Optimization for High Speed Multiplier

Jiaqi Gu1, Yiqian Zhang1

*Abstract*—Multiplication is a critical and unavoidable operation in most of the real-world systems. However, multipliers usually have problems in high latency, low throughput, and high-power consumption. Therefore, we are motivated to propose a new multiplier solution by using pipelined structure, modified Booth encoding, fast sign-extension method, reduced area reduction architecture, and a better final stage adder. We did comparison experiments to illustrate our design’s functionality and effectiveness. Through theoretical analysis and performance estimation, we show that our design has lower latency, higher throughput, and lower energy consumption.

*Index Terms*—Booth encoding, high-speed multiplier, pipeline, reduced area reduction.

# INTRODUCTION

M

ultiplication is of considerable significance in systems with computation demands. However, multipliers usually take many cycles to generate outputs, thus may not be able to deliver data to the next computation stage in a timely manner. In addition, long latency combinational logic leads to relatively low throughput. Latency issue apart, energy consumption per operation is also a concern in modern designs such that the multiplier should be energy efficient, otherwise may induce potentially higher financial cost. This situation inspires us to optimize the multiplier targeting at lower delay, higher throughput, and relatively lower energy consumption. A pipelined high-speed multiplier is thus proposed in this paper.

Booth multiplier [1], is based on an algorithm designed for fast multiplication. It first re-encodes adjacent bits in one of the multiplicands, aimed at reducing the number of partial products. Then all the partial products will be simply calculated and sign-extended in order to achieve multiplication of two’s complement numbers. Finally, weighted summation will be applied to those partial products to generate the result. Therefore, there are several issues for us to address: (1) how to achieve efficient sign extension; (2) how to achieve higher performance in partial product summation; and (3) how to reduce the energy consumption of the hardware design.

To address the above issues, a number of algorithms and methodologies have been proposed in the literature. Particularly, S. Abraham *et al*. [2] proposes a modified Booth multiplier that halves the number of partial products. S. Dubey *et al*. [3] shows a Wallace Tree multiplier where the partial products are computed by Booth multiplier, and this architecture innovation results in significant reduction in delay.

J. Gu is with the Department of Electrical and Computer Engineering, the University of Texas at Austin, TX 78712 USA. (e-mail: jqgu@utexas.edu).

Y. Zhang is with the Department of Electrical and Computer Engineering, the University of Texas at Austin, TX 78712 USA. (e-mail: irenezhang@utexas.edu).

R. D. Kshirsagar *et al*. [4] demonstrates a Wallace-tree-based high throughput multiplier by using a four-stage pipeline.

Inspired by the above work, we propose a high-speed multiplier to achieve lower latency, higher throughput, and lower power consumption. Specifically, we adopt radix-4 modified Booth Encoding to halve the number of partial products a multiplier must add. Then, we adopt the reduced area reduction architecture with full adders and half adders to reduce those partial products to two lines, in order to cut down the use of half adders while keeping the reduction efficiency. Additionally, the final two lines of bits are added by an optimized LING adder, to achieve higher speed compared to several other widely used adders. To improve the throughput of our proposed multiplier, we partition our design into a three-stage pipeline based on its latency analysis. We evaluate the throughput of our design by comparing with the non-pipelined counterpart. We also validate the superiority in performance and energy consumption by comparing our optimized multiplier with other structures, including carry look-ahead adder (CLA) and variable-length carry select adder (CSLA).

# Modified Radix-4 Booth Encoding

When doing multiplication of two N-bit binary two’s complement numbers, there will be N partial products (PP) as well as a lot of sign extension bits. In order to add N lines of partial products, N addition operations are needed, with relatively great computation latency and power dissipation. Booth encoding thus offers a solution to reduce the total number of additions required in the above process. Specifically, as we used in this design, radix-4 Booth Encoding consider three consecutive bits from the least significant bit (LSB) to most significant bit (MSB), with a “0” appended to the LSB. We use {} to represent all bits of the encoded number, and represents the additional “0” in the LSB. Booth algorithm represents three consecutive bits {} with a certain value from set {-2, -1, 0, 1, 2}. The principal in generating Booth Code (BC) is basically giving different weights to {}, as shown below:

(1)

Thus, we have the following BCs for radix 4, as illustrated in Table I. For example, a 16-bit binary 1001101101011101 will be encoded as:

(2)

where represents Booth Encoding operation, and represent -1 and -2 respectively. After Booth Encoding, a two’s complement binary is transformed into two’s complement Booth digits, which have half of the digits as compared to the original number. We denote the multiplicand as A, and multiplier, to which Booth Encoding should be applied, as B. Through this way, the partial products can be generated by multiplexing among pre-calculated 0, -2A, -A, A, and 2A. Detailed methods to implement the pre-calculation will be covered in later sections. Note that the number of partial products for N×N bit multiply is reduced to N/2 accordingly.

TABLE I

Booth codes for radix 4

|  |  |
| --- | --- |
|  | Booth Code |
| 000 | 0 |
| 001 | +1 |
| 010 | +1 |
| 011 | +2 |
| 100 | -2 |
| 101 | -1 |
| 110 | -1 |
| 111 | 0 |

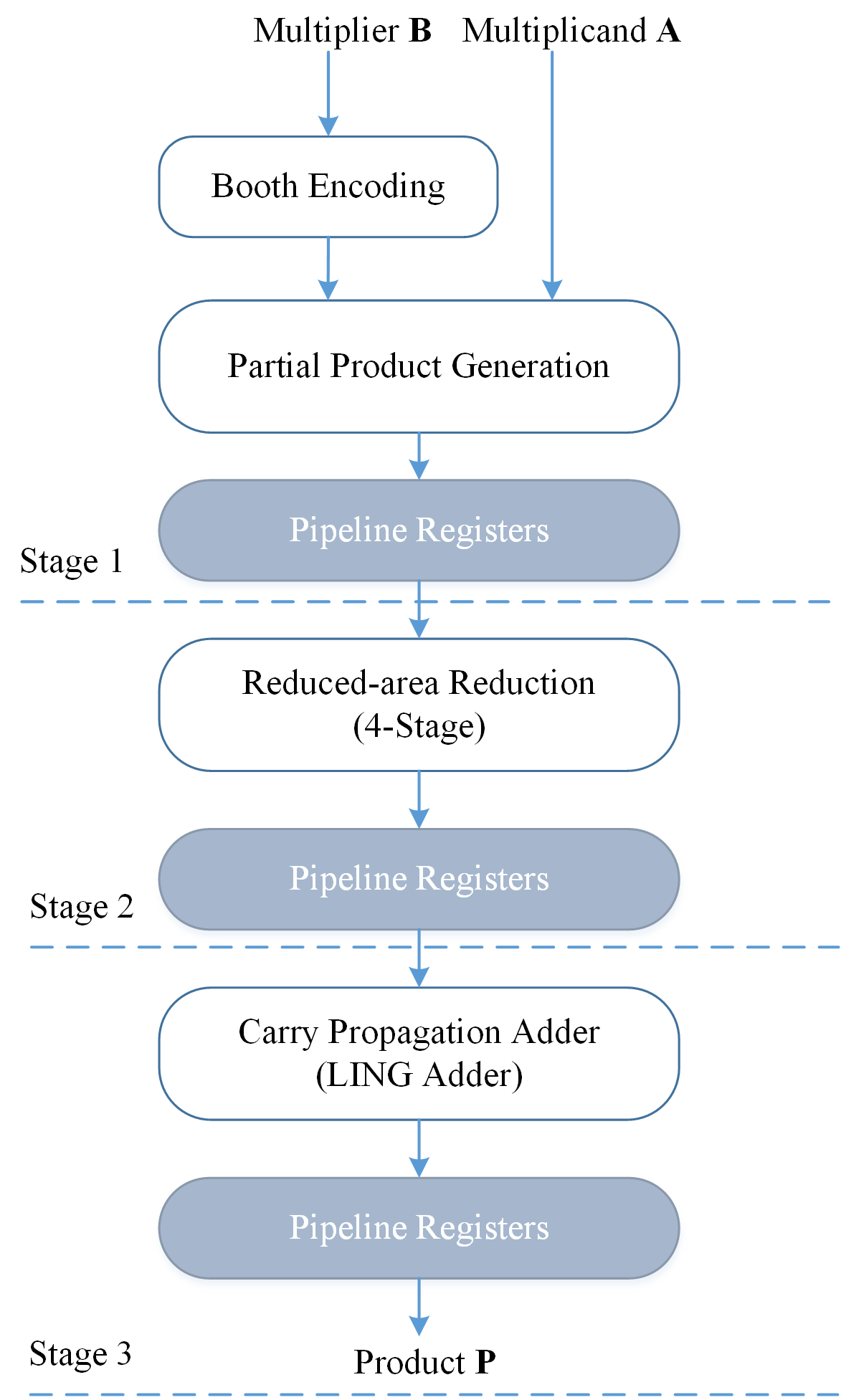


Fig. 1. Framework of our proposed 3-stage pipeline multiplier.

# Reduced-area Reduction

Reduced-area Reduction is the partial product reduction method used in Reduced-area Multiplier [6]. This method only allows the use of half adder when reducing the rightmost column containing exactly two bits or when it is necessary to meet with height limits specified by Dadda series at each stage, which reduces the number of half adder used in each reduction stage and the interconnect overhead as well.

# LING Adder

Similar to a traditional carry lookahead adder, LING adder [5] is an optimized version of prefix adder that relies on fast carry generation to achieve performance. LING adder relies on traditional propagate and generate bits in calculating the carry, but it proposes a novel carry formation that reduces dependency from the previous bit addition to achieve fast sum calculation.

# Theoretical Analysis

The complete design of our proposed three-stage multiplier is shown as Fig. 1. There are mainly three stages in its architecture: (1) Booth encoding and partial product generation; (2) reduced-area reduction; and (3) carry propagation adder. Each of them will be further illustrated in the following sections.

## Booth Encoding and Partial Product Generation

As aforementioned, Booth Encoding operation converts every three consecutive digits to 3-bit sign/magnitude Booth Codes. In our implementation, the gate count for each converting is 12. Therefore, assuming the multiplier A and multiplicand B are both N-bit, with N being an even number, total gate count for Radix-4 Booth Encoding will be:

(3)

