Design of a Dedicated Reconfigurable Multiplier in an FPGA

Yu Hongmin†, Chen Stanley L, and Liu Zhongli

(*Institute of Semiconductors, Chinese Academy of Sciences, Beijing* 100083, *China*)

Abstract: We design a reconfigurable pipelined multiplier embedded in an FPGA. This design is based on the modified Booth algorithm and performs 18×18 signed or 17×17 unsigned multiplication. We propose a novel method for circuit optimization to reduce the number of partial products. A new layout floorplan design of the multiplier block is reported to comply with the constraints imposed by the tile-based FPGA chip design. The multiplier can be configured as synchronous or asynchronous. Its operation can also be configured as pipelined for high-frequency operation. This design can be easily extended for different input and output bit-widths. We employ a novel carry look-ahead adder circuit to generate the final product. The transmission-gate logic is used for the low-level circuits throughout the entire multiplier for fast logic operations. The design of the multiplier block is based on SMIC 0.13μm CMOS technology using full-custom design methodology. The operation of the 18×18 multiplier takes 4.1ns. The two-stage pipelined operation cycle is 2.5ns. This is 29.1% faster than the commercial multiplier and is 17.5% faster than the multipliers reported in other academic designs. Compared with the distributed LUT-based multiplier, it demonstrates an area efficiency ratio of 33∶1.

Key words: FPGA; multiplier; reconfigurable; modified Booth algorithm; CLA; transmission-gate logic
EEACC: 1265A
CLC number: TN492 **Document code**: A **Article ID**: 0253-4177(2008)11-2218-08

1 Introduction

Nowadays, it has been proven that FPGAs are well suited for use as reconfigurable hardware to accelerate software in many applications[1~7]. Image/video processing tasks are particularly well suited to hardware acceleration because of the inherent parallelism and data flow structure. Most image/video processing tasks are multiplication-intensive.

Conventional FPGA architectures are well suited to binary addition. However, configuring FPGAs for binary multiplication results in inefficient usage of the reconfigurable logic resources. It has been reported that a typical multiplication-intensive application can use over 70% of the FPGA logic resources. Reference [4] shows that the silicon area can be saved by a 100∶1 ratio. The speed advantage can be 10 times that of a LUT-based multiplier implementation. One solution to obtain efficiency in both area and speed is to embed dedicated multipliers into an FPGA design. Much effort has been devoted to the techniques of the design and its implementation[1~11]. Kang *et al*.[3] suggested a high-speed multiplier using an algorithm to achieve fast multiplication by generating fewer partial-product rows in 2's complement representation. Haynes and Cheung[4] suggested a reconfigurable multiplier constructed using an array of 4bit flexible array blocks (FABs). Any $4n\times 4m$ bit signed/unsigned binary multiplication can be represented as a combination of FABs. The multiplier has good flexibility and area utilization when it is used on operands with small bit-widths. However, the speed and area are degraded in larger bit-width multiplication.

All the major commercial FPGA providers have embedded dedicated multiplier blocks in their high-density FPGA offerings. For example, up to 104 dedicated 18×18 multipliers are in Xilinx's Spartan-3 family[1] and up to 150 18×18 multipliers are in Altera's Cyclone II family[2]. All Spartan-3 devices from Xilinx offer embedded 18×18 multipliers[1]. Each embedded multiplier can be configured to support synchronous and asynchronous operations. Each Cyclone II device in Altera's FPGA family has one to three columns of embedded multipliers[2]. Each embedded multiplier can be configured to support one 18×18 or two 9×9 multipliers. This can improve the multiplier usability but has a minor performance penalty.

In this paper, we suggest an embedded dedicated pipelined reconfigurable 18×18 multiplier block for our FPGA. We provide a novel design for a reconfigurable multiplier block (MB) based on a modified Booth algorithm (MBA). The MB accepts two 18bit words as the inputs to produce a 36bit product. The input buses to the MB accept data in 2's complement form (either 18bit signed or 17bit unsigned). The MB can be configured to operate in synchronous

