Multiplier Based On Add And Shift Method ByPassing Zero

Shambhavi S [1], K B ShivaKumar[2], M Z Kurian[1], H S Jayaramu[2]
[1]Department of EC, Sri Siddhartha Institute of Technology, Tumkur, Karnataka, India
[2]Department of TC, Sri Siddhartha Institute of Technology, Tumkur, Karnataka, India
E-mail - Shambhavi.nandi@gmail.com, kbsssit@gmail.com

Abstract

In this paper, a low-power structure for shift-and-add multipliers is proposed. The architec-ture considerably lowers the switching activity of conventional multipliers. The modification to the multiplier which multiplies A by B include the removal of the shifting 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 . The proposed multiplier can be used for low-power applications where the speed is not a primary design parameter.

Index Terms: Hot-block ring counter, low-power multiplier, low-power ring counter, shift-and-add multiplier

1. 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.

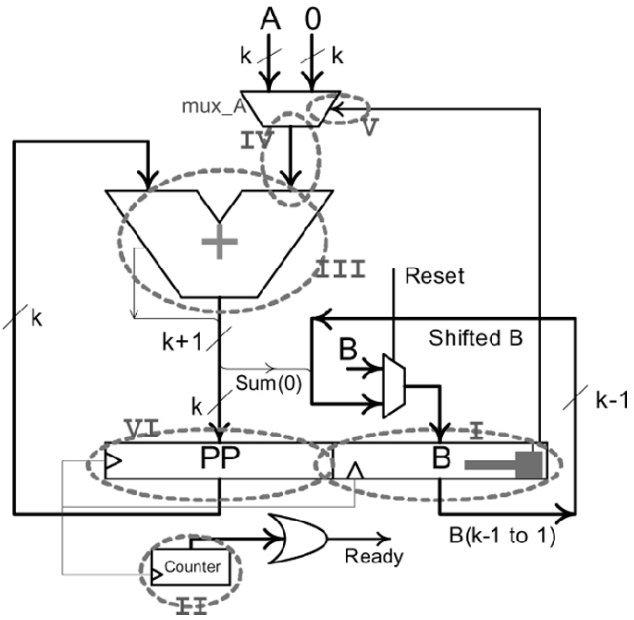


Fig. 1. Architecture of the conventional shift-and-add multiplier with major sources of switching activity

2. 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. 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(n) is nonzero) and unnecessary transitions (when B(n) 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 (n) is the selector line of the multiplexer which is connected to ß gates for a ß-bit multiplier. If we somehow eliminate this node, a noticeable power saving can be achieved. Next, we describe how we minimize or possibly eliminate these sources of switching activity.

B. 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

1) Shift of the B Register: In the traditional architecture (see Fig. 1), to generate the partial product, $_{333}$ is used to decide between ğ and 0. If the bit is "1", ğ 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 should be shifted to the right so that its right bit appears at B(n); this operation gives rise to some switching activity. To avoid this, in the proposed architecture (see Fig. 2) a multiplexer $_{3333}$ with one-hot encoded bus selector chooses the hot bit of in each cycle. A ring counter is used to select b(n) in the nth cycle. As will be seen later, the same counter can be used for block $_{33}$ as well. The ring counter used in the proposed multiplier is noticeably wider (32 bits versus 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 Section II-B2.

2) Reducing Switching Activity of the Adder: In the conventional multiplier architecture (see Fig. 1), in each cycle, the current partial product is added to ğ (when B (n) is one) or to 0 (when B (n) is zero).This leads to unnecessary transitions in the adder when B (n) 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 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 ($_{3}$ 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 ğ, which is constant during the multiplication. This enables us to remove the multiplexer and feed input ğ 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.

3) Shift of the PP Register: In the conventional architecture, the partial product is shifted in each cycle giving rise to transitions. In-specting 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. In Figure 2 for b(n) the lower half of the partial product, we use ß latches (for a ß-bit multiplier). These latches are indicated by the dotted rectangle $_{33}$ in Fig. 2. In the first cycle, the least significant bit *PP (0)* of the product becomes finalized and is stored in the right-most latch of The ring counter output is used to open (unlatch) the proper latch. This is achieved by connecting the 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 (see 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 left-most 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, proposed architecture 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 (see 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. 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 ac-tivities 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 proposed compared to that of the conventional multiplier.

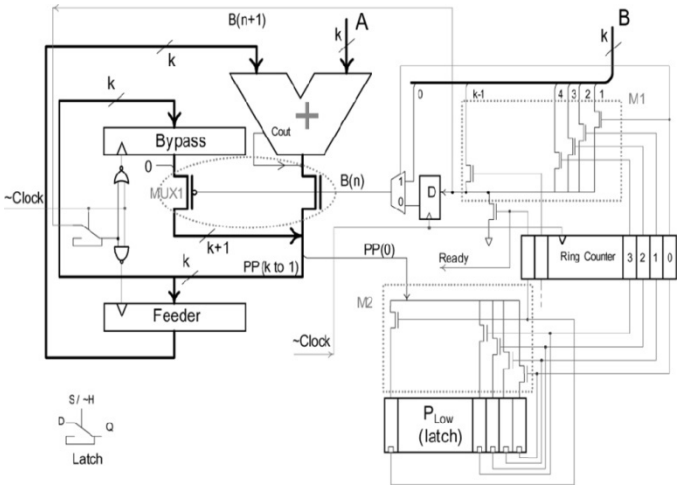
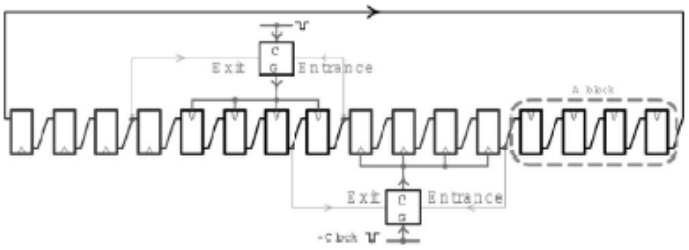


