THE LOGIC OF MODULO $2^k + 1$ ADDERS

by

Jean P. Chinal

Abstract

The design of modulo $2^k + 1$ adders for arbitrary k is considered, with the objective of achieving a logic structure as regular as possible so as to allow a convenient implementation in large-scale integration technology (LSI). It is shown how the design problem can be reduced to the recursive generation of a subtract signal and to the merging, in various degrees, of the corresponding logic with the logic of an ordinary adder or, alternately, of a so-called signed-carry adder which is defined and designed itself in general, with both recursive and explicit carry schemes. Modulo $2^k + 1$ adder designs are given, one with conventional adder, another based on signed-carry adder and a third, derived from the signed-carry scheme, where subtract signal generation and carry logic are merged. This last scheme can be set up with two backward recursion chains and five or six forward ones. Two more basic variants are finally indicated for this integrated scheme, aiming at reducing as much as possible the residual logic structure irregularity presented by the most significant position in the word.

1. Introduction

Modulo m adders are basic to error-coded arithmetic [1,2] for detection or correction of faults in digital processors, for operand or check word arithmetic [3,4,5] or for the design of residue generators [6]. Also they are used in binary-coded residue class arithmetic [7,8]. For multiple-residue error checking [8,9] and for residue class arithmetic it is necessary to implement modulo m addition for several distinct moduli having some mutual relationships such as being mutually prime [8]. Moduli of the form $2^k - 1$ or $2^k + 1$ are often advocated, owing to the simplicity of their implementation. Both are very convenient for residue generation of an arbitrary binary number owing to the possibility of dividing the number in chunks of length k and using tree networks [6]. Also modulus $2^k - 1$ addition is closely related to double zero digit-complement addition [10,11,12] and is identical to non-overflowed, (single) negative-zero digit complement arithmetic [13].

Additionally, modulus 5 adders and generators appears in various hardwired binary to decimal schemes [11], in BCD arithmetic [11] and modulus 9 comes into play for BCD arithmetic checking. Finally large moduli of the form $2^k + 1$ may be used in conjunction with modulus $2^k - 1$ in multiple checking or residue arithmetic [8,9] so as to provide pairs of moduli that would be at the same time, different from 2^k, very close in value and mutually prime [8,9].

In the search for moduli that would lead to logically "simple" modulo m adders, those of the form $2^k + 1$ appear the most interesting ones after the lowcost [3,4] moduli of the $2^k - 1$ type and will be investigated here. For comparison with the better known $2^k - 1$ modulus, the logic for this latter case will be derived anew, using the same type of approach that will be used for the $2^k + 1$ case.

2. Two-level and one-level arithmetic for modulo m adders

Let x and y be two modulo m residues, coded in binary with k bits and let $|x+y|_m$ denote the modulo m sum of x and y, i.e. the modulo m residue of the ordinary sum x+y. The network computing $|x+y|_m$ will be called a (restricted) modulo m adder. Unrestricted adders might similarly be used as a name for adders that would accept any numbers x and y (not necessarily residues already) and compute $|x+y|_m$, but will not be considered here for the moment. Then we can write x, y and m (modulus) as pure binary integers.

$$x = 0\,x_{k-1} \cdots x_i \cdots x_0 \qquad m_i, x_i, y_i \in \{0,1\}$$
$$y = 0\,y_{k-1} \cdots y_i \cdots y_0 \qquad 0 \le x, y < m \tag{2.1}$$
$$m = 0\,m_{k-1} \cdots m_i \cdots m_0 \qquad m_{k-1} = 1 \;(2^{k-1} \le m < 2^k)$$

where the assumption on m means that it is coded with k digits and no less. Then [14]

$$|x+y|_m = x + y - v(t)\,m \tag{2.2}$$
$$t = \overline{sign}\,(x+y-m) \tag{2.3}$$

where v(u) is the arithmetic value of boolean u(1 for 1 and 0 for 0), sign X is the boolean sign [15,16] of X, equal to the leftmost, $(k+1)^{th}$ bit in 2's complement representation [15,16] (here sign x, sign y and sign m are zero, but have been explicitly represented in (2.1).

Equations (2.2) and (2.3) defined a two-level arithmetic procedure which is widely used in BCD arithmetic, iterative division networks [17] and in division algorithms based on the comparison method [14,15,16,22]. If left as such it can be implemented for instance with a sign detection network and an adder. If the sign detection function is a subfunction of a two-adder network computing first x+y-m, then +m may have to be restored by a third adder and this is analogous to the restoration method for a division step [14,15,22]. In the following however we will derive explicit expressions for the subtract signal t. A first solution will be to use t as a gating signal to perform a conditional addition of the classical type. Another is to combine the logic of t with the adder logic to be subsequently used so as to obtain a scheme where sign-detection and adding step are fully intermeshed in what may how be called one level solution. Two level solutions for $2^k - 1$ moduli are well known for any k and one level solutions have been reported for small specific values of k [6,7] and for arbitrary k (e.g. [13]). Similarly two-level solutions are used in BCD arithmetic [11] (moduli 5, and 10) and one level solutions, previously reported for small k's only [6], will be given for arbitrary k in this paper. Basic notations are defined in the appendix.

3. Sign-detection logic

For an arbitrary modulus m, subject to assumptions of §.1, if we call r_i and r'_i the carries generated, respectively, in addition x+y and in subtraction (x+y)-m, then it can be shown [14] that

$$\text{sign}(x+y-m) = maj(o, \bar{o}, r'_{k+1})$$
$$= r'_{k+1}$$
$$= maj(\bar{r}_k, o, r'_k)$$
$$= \bar{r}_k r'_k \qquad (3.0)$$

where r_k and r'_k can be expressed in turn as functions of x_i , y_i and m_i variables to determine an explicit expression of t. Such computation will be presented below for moduli $2^k -1$ and $2^k +1$.

3.1 Sign-detection logic for modulus $2^k -1$

Let us write

$$S = x+y \qquad (3.1)$$

Then

$$S = r_k s_{k-1} \dots s_i \dots s_o \qquad (3.2)$$

with (cf. appendix)

$$\begin{cases} s_i = x_i \oplus y_i \oplus r_i = P_i \oplus r_i & (r_o = o) \\ r_{i+1} = G_i^1 + P_i r_i \end{cases} \qquad (3.3)$$

Similarly, denote by S' the sum x+y-m and s'_i its digits

$$S' = S - m \qquad (3.4)$$

$$S' = \dots s'_{k+1} s'_k \dots s'_i \dots s_o \qquad (3.5)$$

Then s'i and r'i are the sum and carry digits of a subtraction [15,16] given by

$$\begin{cases} s'_i = s_i \oplus m_i \oplus r'_i & (r'_o = o) \\ r'_{i+1} = maj(\bar{s}_i, m_i, r'_i) \end{cases} \qquad (3.6)$$

For modulus $2^k -1$, and for positions 0 to k-1 inclusive

$$m_i = 1$$

and by recursively applying (3.6) or using an explicit carry expression [11,14,16,22] we get

$$r'_k = \overline{\sum_{j=o}^{k-1} s_j}$$
$$= \overline{\prod_{j=o}^{k-1} s_j}$$
$$= \prod_{j=o}^{k-1} P_j$$
$$= P_{o,k-1} \qquad (3.7)$$

Whence

$$t = \overline{G_{o,k-1}}\ \overline{P_{o,k-1}}$$
$$= G_{o,k-1} + P_{o,k-1}$$
$$= T_{o,k-1} \qquad (3.8)$$

This last term is the transmitted carry [16] for block [0,k-1] , i.e. the value of output carry r_k for an input carry r_0 forced to 1. Usually in cyclic carry adders only the generated carry $G_{0,k-1}$ (value of r_k for r_0 forced to zero) is used. The result will remain equal to $|x+y|_{2^k-1}$ modulo 2^k-1, but, when x+y equals $2^k -1$, will be left as such and will thus not be a residue.

