IEICE TRANS. FUNDAMENTALS, VOL.E97–A, NO.2 FEBRUARY 2014

PAPER

Efficient Multiply-by-3 and Divide-by-3 Algorithms and Their Fast Hardware Implementation

Chin-Long WEY[†], *Nonmember*, **Ping-Chang JUI**[††a)], *Student Member*, *and* **Gang-Neng SUNG**[†††], *Nonmember*

SUMMARY This study presents efficient algorithms for performing multiply-by-3 (3N) and divide-by-3 (N/3) operations with the additions and subtractions, respectively. No multiplications and divisions are needed. Full adder (FA) and full subtractor (FS) can be implemented to realize the N3 and N/3 operations, respectively. For fast hardware implementation, this paper introduces two basic cells UCA and UCS for 3N and N/3 operations, respectively. For 3N operation, the UCA-based ripple carry adder (RCA) and carry lookahead adder (CLA) designs are proposed and their speed performances are estimated based on the delay data of standard cell library in TSMC 0.18 μm CMOS process. Results show that the 16-bit UCA-based RCA is about 3 times faster than the conventional FA-based RCA and even 25% faster than the FA-based CLA. The proposed 16-bit and 64-bit UCA-based CLAs are 62% and 36% faster than the conventional FA-based CLAs, respectively. For N/3 operations, ripple borrow subtractor (RBS) is also presented. The 16-bit UCS-based RBS is about 15.5% faster than the 16-bit FS-based RBS.

key words: *ripple carry adder (RCA), carry-lookahead adder (CLA), ripple-borrow subtractor (RBS), multiplier, divider*

1. Introduction

Constant multiplier (divider) performs a multiplication (division) of a data-input with a constant value. They are essential components in various types of arithmetic circuits, such as filters in digital signal processor (DSP) units and they are prevalent in modern VLSI designs. The hardware complexity of digital filter is mainly dominated by the constant multipliers [1].

The multiplication by a fixed-point constant can be done "multiplier-less" using additions/subtractions and shifts only. In such filters the number of adders/subtractors determines the implementation cost. Since the shifters are implemented as hard-wired inter-block connections, they are considered "free" in transposed implementation of an FIR filter; each input is multiplied by several coefficients [2].

Instead of multiplying the processing data by 3, the 3N can be efficiently generated by adding N to its 1-bit left-shifted value 2N. The constant value can be a fractional number, i.e., multiply-by-(1/3). The multiply-by-(1/3) operation is equivalent to the divide-by-3 (N/3) operation. The constant divider can be done "divider-less" using the subtractions.

In addition to the function of constant multiplication/division, the 3N operation can be applied for arithmetic 3N encoding and Radix-8 Booth encodings, while the N/3 operation for arithmetic 3N decoding [3]–[9].

This study presents efficient algorithms for performing 3N and N/3 operations with the additions and subtractions, respectively. No multiplications and divisions are needed.

The addition can be simply done by the combinational circuits, such as full adder (FA), FA-based ripple carry adder (RCA) or carry look-ahead adder (CLA) [8], [9]. Note that the RCA achieves low hardware cost, while the CLA accomplishes high speed performance. Similarly, the subtractions can be performed by full subtractor (FS) or FS-based ripple borrow subtractors (RBS).

In this study, two cells, UCA (Unit Cell for Addition) and UCS (Unit Cell for Subtraction), are introduced for 3N and N/3 operations, respectively. Based on FA and UCA cells, the fast hardware implementations of FA-based and UCA-based RCA and CLA are presented for 3N operation and their speed performances are estimated and compared based on the gate delay data in TSMC 0.18 μm standard cell library. Similarly, based on the FS and UCS cells, the RBS designs and their speed performance will be presented.

Results will show that the proposed 16-bit UCA-based RCA with a delay of 0.7043 ns is about 316% faster than the FA-based RCA with 2.9267 ns; and the 16-bit and 64-bit UCA-based CLAs are about 62% and 36% faster than the FA-based CLAs, respectively. Similarly, the 16-bit UCS-based RBS is about 15.5% faster than the FS-based one.

In the next section, both 3N and N/3 operations and their conversion algorithms are presented. Section 3 describes the conventional and proposed architectures and their speed performances are estimated in Sect. 4. Finally, a brief concluding remark is given in Sect 5.

2. Efficient Algorithm Development

This section first describes the design concept and algorithm development for 3N operation and then presents those for N/3 operation.

2.1 3N and N/3 Operations

Let N = $(a_{n-1}a_{n-2}\ldots a_0)$ be an n-bit code. The 3N can be accomplished by adding A to its 1-bit left-shifted value 2N,

Manuscript received July 3, 2013.
Manuscript revised October 3, 2013.

[†]The author is with the Department of Electrical Engineering, National Chiao Tung University, Hsinchu, Taiwan.
[††]The author is with the Department of Electrical Engineering, National Central University, Jhongli, Taoyuan, Taiwan.
[†††]The author is with the Chip Implementation Center, National Applied Research Laboratories, Hsinchu, Taiwan.
a) E-mail: kwoyei@gmail.com (Corresponding author)

DOI: 10.1587/transfun.E97.A.616

where $2N = (a_{n-1}a_{n-2}\ldots a_0 0)$. Thus, 3N=N+2N as follow,

$$\begin{array}{rccccccc} N & 0 & a_{n-1} & a_{n-2} & \ldots & a_1 & a_0 & \\ +\ 2N & a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_0 & 0 & \\ \hline 3N & s_n & s_{n-1} & s_{n-2} & \ldots & s_1 & s_0 & \end{array} \quad (1A)$$

The computation of (1A) can be achieved using a RCA in which each cell is a full adder (FA). A FA takes two data inputs, a_i and a_{i-1}, and one carry input bit c_i, and produces one carry output bit c_{i+1} and sum bit s_{i+1}, for i = 0~n−1, where $a_{-1} = 0$ and $c_{-1} = 0$, and the functions are

$$\begin{aligned} s_i &= a_i \oplus a_{i-1} \oplus c_{i-1} \\ c_i &= (a_i \oplus a_{i-1})c_{i-1} + a_i a_{i-1} \end{aligned} \quad (2A)$$

