社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－137（5）
2008 / 11 / 17

オペランドの和を利用した小面積乗算器

川島 裕崇† 高木 直史†

† 名古屋大学大学院 情報科学研究科 情報システム学専攻
〒 464-8603 名古屋市千種区不老町
E-mail: †{hkawashi,ntakagi}@takagi.i.is.nagoya-u.ac.jp

あらまし 並列乗算の最初のステップで生成される部分積のビット数を削減する手法を提案する．提案手法におけるビット数を削減した部分積を Integrated Partial Product (IPP) と呼ぶ．提案手法では乗数，被乗数のビットの値の組合せにより 4 つの場合を考え，1 つを選択することで IPP の値を決定する．オペランドの和を利用することにより，IPP の総ビット数は通常の部分積の総ビット数の約半分となる．提案手法は符号なし乗算，符号つき乗算の両方に適用できる．提案手法を用いた乗算器は従来の配列型乗算器や Wallace 乗算器より約 30%，2 ビット Booth の手法を用いた乗算器より約 10%小面積であった．

キーワード VLSI，演算回路，乗算

Area Efficient Multipliers Utilizing the Sum of Operands

Hirotaka KAWASHIMA† and Naofumi TAKAGI†

† Department of Information Engineering, Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8603 Japan
E-mail: †{hkawashi,ntakagi}@takagi.i.is.nagoya-u.ac.jp

Abstract A method to halve the number of partial product bits in multiplication is proposed. An integrated partial product (IPP) is introduced. The method separates the IPP into four cases. Each case is represented in half the number of the original partial product bits by utilizing the sum of the operands. The value of the IPP is obtained by selecting a value from the four cases. The proposed method is applicable to both unsigned and signed multiplication. Multipliers using the proposed method are smaller than array multipliers and Wallace multipliers by approximately 30%, and smaller than multipliers with radix-4 Booth's method by approximately 10%.

Key words VLSI, arithmetic circuit, multiplication

1. Introduction

Multiplication is an important operation and appears in various applications. A parallel multiplier consumes a large area in VLSI. Therefore, reduced area parallel multipliers are desired. Reducing circuit area gives us great advantages in downsizing of products, manufacturing cost, power consumption and so on. Recent years, increasing leakage power in VLSI is regarded as a major problem. The leakage power is always consumed while the power is supplied even if transistors are not activated. Reducing the number of circuit elements is an effective countermeasure to reduce the leakage power.

Parallel multiplication consists of partial product generation, partial product compression and final addition. Usually each partial product is a product of a multiplicand and a multiplier bit.

In this report, we propose a method to halve the number of partial product bits. We introduce an integrated partial product (IPP). The proposed method separates the IPP into four cases. Each case is represented in half the number of the original partial product bits by utilizing the sum of the operands. The value of the IPP is obtained by selecting a value from the four cases. The proposed method is applicable to both unsigned and signed multiplication. We have evaluated multipliers with the proposed method. They are smaller than array multipliers and Wallace multipliers [1] by approximately 30%, and smaller than multipliers with radix-4 Booth's method [1] by approximately 10%. Additionally, we discuss acceleration of the proposed method, where we

divide the addition of the operands into multiple sections to generate IPPs quickly.

The remainder of this report is organized as follows: Section 2 proposes a method to halve the number of partial product bits. Section 3 shows multipliers with the proposed method. Section 4 discusses acceleration of the proposed method. Section 5 evaluates the multipliers with the proposed method. Section 6 concludes this report.

