BZ-FAD: A Low-Power Low-Area Multiplier based on Shift-and-Add Architecture

M. Mottaghi-Dastjerdi, A. Afzali-Kusha, and M. Pedram

***Abstract*— In this paper, a low-power structure called BZ-FAD (Bypass Zero, Feed A Directly) for shift-and-add multipliers is proposed. The architecture considerably lowers the switching activity of conventional multipliers. The modifications to the multiplier which multiplies *A* by *B* include the removal of the shifting the *B* register, direct feeding of *A* to the adder, bypassing the adder whenever possible, using a ring counter instead of a binary counter and removal of the partial product shift. The architecture makes use of a low-power ring counter proposed in this work. Simulation results for 32-bit radix-2 multipliers show that the BZ-FAD architecture lowers the total switching activity up to 76% and power consumption up to 30% when compared to the conventional architecture. The proposed multiplier can be used for low-power applications where the speed is not a primary design parameter.**

***Index Terms*— Low-power multiplier, Switching activity reduction, Shift-and-add multiplier, Hot-block ring counter, Low-power ring counter.**

I. INTRODUCTION

Multipliers are among the fundamental components of many digital systems and, hence, their power dissipation and speed are of prime concern. For portable applications where the power consumption is the most important parameter, one should reduce the power dissipation as much as possible. One of the best ways to reduce the dynamic power dissipation, henceforth referred to as power dissipation in this paper, is to minimize the total switching activity, i.e., the total number of signal transitions of the system.

Many research efforts have been devoted to reducing the power dissipation of different multipliers (e.g. [1]-[3]). The largest contribution to the total power consumption in a multiplier is due to generation of partial product. Among multipliers, tree multipliers are used in high speed applications such as filters, but these require large area. The carry-select-adder (CSA)-based radix multipliers, which have lower area overhead, employ a greater number of active transistors for the multiplication operation and hence consume more power. Among other multipliers, shift-and-add multipliers have been used in many other applications for their simplicity and relatively small area requirement [4]. Higher-radix multipliers are faster but consume more power since they employ wider registers, and require more silicon area due to their more complex logic.

In this work, we propose modifications to the conventional architecture of the shift-and-add radix-2 multipliers to considerably reduce its energy consumption[1]. The paper is organized as follows: toward a low power shift-and-add multiplier (sec. II), hot block ring counter (sec. III), results and discussion in section IV with section V containing summary.

II. TOWARD A LOW POWER SHIFT-AND-ADD MULTIPLIER

A. *Main Sources of Switching Activity*

The architecture of a conventional shift-and-add multiplier, which multiplies *A* by *B* is shown in Fig 1 [4]. There are six major sources of switching activity in the multiplier. These sources, which are marked with dashed ovals in the figure, are: (a) shifts of the B register, (b) activity in the counter, (c) activity in the adder, (d) switching between '0' and A in the multiplexer, (e) activity in the mux-select controlled by B(0), and (f) shifts of the partial product (*PP*) register. Note that the activity of the adder consists of required transitions (when *B(0)* is nonzero) and unnecessary transitions (when *B(0)* is zero).

By removing or minimizing any of these switching activity sources, one can lower the power consumption. Since some of the nodes have higher capacitance, reducing their switching will lead to more power reduction. As an example, *B(0)* is the selector line of the multiplexer which is connected to *k* gates for a *k*-bit multiplier. If we somehow eliminate this node, a noticeable power saving can be achieved.

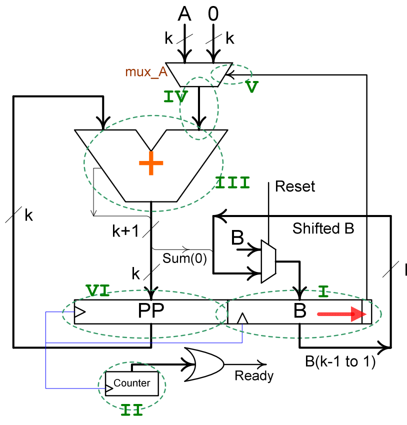


Fig 1. The architecture of the conventional shift-and-add multiplier with major sources of switching activity [4]

Next, we describe how we minimize or possibly eliminate these sources of switching activity.