Similarly, $N = (b_n b_{n-1} b_{n-2} \ldots b_0)$ and $N/3 = (0a_{n-1}a_{n-2} \ldots a_0)$, then $(2N)/3 = (a_{n-1}a_{n-2}\ldots a_0 0)$. Thus, N/3=N−(2N)/3 as follows,

$$\begin{array}{rccccccc} N & b_n & b_{n-1} & b_{n-2} & \ldots & b_1 & b_0 & \\ -\ 2N/3 & a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_0 & 0 & \\ \hline N/3 & 0 & a_{n-1} & a_{n-2} & \ldots & a_1 & a_0 & \end{array} \quad (1S)$$

The computation of (1S) can be performed by a ripple borrow subtractor (RBS), where each unit contains a one-bit full subtractor (FS). Each 1-bit FS performs the subtraction of $(b_i - a_{i-1})$ by taking two data bits, b_{i+1} and a_i, and the borrow-out bit, B_{i-1}, as its inputs, and producing the difference bit (a_i) and the borrow-in bit (B_i). The logic functions of the 1-bit FS can be expressed as follows,

$$\begin{aligned} a_i &= b_i \oplus a_{i-1} \oplus B_{i-1} \\ B_i &= b_i' B_{i-1} + a_{i-1}B_{i-1} + b_i' a_{i-1} \end{aligned} \quad (2S)$$

2.2 3N and N/3 Algorithms

The basic conversion concept behind this development can be explained from the following example. Table 1 shows the addition of N and 2N for an 8-bit number N=(0001101), or |N|=13, and 2N=(00011010), or |2N|=26. Thus, the sum is 3N=(00100111), or |3N| = |2N| + |N|=39.

Let a_i and a_{3i} denote as the i-th bit values of N and 3N, respectively. We define u[i]=0 if $a_i = a_{3i}$; Otherwise u[i]=1, i.e.,

$$u[i] = a_i \oplus a_{3i} \quad (3)$$

The bottom row of Table 1 shows u=(00101010). If we were able to construct the relationship between a_i and a_{3i} as in (3), then both 3N and N/3 operations can be formulated as the following simple conversions,

$$a_{3i} = a_i \oplus u[i] \quad (4)$$

Table 1 Description of example.

	7	6	5	4	3	2	1	0	
2N	0	0	0	1	1	0	1	0	26
N	0	0	0	0	1	1	0	1	13
3N	0	0	1	0	0	1	1	1	39
u[i]	0	0	1	0	1	0	1	0	

$$a_i = a_{3i} \oplus u[i] \quad (5)$$

Given a_i of N, a_i is derived by (4), i.e., 3N operation. On the other hand, in (5), given a_{3i} for 3N, a_i is generated by (5), i.e., N/3 operation.

The 3N operation can be summarized as the following Algorithm.

```
Algorithm I. (3N Operation)
/* Given N=(0a_{n-1}a_{n-2}…a_0), 3N=(s_n s_{n-1}s_{n-2}…s_0),
a[-1]=c[-1]=0;   /* Initialization  */
For i=0 to n-1  {
    /*   for equation (2A),  c[i]: carry bit; z[i]: sum bit */
    z[i]=XOR(XOR(a[i],a[i-1]),c[i-1]);
    c[i]=OR(AND(XOR(a[i],a[i-1]),c[i-1]),AND(a[i],a[i-1]));
    u[i]=XOR(z[i],a[i]);   /* for equation (3)  */
    s[i]=XOR(a[i],u[i]);    /* for equation (4) */
    }
```

The N/3 operation can be summarized as the following Algorithm.

```
Algorithm II. (N/3 Operation)
/* Given N=(b_n b_{n-1}b_{n-2}…b_0), N/3=(0a_{n-1}a_{n-2}…a_0),
w[-1]=B[-1]=0; /* Initialization  */
For i=0 to n-1  {
    /* for equation (2S), B[i]: borrow; w[i]: difference    */
    w[i]=XOR(XOR(b[i],w[i-1]),B[i-1]);
    B[i]=OR(AND(NOT(b[i]),B[i-1]),AND(w[i-1],B[i-1]),
                          AND(NOT(b[i]),w[i-1]);
    u[i]=XOR(w[i],b[i]);  /* for equation (3)  */
    a[i]=XOR(b[i],u[i]);   /* for equation (5)  */
    }
```

The next step is to generate the finite state machines to construct the relationship u[i] for both 3N operation and N/3 operation.

2.2.1 3N Operation

Based on (2A), the 3N operation can be described by a state machine with 3 states, where State A, $(a_i,c_i) = (0, 0)$, State B, $(a_i, c_i) = (0, 1)$, or $(1, 0)$, and State D, $(a_i, c_i) = (1, 1)$.

Let (a_i, c_i) and (a_{i+1}, c_{i+1}) denote as the Present State (PS) and Next State (NS), respectively. The state input and output are a_{i+1} and $u[i+1] = s_{i+1} \oplus a_{i+1}$, respectively. The state machine is constructed as follows.

State A: $(a_i, c_i)=(0, 0)$

By (2A), $s_{i+1} = a_{i+1} \oplus a_i \oplus c_i = a_{i+1}$ and $c_{i+1} = (a_{i+1} \oplus a_i)c_i + a_{i+1}a_i = 0$; The state output $u[i+1] = s_{i+1} \oplus a_{i+1} = 0$ because $s_{i+1} = a_{i+1}$. This concludes that

If $a_{i+1} = 0$, then $a_{i+1}/u[i+1] = 0/0$ and NS $= (a_{i+1}, c_{i+1}) = (0, 0) = A$;

If $a_{i+1} = 1$, then $a_{i+1}/u[i+1] = 1/0$ and NS $= (a_{i+1}, c_{i+1}) = (1, 0) = B$;

State B: $(a_i,c_i) = (0, 1)$ or $(1, 0)$

By (2A), $s_{i+1} = a_{i+1} \oplus a_i \oplus c_i = a_{i+1}$' and $c_{i+1} = a_{i+1}$; The state output $u[i+1] = s_{i+1} \oplus a_{i+1} = a_{i+1}' \oplus a_{i+1} = 1$. It concludes that

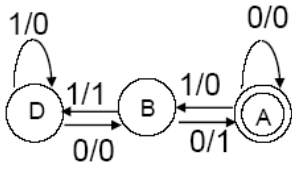

Fig. 1 State diagram of proposed 3N operation.