† Corresponding author. Email: hmyu@red.semi.ac.cn

Received 26 May 2008, revised manuscript received 30 June 2008

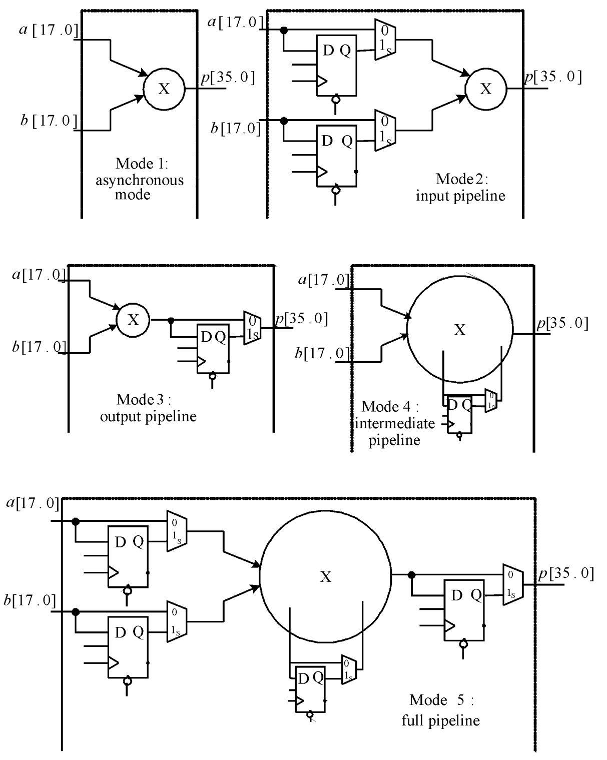
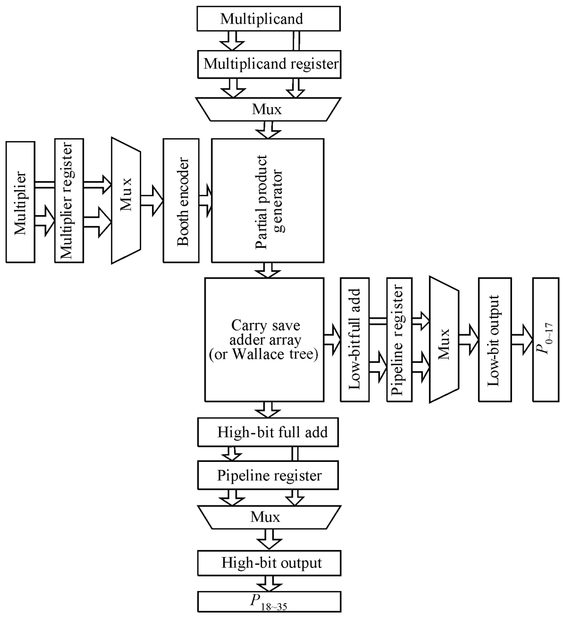


Fig. 1　Five reconfiguration modes in MB

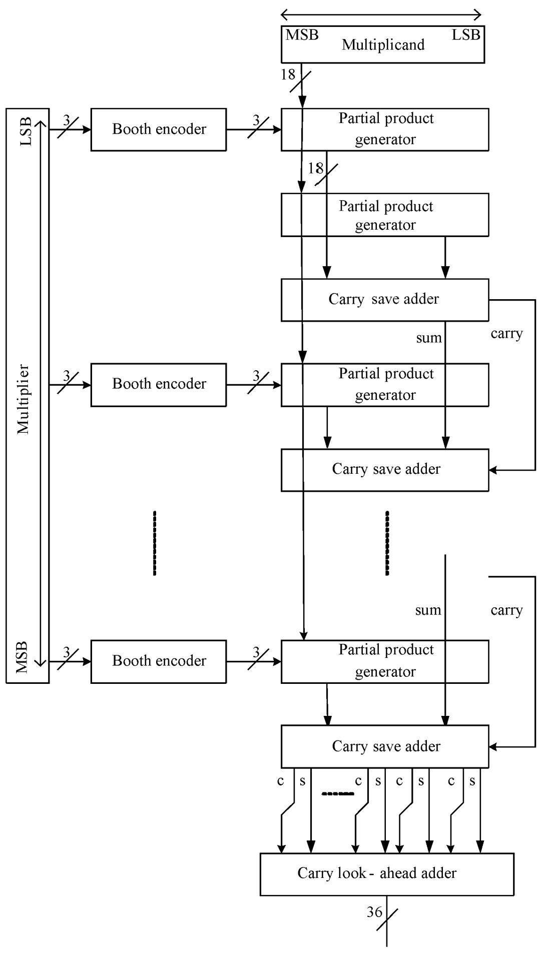
or asynchronous mode, and can also be configured as pipelined. We have tried two different methods to implement the sum of partial products: the CSA (carry save adder) array and the Wallace tree. The results prove that even though these two types of multipliers perform exactly the same in functionality, the regularity of the layout imposes a constraint in the selection of the algorithm. To achieve good extendibility, this constraint favors the CSA approach. For the 18×18 multiplier, 8 stages of partial product sums are chained and followed by the final product calculation. We employ a novel carry look-ahead adder (CLA) circuit to produce the final product. The implementation of the MB is based on SMIC 0.13μm CMOS technology[12].

