An Area-Efficient Iterative Modified-Booth Multiplier Based on Self-Timed Clocking

Myoung-Cheol Shin, Se-Hyeon Kang, and In-Cheol Park
Department of Electrical Engineering and Computer Science, KAIST, Daejeon, Korea

Abstract
A new iterative multiplier based on a self-timed clocking scheme is presented. To reduce the area required for the multiplier, a small partial CSA array consisting of only two CSA rows is iteratively used to complete a multiplication. The partial CSA array is controlled by a fast internal clock generated using a self-timed technique. Compared with the array implementation, the proposed multiplier yields an 86.6% area reduction for $64 \times 64$ multiplication at the expense of 18.8% slow down. The $32 \times 32$ multiplier implemented in 0.35$\mu$m CMOS technology has a size of $0.495 \times 0.215 \text{mm}^2$, a latency of 19ns at 3.3V.

1. Introduction
Area-efficient and fast multipliers are essential building blocks for high-performance computing and digital signal processing systems. For example, digital communications systems require complex digital filtering that should be processed with many fast and area-efficient multipliers. Therefore, in those applications the multipliers should be small enough so that a large number of them can be integrated on a single chip.

Conventional array multipliers as shown in Fig. 1(a) achieve high performance but require large silicon area, whereas add-and-shift multipliers require less hardware but have low performance. Tree structures [1] achieve even higher performance than conventional arrays. However, they require even more area for wiring and are extremely hard to route. The iterative multipliers in Fig. 1(b) using a partial CSA array are regarded as the most suitable architecture for the area-efficient fast multipliers.

This paper describes a new iterative modified-Booth multiplier that works at a self-timed clock. It uses a very fine-grained iteration, i.e. it contains only two CSA rows, which leads to a large reduction in area. In order to provide the fast clock and control signals required to the fine-grained architecture, previous designs such as Stanford Pipeline Iterative Multiplier (SPIM) used internal clocks generated by inverter chains [2], the length of which should be adjusted manually. We implemented the internal clock using self-timed clocking, which does not require manual clock tuning, minimizes timing waste, and allows the multiplier to be integrated with any system regardless of the system clock.

Fig. 2 shows the clock generator. It contains a dummy CSA cell and a cell that simulates half of the delay of a register, which is shown in Fig. 6(b). These cells are connected like a ring oscillator, where the
inverted output is fed back to the input. The dummy CSA cell is connected so as to yield a maximum input-output delay. Because the generated clock period is twice as long as the loop delay, the clock period is closely matched to the delay of two CSA’s and one register in the iteration path. This clock generator is not guaranteed to run at a particular frequency due to process variation, but exactly as fast as the datapath [5]. If the cells in the datapath slow down due to the process variation, the clock generator cells slow down by the same rate, because they are the identical cells on the same chip.

The self-timed clock of the proposed multiplier works as follows. It remains low while the multiplier is idle. When an external start signal is issued, the enable signal goes high, and the internal clock starts. A one-hot encoded shift register is used to count the clock cycles and generate control signals. When the iterative operation of the CSA’s is completed, the enable signal goes low and the clock stops and stays low until the next start signal arrives. Since the clock of the entire multiplier is cut off, it consumes much less power when it is idle.

2.2 Pipeline Architecture

The pipeline architecture of the n×n-bit proposed multiplier is shown in Fig 3. The datapath consists of two rows of Booth multiplexers and two (n+3)-bit CSA rows, a (n+2)-bit half adder row, a (n+2)-bit CPA, and registers; the control logic block contains two Booth recoders and two WCSA’s, which will be discussed in the next section.

In Fig. 3, at the first pipeline stage, multiplier operand and the shift register is shifted right by four bits, the least significant five bits are recoded in the Booth recoders, and the Booth multiplexers select two partial products. At the second stage, two rows of CSA’s iteratively add two partial products and the two feedback outputs. Two WCSA’s calculate the least significant four bits of the product with the CSA output. At the third stage, a shift register receives the WCSA output and shifts the least significant part of the product to right by four bits. When the iteration finishes, the carry and sum outputs of CSA array are added at the CPA. We get (n+2) most significant bits from the CPA output, 2 bits from the upper WCSA, and (n-4) least significant bits from the shift register to form the 2n-bit product.

