I J C T A, 9(13) 2016, pp. 5955-5966

Reviewing Carryfree High-Radix Signed & Unsigned Digit Adders

Palleti Srinidhi* **and U. Hari****

ABSTRACT

Higher radix values of the form b ¼ 2*r* have been employed traditionally for recoding of multipliers, and for determining quotient- and root-digits in iterative division and square root algorithms, usually only for moderate values of *r*, like 2 or 3. For fast additions, in particular for the accumulation of many terms, generally redundant representations are employed, most often binary carry-save or borrow-save, but in a number of publications it has been suggested to recode the Addends into a higher radix. It is shown that there are no speed advantages in doing so if the radix is a power of 2, on the contrary, there are significant savings in using standard 4-to-2 adders, even can save half of the operations in multi-operand addition. In pre simulation results shows up to the range of -7 to 7. And in this paper extending the range of additions from -127 to 127, & implementing stopwatch watch application using the circuit.

Keywords: Multi-operand addition, signed-digit, carry-save, high-radix

I. INTRODUCTION

In the implementation of multiplication, by recoding the binary multiplier into a higher radix b ¼ 2r, the number of multiplicand multiples to be accumulated is reduced by a factor r. However, the advantage of recoding the multiplier into a radix higher than 4 is quite limited, due to the problem of generating multiplicand multiples.

Generally accumulation is performed in a tree structure using redundant representations, and thus in time proportional to the logarithm of the number of addends. In division and square root algorithms with digit-wise quotient or root determination, the number of algorithm cycles is reduced by a factor of r, corresponding to the number of digits to be determined being reduced.

- Recoding the multiplier into a radix higher than 2 (without possible of signed radix operands)is quite limited, due to the problem of generating multiplicand multiples
- In the implementation of multiplication, by recoding the binary multiplier into a higher radix b ¼ 2r, the number of multiplicand multiples to be accumulated is reduced by a factor r
- The advantage of recoding the multiplier into a radix higher than 4
- Each sum digit is rewritten by emitting a carry /transfer-digit1 in f_1; 0; 1g, and an incoming carry is added to the modified digit.

1.1.1 Overview of High radix signed & unsigned digit adders

Note that in such applications, the encoding of output digits is supposed to be the same as that of the input digits, to allow for repeated additions. The use of high-radix, signed-digit redundant digit sets for addition

* M.tech, VLSI Design, Electronics and Communication Department, SRM University, Chennai, Tamil Nadu, *E-mail: sreenidhi.palleti@gmail.com*

** Asst. Prof (S.G), Electronics and Communication Department, SRM University, Chennai, Tamil Nadu, *E-mail: hari.u@ktr.srmuniv.ac.in*

in radix values of the form 2r for r _ 2 has over the years been proposed and investigated in a number of publications, which we shall analyze here.

As the addition takes place in digit parallel, the complete addition takes place in time independent of the number of digits of the operands. Employing for radix-b ¼ 2r the maximally redundant digit set D ¼ f_2r þ 1; . . . ; 0; . . . ; 2r _ 1g, the authors encode the digits as 2's complement numbers in r þ 1 bits, where the leading bit has negative weight _2r. Each sum digit is rewritten by emitting a carry/transfer-digit1 in f_1; 0; 1g, and an incoming carry is added to the modified digit.

The authors claim to obtain high-speed addition (even claims "Ultrahigh-Speed" in the title), yet they never compare the speed against adders based on carry-save or borrow-save binary digit encodings, as generally employed in multiplier implementations and many other applications.

We shall here analyze such radix 2r, r _ 2, addition algorithms, focusing on the area and timing of the digit operations, and how these parameters depend on the chosen radix, the encoding of digits and carries. We compare such adders to equivalent adders based on combining r redundant radix-2 borrow-save (signed digit) adders over the digit set f_1; 0; 1g and carry-save adders over the digit set f0; 1; 2g. These digit sets have been used extensively in multipliers as they allow more regular binary tree structures than tree structures based directly on 3-to-2 full-adders.

These encodings also allow forming the sum of two standard binary numbers by simply pairing bits of the operands, to yield the encoding of the sum digits, thus in zero time and logic, approximately halving the area of a multi-operand addition.

1.1.2. Maximally redundant additions

Illustrated the process of adding digits in a symmetric and redundant signed-digit set D ¼ fa; . . . ; 0; . . . ; ag for radix 2r, r 2, where a ¼ 2r was chosen so that the digit set is maximally redundant.

Actually, it is sufficient that a > 2r1 for the incoming carry to be absorbed. The carries in f1; 0; 1g are determined by inspection of the intermediate sum digits, obtained by adding the two operand digits, and the carry is emitted in some suitable encoding, leaving behind a modified digit of reduced range which can absorb the incoming carry.

- We shall here look at the MRSD designs, choosing only the two fastest designs, respectively the D (using borrow-save encoding of the carry), and the F design (using 2's complement), both illustrated.
- The HA-triangles are half-adders and the FA-boxes are full adders, possibly with inverters on negative signals, but not shown. The L-shaped boxes contain the logic for determining the carry bits to emit, and two modified leading operand bits of negative weight.