B. *The Proposed Low Power Multiplier: BZ-FAD*

To derive a low-power architecture, we concentrate our effort on eliminating or reducing the sources of the switching activity discussed in the previous section. The proposed architecture which is shown in Fig 2 is called BZ-FAD.

1) *Shift of the B Register*

In the traditional architecture (see Fig 1), to generate the partial product, *B(0)* is used to decide between *A* and 0. If the bit is '1', *A* should be added to the previous partial product, whereas if it is '0', no addition operation is needed to generate the partial product. Hence, in each cycle, register *B* should be shifted to the right so that its right bit appears at *B(0)*; this operation gives rise to some switching activity. To avoid this, in the proposed architecture (Fig 2) a multiplexer (*M1*) with

[1]Preliminary versions of this were published in *Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design*, Sept. 2006 and *Proceedings of the 18th International Conference on Microelectronics*, Dec. 2006.

one-hot encoded bus selector chooses the hot bit of *B* in each cycle. A ring counter is used to select *B(n)* in the *n*th cycle. As will be seen later, the same counter can be used for block *M2* as well. The ring counter used in the proposed multiplier is noticeably wider (32 bits vs. 5 bits for a 32-bit multiplier) than the binary counter used in the conventional architecture; therefore an ordinary ring counter, if used in BZ-FAD, would raise more transitions than its binary counterpart in the conventional architecture. To minimize the switching activity of the counter, we utilize the low-power ring counter, which is described in the next section.

2) *Reducing Switching Activity of the Adder*

In the conventional multiplier architecture (Fig 1), in each cycle, the current partial product is added to *A* (when *B(0)* is one) or to 0 (when *B(0)* is zero). This leads to unnecessary transitions in the adder when *B(0)* is zero. In these cases, the adder can be bypassed and the partial product should be shifted to the right by one bit. This is what is performed in the proposed architecture which eliminates unnecessary switching activities in the adder. As shown in Fig 2, the *Feeder* and *Bypass* registers are used to bypass the adder in the cycles where *B(n)* is zero. In each cycle, the hot bit of the next cycle (i.e., $B(n+1)$) is checked. If it is 0, i.e., the adder is not needed in the next cycle, the *Bypass* register is clocked to store the current partial product. If $B(n+1)$ is 1, i.e., the adder is really needed in the next cycle, the Feeder register is clocked to store the current partial product which must be fed to the adder in the next cycle. Note that to select between the *Feeder* and *Bypass* registers we have used NAND and NOR gates which are inverting logic, therefore, the inverted clock (*~Clock* in Fig 2) is fed to them. Finally, in each cycle, *B(n)* determines if the partial product should come from the *Bypass* register or from the *Adder* output.

In each cycle, when the hot bit *B(n)* is zero, there is no transition in the adder since its inputs do not change. The reason is that in the previous cycle, the partial product has been stored in the *Bypass* register and the value of the *Feeder* register, which is the input of the adder, remains unchanged. The other input of the adder is *A*, which is constant during the multiplication. This enables us to remove the multiplexer and feed input *A* directly to the adder, resulting in a noticeable power saving. Finally, note that the BZ-FAD architecture does not put any constraint on the adder type. In this work, we have used the ripple carry adder which has the least average transition per addition among the look ahead, carry skip, carry-select, and conditional sum adders [5].

3) *Shift of the PP Register*

In the conventional architecture, the partial product is shifted in each cycle giving rise to transitions. Inspecting the multiplication algorithm reveals that the multiplication may be completed by processing the most significant bits of the partial product, and hence, it is not necessary for the least significant bits of the partial product to be shifted. We take advantage of this observation in the BZ-FAD architecture. Notice that in Fig 2 for P_{Low}, the lower half of the partial product, we use *k* latches (for a *k*-bit multiplier). These latches are indicated by the dotted rectangle *M2* in Fig 2. In the first cycle, the least significant bit, *PP(0)*, of the product becomes finalized and is stored in the rightmost latch of P_{Low}. The ring counter output is used to open (unlatch) the proper latch. This is achieved by connecting the *S/~H* line of the *n*th latch to the *n*th bit of the ring counter which is '1' in the *n*th cycle. In this way, the *n*th latch samples the value of the *n*th bit of the final product (Fig 2). In the subsequent cycles, the next least significant bits are finalized and stored in the proper latches. When the last bit is stored in the leftmost latch, the higher and lower halves of the partial product form the final product result.

