Performance Comparison of Hybrid Signed Digit Arithmetic in Efficient Computing

VISHAL AWASTHI[1]

[1]Department of Electronics & Communication Engineering
UIET., C.S.J.M.University, Kanpur-24, U.P., India
awasthiv@rediffmail.com

KRISHNA RAJ[2]

[2]Department of Electronics Engineering,
H.B.T.I., Kanpur-24, U.P., India
kraj_biet@yahoo.com

Abstract

In redundant representations, addition can be carried out in a constant time independent of the word length of the operands. Adder forms a fundamental building block in almost majority of VLSI designs. A hybrid adder can add an unsigned number to a signed-digit number and hence their efficient performance greatly determines the quality of the final output of the concerned circuit. In this paper we designed and compared the speed of adders by reducing the carry propagation time with the help of combined effect of improved architectures of adders and signed digit representation of number systems. The key idea is to draw out a compromise between execution time of fast adding process and area available which is often very limited. In this paper we also tried to verify the various algorithms of signed digit and hybrid signed digit adders.

Keywords: ***Fast adders; Redundant arithmetic; Carry free addition; Hybrid and Signed digit numbers.***

1. Introduction

Arithmetic operators designed using redundant number system achieves considerable speed improvement compared with operators designed using conventional number system. The execution speed of an arithmetic operation is directly related to chosen architecture and the architecture and implementation styles are usually decided by the number system employed to represent numbers. Addition is widely recognized as a basis of other arithmetic operation. The number of add operations required in a constant coefficient multiplication equals one less than the number of nonzero bits in the constant coefficient. Early computers using two's complement arithmetic made use of ripple carry adders which forms the basis of fast adders, which had a simple hardware design, and their implementation was easy. The adder had a long carry propagation paths extending from the least significant bit to the most significant bit position. Its computation time is directly proportional to the word length of the operands. Robertson [1] and Avizienis [2] suggested a set of arithmetic rules for redundant signed digit numbers. An idea of logical design of the RBSD adders has been proposed by Chow and Robertson [3]. Takaji and Yazima [4] proposed a high-speed algorithm suitable for VLSI implementation using RBSD numbers. High-speed arithmetic for binary number system has been presented by Macsorley [5] and a fast conditional sum addition technique came into feature in 1960 by Sklanski [6-7]. A carry select addition technique was presented [8] by O. J. Bedrij. Oklobdzija, Zeydel [9] proposed a Logical Effort (LE) based analysis for quick exploration of the energy-delay space for comparing the performance of high-performance adders. A RIC ((Re-computing the Inverse Carry-in) fast adder [10] has been proposed which is a very good option for designing a high performance and a moderate cost hardware.

High speed adders can be designed using the binary look ahead (which is popularly known as carry look ahead) and carry select adders. This technique reduces the carry propagation time by using an associative operator to compute the carry in a binary fashion and using the intermediate carries to generate the sum bits. On the other hand, the carry select addition scheme reduces the computation time by pre-computing some positions of the results for all possible carry bit values (i.e. 1 & 0) and using the carry from the previous bit position (after it becomes available) to choose the proper result [11]. In recent years carry free arithmetic operations (such as multiplication, division, square root etc.) employing redundant number systems [12] have received considerable attention. These redundant arithmetic operations employ a signed digit representation where each digit of a number can be positive or negative. The use of a redundant number systems leads to carry free addition, where the carry propagates only through two or three stages, independent of the word length.

In this paper we discuss the efforts to increase the speed of computing under two heads :- (a) choice of logic design style (arrangement of gates) (b) Number system representation which introduces hybrid number representation which is capable of bounding the maximum length of carry propagation chains during addition to any desired value between 1 and the entire word length.

2. Review of Fast Adders

A fast and energy-efficient adder is essential for a high-performance processor. Ripple Carry adder is the first and the most fundamental adder that is capable of performing binary number additions.