1.2. Carry Free High Radix Addition

1.2.1. F Design and D Design

In these algorithms and in many other, there is a need for repeated additions or subtractions. Traditionally, the redundant binary carry-save or borrow-save representations are being employed, allowing constant-time operations. Such additions are often denoted "carry-free additions".

- Higher radix values of the form b ¼ 2r have been employed traditionally for recoding of multipliers, and for determining quotient- and root-digits in iterative division and square root algorithms, usually only for quite moderate values of r, like 2 or 3.
- For fast additions, in particular for the accumulation of many terms, generally redundant representations are employed, most often binary carry-save or borrow-save, but in a number of publications it has been suggested to recode the addends into a higher radix.

- It is shown that there are no speed advantages in doing so if the radix is a power of 2, on the contrary, there are significant savings in using standard 4-to-2 adders, even saving half of the operations in multi-operand addition
- Starting with the redundant radix 2 addition, several possible (e.g., 2-bit [17], and even 3-bit encodings of the digits are possible, some of which were investigated.
- All of these were shown to be feasible for addition in what have been denoted 4-to-2 adders, realizable by simple modifications of a carry-save, 4-to-2 adder for the addition of two operands over the digit set f0; 1; 2g, as shown in Fig. 2. Using the two-bit encoding, with the digit value being the sum of the encoding bits, this type of adder was originally proposed by wein-burger.
- There are several possible implementations of it, including some very efficient ones based on pass transistor based selectors 0.

II. EXISTING AND PROPOSED DESIGN OF ADDER

Decimal computer arithmetic and the supporting hardware units are once again in the forefront of commercial, financial, scientific, and internet-based applications [1]. The current trend is mirrored, in the industry, by commercialization of digital processors with embedded decimal arithmetic units (e.g., IBM z900 eServer [2], power6 [3] and z10 [4]), and in the literature, via the state of the art parallel decimal multipliers (e.g., [5] and [6]), dividers (e.g., [7], [8] and [9]), function evaluation and CORDIC [10] hardware. In both decimal and binary arithmetic, partial products in multipliers and partial remainders in dividers are often represented via a redundant number system (e.g., Binary signed digit [11], decimal carrysave [5], double-decimal [6], and minimally redundant decimal [9]). Such use of redundant digit sets, where the number of digits is sufficiently more than the radix, allows for carry-free addition and subtraction as the basic operations that build-up the product and remainder, respectively. In the aforementioned works on decimal multipliers and dividers, inputs and outputs are non-redundant decimal numbers. However, a redundant representation is used for the intermediate partial products or remainders. The intermediate additions and subtractions are semi-redundant operations in that only one of the operands as well as the result is redundant. In contrast there are fully redundant add/subtract schemes, where both operands and certainly the result are represented via redundant decimal digit sets (e.g., [12], [13] and [14]). Fully redundant decimal addition is also used within a sequential decimal multiplier [15], where partial products are represented in Svoboda's decimal signed digit encoding [12]. However, we have not encountered any fully redundant radix-10 multiplier or divider in the literature. In a comprehensive study on the frequency of arithmetic operations of a general computation [16], it has been shown that add/subtract operations occur more frequently than multiplication and division. Therefore, carry-free addition/subtraction can have a great impact on the overall execution time. However, there are usually two problems:

2.1.2. PROBLEM 1 (Storage of redundant results)

- The redundant result of an addition/subtraction is not necessarily used immediately as the operand of a subsequent operation. Therefore, it should be appropriately stored for later use. But, redundant results often require wider storage words or registers due to extra redundancy bits within a digit. This may not be required when the radix of the
- Number system is not a power of two. For example, in radix-10 arithmetic, at least four bits are required.
- For representation of a 10-valued non redundant decimal digit.
- Therefore, there are possibly six extra 4-bit codes available to be assigned to extra digits of a redundant decimal digit set.

- For instance, the digit set [–7, 7] has been used in the design of a fully redundant decimal adder and for representation of quotient digits in the design of a non-redundant radix-10 divider [8], where each digit is represented as a 4-bit two's complement number.
- The overloaded decimal digit set used in represents another example of a redundant decimal encoding with no extra redundancy bit. There are however, redundant decimal digit sets that use more than four bits per digit.

2.1.3. PROBLEM 2 (Intermixed operations)

- Other operations such as multiplication and division may be intermixed with additions and subtractions.
- Therefore, use of conventional multipliers and dividers that require non redundant operands enforces the conversion of redundant results, of add/subtract operations, to conventional non redundant format. The conversion may require word wide carry propagation that may jeopardize the speed gained via carry-free add/subtract operations.
- This problem can be avoided if we keep the redundant results intact when they are needed as operands of other operations such as multiplication or division; hence the necessity to design fully redundant multipliers and dividers.