Fig. 2. Proposed low power multiplier architecture .



the right) and has not yet left it (from the left). 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 left-most flip-flop of the block.

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

3. 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 Ĕ 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 (see Fig. 3) fewer superfluous switching activity ex-ists 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. 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

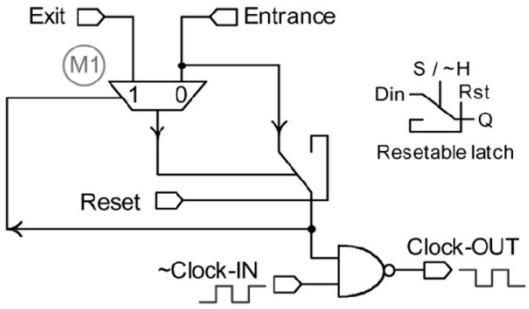


Fig. 4.Clock gating structure used in the proposed architecture.

The clock gating structure (CG) proposed for the Hot Block ring counter is shown in Fig. 4. It is composed of a multiplexer ₃₃, a NAND gate, and a resettable latch. In this work, the multiplexers are implemented with transmission gates. In addition to the *Reset* and clock 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 $_{33\ 3\ 3}$ 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 plays the watchdog role. After the CG is reset, the selector line of multiplexer $_{33}$, 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 right-most 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 right-most flip-flop of the left hand block (see Fig. 3). Once the *Entrance* signal becomes "1", the sample and data-in lines of the latch are set to "1". This causes multiplexer $_{33}$ to select (watch dog) the *Exit* signal which is "0", since all cells of the ring counter ex-cept one, have the value of "0" in them. Through multiplexer $_{33}$, the value of the Exit signal ("0") goes to the $_{33\ 3\ 3}$ 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 $_{33}$; 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 ($_{3\ 3}☼_{3}$ ß 0 $_{3}$β 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 pos-itive-edge triggered, and hence, when the *Entrance* signal, which is the output of some flip-flop, becomes "1" at a positive clock edge, meaning that no extra positive edge is produced at the clock gator output.

Comparison results	Power	Area
Conventional multiplier	151.11 mw	388 of 704 slices 50%
Proposed multiplier	97.85 mw	145 of 704 slices 20%

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 line of the latch becomes "1" through multiplexer $_{33}$ causing the latch to read its input (the *Entrance* signal), which is '0' at this time. The "0" prop-agates through the latch and reaches the selector line of multiplexer $_{33}$ 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 (see Fig. 4) has a total of four inputs.

5. SUMMARY AND CONCLUSION:

The proposed architecture lowers the power dissipation and area when compared to a conventional shift and add multiplier shown in table 6.1 . A multiplexer with one hot encoded bus selector is used for avoiding the switching activity due to the shifting of the multiplier register. Feeder and bypass registers are used for avoiding the unnecessary additions.

The proposed architecture makes use of bit width control logic and a low power ring counter .The design can be verified using Modelsim with verilog code, and power consumption is analyzed using Xilinx software. Proposed architecture can attain 64% power reduction and 30% area saving when compared to the conventional shift and add multipliers.

REFERENCES

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

[2] N.-Y. Shen and O. T.-C. Chen, "Low-power multipliers by minimizing switching activities of partial products," in *Proc. IEEE Int. Symp. Cir-cuits Syst.*, May 2002, vol. 4, pp. 93–96.

[3] B. Parhami, *Computer Arithmetic Algorithms and Hardware Designs*, 1st ed. Oxford, U.K.: Oxford Univ. Press, 2000.

[4] 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.*

[5] *(VLSI) Syst.*, vol. 11, no. 3, pp. 418–433, Jun. 2003.

[6] Z. Huang and M. D. Ercegovac, "High-performance low-power left-toright array multiplier design," *IEEE Trans. Comput.*, vol. 54, no. 2, pp. 272–283, Mar. 2005.

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

[8] V. P. Nelson, H. T. Nagle, B. D. Carroll, and J. I. David, *Digital Logic Circuit Analysis & Design*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[9] M.Mottaghi Dastjerdi ,A.afzali Kusha,m.Pedram "BZFAD A Low Power Low Area Multiplier Based on Shift and Add Architecture " IEEE Trans. Very Large Scale Integr .(VLSI)Syst.,Vol.17, no-2,pp302-306, Feb. 2009