Efficient hardware implementation of 3X for radix-8 encoding

G.A. Ruiz[1] and Mercedes Granda
Dpto. de Electrónica y Computadores. Facultad de Ciencias
Universidad de Cantabria. Avda. de Los Castros s/n. 39005 Santander (SPAIN)

ABSTRACT

Several commercial processors have selected the radix-8 multiplier architecture to increase their speed, thereby reducing the number of partial products. Radix-8 encoding reduces the digit number length in a signed digit representation. Its performance bottleneck is the generation of the term 3**X**, also referred to as hard multiple. This term is usually computed by an adding and shifting operation, 3**X**=2**X**+**X**, in a high-speed adder. In a 2**X**+**X** addition, close full adders share the same input signal. This property permits simplified algebraic expressions associated to a 3**X** operation other than in a conventional addition. This paper shows that the 3**X** operation can be expressed in terms of two signals, H_i and K_i , functionally equivalent to two carries. H_i and K_i are computed in parallel using architectures which lead to an area and speed efficient implementation. For the purposes of comparison, implementation based on standard-cells of conventional adders has been compared with the proposed circuits based on these H_i and K_i signals. As a result, the delay of proposed serial scheme is reduced by roughly 67% without additional cost in area, the delay and area of the carry look-ahead scheme is reduced by 20% and 17%, and that of the parallel prefix scheme is reduced by 26% and 46%, respectively.

Keywords: High-Speed Arithmetic, Arithmetic and Logic Structures, VLSI

1. INTRODUCTION

The binary number system based on two´s complement representation of numbers is commonly used in arithmetic units[1,2]. However, there are other number systems which are very useful for certain applications. Avizienis[3] defined in 1961 a class of redundant signed-digit number systems with a symmetric digit set of a radix-r positional number system. A specific case of this representation used in high speed-arithmetic is the minimum redundancy signed-digit, where the digits are of the form $d_j \in \{\bar{r}/2, \bar{r}-1, \bar{1}, 0, 1, \ldots, r/2-1, r/2\}$ with $r \geq 2$ and $r=2^p$, where $\bar{r}=-r$. For example, for radix-2, this is the digit set $\{\bar{1}, 0, 1\}$, for radix-4, it is $\{\bar{2}, \bar{1}, 0, 1, 2\}$ and for radix-8 it is $\{\bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4\}$. Thus, an n-bit two´s complement number $\mathbf{X}=(x_0, x_1, .., x_{n-1})$ can be expressed in a radix-r minimum redundancy signed representation $\mathbf{D}=(d_0, d_1, .., d_{n'-1})$ as follows:

$$\mathbf{X} = -2^{n-1} x_{n-1} + \sum_{i=0}^{n-2} x_i 2^i = \sum_{j=0}^{n'-1} d_j r^j \qquad (1)$$

where $n'=\left\lceil \frac{n+1}{p} \right\rceil$. Table I shows the word length n' of signed digit representation for different radix and values of n. Note that a higher signed digit representation leads to fewer digits.

Multiplication is perhaps the arithmetic circuit where radix-r minimum redundancy signed representation has been most widely used. It involves two basic operations: generation of partial products and their accumulation. One way to speed up the multiplication is to reduce the number of partial products by using radix-r encoding. The modified Booth's

[1] *ruizrg@unican.es; phone +34 942 20155; fax +34 942 201402*

VLSI Circuits and Systems III, edited by Valentín de Armas Sosa, Kamran Eshraghian, Felix B. Tobajas,
Proc. of SPIE Vol. 6590, 65901I, (2007) · 0277-786X/07/$18 · doi: 10.1117/12.721489

algorithm[4] is the most popular approach for implementing fast multipliers using parallel encoding. This scheme requires the generation of the multiples **X**, $\overline{\mathbf{X}}$, 2**X**, $2\overline{\mathbf{X}}$, 3**X**, $3\overline{\mathbf{X}}$, …, where **X** is the multiplicand. Booth-2 uses a radix-4 encoding which reduces the number of the partial products to $n' = \left\lceil \frac{n+1}{2} \right\rceil$. Moreover, the digit set $\{\bar{2}, \bar{1}, 0, 1, 2\}$ is easily obtained by shifting and/or complement operations, and for this reason, many multipliers are based on this scheme. The Booth-3 scheme is based on radix-8 encoding to reduce the number of the partial products to $n' = \left\lceil \frac{n+1}{3} \right\rceil$. All digit sets $\{\bar{4}, \bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3, 4\}$ are also obtained by simple shifting and complement operations, except the term 3**X** (referred to as hard multiple) which is computed by an adding and shifting operation, 3**X**=2**X**+**X**; $\overline{3\mathbf{X}}$ can be generated by complement 3**X**. Some commercial processors as Fchip[5], Alpha RISC[6], IBM S/390[7], Alpha RISC[8], IA-32[9] and AMD-K7[10] have selected the Booth-3 scheme to reduce the counter tree of partial products and to increase the speed of the multiplier. Other circuits as Goldschmidt's division algorithm with IEEE rounding[11] and adaptative FIR filter[12] are based on Booth-3 to reduce area, power and latency.

	Radix-2 (p=1)	**Radix-4 (p=2)**	**Radix-8 (p=3)**	**Radix-16 (p=4)**	**Radix-32 (p=5)**
n=2	2	1	1	1	1
n=4	4	3	2	1	1
n=8	8	5	3	3	2
n=16	16	9	6	5	4
n=32	32	17	11	9	7
n=64	64	33	22	17	13

Table 1. Word length (n') of minimum redundancy signed representation for different radix and values of an n-bit binary number.