2.2. Block Diagram of Proposed Design

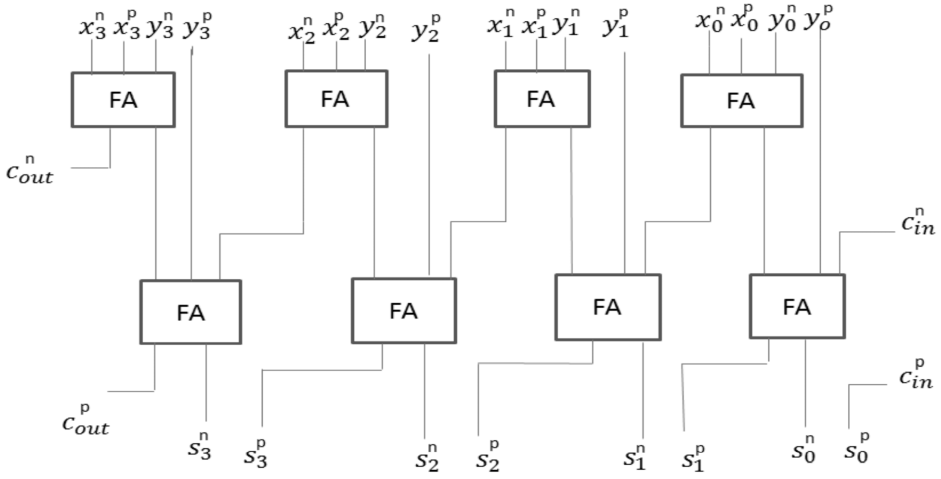

Figure 1: A radix 16 digit adder design in 4-2 borrow save encoding

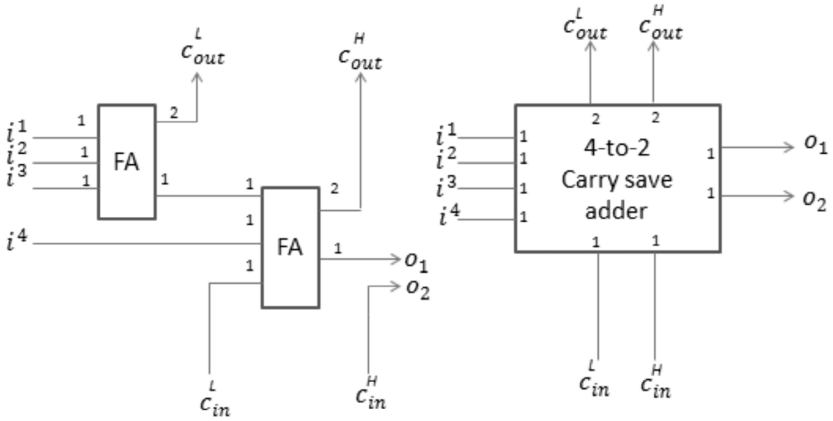

Figure 2: Box representation of 4-2 full adder

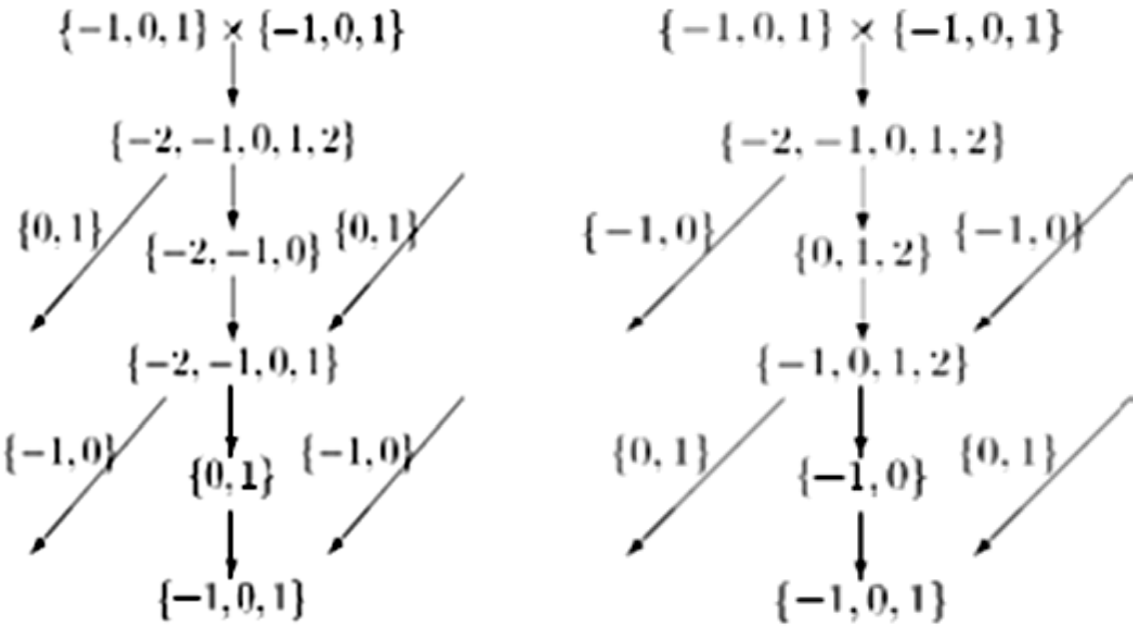

Figure 3: For radix 2, with the digit set f 1*;* 0*;* 1g, the situation seems more complicated. Following Avizienis [1], the digit set conversion here must take place in two phases for a total of five steps, which takes two forms, depending on the sign of the incoming carry