3.2 Sign-detection logic for modulus $2^k +1$

Assume again x and y to be residues. They may require k+1 digits to be represented so that we will extend x, y and m representations one more position to the left and apply (3.0) for k+1 instead of k. Using the same symbols S, S' , s_i , s'_i , r_i r'_i for that new case, we have

$$\begin{cases} x = 0\, x_k\, x_{k-1} \dots x_i \dots x_o \\ y = 0\, y_k\, y_{k-1} \dots y_i \dots y_o \\ m = 0\, 1\ 0\ \dots 0 \dots 1 \end{cases} \qquad (3.9)$$

$$x+y = r_{k+1}\, s_k\, s_{k-1} \dots s_i \dots s_o \quad (r_i \text{ carries}) \qquad (3.10)$$

$$x+y-m = \dots s'_k s'_{k-1} \dots s'_i \dots s'_o \quad (r'_i \text{ carries}) \qquad (3.11)$$

Then carries generated in positions [0,k-1] are decrementation carries [11,12,15] and

$$r'_k = \overline{\prod_{j=o}^{k-1} s_j} \qquad (3.12)$$

$$r'_{k+1} = maj(\bar{s}_k, 1, r'_k) \qquad (3.13)$$
$$= \bar{s}_k + r'_k$$
$$= \bar{s}_k + \overline{\prod_{j=o}^{k-1} s_j} \qquad (3.14)$$

Also

$$r_{k+1} = G_{o,k}$$
$$= G_k + T_k r_k$$

but if (x_k , y_k) equals one of the pairs (0,1) (1,0) (1,1) then carry r_k is zero as the first digits of x or y or both will be zero. Thus

$$T_k r_k = o$$
$$r_{k+1} = G_k$$

Finally

$$\text{sign}[x+y-(2^k+1)] = \overline{r_{k+1}}\, r'_{k+1}$$
$$= \overline{G_k}(\bar{s}_k + r'_k) \qquad (3.15)$$
$$t = G_k + s_k \overline{r'_k} \qquad (3.16)$$

Note : If x and y are not constrained to be residues, then $G_{0,k}$, and not G_k should be used in (3.16).

4. Signed-carry adders for modulo $2^k +1$ arithmetic

Three solutions will be described corresponding to increasing integration of the subtract signal t with the adder logic. The least integrated scheme is an ordinary word-length adder with an iterative correction network on its output. In the second solution, ordinary full adder stages are still apparent (although not a word-length adder) and are supplemented with a correction network feeding signals between stages. In the last and most integrated design, a completely specialized adder is evidenced, with a bidirectional iterative structure.

4.1 Adder with output correction network

In this scheme the ordinary addition is performed by a k position adder. A correction network is applied to outputs s_i ($0 \le i \le k$), computes t according to (3.16) and, using t as a gating signal, conditionally subtracts $2^k + 1$. The latter operation is itself a decrement operation for positions 0 to k-1 inclusive and a special-purpose logic for rank k. Thus

$$S' = x + y - v(t) \tag{4.1}$$

$$s'_i = s_i \oplus t r'_i \qquad (0 \le i \le k-1) \tag{4.2}$$

$$s'_k = s_k \oplus t \oplus t r'_k = s_k \oplus t \overline{r'_k} \tag{4.3}$$

$$t r'_i = \left[G_k + s_k\left(\sum_{j=0}^{k-1} s_j\right)\right]\left(\prod_{j=0}^{i-1} \overline{s_j}\right) \tag{4.4}$$

$$= \left[G_k + s_k\left(\sum_{j=i}^{k-1} s_j\right)\right]\left(\prod_{j=0}^{i-1} \overline{s_j}\right)$$

$$= \left(G_k + s_k S_i\right) S'_i \quad \left(S_i = \sum_{j=i}^{k-1} s_j \,;\; S'_i = \prod_{j=0}^{i-1} \overline{s_j}\right)$$

Terms S_i and S'_i can be computed recursively from the left and the right respectively (Fig. 1 (a) cells) and combined with Gk , sk and s_i terms (Fig. 1(b) and (c) cells) according to (4.3,4.4).

4.2 Adder with integrated sign-detection and add logic

Both adders in this category use the operation

$$x + y - v(u) \qquad u \in \{0,1\} \tag{4.5}$$

If we have an operator to implement it, then it can be used for the k positions from the right, the k+1 th again being obtained by an ad hoc scheme. This will be obtained by using for u the function t. The procedure here parallels the one used for modulo $2^k - 1$ addition where we perform

$$S' = x + y + v(t) \tag{4.6}$$

However, due to the fact that v (t)has now to be subtracted from x+y rather than added, another logic operator is needed which we will call signed carry adder, owing to the property it will evidence.

5. Signed-carry adders

5.1 Definition

By definition such an adder will compute the sum

$$z = x + y + v(t'_0) \qquad t'_0 \in \{0, 1, \overline{1}\} \tag{5.1}$$
$$(\overline{1} = -1)$$

The terminology and the choice of the domain for t'_0 may be justified as follows : if we try to add, digit by digit, the numbers x, y and -1, the partial sum for position 0 will be in the set $\{-1,0,1,2\}$ and, if we code it in two's complement form,this will produce a carry in the set $\{-1,0,1\}$. Then for the next position the sum of the two digits and this carry will be in the set $\{-1,0,1,2,3\}$ which will again yield a carry in the set $\{-1,0,1\}$. So that, as soon as we assume t'_0 capable of taking the value $\overline{1}$, all interstage carries will have a sign and no complication results from assuming t'_0 in the set $\{\overline{1},0,1\}$ (instead of $\overline{1},0$ only).

5.2 Signed-carry adder stage logic

Let us call in general t'_i the signed carry into position i. We can write :

$$t'_i = (-1)^{v(t''_i)} v(t_i) \qquad t'_i, t''_i \in \{0,1\} \tag{5.2}$$

$$t'_i = -2^1 v(t''_i t_i) + 2^0 v(t_i) \tag{5.3}$$

$$= -2^2 v(t''_i t_i) + 2^1 v(t''_i t_i) + 2^0 v(t_i)$$

The absolute value is $v(t_i)$ and the usual sign is $v(t''_i)$ while the Boolean sign[15] is $t''_i\, t_i$. Then we must have for every rank i :

$$v(x_i) + v(y_i) + t'_i = 2^1 v(t'_{i+1}) + 2^0 v(s_i) \tag{5.4}$$

where s_i will be the sum digit and t'_{i+1} the new carry. This is an equation between two numbers, of which the binary digits must then be equal.