2　Circuit design

In our methodology for the FPGA chip design, we provide a dedicated 18×18 multiplier. In design of this multiplier, we introduce a new architecture and circuit design to enhance the performance and give a new layout floorplan to improve the area utilization. Both the inputs and outputs of the multiplier are configurable as registered and unregistered. The intermediate result of the partial-product sum can also be registered to divide the multiplication time by 2 to double the throughput. Neither Spartan-3 nor Cyclone II has the option of pipelined operation. As shown in Fig. 1, the multiplier block can be configured as five different modes.



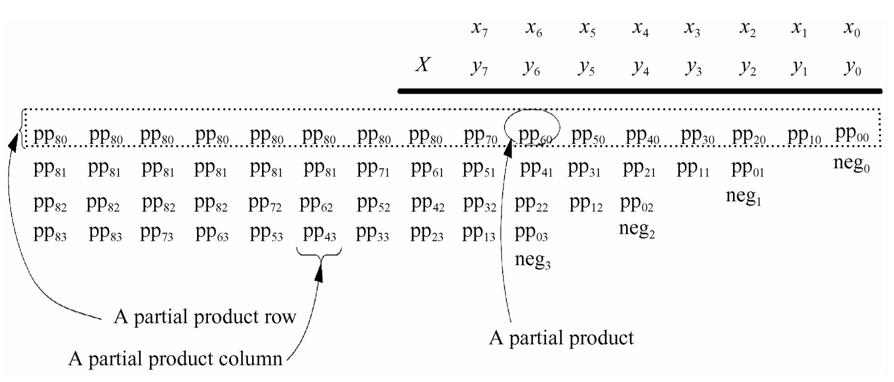
(a)



(b)

Fig. 2　(a) Total block diagram of the MB; (b) Dataflow of the multiplier

As shown in Fig. 2, the multiplication operation is divided into three levels. In level 1, the 9 rows of 18bit



x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
X y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
pp_{80} pp_{80} pp_{80} pp_{80} pp_{80} pp_{80} pp_{80} pp_{80} pp_{70} pp_{60} pp_{50} pp_{40} pp_{30} pp_{20} pp_{10} pp_{00}
pp_{81} pp_{81} pp_{81} pp_{81} pp_{81} pp_{81} pp_{71} pp_{61} pp_{51} pp_{41} pp_{31} pp_{21} pp_{11} pp_{01} neg_0
pp_{82} pp_{82} pp_{82} pp_{82} pp_{72} pp_{62} pp_{52} pp_{42} pp_{32} pp_{22} pp_{12} pp_{02} neg_1
pp_{83} pp_{83} pp_{73} pp_{63} pp_{53} pp_{43} pp_{33} pp_{23} pp_{13} pp_{03} neg_2
neg_3
A partial product row
A partial product column
A partial product

Fig.3 Array of partial products for signed multiplication in Ref.[3]

partial products are generated. In level 2, the partial products are summed to obtain one row of sum bits and one row of carry bits. In level 3, the sum bits and carry bits are added to generate the final result.

The multiplication algorithm is detailed below.

2.1 Level 1: Booth encoder and partial product generator

The modified Booth encoding (MBE) is efficient in reducing the total number of the partial products[13,14].

For $n$bit 2's complement signed numbers X and Y, it can be shown as

$$X = -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + a_{n-3}2^{n-3} + \cdots\cdots + a_1 2^1 + a_0 2^0 = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

$$Y = -b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + b_{n-3}2^{n-3} + \cdots\cdots + b_1 2^1 + b_0 2^0 = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

Here Y can be re-organized as:

$$\begin{aligned} Y &= -b_{n-1}2^{n-2} + b_{n-2}2^{n-2} + b_{n-3}2^{n-3} + \cdots\cdots + b_1 2^1 + b_0 2^0 + b_{n-4}2^{n-3} - b_{n-4}2^{n-4} + \cdots\cdots + b_1 2^2 - b_1 2^1 + b_0 2^1 - b_0 2^0 \\ &= -b_{n-1}2^{n-1} + b_{n-2}2^{n-1} - b_{n-2}2^{n-2} + b_{n-3}2^{n-2} - b_{n-3}2^{n-3} + b_{n-4}2^{n-3} - b_{n-4}2^{n-4} + \cdots\cdots + b_1 2^2 - b_1 2^1 + b_0 2^1 - b_0 2^0 \\ &= (-b_{n-1}2^{n-1} + b_{n-2}2^{n-1}) + (-b_{n-2}2^{n-2} + b_{n-3}2^{n-2}) + (-b_{n-3}2^{n-3} + b_{n-4}2^{n-3}) + (-b_{n-4}2^{n-4} + b_{n-5}2^{n-4}) + \cdots\cdots + (-b_1 2^1 + b_0 2^1) - b_0 2^0 \\ &= [(-b_{n-1}2^{n-1} + b_{n-2}2^{n-1}) + (-b_{n-2}2^{n-2} + b_{n-3}2^{n-2})] + [(-b_{n-3}2^{n-3} + b_{n-4}2^{n-3}) + (-b_{n-4}2^{n-4} + b_{n-5}2^{n-4})] + \cdots\cdots + [(-b_1 2^1 + b_0 2^1) + (-b_0 2^0 + b_{-1}2^0)] \\ &= [(-2b_{n-1}2^{n-2} + 2b_{n-2}2^{n-2}) + (-b_{n-2}2^{n-2} + b_{n-3}2^{n-2})] + [(-b_{n-3}2^{n-4} + 2b_{n-4}2^{n-4}) + (-b_{n-4}2^{n-4} + b_{n-5}2^{n-5})] + \cdots\cdots + [(-b_1 2^1 + b_0 2^1) + (-b_0 2^0 + b_{-1}2^0)] \\ &= \sum_{i=0}^{n/2-1}(b_{2i-1} + b_{2i} - 2b_{2i+1})2^{2i} \end{aligned}$$