If we consider each logic gate cost one gate delay, the total gate delay for Radix-4 Booth Encoding operation is constant as follows:

(4)

Partial product generation for Radix-4 Booth Encoding requires the calculation of +A, -A, +2A, and -2A. Given that the left shift operation and the two’s complement operation extend the number of bits of the partial product and also require sign-extension, we adopted a fast sign-extension method proposed in [7]. This method assumes all the partial products negative and pre-calculates the sign-extended bits. Additionally, this method delays to add 1 when doing two’s complement operation, to accelerate and simplify the process of partial product generation. In our implementation, the partial products are selected from {0, +A, -A, +2A, -2A} by N/2 N-bit wide 5:1 MUX. In addition, we have to generate the sign correction bits if the negative partial product assumption is wrong for a specific partial product, which costs approximately 8N gates. Note that the pre-calculation of sign-extension bits are hard-coded without any hardware cost. Therefore, the estimated total gate count for the partial product generation module in our design is:

(5)

The estimated gate delay of PP Gen module is also constant:

(6)

## Partial Product Reduction

Reduced-area Reduction scheme reduces the height of the partial product matrix at the same speed of Dadda Reduction method. Specifically, in our case, 8 16-bit wide partial products with additional sign-related bits takes 4 reduction stages. Note the full adder we implemented costs 4 gate delay. Thus, the total gate delay of the reduction process is constant:

(7)

As mentioned in [6], the hardware requirements for the original RA multipliers are full adders and half adders, where S is the number of reduction stages. Since we adopted Booth Encoding and different implementation and the total gate count is determined by the specific reduction scheme, we counted the total number of half adder and full adder we used in our case, and we used 103 full adders and 20 half adders. Given that the full adder we adopted costs 7 gates and half adder costs 2 gates. Thus the total gate count for reduction scheme is:

(8)

## Carry Propagation Adder

According to the result of final stage partial product reduction, we have to use an adder to achieve 28-bit addition. In order to accelerate the final addition process, we adopted an optimized version of carry look-ahead adder, called LING adder. LING adder relies on bit propagate, bit generate, and half sum bit. Consider adding two n-bit numbers: and .The bit propagate and bit generate formulation, and , are equivalent to the propagate signal in the traditional prefix adder.