Each side can be calculated with binary addition rules[11,12,15,22] . Denoting by r^j_i the carries for that addition :

$$v(x_i) + v(y_i) + t'_i = -2^2 v(s_i^2) + 2^1 v(s_i^1) + 2^0 v(s_i^0)$$

$$s_i^0 = x_i \oplus y_i \oplus t_i \qquad r'_1 = (x_i \oplus y_i) t_i \tag{5.5, 5.6}$$

$$s_i^1 = maj(x_i, y_i, t_i) \oplus T_i t_i \qquad r'_2 = (x_i + y_i) t''_i t_i$$

$$s_i^2 = \overline{x_i}\,\overline{y_i}\, t''_i t_i$$

$$2^1 v(t'_{i+1}) + 2^0 v(s_i) = -2^2 v(t''_{i+1} t_{i+1}) + 2^1 v(t_{i+1}) + 2^0 v(s_i^0) \tag{5.7}$$

Equating the two sides yields

$$\begin{cases} s_i = x_i \oplus y_i \oplus t_i \\ t_{i+1} = maj(x_i, y_i, t_i) \oplus t''_i t_i \\ t''_{i+1} t_{i+1} = \overline{x_i}\,\overline{y_i}\, t''_i t_i \end{cases} \tag{5.8}$$

The last equation is of the form

$$ax = b$$

and is soluble if[19]

$$\overline{a}\, b = 0$$

$$\left[maj(x_i, y_i, t_i) \oplus \overline{t''_i t_i}\right]\left[\overline{x_i}\,\overline{y_i}\, t''_i t_i\right] = 0$$

which can be seen to be identically true, so that

$$t_{i+1} = \overline{x_i}\,\overline{y_i}\, t''_i t_i + T(x_i, y_i, t''_i, t_i) \tag{5.9}$$

where T is an arbitrary function of x_i , y_i , t''_i and t_i . It is possible to choose T so that, for instance

$$t_{i+1} = t''_i t_i \tag{5.10}$$

Whatever the choice of T,the product $t''_i\, t_i$ meets the simple relationship (cf (5.8) and notations in appendix)

$$t''_{i+1}\, t_{i+1} = G^0_i\, t''_{i+1}\, t_i \qquad (5.11)$$
$$= \left(\prod_{j=0}^{i} G^0_j\right) t''_0\, t_0$$

The logic for the signed-carry adder stage can be written in terms of zero generate (G^0_i) one generate (G^1_i) [16,18] and propagate symbols (P_i) [15,16,22] as follows :

$$s_i = P_i \oplus t_i \qquad (0 \le i < k-1) \qquad (5.12)$$

$$t_{i+1} = G'_i \oplus P_i t_i \oplus t''_i t_i \qquad (5.13)$$

$$t''_{i+1} = t'_i t_i \qquad (t''_{i+1} t'_{i+1} = G^0_i t''_i t'_i) \qquad (5.14)$$

An iterative adder with cells implementing the above equations can be devised (Fig. 2). Each cell is slightly more complicated than an ordinary full-adder stage. But it is also possible to make use of ordinary adder stages, if we separately generate the $t''_i\, t_i$ products, to be used in the t_{i+1} relationship , by means of (5.13). We then get the scheme of Fig. 3. This is then very close in complexity to the scheme that would use an ordinary adder for x+y and a parallel decrementer [11,15] to subtract one. However the recursive computation with the (b) cells determines products of the G^0_i terms (known before the adder starts computing) while in the solution with decrementer the product formed are of the s_i sum digits of the adder.

In general, the above computation is very close to the one that may be used for base -2 adders [19].

5.3 Explicit carry logic for signed-carry adders

We can also express t_{i+1} as an explicit (non-recursive) function of variables G^0_j , G^1_j , P_j in ranks 0 to i inclusive and of t''_0 and t'_0 . Let us denote by $G_{0,i}$, $P_{0,i}$ the ordinary generate and propagate functions for block 0,i-1 and by $G'_{0,i}$ the product of G^0_i terms (zero generate terms) from 0 to i-1 inclusive

$$G'_{0,i} = \prod_{j=0}^{i-1} G^0_j \qquad (5.15)$$

Then

$$t''_{i+1}\, t'_{i+1} = G'_{0,i}\, t''_0\, t_0 \qquad (5.16)$$

$$t_{i+1} = G^1_i \oplus P_i t_i \oplus G'_{0,i-1} t''_0 t_0 \qquad (5.17)$$

We can now use this last formula to substitute t_i , then t_{i-1} etc. until all intermediate t'_j s have been eliminated. It can be seen that we have :

$$t_{i+1} = G_{0,i} \oplus P_{0,i} t_0 \oplus G''_{0,i} t''_0 t_0 \qquad (5.18)$$

with

$$G_{0,i} = G^1_i + P_i G_{0,i-1} \qquad (5.19)$$

$$\overline{G_{0,i}} = G^0_i + P_i \overline{G_{0,i-1}} \qquad (5.20)$$

$$P_{0,i} = P_i P_{0,i-1} \qquad (5.21)$$

$$G''_{0,i} = P_i G''_{0,i-1} \oplus G'_{0,i-1} \qquad (5.22)$$

$$G'_{0,i} = G^0_i G'_{0,i-1} \qquad (5.23)$$

where the first three are formulas [22] for the classical generate and propagate terms and the fourth one results from the elimination scheme.

Expression (5.18) is explicit and only uses three parameters $G_{0,i}$, $P_{0,i}$ and $G''_{0,i}$ (versus $G_{0,i}$ and $P_{0,i}$ for classical adders) and the signed-carry of least significant weight. The $G''_{0,i}$ term can be simplified in (5.22), as

$$G''_{0,i} G'_{0,i} = \left[P_i G''_{0,i-1} \oplus G'_{0,i-1}\right] G^0_i G'_{0,i-1} \qquad (5.24)$$
$$= G^0_i G'_{0,i-1}$$
$$= G'_{0,i}$$
$$\overline{G''_{0,i}}\, G'_{0,i} = 0 \qquad (5.25)$$

Hence, applying this to rank i-1

$$G''_{0,i} = \left(P_i \overline{G'_{0,i-1}}\right)\overline{G''_{0,i-1}} + \overline{P_i}\, G'_{0,i-1} \qquad (5.26)$$

which shows $G''_{0,i}$ to be of the form $A+BG''_{0,i-1}$ where A and B are separately computable. To build an explicit carry adder according to (5.18) it is sufficient to generate $G_{0,i}$, $P_{0,i}$ and $G''_{0,i}$ and combine the terms according to (5.18).
Explicit expressions may be obtained for $G''_{0,i}$ from repeated application of formulas (5.22) or (5.26) in much the same manner as for ordinary formulas (5.19,5.20) and yields terms of a similar complexity. For instance

$$G''_{0,3} = G^0_2 G^0_1 G^0_0 \oplus P_3 G^0_1 G^0_0 \oplus P_3 P_2 G^0_0 \oplus P_3 P_2 P_1 \qquad (5.27)$$

$$G''_{0,2} = G^0_1 G^0_0 \oplus P_2 G^0_0 \oplus P_2 P_1$$

$$G''_{0,1} = G^0_0 \oplus P_1$$

Also, recursion formulas (5.15, 5.19, 5.21, 5.26) allow to build another adder, in which all carry magnitudes are function of the carry sign and magnitude in the rightmost position only ($t''_0\, t_0$). The logic is

$$s_i = P_i \oplus t_i \qquad (0 \le i \le k-1) \qquad (5.28)$$

$$t_{i+1} = G_{0,i} \oplus P_{0,i} t_0 \oplus G''_{0,i} t''_0 t_0$$

with $G_{0,i}$, $P_{0,i}$, $G'_{0,i}$ and $G''_{0,i}$ generated recursively.