2. A Method to Halve the Number of Partial Product Bits

First, we describe the proposing method for n-bit unsigned multiplication. We express multiplicand X as $[x_{n-1}x_{n-2}\cdots x_1x_0]$ and multiplier Y as $[y_{n-1}y_{n-2}\cdots y_1y_0]$. The values of X and Y are $\sum_{i=0}^{n-1} 2^i x_i$ and $\sum_{i=0}^{n-1} 2^i y_i$, respectively. We define X_i as $[x_ix_{i-1}\cdots x_1x_0]$ and Y_i as $[y_iy_{i-1}\cdots y_1y_0]$. We can transform $X \times Y$ as follows:

$$\begin{aligned} X \times Y &= \left(\sum_{i=0}^{n-1} 2^i x_i\right) \times \left(\sum_{i=0}^{n-1} 2^i y_i\right) \\ &= X_{n-2} \times Y_{n-2} \\ &\quad +2^{n-1}(2^{n-1}x_{n-1}y_{n-1} + x_{n-1}Y_{n-2} + y_{n-1}X_{n-2}) \end{aligned}$$

By transforming iteratively, $X \times Y$ is calculated as follows:

$$X \times Y = x_0y_0 + \sum_{i=1}^{n-1} 2^i \left(2^i x_iy_i + x_iY_{i-1} + y_iX_{i-1}\right)$$

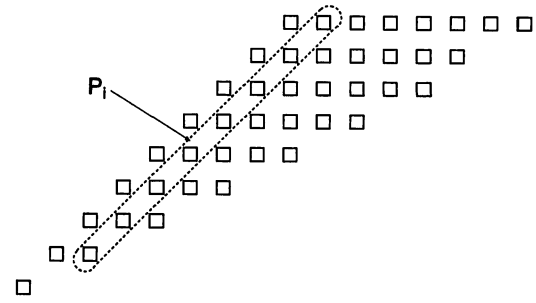
This equation is used for serial-serial multiplication in [4]~[6]. We call $2^i x_iy_i + x_iY_{i-1} + y_iX_{i-1}$ the i-th integrated partial product (IPP) P_i. We can obtain an IPP efficiently utilizing the sum of the multiplicand and the multiplier. We separate an IPP into four cases by (x_i, y_i) as follows:

$$P_i = \begin{cases} 0 & \text{if } (x_i, y_i) = (0,0) \\ X_{i-1} & \text{if } (x_i, y_i) = (0,1) \\ Y_{i-1} & \text{if } (x_i, y_i) = (1,0) \\ 2^i + X_{i-1} + Y_{i-1} & \text{if } (x_i, y_i) = (1,1) \end{cases}$$

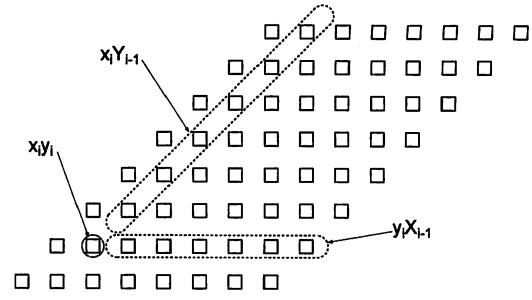
We have to calculate $X_{i-1} + Y_{i-1}$ in the case of $(x_i, y_i) = (1,1)$. We define S as $X + Y$, and express S as $[s_ns_{n-1}s_{n-2}\cdots s_1s_0]$. We define S_i as $[s_is_{i-1}\cdots s_1s_0]$. We can express P_i in an $(i+2)$-bit unsigned binary representation as follows:

$$\begin{cases} [0000\cdots00] & \text{if } (x_i, y_i) = (0,0) \\ [00x_{i-1}x_{i-2}\cdots x_1x_0] & \text{if } (x_i, y_i) = (0,1) \\ [00y_{i-1}y_{i-2}\cdots y_1y_0] & \text{if } (x_i, y_i) = (1,0) \\ [s_i\overline{s_i}s_{i-1}s_{i-2}\cdots s_1s_0] & \text{if } (x_i, y_i) = (1,1) \end{cases}$$

Note that when $(x_i, y_i) = (1,1)$, $X_{i-1} + Y_{i-1} = S_i$. Since we can use S to calculate all IPPs, we calculate S only once. Bit diagrams of the IPPs and the ordinary partial products are shown in Fig. 1. In Fig. 1(a) the positions of the bits of P_i are shown, and in Fig. 1(b) the positions of the ordinary partial product bits corresponding to P_i are shown. The proposed method generates only $\frac{1}{2}n^2 + \frac{3}{2}n - 1$ bits, while an ordinary n-bit multiplier generates n^2 partial product bits. The proposed method achieves to halve the number of partial product bits.