Using this method, no shifting of the lower half of the partial product is required. The higher part of the partial product, however, is still shifted. Comparing the two architectures, BZ-FAD saves power for two reasons: first, the lower half of the partial product is not shifted, and second, this half is implemented with latches instead of flip-flops. Note that in the conventional architecture (Fig 1) the data transparency problem of latches prohibits us from using latches instead of flip-flops for forming the lower half of the partial product. This problem does not exist in BZ-FAD since the lower half is not formed by shifting the bits in a shift register.

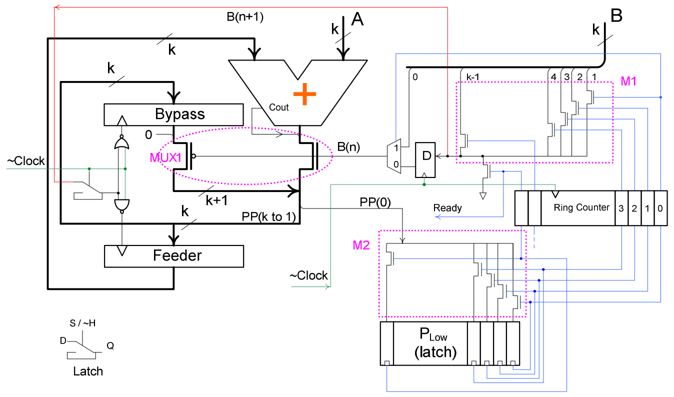


Fig 2. The proposed low power multiplier architecture (BZ-FAD)

In brief, from the six sources of activity in the multiplier, we have eliminated the shift of the B register, reduced the activities of the right input of the adder, and lowered the activities on the multiplexer select line. In addition, we have minimized the activities in the adder, the activities in the counter, and the shifts in the PP (partial product) register. The proposed architecture, however, introduces new sources of activities. These include the activities of a new multiplexer which has the same size as that of the multiplexer of the conventional architecture. Note that the higher part of the partial product in both architectures has the same activity. As will be seen in Section IV the net effect is a lower switching activity for BZ-FAD compared to that of the conventional multiplier.

III. Hot Block Ring Counter

In the proposed multiplier, we make use of a ring counter the architecture of which is described in this section.

In a ring counter always a single '1' is moving from the right to the left. Therefore in each cycle only two flip-flops should be clocked. To reduce the switching activity of the counter, we propose to partition the counter into *b* blocks which are clock-gated with a special multiple-bit clock gating structure shown in Fig 4, whose power and area overheads are independent of the block size. In the proposed counter, called *Hot Block* ring counter (Fig 3) fewer superfluous switching activity exists and there are many flip-flops whose outputs do not go to any clock gating structure. This noticeably reduces the total switching activity of the ring counter.

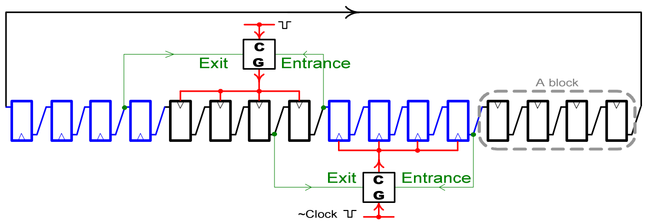


Fig 3- The Hot Block architecture for a 16-bit ring counter - The ring counter is partitioned into *b* blocks of size *s* (*s* is 4 in this figure). Only two clock gators are shown.

We have utilized the property that in each cycle, the outputs of all flip-flops, except for one, are ‘0’. Thus in the partitioned ring counter of Fig 3, there is exactly one block that should be clocked (except for the case that the '1' leaves a block and enters another). We call this block the Hot Block. Therefore, for each block, the clock gating structure (CG) should only know whether the '1' has entered the block (from the right) and has not yet left it (from the left). The CG starts passing the clock pulses to the block once the '1' appears at the input of the first flip-flop of the block. It shuts off the clock pulses after the '1' leaves the leftmost flip-flop of the block.

