J Sign Process Syst (2016) 82:41–53
DOI 10.1007/s11265-015-0978-4

Multiplication of a Constant ($2^k \pm 1$) and Its Fast Hardware Implementation

Pin-Chang Jui · Chin-Long Wey · Muh-Tian Shiue

Received: 9 November 2014 / Revised: 25 January 2015 / Accepted: 2 February 2015 / Published online: 14 February 2015

Abstract Constant multiplier performs a multiplication of a data-input with a constant value. Constant multipliers 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. This study presents efficient algorithms and their fast hardware implementation for performing multiplying-by-($2^k \pm 1$), or ($2^k \pm 1$)N, operation with additions. No multiplications are needed. The value of ($2^k \pm 1$)N can be computed by adding (±N) to its k-bits left-shifted value 2^kN. The additions can be performed by the full-adder-based (FA-based) ripple carry adder (RCA) for simple architecture. This paper presents the unit cells for additions (UCAs). Results show that the UCA-based RCA achieves 34 % faster than the FA-based RCA. Further, in order to improve the speed performance with lower hardware cost, this paper also presents a simple and modular hybrid adder with the proposed UCA concept, where the hybrid adder takes the lower-bit carry lookahead adder (CLA) as a module and many of the CLA modules are serially connected in a fashion similar to the RCA. Results show that the proposed hybrid adder achieved speed performance improvement while maintaining its modular and regular structure.

Keywords Constant multiplier · Ripple Carry Adder (RCA) · Carry-Lookahead Adder (CLA) · Hybrid Adder (HyA) · Booth algorithm

P.-C. Jui (✉) · M.-T. Shiue
Department of Electrical Engineering, National Central University, Jhongli, Taiwan
e-mail: kwoyei@gmail.com

C.-L. Wey
Department of Electrical and Computer Engineering, National Chiao Tung Univesity, Hsinchu, Taiwan

1 Introduction

Constant multiplier performs a multiplication of a data-input with a constant value. Constant multipliers are essential components in various types of arithmetic circuits, such as filters in digital signal processor (DSP) units, dominate the hardware complexity of digital filters [1]. In addition, they are prevalent in modern VLSI designs.

The multiplication by a fixed-point constant can be done "multiplier-less" using additions and shifts only. In such filters the number of adders 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 [1–3]. Constant multiplier design has been investigated for several decades. However, the emphasis was placed on minimizing the number of additions required to achieve the multiplication of a given constant [4].

In this study, the emphasis is placed on performing the constant multiplication with a faster adder in only one addition operation. This paper targets the development of the multiplication of a constant ($2^k \pm 1$). The value of (2^k+1)N can be computed by adding N to its k-bit left-shifted value 2^kN. On the other hand, The value of (2^k-1)N can be computed by subtracting N from its k-bits left-shifted value 2^kN, or adding (−N) to 2^kN. The additions can be performed by a simple ripple carry adder (RCA), or a higher speed carry-lookahead adder (CLA).

The unit cells for additions (UCAs) are introduced in this study to construct the UCA-based RCA. Results will show that the UCA-based RCA achieves approximately 34 % faster than Full-adder-(FA)-based RCAs. In order to further improve the speed performance, a simple and modular hybrid adder is also presented, where reasonably smaller bit size of CLA is used as a module and many modules are serially connected in a fashion similar to the RCA.

In the next section, the conventional multiplication of a constant $(2^k \pm 1)$ using FA-based RCA is discussed. Section III presents the proposed UCA-based RCA structures for $(2^k \pm 1)N$ operation. Section IV describes a hybrid adder for the constant multiplication. Finally, a brief concluding remark is given in Section V.

2 FA-based RCAs for $(2^k \pm 1)N$ Operations

Let $N=(a_{n-1}a_{n-2}\ldots a_0)$ be an n-bit number. For the $(2^k+1)N$ operation, there exists a number m such that $n=m\times k$ (if $n< m\times k$, sign extension is applied). Thus, N can be expressed as $(A_{m-1}A_{m-2}\ldots A_0)$, where $A_i=(a_{(i+1)k-1}\ldots a_{ik+1}a_{ik})$, $i=0, 1, \ldots, m-1$, and the $(2^k+1)N$ operation can be performed by adding N to its k-bits left-shifted value 2^kN, i.e., $(1+2^k)N=N+2^kN$, where $2^kN=(A_{m-1}A_{m-2}\ldots A_00)$ and $0=(0..00)$.

$$\begin{array}{lccccccccc} N & 0 & A_{m-1} & \ldots & A_{i+1} & A_i & A_{i-1} & \ldots & A_1 & A_0 \\ \underline{+2^kN} & A_{m-1} & A_{m-2} & \ldots & A_i & A_{i-1} & A_{i-2} & \ldots & A_0 & 0 \\ (1+2^k)N & S_m & S_{m-1} & \ldots & S_{i+1} & S_i & S_{i-1} & \ldots & S_1 & S_0 \end{array} \quad (1)$$

The 3N operation is performed by 3N=N+2 N,

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

and the 9N=N+8 N operation with $k=3$ is operated as

$$\begin{array}{lccccccccc} N & 0 & 0 & 0 & a_{3m-1} & \ldots & a_{3i+2}a_{3i+1}a_{3i} & \ldots & a_3a_2\ a_1a_0 \\ \underline{+\ 8N} & a_{3m-1} & a_{3m-2} & a_{3m-3} & a_{3m-4} & \ldots & a_{3i-1}\ a_{3i-2}\ a_{3i-3} & \ldots & a_0\ 00\ 0 \\ 9\ N & s_{3m+2} & s_{3m+1} & s_{3m} & s_{3m-1} & \ldots & s_{3i+2}s_{3i+1}s_{3i} & \ldots & s_3s_2\ s_1s_0 \end{array} \quad (3)$$

On the other hand, $(2^k-1)N=2^kN+(-N)=2^kN+N^*+1$, where N* is the bit-complement of N, i.e., $N^*=(A_{m-1}'A_{m-2}'\ldots A_0')$ and $A_i'=(a_{(i+1)k-1}'\ldots a_{ik+1}'a_{ik}')$, $i=0,1,\ldots, m-1$. Thus, $(2^k-1)N$ operation can be performed by adding the k-bits left-shifted value 2^kN to N* with an initial carry of 1. For example, the 7 N operation is performed as follows,

$$\begin{array}{lccccccccc} N^* & 1 & 1 & 1 & a_{3m-1}' & \ldots & a_{3i+2}'a_{3i+1}'a_{3i}' & \ldots & a_3'a_2'a_1'a_0' \\ \underline{+\ 8N} & a_{3m-1} & a_{3m-2} & a_{3m-3} & a_{3m-4} & \ldots & a_{3i-1}\ a_{3i-2}\ a_{3i-3} & \ldots & a_0\ 00\ 0 \\ 7\ N & s_{3m+2} & s_{3m+1} & s_{3m} & s_{3m-1} & \ldots & s_{3i+2}\ s_{3i+1}\ s_{3i} & \ldots & s_3s_2\ s_1s_0 \end{array} \quad (4)$$

Figure 1a shows a full-adder (FA) cell [5, 6]. A FA cell takes two data inputs, a_{i-1} and a_i, and a carry input bit c_{i-1}, and produces a carry output bit c_i and a sum bit s_i, for $i=0\sim n$, where $a_{-1}=0$ and $c_{-1}=0$. The logic functions are

$$\begin{aligned} s_i &= a_i \oplus a_{i-1} \oplus c_{i-1} \\ c_i &= (a_i \oplus a_{i-1})c_{i-1} + a_i a_{i-1} \end{aligned} \quad (5)$$

The point-to-point delays of the FA and HA (Half-adder) cells are

$$\begin{aligned} \Delta_{cc(FA)} &= \Delta_{carry-in-to-carry-out(FA)} = 2\Delta_{NAND2} \\ \Delta_{cs(FA)} &= \Delta_{carry-to-sum(FA)} = \Delta_{XOR} \\ \Delta_{ic(FA)} &= \Delta_{input-to-carry(FA)} = \Delta_{XOR} + 2\Delta_{NAND2} \\ \Delta_{cc(HA)} &= \Delta_{ic(HA)} = \Delta_{NOR2}\ \Delta_{cs(HA)} = \Delta_{XOR} \end{aligned} \quad (6)$$

Figure 1b shows an (n+1)-bit FA-based RCA for 3 N operation. The RCA is comprised of (n-2) FAs and 2 HAs connected in series. The critical path includes the input-to- carry of the right-most HA ($\Delta_{ic(HA)}$), the carry-to-carry ($\Delta_{cc(FA)}$) of (n-2) FAs, and the carry-to-sum ($\Delta_{cs(HA)}$) of the left-most HA, i.e.,

$$\begin{aligned} \Delta_{RCA(FA)(3N)} &= \Delta_{ic(HA)} + (n-2)\Delta_{cc(FA)} + \Delta_{cs(HA)} \\ &= \Delta_{NOR2} + 2(n-2)\Delta_{NAND2} + \Delta_{XOR} \end{aligned} \quad (7)$$