(a) A bit diagram of the IPPs



(b) A bit diagram of the partial products in ordinary multiplication

図 1 Bit diagrams of the IPPs and the partial products

We can apply the proposed method to signed multiplication by slight modification. X and Y are expressed in two's complement representation as $[x_{n-1}x_{n-2}\cdots x_1x_0]$ and $[y_{n-1}y_{n-2}\cdots y_1y_0]$, respectively. The values of X and Y are $-2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i$ and $-2^{n-1}y_{n-1} + \sum_{i=0}^{n-2} 2^i y_i$, respectively. Signed multiplication is shown as:

$$\begin{aligned} X \times Y &= \left(-2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i\right) \times \left(-2^{n-1}y_{n-1} + \sum_{i=0}^{n-2} 2^i y_i\right) \\ &= X_{n-2} \times Y_{n-2} \\ &\quad +2^{n-1}\left(2^{n-1}x_{n-1}y_{n-1} - x_{n-1}Y_{n-2} - y_{n-1}X_{n-2}\right) \end{aligned}$$

The IPPs for $i = 0$ to $n-2$ are the same as those of the unsigned multiplication. Namely, $X_{n-2} \times Y_{n-2}$ can be calculated in the same way as unsigned multiplication. We separate $P^*_{n-1} = 2^{n-1}x_{n-1}y_{n-1} - x_{n-1}Y_{n-2} - y_{n-1}X_{n-2}$ into four cases by (x_{n-1}, y_{n-1}).

$$P^*_{n-1} = \begin{cases} 0 & \text{if } (x_{n-1}, y_{n-1}) = (0,0) \\ -X_{n-2} & \text{if } (x_{n-1}, y_{n-1}) = (0,1) \\ -Y_{n-2} & \text{if } (x_{n-1}, y_{n-1}) = (1,0) \\ 2^{n-1} - X_{n-2} - Y_{n-2} & \text{if } (x_{n-1}, y_{n-1}) = (1,1) \end{cases}$$

We define the $(n-1)$-th IPP P_{n-1} as $P^*_{n-1} - 1$. Then, P_{n-1} is expressed in a $(n+1)$-bit two's complement binary representation as follows:

$$\begin{cases} [1111\cdots11] & \text{if } (x_{n-1}, y_{n-1}) = (0,0) \\ [11\overline{x_{n-2}}\ \overline{x_{n-3}}\cdots\overline{x_1}\ \overline{x_0}] & \text{if } (x_{n-1}, y_{n-1}) = (0,1) \\ [11\overline{y_{n-2}}\ \overline{y_{n-3}}\cdots\overline{y_1}\ \overline{y_0}] & \text{if } (x_{n-1}, y_{n-1}) = (1,0) \\ [s_{n-1}s_{n-1}\overline{s_{n-2}}\ \overline{s_{n-3}}\cdots\overline{s_1}\ \overline{s_0}] & \text{if } (x_{n-1}, y_{n-1}) = (1,1) \end{cases}$$

Note that when $(x_{n-1}, y_{n-1}) = (1,1)$, $X_{n-2} + Y_{n-2} = S_{n-1}$, and that $-S_{n-1} = -2^n + \overline{S_{n-1}} + 1$ where $\overline{S_{n-1}}$ is the bitwise inversion of S_{n-1}. Finally, signed multiplication is performed as

$$X \times Y = x_0 y_0 + \left(\sum_{i=1}^{n-2} 2^i P_i\right) + 2^{n-1} P_{n-1} + 2^{n-1}.$$

The number of IPP bits of signed multiplication is $\frac{1}{2}n^2 + \frac{3}{2}n$.