The "bottleneck" of radix-8 multiplier architecture is the generation of 3**X**. This term must be computed, generally by an adder, before the partial product producing an increase to the latency of multiplier. In pipelined radix-8 multipliers[5, 11], 3**X** is generated in the first stage in parallel with booth-3 encoding; any interested reader can find a detailed description of radix-8 CMOS S/390 pipelined multiplier in [14,15]. A solution for non-pipelined multipliers is the hybrid radix-4/radix-8 architecture presented in [16]. In this scheme, radix-4 and radix-8 partial products are performed in parallel, reducing by 13% the power with a 9% increase in delay, as compared with a radix-4 implementation. Another idea based on partially redundant partial products with bias constant has been proposed in [17]. It uses a series of small-length adders with no carry propagation and one compensation constant must be added. However, a design tradeoff must be resolved. Radix-8 encoding solution based on redundant logic to eliminate the 3**X** computing is presented in [18], although parameters as speed or area are not given or compared. In other special architectures, as described in the design of filter FIR[13], the 3**X** is pre-computed off the critical path resulting in a fast and low power multiplier.

This paper presents some simplified algebraic expressions of 3**X** operation, resulting in more efficient circuits in terms of area and speed in comparison with those whose implementations are based on conventional adders. To do this, two signals, H_i and K_i, functionally equivalent to two carries, are introduced. These signals are computed in parallel reducing the critical path of the circuit and minimizing the hardware implementation. Three architectures based on different schemes (serial, carry look-ahead (CLA) and parallel prefix) have been proposed and compared with conventional ones using a standard cell CMOS library. The results show a reduction in delay of 67% for serial scheme, 20% for CLA scheme and 26% for parallel prefix. Important reductions in area are also achieved for both.

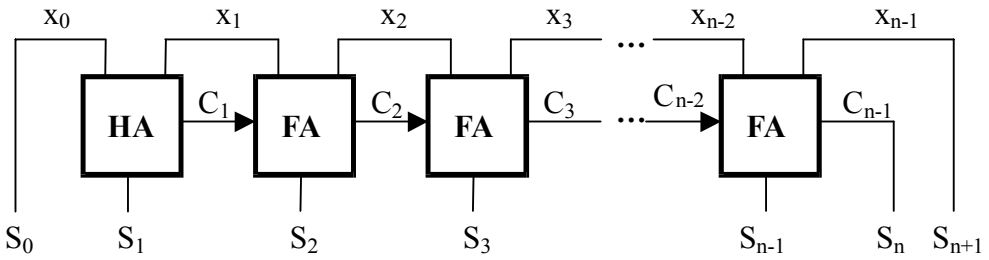


Fig. 1. Conventional adder to generate 3**X** as 2**X**+**X**.

2. SERIAL ADITTION

Let **X**=(x_0, x_1,...,x_{n-1}) be a binary number of n-bit in two's complement and **S**=(S_0, S_1,...,S_{n+1}) the result in n+2-bit of performing the 3**X** operation. A trivial way to generate 3**X** is to add 2**X**+**X** as shown in Figure 1. In this circuit, $S_0=x_0$, $S_n=C_{n-1}$ and the sign of S are directly defined by x_{n-1} ($S_{n+1}=x_{n-1}$) and, thus, sign extension is not necessary. The 2**X**+**X** operation means that the adjoining FA share the same input variable. This characteristic allows the algebraic expressions of the adders to be simplified in order to obtain area and speed-efficient circuits.

In the full adder (FA) of Figure 1, the sum (S_i) and carry (C_i) output are defined as $S_i=x_i \oplus x_{i-1} \oplus C_{i-1}$ and $C_i=x_i x_{i-1}+(x_i+x_{i-1})C_{i-1}$, respectively. Developing the expressions of this circuit and then grouping together terms, it is verified that C_i can be defined as:

$$\begin{aligned}
C_1 &= x_1\, x_0 \\
C_2 &= x_1\, x_0 + x_2\, x_1 \\
C_3 &= x_3(x_2 + x_1\, x_0) + x_2\, x_1 \\
C_4 &= x_3(x_2 + x_1\, x_0) + x_4(x_3 + x_2\, x_1) \\
C_5 &= x_5(x_4 + x_3(x_2 + x_1\, x_0)) + x_4(x_3 + x_2\, x_1) \\
C_6 &= x_5(x_4 + x_3(x_2 + x_1\, x_1)) + x_6(x_5 + x_4(x_3 + x_2\, x_1))
\end{aligned} \tag{2}$$

Thus, C_i can be expressed in terms of two variables, H_i and K_i , defined by means of the following recursive relations:

$$H_i = \begin{cases} x_i H_{i-1} & \text{for i odd} \\ x_i + H_{i-1} & \text{for i even} \end{cases} \tag{3}$$

$$K_i = \begin{cases} x_i + K_{i-1} & \text{for i odd} \\ x_i K_{i-1} & \text{for i even} \end{cases} \tag{4}$$

where i=1,2,….,n-1, $H_0=x_0$ and $K_0=0$. These signals have the following properties

$$\begin{cases} H_i \subset K_i \Rightarrow H_i \overline{K}_i = 0,\ H_i K_i = H_i \text{ and } H_i + K_i = K_i, & \text{for i odd} \\ K_i \subset H_i \Rightarrow \overline{H}_i K_i = 0,\ H_i K_i = K_i \text{ and } H_i + K_i = H_i & \text{for i even} \end{cases} \tag{5}$$

From (1)-(5), C_i can be expressed in terms of H_i and K_i in the following way:

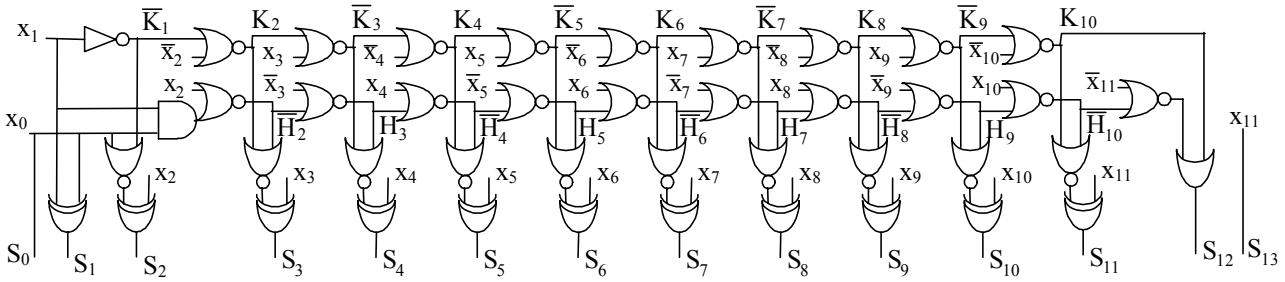


Fig. 2. Ripple carry implementation of 3**X**.

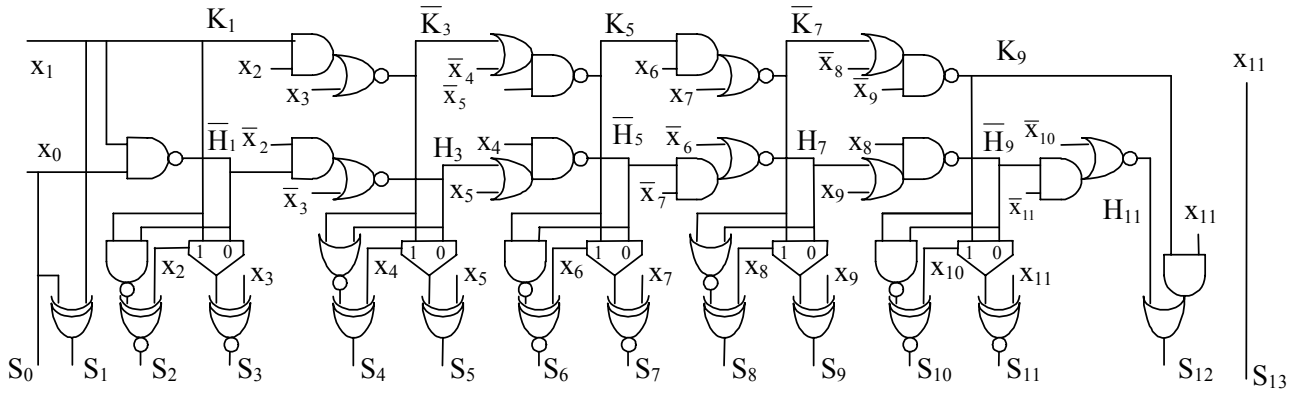


Fig. 3. Fast ripple carry implementation of 3**X**

$$C_i = \begin{cases} H_i + K_{i-1} = x_i H_{i-1} + K_{i-1} = H_{i-1}(x_i + K_{i-1}) = H_{i-1}K_i & \text{for i odd} \\ H_{i-1} + K_i = H_{i-1} + x_i K_{i-1} = K_{i-1}(x_i + H_{i-1}) = H_i K_{i-1} & \text{for i even} \end{cases} \tag{6}$$

The output sum S_i can be directly obtained from H_i and K_i without it being necessary to generate C_i. We get:

$$S_i = x_i \oplus x_{i-1} \oplus C_{i-1} = \begin{cases} x_i \oplus \left(\overline{x}_{i-1}H_{i-2} + x_{i-1}\overline{K}_{i-2}\right) = x_i \oplus \left(H_{i-1}\overline{K}_{i-1}\right) & \text{for i odd} \\ x_i \oplus \left(x_{i-1}\overline{H}_{i-2} + \overline{x}_{i-1}K_{i-2}\right) = x_i \oplus \left(\overline{H}_{i-1}K_{i-1}\right) & \text{for i even} \end{cases} \tag{7}$$

for i =1,2,…,n-1 and with $K_{-1}=H_{-1}=0$. Eq. (7) can be also transformed applying (5) in

$$S_i = x_i \oplus H_{i-1} \oplus K_{i-1} \tag{8}$$

Figure 2 shows the 3**X** addition implementation derived from equations (3), (4) and (7) for n=12. Note the propagation of H_i y K_i signals are generated in a parallel ripple configuration through the NOR gates with an asymptotic time O(n). A more efficient implementation of this circuit can be made taking advantage of the properties of H_i y K_i. Figure 3 shows a new implementation for n=12 using the expressions derived from Eq. (7) indicated below:

$$\begin{cases} S_i = x_i \oplus \left(\overline{H}_{i-1} K_{i-1}\right) \\ S_{i+1} = x_{i+1} \oplus \left(\overline{x}_{i-1} H_{i-1} + x_{i-1} \overline{K}_{i-1}\right) \end{cases} \tag{9}$$

for i=0,2,4,… . This circuit generates simultaneously the S_i and S_{i+1} outputs from H_{i-1} and K_{i-1} signals and propagates these signal in parallel by means of OR-NAND and AND-NOR gates.