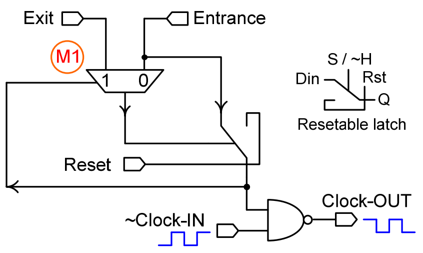


Fig 4. The clock gating structure used in the proposed architecture.

The clock gating structure (CG) proposed for the Hot Block ring counter is shown in Fig 4. It is composed of a multiplexer *M1*, a NAND gate, and a resettable latch. In this work, the multiplexers are implemented with transmission gates. In addition to the *Reset* and *~Clock-IN* signals, there are two other signals called *Entrance* and *Exit*, coming from the neighboring left and right blocks. These are used to determine whether the '1' is present in the block to which the output of the CG goes. When the active high Reset signal is '1', the latch is reset which causes the value of the Entrance signal to be placed on the S/~H line of the latch through multiplexer M1. This in turn causes the latch to read the *Entrance* signal, which was previously reset to '0', since the whole ring counter is reset and all the bits except the first are reset to '0'. After a sufficiently long interval, *Reset* goes to '0' and since *Entrance* has a value of '0', the latch keeps holding ‘0’ on its output, forcing *Clock-OUT* to '1' after the CG is reset. This condition should persist until the '1' is about to enter to the block.

Multiplexer *M1* plays the watchdog role. After the CG is reset, the selector line of multiplexer *M1*, has the value of ‘0’ which causes the *Entrance* signal to be selected (watch dogged) by this multiplexer. The output of the latch is also connected to the NAND gate which causes the input clock signal to be shut off (gated), after the CG is reset. The *Entrance* and *Exit* signals have special meanings as follows. When ‘1’, *Entrance* means that the '1' is about to enter the block in the next cycle. This line is connected to the block input, namely, the input of the rightmost flip-flop in the block, as shown in Fig 3. The *Exit* signal on the other hand indicates the '1' has left the block and hence it should no longer be clocked. Notice that the *Exit* signal is connected to the output of the rightmost flip-flop of the left hand block (Fig 3).

Once the *Entrance* signal becomes '1', the sample and data-in lines of the latch are set to '1'. This causes multiplexer *M1* to select (watch dog) the *Exit* signal which is '0', since all cells of the ring counter except one, have the value of '0' in them. Through multiplexer *M1*, the value of the Exit signal ('0') goes to the *S/~H* line of the latch, which in turn causes the latch to hold '1' (the value of the *Entrance* signal) on its output. From this moment on, the *Exit* signal is watch dogged by multiplexer *M1*; in addition, clock pulses are no longer gated by the NAND gate. To reduce the layout area, we have used a NAND gate instead of an AND gate, and thus, the input clock signal to the clock gator should be the inverted clock (*~Clock-IN* in Fig 4). In the cycles when the *Entrance* signal becomes '1', no positive clock edges should appear at the output of the clock gator; instead it should only prepare to pass clock pulses during the next clock cycles. This is achieved by using the inverted clock signal; the flip-flops are positive-edge triggered, and hence, when the *Entrance* signal, which is the output of some flip-flop, becomes '1' at a positive clock edge, the *~Clock-IN* (see Fig 4) is '0' (negative edge), meaning that no extra positive edge is produced at the clock gator output.

The clock pulses come to the clock gating structure, propagate through the NAND gate, and go to the block cells via *Clock-OUT*, until the *Exit* signal becomes '1'. Then the *S/~H* line of the latch becomes '1' through multiplexer *M1* causing the latch to read its input (the *Entrance* signal), which is ‘0’ at this time. The ‘0’ propagates through the latch and reaches the selector line of multiplexer *M1* giving rise to the *Entrance* signal to be watch dogged again. The output of the latch, which is ‘0’ in this state, also forces the NAND gate to shut off the input clock pulses. Note that regardless of the block size, the proposed CG (Fig 4) has a total of 4 inputs.

