Chapter 6. High-Speed Multiplication

6.1. Introduction

A multiplication operation can be further divided into two steps

Step 1. Generation of partial products;

Step 2. Accumulation of the partial products.

Therefore, in order to have high-speed multiplication, we need to speedup either or both of the steps.

Lets look at the following multiplication example of $13_{10} \times 15_{10} = (1101)_2 \times (1111)_2$.

$$\begin{array}{cccccccccc}
 & & & & & 1 & 1 & 0 & 1 \\
 & & & \times & & 1 & 1 & 1 & 1 \\
\hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
+ & & 1 & 1 & 0 & 1 & & & \\
\hline
 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}$$

The first step of the multiplication generates 4 partial products, and then the following accumulation step performs 3 two-operand addition or one four-operand addition.

Consider to convert the multiply $15_{10} = (1111)_2$ into NAF (refer to §.6.2.1.3) using canonical recoding, then we have $15_{10} = (1000\overline{1})_2$. Now lets perform the multiplication as follows

$$\begin{array}{cccccccccc}
 & & & & & & 1 & 1 & 0 & 1 \\
 & & & & \times & 1 & 0 & 0 & 0 & \overline{1} \\
\hline
 & & & & & & -(1 & 1 & 0 & 1) \\
+ & 1 & 1 & 0 & 1 & & & & & \\
\hline
 & 1 & 1 & 0 & 0 & & 0 & 0 & 1 & 1
\end{array}$$

It can be seen that the accumulation step now requires only one subtraction. In fact, reduction of the number of partial products is one effective method to enhance multiplier performance.

6.2. High-Speed Multiplier

6.2.1 Reducing the Number of Partial Products

1. From the fact
$$\text{Product} \;=\; \text{Multiplicand} \;\times\; \text{Multiplier},$$
we have
$$\#\text{ Partial products} \;=\; \#\text{ of 1's in the binary multiplier.}$$
2. An example: $A \times X$, where $A = 10110$ and $X = 11111$
3. Reducing the number of partial products can be realized by reducing the number of 1's in the multiplier.
4. In the above example of $A \times X$, convert the multiplier X into BSD: $X = 11111 = 10000\bar{1}$.
   - Redo the multiplication with X in BSD form.
   - It can be seen that the partial products count is reduced from 5 to 2.

6.2.1.1 Booth's Algorithm

1. Idea: to replace a sequence of consecutive 1's with a one, a minus one, and a zero-run.
2. The next step is to use the carry-out signal obtained with conditions to select the results for its neighboring bit with higher weight.
3. For example,
$$X: \quad \cdots \;\; 0 \;\; 0 \;\; 1 \;\; 1 \;\; \cdots \;\; 1 \;\; 1 \;\; 0 \;\; 0 \;\; \cdots$$
$$\Downarrow$$
$$Y: \quad \cdots \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; \cdots \;\; 0 \;\; \bar{1} \;\; 0 \;\; 0 \;\; \cdots$$

This transformation can be proved to be correct: Assume that there are k 1's in X. And let the least significant bit 1 have the weight 2^i. Then X has value of

$$2^i + 2^{i+1} + \cdots + 2^{i+k-1} = \sum_{j=i}^{i+k-1} 2^j = 2^{i+k} - 2^i.$$

On the other hand, the value of Y can be shown as

$$(-1) \times 2^i + 2^{i+k-1} = 2^{i+k} - 2^i.$$

So Y has the same value as X.

4. Booth's algorithm can be summarized into a table form:

x_i	x_{i-1}	y_i
0	0	0
0	1	1
1	0	$\overline{1}$
1	1	0

- In the table, x_{i-1} serves as a reference bit.
- To convert a binary number $X = (x_{n-1}x_{n-2}\cdots x_0)$ into Booth form, we can follow the above table by letting $i = 0, 1, \ldots, n$.
- We always assume that $x_{-1} = 0$ in Booth's algorithm.
- Note that this table is a parallel algorithm where we can start recoding for all values of i at the same time.
- The worst case scenario: Booth's algorithm does not always reduce the count of 1's. (show with an example: $X = 10101$)
- Question: If we convert an n-bit binary number into Booth form, what is the average number of nonzeros (1 or $\overline{1}$) in the result?

6.2.1.2 Two-Bit Booth's Algorithm

1. An improvement to Booth's algorithm with reduced average number of 1's.
2. Instead of checking two bits x_ix_{i-1} to generate one digit y_i, 2-bit Booth's algorithm each time generates two digits by examining three bits. This strategy will eliminate the inefficiency of Booth's method for the case of the isolated 1's.