3. CARRY LOOK-AHEAD ADDITION

Adders based on the carry look-ahead principle remain dominant, since the carry delay can be improved by calculating the carries to each stage in parallel. The expressions of H_i and K_i defined in Eq. (3) and (4) are of a similar form to those used in conventional carry look-ahead circuits. For example, H_8 and K_8 are defined as

$$\begin{aligned} H_8 &= x_8 + x_7(x_6 + x_5(x_4 + x_3(x_2 + x_1\, x_0))) \\ K_8 &= x_8(x_7 + x_6(x_5 + x_4(x_3 + x_2\, x_1))) \end{aligned} \tag{10}$$

The propagate and generate signals of conventional adders are replaced in Eq. (10) by the input variables themselves. H_i "propagates" input variable x_0 and K_i "propagates" x_1. Thus, the most commonly used schemes for accelerating carry based on domino carry look-ahead, multi-level carry look-ahead and carry-skip circuits can be used directly to compute the signals H_i and K_i.

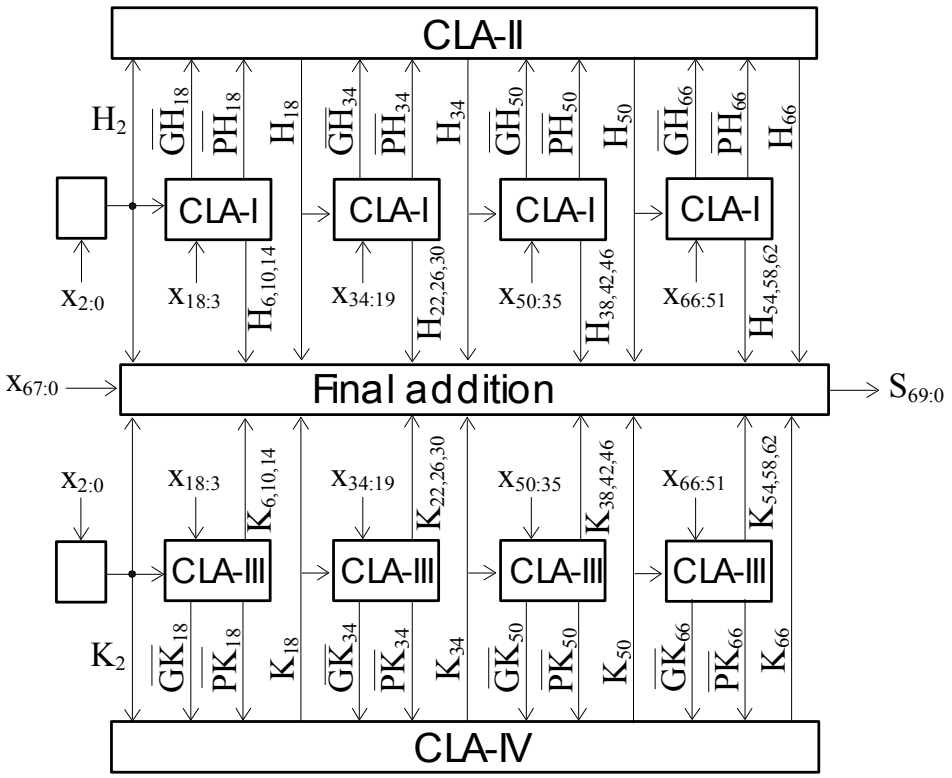
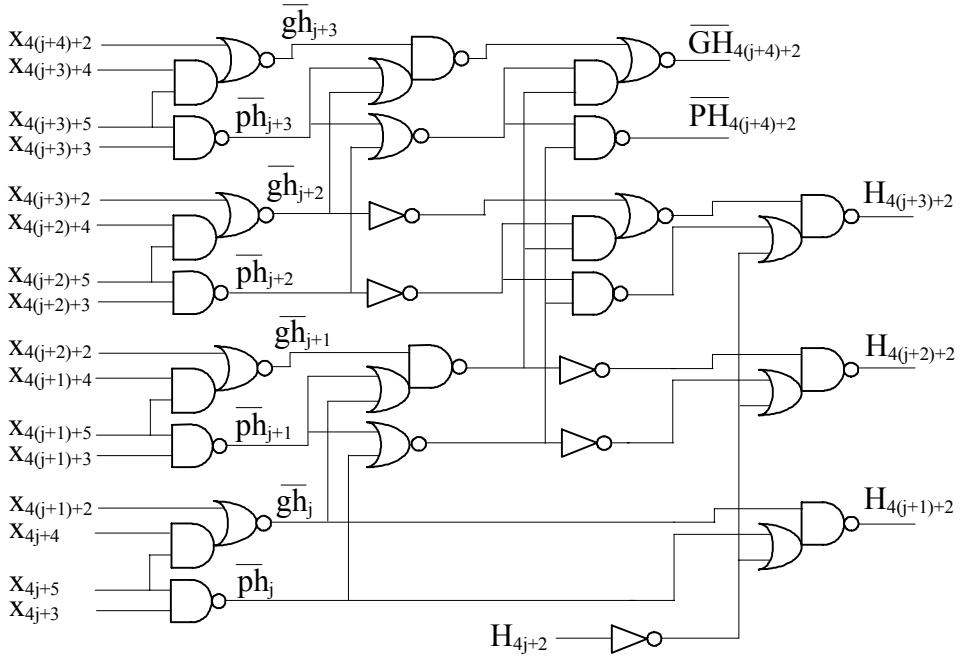
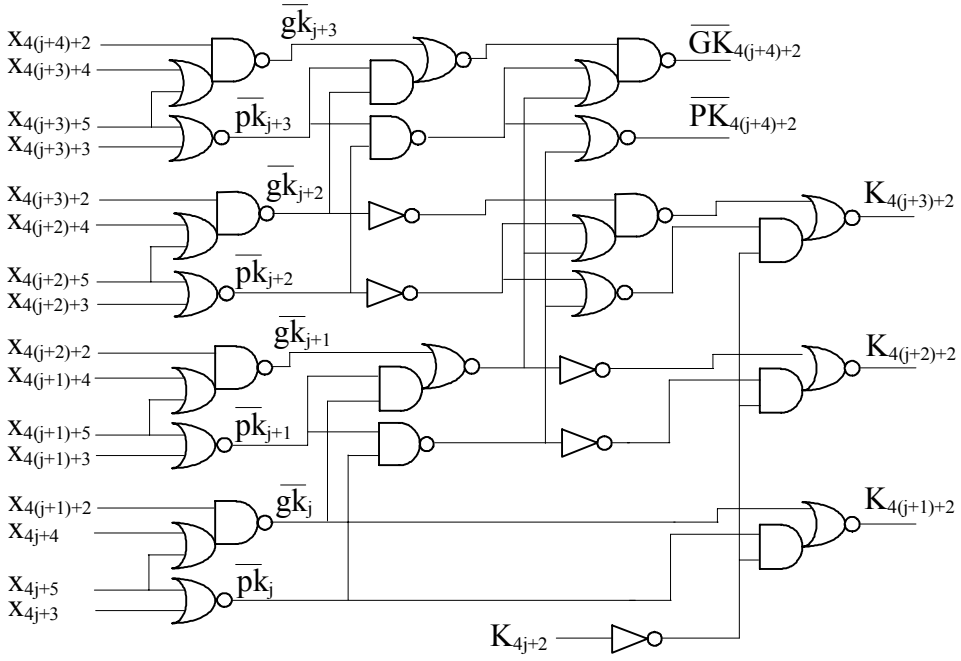