IV. Results and Discussion

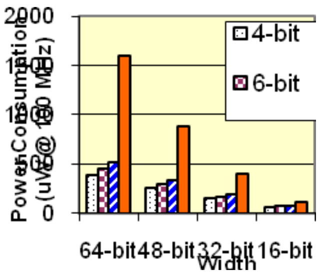
In this section, we present experimental results for the proposed ring counter and multiplier. We used Synopsys Design Compiler for the synthesis and Synopsys Prime Power for the power simulation with the TSMC 0.13µm CMOS technology. Since we have used a standard cell library for this technology, all pass-transistors have been replaced with buffers during the synthesis of the implementations.

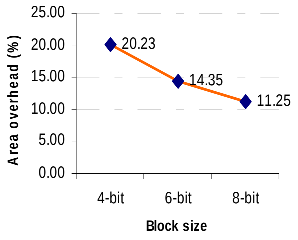
A. Ring Counter

In Fig 5-a, you can see the power consumption of the conventional and Hot Block (Fig 3) ring counters of 16-, 32-, 48- and 64- bits. As seen in this diagram, the efficiency of the Hot Block architecture is more pronounced as the width of the ring counter increases; e.g. with the width of 64 bits, the conventional and Hot-Block consume 1591µW and 389µW respectively. The maximum power reduction is achieved for a 64-bit ring counter with blocks of 4 flip-flops, where a power reduction of 75% is achieved.

Now, we estimate the area overhead of the proposed ring counter. Note that the hot block clock gating structure (Fig 4) can be implemented using 18 transistors, which include 10 for the resettable latch, 4 for the multiplexer, and 4 for the NAND gate. As mentioned earlier, the multiplexers were implemented with transmission gates. Each flip-flop needs 18 transistors and hence for a block size of *f* (*f* flip-flops) the area overhead of the hot block clock gating structure, in terms of the number of transistors is

$$AreaOverhead = \frac{18}{18 + f \times 18} \qquad (2)$$





(a) (b)

Fig 5. Power consumption of the conventional ring counter versus that of the Hot-Block ring counter with different block sizes (a) - The area overhead for different block sizes (b)

The area overhead (Fig 5-b) is dependent on the block size such that as the block size increases the area overhead decreases. However, the larger the block size is, the higher the power consumption is.

The critical path of the Hot Block architecture is the same as that of the conventional architecture except that the clock signal in the Hot Block passes through a NAND gate (Fig 4). Therefore, the edge difference between the *~Clock-IN* and *Clock-OUT*, which is independent of the counter width, is equal to the delay of a NAND gate. This technology dependent delay is very short (e.g. about 10ps for the TSMC 0.13µm CMOS).

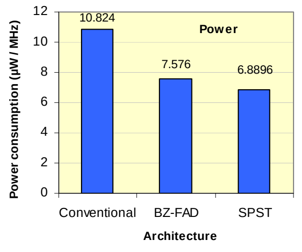
B. Multiplier

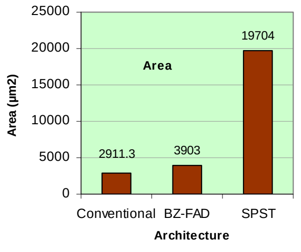
To evaluate the efficiency of the proposed architecture, we implemented three different Radix-2 16-bit multipliers corresponding to the conventional, BZ-FAD and SPST architectures. The SPST (Spurious Power Suppression Technique) architecture is a very low-power tree-based array multiplier published recently in the literature [6]. In general, array multipliers offer high speed and low power consumption. However they occupy a lot of silicon area. The SPST results presented in this paper are based on our implementation of this multiplier and are in agreement with the authors' published results. For SPST implementation we used the circuit details of [6], [7] and [8].

To determine the effectiveness of the power reduction techniques discussed in section II, we have reported in Table I the switching activities of major common blocks of the BZ-FAD and conventional multipliers. As an example, the adder in BZ-FAD has 38.16% less switching activity compared to that of the conventional architecture. The higher switching activity of the BZ-FAD multiplexer is due to its higher fan-out.