3. Two-bit Booth's algorithm can be summarized into the following table:

x_{i+1}	x_i	x_{i-1}	y_{i+1}	y_i	Operation
0	0	0	0	0	$+0$
0	1	0	0	1	$+A$
1	0	0	$\overline{1}$	0	$-2A$
1	1	0	0	$\overline{1}$	$-A$
0	0	1	0	1	$+A$
0	1	1	1	0	$+2A$
1	0	1	0	$\overline{1}$	$-A$
1	1	1	0	0	$+0$

- This algorithm is also called Radix-4 Booth's Algorithm.
- In the table, x_{i-1} serves as a reference bit.
- To convert a binary number $X = (x_{n-1}x_{n-2}\cdots x_0)$, one can follow the above table by letting $i = 0, 1, \ldots, n$. Note that we assume that $x_{-1} = 0$.
- This table is also a parallel algorithm where we can start recoding for the input bits in parallel, or start from either end of the binary number.
- The worst case scenario: Two-bit Booth's is better than Booth's, but it still does not always reduce the count of 1's. (show with an example: $X_1 = 10101$ and $X_2 = 101010$)
- Question: If we convert an n-bit binary number into two-bit Booth form, what is the average number of nonzeros (1 or $\overline{1}$) in the result?

6.2.1.3 Canonical Recoding

1. Is there a recoding algorithm that generates a BSD form with nonzeros count always less than or equal to the that of the binary input?

2. Minimal weight SD form: The number of nonzero digits in an SD form is the minimal. (show this concept with an example)

3. Non-adjacent form (NAF): An SD form where there is no two consecutive nonzero digits. Note that NAF is a special case of minimal weight SD form. (show this concept with an example)

4. Canonical recoding algorithm can convert a binary input into its NAF, which can be summarized into the following table:

x_{i+1}	x_i	c_i	y_i	c_{i+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	0	0
1	1	0	$\overline{1}$	1
0	0	1	1	0
0	1	1	0	1
1	0	1	$\overline{1}$	1
1	1	1	0	1

   - In the table, c_i serves a similar role as a carry in addition operation.
   - To convert a binary number $X = (x_{n-1}x_{n-2}\cdots x_0)$, one can follow the above table by letting $i = 0, 1, \ldots, n$. Note that we assume that $c_0 = 0$.
   - Is this a sequential algorithm or a parallel algorithm?
   - The worst case scenario: The resultant BSD has the same number of nonzero digits as in the binary input.
   - Question: If we convert an n-bit binary number into NAF using Canonical recoding, what is the average number of nonzeros (1 or $\overline{1}$) in the result?
     - Hamming weight: The number of nonzero digits in a sequence.
     - The average Hamming weight of an n-bit binary number is $n/2$.
     - The average Hamming weight of an n-digit NAF is $n/3$.

6.2.2 Accumulating Partial Products

Consider (n-bit) $\times$ (n-bit) multiplication, where we let multiplicand and multiplier be $A = (a_{n-1}\cdots a_0)$ and $X = (x_{n-1}\cdots x_0)$, respectively. Let P denote the product of A and X. Since $0 \leqslant A, X \leqslant 2^n - 1$, product $P = (p_{2n-1}\cdots p_0)$ has up to $2n$ bits. This can be seen as follows:

$$\begin{aligned} P &= A \times X \\ &\leqslant (2^n - 1)(2^n - 1) \\ &= 2^{2n} - 2^{n+1} + 1 \\ &\leqslant 2^{2n} - 1. \end{aligned}$$

Lets take an example of (4-bit) $\times$ (4-bit) multiplier, where $A = (a_3a_2a_1a_0)$ and $X = (x_3x_2x_1x_0)$. Then the ten product bits $P = (p_7p_6 \cdots p_0)$ can be obtained with the scheme given below:

				a_3	a_2	a_1	a_0
		$\times$		x_3	x_2	x_1	x_0
				a_3x_0	a_2x_0	a_1x_0	a_0x_0
			a_3x_1	a_2x_1	a_1x_1	a_0x_1	
		a_3x_2	a_2x_2	a_1x_2	a_0x_2		
	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