Defining $K_i = b_{2i-1} + b_{2i} - 2b_{2i+1}$, where $i = 0,1,2,\cdots,(n/2-1)$; then Y can be:

$$Y = \sum_{i=0}^{n/2-1}(K_i)2^{2i}$$

Thus the product is:

$$XY = X\sum_{i=0}^{n/2-1}(K_i)2^{2i} = \sum_{i=0}^{n/2-1} K_i X 4^i$$, where b_{-1} is $0,1,\cdots,(n/2-1)$.

The number of partial product rows to be accumulated in the multiplication of two $n$bit numbers can be reduced from n to $n/2$. The advantages of the MBE in both speed and area make it popular in various applications[3~7,9~11].

Kang *et al*.[3] give a full account of an 8× 8 multiplier to demonstrate the pros and cons of MBE. As shown in Fig. 3, one additional partial-product row results from the last increment operation (second part of the 2's complement operation) controlled by the neg3 signal. In handling this last increment operation, they used the logarithmic way of finding the 2's complement of the multiplicand. Unlike the other rows, a 2's complement logic followed by a 5-to-1 selector was used in generating the last partial-product row. The need for the last increment operation is eliminated and the area is saved. However, this area saving is traded for additional irregularity in the bit-slice layout structure. We postpone the increment operation needed for partial products to the Carry-Save Adder stage and streamline the layout of the multiplier block. This postponed increment technique saves 1/9 of the CSA area and reduces the delay in the CSA operation by ~10%.

The sign extension needed in the partial-product accumulation is predicted and minimized during the formation of the partial products, generally called sign extension prevention. Our implementation of the sign extension, as opposite to the traditional approach, reduces the CSA adders by half. This further reduces the delay in the CSA operation.

We now detail our novel Booth encoder and partial product generator (PPG) implementation that minimize the multiplication delay and maximize the area savings. In the MBE scheme[14], the multiplier is portioned into three-bit groups that overlap by one

Table 1 Truth table for the Booth encoder

b_{i+1}	b_i	b_{i-1}	Booth encoder	comp	shift	zero
0	0	0	+0x	0	1	1
0	0	1	+1x	0	0	0
0	1	0	+1x	0	0	0
0	1	1	+2x	0	1	0
1	0	0	−2x	1	1	0
1	0	1	−1x	1	0	0
1	1	0	−1x	1	0	0
1	1	1	−0x	1	1	1

bit. The Booth encoder uses each group of three-bits to create three control signals: comp, shift, and zero. As shown in Table 1, the partial product generator uses the three signals to generate the 2's complement partial-product rows by controlling the data operation of the multiplicand.

The Boolean expressions from Table 1 are as follows.

$\text{comp} = b_{i+1}$

$\text{shift} = b_i\, b_{i-1} + (\sim b_i)(\sim b_{i-1})$

$\text{zero} = b_{i+1}\, b_i\, b_{i-1} + (\sim b_{i+1})(\sim b_i)(\sim b_{i-1})$

As follows, we explain the operation of the Booth encoder.

Recoded digit	Partial product operation on multiplicand
0	0
+1	as is
+2	shift left by one
−1	complement
−2	shift left followed by complement

As shown in Fig. 4, the Booth encoder generates three partial-product control signals. The final increment operation controlled by the negation signal in Ref. [3] at the end of the CSA operation is eliminated and thus leads to an array of perfectly parallelogram-shaped partial products. The sign extension to form the partial products uses the sign extension prevention procedure from Agrawal and Rao[15]. Each partial product needs to be sign-extended prior to the CSA operation. As proposed by Agrawal and Rao[15], we use two-bit sign generation to eliminate the need for a full-range sign extension, which requires full-range adders in all stages of the CSA operation and leads to significantly increased delay, power, and area. The sign extension is summarized as follows:

(1) Complement the sign bit.