In general, the critical path delay of the (km)-bits FA-based RCA for $(2^k+1)N$ operation can be expressed as

$$\begin{aligned} \Delta_{RCA(FA)((2^{k+1})N)} &= \Delta_{ic(HA)} + (km-k-1)\Delta_{cc(FA)} + (k-1)\Delta_{cc(HA)} + \Delta_{cs(HA)} \\ &= k\Delta_{NOR2} + 2(km-k-1)\Delta_{NAND2} + \Delta_{XOR} \end{aligned} \quad (8)$$

Similarly, the critical path delay of the (km)-bits FA-based RCA for $(2^k-1)N$ operation can be expressed as

$$\begin{aligned} \Delta_{RCA(FA)((2^{k-1})N)} &= \Delta_{ic(HA)^*} + (km-k-1)\Delta_{cc(FA)} + (k-1)\Delta_{cc(HA)^*} + \Delta_{cs(HA)^*} \\ &= \Delta_{inv} + [2(km-1)-k]\Delta_{NAND2} + \Delta_{XNOR} \end{aligned} \quad (9)$$

The term, (HA)* in (9), indicate a HA resulted from a FA with an input "1", where $\Delta_{ic(HA)^*}=\Delta_{inv}+\Delta_{NAND2}$, $\Delta_{cc(HA)^*}=\Delta_{NAND2}$, *and* $\Delta_{cs(HA)^*}=\Delta_{XNOR}$.

By (8) and (9), with $k=3$, the delays of the RCA in Fig. 1c and d for 9 N and 7 N operations respectively are

$$\Delta_{RCA(FA)(9N)} = 3\Delta_{NOR2} + 2(3m-4)_{NAND2} + \Delta_{XOR} \quad (10)$$

$$\Delta_{RCA(FA)(7N)} = \Delta_{inv} + (6m-5)\Delta_{NAND2} + \Delta_{XNOR} \quad (11)$$

Finally, Fig. 1e illustrates the (3 m)-bits FA-based RCA with dual mode for both 7 N and 9 N operations, where mode=0 for 9 N operation and mode=1 for 7 N operation.

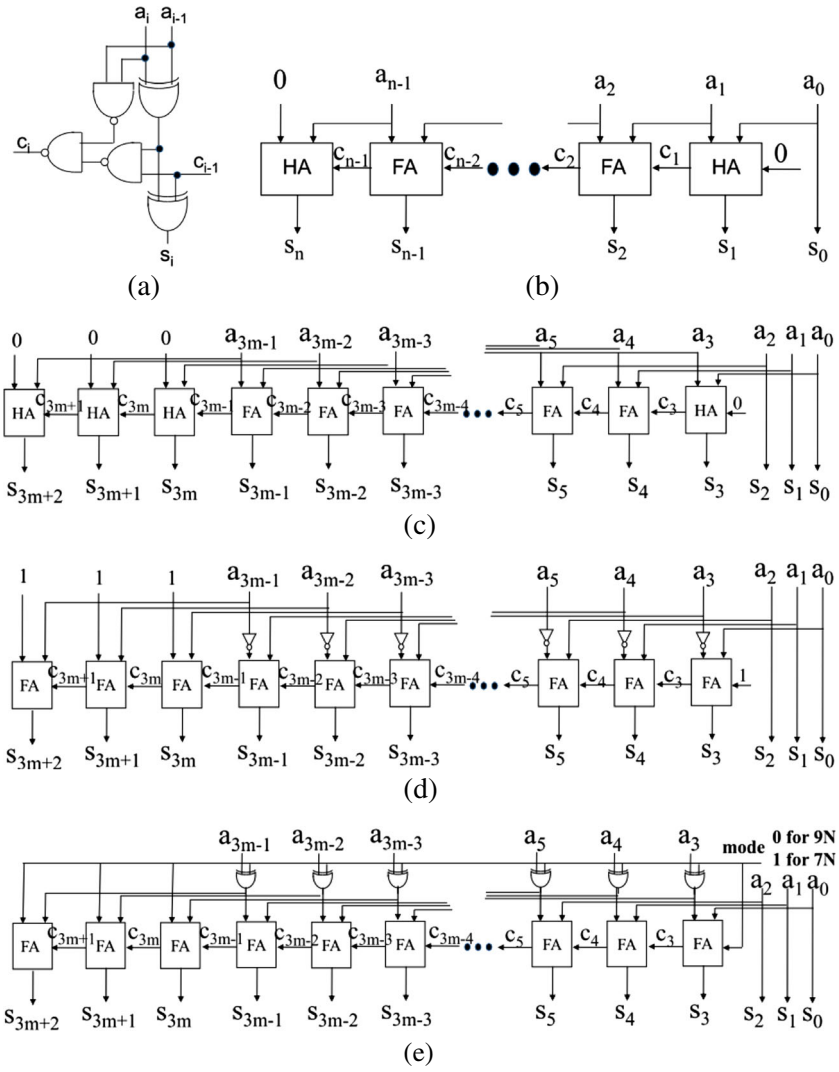


Figure 1 FA-based RCA: **a** FA cell; **b** for 3N; **c** for 9N; **d** for 7N; and **e** for dual mode – 7N and 9N.

Similarly, one can easily realize a (km)-bits FA-based RCA for $(2^k \pm 1)N$ operations.

3 UCA-based RCAs for $(2^k \pm 1)N$ Operations

This section presents the UCA-based RCAs for $(2^k+1)N$ operation. The implementation concept is readily applied for $(2^k-1)N$ operations.

3.1 UCA-based RCA for 3 N Operation

Consider the 3 N operation, as illustrated in (2). By Shannon expansion theorem, the carry and sum functions in (5) can be re-written as

$$\begin{aligned} c_i &= a_i'(a_{i-1}c_{i-1}) + a_i(a_{i-1} + c_{i-1}) \\ &= a_i'W0_{i-1} + a_iW1_{i-1} \end{aligned} \tag{12}$$

$$s_i = a_i \oplus u_i, \text{ where } u_i = a_{i-1} \oplus c_{i-1} \tag{13}$$

where $W0_{i-1}=a_{i-1}c_{i-1}$ and $W1_{i-1}=a_{i-1}+c_{i-1}$. To construct the carry propagation paths, we consider both $W0_i$ and $W1_i$, in the next stage. By (12), we can easily derive

$$\begin{aligned} W0_i &= a_ic_i = a_i(a_i'W0_{i-1} + a_iW1_{i-1}) = a_iW1_{i-1}; \text{ and} \\ W1_i &= a_i + c_i = a_i + a_i'W0_{i-1} + a_iW1_{i-1} = a_i + W0_{i-1} \end{aligned} \tag{14}$$

By (13), the functions u_i can be expressed as

$$\begin{aligned} u_i &= [a_{i-1}c_{i-1} + a_{i-1}'c_{i-1}']' = [W0_{i-1} + W1_{i-1}']' \\ &= W0_{i-1}'W1_{i-1} \end{aligned} \tag{15}$$

Figure 2a shows a UCA cell for 3 N operation. The cell is comprised of three blocks: Carry Propagation Path (CPP) block, Function u_i Generation (FUG) block, and Sum Generation (SMG) block. The critical path of an n-bit UCA-based RCA, as illustrated in Fig. 2b, includes an NAND gate in cell#1, (n-3) UCAs in cell#2 to cell#(n-2), and the inverter, NOR2, and XOR gates in cell#(n-1), where $\Delta_{UCA}=\Delta_{NOR2}$, i.e., the delay is

$$\Delta_{RCA(UCA)(3N)} = \Delta_{NAND2} + (n-2)\Delta_{NOR2} + \Delta_{inv} + \Delta_{XOR} \quad (16)$$

3.2 UCA-based RCA for 5 N Operation

Consider the 5 N operation, as shown in (1) with k=2, where $A_i=(a_{2i+1},a_{2i})$, i=0,1,…,m-1, and n=2 m. (Note that a zero is added as the most significant bit if n is not an even number.) Two additions, $(a_{2i+1}+a_{2i-1}+c_{2i})$ and $(a_{2i}+a_{2i-2}+c_{2i-1})$, are performed. By (5),

$$\begin{aligned} c_{2i} &= a_{2i}'(a_{2i-2}c_{2i-1}) + a_{2i}(a_{2i-2} + c_{2i-1});\\ s_{2i} &= a_{2i}\oplus u_{2i},\ \text{where } u_{2i} = a_{2i-2}\oplus c_{2i-1};\\ c_{2i+1} &= a_{2i+1}'(a_{2i-1}c_{2i}) + a_{2i+1}(a_{2i-1} + c_{2i});\\ s_{2i+1} &= a_{2i+1}\oplus u_{2i+1},\ \text{where } u_{2i+1} = a_{2i-1}\oplus c_{2i}; \end{aligned} \quad (17)$$