Table 2 3N operation.

	7	6	5	4	3	2	1	0	
N	0	0	0	0	1	1	0	1	13
Next State	A	A	A	B	D	B	A	B	A
Output	0	0	1	0	1	0	1	0	
3N	0	0	1	0	0	1	1	1	39

If $a_{i+1} = 0$, then $a_{i+1}/u[i+1] = 0/1$ and NS = (a_{i+1}, c_{i+1}) = (0, 0) = A;

If $a_{i+1} = 1$, then $a_{i+1}/u[i+1] = 1/1$ and NS = (a_{i+1},c_{i+1}) = (1, 1) = D;

State D: (a_i,c_i) = (1, 1)

By (2A), $s_{i+1} = a_{i+1} \oplus a_i \oplus c_i = a_{i+1}$ and $c_{i+1} = 1$; The state output $u[i+1] = s_{i+1} \oplus a_{i+1} = a_{i+1} \oplus a_{i+1} = 0$. It concludes that

If $a_{i+1} = 0$, then $a_{i+1}/u[i+1] = 0/0$ and NS = (a_{i+1},c_{i+1}) = (0, 1) = B;

If $a_{i+1} = 1$, then $a_{i+1}/u[i+1] = 1/0$ and NS = (a_{i+1}, c_{i+1}) = (1, 1) = D;

Figure 1 summarizes the state machine. Based on the state diagram in Fig. 1, Table 2 lists the detailed conversion process of the example in Table 1.

Note that State A is the initial state and also the terminal state. If the terminated state is not at State A, it implies that the final result is not 3N.

2.2.2 N/3 Operation

Based on (2S), the N/3 operation can be described by a state machine with 3 states, where State S0, (a_i, B_i) = (0, 0), State S1, (a_i, B_i) = (0, 1), or (1, 0), and State S2, (a_i, B_i) = (1, 1).

Let (a_i, B_i) and (a_{i+1}, B_{i+1}) be the Present State (PS) and Next State (NS), respectively. The state input and output are b_{i+1} and $u[i+1] = a_{3(i+1)} \oplus a_{i+1} = b_{i+1} \oplus a_{i+1}$, respectively. The state machine is constructed as follows,

State S0: (a_i,B_i) = (0, 0)

By (2S), $a_{i+1} = b_{i+1}$ and $B_{i+1} = 0$; The state output $u[i+1] = 0$ because $a_{i+1} = b_{i+1}$. This concludes that

If $b_{i+1} = 0$, then $a_{i+1}/u[i+1] = 0/0$ and NS = (0, 0) = S0;

If $b_{i+1} = 1$, then $a_{i+1}/u[i+1] = 1/0$ and NS = (1, 0) = S1;

State S1: (a_i,B_i) = (0, 1) or (1, 0)

By (2S), $a_{i+1} = b_{i+1}$', and $B_{i+1} = b_{i+1}$'. With the state output $u[i+1] = 1$, the followings are concluded.

If $b_{i+1} = 0$, then $a_{i+1}/u[i+1] = 1/1$ and NS = (1, 1) = S2;

If $b_{i+1} = 1$, then $a_{i+1}/u[i+1] = 0/0$ and NS = (0, 0) = S0;

State S2: (a_i,B_i) = (1, 1)

By (2S), $a_{i+1} = b_{i+1}$, $B_{i+1} = 1$. With the state output $u[i+1] = 0$, the followings are concluded.

If $b_{i+1} = 0$, then $a_{i+1}/u[i+1] = 0/0$ and NS = (0, 1) = S1;

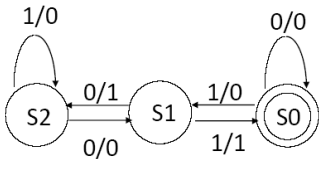

Fig. 2 State diagram of proposed N/3 operation.

Table 3 Conversion process.

	7	6	5	4	3	2	1	0	
B=3N	0	0	1	0	0	1	1	1	39
Next State	S0	S0	S0	S1	S2	S1	S0	S1	S0
Output	0	0	1	0	1	0	1	0	
N	0	0	0	0	1	1	0	1	13

If $b_{i+1} = 1$, then $a_{i+1}/u[i+1] = 1/0$ and NS = (1, 1) = S2;

Therefore, the state machine is shown in Fig. 2.

Based on the state diagram, Table 3 lists the detailed conversion process of the example in Table 1. At bit 0, the present state is the initial state S0, with the input is 1, the next state is moved to S1 with the output u[0] = 0. Thus, with $b_0 = 1$, we have $a_0 = b_0 \oplus u[0] = 1$.

Note that State S0 is the initial state and also the terminal state. If the terminated state is not at State S0, it implies that the final result is not N/3.

3. Fast Hardware Implementation

This section presents the fast hardware implementation for the building blocks for both 3N and N/3 operations.

3.1 Basic Cells

The three states of the state machine in Fig. 1 can be represented by two state variables x and y, where A = (x, y) = (0, 0), B = (0, 1), and D = (1, 1). Figure 3 shows the state table, logic functions, and logic circuit implementation for Fig. 1. Note that u[i] in (4) is the term u_i in Fig. 3(b). Figure 3(d) is the logic circuit implementation for (4).

Similarly, two state variables x and y represent the three states in Fig. 2, S0 = (x, y) = (0, 0), S1 = (0, 1), and S2 = (1, 1). Figure 4 shows the state table, logic functions, and logic circuit implementation. Note that u[i] in (5) is the term u_i in Fig. 4(b). Figure 4(d) is the logic circuit implementation for (5).