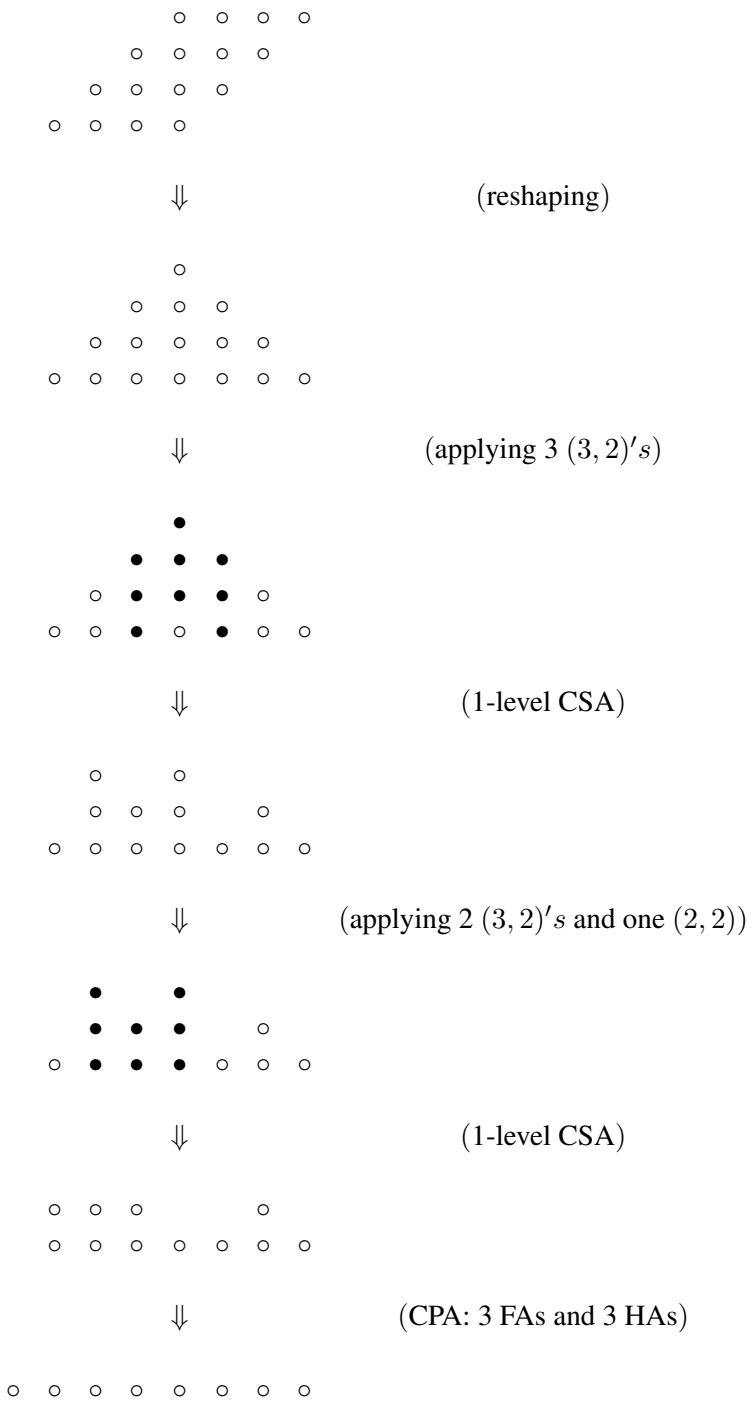
1. The second step in a multiplication operation is accumulating partial products, which can be implemented using carry-save addition and carry-propagate addition.

2. Therefore, this is a task of designing efficient Carry-save adder for a class of special CSA tree.

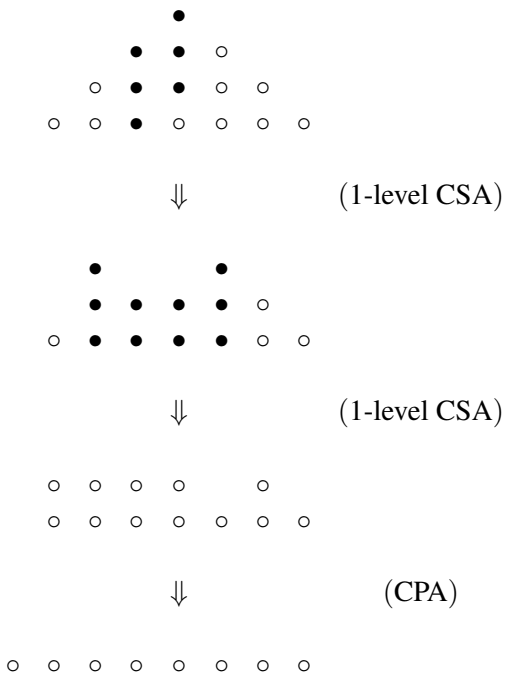
3. Consider a multiplier of 4-bit by 4-bit.