Fig. 4. Two-level carry look-ahead structure of 3**X** for n=68.



a)



b)

Fig.5. Carry look-ahead generator of a) H signals (CLA-I) and b) K signals (CLA-III).

However, the critical path of the carry look-ahead scheme can be significantly reduced by introducing new generate and propagate signals associated to H_i and K_i. Let gh_j be the generate signal and ph_j is the propagate signal of H_i. These signals are defined for a group of 4 inputs by

$$gh_j = x_{4j+2} + x_{4j+1}x_{4j} \quad \text{for} \quad 0 \le j < (n-2)/4 \tag{11}$$

$$ph_j = x_{4j+1}x_{4j-1} \quad \text{if } 1 \le j < (n-2)/4 \tag{12}$$

Then the following equation of recurrence is established

$$H_{4j+2} = gh_j + ph_j H_{4(j-1)+2} \quad \text{for} \quad 1 \le j < (n-2)/4 \tag{13}$$

with $H_2=gh_0$. The number of these signals is reduced to roughly n/4 in comparison with n in a conventional adder. For example, for j=4 we get:

$$H_{18} = gh_4 + ph_4(gh_3 + ph_3(gh_2 + ph_2(gh_1 + ph_1 gh_0))) \tag{14}$$

In a similar way, let gk_j be the generate signal and pk_j the propagate signal of K_{4j+2}. These signals are defined by

$$gk_j = \begin{cases} x_2 x_1 & \text{if } j = 0 \\ x_{4j+2}(x_{4j+1} + x_{4j}) & 1 \le j < (n-2)/4 \end{cases} \tag{15}$$

$$pk_j = x_{4j+1} + x_{4j-1} \quad \text{for } 1 \le j < (n-2)/4 \tag{16}$$

The following equation of recurrence is established

$$K_{4j+2} = gk_j(pk_j + K_{4(j-1)+2}) \quad \text{for} \quad 1 \le j < (n-2)/4 \tag{17}$$

with $K_2=gk_0$. For example, for j=4 we get:

$$K_{18} = gk_4(pk_4 + gk_3(pk_3 + gk_2(pk_2 + gk_1(pk_1 + gk_0)))) \tag{18}$$

Note that the definition of gh_j, ph_j, gk_j and pk_j only allow one H_{4j+2} and one K_{4j+2} signal to be obtained for every four input signals, but it has the advantage of reducing the number of levels in a look-ahead scheme. Figure 4 shows the structure of a 3**X** implementation for n=68 using two-level of 4-bit CLA modules. H_{4j+2} and K_{4j+2} are computing in parallel through CLA-I/II and CLA-III/IV modules, respectively. CLA-I implements the following expressions:

$$\begin{aligned}
H_{4(j+1)+2} &= gh_j + ph_j H_{4j+2} \\
H_{4(j+2)+2} &= gh_{j+1} + ph_{j+1}(gh_j + ph_j H_{4j+2}) \\
H_{4(j+3)+2} &= gh_{j+2} + ph_{j+2}(gh_{j+1} + ph_{j+1}(gh_j + ph_j H_{4j+2})) \\
\overline{PH}_{4(j+4)+2} &= \overline{ph_{j+3} ph_{j+2} ph_{j+1} ph_j} \\
\overline{GH}_{4(j+3)+2} &= \overline{gh_{j+3} + ph_{j+3}(gh_{j+2} + ph_{j+2}(gh_{j+1} + ph_{j+1} gh_j))}
\end{aligned} \tag{19}$$