3. Reduced Area Multipliers

A block diagram of a multiplier using the proposed method is shown in Fig. 2. The multiplier consists of an operand adder, an operand recoder, an IPP generator, an IPP compressor and a final adder. The operand adder is a carry propagate adder to calculate the sum of the operands. The operand recoder generates $x_i \wedge y_i$, $\overline{x_i} \wedge y_i$ and $x_i \wedge \overline{y_i}$. It consists of $3n$ 2-input AND gates and $2n$ inverters. The IPP generator consists of $\frac{1}{2}n^2 + \frac{3}{2}n - 1$ decoder cells. Each decoder cell generates an IPP bit from (x_j, y_j, s_j) and $(x_i \wedge y_i, \overline{x_i} \wedge y_i, x_i \wedge \overline{y_i})$. The IPP compressor compresses the IPPs into two numbers by carry save additions. The final adder is a carry propagate adder and sums up the two numbers.

We define $p_{i,j}$ as the j-th bit of P_i. The decoder cell calculates $p_{i,j}$ where $0 \leqq j \leqq i-1$, $p_{i,i}$ and $p_{i,i+1}$ where $1 \leqq i \leqq n-1$ as follows:

$$\begin{aligned} p_{i,j} &= ((\overline{x_i} \wedge y_i) \wedge x_j) \vee ((x_i \wedge \overline{y_i}) \wedge y_j) \vee ((x_i \wedge y_i) \wedge s_j) \\ p_{i,i} &= (x_i \wedge y_i) \wedge \overline{s_i} \\ p_{i,i+1} &= (x_i \wedge y_i) \wedge s_i \end{aligned}$$

For signed multiplication, P_{n-1} is calculated as follows, where $0 \leqq j \leqq n-2$:

$$\begin{aligned} p_{n-1,j} &= \overline{((\overline{x_{n-1}} \wedge y_{n-1}) \wedge x_j)} \\ &\quad \wedge \overline{((x_{n-1} \wedge \overline{y_{n-1}}) \wedge y_j)} \\ &\quad \wedge \overline{((x_{n-1} \wedge y_{n-1}) \wedge s_j)} \\ p_{n-1,n-1} &= s_{n-1} \vee \overline{(x_{n-1} \wedge y_{n-1})} \\ p_{n-1,n} &= p_{n-1,n-1} \end{aligned}$$

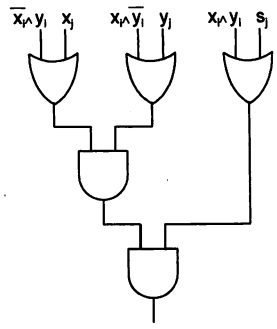


図 3 The decoder cell of the proposed method

The decoder cell of the proposed method is shown in Fig. 3. It requires three 2-input OR gates and two 2-input AND gates.

4. Acceleration of the Proposed Method

In the proposed method, IPP bits with higher weight depend on higher bits of S. Therefore, they can be generated after the addition of the operands having proceeded.

We can accelerate the method by dividing the addition of the operands into multiple sections. As an example, we explain the case of dividing the addition into two sections. We define X_L, X_H, Y_L, Y_H, S_L and S_H as follows:

$$\begin{aligned} X_L &= [x_{\frac{1}{2}n-1} x_{\frac{1}{2}n-2} \cdots x_1 x_0] \\ X_H &= [x_{n-1} x_{n-2} \cdots x_{\frac{1}{2}n+1} x_{\frac{1}{2}n}] \\ Y_L &= [y_{\frac{1}{2}n-1} y_{\frac{1}{2}n-2} \cdots y_1 y_0] \\ Y_H &= [y_{n-1} y_{n-2} \cdots y_{\frac{1}{2}n+1} y_{\frac{1}{2}n}] \\ S_L &= X_L + Y_L \\ S_H &= X_H + Y_H \end{aligned}$$

Note that $S_L + 2^{\frac{n}{2}} S_H = S$.

We generate the IPP bits with higher weight using the bits of S_H instead of the higher bits of S. Then, for $i \geqq \frac{n}{2}$, two shorter IPPs, P_{Li} and P_{Hi}, corresponding to P_i are generated. P_{Li} depends on S_L and consists of $\frac{1}{2}n + 1$ bits. P_{Hi} depends on S_H and consists of $i + 2 - \frac{n}{2}$ bits. Note that $P_{Li} + 2^{\frac{n}{2}} P_{Hi} = P_i$. A bit diagram of IPPs in this case is shown in Fig. 4. The bits above the dashed line belong to P_{Li}'s and those below the dashed line belong to P_{Hi}'s. Thus, P_{Hi} and P_{Li} are generated in parallel.