TABLE I. COMPARISON OF THE TRANSITION COUNTS OF THE BZ-FAD AND CONVENTIONAL MULTIPLIERS FOR COMMON COMPONENTS (WHEN APPLYING A SUBSET WITH 100 OPERAND PAIRS)

Component	BZ-FAD	Conventional	Reduction (%)
Low order partial product	6,564 (latch)	82,208 (Register *B*)	92.02 %
Adder	46,301	74,870	38.16 %
Multiplexer	56,722	10,013 (mux_A)	-82.35 %
Counter	20,965	22,937	8.60 %





(a) (b)

Fig 6. Comparison of the multipliers in terms of (a) power consumption (b) area.

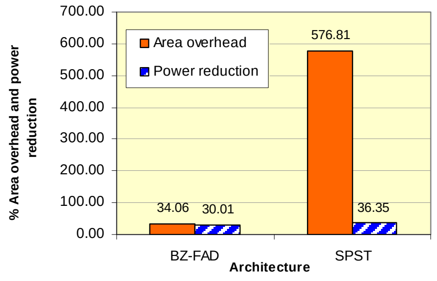


Fig 7. Power reduction and area overhead of BZ-FAD and SPST [6] in comparison with the conventional shift-and-add multiplier.

As another comparison, the power consumption of the multipliers for normally distributed input data are reported in Fig 6 and Fig 7. As seen in Fig 6, the BZ-FAD multiplier consumes 30% lower power compared to the conventional multiplier. The BZ-FAD multiplier has about 34% area overhead. In Fig 7, the area overhead and the power reduction of the BZ-FAD multiplier are compared with those of the SPST multiplier. As seen in the figure, the power saving of SPST is slightly (about 6%) higher than that of BZ-FAD while its area is considerably larger (about 5 times). Finally, in Table II, we have compared the BZ-FAD multiplier with some other published low-power multipliers (excerpted from [6]). The results reveal that the BZ-FAD multiplier may be considered as a very low-power, yet highly area efficient multiplier.

In terms of the area, the proposed technique has some area overhead compared to the conventional shift-and-add multiplier. Comparison between Fig 1 and Fig 2 reveals that M1, M2 and the ring counter are responsible for additional area in the proposed architecture. The area overheads of the ring counter and multiplexers *M1* and *M2* scale up linearly with the input data width. This leads to a small increase in the leakage power which, as the results reveal, is less than the overall power reduction. The leakage power of the 16-bit BZ-FAD architecture is about 11% more than that of the conventional architecture but the contribution of the leakage power in these multipliers is less than 3% of the total power for the technology used in this work. Finally, note that since the critical paths for both architectures are the same (see the next subsection), neither of the two architectures has a speed advantage over the other.

TABLE II. COMPARISON OF DIFFERENT MULTIPLIERS FOR NORMALLY DISTRIBUTED RANDOM DATA IN TERMS OF ENERGY PER MULTIPLICATION

Multiplier	Width	Technology	Energy (pJ)	Area
Wang [9]	32-bit	0.35µ	798.6	19,743 (gates)
Chen [10]	16-bit	0.25µ	346	0.337 (mm^2)
Huang [11]	32-bit	0.18µ	406.5	74,598 (tr.)
Lee [12]	4, 8, 16-bit	0.13µ	52	6,388 (gates)
SPST [6]	16-bit	0.13µ	42	11,028 (tr.)
BZ-FAD	16-bit	0.13µ	48.5	3,903 (µm^2)
Conventional	16-bit	0.13µ	69.2	2,911 (µm^2)