   (a) The first step is generation of 16 partial product bits, which requires 16 two-input AND gates and incurred one gate delay with Δ_G.

   (b) Denote a partial product bit with a circular-dot $\circ$. Then it can be seen that the CSA tree of 16 partial product bits form a triangular tree with 4 rows and the base consisting of 7 dots. By checking with the table given in the section of Carry-Save Adder in Chapter 5, we find that it requires 2 levels of CSA before the final CPA.



(c) An alternate method has also 2 units of CSA and one unit of CPA.



4. In general, a multiplier of n-bit by n-bit will have a triangular CSA tree with n rows and a base consisting of $2n - 1$ dots.

5. In multiplication, if the multiplier has different number of bits from the multiplicand, the CSA tree usually forms an isosceles trapezoid.

6.2.3 Squaring Unit (optional)

Consider squaring operation $P = A^2$. Let A be a 4-bit binary number. Then it follows

$$
\begin{array}{cccccccc}
 & & & & a_3 & a_2 & a_1 & a_0 \\
\times & & & & a_3 & a_2 & a_1 & a_0 \\
\hline
 & & & & a_3a_0 & a_2a_0 & a_1a_0 & a_0^2 \\
 & & & a_3a_1 & a_2a_1 & a_1^2 & a_0a_1 & \\
 & & a_3a_2 & a_2^2 & a_1a_2 & a_0a_2 & & \\
 & a_3^2 & a_2a_3 & a_1a_3 & a_0a_3 & & & \\
 & & & & \Downarrow & & & \\
 & a_3 & a_3a_2 & a_2 & a_3a_0 & a_2a_0 & a_1a_0 & a_0 \\
 & & a_2a_3 & a_3a_1 & a_2a_1 & a_1 & a_0a_1 & \\
 & & & a_1a_3 & a_1a_2 & a_0a_2 & & \\
 & & & & a_0a_3 & & & \\
 & & & & \Downarrow & & & \\
 & a_3 & a_1a_3 & a_2 & a_0a_2 & a_1 & & a_0 \\
 & a_2a_3 & & a_1a_2 & & a_0a_1 & & \\
 & & & a_0a_3 & & & & \\
+ & & & & & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

Note that $a_1 + a_0a_1 = a_1(\overline{a_0} + a_0) + a_0a_1 = 2a_0a_1 + \overline{a_0}a_1$.

6.3. Array Multipliers

We consider (5-bit) $\times$ (5-bit) multiplier, where $A = (a_4a_3a_2a_1a_0)$ and $X = (x_4x_3x_2x_1x_0)$. Then the ten product bits $P = (p_9p_8 \cdots p_0)$ can be obtained with the scheme given below:

					a_4	a_3	a_2	a_1	a_0
			$\times$		x_4	x_3	x_2	x_1	x_0
					a_4x_0	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				a_4x_1	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			a_4x_2	a_3x_2	a_2x_2	a_1x_2	a_0x_2		
		a_4x_3	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
$+$	a_4x_4	a_3x_4	a_2x_4	a_1x_4	a_0x_4				
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Assume all the partial product bits $a_ix_j, i, j = 0, \ldots, 4$, have been generated, an array architecture to realize the above multiplication scheme using full adder as cell can be given as