The number of IPP bits increases by dividing the addition of the operands. Both S_L and S_H consist of $\frac{1}{2}n + 1$ bits, while S consists of $n+1$ bits, because of the carry-out of the additions. Therefore, there are $\frac{1}{2}n$ additional IPP bits corresponding to the most significant bit of S_L.The IPP bits on the 5th row in Fig. 4 are corresponding to the most significant bit of S_L. The more the number of sections is, the faster the calculation is and the more the additional IPP bits

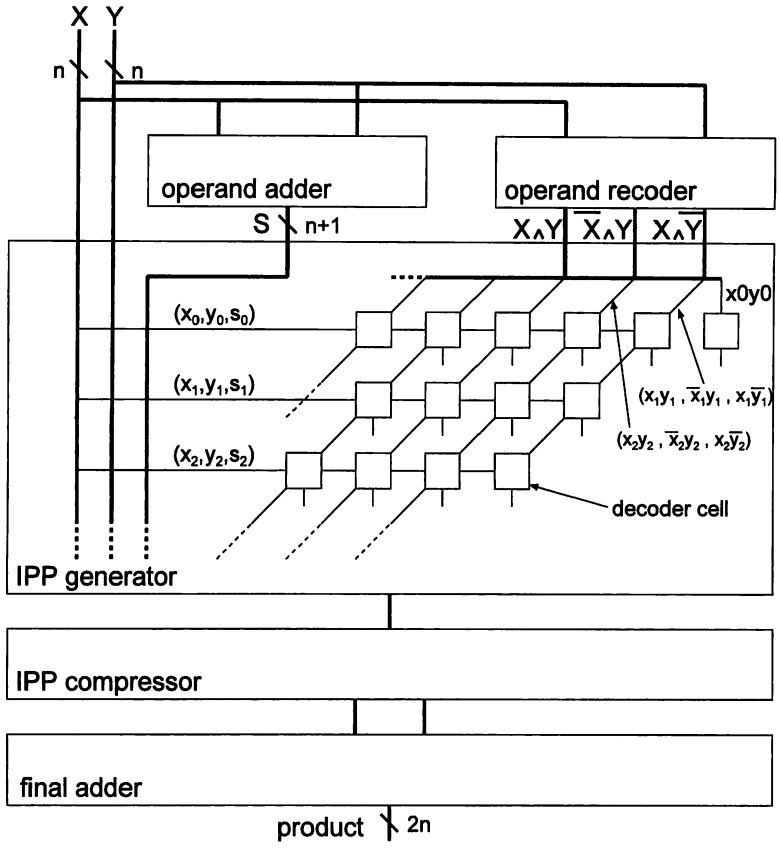


図 2 A block diagram of a multiplier with the proposed method

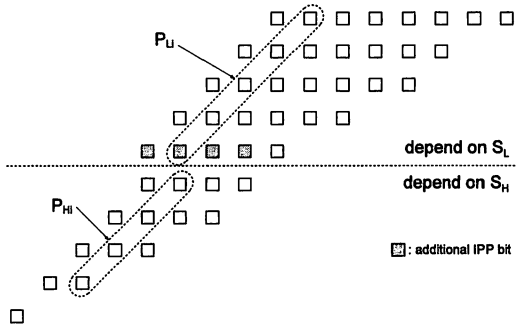


図 4 A bit diagram of the IPPs in the case of dividing the addition into two sections

are.

5. Evaluation

We evaluate the multipliers with the proposed method by comparing their circuit area and delay with those of conventional multipliers. Input widths are 16-bit, 32-bit and 64-bit. We evaluate three constructions of the multipliers with the proposed method.

proposed(1) : multipliers without dividing the addition of the operands, and using array-type IPP compressor.

proposed(2) : multipliers with dividing the addition of the operands into two sections, and using array-type IPP compressor.

proposed(3) : multipliers without dividing the addition of the operands, and using tree-type IPP compressor.