2.1 Ripple Carry Adder (RCA)

A ripple carry adder is a simple adder consisting of several full adders connected in series so that the carry propagates through every full adder before the addition is complete. Hence they are slowest as the final carry depends upon the intermediate carry propagation. The sum and carry are calculated in this adder according to the following equations-

$$S = xy'z' + x'yz' + xyz + x'y'z$$
$$C = xy'z + x'yz + xy$$

Example:

```
   1 0 1 1    Addend
 + 1 1 1 0    Augend
 ---------
 1 1 0 0 1    Final sum
 ↓
Carry
```

1. x=1, y=0, z=0, $s_0 = 1, c_0 = 0$
2. x=1, y=1, z=0, $s_1 = 0, c_1 = 1$
3. x=0, y=1, z=1, $s_2 = 0, c_2 = 1$
4. x=1, y=1, z=1, $s_3 = 1, c_3 = 1$

Long circuit delay due to many gates in carry path from the LSB to MSB is the major disadvantage of Ripple Carry Adder. [6]

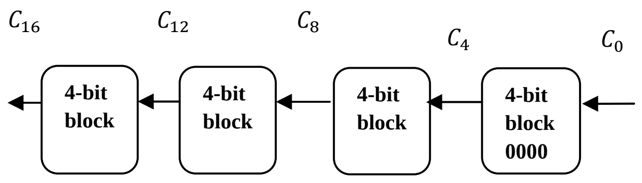

Figure 1: Ripple Carry Adder (RCA)

2.2 Carry Look Ahead (CLA) Adder

Weinberger and Smith proposed this scheme in 1958[11]. It uses look ahead carry technique rather than carry rippling technique to speed up the carry propagation. Carry look ahead depends upon two things-

(1) Calculating for each digit position whether that position is going to propagate a carry if one comes in from the right.

(2) Combining these calculated values so as to be able to deduce quickly whether, for each group of digits, that group is going to propagate a carry that comes in from the right.

Supposing that groups of four digits are chosen, then all 1- bit adders calculate their results. Simultaneously, the look ahead units perform their calculations. Suppose that a carry arises in particular group within at most 3 gate delays that carry will emerge at the left hand end of the group and start propagating through the group to its left. If that carry is going to propagate all the way through next group, the look ahead unit will already have deduced this. The net effect is that the carries start by propagating slowly through each 4-bit group but then moves 4 times as fast, leaping from one look ahead carry unit to the next. Finally, within each group that receives a carry the carry propagates slowly within the digits in that group [7].

2.3 Carry Select Adder (CSA)

Sklansky proposed the conditional sum addition in 1960. Carry select adder scheme divides adders into blocks of ripple carry adder each with two replicas, one replica evaluates with carry- in of 0, the other one with carry-in of 1.In this scheme the carry out is from less significant block which then conditionally selects the output of succeeding blocks [7].

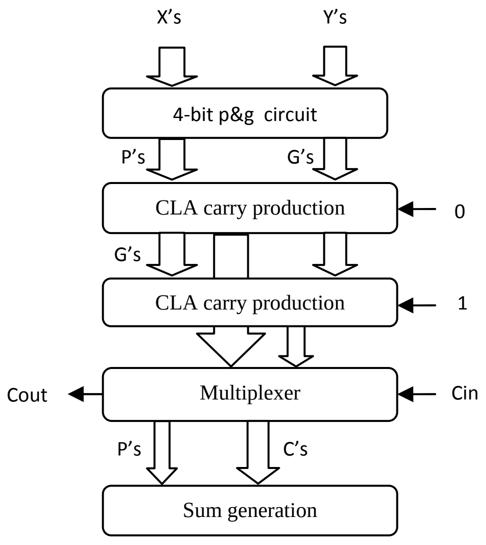

Figure 2: 4-bit Carry Select Adder

2.4 Ling Adder

Adder proposed by Ling is an improved version of conventional carry-look ahead adder. This approach is faster and less expensive. It replaces the conventional propagate operator from a XOR with an OR gate which results in a much cheaper operation. Ling proposed that instead of having a single signal at each bit position for encoding to be spread into two signals, relaxing the carry computation [13].

2.5 Carry Skip Adder

Carry skip scheme is proposed by Kilburnet [14] to accelerate the carry propagation. The skip adder provides a compromise between a ripple carry adder and a CLA adder. The carry skip adder divides the words to be added into blocks and ripple carry adder is used to produce the sum bit and carry.

The carry of the first block is generated with two gate delays per block. The second block waits until the first ripple carry block generates a carry. Although the modified carry skip adder uses a few extra gates because of the CLA logic, it operates with small gate delay. [10]

3. Number System Representations

In computing signed digit number representation is required to encode negative numbers in binary number systems. In mathematics, negative numbers in any base are represented by prefixing them with a – sign. However, in computer hardware, numbers are represented in binary only without extra symbols, requiring a method of encoding the minus sign. The signed digit number representation makes it possible to perform addition without carry propagation chains that are used to speed up arithmetic operations.