Figure 5.a shows the schema of a CLA-I module where complementary gates are used to reduce the propagation time. In this circuit, $\overline{PH}_{4(j+4)+2}$ and $\overline{GH}_{4(j+4)+2}$ are the 4-bit group propagate and generate variables, and $H_{4(j+3)+2}$,

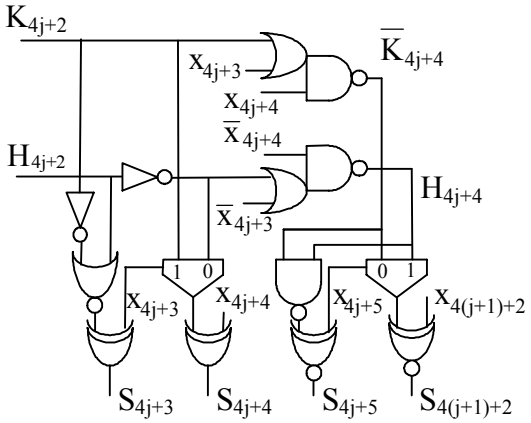


Fig. 6. 4-b adder module.

$H_{4(j+2)+2}$ and $H_{4(j+1)+2}$ are computed when the input signal H_{4j+2} is known. CLA-II generates H_{18}, H_{34}, H_{50} and H_{66} defined as:

$$\begin{aligned}
H_{18} &= \overline{\overline{GH}_{18}(\overline{PH}_{18} + \overline{H}_2)} \\
H_{34} &= \overline{\overline{GH}_{34}(\overline{PH}_{34} + \overline{GH}_{18}(\overline{PH}_{18} + \overline{H}_2))} \\
H_{50} &= \overline{\overline{GH}_{50}(\overline{PH}_{50} + \overline{GH}_{34}(\overline{PH}_{34} + \overline{GH}_{18}(\overline{PH}_{18} + \overline{H}_2)))} \\
H_{66} &= \overline{\overline{GH}_{66}(\overline{PH}_{66} + \overline{GH}_{50}(\overline{PH}_{50} + \overline{GH}_{34}(\overline{PH}_{34} + \overline{GH}_{18}(\overline{PH}_{18} + \overline{H}_2))))}
\end{aligned} \tag{20}$$

In a similar way as for K signals, CLA-III constitutes the first level of computation defined by the following expression:

$$\begin{aligned}
K_{4(j+1)+2} &= gk_j(pk_j + K_{4j+2}) \\
K_{4(j+2)+2} &= gk_{j+1}(pk_{j+1} + gk_j(pk_j + K_{4j+2})) \\
K_{4(j+3)+2} &= gk_{j+2}(pk_{j+2} + gk_{j+1}(pk_{j+1} + gk_j(pk_j + K_{4j+2}))) \\
\overline{PK}_{4(j+4)+2} &= \overline{pk_{j+3} + pk_{j+2} + pk_{j+1} + pk_j} \\
\overline{GK}_{4(j+3)+2} &= \overline{gk_{j+3}(pk_{j+3} + gk_{j+2}(pk_{j+2} + gk_{j+1}(pk_{j+1} + gk_j)))}
\end{aligned} \tag{21}$$

Figure 5.b shows the schema of CLA-III. The CLA-IV implements these switching functions:

$$\begin{aligned}
K_{18} &= \overline{\overline{GK}_{18} + \overline{PK}_{18}\overline{K}_2)} \\
K_{34} &= \overline{\overline{GK}_{34} + \overline{PK}_{34}(\overline{GK}_{18} + \overline{PK}_{18}\overline{K}_2))} \\
K_{50} &= \overline{\overline{GK}_{50} + \overline{PK}_{50}(\overline{GK}_{34} + \overline{PK}_{34}(\overline{GK}_{18} + \overline{PK}_{18}\overline{K}_2)))} \\
K_{66} &= \overline{\overline{GK}_{66} + \overline{PK}_{66}(\overline{GK}_{50} + \overline{PK}_{50}(\overline{GK}_{34} + \overline{PK}_{34}(\overline{GK}_{18} + \overline{PK}_{18} + \overline{K}_2))))}
\end{aligned} \tag{22}$$

The final addition module computes the output S in 4-bit modules according to the following expressions

$$
\begin{aligned}
S_{4j+3} &= x_{4j+3} \oplus (H_{4j+2}\overline{K}_{4j+2}) \\
S_{4j+4} &= x_{4j+4} \oplus (x_{4j+3}\overline{H}_{4j+2} + \overline{x}_{4j+3}K_{4j+2}) \\
S_{4j+5} &= x_{4j+5} \oplus (H_{4j+4}\overline{K}_{4j+4}) \\
S_{4(j+1)+2} &= x_{4(j+1)+2} \oplus (x_{4j+5}\overline{H}_{4j+4} + \overline{x}_{4j+5}K_{4j+4})
\end{aligned}
\tag{23}
$$

for $0 \le j < (n-6)/4$. This equation is implemented in the circuit of Figure 6. S_{4j+3} and S_{4j+4} are computed from H_{4j+2} and K_{4j+2}, and S_{4j+5} and $S_{4(j+1)+2}$ from H_{4j+4} and $\overline{K}_{4j+4}$, which are generated in two OR-NAND gates. Note that the complexity of this circuit is lower than a conventional 4-b adder. In the final addition, the first 3 bits of S are defined as

$$
\begin{aligned}
S_0 &= x_0 \\
S_1 &= x_1 \oplus x_0 \\
S_2 &= x_2 \oplus (x_1\overline{x}_0)
\end{aligned}
\tag{24}
$$

and the last 3 bits are

$$
\begin{aligned}
S_{n-1} &= x_{n-1} \oplus (H_{n-2}\overline{K}_{n-2}) \\
S_n &= H_{n-2}x_{n-1} + K_{n-2} \\
S_{n+1} &= x_{n-1}
\end{aligned}
\tag{25}
$$