By (17), plugging c_{2i} to c_{2i+1}, we have

$$\begin{aligned} c_{2i+1} = {} & a_{2i+1}' \ a_{2i-1}[a_{2i}'(a_{2i-2}c_{2i-1}) + a_{2i}(a_{2i-2} + c_{2i-1})] \ + \\ & a_{2i+1}(a_{2i-1} + a_{2i}'(a_{2i-2}c_{2i-1}) + a_{2i}(a_{2i-2} + c_{2i-1}));\\ = {} & a_{2i+1}'a_{2i}'(a_{2i-1}a_{2i-2}c_{2i-1}) + a_{2i+1}'a_{2i}[a_{2i-1}(a_{2i-2} + c_{2i-1})] + \\ & a_{2i+1}a_{2i}'[a_{2i-1} + a_{2i-2}c_{2i-1}] + a_{2i+1}a_{2i}[a_{2i-1} + a_{2i-2} + c_{2i-1}]\\ = {} & a_{2i+1}'a_{2i}'W00_{2i-1} + a_{2i+1}'a_{2i}W01_{2i-1} + \\ & a_{2i+1}a_{2i}'W10_{2i-1} + a_{2i+1}a_{2i}W11_{2i-1} \end{aligned} \quad (18)$$

where

$$\begin{aligned} W00_{2i-1} &= a_{2i-1}a_{2i-2}c_{2i-1}; \quad W01_{2i-1} = a_{2i-1}(a_{2i-2} + c_{2i-1});\\ W10_{2i-1} &= a_{2i-1} + a_{2i-2}c_{2i-1}; \quad W11_{2i-1} = a_{2i-1} + a_{2i-2} + c_{2i-1}; \end{aligned}$$

The 2-bits UCA cell for 5 N, referred to as a UCA2 cell can be derived as in the following property.

Property 1 The logic functions of the CPP block are expressed as

$$\begin{aligned} W00_{2i+1} &= a_{2i+1}\cdot a_{2i}\cdot W11_{2i-1}\\ W01_{2i+1} &= a_{2i+1}\cdot(a_{2i} + W10_{2i-1})\\ W10_{2i+1} &= a_{2i+1} + (a_{2i}\cdot W01_{2i-1})\\ W11_{2i+1} &= a_{2i+1} + a_{2i} + W00_{2i-1} \end{aligned} \quad (18)$$

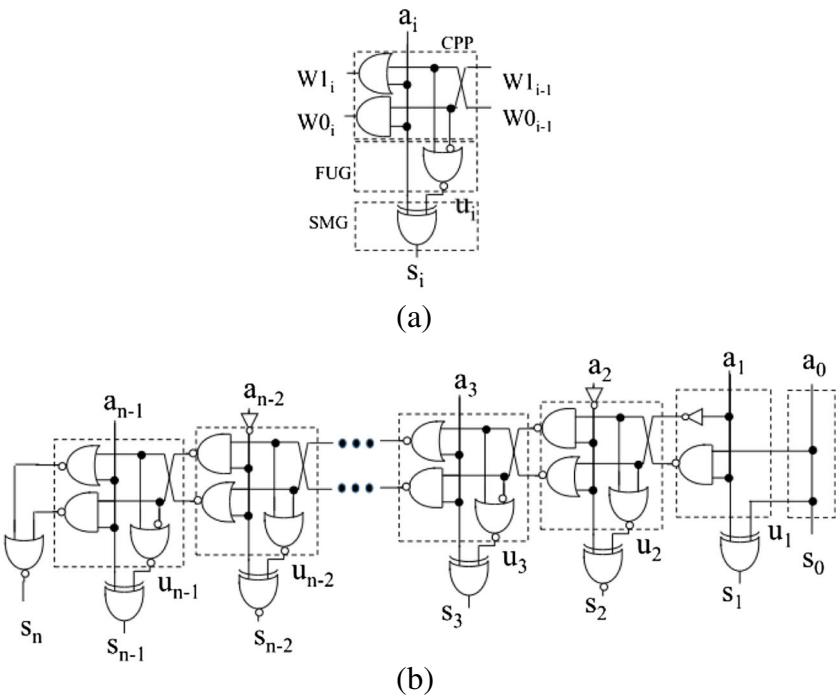


Figure 2 UCA-based RCA: **a** UCA cell for 3N; and **b** (n+1)-bit RCA.

and the functions u_{2i} and u_{2i+1} of the FUG block are

$$\begin{aligned} u_{2i} &= (W00_{2i-1}'W01_{2i-1}) + (W10_{2i-1}'W11_{2i-1}) \\ u_{2i+1} &= a_{2i}'[W00_{2i-1}'W10_{2i-1}] + a_{2i}[W01_{2i-1}'W11_{2i-1}] \end{aligned} \quad (20)$$

Proof Consider $W00_{2i+1}=a_{2i+1}a_{2i}c_{2i+1}$, with c_{2i+1} in (18), we obtain $W00_{2i+1}=a_{2i+1}a_{2i}W11_{2i-1}$. Similarly, one can easily derive the remaining terms in (19). By (17),

$$\begin{aligned} u_{2i} &= a_{2i-2} \oplus c_{2i-1} = (a_{2i-2}c_{2i-1}) \oplus (a_{2i-2} + c_{2i-1}) \\ &= [a_{2i-1} \oplus (a_{2i-2}c_{2i-1})] \oplus [a_{2i-1} \oplus (a_{2i-2} + c_{2i-1})] \\ &= (W00_{2i-1}'W10_{2i-1}) \oplus (W01_{2i-1}'W11_{2i-1}) \\ &= (W00_{2i-1}'W10_{2i-1})'(W01_{2i-1}'W11_{2i-1}) + \\ &\quad (W00_{2i-1}'W10_{2i-1})\,(W01_{2i-1}'W11_{2i-1})' \end{aligned}$$

Since

$$\begin{aligned} &W00_{2i-1}W11_{2i-1} = W00_{2i-1},\ W10_{2i-1}'W01_{2i-1}' = W10_{2i-1}', \\ &W10_{2i-1}W01_{2i-1} = W01_{2i-1},\ \text{and}\ W00_{2i-1}'W11_{2i-1}' = W11_{2i-1}', \\ &u_{2i} = W00_{2i-1}W01_{2i-1}' + W10_{2i-1}'W11_{2i-1} + \\ &\quad W00_{2i-1}'W01_{2i-1} + W10_{2i-1}W11_{2i-1}' \end{aligned}$$

and since

$$\begin{aligned} &W00_{2i-1}W01_{2i-1}' = 0 \text{ and } W10_{2i-1}W11_{2i-1}' = 0, \text{ thus} \\ &u_{2i} = (W00_{2i-1}'W01_{2i-1}) + (W10_{2i-1}'W11_{2i-1}) \end{aligned}$$

Similarly,

$$\begin{aligned} u_{2i+1} &= a_{2i-1} \oplus c_{2i} = a_{2i-1} \oplus [a_{2i}'(a_{2i-2}c_{2i-1}) + a_{2i}(a_{2i-2} + c_{2i-1})] \\ &= a_{2i}'[a_{2i-1} \oplus (a_{2i-2}c_{2i-1})] + a_{2i}[a_{2i-1} \oplus (a_{2i-2} + c_{2i-1})] \\ &= a_{2i}'[W00_{2i-1}'W10_{2i-1}] + a_{2i}[W01_{2i-1}'W11_{2i-1}] \end{aligned}$$

Figure 3a shows the UCA2 cell, where the CPP block can be realized by using NAND2/NOR2 gates, as illustrated in Fig. 3b. The critical path, as indicated in red, of the (2 m)-bits UCA2-based RCA in Fig. 3c includes an INV and an NOR2 gate in cell #0, two NOR2 gates in each of cells #1 to #(m-2), and the FUG and SMG blocks in cell #(m-1). Thus, the propagation delay is

$$\begin{aligned} \Delta_{\mathrm{RCA(UCA2)(5N)}} &= \Delta_{\mathrm{INV}} + \Delta_{\mathrm{NOR2}} + (\mathrm{m}-2)\Delta_{\mathrm{UCA2}} \\ &\quad + \Delta_{\mathrm{FUG(5N)}} + \Delta_{\mathrm{SMG}} \end{aligned} \quad (21)$$