3.1 Redundant Signed Digit- Carry-Free Addition Algorithm

A symmetrical signed digit (SD) number can assume the following values i.e. [–α,....,–1,0,1,....,α]. Where the maximum value of α must be within the following range: [(r – 1)/2] ≤ α ≤ r – 1. In order to yield minimum redundancy, one can choose the maximum magnitude α = [r/2]. If r = 2, then number representation is known as Redundant Binary Signed Digit (RBSD) number system.

A digit set need not be the standard. Radix-r, a digit set [–k, μ], having the condition k + μ + 1≥ r may be converted to a radix-r digit set$[\alpha, \beta]$, for $\alpha + \beta + 1 \geq r$. The redundancy δ of the number system is defined as $\delta = \alpha + \beta + 1 - r$.

In the binary signed digit number system, each digit can assume any one of three values {-1, 0, 1}[11]. As a result redundancy is introduced in a system i.e. a number can be represented in more than one way. Due to the presence of redundancy one can perform carry-propagation free addition and hence parallel addition of two redundant numbers can be performed in a constant time independent of the word length of operands.

Example:

$$\begin{array}{rccc} & 1\ 0\ 1 & - & (5) \\ + & 1\ 1\ 0 & - & (6) \\ \hline & 1\ 0\ 1\ 1 & - & (11) \\ & \searrow \text{(Carry generated)} & & \end{array}$$

For 4-bit redundant representation of above example, then 6 can be written as = 1 0 $\bar{1}$ 0 = 1x2^3+$\bar{1}$x2 = 8-2 = 6

$$\begin{array}{rccc} & 0\ 1\ 0\ 1 & & \\ + & 1\ 0\ \bar{1}\ 0 & & \\ \hline & 1\ 1\ \bar{1}\ 1 & - & (11) \end{array}$$

Thus no carry propagation was required. The above representation is possible using Binary Signed Arithmetic, for radix 2 & for more general case Signed number representation for radix greater than or equal to 2.

3.2 Signed Binary Digit (SBD) Arithmetic

Signed–digit (SD) number representation makes it possible to perform addition with carry propagation chains that are limited to a single digit position, and has been used to speed up arithmetic operations. This redundancy can be exploited to limit the length of carry propagation chains to only one digit position, making it possible to add two numbers in fixed time, irrespective of the word length.

This section of program given below shows the algorithm for Binary Signed digit number (BSD) addition. The tabular representation of this algorithm is taken from [11]. The above algorithm can be clearly understood with the help of following example:

$$\begin{array}{rcl} \bar{1}\ 0\ 1\ \bar{1} & (-7) & \text{Addend} \\ 1\ \bar{1}\ \bar{1}\ 0 & (2) & \text{Augend} \\ \hline 0\ 1\ 0\ \bar{1} & & \text{Intermediate sum} \\ 0\ \bar{1}\ 0\ 0\ 0 & & \text{Intermediate carry} \\ \hline 0\ \bar{1}\ 1\ 0\ \bar{1} & (-5) & \text{Final sum} \end{array}$$

3.3 Algorithm of Signed Digit Number Representation

For a given radix r, each digit mi in an sign digit number system is typically in the range,

$$-\alpha \leq m_i \leq +\alpha\ , \ \ where \quad \left\lceil r-\tfrac{1}{2}\right\rceil \leq \alpha \leq r-1 \qquad (1)$$

In this system a "carry free addition" can be done, which means that carry propagation is limited to a single digit position i.e. the carry propagation length is fixed irrespective of the word length. The addition here is done in two steps. In the first step, an intermediate sum s_i and a carry c_i is generated, based on the operand digits m_i and $n_i$at each digit position i this is done in parallel for all digit passions. In the second step, the summation $p_i = s_i + c_{i-1}$ is carried out to produce the final sum digit p_i the most important fact is that it is always possible to select the intermediate sum s_i and carry c_{i-1} such that the summation in the second step does not generate a carry. If the selected value of b in the equation 1 satisfies the condition:

$$\left\lceil r+\tfrac{1}{2}\right\rceil \leq \alpha \leq r-1 \qquad (2)$$

Then the intermediate sum s_i and c_i depend only on the input operands i.e. on m_i and n_i.The interim sum is

$$s_i = m_i + n_i - rc_i \qquad (3)$$

$$\text{where} \quad c_i = \begin{cases} 1 & \text{if } m_i + n_i \leq +b \\ -1 & \text{if } m_i + n_i \leq -b \\ 0 & \text{if } m_i + n_i < b \end{cases}$$