4. PARALLEL PREFIX ADDITION

The associative property of the well-known concatenation operator "○" introduced by Brent and Kung[19,20] for prefix adders allows the ripple configuration to be transformed into a parallel binary tree structure to make high-speed addition. As a result, these adders have a structure, which is very adequate for VLSI. The similitude between the expressions for conventional adders and the expressions of H_{4j+2} and K_{4j+2} described in Eq. (11)-(17) allow this operator to be applied to these signals. The operator ○ associated to H_{4j+2} can be defined as

$$(gh, ph)\circ(gh', ph')=(gh+ph\ gh', ph\ ph') \tag{26}$$

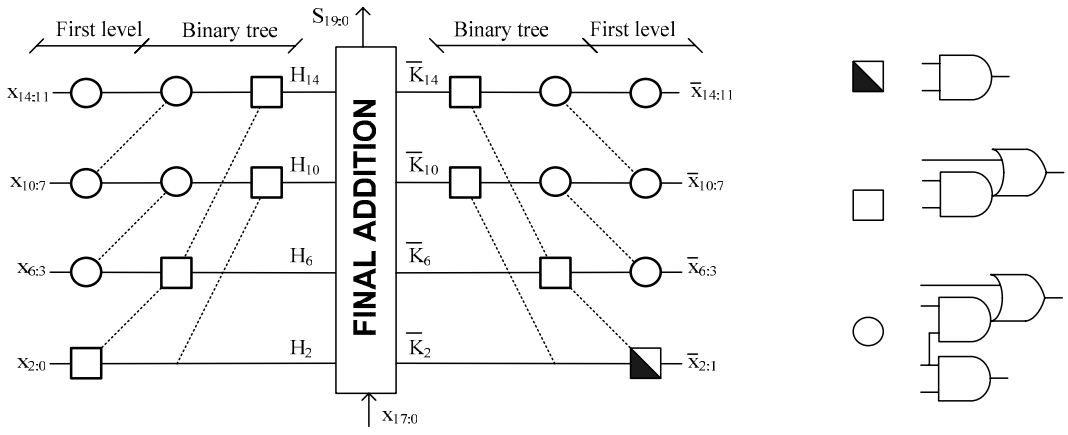


Fig.7. Parallel prefix scheme for n=18.

and the operator ● associated to K_{4j+2} as

$$(gk, pk) \bullet (gk', pk') = (gk\ (pk+gk'),\ pk+pk') \tag{27}$$

Figure 7 shows the circuit to compute S=3**X** for n=18 based on a parallel prefix scheme. This circuit is made up of two binary parallel structures to generate the $\overline{H}_{4j+2}$ and $\overline{K}_{4j+2}$ signals (j=0,1, 2, 3) and final addition to obtain S similar to those described in eq. (23)-(25). Each binary structure has a first level to compute the gh_j, ph_j, gk_j and pk_j signals and a binary tree to get $\overline{H}_{4j+2}$ and $\overline{K}_{4j+2}$; complementary gates are used to reduce the propagation time.

5. SIMULATION AND COMPARISONS

For the purposes of comparison, 3**X** implementations based on conventional adders have been compared with the proposed circuits. The designs have been made in a standard-cell methodology using a 0.35μm CMOS 3M technology. The CADENCE Silicon Ensemble Place&Route tool has been used to include wire delays since these are important in this technology. Table I lists the number of cells, area and maximum delay of the following 3**X** implementations for different values of n: ripple carry adder (RCA) based on full-adder (FA) and half-adder (HA) cells, CLA adder and Kogge-Stone[20] parallel prefix adder (PPA) with $\log_2 n$ levels, and the proposed circuits in Figure 3 (RCAHK), Figure 4 (CLAHK) and Figure 7 (PPAHK).

The results in Table II highlight the advantages of the circuits proposed in terms of speed and area. The RCAHK reduces by roughly 67% the delay with respect to the RCA without any additional area cost. This result is important for some adders such as the carry-select adder and the conditional-sum adder, which are based on ripple carry adders. The CLAHK reduces the area by 17% and the delay by 20% with respect to conventional CLA. The Kogge-Stone PPA is one of the fastest adders, which can be built with standard cells. The PPAHK improves its speed by 26% and its area by 46%. As a result, Table II demonstrates that the expressions developed to implement 3**X** addition lead to circuits whose area and speed are significantly improved in comparison with implementations based on classical adders.

		8-bit	16-bit	32-bit	64-bit
RCA	**Delay**	3.73 ns	7.82 ns	16.29 ns	33.9 ns
	Cells	8	16	32	64
	Area	2493 μm^2	5405 μm^2	11229 μm^2	23131 μm^2
RCAHK	**Delay**	1.37 ns	2.72 ns	5.34 ns	11.1 ns
	Cells	28	60	124	256
	Area	2439 μm^2	5252 μm^2	10738 μm^2	21286 μm^2
CLA	**Delay**	1.84 ns	2.49 ns	3.23 ns	4.07 ns
	Cells	53	106	239	502
	Area	3956 μm^2	8244 μm^2	18346 μm^2	38384 μm^2
CLAHK	**Delay**	1.14 ns	1.63 ns	2.49 ns	3.27 ns
	Cells	31	81	189	411
	Area	2857 μm^2	6607 μm^2	14815 μm^2	31741 μm^2
PFA	**Delay**	1.67 ns	2.26 ns	3.12 ns	3.84 ns
	Cells	51	137	343	821
	Area	4168 μm^2	11011 μm^2	26081 μm^2	64610 μm^2
PFAHK	**Delay**	1.14 ns	1.63 ns	2.20 ns	2.86 ns
	Cells	31	81	194	450
	Area	2857 μm^2	6607 μm^2	14961 μm^2	35072 μm^2

Table II. Comparisons for different implementations of 3**X**.