(2) Pad 1 to the left of the sign bit.

(3) Add 1 to the sign bit of the first partial product.

Steps 1 and 2 are shown in Fig. 4. Step 3 is folded into the CSA as an extra operation on the sign bit. The pad 1 and add 1 operations are carried out in parallel to the CSA operation and do not add additional delay to the multiplication. The speed increase from the sign-bit extension prevention is significant for large multiplicand. The transistor-level implementation of the Booth encoder and PPG uses the transmission-gate logic to gain further performance improvement.

2.2 Level 2: partial products summation

For the partial products summation, most multipliers use ripple carry adder (RCA) arrays, Wallace trees (WT), or carry save adder (CSA) arrays to accumulate the multiple rows of partial products[14]. The RCA array has the best structural regularity in layout but the worst delay. The order of magnitude of its critical path is $O(M+2N)$[14]:

$$t_{\text{mult}} \approx [(M-1)+(N-1)]t_{\text{carry}} + (N-1)t_{\text{sum}} + t_{\text{and}}$$

where M and N are the bit-widths of the multiplier. The Wallace tree delay is $O(\log_2 N)$ and has the best speed among the three methods[8,11,14]. Its layout, however, cannot fit into a circuit, like a multiplier, with a regular area constraint. In our design, we used the CSA array for the best trade-offs in speed and layout structure. The critical path delay is $O(N-1)$[14]:

$$t_{\text{mult}} \approx t_{\text{and}} + (N-1)t_{\text{carry}} + t_{\text{merge}}$$

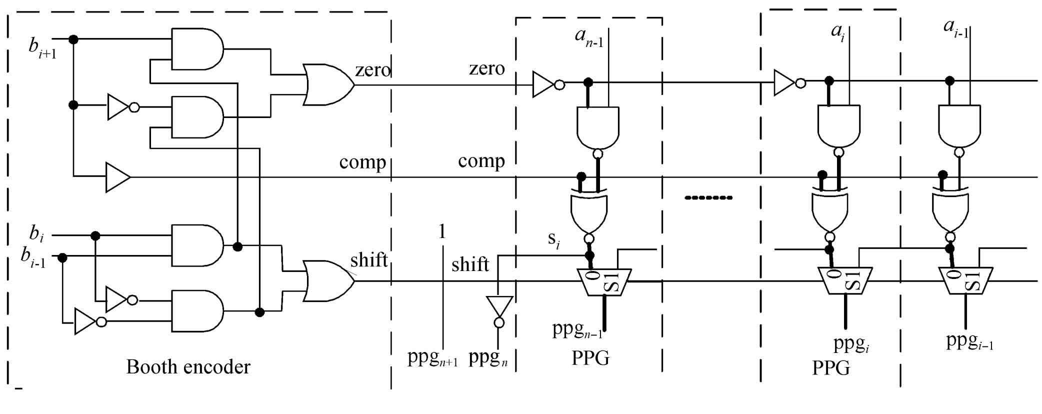


Fig. 4 Booth encoder and PPG

where t_{and} and t_{carry} are the delays through the starting AND gate and the carry-chain. An additional advantage of the CSA method is the addition of the pipelined structure for reconfigurability.

2.3 Level 3: final adder

We used a carry look-ahead adder (CLA) to speed up the addition of the final 36bit sum and carry vectors. The two operands are partitioned into multiple $k$bit groups. Each group produces sum (s_i) and carry (c_i) through generate (g_i) and propagate (p_i) signals:

$$g_i = a_i b_i,$$
$$p_i = a_i \oplus b_i \quad \text{or} \quad p_i = a_i + b_i$$
$$c_i = g_i + p_i c_{i-1}$$
$$s_i = a_i \oplus b_i \oplus c_{i-1} (\text{when} \quad p_i = a_i + b_i) \quad \text{or}$$
$$s_i = p_i \oplus c_{i-1} \qquad (\text{when} \quad p_i = a_i \oplus b_i)$$
$$c_i = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} +$$
$$p_i p_{i-1} \cdots p_1 p_0 g_0 + p_i p_{i-1} \cdots p_1 p_0 c_0$$

The critical path delay and area are $O(\log_2 N)$ and $O(N\log_2 N)$, respectively. It has been shown that the best trade-off in speed and area comes from the 4bit CLA[14]. Since the bit-widths of the final sum and carry vectors are both multiples of 4, the 4bit CLAs are the building blocks of the final adder.

In a typical 4bit CLA design, the critical path contains 6 stages of AND-OR delays (Fig. 5(a)). A modified architecture[9], as shown in Fig. 5(b), reduces it to 4 stages. We propose an architecture, shown in Fig. 5(c), to further reduce it to 3 stages. We used the transmission-gate logic to implement the 3-stage CLA for the best performance.