3. Design and Implementation Issues

We have solved many problems using several algorithmic and circuit-level techniques. Most of the problems come from the implementation of Booth algorithm. It is clear that we should use modified Booth algorithm in iterative array architecture because it halves the delay of the array [6]. In this section, we also describe the register circuit we used.

3.1 Weighted Carry-Save Adders

Fig. 4(a) shows a conventional 16-bit Booth array multiplier. As a result of Booth recoding, the consecutive partial products are shifted by two bits at each stage, while the consecutive CSA rows are shifted by one bit. The resulting CSA array will have the trapezoidal shape as shown in Fig. 4(a). Because of the bit increasing at each CSA stage, which is shown as black cells in Fig. 4(a), the final CPA should add more bits than the operand size. If we apply this structure to an iterative array, the array will have the shape as shown in Fig. 4(b).
In [3], Park et al. proposed a new array multiplier structure based on a new circuit called a weighted carry-save adder (WCSA). As shown in Fig. 4(c) and (d), this cell prevents the cell increasing in consecutive rows by processing the least significant 2 bits at once. A WCSA cell accepts four inputs, three from the two least significant digits of the previous CSA stage, \(x_{i+1}, y_{i+1}\), and \(x_i\), and one from the previous WCSA cell, \(c_i\). It generates two sum outputs, \(s_{i+1}\) and \(s_i\), and one-carry output, \(c_{i+2}\). The operation of a WCSA can be shown as follows:

\[
\begin{align*}
4 \cdot c_{i+2} + 2 \cdot s_{i+1} + s_i &= 2 \cdot x_{i+1} + 2 \cdot y_{i+1} + x_i + c_i, \\
s_i &= x_i \oplus c_i, \\
s_{i+1} &= x_{i+1} \oplus y_{i+1} \oplus x_i c_i, \\
c_{i+2} &= x_{i+1} y_{i+1} + x_i (x_{i+1} + y_{i+1}).
\end{align*}
\]

As shown in Fig. 4(c) and (d), only the \(c_{i+2}\) output of WCSA’s are connected to the next stage. We can use WCSA’s in the array without any overhead as long as \(c_{i+2}\) is not slower than the slowest output of a normal CSA cell. In addition that the WCSA cell removes the redundant area caused by the additional bits of stages, it improves the overall performance because it reduces the operand size of the final CPA.

We used the transmission gate adders, where \(x_i\) and \(y_i\) are XORed first and then the result is multiplexed with \(c_i\) to obtain the sum and carry output, as shown in Fig. 5(a). As the \(c_i\) input has a path to the outputs than others, a late arrival of the input signal is allowed. Therefore, we designed a WCSA as shown in Fig. 5(b), which is implemented by adding some circuit to the \(c_i\) input of a CSA. In this design, all the outputs of the WCSA are no slower than those of the CSA. The WCSA neither increases critical path delay nor requires any additional pipeline registers.

### 3.2 True Single-Phase Clock Registers

Conventional static CMOS registers (edge-triggered flip-flops) are very slow. If we use them in our design, they will yield a large portion of the delay. We employed a much faster true single-phase clock (TSPC) dynamic CMOS registers [4]. A positive edge-triggered TSPC register proposed originally in [4] is shown in Fig. 6(a).

This register is about twice as fast as and occupies smaller area than a static one. We attached a clamping PMOS to avoid the short circuit current and an inverter to obtain noninverting output to form the register described in Fig. 6(b), which is used in our design.

### 4. Experimental Results

To compare the area efficiency and the performance, three different types of 16×16-, 32×32-, and 64×64-bit multipliers were implemented in the Hynix 0.35μm CMOS technology and simulated. We compared the proposed multiplier with a conventional array multiplier, and a classical add-and-shift multiplier using one CPA. For a fair comparison, we applied the modified Booth recoding algorithm to all of them. We implemented them with the same datapath cells: CSA’s shown in Fig. 5(a), Booth multiplexers in [7], TSPC registers, and a carry select adder as a CPA. As an exception, the CSA used in the array multiplier has output inverter to prevent long successive connections of the transmission gates.