C. Critical Path Analysis

By comparing the two architectures, it is observed that in BZ-FAD, the value of *B* does not contribute to the critical path whereas in the conventional multiplier, it should first settle since *B(0)* is required for the multiplexer to select either *A* or zero. In BZ-FAD, '*A*' has already settled and only the output of the *Feeder* register which is the other input to the adder needs to settle. Next, we consider the BZ-FAD *MUX1*, whose select signal does not contribute to the critical path. This is because the adder in the BZ-FAD, in contrast to the conventional architecture, can begin its work independent of multiplexer *MUX1*. In fact, while the adder is busy with performing the addition, there is enough time for the ring counter and multiplexer *M1* to deliver the value of the next hot bit. All delays in this path are shorter than the adder delay and, hence, do not increase the delay of BZ-FAD. The synthesis timing reports estimates the critical path delay for the BZ-FAD and the conventional multipliers to be 9.76ns and 9.74ns, respectively, which agrees with the above discussion. The slight difference between the reported delays originates from the fact that the input clock signal to the *Feeder* and *Bypass* registers pass through a NAND and a NOR gate in the BZ-FAD architecture. For SPST (synthesized in gate level) the critical path is about 25 ns.

V. SUMMARY AND CONCLUSION

In this paper, a low-power architecture for shift-and-add multipliers was proposed. The modifications to the conventional architecture included the removal of the shift of the *B* register (in $A \times B$), direct feeding of *A* to the adder, bypassing the adder whenever possible, use of a ring counter instead of the binary counter, and removal of the partial product shift. The results showed an average power reduction of 30% by the proposed architecture. We also compared our multiplier with SPST [6], a low-power tree-based array multiplier. The comparison showed that the power saving of BZ-FAD was only 6% lower than that of SPST whereas the SPST area was five times higher than that of the BZ-FAD. Thus, for applications where small area and high speed are important concerns, BZ-FAD is an excellent choice.

Additionally we proposed a low-power architecture for ring counters based on partitioning the counter into blocks of flip-flops clack gated with a special clock gating structure the complexity of which was independent of the block sizes. The simulation results showed that in comparison with the conventional architecture, the proposed architecture reduced the power consumption more than 75% for the 64-bit counter.

ACKNOWLEDGEMENT

MMD and AAK acknowledge the financial support by the Iranian National Science Foundation.

REFERENCES

[1] A. Chandrakasan and R. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circ.*, vol. 27, no. 4, April 1992, pp. 473 – 484.

[2] Nan-Ying Shen and Oscal T.-C. Chen, "Low-power multipliers by minimizing switching activities of partial products," in *Proc. IEEE International Symposium on Circuits and Systems*, vol. 4, pp. 93 – 96, May 2002.

[3] O.T. Chen, S. Wang, and Yi-Wen Wu, "Minimization of switching activities of partial products for designing low-power multipliers," *IEEE Transactions on VLSI Systems*, vol. 11, pp. 418 – 433, June 2003.

[4] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, Oxford University Press, 1st edition, 2000.

[5] V.P. Nelson, H.T. Nagle, B.D. Carroll, and J. David Irwin, *Digital Logic Circuit Analysis & Design*, Prentice-Hall, Inc., 1996

[6] Kuan-Hung Chen and Yuan-Sun Chu, "A Low-Power Multiplier With the Spurious Power Suppression Technique," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 7, July 2007, pp. 846-850.

[7] K. H. Chen, K. C. Chao, J. I. Guo, J. S. Wang, and Y. S. Chu, "An efficient spurious power suppression technique (SPST) and its applications on MPEG-4 AVC/H.264 transform coding design," *in Proc. IEEE Int. Symp. Low Power Electron. Des.*, 2005, pp. 155–160.

[8] K. H. Chen, Y. M. Chen, and Y. S. Chu, "A versatile multimedia functional unit design using the spurious power suppression technique," *in Proc. IEEE Asian Solid-State Circuits Conf.*, 2006, pp. 111–114.

[9] J. S. Wang, C. N. Kuo, and T. H. Yang, "Low-power fixed-width array multipliers," *in Proc. IEEE Symp. Low Power Electron. Des.*, 2004, pp. 307–312.

[10] O. Chen, S.Wang, and Y.W. Wu, "Minimization of switching activities of partial products for designing low-power multipliers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 3, pp. 418–433, Jun. 2003.

[11] Z. Huang and M. D. Ercegovac, "High-performance low-power left-toright array multiplier design," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 272–283, Mar. 2005.

[12] H. Lee, "A power-aware scalable pipelined Booth multiplier," *in Proc. IEEE Int. SOC Conf., 2004,* pp. 123–126.