6. Iterative modulo 2^k +1 adders

Modulo 2^k +1 adders will have a regular structure for positions 0 to k-1 inclusive and a separate logic for position k. We will consider for the moment only the carry generation in block [0,k-1] (which includes r_k) and the sum generation in [0.k-1] and only later will consider the sum logic of the irregular stage k.

6.1 Adder with single decrement signal generation

Such an adder uses the signal t (3.16) according to (5.13) with :

$$t''_0\, t_0 = t \qquad (6.1)$$

The modulo 2^k +1 adder thus uses, to generate the sum and carries in block [0,k-1] a signed-carry adder to which the subtract signal t is fed as a least significant carry. It is convenient to take both t''_0 and t_0 equal to t. In this solution , the main problem is to generate the signal t. As we do

not have a ordinary adder to provide the ordinary sums s_j from which it could be computed, other logic must be used to obtain t in terms of what is now available as inputs, i.e. the G_i^0 , G_i^1 , P_i terms. Basically the difficult part is to obtain r_k of which t is a simple function (3.16).

Computation of r_k

The term r_k , equal to the product of all $\overline{s_j}$ is one if the result of ordinary addition, with r_0 carry-in at the rightmost position, is all zeros, i.e. if the two numbers, limited to their first k positions, are either the boolean complement of each other ($r_0 = 1$), or the true complement of each other ($r_0 = 0$). The first situation happens if the addition block propagate $P_{0,k-1}$ is one while the second happen if at some place in the two numbers, there is generation of 1, together with propagation on the left of it and generation of zero in all positions on the right. Thus we have a first way of expressing r_k as an explicit function of all positions :

$$r'_k = \sum_{j=0}^{k-1} P_{j+1,k-1}\, G_j^1\, G'_{0,j-1}\, \overline{r_0} + P_{0,k-1}\, r_0 \qquad (G'_{0,-1}=0;\ r_0=0) \qquad (6.2)$$

which may be used for short modules. For long modules however some recursion formula is necessary. To obtain one, consider the two conditions for a block of digits to produce an all-zero block of sum digits with respectively an input carry of 1 and an input carry of zero. For a block $[i,j]$ and a carry-in of 1, the function that signals that condition is the block propagation $P_{i,j}$. For the same block and a carry-in of 1, let us denote the all-zero condition by $R_{i,j}$ and also let us designate by $Z_{i,j}$ the all-zero condition regardless of carry-in. Then we have.

$$Z_{i,j} = R_{i,j}\, \overline{r_i} + P_{i,j}\, r_i \qquad (6.3)$$

where r_i is the carry of ordinary addition. Then

$$r'_k = Z_{0,k-1} \qquad (6.4)$$

$$= R_{0,k-1}\cdot(\overline{0}) + P_{0,k-1}\cdot(0)$$

$$= R_{0,k-1} \qquad (6.5)$$

Now the $R_{0,k-1}$ can be computed recursively, it turns out, either from the left or from the right of the words. For instance, from the left ("backward recursion") we have

$$R_{i,k-1} = R_{i+1,k-1}\, G_i^0 + P_{i+1,k-1}\, G_i^1 \qquad (6.6)$$

$$P_{i,k-1} = P_{i+1,k-1}\, P_i \qquad (6.7)$$

Once r'_k is generated, t_k itself is obtained thru (3.16^k) and may be then connected to the t_0 , t''_0 or t''_0 t_0 inputs of signed-carry adders of Fig. 2.3 and 4. The total propagation length is 2k positions (k positions in each direction), the same as if r_k were computed isolately by a forward recursion (6.18).

6.2 Adder with multiple decrement signal generation

The basic idea for this adder is to recompute separately for each variable position i the fixed term $R_{0,k-1}$ thru an expression adapted to each i, then to look for a recursive generation of such an expression, and finally, to combine it for possible simplification, with the $P_{0,i}$ terms which come into play for the same position i.
For any position i, we have

$$Z_{0,k-1} = Z_{i+1,k-1}\, Z_{0,k} \qquad (6.8)$$

$$= Z_{i+1,k-1}\, R_{0,i}$$

$$= \left(R_{i+1,k-1}\, \overline{r_{i+1}} + P_{i+1,k-1}\, r_{i+1}\right) R_{0,i}$$

$$= \left(R_{i+1,k-1}\, \overline{G_{0,i}} + P_{i+1,k-1}\, G_{0,i}\right) R_{0,i} \qquad (6.9)$$

an expression which as such would require the generation of the R terms both forward and backward. This is a possible scheme (in combination with (5.28)), for $R_{0,i}$ can also be obtained recursively as we will see below (6.18). However, another solution which we now consider is to determine the recursive parameters which will actually be needed after t is multiplied out by $P_{0,i}$ and $G''_{0,i}$. Thus

$$t\, P_{0,i} = \left(G_k + s_k\, \overline{r'_k}\right) P_{0,i} \qquad (6.10)$$

$$\overline{r'_k}\, P_{0,i} = \left(\overline{R_{i+1,k-1}}\, \overline{G_{0,i}} + \overline{P_{i+1,k-1}}\, G_{0,i} + \overline{R_{0,i}}\right) P_{0,i} \qquad (6.11)$$

$$= \overline{R_{i+1,k-1}}\, P_{0,i} + \overline{R_{0,i}}\, P_{0,i} \qquad (6.12)$$

Now, for $G''_{0,i}$

$$\overline{r'_k}\, G''_{0,i} = \overline{R_{i+1,k-1}}\, \overline{G_{0,i}}\, G''_{0,i} + \overline{P_{i+1,k-1}}\, G_{0,i}\, G''_{0,i} + \overline{R_{0,i}}\, G''_{0,i} \qquad (6.13)$$

The five compound "forward" parameters $P_{0,i}$, $\overline{R_{0,i}}\ P_{0,i}$, $\overline{G_{0,i}}\ G''_{0,i}$, $G_{0,i}\ G''_{0,i}$ and $\overline{R_{0,i}}\ G''_{0,i}$ could be computed from $P_{0,i}$, $G_{0,i}$, $G''_{0,i}$ and $R_{0,i}$ the first three of which are computable recursively as we saw before, while the fourth one will be found of that type also. Thus, we can set up two backward recursion chains and four forward chains and in each point i combine their values according to (6.6.6.7) , and (5.15,5.19,5.21,5.26). The maximum propagation length for any i will be a constant number k of positions.

A last type of solution may be obtained by showing that the five above parameters are themselves computable thru recursions which are of the same type as for the four "primitive" parameters of which they are functions.