It is observed that for the most commonly used binary number system (r=2) condition 2 cannot be satisfied. Carry free addition according to the rules in 3 therefore cannot be performed with binary operands but, by observing the input operands in position $i-1$ together with the operands in position i, it is possible to select a carry c_i and an interim sum si is made depend on two digit positions i.e. i,i-1, then condition 2 can be relaxed and a=r-1/2 can also be used to achieve carry free addition.

Let $\alpha i = mi + ni$ & $\alpha_{i-1} = m_{i-1} + n_{i-1}$ denote the sums of the input digits at the two positions respectively. Given in table, the rules for generating the intermediate sum s_i & carryc_i. In the table, the symbol X indicates a don't care i.e. the value of α_{i-1} doesn't matter. The table given below is valid for only Even radix, i.e. $r = 2a$.

Example: For radix 6

$\bar{1}\ 0\ \bar{3}\bar{2}$	(- 48)	Addend
$+\bar{2}\ 3\ 2\ \bar{1}$	(- 19)	Augend
$\bar{3}\ 1\ 2\ 2$		Intermediate sum
$0\ 0\ 0\ \bar{1}\ 0$		Intermediate carry
$0\ \bar{3}\ 1\ 1\ 2$	(- 67)	Final sum

3.4 Hybrid Signed Digit (HSD) Number System

In this number system, instead of insisting that every digit be a signed digit, we let some of the digits to be signed and leave the others unsigned. For example, every alternate or every third or fourth digit can be signed; all the remaining ones are unsigned. We refer to this representation as a Hybrid Signed-Digit (HSD) representation. In this representation we can limit the maximum length of carry propagation chains to any desired value. In particular, maximum length of a carry propagation chain equals (*d* + 1), where *d* is the longest distance between neighbouring signed digits.

Addition in such a representation requires the carry in between all digit positions (signed or unsigned) to assume any value in the set {-1, 0, 1} as in the SD system. The Hybrid Signed Digit (HSD) number system has maximum length of a carry propagation chain limited to the (longest) distance between adjacent signed digits. The multipliers in the digital filters are realized with shifters, adders and subtractors. The use of HSD expression can reduce the number of adders and subtractors with high computational speed. To multiply with HSD number only shift and add operations are required.

Table 1: Rules for selecting carry c_i and intermediate sum s_i based on $\alpha_i = m_i + n_i$for radix $r = 2b$

$\boldsymbol{\alpha_i}$	$\boldsymbol{\alpha_{i-1}}$	$\boldsymbol{c_i}$	$\boldsymbol{s_i}$
$\alpha_i = -2b$	X	-1	0
$-2b < \alpha_i < -b$	X	-1	$\alpha_i + r$
$\alpha_i = -b$	$\alpha_{i-1} \leq -b$	-1	b
	$\alpha_{i-1} > -b$	0	-b
$-b < \alpha_i < b$	X	0	α_i
$\alpha_i = b$	$\alpha_{i-1} < b$	0	b
	$\alpha_{i-1} \geq b$	1	-b
$b < \alpha_i < 2b$	X	1	$\alpha_i - r$
$\alpha_i = 2b$	X	1	0

Example: For radix 2

$\bar{1}\ 1\ 0\ \bar{1}\ 1$	(-9)	Addend
$0\ 1\ 0\ \bar{1}\ \bar{1}$	(5)	Augend
$\bar{1}\ 0\ 0\ 0\ 0$		Intermediate sum
$0\ 1\ 0\ \bar{1}\ 0\ 0$		Intermediate carry
$0\ 0\ 0\ \bar{1}\ 0\ 0$	(-4)	Final sum