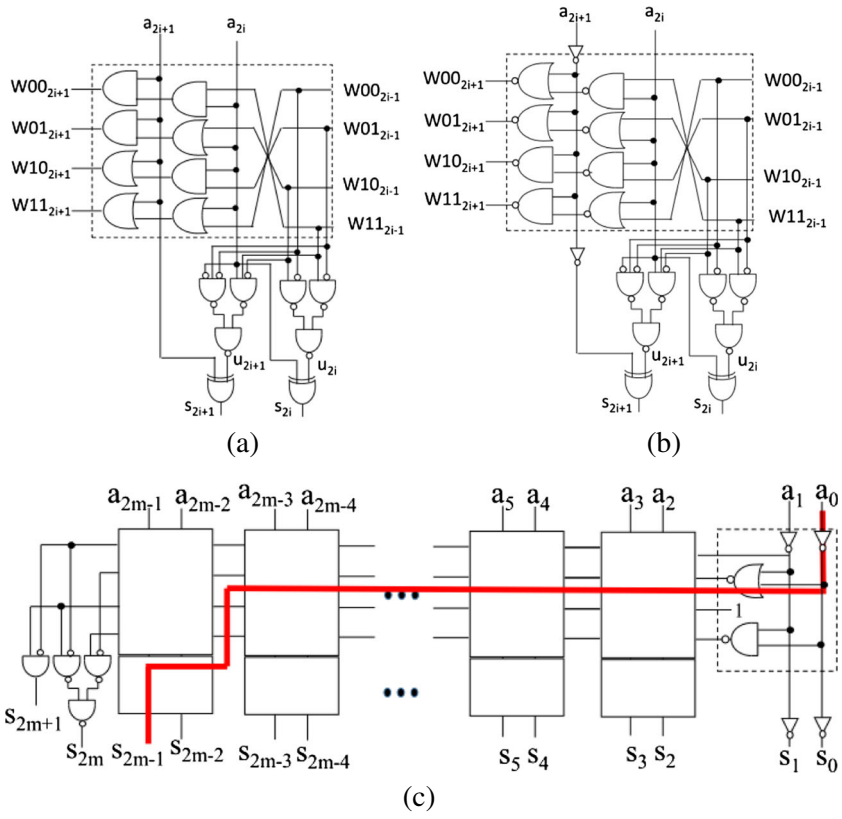


Figure 3 5N operation: **a & b** UCA2 cell; and **c** UCA2-based RCA.

3.3 UCA-based RCA for (2^k+1)N Operation

We first construct the CPP block and then the FUG and SMG blocks

CPP Block Let r_j be a binary number, either a 0 or a 1,

$$\begin{aligned} \#_{ri} = \#_0 = \text{“ ”(Logic AND operation), if } r_i = 0, \\ \#_{ri} = \#_1 = \text{“ + ”(logic OR operation), if } r_i = 1. \end{aligned} \tag{22}$$

Thus, $W01_{2i+1}$ in (19) can be expressed as

$$W(r_1r_0)_{2i+1} = a_{2i+1}\#_0\left(a_{2i}\#_1 W(r_1'r_0')_{2i-1}\right) \tag{23}$$

where r_1' and r_0' are the bit-complement of r_1 and r_0, respectively.

Figure 3b shows the block diagram of the UCA2 cells, where the CPP block is highlighted. The CPP block contains 4 carry propagation paths, Pa, a=0~3, and each path contains 2 gates, Jab, b=0 or 1. The gate type of Jab is determined by the index of the path output Wab. For example, the output W10 of the path pa which includes both gates J21 and J20. Here, W10 means a=1 and b=0, i.e., J21 is an OR gate (because a=1) and J20 is an AND gate (because b=0). Fig. 4a is a symbolic representation of that in Fig. 3b.

Similarly, the CPP block of the UCAk cells for (2^k+1)N operation can be expressed as

$$\begin{aligned} &W(r_{k-1}..r_1r_0)_{k(i+1)-1} \\ &= a_{k(i+1)-1}\#_{rk-1}\left(\ldots\#_{r1}\left(a_{ki}\#_{r0}W(r_{k-1}'\ldots r_0')_{ki-1}\right)\right) \end{aligned} \tag{24}$$

Figure 4b shows the symbolic representation of the logic function for the CPP block of the UCAk cell and its delay is $\Delta_{UCA(k)}=k\Delta_{NOR2}$.

FUG and SMG Blocks Let U(x0) and U(x1) be defined as follows: (“x” means “don’t care term”)

$$U(x0) = W00_{2i-1}'W10_{2i-1},\ U(x1) = W01_{2i-1}'W11_{2i-1} \tag{25}$$

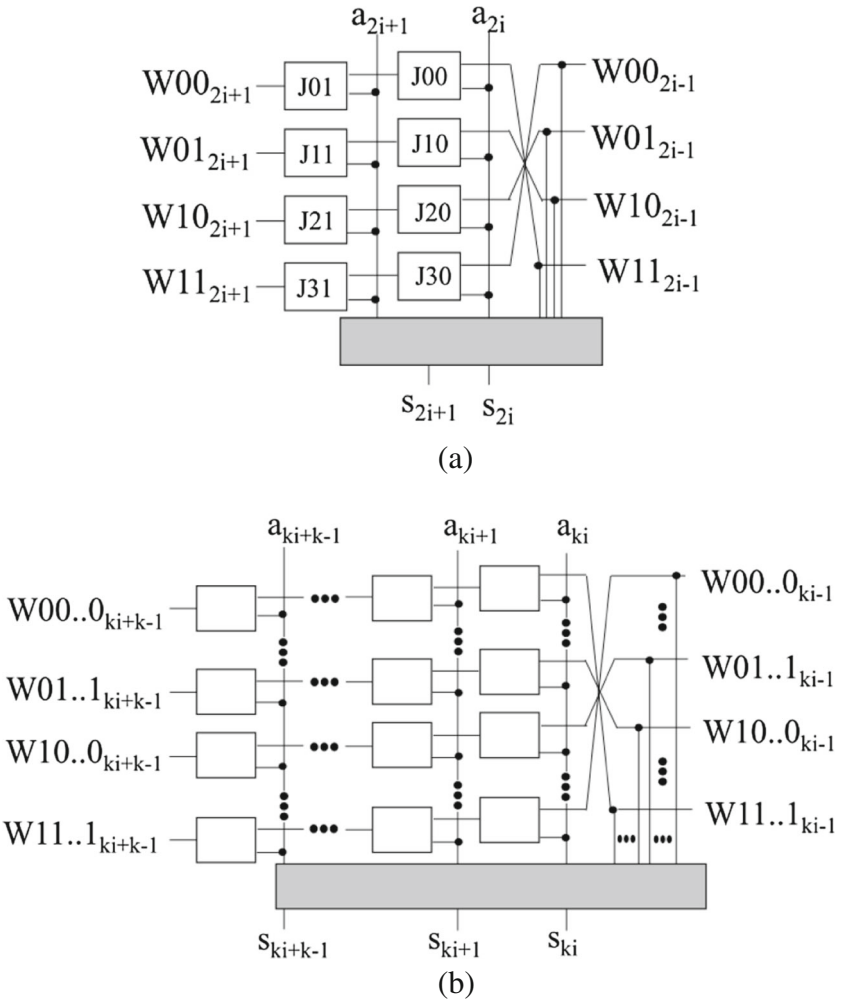


Figure 4 Symbolic representation: **a** for 5N; and **b** for (2^k+1)N.

U(x0) takes the terms Wx0, while U(x1) is formed by the terms Wx1. Let V(0x) and V(1x) be defined as

$$V(0x) = W00_{2i-1}'W01_{2i-1}, V(1x) = W10_{2i-1}'W11_{2i-1} \quad (26)$$

Thus, (19) can be expressed as

$$\begin{aligned} u_{2i+1} &= a_{2i}'U(x0) + a_{2i}U(x1) \\ u_{2i} &= V(0x) + V(1x) \end{aligned} \quad (27)$$

Figure 5a is the symbolic representation of FUG and SMG blocks of UCA2 cell for 5 N operation, where U(x0)* is an AND3 gate with 3 inputs a_{2i}', $W00_{2i-1}'$, and $W10_{2i-1}$, and U(x1)* is that with a_{2i}, $W01_{2i-1}'$, and $W11_{2i-1}$. Thus, the delay path, as shown in Fig. 5b, includes an AND3, a OR2, and a XOR, and it can be implemented with an INV, an NAND3, an NAND2, and a XOR as follows,

$$\begin{aligned} \Delta_{FUG(5N)} &= \Delta_{INV} + \Delta_{NAND3} + \Delta_{NAND2}; \Delta_{SMG} \\ &= \Delta_{XOR} \end{aligned} \quad (28)$$