Figure 5 shows the basic cells of FA and FS which realizes the equations in (2A) and (2S), respectively.

3.2 Sequential Circuit Implementation

Figure 6 presents the sequential type of hardware implementation for the 3N and N/3 operations which achieve lower hardware cost and high flexibility for converting any bit sizes. However, the operating speed is relatively slow.

In order to develop fast hardware implementation, both

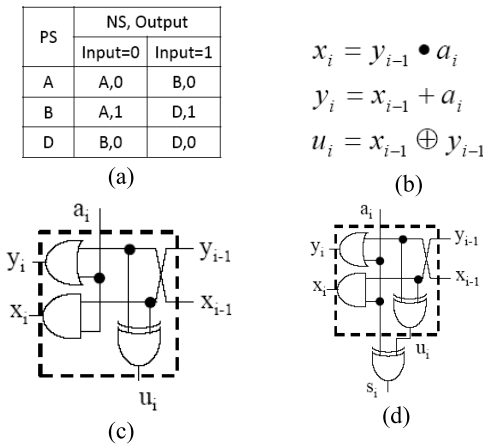

Fig. 3 Unit cell (UCA) for 3N operation: (a) state table; (b) logic functions; (c) logic circuit implementation for Fig. 1; and (d) logic circuit for (4).

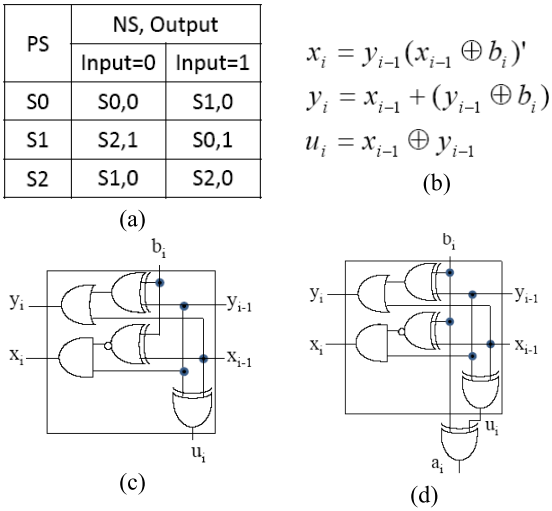

Fig. 4 Unit cell (UCS) for N/3 operation: (a) state table; (b) logic functions; (c) logic circuit implementation for Fig. 2; and (d) logic circuit for (5).

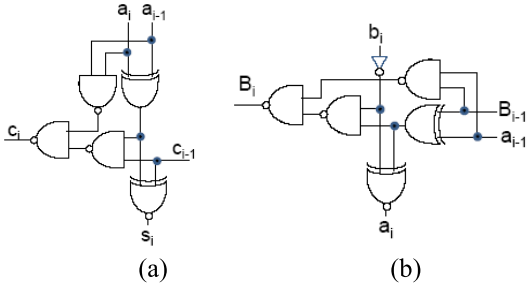

Fig. 5 Basic cells: (a) FA; and (b) FS.

RCA and RBS structures and CLA are presented.

3.3 Carry/Borrow Ripple Structures

Figure 7(a) and Fig. 7(b) present the n-bit FA-based RCA and FS-based RBS, respectively. Similarly, Fig. 7(c) and

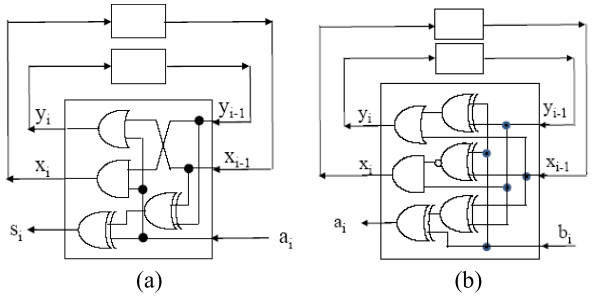

Fig. 6 Sequential circuit types: (a) 3N operation; and (b) N/3 operation.

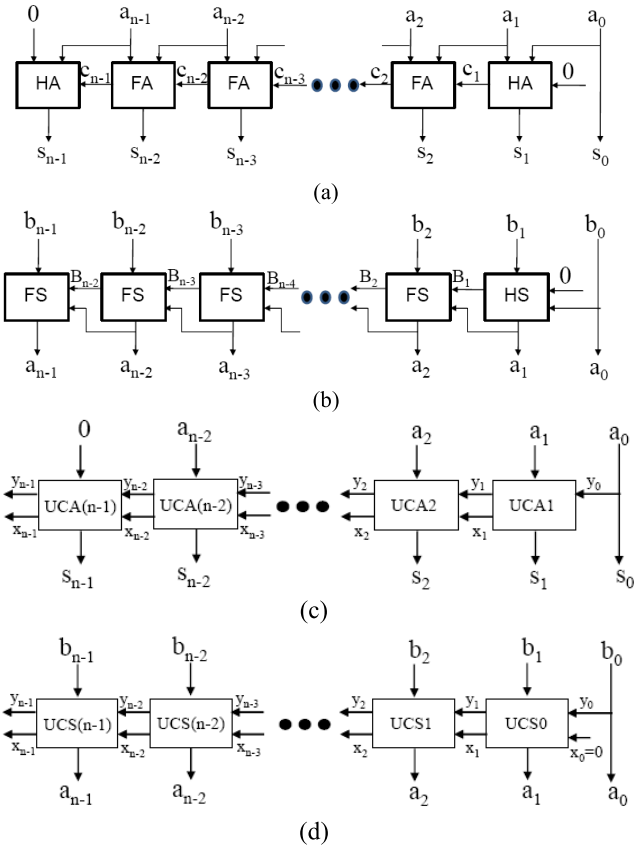

Fig. 7 n-bit ripple structures: (a) RCA(FA); (b) RBS(FS); (c) RCA(UCA); and (d) RBS(UCS).

Fig. 7(b) show the UCA-based RCA and UCS-based RBS, respectively.