4. Result & Discussion

The speed up in addition time in the SD number does not come without cost, as more bits are needed to represent a binary signed digit. Also, the basic adder cell that adds two signed digits and an input carry to produce a signed digit and a carry is more complex than a full adder for unsigned digits. Thus more area is traded off for the constant addition time. In the SD number system more bits, switching devices & routing are required per digit. In return, the carry propagation is limited to a single digit position. The performnace of algorithm can also be described by its efficiency and that is given by-

$$\text{Efficiency } (\eta) = \frac{n}{[\rho.log_2 G]}$$

Where ρ=Total addition time; n= No. Of bits to be added; G= Gate count; Δ= Delay of NAND/NOR gate

Table 2: Efficiency of Different Addition Algorithms

No. of Bits	RCA (Binary)	CCA	CSA	RBSD
2	0.12	0.118	0.115	0.1
4	0.13	0.15	0.15	0.3
8	0.145	0.25	0.25	0.9
16	0.17	0.31	0.31	1.4
32	0.19	0.33	0.81	1.6
64	0.21	0.37	0.98	1.95
128	0.23	0.42	1.25	3.7
256	0.27	0.64	2.01	6.5
512	0.28	0.79	2.54	8.6
1024	0.3	0.81	2.89	11.1

Similarly the comparison of the Execution time of Hybrid signed digit (HSD) adder algorithm with the algorithm of Ripple carry adder (RCA), Redundant Binary signed digit (RBSD) adder, signed digit (SD) adder is described in table 3. It is clear that the Execution speed of Hybrid signed digit (HSD) adder algorithm increases as the word length of operand increases it is so because it provides the carry- propagation free addition.

Table 3: Execution Time of Different Addition Algorithms

No. of Bits	RCA (Binary)	RBSD	SD	HSD
2	1.026454	0.991452	0.990129	0.961259
4	1.022589	0.988947	0.983792	0.954266
8	1.014381	0.976212	0.97508	0.947488
16	1.005479	0.973142	0.972182	0.922707
32	1.002994	0.970945	0.970019	0.893785
64	1.000071	0.96166	0.95827	0.881939
128	0.991526	0.943726	0.938739	0.872127
256	0.972539	0.924521	0.915266	0.861925
512	0.954367	0.901291	0.899257	0.855027
1024	0.904369	0.891793	0.878529	0.837102

In the conventional number systems on the other hand, fewer bits switches & routing are needed per digit, but the carry propagates across the entire word length. Hybrid number system offers an optimization where the maximum carry propagation length can be set to any desired value between one & the full word length. The area required decreases as the length of the carry propagation chain increases.

In Hybrid number representation, it is not necessary that every digit be a signed digit, we can let some of the digits to be signed and leave the others unsigned. For example, every alternate or every third or fourth digit can be signed, rest three remaining ones can be unsigned.