2.2.1. Methodologies

The HA-triangles are half-adders and the FA-boxes are full adders, possibly with inverters on negative signals, but not shown. The L-shaped boxes contain the logic for determining the carry bits to emit, and two modified leading operand bits of negative weight. Observe that the internal adders are connected in a ripple-carry structure, due to the digit representation being the non-redundant 2's complement, hence the timing of the digit adder is OπrÞ, i.e., linear in r. But the digit addition may also be performed using a carry look-ahead structure, thus in time Oπlog rÞ, at some increase in area. To obtain a faster addition and carry absorption, it is necessary to use a redundant digit representation. One such possibility for adding radix-16 digits, is combining four 4-to-2 borrow-save adders over the digit-set f_1; 0; 1g, which together provides an equivalent radix-16 digit adder over the same maximally redundant digit set as above. Since each digit now is encoded in 8 bits, compared to 5, a radix-16 digit adder now has 16 input and eight output lines, compared to respectively 10 and 5. Thus the interconnect structure of the adders is more complex and requires more area. But note that only half as many such adders may be needed in a multi-operand addition array, since the sum of two standard non-redundant binary operands is simply obtained by pairing the bits of the operands, directly forming their sum in carry-save encoding (or by inverting one operand in borrow-save), to be used as further input.

- Here our process is dealing with high multiple redundant additions. When adding a multiple of operands, say k of n digits, the fastest possible way to do it is to add them in a binary tree, each addition performed using a redundant representation of the intermediate sums, to allow constant time addition at the nodes of the tree, with bounded carry-propagation between Positions, i.e., without any "ripple-effect".
- We will here concentrate on adding a pair of digits from two operands, where it alternatively is possible to accumulate several digits of the same weight in some redundant representation, and only at the end converting the digit sums (as over-redundant digits) back into the wanted digit set, as suggested for decimal multi-operand addition.
- We are assuming that the initial operands are in some non-redundant representation, at most needing some constant time conversion or recoding of the non-redundant representation. The leaves of the

tree must be able to add two such addends, with their sum in the chosen redundant representation employed internally in the tree.

- At the root of the tree, the final result must in general be converted back to some non-redundant representation, most likely the same as that of the original operands. The accumulation of k, n-digit operands can thus be performed in time oðlog k), with final conversion in time oðlog n).