With the initial conditions, UCA0 and UCA1 in Fig. 7(c) can be simplified as Fig. 8(a), and the UCA cell in Fig. 3(d) can be realized as shown. Similarly, Fig. 8(b) is the simplified building block for the ripple structure in Fig. 7(d).

3.4 Carry Lookahead Adder (CLA) for 3N Operation

Figure 9(a) shows a 16-bit conventional CLA [8], [9] with the basic building blocks 4-bit BCLA and 4-bit CLA [8], [9]. For 3N operation, the initial carry $c_{-1} = 0$, the 4-bit CLA can be simplified as shown in Fig. 9(b), denoted as CLA-c.

Now, consider the 16-bit UCA-based CLA. Based on

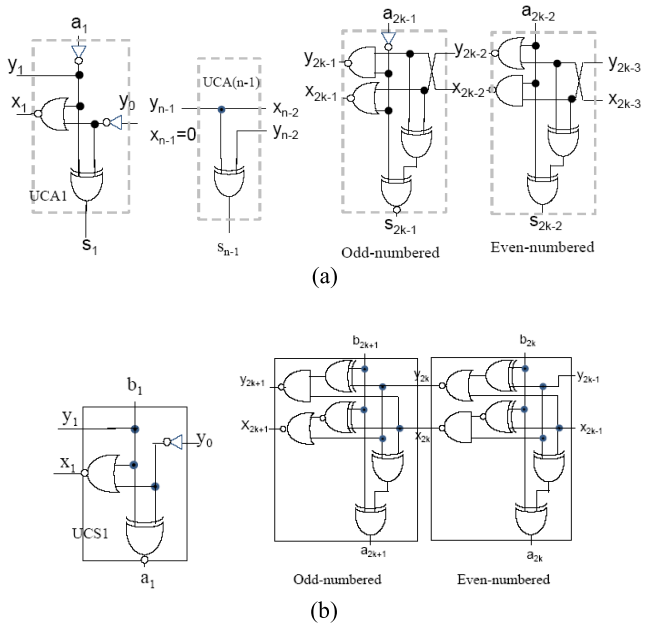

Fig. 8 Basic cells: (a) UCA; and (d) UCS.

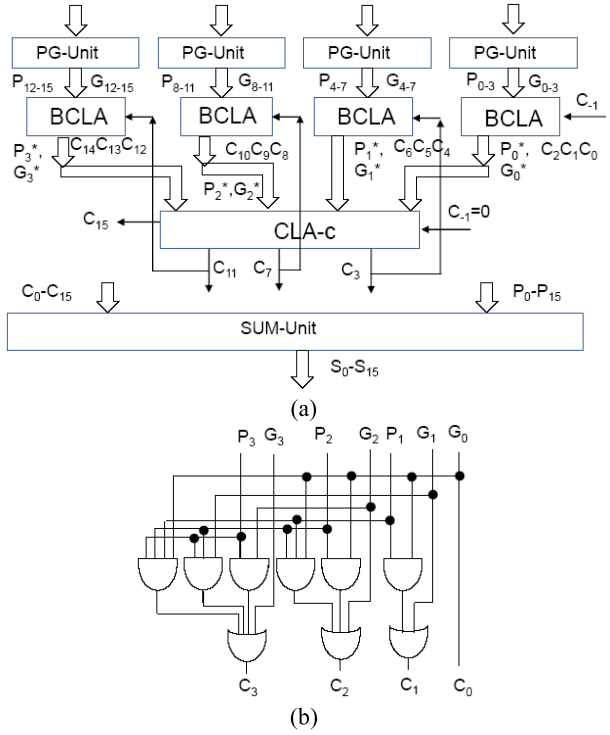

Fig. 9 Conventional CLA: (a) 16-bit CLA; and (b) 4-bit CLA with $C_{-1} = 0$.

the logic functions described in Fig. 3(b), both x_i and y_i, i = 0~3, can be expressed as follows,

$$x_0 = y_{-1}a_0 \qquad y_0 = x_{-1} + a_0$$
$$x_1 = y_0a_1 = x_{-1}a_1 + a_0a_1 \qquad y_1 = x_0 + a_1 = y_{-1}a_0 + a_1$$
$$x_2 = y_1a_2 = y_{-1}a_0a_2 + a_1a_2 \qquad y_2 = x_1 + a_2 = x_{-1}a_1 + a_0a_1 + a_2$$
$$x_3 = y_2a_3 = x_{-1}a_1a_3 + a_0a_1a_3 + a_2a_3 \quad y_3 = x_2 + a_3 = y_{-1}a_0a_2 + a_1a_2 + a_3$$

Therefore, both x_3 and y_3 can be written as

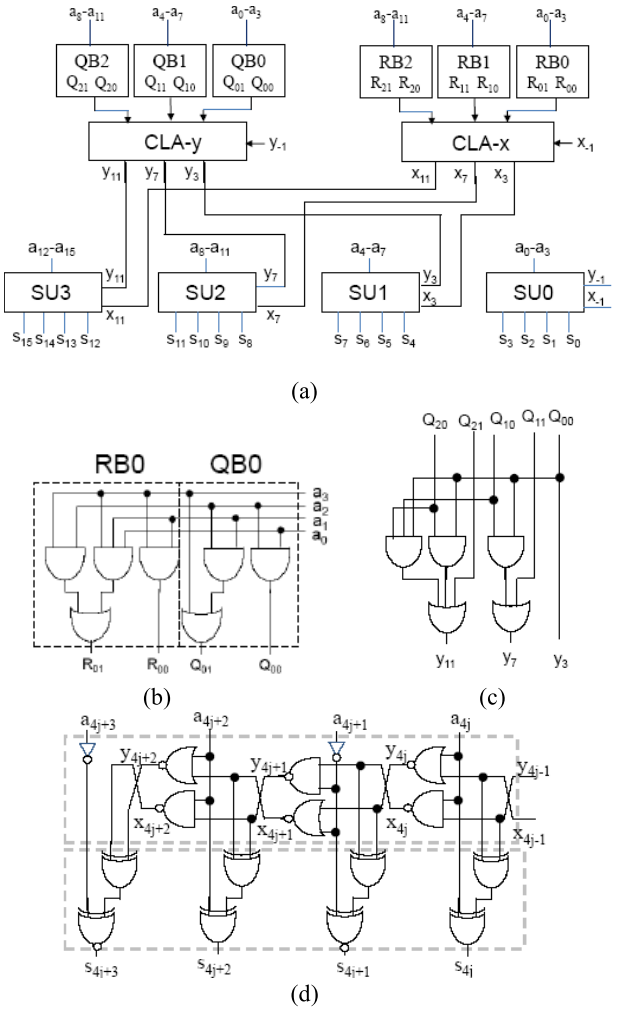

Fig. 10 A 16-bit CLA(UCA): (a) block diagram; (b) RB0 & QB0; (c) inputs to CLA-4; (d) CLA-y with $C_{-1} = 0$; and (d) SUj circuit.