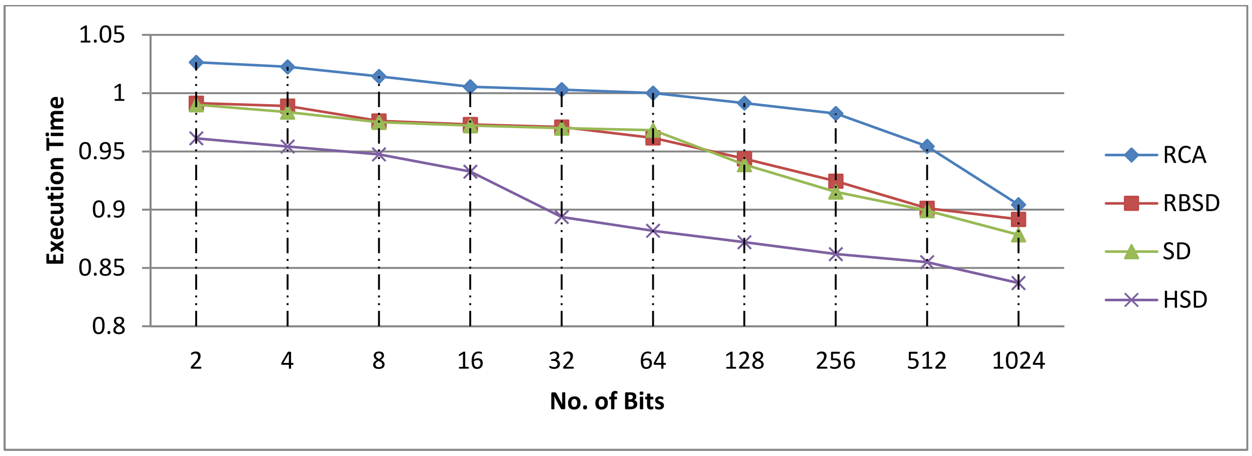

Figure 3(a): Comparison of Execution Time of different Algorithm

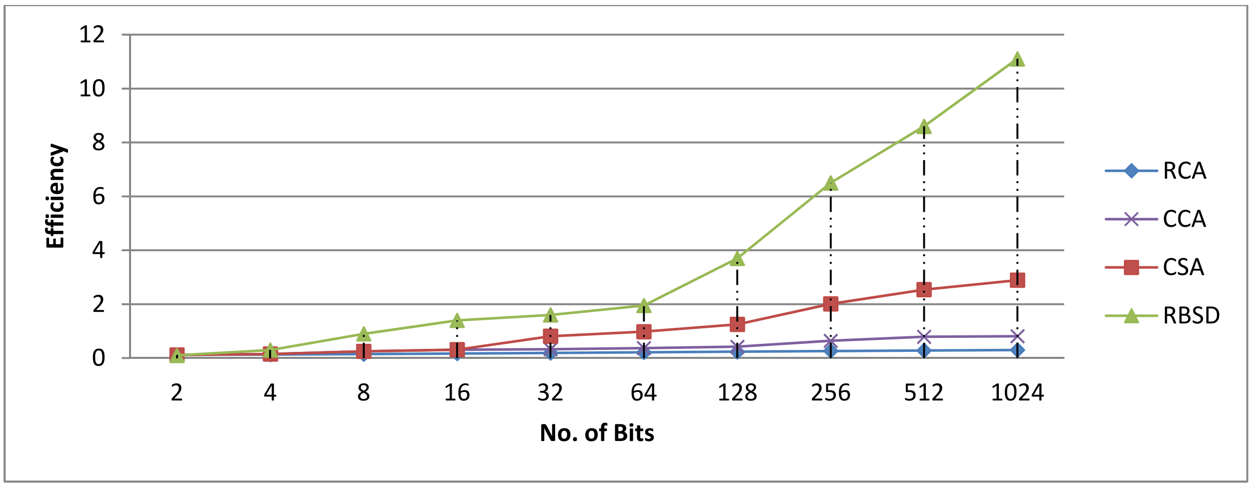

Figure 3(b): Comparison of Efficiency of different Addition Algorithms

5. Conclusion

Hybrid numbers has simply an added advantage over signed numbers that, here we can limit the maximum length of carry propagation chains to any desired value but with the limitation for radix. Through examples and MATLAB programming we conclude that algorithm for signed digit numbers is more flexible, where it is a generalized form & very well includes case of hybrid numbers. The added feature of our this paper is that, we have tried to graphically compare the Execution Times of each program with the help of MATLAB and deduced that representation in Hybrid Number System is more faster i.e. numerical value of execution times after running the respective programs comes out to be minimum for Hybrid Number Representation.