We compare the above multipliers with the following three multipliers.

array : array multipliers without radix-4 Booth's method

array+Booth : array multipliers with radix-4 Booth's method

Wallace : Wallace multipliers without radix-4 Booth's method

We designed the multipliers using the cell libraries for Rohm 0.18 μm 5-metal CMOS technology and Semiconductor Technology Academic Research Center (STARC) 90 nm 6-metal

CMOS technology. Both libraries are provided by VLSI Design and Education Center (VDEC), the University of Tokyo. We synthesized the multipliers with Synopsys Design Compiler. We used Cadence SOC Encounter and Synopsys Astro for the physical design with the 0.18 μm cell library and the 90 nm cell library, respectively. All cells and wires of the multipliers are placed and routed with over 95% core utilization.

We have optimized the multipliers in two ways. In one optimization, we have optimized the multipliers to minimize the circuit area. Ripple carry adders are used as the operand adder and the final adder. The circuit area and the delay of the multipliers are shown in Table 1. **Proposed(1)** is approximately 30% smaller than **array** and **Wallace**. **Proposed(1)** is approximately 10% smaller than **array+Booth**.

In the other optimization, we have optimized the multipliers to minimize the delay. We use an adder from DesignWare IP Library, Synopsys, Inc. as the operand adder and the the final adder. The circuit area and the delay of the multipliers are shown in Table 2. Although **proposed(2)** is larger than **proposed(1)**, **proposed(2)** is much faster than **proposed(1)**. The delay of **proposed(2)** is almost the same as **array+Booth**. The circuit area of **proposed(2)** is smaller than that of **array+Booth**. When the 90 nm cell library is used, **proposed(3)** is faster and smaller than **Wallace**.

6. Conclusion

We have proposed a method to halve the number of partial product bits. We have introduced the integrated partial product (IPP). The IPPs are calculated efficiently utilizing the sum of the operands. The IPPs have half the number of the original partial product bits. Additionally, we have accelerated the proposed method, where we divide the addition of the operands into multiple sections to generate IPPs quickly.

We have shown that multipliers using the proposed method are smaller than existing multipliers. The proposed method is useful for systems that need to be compact, and require little cost and power. Although we have evaluated only three constructions of multipliers using the proposed method, there are many other constructions of the multipliers using the proposed method. It may be possible to improve the multipliers by changing the number of sections and the point of division.

References

[1] C. S. Wallace, "A Suggestion for a Fast Multiplier," IEEE Transactions on Electronic Computers, Vol.EC-13, pp.14-17, 1964.

[2] A. D. Booth, "A Signed Binary Multiplication Technique," Quart. J. Mech. Appl. Math., vol.4, part 2, pp.236-240, 1951.

[3] C. R. Baugh and B. A. Wooley, "A Two's Complement Parallel Array Multiplication Algorithm," IEEE Trans. Comput., vol.C-22, no.12, pp.1045-1047, Dec. 1973.

[4] I. N. Chen and R. Willoner, "An O(n) Parallel Multiplier with Bit Sequential Input and Output," IEEE Trans. Comput., vol.C-28, no.10, pp.721-727, Oct. 1979.

[5] N. R. Strader and V. T. Rhyne, "A Canonical Bit-Sequential Multiplier," IEEE Trans. Comput., vol.C-31, no.8, pp.791-795, Aug. 1982.

[6] R. Gnanasekaran, "On a Bit-Serial Input and Bit-Serial Output Multiplier," IEEE Trans. Comput., vol.C-32, no.9, pp.878-880, Sep. 1983.