3 Layout implementation

As shown in Fig. 6(a), the tile-based FPGA contains multiple multipliers occupying contiguous logic block tiles. Each multiplier block occupies multiple contiguous logic block tiles in the same column. Multiple multiplier blocks can also be contiguous and replace the entire logic block column. The layout of each multiplier block has a height of 4 logic tiles and, similar to the logic blocks, the routing of the multiplier input and output signals are arranged to switch from and to the surrounding channel wires.

The layout of the multiplier block is thus constrained by the tile-based FPGA architecture and the routing structure. The implementation of the multiplier block is based on SMIC 0.13μm CMOS technology. The schematic and full layout is shown in Figs. 6(b) and 6(c). The height of the multiplier block is displayed as the horizontal side. The entire multiplier circuit contains 27,605 transistors laid out in a block

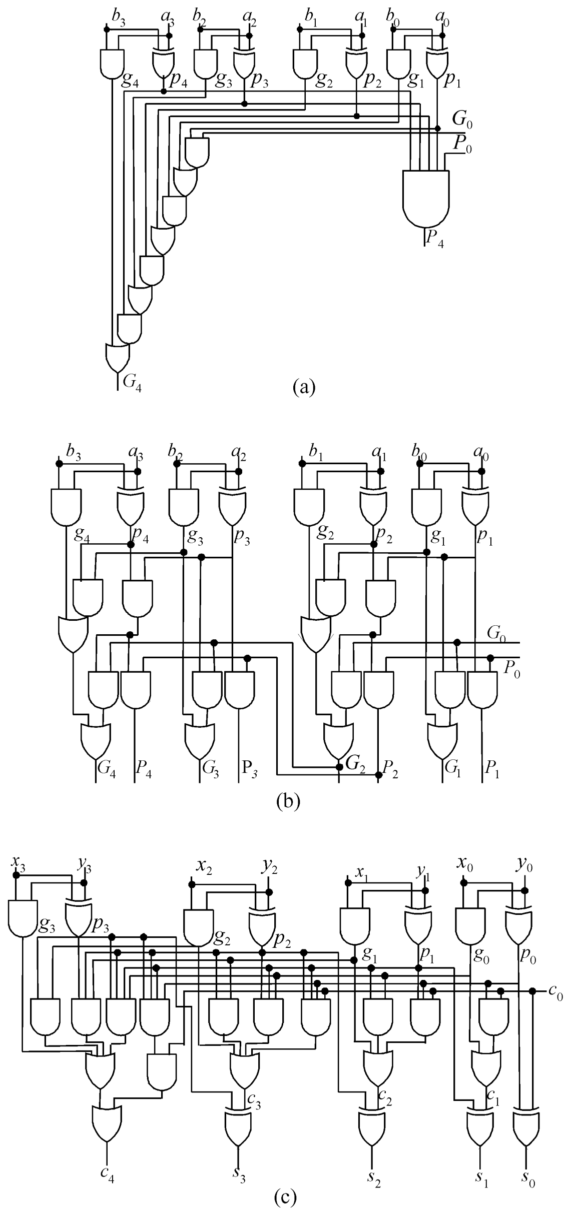


Fig.5 (a) Typical 4bit CLA architecture; (b) Architecture in Ref.[14]; (c) Our 4bit CLA architecture

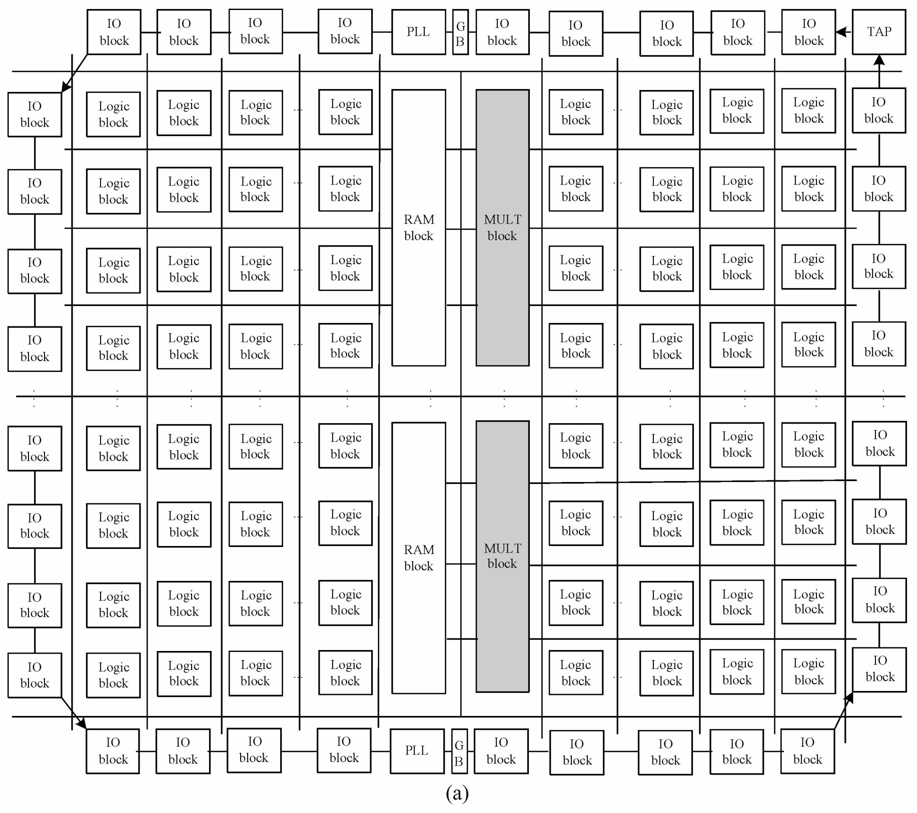
area of 63.50μm × 892.00μm.