$$x_3 = x_{-1}R_{00} + R_{01} \qquad y_3 = y_{-1}Q_{00} + Q_{01} \tag{6}$$

where

$$R_{00} = a_1a_3,\ R_{01} = a_0a_1a_3 + a_2a_3,\ Q_{00} = a_0a_2,\ Q_{01} = a_1a_2 + a_3 \tag{7}$$

As a result, we have

$$\begin{aligned}
&x_3 = x_{-1}R_{00} + R_{01} \quad y_3 = y_{-1}Q_{00} + Q_{01}\\
&x_7 = x_3R_{10} + R_{11} = x_{-1}R_{00}R_{10} + R_{01}R_{10} + R_{11}\\
&y_7 = y_3Q_{10} + Q_{11} = y_{-1}Q_{00}Q_{10} + Q_{01}Q_{10} + Q_{11}\\
&x_{11} = x_{-1}R_{00}R_{10}R_{20} + R_{01}R_{10}R_{20} + R_{11}R_{20} + R_{21}\\
&y_{11} = y_{-1}Q_{00}Q_{10}Q_{20} + Q_{01}Q_{10}Q_{20} + Q_{11}Q_{20} + Q_{21}\\
&x_{15} = x_{-1}R_{00}R_{10}R_{20}R_{30} + R_{01}R_{10}R_{20}R_{30} + R_{11}R_{10}R_{30}\\
&\qquad + R_{21}R_{30} + R_{31}\\
&y_{15} = y_{-1}Q_{00}Q_{10}Q_{20}Q_{30} + Q_{01}Q_{10}Q_{20}Q_{30} + Q_{11}Q_{20}Q_{30}\\
&\qquad + Q_{21}Q_{30} + Q_{31}
\end{aligned} \tag{8}$$

Figure 10(a) shows a 16-bit UCA-based CLA. The RB0 and QB0, as shown in Fig. 10(b), generate R_{00}, R_{01}, Q_{00}, and Q_{01} in (7). The CLA-x and CLA-y are exactly the same as CLA-c in Fig. 9(b) because the initial carries x_{-1} and y_{-1} are all zeros. However, since both x_{15} and y_{15} can

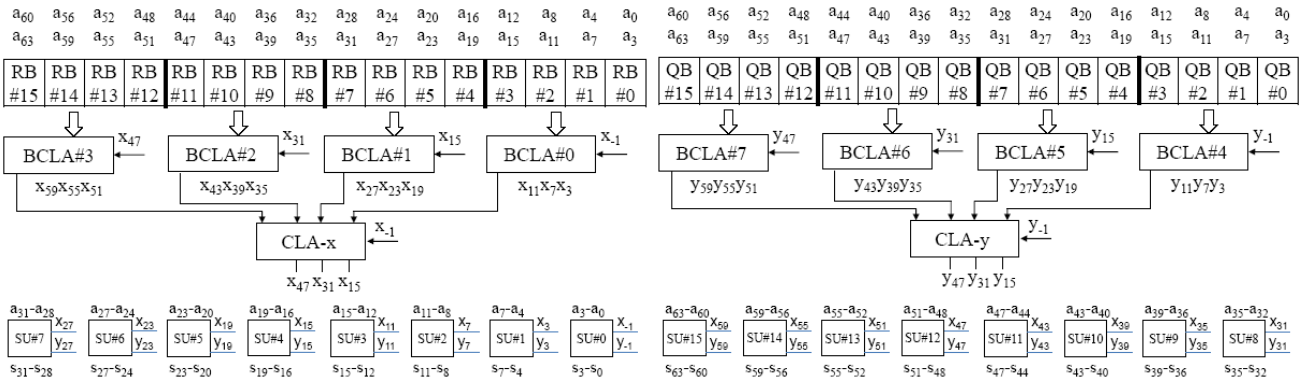

Fig. 11 Block diagram of A 64-bit CLA(UCA).

be generated by Block SU3 with better speed performance and hardware cost, only 3-bit CLA-x and CLA-y, as shown in Fig. 10(c), are used. Figure 10(d) illustrates the circuit of SUj, j = 0∼3.

Further, Fig. 11 shows the block diagram of a 64-bit CLA structure with UCA cells. It comprises of 16 QBs and 16 RBs, 8 4-bit BCLAs, 3-bit CLA-x and CLA-y, and 16 SUj units.

4. Speed Performance Evaluation

This section presents the performance evaluation for the speed performance of the hardware implementation of both 3N and N/3 circuits. The circuit performance is evaluated based on the TSMC 0.18 μm process technology. Table 4 lists the information for the standard cells in TSMC 0.18 μm CMOS process, where the cell height is 5.04 μm.

For simplicity of notation, the FA-based RCA and CLA are denoted as RCA(FA) and CLA(FA), respectively. Similarly, RCA(UCA) and CLA(UCA) are referred to UCA-based RCA and CLA, respectively. RBS(FS) and RBS(UCS) are referred to FS-based and UCS-based RBS, respectively.

The following delay calculation only considers the total delay of the gates included in the critical path, where the path delays and loading effects are not taken into consideration for rough estimation and comparison.