2.2.2. Advantages In Proposed System

1. Better performance in power consumption
2. Less number of gate counts.

2.2.3. Proposed Design

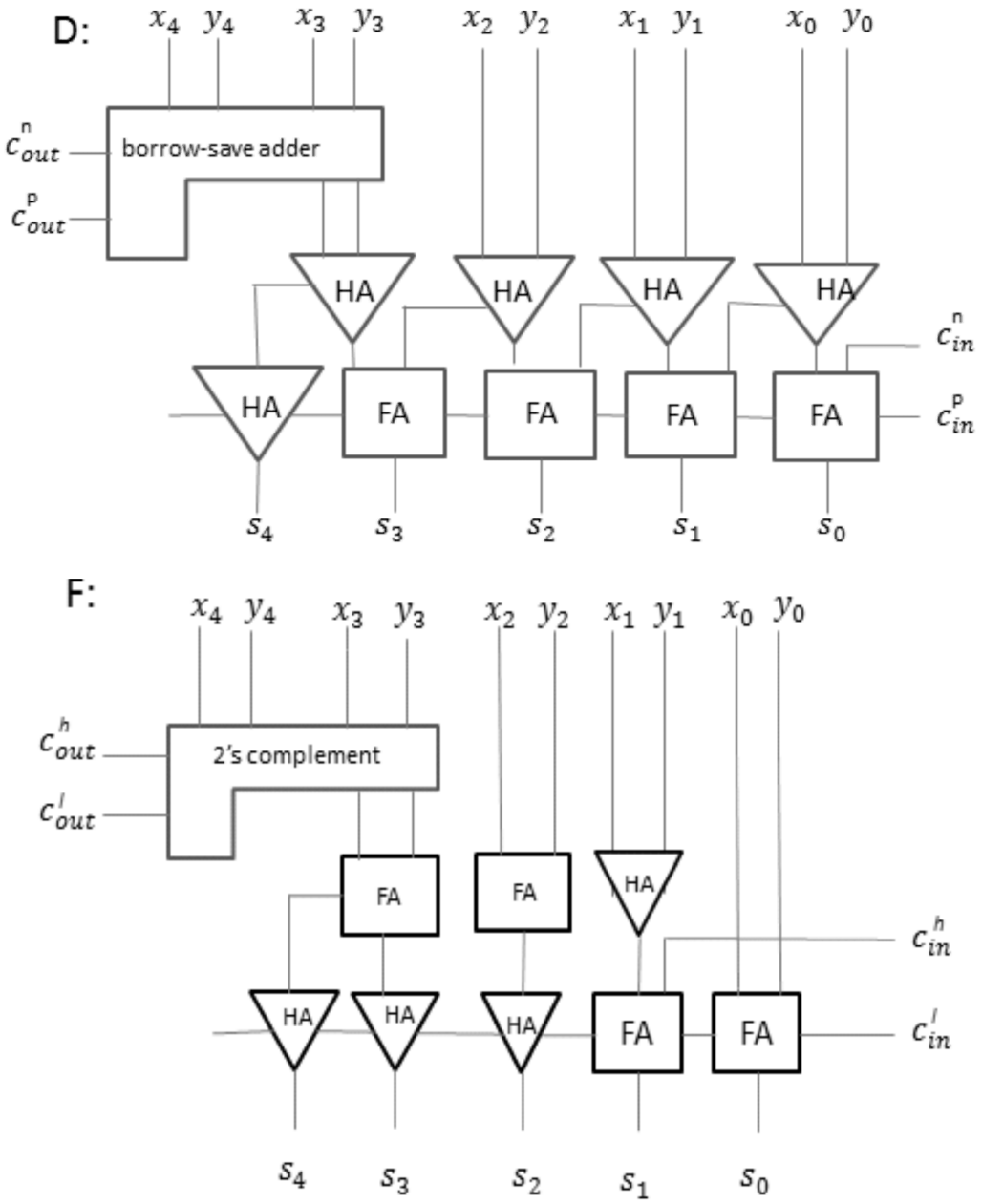

Figure 4: The radix 16 Digit Adder circuit

2.2.4. Uses of Borrow-save Adder

- We aim at showing the interest of using the BS representation to optimize subtractions. It is based on two assumptions.
- First of all, the BS architectures are as optimized as the CS ones.
- Second of all, the use of the CS architectures leads to the introduction of Inverters in arithmetical chains (because, in order to handle subtractions, it is done after having changed the subtractions into additions using two's complement), which can block possible optimizations, such as shown in Following diagram.
- It shows that the use of the BS representation allows avoiding that issue.

2.3. Borrow Save Adders Algorithm

- Carry Select Adder (CSLA) is one of the fastest adders used in many data-processing processors to perform fast arithmetic functions. From the structure of the CSLA, it is clear that there is scope for reducing the area and power consumption in the CSLA.
- Carry select adder is transformed as a borrow save adder in our design by bubbling the input and output ports

We are assuming that the initial operands are in some non-redundant representation, at most needing some constant time conversion or recoding of the non-redundant representation. The leaves of the tree must be able to add two such addends, with their sum in the chosen redundant representation employed internally in the tree. At the root of the tree, the final result must in general be converted back to some non-redundant representation, most likely the same as that of the original operands.Borrow-Save (BS) is a radix-2 signed-digit redundant representation [10]. The integer X is represented by (xl–1 · · · x1x0)BS where the l digits xi are in {–1, 0, 1} and coded using 2 bits x + i and x – i such that

$$X = \sum_{i=0}^{I-1} x_I 2^i = \sum_{i=0}^{i-1} \left(x_i^+ - x_i^- \right) 2^i$$

ALGORITHM 1

Double-and-add

Input: P, k = (kl–1...k1k0)2

Output: Q = [k]P

1. Q ← ∞

2. for i = l – 1 downto 0 do

3. Q ← [2]Q 4. if ki = 1

then 5. Q ← Q ⊕ P

6. end if

7. end for

8. return Q

Borrow-Save Addition (BSA) can be performed using the constant time algorithm presented in Algorithm 2 and illustrated on Figure 1. Due to the signed-digit representation, –X is obtained by swapping x + i and x – i for all ranks I.

ALGORITHM 2

Borrow-Save Addition

Input: X = (xl–1...x1x0)BS and Y = (yl”1...y1y0)BS

Output: S = (sl ...s1s0)BS = X + Y

1. c + 0 ← 0, s– 0 ← 0
2. for i = 0 to l – 1 do B parallel loop
3. 2c + i+1 – c – i P PM ←——— x + i + y + i “ x “ i
4. end for
5. for i = 0 to l – 1 do B parallel loop
6. 2s – i+1 – s + i P PM ←—— y – i + c – i – c + i
7. end for
8. s + l ← c + l 9. return S

- The BSA operator depicted on Figure 1 uses 2 rows of PPM cells [10]. A PPM is very close to a full-adder (just 1 extra inverter) and computes $2c \pm - s \mp = x \pm + y \pm - x \mp$. A logical implementation can be $s = x \pm \oplus y \pm \oplus x \mp$ and $c = x \pm y \pm + x \pm x \mp + y \pm x \mp$. Figure 1 clearly shows that the BSA computation time does not depend on the operand size l and is $T(BSA(l)) = 2 \cdot T(PPM)$
- The most significant guard digit can be eliminated expanding the 4-to-2 adders in terms of full adders with their interconnections, as an equivalent to the above digit adders in Fig. 5, using the borrow-save encoding we find Fig. 6, again not showing inversions on negative signals. For general r the delay of the circuit in Fig. 6 is independent of the radix 2r, and identical to that of a single 4-to-2 adder. Obviously, when generalized as a function of radix 2r the delays of the circuits in Fig. 5 will be linear in r, whereas the delays in Fig. 6 are constant.
- Specialized for r ¼ 1, the circuit in Fig. 5Fmust be functionally equivalent to the circuit in Fig. 6 for r ¼ 1, such that the latter circuit corresponds to Fig. 4 with 2’s complement encoding of operands.
- Thus optimal implementations of the two circuits must have identical delays. For r ¼ 2 the circuit in Fig. 5F must be slower than for r ¼ 1, as signals have to go through additional logic.
- It is impossible that even if employing a Manchester carry chain or some carry look ahead structure, that the logic can be faster for r ¼ 2 than for ¼ 1, and thus faster than the circuit For r _ 2, obviously the circuit of Fig. 6 will be faster than both of those shown in Fig. 5, due to the non-redundant representation of the digits in these.
- When using the Fig. 6 design there is no need to group r bits of the operands, standard binary addends can immediately be used, whereas in the designs of Fig. 5 it is necessary to split the operands and convert the digits into radix 2r, 2’s complement.
- Recall also that, when using the Fig. 6 design, half of the additions essentially come for free.
- When performing multiple additions using the borrow-save digit encoding, it is necessary to use two leading guard digits, beyond those in a constant time operation, if non-zero by just inverting the digit in the next position.
- But eliminating the least significant guard digit is at best a log-time operation, equivalent to converting the sum into a non-redundant representation. However, when high radixes like 2r; r _ 2 or 10 is employed, a single guard digit is sufficient.