4 Design verification

In addition to the standard DRC/ERC, LVS verifications, an extensive set of random test vectors have been applied to the multiplier circuit in the pre- and post- layout simulations. In particular, the multiplier circuit is simulated in the mixed-level simulator ModelSim-Hsim. In the mixed-level simulations, the multiplier is described in both the behavioral and transistor levels. The simulation vectors from the two levels of simulation are compared and verified. For the 18 × 18 multiplication, the worst-case delay is from

00 0000 0000 0000 0001 × 11 1111 1111 1111 1111

which results in the carry signal generation in all



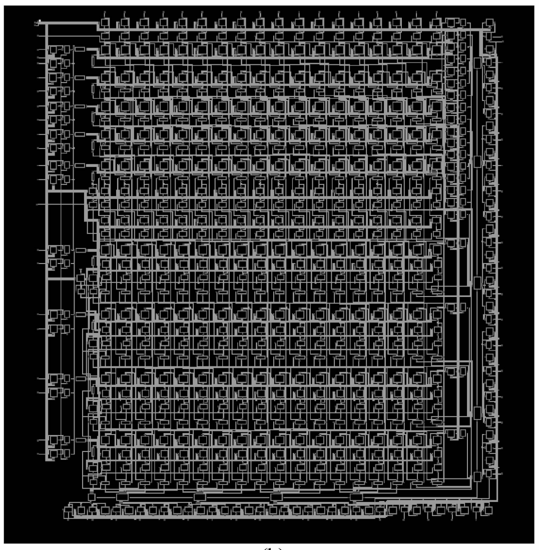


Fig.6 (a) Floorplan of the full chip; (b) Schematic of the multiplier block; (c) Layout of the multiplier block The layout has been rotated 90° to display better.

CLA bits. The pipelined and non-pipelined operation delays are 2.5 and 4.1ns, respectively, under the worst-corner condition.

5 Performance and area comparison

The comparison of the last-stage PPG from our proposed postponed increment technique versus the conventional 2's complement implementation and the 2's complementation technique proposed by Kang, *et al*.[3] is given in Table 2. All three cases are based on 0.13μm CMOS technology. In speed, our implementation is 88% and 72.4% faster. In area, it is 64.6% and 17.3% smaller.

Table 2 Comparison with two other PPG methods

	Conventional method (16×16 bit)	Conventional method (scaled to 18×18bit)	Kang *et al*. proposed method (16×16bit)	Kang *et al*. proposed method (scaled to 18×18bit)	Our method (18×18 bit)
Delay /ns	1.64	1.845	0.71	0.80	0.22
Area /μm^2	997	1122	480	1080	794

Table 3 shows the comparison of performance and area for multipliers from different providers.

Table 3 Comparison with proposed multipliers

	Reconfigurable	Delay /ns	Area /μm^2
Conventional LUT-based multiplier	yes		1938601
Conventional multiplier[5] (16×16bit)	no	4.84	14840
Conventional multiplier[5] (scaled to 18×18bit)	no	5.45	16695
Kang *et al*. proposed multiplier (16×16bit)	no	4.42	14859
Kang *et al*. proposed multiplier (scaled to 18×18bit)	no	4.97	16716
Haynes and Cheung[6] proposed multiplier	yes		79194
Multiplier block used in Spartan-3 family (0.09μm technology)	yes	4.00	
Multiplier block used in Spartan-3 family (scaled to 0.13μm technology)	yes	5.78	
Multiplier block used in Cyclone II family (0.09μm technology)	yes	4.00	
Multiplier block used in Cyclone II family (scaled to 0.13μm technology)	yes	5.78	
Our MB (18×18bit)	yes	4.10	56642

Table 3 shows that our multiplier is 29% smaller than the multiplier based on 4 × 4bit flexible array blocks (FABs) proposed by Haynes and Cheung[4]. As an embedded multiplier in FPGA, our implementation is smaller than the LUT-based multiplier by a factor of 33. Our implementation is 24.8% faster than the conventional multiplier[3] and 17.5% faster than the approach proposed by Kang *et al*.[3] Compared with the embedded multipliers in commercial FPGAs, our implementation is 29.1% faster than Xilinx's Spartan-3 family[1] and Altera's Cyclone II family[2].