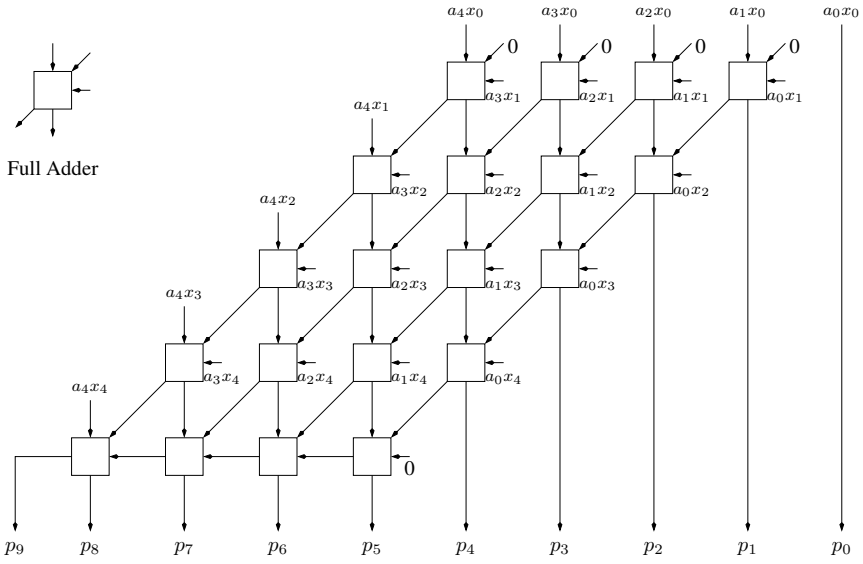


Figure 1: A 5-bit $\times$ 5-bit array multiplier using full adder as building block.

It can be seen that 20 cells or full adders are used. There are 5 cells can be replaced with half adders since one of their inputs is constant zero. A critical path can be easily found with six full adders and two half adders. A summary of the array multiplier complexities can be given as follows.

Note that the time delay of one full adder is two Δ_G and one half adder is one Δ_G.

$$\begin{cases} \text{Space complexity:} & 15\text{FAs} + 5\text{HAs} \\ \text{Time complexity:} & 6\Delta_{\text{FA}} + 2\Delta_{\text{HA}} = 14\Delta_G \end{cases}$$

6.4. Building Large Multiplier Using Small Multipliers

1. Consider to build a $2n \times 2n$ multiplier by using $n \times n$ multipliers. Let both the multiplicand A and the multiplier X be $2n$-bit numbers,

$$\begin{cases} A &= A_H \cdot 2^n + A_L, \\ X &= X_H \cdot 2^n + X_L, \end{cases}$$

   where A_H, A_L, X_H, X_L are n-bit numbers.

2. Then it follows

$$A \cdot X = (A_H 2^n + A_L) \times (X_H 2^n + X_L) = \underbrace{A_H X_H}_{1} 2^{2n} + (\underbrace{A_H X_L}_{2} + \underbrace{A_L X_H}_{3}) 2^n + \underbrace{A_L X_L}_{4}.$$

   Clearly, such a $2n \times 2n$ multiplication can be implemented with 4 $n \times n$ multipliers along with some adders.

3. Note that a $n \times n$ multiplier yields $2n$-bit product. The above equation can be realized with a CSA tree of three rows and width of $4n$-bit.

   - Such a carry-save adder requires one unit of CSA and one unit of CPA.
   - The CSA unit uses $2n$ FAs with a delay of $2\Delta_G$.
   - The CPA unit uses $(2n-1)$ FAs and n HAs with a time delay of $(2(2n-1)+n)\Delta_G = (5n-2)\Delta_G$.
   - The complexities of this carry save adder can be estimated as
     - $C = (4n-1)\text{FAs} + n\text{HAs}$.
     - $T = 5n\Delta_G$.

4. The complexities of the $2n \times 2n$ multiplier can now be estimated as

   - $C = 4C_{n\times n \text{ multiplier}} + (4n-1)\text{FAs} + n\text{HAs}$.

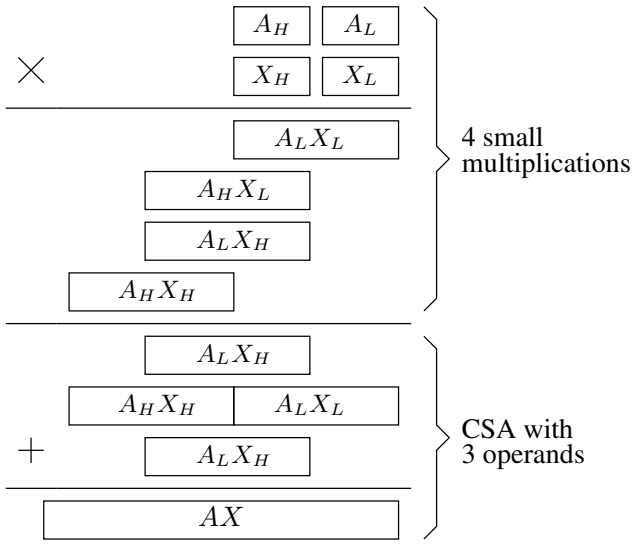


Figure 2: Multiplier built with small multipliers.

- $T = T_{n\times n \text{ multiplier}} + 5n\Delta_G$.

Note that an $n \times n$ multiplier can be built with either methods discussed in the last two sections.