline frequency is defined as the inverse of the clock period:

$$f = \frac{1}{\tau} \tag{7}$$

Ideally, a linear pipeline of *k* stages can process *n* tasks in $k + (n-1)$ clock cycles, where *k* cycles are needed to complete the execution of the very first task and the remaining $n-1$ tasks require $n-1$ cycles. Thus the total time is:

$$T_k = [k + (n-1)]\ \tau \tag{8}$$

where τ is the clock period.

Behaviourally our proposed pipelined array multiplier is represented by a multiplier followed by two stage registers shown in Figure 5a and the summary of the Hardware Descriptive Language (HDL) code is demonstrated in Figure 5c. The multiplier consists of eight breaking up input adders in two sets of the adders with two stage registers explained in Figure 5b. The HDL coding style can be helpful to improve the timings of the data flow. We have implemented the two stage pipelined array multiplier as demonstrated in Figure 6b. Introduction of the registers increases the area of the architecture, when compared to the non-pipelined architectures.

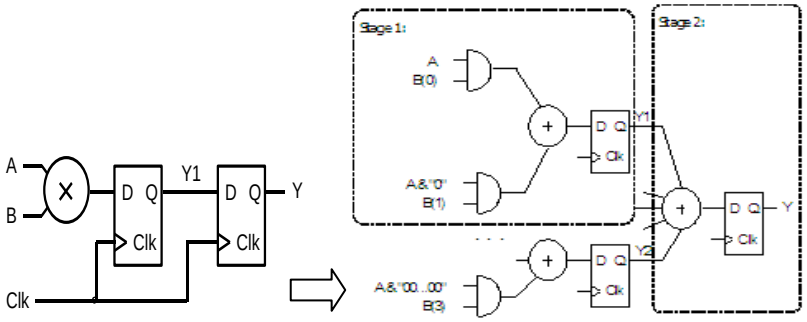

Figure 5. (a) Visualizing pipelining (b) VHDL code for pipelining.