III. SIMULATIONS AND RESULT

3.1. Design of D Circuit

3.1.1. Schematic of D circuit

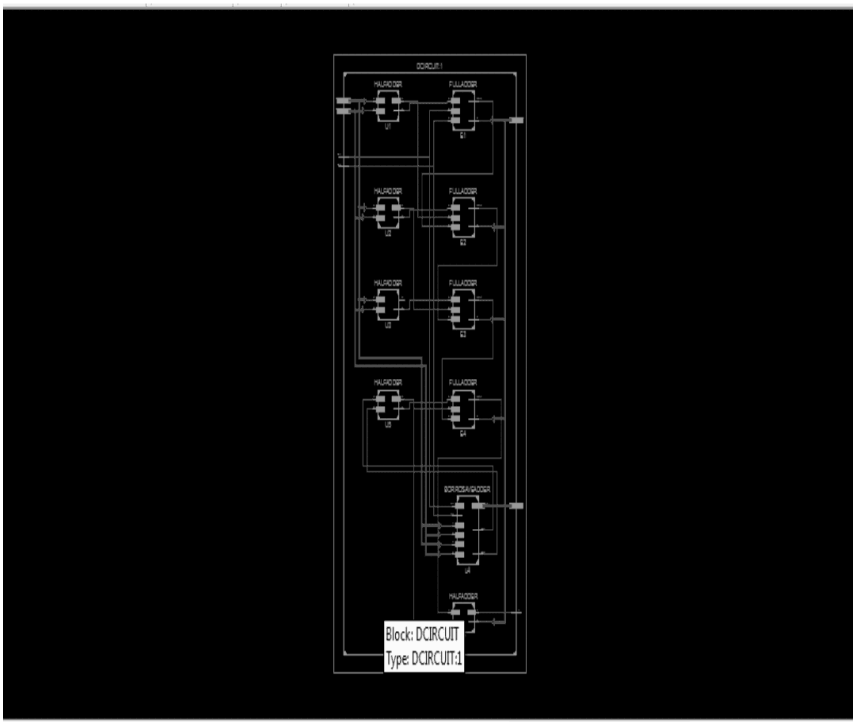

Figure 5: Schematic of D-Circuit

3.1.2. RTL Design

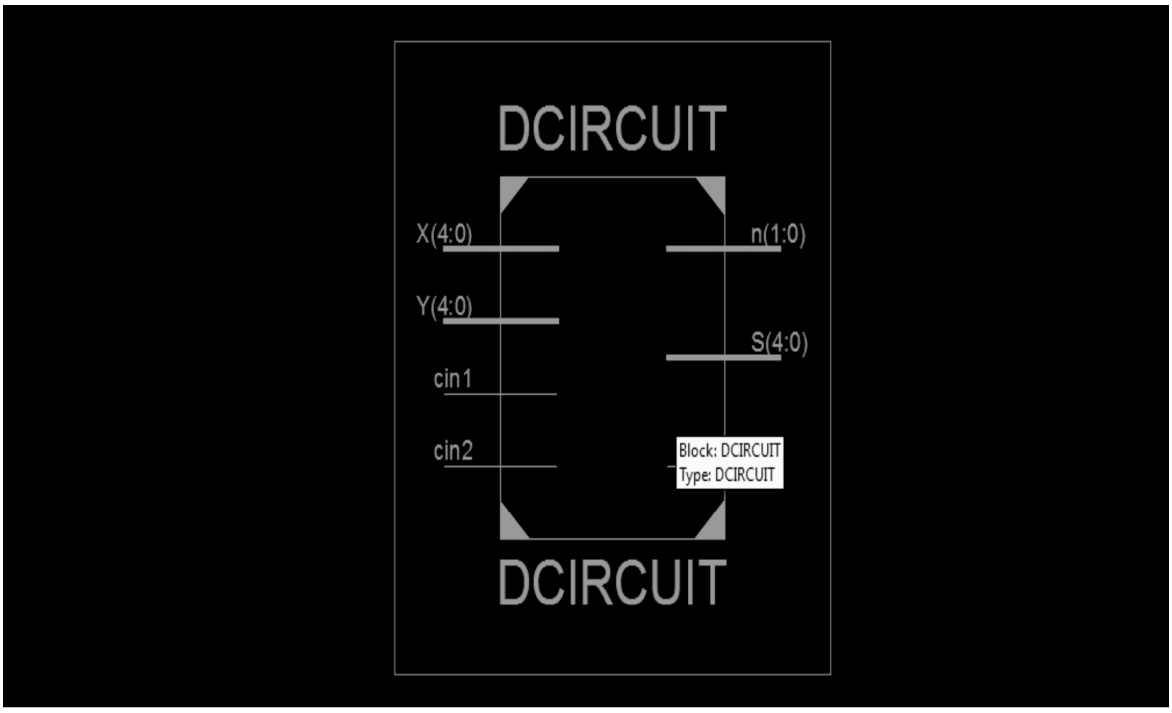

Figure 6: RTL design of D Circuit

3.1.3 Waveforms of D-Circuit

Figure 7: Simulation result of D-Circuit

3.2. Design of F Circuit

3.2.1. Schematic of F-Circuit

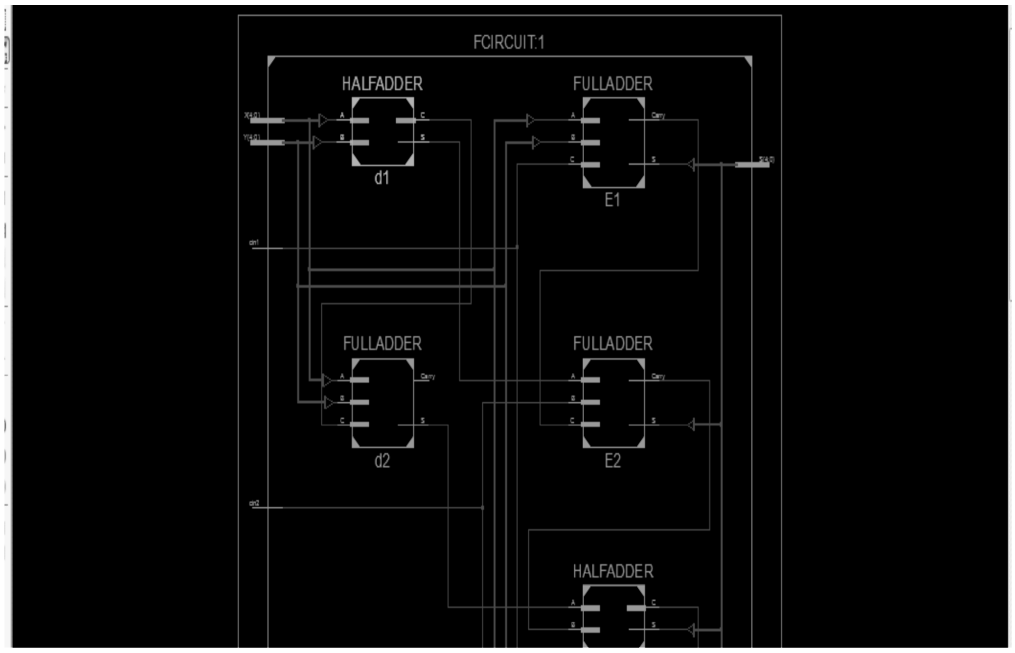

Figure 8: Schematic of D-Circuit

2.2. RTL Design

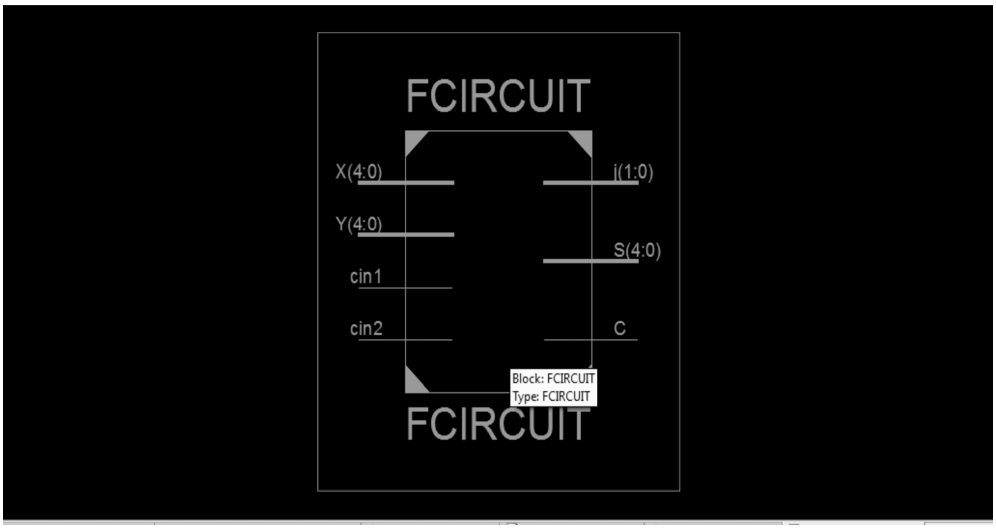

Figure 9: RTL Design of D-Circuit

3.2.3. Waveforms of F-Circuit

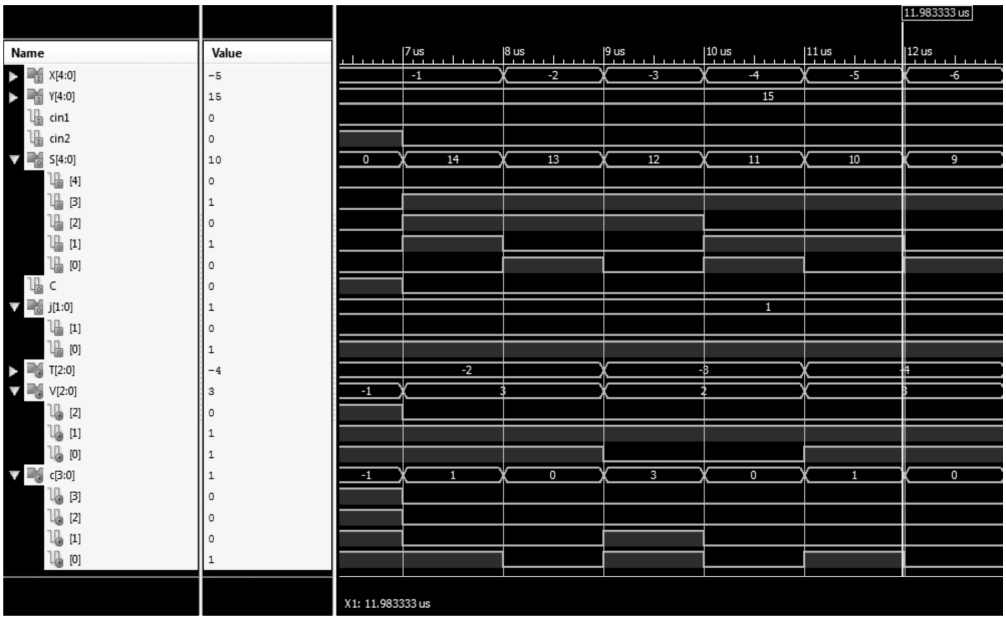

Figure 10: Simulation result of D-Circuit

3.3. Results

3.3.1. Delay of D & F Circuits

Delay of D Circuit: 7.871NS

Delay of F Circuit: 6.652NS

CONCLUSION

It has been shown, that the proposed recoding of addends into a higher radix of the form 2r for r _ 2, with digits encoded on-redundant 2's complement representation, cannot provide faster addition than using

standard radix 2, carry-save or borrow-save adders applied directly on non-redundant binary addends. On the contrary, for multi-operand addition the delay is significantly larger when using such recoding into a higher radix, despite the claim "Ultrahigh-Speed" in the title. Furthermore, when employing the carry- or borrow-save encoding, half of the additions come for free, since just pairing two