4.1 Basic Cells

The FA cell, as shown in Fig. 5(a), can be realized by 3 NAND2/NOR2 gates, a XOR gate, and a XNOR gate, and its delays are

$$\Delta_{\text{FA(input-to-carry)}} = 2\Delta_{\text{NAND2}} + \Delta_{\text{XOR}} = 0.2095\,\text{ns}$$
$$\Delta_{\text{FA(input-to-sum)}} = \Delta_{\text{XOR}} + \Delta_{\text{NXOR}} = 0.2900\,\text{ns}$$

Similarly, the FS, as shown in Fig. 5(b), can be realized by 3 NAND2 gates, a XOR gate, and a XNOR gate, and one inverter. Its delays are exactly the same as those of FA.

The UCA cell in Fig. 3(d) and Fig. 8(a) can be realized by one NAND2 gate, one NOR2 and two XOR gates. Since NOR2 has longer delay than NAND2, the delays of UCA are

$$\Delta_{\text{UCA(input-to-carry)}} = \Delta_{\text{NOR2}} = 0.0426\,\text{ns}$$
$$\Delta_{\text{UCA(input-to-sum)}} = \Delta_{\text{XOR}} + \Delta_{\text{NXOR}} = 0.2900\,\text{ns}$$

By Fig. 4(d) and Fig. 8(b), the delay of an adjacent pair of an odd-numbered UCS cell and an even-numbered UCS cell includes the delays of two XNOR gate, one NAND2 gate, and one NOR2 gate. Thus, the average delay for the UCS cell from input-to-borrow is ($\Delta_{\text{NAND2}} + \Delta_{\text{XNOR}} + \Delta_{\text{NAND2}} + \Delta_{\text{XNOR}}$)/2; and the delay for UCS cell from input-to-sum is the delays of two XOR gates, by Table 4, the delays are

$$\Delta_{\text{UCS(input-to-borrow)}} = 0.1828\,\text{ns}$$
$$\Delta_{\text{UCS(input-to-sum)}} = 0.2894\,\text{ns}$$

These four basic cells have almost the same delay from the inputs to the sum outputs. However, the UCA cell is about 4.92 times faster than the FA cell for their carry delays. If the *improvement ratio* of A over B is defined as (A−B)/B, the improvement ratio of UCA cell over the FA cell is 3.92. Similarly, the improvement ratio of UCS cell over the FS cell is 14.6%.

Table 4 Cell delay data.

	Delay (ns)		Delay (ns)
NAND2	0.0324	NOR2	0.0426
NAND3	0.0453	NOR3	0.0591
AND2	0.0841	OR2	0.0656
XOR	0.1447	XNOR	0.1453
INV	0.0261		

4.2 Ripple Structures

Consider the delays of 16-bit RCA(FA) and RCA(UCA), as shown in Fig. 7(a) and Fig. 7(c). The half-adder (HA) for the least significant bit is an AND2 gate which can be realized by an NAND2 gate and an inverter for shorter delay, i.e., $\Delta_{\text{AND2}} = \Delta_{\text{NAND2}} + \Delta_{\text{INV}} = 0.0585$ ns. Thus, their delays are

$$\Delta_{\text{RCA16(FA)}} = \Delta_{\text{NAND2}} + \Delta_{\text{INV}} + 13\Delta_{\text{FA}} + \Delta_{\text{XOR}} = 2.9267\,\text{ns}$$
$$\Delta_{\text{RCA16(UCA)}} = \Delta_{\text{INV}} + 12\Delta_{\text{UCA}} + 2\Delta_{\text{XOR}} = 0.7043\,\text{ns}$$

The proposed 16-bit RCA(UCA) is 4.16 times faster than the RCA(FA). The improvement ratio is 316%.

In Fig. 8(b), the delay of UCS1 with $x_0 = 0$ is (Δ_{INV} + Δ_{NOR2}), or 0.0687 ns. The delays of 16-bit RBS(FS) and RBS(UCS), as shown in Fig. 7(b) and Fig. 7(d), are

$$\Delta_{RBS16(FS)} = \Delta_{XOR} + 13\Delta_{FS} + \Delta_{XOR} + \Delta_{XNOR} = 3.1582\,\text{ns}$$
$$\Delta_{RBS16(UCS)} = (\Delta_{INV} + \Delta_{NOR2}) + 13\Delta_{UCS} + 2\Delta_{XOR} = 2.7345\,\text{ns}$$

Thus, the improvement ratio of the proposed 16-bit RBS(UCS) over RBS(FS) is 15.5% for N/3 operation.

4.3 Carry Lookahead Structures for 3N Operation

The delay of the conventional 16-bit CLA(FA) includes the delays of P-G unit, 3 levels of CLA/BCLA, and sum unit. The NAND4 in BCLA can be realized with two NAND2 gates connected with an OR gate, i.e., $\Delta_{NAND4} = \Delta_{OR2} + \Delta_{NAND2}$. Thus, the delay of 16-bit CLA(FA) can be expressed as

$$\Delta_{CLA16(FA)} = \Delta_{PG} + 3\Delta_{CLA\text{-}c/BCLA} + \Delta_{sum} = 0.8774\,\text{ns}$$

where

$$\Delta_{PG} = \Delta_{XOR} = 0.1447\,\text{ns}$$
$$\Delta_{BCLA} = 2\Delta_{NAND4} = 2(\Delta_{OR2} + \Delta_{NAND2}) = 0.196\,\text{ns}$$
$$\Delta_{CLA\text{-}c} = \Delta_{BCLA} = 0.196\,\text{ns}$$
$$\Delta_{sum} = \Delta_{XOR} = 0.1447\,\text{ns}$$

The 16-bit CLA(UCA) in Fig. 10(a) takes only one level of 3-bit CLA-x, and its delay can be estimated as follows,

$$\Delta_{CLA16(UCA)} = \Delta_{RQ} + \Delta_{CLA\text{-}x} + \Delta_{su} = 0.5426\,\text{ns}$$

where

$$\Delta_{RQ} = 2\Delta_{NAND2} = 0.0648\,\text{ns}$$
$$\Delta_{CLA\text{-}x} = 2\Delta_{NAND3} = 0.0906\,\text{ns}$$
$$\Delta_{su} = 3\Delta_{NAND2} + \Delta_{XOR} + \Delta_{XNOR} = 0.3872\,\text{ns}$$