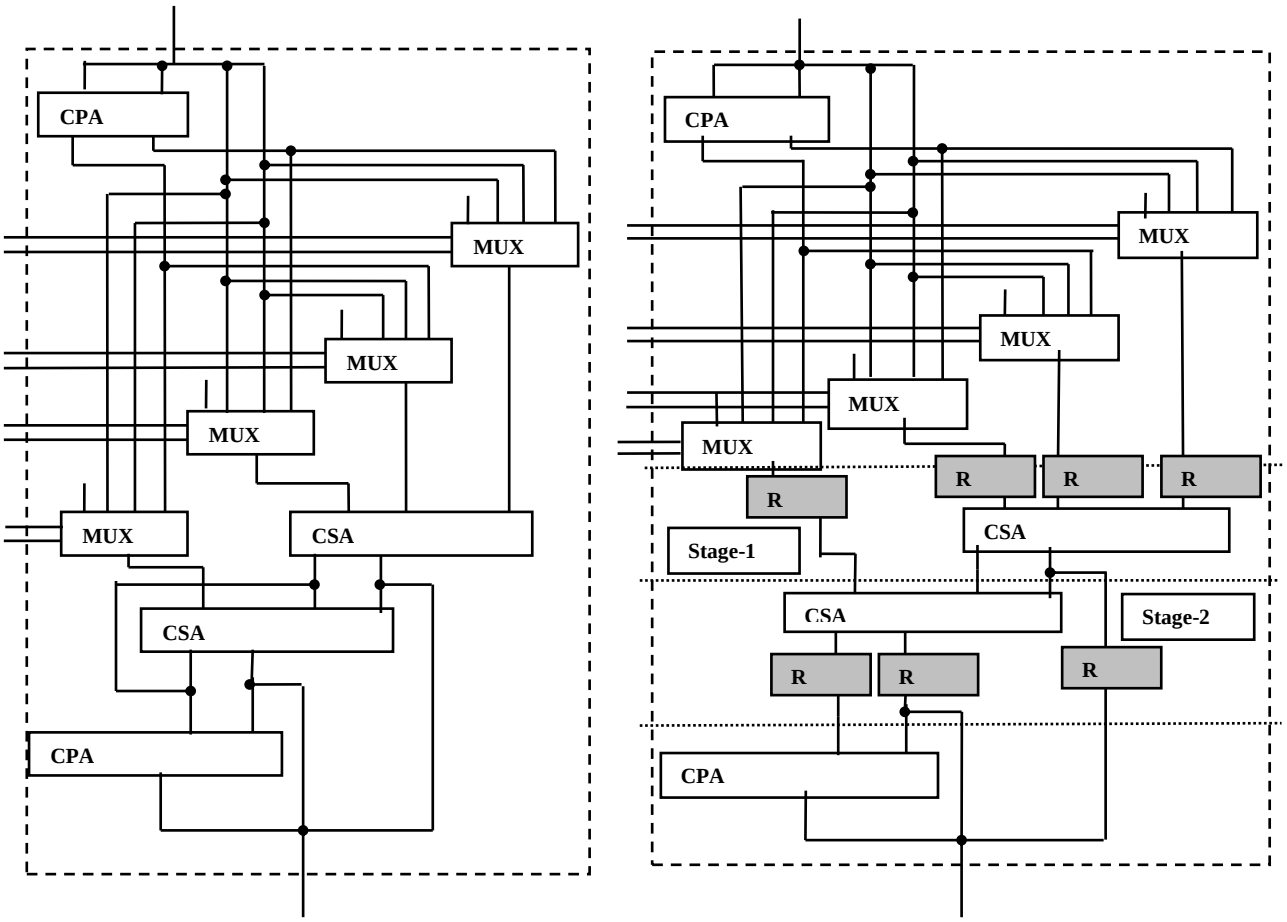

Figure 6. (a) Schematics of array multiplier, (b) Schematics of Pipelined array multiplier.

4. Power Macromodeling for Array Multiplier

Several approaches have been proposed to construct power macro-models using International Symposium on Circuit and Systems (ISCAS-85) benchmark circuits [11-13]. We have observed that the same methodology works as well for different blocks of array multipliers in terms of the statistical knowledge of their primary I/O.

The power estimation procedure is illustrated in Figure 7. In the high-level flow, the first step is the logic synthesis of the parameterized and structural HDL description of the arithmetic modules. For power characterization, we considered only the

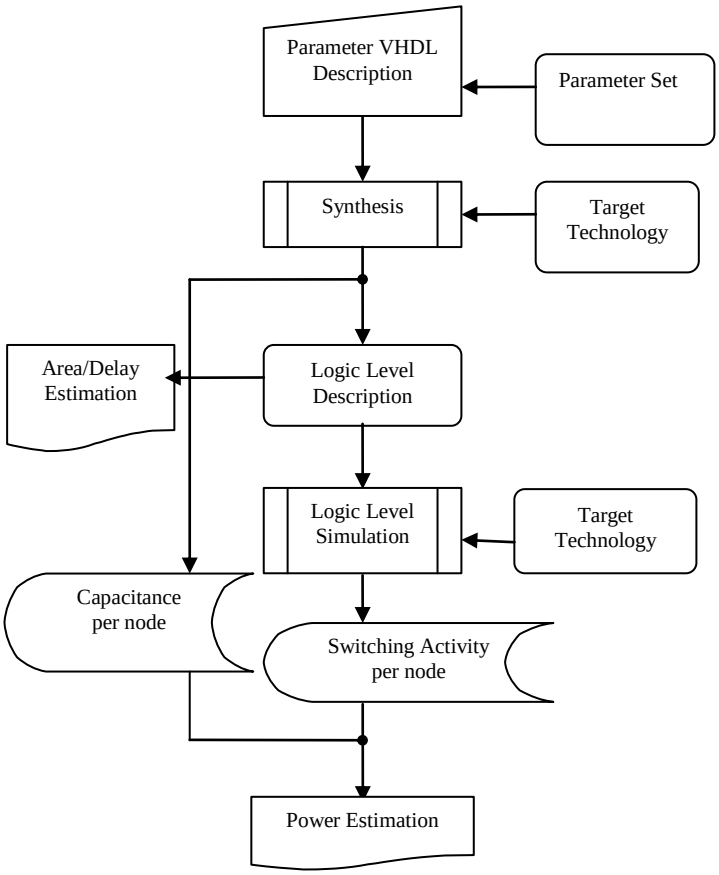

Figure 7. High-level characterization flow.