Similarly, the UCA3 cell for 9 N operation, the FUG block can be expressed as follows,

$$\begin{aligned} u_{3i+2} &= a_{3i+1}'a_{3i}'U(x00) + a_{3i+1}'a_{3i}U(x01) + \\ &\quad a_{3i+1}a_{3i}'U(x10) + a_{3i+1}a_{3i}U(x11) \\ u_{3i+1} &= a_{3i}'[V(0x0) + V(1x0)] + a_{3i}[V(0x1) + V(1x1)] \\ u_{3i} &= V(0x0) + V(1x0) + V(0x1) + V(1x1) \end{aligned} \quad (29)$$

Figure 5c illustrates the delay path for 9 N operation, where u_{3i+2}, in (29), can be realized by four AND4 gates and one OR4 gates. However, the term $a_{3i+1}'a_{3i}'U(x00)$ can be realized by an AND2 with a_{3i+1}' and a_{3i}' and an AND3 which takes the output of AND2 and two inputs of U(x00). Since the function AND2 with a_{3i+1}' and a_{3i}' can be pre-calculated and the AND2 gate will not be in the critical path, as shown in Fig. 5c. The OR4 is realized by a NOR2 followed by a NAND2. This results in an INV, an NAND3, an NOR2, and an NAND2 included in the critical path of the FUG block of the UCA3 cell, i.e.,

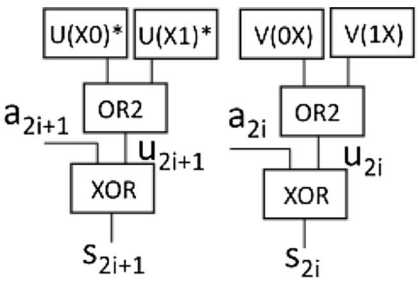
$$\begin{aligned} \Delta_{FUG(9N)} &= \Delta_{INV} + \Delta_{NAND3} + \Delta_{NOR2} \\ &\quad + \Delta_{NAND2}; \Delta_{SMG} \\ &= \Delta_{XOR} \end{aligned} \quad (30)$$

Similarly, the delay path in Fig. 5d for UCA cell is,

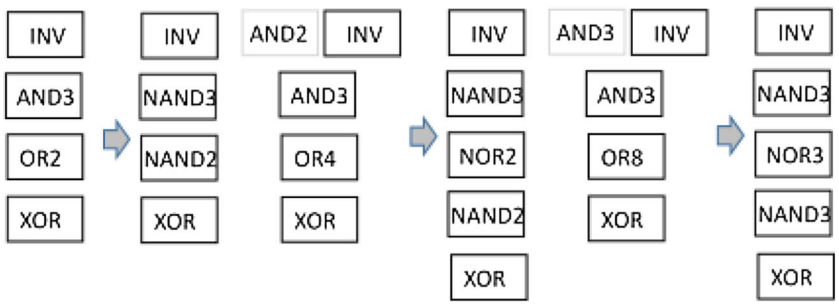
$$\begin{aligned} \Delta_{FUG(17N)} &= \Delta_{INV} + 2\Delta_{NAND3} + \Delta_{NOR3}; \Delta_{SMG} \\ &= \Delta_{XOR} \end{aligned} \quad (31)$$

3.4 Performance Evaluation

The proposed design methodology is mainly about the structure of hardware implementation which is based on the newly developed equations. The actual realization of the proposed



(a)



(b) (c) (d)

Figure 5 Delay paths: **a** FUG and SMG blocks for 5N; **b** Delay paths for 5N; **c** for 9N; and **d** for 17N.

UCA, for example, may vary significantly. Therefore, the following performance comparisons were conducted assuming the same type of hardware realization. Thus, the speed performance estimation only consider the total gate delay in the critical path under the same conditions, where the path delays and loading effects were not considered for this rough estimation. In fact, the performances are evaluated based on the gate delays of the standard cells in TSMC 0.18 μm CMOS process technology, as tabulated in Table 1 [2], where the cell height is 5.04 μm.

The delays of UCA-based RCAs are, $n=2^t$, t≥4, where $(\Delta_{inv}+(k-1)\Delta_{NOR2})$ is in Cell#0 for 5 N and 17 N operations,

$$\begin{aligned}\Delta_{RCA(UCA)(3N)} &= \Delta_{NAND2} + (n-2)\Delta_{NOR2} + \Delta_{inv} + \Delta_{XOR}\\ \Delta_{RCA(UCA)(5N)} &= (n-3)\Delta_{NOR2} + 2\Delta_{inv} + \Delta_{NAND3} + \Delta_{NAND2} + \Delta_{XOR}\\ \Delta_{RCA(UCA)(17N)} &= (n-5)\Delta_{NOR2} + 2\Delta_{inv} + 2\Delta_{NAND3} + \Delta_{NOR3} + \Delta_{XOR}\end{aligned} \tag{32}$$

By (8), the delays of FA-based RCAs are, $n=2^t$, $t\geq 4$,

$$\begin{aligned}\Delta_{RCA(FA)(3N)} &= \Delta_{NOR2} + 2(n-2)\Delta_{NAND2} + \Delta_{XOR}\\ \Delta_{RCA(FA)(5N)} &= 2\Delta_{NOR2} + 2(n-3)\Delta_{NAND2} + \Delta_{XOR}\\ \Delta_{RCA(FA)(17N)} &= 4\Delta_{NOR2} + 2(n-5)\Delta_{NAND2} + \Delta_{XOR}\end{aligned} \tag{33}$$

Based on (32), (33), and Table 1, the delays of both FA-based RCAs and UCA-based RCAs for 3 N (k=1), 5 N (k=2), and 17 N (k=4) with the 16~1024 bits were computed and the values were tabulated in Table 2. Results show that UCA-RCA for 17 N operation has slightly better than that for 5 N operation. This is so simply because $(2\Delta_{NOR2}+\Delta_{NAND2})$= 0.1176 ns and $(\Delta_{NAND3}+\Delta_{NOR3})$=0.1044 ns, where the difference is about 0.01 ns.

Based on Table 2, Fig. 6a and b plot the delays of both FA-based RCA and UCA-based RCA, respectively, for 3 N, 5 N, and 17 N operations. Results show that their delays are almost the same for the same size with the same approach.

Table 2 tabulates the normalized speed performance, where the delay of FA-based RCA is normalized and set to 1. The delay ratios of UCA-based UCAs for various bit sizes are shown in Fig. 6c. Results show that the delay ratios are approximately 65 % which can be simply calculated from the delay ratio of UCA cell (Δ_{NOR2}) over a FA cell ($2\Delta_{NAND2}$), i.e., $\Delta_{NOR2}/(2\Delta_{NAND2})$=0.0426/0.0648=0.66=66%.

Even though the proposed UCA-based RCAs are about 34 % faster than the FA-based ones, their speed performance is still too slow particularly for wider bit sizes. The following section attempts to further improve the speed performance.

Table 1 Cell delay data [2]

	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 Hybrid Adders for $(2^k\pm1)$N Operations

As mentioned, the RCA achieves lower hardware cost, while the CLA offers high speed performance. In order to achieve higher speed performance for the additions, a hybrid adder combining RCA and CLA (or generate-propagate adder), as shown in Fig. 7, is presented. The adder is simple and modular, where the lower bit CLA is taken as a module.

4.1 UCA-based Hybrid Adder for 3N Operation

For 3N operation, by (14), the carry-out bits at the first 4 stages of the CPP blocks can be written as follows,

$$\begin{aligned} W\ 0_0 &= a_0 W\ 1_{-1} & W\ 1_0 &= a_0 + W\ 0_{-1}\\ W0_1 &= a_1 W1_0 = a_0 a_1 + a_1 W0_{-1} & W1_1 &= a_1 + W0_0 = a_1 + a_0 W1_{-1}\\ W\ 0_2 &= a_1 a_2 + a_0 a_2 W\ 1_{-1} & W\ 1_2 &= a_2 + a_0 a_1 + a_1 W\ 0_{-1}\\ W0_3 &= a_2 a_3 + a_0 a_1 a_3 + a_1 a_3 W0_{-1} & W\ 1_3 &= a_3 + a_1 a_2 + a_0 a_2 W\ 1_{-1}\end{aligned} \tag{34}$$

Thus, both $W0_3$ and $W1_3$ can be written as

$$W0_3 = R_{01} + R_{00} W0_{-1} \quad W1_3 = Q_{01} + Q_{00} W1_{-1} \tag{35}$$

where

$$R_{00} = a_1 a_3,\ R_{01} = a_2 a_3 + a_0 a_1 a_3, Q_{00} = a_0 a_2,\ Q_{01} = a_3 + a_1 a_2 \tag{36}$$

Table 2 Performance evaluation for RCAs

	Bits	16	32	64	128	256	512	1024
3N Operation	FA	1.09	2.13	4.2	8.35	16.65	33.24	66.41
	UCA	0.8	1.48	2.84	5.57	11.02	21.93	43.74
5N Operation	FA	1.07	2.11	4.18	8.33	16.62	33.21	66.39
	UCA	0.83	1.51	2.87	5.60	11.05	21.96	43.77
17N Operation	FA	1.03	2.06	4.14	8.29	16.58	33.17	66.35
	UCA	0.82	1.50	2.86	5.59	11.04	21.94	43.76