References

[1] J. E. Robertson, “A Deterministic Procedure for the design of Carry-Save Adders and Borrow-Save Subtractors,” University of Illinois, Urbana-Champaign, Dept. of Computer Science, Report No. 235, July 1967.
[2] A. Avizienis, "Signed-Digit Number Representation for Fast Parallel Arithmetics", IRE Transactions on Electronic Computers, pp. 389, 1961.
[3] C. Y. Chow and J.E. Robertson, “Logical Design of a Redundant Binary Adder,” in Proc. 4th Symp. Compter arithmetic, pp. 109 -115, Oct. 1978.
[4] N. Takagi, H. Yasuura and S. Yajima, “High Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree,” IEEE Trans. Comp. vol. 34 pp. 789-796, Sept. 1985.
[5] O. L. MacSorley, “High-Speed Arithmetic in Binary Computers,” Proc. IRE, vol. 49, pp. 67–91, Jan. 1961.
[6] I. J. Sklansky,“An Evaluation of Several Two-Summand Binary Adders,” IRE Trans. Elec. Comp., Vol. 9, No. 2, June 1960, pp. 213-225.
[7] I. J. Sklanski, "Conditional-Sum Addition Logic", IRE Transaction on Electronic Computers, EC-9, pp. 226-231, 1960
[8] O. J. Bedrij, "Carry-Select Adder", IRE Transactions on Electronic Computers, p. 340, June 1962.
[9] Vojin G. Oklobdzija1, Bart R. Zeydel, Hoang Dao1, Sanu Mathew, Ram Krishnamurthy, “Energy-Delay Estimation Technique for High-Performance Microprocessor VLSI Adders”, Proceedings of the 16th IEEE Symposium on Computer Arithmetic, 1063-6889, 2003
[10] Eduardo Mesquita, Helen Franck, Luciano Agostini, Jose' Luis Guintzel, " RIC Fast Adder and its Set-Tolerant Implementation in FPGAs”, 1- 4244 – 1060 - 6/07 C 2007 IEEE, pp. 838-641, 2007
[11] Weinberger, J.L. Smith, “A Logic for High- Speed addition”, National Bureau of Standards, Circulation 591, pp. 3-12, 1958.
[12] K. Hwang, "Computer Arithmetic : Principles, Architecture and Design", John Wiley and Sons, 1979.
[13] Ling, "High Speed Binary Parallel Adder", IEEE Transactions on Electronic Computers, EC-15, pp.799 - 809, October, 1966.
[14] T. Kilburn, D.B.G. Edwards and D. Aspinall, Parallel Addition in Digital Computers: A New Fast "Carry” Circuit, Proced. IEE, Vol.106, Pt.B. pp.464, September 1959.

Appendix:

Ripple Carry Adder (RCA) Algorithm

```
for i=(n-1):-1:1
    s(n)=0;
    c(n)=0;
    p(i)=x(i)+y(i)+c(i+1)
    if p(i)<2
        s(i)=p(i);
        c(i)=0;
    end
    if p(i)==2
        s(i)=0;
        c(i)=1;
    end
    if p(i)==3
        s(i)=1;
        c(i)=1;
    end
end
```

Signed Binary Digit (SBD) Arithmetic Algorithm for any radix

```
for i=(n-1):-1:1
    s(n)=0; c(n)=0;
    if (x(i)+y(i))>=a
        c(i)=1;
        s(i)=x(i)+y(i)-r ;
    end
    if (x(i)+y(i))<=-a
        c(i)=-1;
        s(i)=x(i)+y(i)+r ;
    end
    if abs(x(i)+y(i))<a
        c(i)=0;
        s(i)=x(i)+y(i) ;
    end
end
```

Signed Binary Digit (SBD) Algorithm for Radix 2

```
for i=(n-1):-1:1
    s(n)=0; c(n)=0;
    if m(i) == -2
        s(i) = 0;
        c(i) = -1;
    end
    if m(i) == -1
        if (x(i+1)<0)||(y(i+1)<0)
            s(i) =1;
            c(i) = -1;
        end
        if (x(i+1)>=0)&&(y(i+1)>=0)
            s(i) = -1;
            c(i) = 0;
        end
    end
    if m(i) == 1
        if (x(i+1)<0)||(y(i+1)<0)
            s(i) = 1;
            c(i) = 0;
        end
        if (x(i+1)>=0)&&(y(i+1)>=0)
            s(i) = -1;
            c(i) = 1;
        end
    end
    if m(i) == 2
        s(i) = 0;
```

Hybrid Signed Binary Digit (SBD) Arithmetic Algorithm

```
for i = (n-1):-1:1
    s(n) = 0; c(n) = 0;
    if m(i) == -2
        s(i) = 0;
        c(i) = -1;
    end
    if m(i) == -1
        if ((x(i+1) == 0) && (y(i+1) == 0))
            s(i) = 1;
            c(i) = -1;
        end
        if (x(i+1) ~= 0)||(y(i+1) ~= 0)
            s(i) = -1;
            c(i) = 0;
        end
    end
    if m(i) == 0
        s(i) = 0;
        c(i) = 0;
    end
    if m(i) == 1
        if ((x(i+1) == 0) && (y(i+1) == 0))
            s(i) = 1;
            c(i) = 0;
        end
      end
    if m(i) == 2
        s(i) = 0;
```

c(i) = 1;
    end
end
for i = n:-1:2
    z(i) = s(i-1) + c(i);
end
	c(i) = 1;
	   end
	end
	for i = n:-1:2
	    z(i) = s(i-1) + c(i);
	end