The comparison of the layout area is given in Table I. We measured the area of the surrounding rectangles. The proposed multiplier occupies small area, which is about twice as large as the classical add-and-shift multiplier. As the operand size increases, the size of the proposed multiplier increases linearly, while the array multiplier size is proportional to the square of the operand size. For the 64×64-bit implementations, the proposed one occupies only 13.4% of the area of the array multiplier.

The performance was measured as the worst-case latency, using the post-layout HSPICE simulation with the worst-case input pattern at 3.3V power supply and the junction temperature of 85°C. The results are shown in Table II. The proposed one has the smallest product of area and latency at any case.

### Table I

<table>
<thead>
<tr>
<th>AREA COMPARISON OF THE LAYOUT (unit: mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>16×16</td>
</tr>
<tr>
<td>32×32</td>
</tr>
<tr>
<td>64×64</td>
</tr>
<tr>
<td>TABLE II</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>COMPARISON OF THE LATENCY (unit: ns)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>16x16</td>
</tr>
<tr>
<td>32x32</td>
</tr>
<tr>
<td>64x64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE III</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIP TEST RESULT (unit: ns)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Multiplier 1</td>
</tr>
<tr>
<td>Multiplier 2</td>
</tr>
<tr>
<td>Multiplier 3</td>
</tr>
<tr>
<td>Multiplier 4</td>
</tr>
</tbody>
</table>

Fig. 7. Microphotograph of the 32x32-bit proposed multiplier.

We fabricated the proposed 32x32-bit multipliers in the Hynix 0.35μm CMOS technology at the 12th IDEC MPW, each of which contains 5,400 transistors within a core size of 0.495x0.215mm². A microphotograph of one of the multipliers is shown in Fig. 7.

Four multipliers with four different clock generator circuits were fabricated in one chip. Their clock generators are different in the number of inverter stages in the chain that have four different lengths of clock cycles. Multiplier 2 contains the clock generator shown in Fig. 2. Multiplier 1 contains the fastest clock generator with one less inverter stage than that of Fig. 2, multiplier 3 and 4 have the clock generator with one and two more inverter stages, respectively. The first column of Table III shows the worst-case latencies of four multipliers obtained from HSPICE simulation.

Using an XOR gate, we extracted from each multiplier a test pulse signal pin that has the pulse length equal to the multiplier latency. We tested all of them at 3.3V and measured the worst-case pulse length of the test signal with the test station IMS ATS2 and an oscilloscope. The second column shows the measured worst-case latencies of the fabricated chip. The multipliers with the fastest two generators, including the one presented in Fig. 2, failed to work correctly. The rest of them calculated correct results and showed very similar speed as the HSPICE simulation. Multiplier 3, the fastest working multiplier, has the latency of 19ns at 3.3V, which indicates that it generates the self-timed clock of approximately 650MHz. It seems that the failure of Multiplier 2, the original design that works fine in the HSPICE simulation, is caused by the calculation error of the delays in datapath interconnection. We expect the calculated latency of 14.52ns with 906MHz internal clock in Multiplier 2 can be achieved by a more careful layout.

5. Conclusions

We have proposed a new area-efficient modified Booth iterative multiplier. A clock generated by a self-timed technique makes it efficient to iterate only two rows of carry-save adders (CSA). It requires no external adjustment and is insensitive to process variation. Other techniques such as WCA, fast TSPC registers are used to reduce the area, to improve the performance, and to reduce the overhead of the modified Booth algorithm.

The area and the performance of the proposed multiplier were compared with others. It reduces the area drastically while maintaining the performance. It has the best area-latency product among them. The silicon area reduction becomes more significant as the operand bit size increases. We also fabricated the multiplier on a chip. Although the speed was slower than what is expected, we confirmed that the multiplier operates correctly.

We believe that the proposed iterative multiplier is suitable for use in the large systems that require a lot of small multipliers because it has small area, reasonable performance, and independent self-timed control.

References