6 Conclusion

A novel reconfigurable embedded pipelined 18×18 multiplier block used in FPGAs was proposed. We presented a novel Booth encoder and partial product generator and used a postponed increment technique to further reduce the delay and area. We used a 2-stage reconfigurable pipeline to enhance the throughput. Our method provides a more regular architecture, which can more easily extend the MB to a DSP block.

References

[1] Xilinx Inc, Spartan-3 FPGA Family: Complete Data Sheet, 2005
[2] Altera Corporation, Cyclone II Device Handbook, 2006
[3] Kang J Y, Gaudiot J L. A simple high-speed multiplier design. IEEE Trans Comput, 2006, 55(10): 1253
[4] Haynes S D, Cheung P Y K. A reconfigurable multiplier array for video image processing tasks, suitable for embedding in an FPGA structure. IEEE Symposium on FPGAs for Custom Computing Machines, 1998: 226
[5] Elguibaly F. A fast parallel multiplier-accumulator using the modified Booth algorithm. IEEE Trans Circuit Syst: Analog and Digital Signal Processing, 2000, 47(9): 902
[6] Lee H. Paramererizable multiplier library. MS Thesis, Berkeley Wireless Research Center, 1999
[7] Bewick G W. Fast multiplication: algorithms and implementation. PhD Thesis, Stanford University, 1994
[8] Wallace C S. A suggestion for a fast multiplier. IEEE Trans Comput, 1964, 1(2): 14
[9] Yu D S, Shen X B. Design of a 32-bit CMOS fix floating point multiplier. Chinese Journal of Semiconductors, 2001, 22(1): 91
[10] Xu Qi, Yuan Wei, Shen Xubang. The design of a new tree multiplier. Journal of Xidian University, 2002, 29(5): 580
[11] Fadavi-Ardekani J. M×N Booth encoded multiplier generator using optimized wallace trees. IEEE Trans Very Large Scale Integration, 1993, 1(2): 120
[12] SMIC. 0.13μm logic 1P8M Salicide 1.2/2.5/3.3V design rule. SMIC, 2001, Doc. No. TD-LO13-DR-2001
[13] Booth A D. A signed binary multiplication technique. Quarterly J Mechanical and Applied Math, 1951, 4: 236
[14] Rabey J M, Chandrakasan A, Nikolic B. Digital integrated circuits: a design perspective. 2nd ed. Prentice Hall, 2003
[15] Agrawal D P, Rao T R N. On multiple operand addition of signed binary numbers. IEEE Trans Comput, 1978, 27: 1068

FPGA 中专用可重构乘法器的设计

余洪敏[†]　陈陵都　刘忠立

（中国科学院半导体研究所，北京　100083）

摘要： 提出了一种新的嵌入在 FPGA 中可重构的流水线乘法器设计.该设计采用了改进的波茨编码算法,可以实现 18×18 有符号乘法或 17×17 无符号乘法.还提出了一种新的电路优化方法来减少部分积的数目,并且提出了一种新的乘法器版图布局,以便适应 tile-based FPGA 芯片设计所加的约束.该乘法器可以配置成同步或异步模式,也可以配置成带流水线的模式以满足高频操作.该设计很容易扩展成不同的输入和输出位宽.同时提出了一种新的超前进位加法器电路来产生最后的结果.采用了传输门逻辑来实现整个乘法器.乘法器采用了中芯国际 0.13μm CMOS 工艺来实现,完成 18×18 的乘法操作需要 4.1ns.全部使用 2 级的流水线时,时钟周期可以达到 2.5ns.这比商用乘法器快 29.1%,比其他乘法器快 17.5%.与传统的基于查找表的乘法器相比,该乘法器的面积为传统乘法器面积的 1/32.

关键词： FPGA；乘法器；可重构；改进的波茨算法；超前进位加法器；传输门逻辑

EEACC： 1265A

中图分类号： TN492　　**文献标识码：** A　　**文章编号：** 0253-4177(2008)11-2218-08

[†] 通信作者.Email:hmyu@red.semi.ac.cn

2008-05-26 收到,2008-06-30 定稿