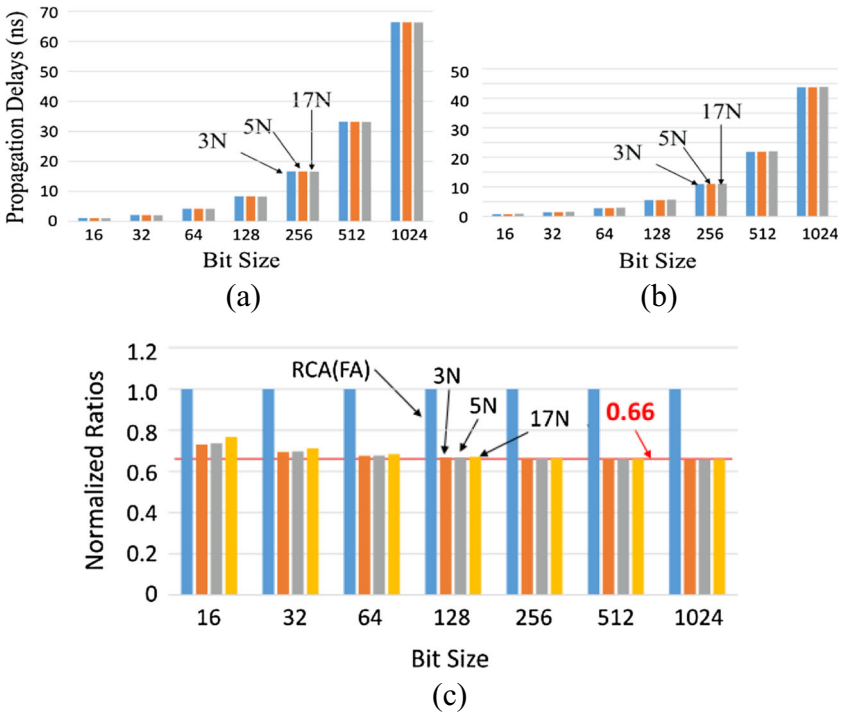


Figure 6 Speed performance: **a** FA-based RCA; **b** UCA-based RCA;and **c** comparison.

Similarly, both $W0_7$ and $W1_7$ are expressed as

$$W0_7 = R_{11} + R_{10}W0_3 \quad W1_7 = Q_{11} + Q_{10}W1_3 \tag{37}$$

where

$$R_{10} = a_5a_7,\ R_{11} = a_6a_7 + a_4a_5a_7,\ Q_{00} = a_4a_6,\ Q_{01} = a_7 + a_5a_6$$

By (35), both $W0_7$ and $W1_7$ in (37) can be re-written as

$$\begin{aligned} W0_7 &= R_{11} + R_{10}R_{01} + R_{10}R_{00}W0_{-1} \\ W1_7 &= Q_{11} + Q_{10}Q_{01} + Q_{10}Q_{00}W1_{-1} \end{aligned} \tag{38}$$

Figure 8a shows a 32-bits hybrid adder, where each CLA unit processes 8-bit data. Each CLA unit, adopting the parallel prefix adder structure [7, 8], is comprised of two 4-bit RQ generator units (RQGUs, or RQGU-4), a 2-bit CLA (or CLA-2), and two 4-bit Function U generation units (FUGUs, or FUGU-4). The RQGU-4, as shown in Fig. 8b, realizes the functions in (34) and it replaces both PG unit and Block CLA-4 (BCLA-4) in the conventional CLA structure. The CLA-2, as illustrated in Fig. 8c, implements the functions in (35) and (38), and the FUGU-4 in Fig. 8d, is for the functions in (34) and (15). Note that $c_i=(W0_i,W1_i)$ in Fig. 8a, The delays of these units can be expressed as

$$\begin{aligned} &\Delta_{RQGU-4} = \Delta_{NAND3} + \Delta_{NAND2};\ \Delta_{CLA-2} = 2\Delta_{NAND3};\\ &\Delta_{FUGU-4} = \Delta_{NAND3} + \Delta_{NAND2} + \Delta_{INV} + \Delta_{NOR2};\ \Delta_{SUM} = \Delta_{XOR}; \end{aligned}$$

The critical path of the 32-bit hybrid adder includes a RQGU-4, four CLA-2 s, a FUGU-4, and a SUM, i.e.,

$$\begin{aligned} \Delta_{HyA(3N)(32)} &= \Delta_{RQGU-4} + 4\Delta_{CLA-2} + \Delta_{FUGU-4} + \Delta_{SUM} \\ &= 10\Delta_{NAND3} + 2\Delta_{NAND2} + \Delta_{INV} + \Delta_{NOR2} + \Delta_{XOR} \end{aligned}$$

In general, for $n=2^t$, $t>3$, the delay

$$\begin{aligned} \Delta_{HyA(3N)(n)} &= \Delta_{RQGU-4} + (n/8)\Delta_{CLA-2} + \Delta_{FUGU-4} + \Delta_{SUM} \\ &= (n/4+2)\Delta_{NAND3} + 2\Delta_{NAND2} + \Delta_{INV} + \Delta_{NOR2} + \Delta_{XOR} \end{aligned} \tag{39}$$

Similarly, the 4-bit CLA module of the hybrid adder in Fig. 8 can be replaced by 8-bit one. The delay is

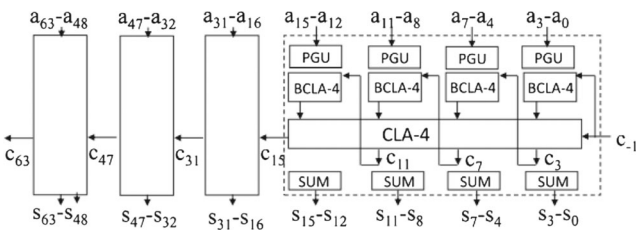


Figure 7 Hybrid adder – RCA & CLA.

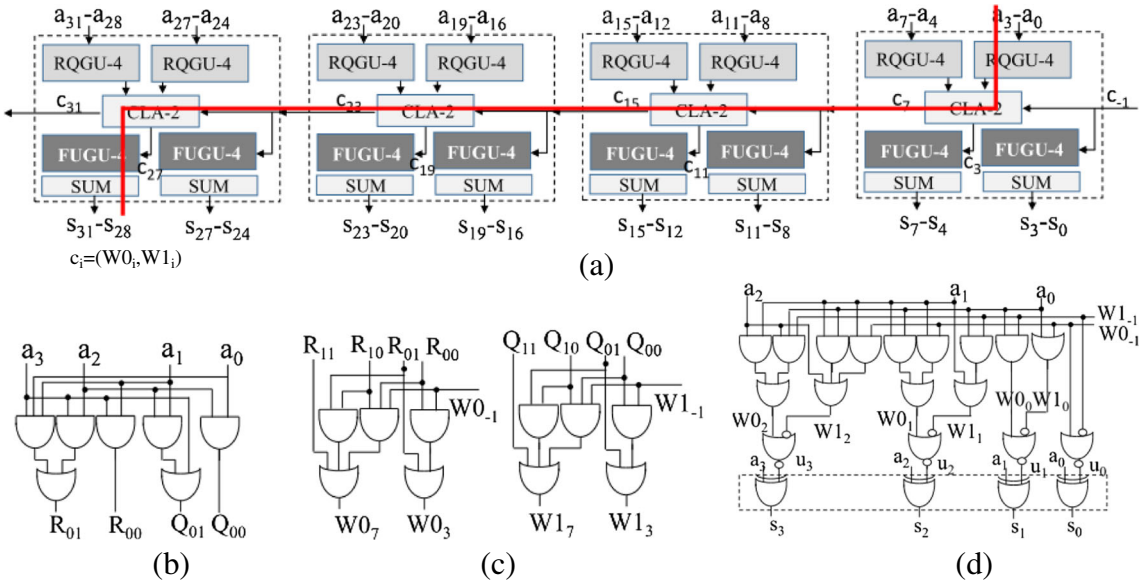


Figure 8 Proposed hybrid adder for 3N operation: **a** 32-bit hybrid adder; **b** RQGU-4; **c** CLA-2; and **d** FUGU-4 & SUM.