CONCLUSIONS

The generation of the term 3**X** used in radix-8 encoding can be efficiently implemented using two signals, H_i and K_i , functionally equivalent to two carries. H_i and K_i are computed in parallel using architectures which are efficient in terms of delay and area when compared with implementations based on conventional adders. Simulations made in circuits implemented using a standard-cell have demonstrated that the expressions developed to implement 3**X** reduce the delay of serial scheme by 67%, the carry look-ahead scheme by 20% and the parallel prefix scheme by 26%. Important reductions in area are also achieved for both. Other schemes used for accelerating carry based on transistor structures such as domino carry look-ahead, multi-level carry look-ahead and carry-skip circuits can be directly used to compute the signals H_i and K_i. These circuits allow high-speed computation and they can even lead to a reduction in the latency of the multiplier.

ACKNOWLEDMENTS

This research has been supported by funds of the Spanish Ministry of Science and Technology (TIC2006-12438).

REFERENCES

1. B. Parhami, 'Computer arithmetic, algorithms and hardware' (Oxford University Press (New York, Oxford, 2000).
2. M.D. Ercegovac, T. Lang, 'Digital arithmetic' (Morgan Kaufmann Publishers, 2004).
3. Avizienis, A., "Signed-digit Number representation for fast parallel Arithmetic," IRE Trans. Electronic Computers, 10, 389-400 (1961)
4. O.L. MacSorley, "High-speed arithmetic in Binary computers", Proc. of the IRE, (49) 1, 67-71 (Jan 1961)
5. B.J. Benschneider et al, "A pipelined 50MHz CMOS 64-bit floating-point arithmetic processor," IEEE J. of Solid-State Circuits, (24) 5, 1317-1323, (October 1989)
6. D.W. Dobberpuhl et al, "A 200-MHz 64-b Dual-Issue CMOS microprocessor," IEEE J. of Solid-State Circuits, vol. (27) 11, 1555-1567 (November 1992)
7. C.F. Webb et al, "A 400-MHz s/390 microprocessor," IEEE J. of Solid-State Circuits, (32) 11, 1665-1675 (November 1997)
8. J. Clouser, M. Matson, R. Badeau, R. Dupcak, S. Samudrala, R. Allmon and N. FairBanks, "A 600-MHz superscalar floating-point processor," IEEE J. of Solid-State Circuits, (34) 7, 1026-1029 (July 1999)
9. R. Senthinathan, S. Fischer, H. Rangchi and H. Yazdanmehr, "A 650-MHz, IA-32 Microprocessor with Enhanced data Streaming for Graphics and Video," IEEE J. of Solid-State Circuits, (34) 11, 1454-1465 (November 1999)
10. A. Scherer, M. Golden, N. Juffa, S. Meier, S. Oberman, H. Partovi, F. Weber, "An out-of-order tree-way superscalar multimedia floating point unit," 1999 IEEE Int. Solid-State Circuits Conf., 94-95 (1999)
11. G. Even and P.M. Seidel, "Pipelined multiplicative division with IEEE rounding," Proc. 21st Int. Conf. On Computer Design, 240-245 (2003)
12. S. Haijun, S. Zhibiao, Z. Gang and Z. Ning, "The research on optimization techniques of 32-bit floating-point RISC microprocessor," Proc. 2005 IEEE Int.Workshop on VLSI Design and Video Techn., 63-66 (May 2005)
13. K. Muhammad, R. B. Staszewski and P.T. Basala, "Speed, power, area and latency tradeoffs in adaptive FIR filtering for PRML read channel," IEEE Trans. on VLSI Systems, (9) 1, 42-51 (Feb 2001)
14. E.M. Schwarz, B. Averill and L. Sigal, "A radix-8 CMOS S/390 multiplier," Thirteenth Symp. On Compuer Arithmetic, Asilomar, CA, 2-9 (July 1997)

15. E.M. Schwarz, B. Sigal and T.J. McPherson, “CMOS floating-point unit for the S/390 parallel enterprise server G4,” IBM J. Res. Develop., (41) 4/5, 475-488 (July/Sept. 1997)
16. B.S. Cherkauer and E.G. Friedman, “A hybrid radix-4/radix-8 low power signed multiplier architecture,” IEEE Trans. On Circuits and Systems-II: Analog and Digital Signal Processing, (44) 8, 656-659 (August 1997)
17. M.J. Flynn and S.F. Oberman, Avanced Computer Arithmtic Design, John Wiley & Sons, Inc, 2001.
18. N. Besli and R.G. Deshmukh, “A 54x54-bit multiplier with a new redundant binary booth’s encoding,” Proc. of the 2002 IEEE Canadian Conference on Electrical&Computing Engineering, 597-602 (2002)
19. R.P. Brent and H.T. Kung, “A regular layout for parallel adders,” IEEE Transactions on Computers, (C-31) 3, 260-264 (March 1982)
20. P.M. Kogge and H.S. Stone, “A parallel algorithm for the efficient solution of a general class of recurrence equations,” IEEE Transactions on Computers, (C-22) 8, 783-791 (August 1973).