dynamic power dissipation, which forms the dominant component of the total power. In the second step, the switching activity per node via logic-level simulation has been taken place to compute the power for a certain input vector such as:

$$Power = \sum_{i=1}^{N} C_{load_i} V_{dd}^2 f E_i \quad (9)$$

Where C_{load} is the capacitance at node *i*, V_{dd} is the power supply voltage, *f* is the frequency and is the activity factor at node i. The term E_i of "(9)" is the actually the number of transitions from logic '1' to logic '0' per time unit for the node *i* , which is equal to the ratio of number of node transitions from logic '1' to logic '0', divided by the total number of input vectors:

$$f.E_i = f_{1\to 0} = \frac{\#trans_{1\to 0_i}}{\#vectors} \quad (10)$$

From "(9)" and "(10)" the power is:

$$Power = \frac{V_{dd}^2}{\#vectors} \sum_{i=1}^{N} C_{load_i} \#trans_{1\to 0_i} \quad (11)$$

The array multiplier is simulated under different input sample streams with the signal probability$P(x)$. Signal probability is used for accurate estimation of signal activity. Therefore it is essential to accurately calculate signal probability for further use in estimating activity.

The *P(x)* in "(12)" of a node *x* in the circuit corresponds to the average fraction of clock cycles with a period of *T* in which the node has a steady state logic value of ONE.

$$P(x) = \frac{\sum_{i=1}^{r} \sum_{j=1}^{s} q_{ij}}{r \times s} \quad (12)$$

Given with the number of primary inputs *r* and the input binary stream

$$\mathrm{q} = \left\{ \begin{matrix} (\mathrm{q}_{11}, \mathrm{q}_{12}, \dots, \mathrm{q}_{1r}), (\mathrm{q}_{21}, \mathrm{q}_{22}, \dots, \mathrm{q}_{2r}), \\ \dots, (\mathrm{q}_{s1}, \mathrm{q}_{s2}, \dots, \mathrm{q}_{sr}) \end{matrix} \right\}$$

Table 1. Estimated area for the 4x4-bit multiplier.

Multiplier	Area
4x4-Bit Multiplier	77494
4x4-Bit Pipelined Multiplier Stage-(1)	89434
4x4-Bit Pipelined Multiplier Stage-(2)	104166

Table 2. Power optimization of the *4x4*-bit multiplier using two stage pipelining.

	Non-Pipelined Multiplier	Pipelined Multiplier			
Input Signal Probability	Power (mW)	Stage-(1) Power (mW)	Stage-(1) Power Optimization in %	Stage-(2) Power (mW)	Stage-(2) Power Optimization in %
0.125	0.7490	0.43424	72.48	0.4480	67.18
0.250	1.947	1.1555	68.49	1.2063	61.40
0.375	3.0628	1.7802	72.04	1.864	64.31
0.500	4.1441	2.2708	82.49	2.3812	74.03
0.625	4.8275	2.6061	85.23	2.739	76.25
0.750	4.5799	2.6728	71.35	2.8194	62.44
0.875	3.4788	2.5384	37.046	2.6793	29.83

of length *s*, the multiplier is simulated with eight different signal probabilities with length of 1,000 random vectors. The power measurements were normalized by different operating frequencies.

5. Experimental Results

In this section, we show the results of our power optimization approach. We have implemented a pipelined array multiplier of simple accumulation algorithm by HDL language as shown in Figure 6(a,b). During the characterization phase, the average power consumption measured using power function *f(.)* in "(11)", while least square fitting is used to perform linear regression. The input chosen sequences are highly correlated and they are generated by our new method. The accuracy is tested running gate-level and RTL simulations. The power is estimated using Monte Carlo zero-delay simulation technique. We compare our power macro-modelling results $P_{estimated}$ with Synopsys Power Compiler tool $P_{simulated}$ and compute the average absolute and maximum percentage errors using "(13)".

$$P_{error} = \frac{|P_{simulated} - P_{estimated}|}{P_{simulated}} \times 100\% \quad (13)$$

The given input metrics values are more accurate for the specify range between [0.2, 0.8] and less accurate between [0, 0.2] and [0.8, 1]. The minimum simulations length can be determined through convergence analysis. Converging on the average power figure help us to identify the minimum length necessary for each simulation by considering when the power consumption gets close to a steady value given an arbitrary acceptance threshold. Also the convergent sample size is not a function of circuit size, it depends on how "widely" the power distributes. Regression analysis is performed to fit the model's coefficients. For the non-pipelined and pipelined array multipliers, we measured correlation coefficient 96% and 87%, respectively.