non-redundant binary numbers forms their sum in carry save, or (with one of them inverted) in borrow-save. It is noted that (except for some inverters at the boundary) the logic of the adder array is identical, whether the carry-save or the borrow save representation is used. The situation is different for decimal addition. Encoding decimal digits in the set f -127; . . . ; 0; . . . ; 127g in the compact 2's complement representation, has been used efficiently in [3], Finally, note that this analysis does not apply to FPGA implementations, in particular due to the availability of small fast carry-ripple adders in this technology. And implementation of 2's compliment circuit in stopwatch circuit apply to FPGA as an application.

REFERENCES

[1] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, no. 3, pp. 389–400, Sep. 1961.

[2] M. Ercegovac and T. Lang, "Effective coding for fast redundant adders using the radix-2 digit set f0*;* 1*;* 2*;* 3g," in *Proc. 31st Asilomar Conf. Signals Syst. Comput.*, 1997, pp. 1163–1167.

[3] S. Gorgin and G. Jaberipur, "Fully redundant decimal arithmetic," in *Proc. 19th IEEE Symp. Comput. Arithmetic*, Jun. 2009, pp. 145–152.

[4] S. Gorgin and G. Jaberipur, "A family of high radix signed digit adders," in *Proc. 20th IEEE Symp. Comput. Arithmetic*, Jul. 2011, pp. 112–121.