$$\Delta_{\mathrm{HyA(3N)(n)}} = \Delta_{\mathrm{RQGU-8}} + (\mathrm{n}/16)\Delta_{\mathrm{CLA-2}} + \Delta_{\mathrm{FUGU-8}} + \Delta_{\mathrm{SUM}} \tag{40}$$

where

$$\begin{aligned} \Delta_{\mathrm{RQGU-8}} &= \Delta_{\mathrm{AND5}} + \Delta_{\mathrm{OR3}}; \\ \Delta_{\mathrm{FUGU-8}} &= \Delta_{\mathrm{AND5}} + \Delta_{\mathrm{OR4}} + \Delta_{\mathrm{INV}} + \Delta_{\mathrm{NOR2}}. \end{aligned} \tag{41}$$

The CLA module of the hybrid adder may include multi-level structure similar to the conventional CLA. For example, the CLA module of the 64-bit hybrid adder in Fig. 9 is with 2-level CLA structure. Let L denote as the number of levels in the CLA module. Thus, the delay of such hybrid adder is, r=m*2^{L}, m=4 or 8, $L \geq 2$, and $n \geq 2$r,

$$\Delta_{\mathrm{HyA(3N)(n)}} = \Delta_{\mathrm{RQGU-m}} + 2(\mathrm{L}-1)\Delta_{\mathrm{BCLA-2}} + (\mathrm{n}/\mathrm{r})\Delta_{\mathrm{CLA-2}} + \Delta_{\mathrm{FUGU-m}} + \Delta_{\mathrm{SUM}} \tag{42}$$

4.2 UCA-based Hybrid Adder for $(2^{k}+1)$N Operation

Similar to the derivation process for 3 N operation in (35)-(38), the following component delays can be derived from Property 1 for 5 N operation, where

$$\begin{aligned} \Delta_{\mathrm{RQGU-4(5N)}} &= \Delta_{\mathrm{RQGU-4(3N)}} = \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}}; \\ \Delta_{\mathrm{FUGU-4(5N)}} &= 2\Delta_{\mathrm{NAND3}} + 2\Delta_{\mathrm{NAND2}} + \Delta_{\mathrm{INV}}; \\ \Delta_{\mathrm{RQGU-8(5N)}} &= \Delta_{\mathrm{RQGU-8(3N)}} = \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}} + 2\Delta_{\mathrm{NOR2}} \\ \Delta_{\mathrm{FUGU-8(5N)}} &= 2\Delta_{\mathrm{NAND3}} + 2\Delta_{\mathrm{NAND2}} + 2\Delta_{\mathrm{NOR2}} + \Delta_{\mathrm{INV}}; \end{aligned} \tag{43}$$

Similarly, one can also derive for 17 N operation as

$$\begin{aligned} \Delta_{\mathrm{RQGU-4(17N)}} &= \Delta_{\mathrm{RQGU-4(3N)}} = \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}}; \\ \Delta_{\mathrm{FUGU-4(17N)}} &= 3\Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}} + \Delta_{\mathrm{INV}} + \Delta_{\mathrm{NOR3}}; \\ \Delta_{\mathrm{RQGU-8(17N)}} &= \Delta_{\mathrm{RQGU-8(3N)}} = \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}} + 2\Delta_{\mathrm{NOR2}}; \\ \Delta_{\mathrm{FUGU-8(17N)}} &= 3\Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}} + 2\Delta_{\mathrm{NOR2}} + \Delta_{\mathrm{INV}} + \Delta_{\mathrm{NOR3}}. \end{aligned} \tag{44}$$

For 9N operation with k=3, the UCA3 cells are employed and each cell contains 3 bits. Thus, the delays are

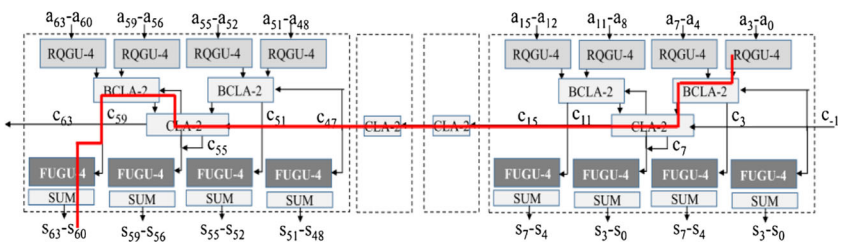


Figure 9 Sixty-four-bits hybrid adder with 2-level CLA module.

Table 3 Performance comparison

(a) 4-bit modules								
	Bits	16	32	64	128	256	512	1024
3 N Operation	L=1	0.55	0.73	1.09	1.82	3.27	6.17	11.94
	L=2		0.68	0.86	1.22	1.95	3.40	6.30
	L=3			0.81	0.99	1.35	2.08	3.53
	L=4				0.94	1.12	1.48	2.21
	L=5					1.07	1.25	1.61
5 N Operation	L=1	0.95	0.77	1.13	1.85	3.30	6.20	12.00
	L=2		0.71	0.90	1.26	1.98	3.43	6.33
	L=3			0.84	1.03	1.39	2.11	3.56
	L=4				0.97	1.16	1.52	2.24
	L=5					1.10	1.28	1.65
17 N Operation	L=1	0.66	0.84	1.20	1.93	3.38	6.27	12.07
	L=2		0.79	0.97	1.33	2.06	3.50	6.40
	L=3			0.92	1.10	1.46	2.18	3.63
	L=4				1.05	1.23	1.59	2.31
	L=5					1.18	1.36	1.72
		Bits	32	64	128	256	512	1024
(b) 8-bit modules								
3 N Operation		Bits	32	64	128	256	512	1024
		L=1	0.72	0.90	1.26	1.99	3.44	6.34
		L=2		0.85	1.03	1.39	2.12	3.57
		L=3			0.98	1.16	1.52	2.25
		L=4				0.89	1.07	1.43
5 N Operation		L=1	0.76	0.94	1.30	2.02	3.47	6.37
		L=2		0.89	1.07	1.43	2.15	3.6
		L=3			1.01	1.2	1.56	2.28
		L=4				1.14	1.33	1.69
17 N Operation		L=1	0.83	1.01	1.37	2.1	3.55	6.44
		L=2		0.96	1.14	1.5	2.23	3.68
		L=3			1.09	1.27	1.63	2.36
		L=4				1.22	1.4	1.76

$$\begin{aligned}
&\Delta_{\mathrm{RQGU-3(9N)}} = 2\Delta_{\mathrm{NAND2}};\\
&\Delta_{\mathrm{RQGU-6(9N)}} = \Delta_{\mathrm{AND4}} + \Delta_{\mathrm{OR3}} = \Delta_{\mathrm{NAND2}} + \Delta_{\mathrm{NOR2}} + \Delta_{\mathrm{OR3}}\\
&\Delta_{\mathrm{FUGU-3(9N)}} = 3\Delta_{\mathrm{NAND2}} + \Delta_{\mathrm{INV}} + \Delta_{\mathrm{NOR2}} + 2\Delta_{\mathrm{NAND3}};\\
&\Delta_{\mathrm{FUGU-6(9N)}} = 2\Delta_{\mathrm{NAND2}} + 2\Delta_{\mathrm{NOR2}} + \Delta_{\mathrm{INV}} + \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{OR3}}.
\end{aligned} \tag{45}$$

Similarly, for (2^k+1)N operations, the k-bit cells, UCAk cells, are employed. Thus, the hybrid adder may include the k-bit CLA modules.

4.3 Performance Evaluation

This section roughly estimates the speed performances of the FA-based RCAs, UCA-based RCA, and the hybrid adders with various bit sizes for 3N, 5N, 9N, and 17N operations. However, the performance evaluation process is readily applied for any bit sizes and $(2^k\pm1)$N operations.

The speed performance is roughly evaluated based on the gate delays of the standard cells listed in Table 1, where the AND (NAND) and OR (NOR) gates are limited to their fan-ins up to 3. Thus, the AND5 and OR4 gates in (41) are implemented in two-level logic gates, i.e., AND5=(AND3)•(AND2), and OR4=(OR2)+(OR2).

The delays of the RQGU-4 and RQGU-8 for 3N, 5N, and 17N operations are the same. By Table 1,

$$\begin{aligned}
\Delta_{\mathrm{RQGU-4}} &= \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}} = 0.0777\mathrm{ns}\\
\Delta_{\mathrm{RQGU-8}} &= \Delta_{\mathrm{AND5}} + \Delta_{\mathrm{OR4}}\\
&= \Delta_{\mathrm{NAND3}} + \Delta_{\mathrm{NAND2}} + 2\Delta_{\mathrm{NOR2}} = 0.1629\mathrm{ns}\\
\Delta_{\mathrm{RQGU-3}} &= 2\Delta_{\mathrm{NAND2}} = 0.0648\mathrm{ns}\\
\Delta_{\mathrm{RQGU-6}} &= \Delta_{\mathrm{NAND}} + \Delta_{\mathrm{NOR2}} + \Delta_{\mathrm{NOR3}} + \Delta_{\mathrm{INV}} = 0.1602\mathrm{ns}
\end{aligned} \tag{46}$$

Based on (39, 40, 41, 42, 43, 44, 45, and 46), the delays of the hybrid adders are calculated and tabulated in Table 3.