Because of the regular circuit architecture in the pipelined array multiplier, we annotated the activities of the registers to evaluate the activities and power consumption of whole circuits. The multiplier can be divided into two pipeline stages as indicated in Figure 6-b, each of the stage contains the critical path of adders. Introduction of the registers along the layer of arrays increase the area of both architectures when compared to non-pipelined architectures. Moreover, the pipelined array multiplier presents the highest area value due to the higher complexity presented by the modules which process the product terms from Table 1. We found that a non-pipelined architecture consumes more power than pipelined multiplier. In our pipelined approach, glitches were reduced scientifically, and this reduction put greater impact on the power dissipation of the multiplier. However, the less logic depth and delay values presented by our architecture still make it significantly more efficient, for a sinusoidal signal. For a random pattern at the inputs of the multipliers, where signal correlation was not present, power saving up to 60-85% was achieved in the multiplier as shown in Table 2. Reference values of the power dissipation were obtained using time delays from Synopsys Power Compiler.

6. Conclusions

We have presented a pipelined array architecture multiplier. The structure of this array maintains the same level of regularity as the normal array multiplier. We have presented results that show significant improvement in delay and power. The regularity of our architecture makes it suitable for applying other reducing power techniques. In this work we were able to test the use of pipelining approach in order to reduce the critical path and unnecessary signal transitions that are propagated through the array. As we observed, our multiplier is more efficient due to the logic depth that reduces the glitches along the circuit. For a random pattern at the inputs of the multiplier, where signal correlation is not present, power saving up to 60-85% was achieved in the multiplier.

References

[1] Y. Oowaki et al., IEEE J. Solid-State Circuits **22**, No. 5 (1987).

[2] R. Sharma et al., IEEE J. Solid-State Circuits **24**, No. 4 (1989).

[3] G. Goto, et. al., IEEE J. Solid-State Circuits **27**, No. 9 (1992).

[4] N. Itoh et. al., IEEE J. Solid-State Circuits **36**, No. 2 (2001) 249.

[5] V.G. Oklobdzija, D. Villeger and S.S. Lui, IEEE Trans. Compt. **45**, No. 3 (1996) 249.

[6] Dumitru and R. Nouta, VHDL model of an array-of-array multiplier implemented in CMOS Sea-of-Gates, IEEE Solid-State Circuits Conference (1995) pp. 358-361.

[7] K.-S. Chong, B.H. Gwee and J.S. Chang, IET Circuit Devices System **1**, No. 2 (2007) 170.

[8] P. Chan-Ho, C. Byung-soo, L. Dong-ik and C. Hon-Yong, Asynchronous Array Multiplier with an Asymmetric Parallel Array Structure, Advanced Research in VLSI (2001) 202.

[9] A. Asati and Chandrashekhar, A High Speed Hierarchical 16x16 Array of Array Multiplier Design, International Conference on Multimedia, Signal Processing and Communication Technologies (2009) pp. 161-164.

[10] V. Tiwari, S. Malik and P. Ashar, CAD of Integrated Circuits and Systems **17**, No. 10 (1998) 1051.

[11] C. Tsai et al., A Low Power-Delay-Product Multiplier with Dynamic Operand Exchange, IEEE Asia-Pacific Conference on ASICs (2000) pp. 501-504.

[12] Huang, et al., IEEE International Symposium on Circuit & Systems (2002) pp. 489-492.

[13] S. Kim and M.C. Papaefthymiou, Reconfigurable Low Energy Multiplier for Multimedia System Design, Proceedings of IEEE Computer Society Workshop on VLSI (2000).

[14] Y. A. Durrani, A. Abril and T. Riesgo, Efficient Power Macromodeling Technique for IP-Based Digital System. Proceedings for IEEE International Symposium on Circuits & Systems (May, 2007) pp.1145-1148.

[15] Y. A. Durrani and T. Riesgo, Journal of Low Power Electronics **3**, No. 3 (2007) 271.

[16] Y. A. Durrani and T. Riesgo, Elsevier Journal of Digital Signal Processing **19**, No. 2 (2009) 213.