表 1 Area(μm^2) and delay(ns) of multipliers of the smallest construction

		unsigned				signed			
		0.18 μm		90 nm		0.18 μm		90 nm	
		area	delay	area	delay	area	delay	area	delay
16bit	array	24946.8	11.47	5790.6	6.70	24753.1	11.61	5748.2	6.73
	array+Booth	22902.3	13.37	4522.5	5.98	21618.5	13.08	4251.6	5.59
	Wallace	25085.9	10.18	5823.8	4.69	24892.2	10.30	5780.0	4.68
	proposed(1)	19459.4	11.35	4443.9	5.52	19615.9	11.40	4479.6	5.71
	proposed(2)	20470.9	11.42	4667.5	5.34	21652.1	11.42	4939.4	5.40
	proposed(3)	19650.0	11.44	4491.4	5.06	19765.6	11.40	4515.3	5.13
32bit	array	103775.1	23.67	24127.1	14.09	103362.3	23.83	24037.2	14.21
	array+Booth	84552.3	22.37	17362.1	10.84	81821.4	23.36	16651.1	10.77
	Wallace	103911.6	20.73	24160.9	9.27	103502.0	20.62	24069.5	9.37
	proposed(1)	72165.1	22.20	16471.4	11.01	72483.0	22.39	16542.5	11.13
	proposed(2)	74397.7	22.58	16967.9	10.35	76746.4	22.24	17513.5	10.35
	proposed(3)	72357.5	22.66	16518.7	10.35	72632.6	22.82	16577.8	10.15
64bit	array	423104.6	48.00	98448.4	28.95	422256.8	48.18	98264.0	28.90
	array+Booth	322622.9	41.93	66727.6	20.84	317325.6	41.25	65437.1	20.25
	Wallace	423241.2	42.59	98482.8	18.85	422398.8	42.28	98295.3	18.79
	proposed(1)	279052.3	43.07	63473.9	22.42	279697.0	43.20	63616.1	23.00
	proposed(2)	283645.9	43.41	64508.2	21.01	288414.8	43.61	65600.0	21.10
	proposed(3)	279246.2	44.34	63521.6	20.23	279849.0	44.70	63652.7	20.19

表 2 Area(μm^2) and delay(ns) of multipliers of the fastest construction

		unsigned				signed			
		0.18 μm		90 nm		0.18 μm		90 nm	
		area	delay	area	delay	area	delay	area	delay
16bit	array	27478.9	8.15	6038.6	5.28	27262.8	8.29	5979.2	5.26
	array+Booth	35103.0	5.72	5578.9	3.14	35041.5	6.42	6090.7	3.10
	Wallace	29417.4	4.43	6040.1	2.23	29214.1	4.44	5979.2	2.20
	proposed(1)	24423.4	7.42	4755.0	4.81	24865.6	7.14	4675.0	5.07
	proposed(2)	27122.5	5.88	5003.8	3.27	27214.4	6.37	5374.7	3.46
	proposed(3)	29893.8	4.96	6845.9	1.69	31506.8	4.72	6829.0	1.76
32bit	array	108445.4	15.73	24676.5	10.65	108288.2	15.88	24614.8	10.71
	array+Booth	105590.0	9.80	17550.2	5.93	107904.9	10.73	18226.3	6.03
	Wallace	114321.6	5.68	24687.5	3.16	112352.5	5.78	24564.8	3.10
	proposed(1)	83170.4	13.49	17400.1	10.23	86848.3	13.13	17304.5	10.43
	proposed(2)	89573.8	10.30	17975.8	6.17	91805.6	10.48	19137.6	6.42
	proposed(3)	98790.6	7.04	23685.2	2.60	101488.9	6.81	24273.9	2.71
64bit	array	432733.4	31.22	99582.5	21.33	432169.9	30.53	99352.6	21.50
	array+Booth	383888.0	18.14	67846.4	11.72	384976.5	19.35	65672.5	11.97
	Wallace	442830.3	8.55	100274.0	4.38	442896.9	7.96	100299.2	4.33
	proposed(1)	301362.8	26.54	64956.0	21.68	309527.1	26.01	65262.7	21.03
	proposed(2)	319258.6	18.80	67028.3	12.36	320856.1	19.91	68686.1	12.31
	proposed(3)	345247.5	10.32	89151.5	4.03	341909.2	9.72	88482.4	4.12