(9)

(10)

The half sum can be calculated by

(11)

The key advantage of LING adder is its proposal of LING carry , which is given by the following equation

(12)

The sum bit can be derived from

(13)

To better analyze LING adders, let

(14)

Then and can be written as

(15)

where

(16)

The delay of a 28-bit LING adder consists of generating bit propagate/generate, , and . Producing all bit propagate and bit generate requires 1 gate delay. and need additional 1 and 3 gate delay(s), respectively. Each and can be calculated in 2 additional clocks, from to , which means and are both stable after 28 gate delays. According to the equation of , should arrive before . Therefore, by equation (13), will require 2 additional delay to calculate. By summing up all the delays, the total delay of a 28-bit LING adder is

(17)

For an N bit LING adder, the number of total gates can be represented by 6N - 4 gates. Bit propagate and bit generate can be implemented with 2N gates. For an N bit LING adder, it has N – 2 that requires 2 gates each. Each sum bit also needs 2 gates in implementation. As a result, the total gate count for a 28-bit LING adder is

(18)

## Pipelined Structure

In order to make a trade-off between additional hardware overhead and throughput, we divided the multiplier into three stages according to the timing report generated by Xilinx Vivado Design Suite 2018.2. The first stage consists of Booth Encoding module and Partial Product Generation module; the second stage consists of Reduced-area reduction module; and the third stage consists of the carry propagation adder. Given this pipeline strategy, the whole multiplier can be divided into three parts with relatively independent functionalities and similar latency, which can increase the maximum frequency it can work at.

# Experimental Results

To validate the functionality and demonstrate the effectiveness of our optimized multiplier, we conduct the first comparison experiment between pipelined version of our proposed design and its non-pipelined counterparts, to show the actual throughput improvement. And we also conduct a comparison experiment among LING adder, Carry Look-ahead Adder (CLA), and Carry Select Adder (CSLA) with variable-length block, to validate the superiority in delay of our proposed design. Each of those comparison designs will be detailed below.

In order to construct a 28-bit CLA, we first constructed a 16-bit CLA with 4-bit wide look-ahead logic block and then constructed 3 4-bit CLAs with the same block size.

As for 28-bit CSLA, we adopted variable-length blocks to reduce its delay. Specifically, we set the block width as 5, 7, 6, 5, and 5 from most significant bit to least significant bit. Note that we adopted ripple carry adders within each block.

As for 28-bit LING adder, we developed a 4-bit LING adder and constructed a 16-bit LING adder based on the 4-bit adder. To form a 28-bit LING adder, we combined 1 16-bit LING adder with 3 4-bit LING adders.

## Functionality Validation

We designed all hardware by using hardware description language Chisel3 [8], and generated the corresponding Verilog codes by using the compiler it offers. Then we ran the post-implementation functional simulation in Xilinx Vivado. We input randomly generated numbers in the range of [-32768, 32767] to our proposed multiplier, and it successfully passed 65,535 random test cases and several corner test cases, which validated the correct functionality of all our designs.

## Synthesis and Quantitative Evaluation

In order to validate the effectiveness of the proposed multiplier, we synthesized the generated Verilog code with Synopsys Design Vision. The experiment results are generated through automatic place and route with constraints that align with theoretical analysis. The constraints include but are not limited to: zero wire load (no interconnect delay) and no design rule check (to avoid buffer insertion). The standard cell library used for the evaluation is gscl45nm.

For quantitative evaluation, we adopted the following metrics: area, number of cells, power, delay, throughput, and energy consumption, for performance evaluation. Area represents…... Number of cells represents…... In our case, delay is formulated as:

(19)

where represents the delay from register clock port to output port, represents maximum combinational logic delay, and represents setup time for pipeline registers. The above delay metric defines the upper bound of clock period. In other words, lower delay means higher clock frequency the multiplier can work at. Defined as the reciprocal of latency, higher throughput represents more results the multiplier can output given certain amount of time. Since lower power does not necessarily lead to lower energy dissipation given certain amount of multiplication operations, we show energy consumption metric to demonstrate the energy spent on one operation which in our case is one 16-bit multiplication.