Results show that the delays are indeed improved as the number of levels, L, increases. For example, for 17 N operation with 1024 bits, the delays was improved from 66.51 ns for the FA-based RCA to 43.64 ns for the proposed UCA-based RCA, as shown in Table 2, Further, the delay is reduced to 12.07 ns and 6.44 ns for the proposed hybrid adders with 4-bits and 8-bits modules, respectively, as illustrated in Table 3 and plotted in Fig. 10a. The delays can be further decreased to 2.31 ns and 1.76 ns by using the proposed hybrid with 4-levels of 4-bits and 8-bits modules, respectively, as plotted in Fig. 10b.

4.4 Discussion

The proposed UCA design concept has the advantages of better speed performance than the conventional FA design approach. This sub-section compares the speed performance of the UCA-CLA and conventional CLA (CV-CLA). The delays of both BCLA-4 and CLA-4 [5–7] are

$$\Delta_{CLA-4} = \Delta_{AND5} + \Delta_{OR5} = \Delta_{NAND3} + \Delta_{NAND2} + \Delta_{NOR3} + \Delta_{NOR2}$$
$$\Delta_{BCLA-4} = \Delta_{AND4} + \Delta_{OR4} = 2(\Delta_{NAND2} + \Delta_{NOR2})$$

For n-bit CV-CLAs, $n=4^t$, with CLA-4 and BCLA-4 modules, the delay is

$$\Delta_{CV-LA} = \Delta_{PGU} + 2(t-1)\Delta_{BCLA-4} + \Delta_{CLA-4} + \Delta_{SUM} \quad (47)$$

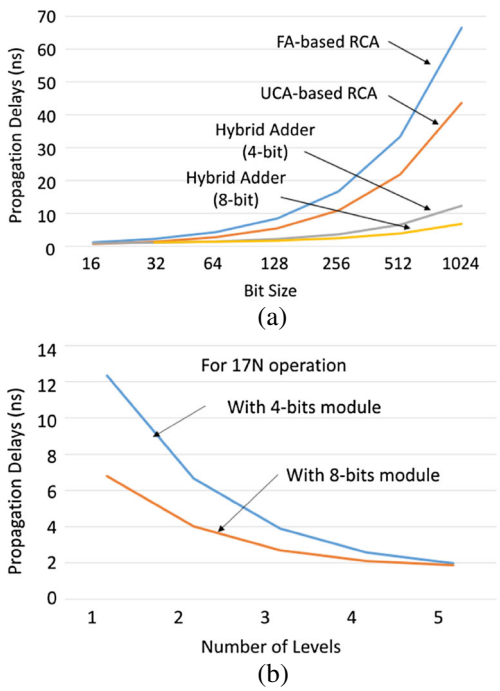


Figure 10 Propagation delays for 17N operation with 1024 bits: **a** Various adders; and **b** Hybrid adders with 4-bits and 8-bits modules.

Table 4 Performance comparison (CLA Modules)

	CLA-4 module				Normalized ratio			
Bits	16	64	256	1024	16	64	256	1024
CLA	0.77	1.07	1.37	1.67	1	1	1	1
3N	0.55	0.85	1.15	1.45	0.71	0.79	0.84	0.87
5N	0.58	0.88	1.18	1.48	0.76	0.83	0.86	0.89
17N	0.66	0.96	1.26	1.56	0.85	0.89	0.92	0.93

and the delay of an n-bit UCA-CLA is

$$\Delta_{UCA-CLA} = \Delta_{RQGU-4} + 2(t-2)\Delta_{BCLA-4} + \Delta_{CLA-4} + \Delta_{FUGU-4} + \Delta_{SUM} \quad (48)$$

Note that the delay ($\Delta_{RQGU-4}+\Delta_{FUGU-4}$) in UCA-CLA is compared with ($\Delta_{PGU}+2\Delta_{BCLA-4}$) in CV-CLA. Table 4 compares the speed performance of CV-CLA and UCA- CLA for 3 N, 5 N, and 17 N operations with various bit sizes.

The UCA-CLAs for 3N, 5N, and 17N operations respectively achieve approximately 30 %, 25 %, and 15 % less delays than CV-CLA for 16-bit addition, and about 13 %, 11 %, and 7 % for 1024 bit addition. As mentioned, the delay of FUGU-2^k block increases with k considerably. Thus, the proposed UCA-CLAs may gain the advantage of better speed performance for lower values of k. The proposed simple and modular UCA-RCA are perfectly applied for those constant multiplications with smaller bit sizes, such as 3 N, 5 N, and 7 N operations for arithmetic coding, Booth encoding, and the divider design [9] as a cost-effective solution. For higher speed performance, the proposed UCA-CLA can also be applied.

5 Conclusion

Constant multiplier performs a multiplication 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. This study has presented efficient algorithm and fast hardware implementation for $(2^k \pm 1)$N operation with additions. No multiplications were needed.

The salient features of the proposed UCA design concept came from the use of the carry function of (12) instead of (5), thus improving the propagation delay path and achieving 34 % in speed performance improvement. The design concept is perfectly applied for the RCA design for faster speed. The proposed UCA-HyA provides a cost-effective solution for constant multiplication, and the proposed UCA-CLA further improves the speed performance.

In addition, the study presents the hardware implementation for $(2^k+1)N$ operation, how to apply the proposed UCA design concept for any other form of constants with one addition also leads a very interesting topic for future research. Finally, as mentioned in [2], the constant divider can be developed in a similar way.

References

1. Cappello, P. R., & Steiglitz, K. (1984). Some complexity issues in digital signal processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 32*(No. 5), 1037–1041.
2. Wey, C. L., Jui, P.-C., & Sung, G.-N. (2014). Efficient Multiply-by-3 and Divide-by-3 Algorithms and Their Fast Hardware Implementation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E97-A*(2), 616–623.
3. Jui, P.-C., Sung, G.-N., & Wey, C. L. (2012). *Efficient algorithm and hardware implementation of 3N for arithmetic and for radix-8 encodings, Proc. of IEEE Midwest Symp. on Circuits and Systems, Boise, Idaho* (pp. 418–421).
4. Oudjida, A. K., & Chaillet, N. (2014). Radix-2r Arithmetic for multiplication by a constant. *IEEE Transactions on Circuits and Systems – II: Express Briefs, 61*, 349–353.
5. Koren, I. (1993). *Computer arithmetic algorithms*. New Jersey: Prentice-Hall, Inc.
6. Huang, K. (1979). *Computer arithmetic: principles, architecture, and design*. New York: Wiley.
7. Chang, S.-K., & Wey, C. L. (2012). *A fast 64-bit hybrid adder design in 90 nm CMOS process*. Boise: Proc. of IEEE Midwest Symp. on Circuits and Systems.
8. Brent, R. P., & Kung, H. T. (1982). A regular layout for parallel adders. *IEEE Transactions on Computers, 31*(3), 260–264.
9. Wey, C. L. (1996). Built-In Self-Test (BIST) design of high-speed carry-free dividers. *IEEE Transactions on VLSI Systems, 4*(1), 141–145.

Ping-Chang Jui received her MS 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 algorithm development for computer arithmetic; VLSI circuit design; and Technology management/strategic development.

Chin-Long Wey received his Ph.D. degree in Electrical Engineering from Texas Tech University, Lubbock, Texas, in 1983. He is currently a Distinguished Professor of Electrical and Computer 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 for 20 years from 1983 to 2003. His research interests include design, testing, and fault diagnosis of analog/mixed-signal VLSI circuits and systems; and Power electronics 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.

Muh-Tian Shiue was born in Taichung, Taiwan, in 1963. He received the B.S. degree in 1986 from the Department of Industry Education, National Taiwan Normal University, the M.S. degree in 1988 from the Department of Control Engineering, National Chiao Tung University, and the Ph.D. degree in 1998 from the Department of Electrical Engineering, National Central University, Taiwan. From 1988 to 1996, he was with the Transmission Technology Laboratory of Telecommunication Laboratories, Ministry of Transportation Communications, Taiwan, where he was involved in the digital subscriber loop technologies. From 1998 to 2000, he was with the Computer and Communications Research Laboratories of the Industrial Technology Research Institute, Hsin-chu, Taiwan, where was involved in the development of ADSL chip-set.

He jointed IC Plus Corp. in 2000 corresponding to the digital baseband chip design for ADSL transceiver. From 2001 to 2003, he was the CTO of Trendchip Technologies Corp.. Currently, he is with the Department of Electrical Engineering, National Central University, R.O.C..

His research interests include local access technologies, digital subscriber loop technologies, wireless communication systems, biomedical engineering, digital signal processing, VLSI techniques, power electronics, and control system.