6.3 Recursion relationships for compound parameters

$$G''_{0,i}\, G_{0,i} = \left(P_i\, \overline{G'_{0,i-1}}\, G''_{0,i-1} + P_i\, G'_{0,i-1}\right)\left(G_i^1 + P_i\, G_{0,i-1}\right) \qquad (6.14)$$

$$= P_i\, \overline{G'_{0,i-1}} \left[G''_{0,i-1}\, G_{0,i-1}\right] + G_i^1\, G'_{0,i-1} \qquad (6.15)$$

$$G''_{0,i}\, \overline{G_{0,i}} = \left(P_i\, \overline{G'_{0,i-1}}\, G''_{0,i-1} + \overline{P_i}\, G'_{0,i-1}\right)\left(G_i^0 + P_i\, G_{0,i-1}\right) \qquad (6.16)$$

$$= P_i\, \overline{G'_{0,i-1}} \left[G''_{0,i-1}\, \overline{G_{0,i-1}}\right] + G_i^0\, G'_{0,i-1} \qquad (6.17)$$

$$R_{0,i} = \left(R_{i,i}\, \overline{r_i} + P_{i,i}\, r_i\right) R_{0,i-1} \qquad (6.18)$$

$$R_{o,i} = \left(\overline{P_i}\,\overline{r_i} + P_i r_i\right) R_{o,i-1}$$

$$= \left(\overline{P_i}\,\overline{G_{o,i-1}} + P_i G_{o,i-1}\right) R_{o,i-1}$$

$$\overline{R_{o,i}} = \left(\overline{P_i} G_{o,i-1} + P_i \overline{G_{o,i-1}}\right) + \overline{R_{o,i-1}} \qquad (6.19)$$

$$= \overline{P_i}\left(G_{o,i-1} + \overline{R_{o,i-1}}\right) + P_i\left(\overline{G_{o,i-1}} + \overline{R_{o,i-1}}\right)$$

$$G''_{o,i}\,\overline{R_{o,i}} = \overline{P_i}\,G'_{o,i-1}\left(G_{o,i-1} + \overline{R_{o,i-1}}\right) \qquad (6.20)$$

$$+ P_i\,\overline{G'_{o,i-1}}\,G''_{o,i-1}\left(\overline{G_{o,i-1}} + \overline{R_{o,i-1}}\right)$$

$$= P_i\,G'_{o,i-1}\left[G''_{o,i-1}\,\overline{R_{o,i-1}}\right]$$

$$+ \overline{P_i}\,G'_{o,i-1}\,\overline{R_{o,i-1}} + P_i\,\overline{G'_{o,i-1}}\,G^{*}_{o,i-1}\,\overline{G_{o,i-1}}$$

For the last identity we have used the fact that

$$G'_{0,i}\;G_{0,i} = 0 \qquad (6.21)$$

Also, it introduces two new compound parameters of the types $G'_{0,i}\;\overline{R_{0,i}}$ and $\overline{G'_{0,i}}\;G''_{0,i}\;\overline{G_{0,i}}$ which themselves can be obtained thru recursions of the previous types. The first one is constantly zero and can be eliminated. The second is identical to the $P_{0,i}$ parameter already generated :

$$G'_{o,i}\,\overline{R_{o,i}} = \left[\overline{P_i}\left(G_{o,i-1} + \overline{R_{o,i-1}}\right) + P_i\left(\overline{G_{o,i-1}} + \overline{R_{o,i-1}}\right)\right] G^{o}_{i}\,G'_{o,i-1} \qquad (6.22)$$

$$= G^{o}_{i}\left[G'_{o,i-1}\,\overline{R_{o,i-1}}\right]$$

$$= G^{o}_{i} \cdots G^{o}_{1} G^{o}_{o} P_o = 0$$

$$\overline{G'_{o,i}}\,\overline{G''_{o,i}}\,\overline{G_{o,i}} = \left(\overline{G^{o}_{i}} + \overline{G'_{o,i-1}}\right)\left[P_i G'_{o,i-1}\left(G''_{o,i-1}\,\overline{G'_{o,i-1}}\right) + G^{o}_{i} G'_{o,i-1}\right] \qquad (6.23)$$

$$= P_i\left[\overline{G'_{o,i-1}}\,G''_{o,i-1}\,\overline{G_{o,i-1}}\right]$$

$$= P_i \cdots P_1\,\overline{G^{o}_{o}}\,1\,\overline{G^{1}_{o}}$$

$$= P_i \cdots P_1 P_o = P_{o,i}$$

Hence

$$G''_{o,i}\,\overline{R_{o,i}} = P_i\,\overline{G'_{o,i-1}}\left[G''_{o,i-1}\,\overline{R_{o,i-1}}\right] + P_i P_{o,i-1} \qquad (6.24)$$

Lastly

$$P_{o,i}\,\overline{R_{o,i}} = P_i P_{o,i-1}\left(\overline{G''_{o,i-1}} + \overline{R_{o,i-1}}\right) \qquad (6.25)$$

$$= P_i\left(P_{o,i-1} + P_{o,i-1}\,\overline{R_{o,i-1}}\right)$$

$$= P_i P_{o,i-1}$$

$$= P_{o,i}$$

wich entails

$$\overline{r'_k}\,P_{o,i} = P_{o,i} \qquad (6.26)$$

$$t\,P_{o,i} = \left(G_k + s_k\right) P_{o,i} \qquad (6.27)$$

Thus t_{i+1}, for any i, can be generated with the six forward parameters $G_{0,i}$, $P_{0,i}$, $G'_{0,i}$, $G''_{0,i}\;G_{0,i}$, $G''_{0,i}\;\overline{G_{0,i}}$, $G''_{0,i}\;\overline{R_{0,i}}$ which may be used instead of the five parameters $G_{0,i}$, $P_{0,i}$, $G'_{0,i}$, $G''_{0,i}$, $R_{0,i}$.

Also in both solutions the backward parameters $R_{i+1,k-1}$ and $P_{i+1,k-1}$ are needed. With the six forward parameters solution, the cost of having an additional recursion chain should be weighed against the fact that $\overline{r'_k}\;G''_{0,i}$ may be then generated, for each rank i, with 3 gates having 2, 2 and 3 inputs instead of 4 gates having 2,3,3,3 inputs. A design with the six parameter solution is given in Fig. 6. cells (a,b,c,c',c") correspond to equations (6.6, 6.7,6.14 6.16,6.20,6.24).

Initial conditions

$$G_{o,o} = G^{1}_{o} \;;\; G_{o,-1} = o$$

$$P_{o,o} = P_o \;;\; P_{o,-1} = 1$$

$$G'_{o,o} = G^{o}_{o} \;;\; G'_{o,-1} = 1$$

$$R_{o,o} = \overline{P_o} \;;\; R_{o,-1} = 1$$

$$G''_{o,o} = 1 \;;\; G''_{o,-1} = o$$

$$G''_{o,o}\,G_{o,o} = G^{1}_{o} \;;\; G''_{o,-1}\,G_{o,-1} = o$$

$$G''_{o,o}\,\overline{G_{o,o}} = G^{1}_{o} \;;\; G''_{o,-1}\,\overline{G_{o,-1}} = o$$

$$G''_{o,o}\,\overline{R_{o,o}} = P_o \;;\; G''_{o,-1}\,\overline{R_{o,-1}} = o$$

$$R_{k,k} = \overline{P_k} \;;\; R_{k,k-1} = 1$$

Logic for rank k.