[5] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi, "A high- speed multiplier using a redundant binary adder tree," *IEEE J. Solid State Circuits*, vol. SSC-22, no. 1, pp. 28–34, Feb. 1987.

[6] G. Jaberipur and S. Gorgin, "An improved maximally redundant signed digit adder," *Comput. Elect. Eng.*, vol. 36, pp. 491–502, May 2010.

[7] G. Jaberipur and B. Parhami, "Stored-transfer representations with weighted digit-set encodings for ultra high-speed arithmetic," *IET Circuits Devices Syst.*, vol. 1, no. 1, pp. 102–110, Feb. 2007.

[8] G. Jaberipur, B. Parhami, and M. Ghodsi, "Weighted two-valued digit-set encodings: Unifying efficient hardware representation schemes for redun- dant number systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 7, pp. 1348–1357, Jul. 2005.

[9] P. Kornerup and J.-M. Muller, "Leading guard digits in fi te precision redundant representations," *IEEE Trans. Comput.*, vol. 55, no. 5, pp. 541–548, May 2006.

[10] P. Kornerup and D. Matula, *Finite Precision Number Systems and Arithmetic*, vol. 133 of *Encyclopedia of Mathematics and its Applications*. Cambridge, U.K.: Cambridge Univ. Press, Sept. 2010.

[11] P. Kornerup, "Reviewing 4-to-2 Adders for Multi-Operand Addition," *J. VLSI Signal Process.*, vol. 40, pp. 143–152, May 2005, Previously presented at ASAP2002.

[12] R. D. Kenney and M. J. Schulte, "High-speed multioperand decimal add- ers," *IEEE Trans. Comput.*, vol. 54, no. 8, pp. 953–963, Aug. 2005.