The improvement ratio is 62%, i.e., the proposed 16-bit CLA(UCA) is 62% faster than the conventional CLA(FA) for 3N operation. Similarly, the 64-bit CLA(FA) takes 5-level of CLA/BCLA and its delay can be expressed as

$$\Delta_{CLA64(FA)} = \Delta_{PG} + 5\Delta_{CLA/BCLA} + \Delta_{sum} = 1.2694\,\text{ns}$$

The 64-bit CLA(UCA) in Fig. 11 takes only 3 levels of BCLA/ CLA and its delay is

$$\Delta_{CLA64(UCA)} = \Delta_{RQ} + \Delta_{BCLA} + \Delta_{CLA\text{-}x} + \Delta_{BCLA} + \Delta_{su} = 0.9346\,\text{ns}$$

The improvement ratio is 36%, i.e., the proposed 64-bit CLA(UCA) is 36% faster than the conventional CLA(FA) for 3N operation. Table 5 summarizes the speed performance comparison of various structures discussed above for 3N and N/3 operations. Results also show that the proposed 16-bit RCA(UCA) with a delay 0.7343 ns is about 25% faster than the conventional 16-bit CLA(FA) with a delay of 0.8774 ns.

Table 5 Speed performance comparison.

3N Operation	FA	UCA	Improvement Ratio
Cell	0.2095 ns	0.0426 ns	392%
RCA(16)	2.9267 ns	0.7043 ns	316%
CLA(16)	0.8774 ns	0.5426 ns	62%
CLA(64)	1.2694 ns	0.9346 ns	36%
N/3 Operation	**FS**	**UCS**	**Improvement Ratio**
Cell	0.2095 ns	0.1828 ns	14.6%
RBS(16)	3.1582 ns	2.7345 ns	15.5%

(Improvement Ratio of A over B=(A-B)/B.)

5. Conclusion

This paper presents simple, yet efficient, algorithms for both 3N and N/3 operations and their fast hardware implementation. A simple relationship between the input data and the output data for both 3N and N/3 operations was derived to simplify the circuit design. This study has shown that, for 3N operations, the improvement ratio of the proposed 16-bit UCA-based RCA over the conventional 16-bit FA-based RCA is about 316%. The proposed 16-bit and 64-bit UCA-based CLA's are approximately 62% and 36% faster than the conventional FA-based CLA structures, respectively. It should be mentioned that the proposed 16-bit UCA-based RCA with a delay of 0.7043 ns is about 25% faster than the conventional 16-bit FA-based CLA, due to the simplicity of the UCA cell. On the other hand, the proposed 16-bit UCS-based RBS with a delay of 2.7345 ns is approximately 15.5% faster than the conventional 16-bit FS-based RBS with 3.1582 ns.

References

[1] P.R. Cappello and K. Steiglitz, "Some complexity issues in digital signal processing," IEEE Trans. Acoust. Speech Signal Process., vol.ASSP-32, no.5, pp.1037–1041, 1984.
[2] D. Pradhan, Fault-tolerant Computer System Design, Prentice Hall, 1996.
[3] B. Johnson, Design and Analysis of Fault-Tolerant Digital Systems, Addison-Wesley, 1989.
[4] S. Wang, Z. Wen, and L. Yu, "High-performance fault-tolerant CORDIC processor for space applications," Proc. International Symp. on Systems and Control in Aerospace and Astronautics, pp.360–363, 2006.
[5] P.-C. Jui, G.-N. Sung, and C.L. Wey, "Efficient algorithm and hardware implementation of 3N for arithmetic and for radix-8 encodings," Proc. IEEE Midwest Symp. on Circuits and Systems, pp.418–421, Boise, Idaho, Aug. 2012.
[6] N. Gaitanis, "Totally self-checking checker for 3N arithmetic codes," Electron. Lett., vol.19, pp.685–686, 1983.
[7] D. Knuth, The Art of Computer Programming: Seminumerical Algorithms, vol.2, Addison-Wesley, 1969.
[8] I. Koren, Computer Arithmetic Algorithms, Prentice-Hall, New Jersey, 1993.
[9] K. Huang, Computer Arithmetic: Principles, Architecture, and Design, John Wiley & Sons, 1979.

Chin-Long Wey received his Ph.D. degree in Electrical Engineering from Texas Tech University, Lubbock, Texas, in 1983. He is currently the University Distinguished Professor of Electrical Engineering at National Chiao Tung University, Hsinchu, Taiwan. He was the Director General of the National Chip Implementation Center (CIC), Hsinchu, Taiwan, in 2007–2010, and the Dean of College of Electrical Engineering and Computer Science at National Central University (NCU) in 2003–2006. He came to NCU from Michigan State University where he was a tenured full professor of Electrical and Computer Engineering Department from 1983 to 2003 for 20 years. His research interests include design, testing, and fault diagnosis of analog/mixed-signal VLSI circuits and systems; digital circuit design automation; and Battery management systems. He has published more than 250 technical journal and conference papers in these areas. Dr. Wey is a Fellow of the IEEE.

Ping-Chang Jui received her M.S. degree in E.E. of National Cheng-Kung University, Tainan, Taiwan. She is currently working toward her Ph.D. degree in E.E. at National Central University, Jhongli, Taiwan. Her research interests include communication systems and infrastructures; and Technology management/strategic development.

Gang-Neng Sung was born in Taiwan in 1981. He received the B.S. degree in Department of Computer and Communication Engineering in National Kaohsiung First University of Science and Technology in 2004, and the M.S. degree in Department of Electrical Engineering in National Sun Yat-Sen University (NSYSU) in 2006. In 2010, he received the Ph.D. degree in the Department of Electrical Engineering at NSYSU. He is currently working in the National Chip Implementation Center (CIC), National Applied Research Laboratories (NARL), Hsinchu, Taiwan. His recent research interests include VLSI mixed-signal circuit design, low power design and car electronics.