In rank k we must add the signed carry for signed-carry addition x+y-v (t) and also subtract v(t) (corresponding to the 2^k component of modulus 2^k+1). The sign t''_k of the incoming carry into position k will not appear in the expression of the sum digit and we will have

$$s''_k = \left(P_k \oplus t_k\right) \oplus t \qquad (6.28)$$

$$= s_k \oplus t$$

where s_k is the sum resulting from the use of t_k in the same way as for positions 0 to k-1 and t is $G_k + s_k\,\overline{R_{0,k-1}}$, in which all three terms are also

generated from the network. Thus the above network is irregular for position k.

7. Alternate designs

We may wish to fuse the s_k term, in the expression of t, with the logic expressions of $\overline{r'_k}$, so as to leave only the variables at rank k (G_k, s_k or T_k) as the irregular terms. As we have seen (3.16)

$$t = r_{k+1} + s_k \overline{r'_k}$$

$$r_{k+1} = G_{0,k} \tag{7.1}$$

$$= G_k + T_k G_{0,k-1} \tag{7.2}$$

$$s_k = P_k \oplus r_k \tag{7.3}$$

$$= P_k \oplus G_{0,k-1} \tag{7.4}$$

$$= \overline{G_k} T_k \oplus G_{0,k-1} \tag{7.5}$$

Then

$$t = (G_k + T_k G_{0,k-1}) + (\overline{G_k} T_k \oplus G_{0,k-1})\overline{r'_k} \tag{7.6}$$

$$= \overline{G_k}\,\overline{T_k} G_{0,k-1} \overline{r'_k} + \overline{G_k} T_k (G_{0,k-1} + \overline{G_{0,k-1} r'_k}) + G_k T_k$$

$$= G_k + \overline{T_k} G_{0,k-1} \overline{r'_k} + T_k (G_{0,k-1} + \overline{r'_k}) \tag{7.7}$$

Hence (cf (6.26))

$$tP_{0,i} = G_k P_{0,i} + \overline{T_k} G_{0,k-1} P_{0,i} + T_k P_{0,i} \tag{7.8}$$

$$= (G_k + \overline{T_k} G_{0,k-1} + T_k) P_{0,i}$$

$$= (G_k + T_k + G_{0,k-1}) P_{0,i}$$

$$= (T_k + G_{0,k-1}) P_{0,i}$$

$$= (T_k + G_{i+1,k-1}) P_{0,i} \tag{7.9}$$

$$tG''_{0,i} = G_k G''_{0,i} + \overline{T_k} G_{0,k-1} \overline{r'_k} G''_{0,i} + T_k (G_{0,k-1} G''_{0,i} + \overline{r'_k} G''_{0,i}) \tag{7.10}$$

with

$$G_{0,k-1} \overline{r'_k} G''_{0,i} = (G_{i+1,k-1} + P_{i+1,k-1} G_{0,i}) \cdot (\overline{R_{i+1,k-1}}\,\overline{G_{0,i}} + \overline{P_{i+1,k-1}} G_{0,i} + \overline{R_{0,i}}) G''_{0,i} \tag{7.11}$$

$$= G_{i+1,k-1} \overline{R_{i+1,k-1}} + G_{i+1,k-1} G_{0,i} + G_{i+1,k-1} \overline{R_{0,i}} + P_{i+1,k-1} G_{0,i} \overline{R_{0,i}} G''_{0,i}$$

$$G_{0,k-1} + \overline{r'_k} = (G_{i+1,k-1} \overline{G_{0,i}} + P_{k+1,k-1} G_{0,i}) + (\overline{R_{i+1,k-1}}\,\overline{G_{0,i}} + \overline{P_{i+1,k-1}} G_{0,i} + \overline{R_{0,i}}) \tag{7.12}$$

$$= G_{i+1,k-1} + \overline{R_{i+1,k-1}} + G_{0,i} + \overline{R_{0,i}}$$

$$tG''_{0,i} = \Big[G_k + \overline{T_k}\,(G_{i+1,k-1} \overline{R_{i+1,k-1}} + G_{i+1,k-1} \overline{R_{0,i}} + G_{i+1,k-1} G_{0,i} + G_{i+1,k-1} \overline{R_{0,i}} + P_{i+1,k-1} G_{0,i} \overline{R_{0,i}}) + T_k (G_{i+1,k-1} + \overline{R_{i+1,k-1}} + G_{0,i} + \overline{R_{0,i}}) G''_{0,i} \tag{7.13}$$

$$= \Big[G_k + \Big(G_{i+1,k-1} \overline{R_{i+1,k-1}} + G_{i+1,k-1} G_{0,i} + G_{i+1,k-1} \overline{R_{0,i}} + P_{i+1,k-1} G_{0,i} \overline{R_{0,i}}\Big) + T_k \Big(G_{i+1,k-1} + \overline{R_{0,i}}\Big) G''_{0,i} \tag{7.14}$$

$$= \Big[G_k + \Big(G_{i+1,k-1} \overline{R_{i+1,k-1}} + G_{i+1,k-1} G_{0,i} + G_{i+1,k-1} \overline{R_{0,i}} + P_{i+1,k-1} G_{0,i} \overline{R_{0,i}}\Big) + T_k \Big(G_{i+1,k-1} + \overline{R_{0,i}}\Big) G''_{0,i} \tag{7.15}$$

Thus the carry t_{i+1} (cf (5.18)) can be computed from $G_{0,i}$, $tP_{0,i}$ and $tG''_{0,i}$ obtained above. With expressions (7.9, 7.14) it can be implemented with forward parameters $G_{0,i}$, $P_{0,i}$, $G'_{0,i}$ $G''_{0,i}$ $R_{0,i}$ (or $G_{0,i}$, $P_{0,i}$, $G'_{0,i}$, $G''_{0,i}\,\overline{R_{0,i}}$, $G''_{0,i}\,G_{0,i}$ and $G''_{0,i}$ which again involves one more chain, but simpler gates). The generation of $G''_{0,i}$ may be then avoided if T_k may be assumed zero. This would be the case if we treated separately the case where one or both operands equal 2^k and the other cases. Such a scheme would involve a T_k gate, to switch the computation to either one of two networks, one to form a number equal to $v(G_k)$ $(2^k - 1)$ if T_k is one and to use the above network for $T_k = 0$. Then this operating constraint, the above network would simplify to

$$t_{i+1} = G_{0,i} \oplus P_{0,i}\, t \oplus G''_{0,i}\, t \tag{7.16}$$

$$P_{0,i}\, t = G_{i+1,k-1}\, P_{0,i} \tag{7.17}$$

$$G''_{0,i}\, t = \Big[G_{i+1,k-1} \overline{R_{i+1,k-1}} + G_{i+1,k-1} G_{0,i} + G_{i+1,k-1} \overline{R_{0,i}} + P_{i+1,k-1} G_{0,i} \overline{R_{0,i}}\Big] G''_{0,i} \tag{7.18}$$

This latter network assumes x and y to be residues, but this is not necessary for the more general one described by (7.9, 7.14).