Table II shows the experiment result for a pipelined/non-pipelined Reduced-area multiplier with CSLA. Despite the fact that the area of the pipelined multiplier increases by ~40% and power increases by ~108%, its delay has improved by more than 100%. To further improve the design, we proposed to replace the CSLA with CLA or LING adder.

TABLE III

Quantitative Comparison among pipelined reduced-area multiplier with different carry propagation adders: area, cell, power, delay, throughput, and energy consumption

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Area (unit area) | Cell | Power (mW) | Delay (ns) | Throughput (ns-1) | Energy Consumption (pJ/Mul.) |
| CSLA | 6681 | 2150 | 0.6306 | 0.94 | 1.064 | 0.5928 |
| CLA | **6361** | **2035** | 0.6077 | 0.90 | 1.111 | 0.5469 |
| LING | 6396 | 2073 | **0.6076** | **0.64** | **1.563** | **0.3889** |

Bold text indicates the best performance.

TABLE II

Quantitative Comparison between pipelined and non-pipelined reduced-area multiplier: area, cell, power, delay, throughput, and energy consumption

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Area (unit area) | Cell | Power (mW) | Delay (ns) | Throughput (ns-1) | Energy Consumption (pJ/Mul.) |
| Pipelined | 6681 | 2150 | 0.6306 | **0.94** | **1.064** | 0.5928 |
| Non-pipelined | **4784** | **1779** | **0.3020** | 1.92 | 0.521 | **0.5798** |

The area metric is measured in the unit of unit area; The delay metric means the minimum clock period required to get correct output. Specifically, the delay here represents the summation of pipeline register delay, maximum combinational logic delay, and setup time for pipeline registers; Energy consumption represents the energy spent for each 16-bit multiplication. Note that non-pipelined multiplier is considered as a single-stage pipeline with registers on its outputs. Bold text indicates the best performance.

As shown in Table III, multipliers with CLA or LING adder improve their area and power dissipation by ~5% and 3.77% respectively. Furthermore, the performance of the multiplier built with LING adder improves by 46.8%. When comparing the traditional non-pipelined Reduced Area multiplier (using CSLA) with pipelined Reduced Area multiplier (using LING), the later improves its throughput by 300% with 16.5% increase in number of cells and 100% in power.

# Reference

* 1. A. D. Booth, “A Signed Binary Multiplication Technique,” *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. (2), pp. 236-240, Aug. 1951.
  2. S. Abraham et al., “Study of Various High Speed Multipliers,” in *Proc. ICCCI*, Coimbatore, India, 2015.
  3. S. Dubey et al., “A High Speed Wallace Tree Multiplier Using Modified Booth Algorithm for Fast Arithmetic Circuits,” *IOSRJECE*, vol. 3, no. 1, pp. 7-11, Sep. 2012.
  4. R. D. Kshirsagar, E. V. Aishwarya, A. S. Vishwanath and P. Jayakrishnan, “Implementation of pipelined Booth Encoded Wallace tree Multiplier architecture,” in *Proc. ICGCE*, Chennai, 2013, pp. 199-204.
  5. H. Ling, “High-Speed Binary Adder,” *IBM J. of Research and Development*, vol. 25, no. 2 and 3, pp. 156-166, May. 1981.
  6. K. A. C. Bickerstaff, M. Schulte and E. E. Swartzlander, "Reduced area multipliers," Proceedings of International Conference on Application Specific Array Processors (ASAP '93), Venice, Italy, 1993, pp. 478-489.
  7. J. Fadavi-Ardekani, "M\*N Booth encoded multiplier generator using optimized Wallace trees," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 1, no. 2, pp. 120-125, June 1993.
  8. J. Bachrach et al., “Chisel: Constructing hardware in a Scala embedded language,” DAC Design Automation Conference 2012, San Francisco, CA, 2012, pp. 1212-1221.