Both types of networks in §.7 require a backward recursion to generate the $G_{i,k}$ terms. This can be done with the relationship [13]

$$G_{i,k} = G_{i+1,k} + P_{i+1,k}\, G_i \tag{7.19}$$

Appendix

In ordinary addition of binary numbers x and y the generate, propagate and transmit functions are defined as follows. For a block [0,i] of digits and an adder assumed to operate on the digits x_j, y_j for this block $(o \leq j < i)$, then the generate function is the carry out of position i, for a carry into position 0 equal to zero and the transmit function is the carry out of the same position when the carry into position 0 is a one. The propagate function for block [0,i] is the product of propagate functions for each position j. Let us use + for the logical sum (inclusive OR), $\oplus$ for the exclusive OR, concatenation for the product (AND) of variables and overbars for complements. Let us denote for any block [i,j] $(i \leq j)$ of digits $G_{i,j}$, $T_{i,j}$ and $P_{i,j}$ the generate, transmit and propagate functions. Then we have the following relationships and conventions [15,16,18,21]:

(i) properties :

$$P_{o,i} = \prod_{j=0}^{i} P_j$$

$$P_j = x_j \oplus y_j$$

$$G_{o,i} = G_i^1 + P_i G_{o,i-1} \quad ; \quad G_i^1 = x_i y_i$$

$$G_{o,i} = G_i^0 + P_i G_{o,i-1} \quad ; \quad G_i^0 = \bar{x}_i \bar{y}_i$$

$$T_{o,i} = G_{o,i} + P_{o,i}$$

$$G_{o,i} = G_{j+1,i} + P_{j+1,i} G_{o,j}$$

(ii) conventions :

$$G_{i,i} = G_i^1 \quad ; \quad G_{i,i-1} = 0$$

$$P_{i,i} = P_i \quad ; \quad P_{i,i-1} = 1$$

$$T_{i,i} = T_i \quad ; \quad T_{i,i-1} = 1$$

G_i^1 and G_i^0 are so-called 0-generate and 1-generate terms respectively.

References

1. H.L. Gardner, "Error Codes for Arithmetic Operations," IEEE Trans. Elect. Computers, vol. EC-15, pp. 763-770, Oct. 1966.

2. W.W. Peterson, Error Correcting Codes, MIT Press Cambridge, 1965.

3. A. Avižienis, "Arithmetic Algorithms for Error coded Operands," 1972 Intern. Fault-tolerant Computing Symposium, Papers, pp. 25-34, June 1972.

4. A. Avižienis, "Arithmetic Error Codes : Cost and Effectiveness Studies for Application in Digital System Design," IEEE Trans. on Computers, Vol. C-20, N° 11, pp. 1321-1331, Nov. 1971.

5. T.R.N. Rao, "Error-Checking Logic for Arithmetic-type Operations of a Processor, IEEE Trans. on Computers, Vol. C-17, pp. 845-849, Sept. 1968.

6. A.E. Pertmann, Circuits for Checking Arithmetic Errors by Means of Residue Coding, AD 687095, Feb. 1969.

7. A. Svoboda, "Le Système Numérique des Classes Résiduelles Dans les Machines Mathématiques," Automatisme, Vol. 5, Nos 1 and 2, pp. 16-24, pp. 65-69, 1960.

8. N.S. Szabo, R.I. Tanaka, Residue Arithmetic and its Application to Computer Technology, McGraw Hill, New York, 1967.

9. T.R.N. Rao, O.N. Garcia, "Cyclic and Multiresidue Codes for Arithmetic Operations," IEEE Trans. Inform. Theory, Vol. IT-17, pp. 85-91, Jan. 1971.

10. A. Avižienis, "Digital Computer Arithmetic : a Unified Algorithmic Specification, " Symposium on Computers and Automata, Pol. Inst. of Brooklyn (13-15 April 1971) Papers. pp. 509 - 525.

11. R.K. Richards, Digital Design, Wiley, New York 1971.

12. N.R. Scott, Electronic Computer Technology, McGraw Hill, New York, 1970.

13. J.P. Chinal, Single-Zero Digit Complement Adders, UCLA Comp. Science Department, Internal Memorandum, Nov. 1974. (in press)

14. J.P. Chinal, Détection et Correction Synchrones d'Erreurs Arithmétiques dans les Processeurs, ENSAE, Toulouse, 100p, March 1973.

15. J.P. Chinal, Microsystèmes Numériques, ENSAE, Toulouse, 270 p. Jan. 1972.

16. J.P. Chinal, Théorie Microanalytique des Processeurs Arithmétiques, ENSAE, Toulouse, 90 p. June 1973.

17. K.J. Dean, "Logical Circuits for Use in Iterative Arrays," Electronic Letters, 4, pp. 81-82, March 1968.

18. M. Carvallo, Principes et Applications de l'Analyse Booléenne, Gauthiers-Villars, Paris 1965.

19. J.P. Chinal, Boolean Features of Base-2 Signed Carry Adders, UCLA, Computer Science Dept.Note, (in press)

20. G.C. Langdon, C.K. Tang, "Concurrent Error Detection for Group Look-Ahead Binary Adders," IBM Journal of Res. and Development. , pp. 563-573, Sept. 1970.

21. I. Flores, The Logic of Computer Arithmetic, Prentice Hall. 1963.

ACKNOWLEDGMENTS

This work was done while the author was with UCLA Computer Science Department as a US-France exchange scientist. The support of National Science Foundation, Washington. and of ENSAE, Toulouse, is gratefully acknowledged.

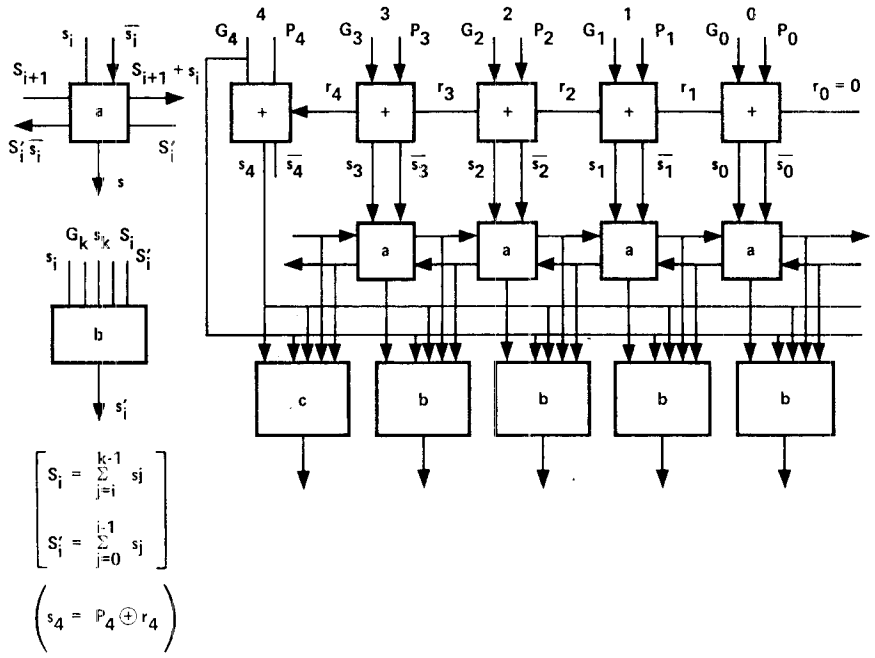

Figure 1. Classical Adder with Output Correction Network for Moduls 2^k+1 Addition. (k=4)

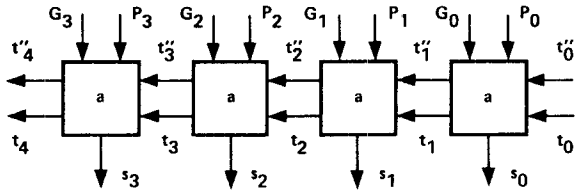

Figure 2. Signed-Carry Adder.

Logic:

$s_i = P_i \oplus t_i$; $t_{i+1} = G_i \oplus P_i t_i \oplus t''_i t_i$; $t''_{i+1} = t''_i t_i$

Arjthmetic Value of Carry:

$t'_i = (-1)^{v(t''_i)} v(t_i)$

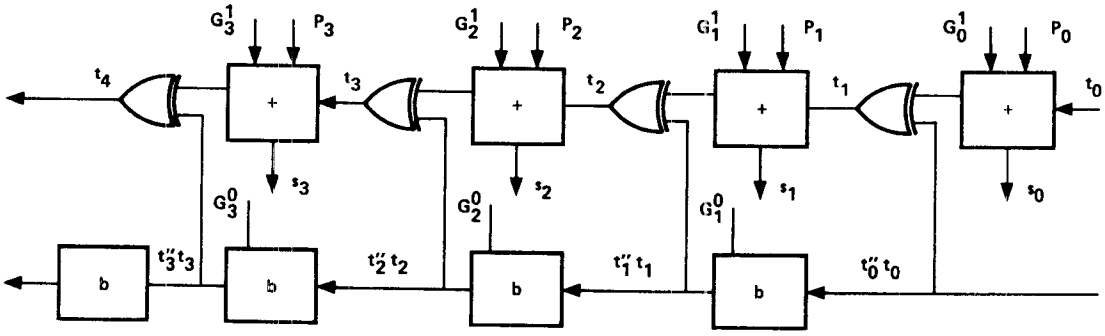

Figure 3. Signed-Carry Adder with Ordinary Adder Stages.

(+) cell: ordinary full adder
(b) cell: and gates

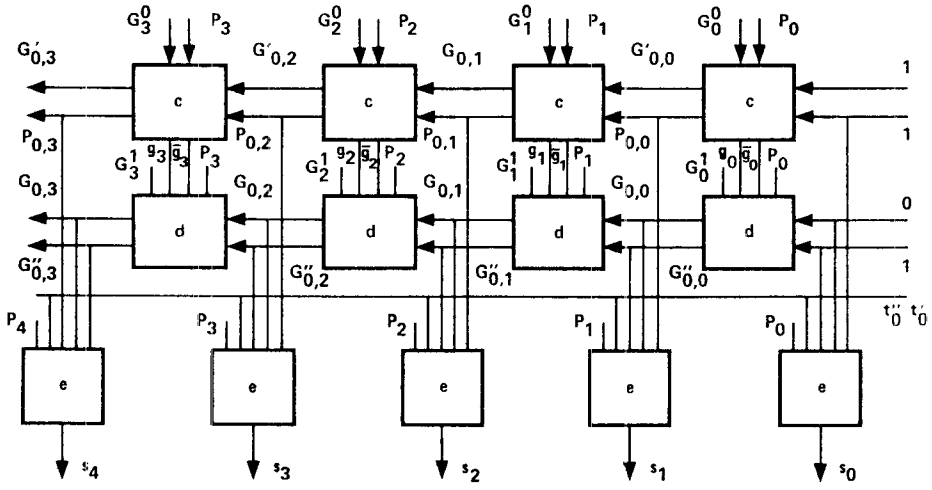

Figure 4. Signed Carry Adder with $G_{0,i}$, $G'_{0,i}$,$G''_{0,i}$ $P_{0,i}$ Interstage Variables.(g_i=$G'_{0,i-1}$)

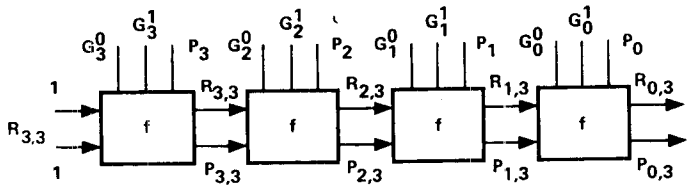

Figure 5. Backward r_k Generator.

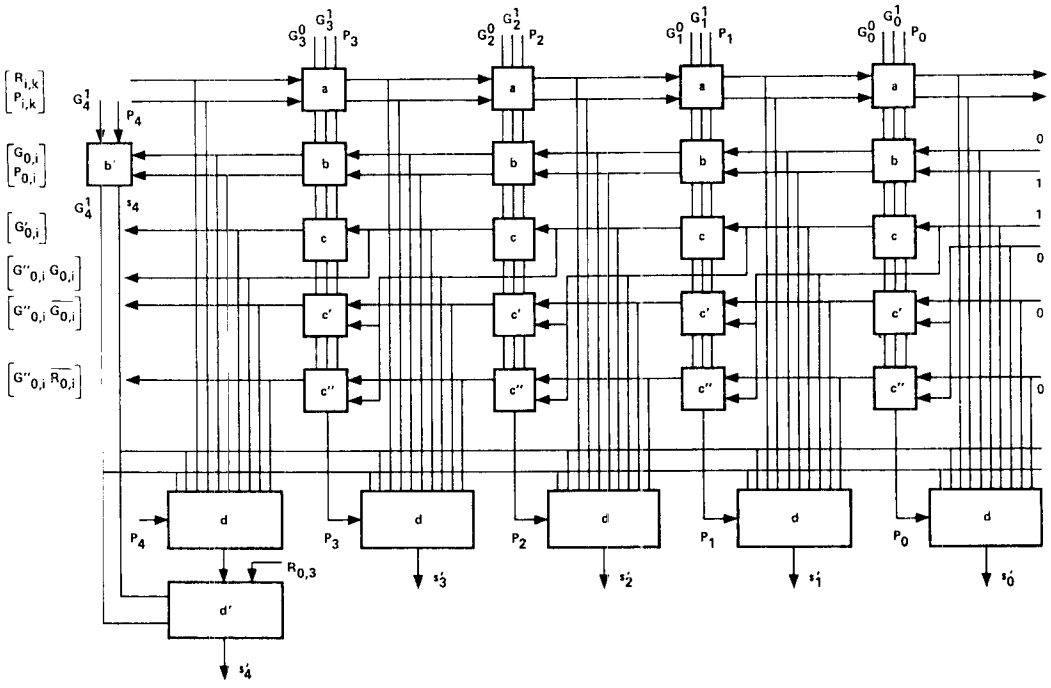

Figure 6. Another Modulo 2^k+1 Adder with Six Forward Chains and Two Backward Chains.