Redundant Radix-4 Representation With High Speed Arithmetic Coprocessor Using Carry Save And Redundant Signed Digit Technique

Ashish Manoharrao Ingale

Abstract: Division is the "inverse" of multiplication so basic division consist of a sequence of subtraction, which are just additions of the negations of the subtrahends; Therefore, CS addition can be used in the arithmetic for division as well. Division is, however more complicated than multiplication, in that subtrahend (multiple of divisor) chosen at any steps depends on the magnitude result of the preceding subtraction and that magnitude is not readily available with CS representation. Assuming two's complement representation, subtraction with carry save representation is carried out in usual manner of the farming the one's complement of the subtrahend and then adding that with a 1 also added into the least significant bit position subtraction is performed by adding the negation of the subtrahend, which two's complement representation consist of the one's complement addition of 1 in the least significant bit position.

Index Term: Redundant Radix-4 (RR-4), Carry Save (CS), Redundant Signed Digit (RSD), Carry Propagate Adder (CPA).

1. Introduction

The RSD representation, first introduced by Avizienisis a carry free arithmetic where integers are represented by the difference of two other integers. An integer X is represented by the difference of its x+ and x− components, where x+ is the positive component and x− is the negative component. The nature of the RSD representation has the advantage of performing addition and subtraction without. the need of the two's complement representation On the other hand, an overhead is introduced due to the redundancy in the integer representation; since an integer in RSD representation requires double word length compared with typical two's complement representation. In radix-2 balanced RSD represented integers, digits of such integers are either 1, 0, or −1. Carry-save arithmetic is frequently used in multiplier, multiply-add, multiply accumulate, and digital filter design. Implementing such arithmetic operations implies the need of reducing partial products. When solving this by carry-save addition, different carry-save reduction strategies can be applied. Digital designers should choose a strategy that yields favorable results in terms of area, latency, and low power consumption. We contribute our own level-based carry-save reduction method. We provide a software tool for time-efficient analysis and rapid prototyping of carry-save arithmetic using the presented reduction strategies. We show results in terms of the expected area, latency, and power consumption, gained by our tool employing to multiplication and various multiply accumulate implementations and outline the relevance for low power design. We include timing issues into our strategy and present a list-based approach for reducing partial products.

We state several heuristics for list-based reduction strategies and implement them into a second software tool. We apply this tool to multiply-accumulate implementations, further improving their area, latency, and power consumption characteristics.

2. RR-4 Sequential Multiplier

The multiple formation unit forms 2MD by ones bit wired shift and also produces the require complement. The 1 is included in the small CPA, in the free slot created by right shift. A carry output of the CPA in one cycle is saved say in flip-flop and become its carry input in the next cycle. In the least cycle carry out of the 2-bit CPA become a carry into the CPA used to complete the top half of product.

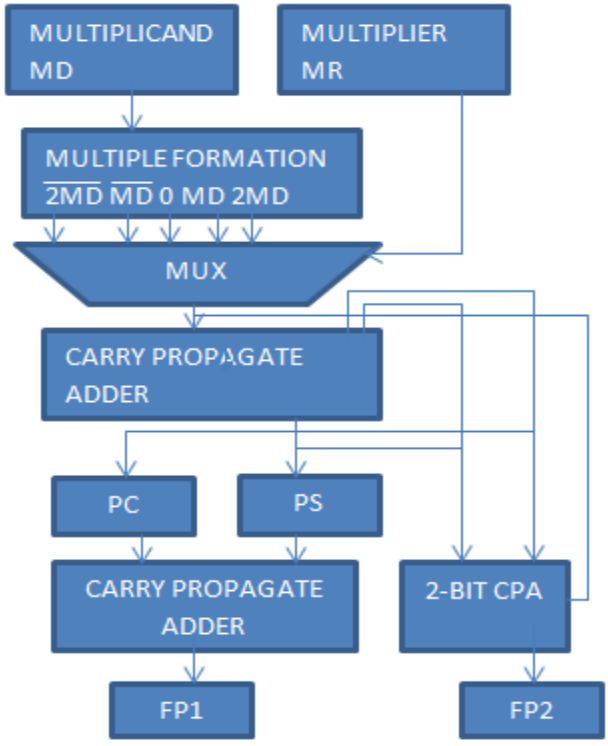

Figure 2.1: *RR-4 Sequential Multiplier*

- *Ashish Manoharrao Ingale, Research Scholar, Master of Technology in Electronic System and Communication, Government College of Engineering, Amravati.*

Depending on the particular situation, the addition of the 1 may be a “delicate”. RSD is also used to perform division, multiplication, subtraction, addition. The multiplicand and Multiplier are held in two registers, MD and MR, with latter shift register. Two registers, PC and PS, hold the partial carry and partial sum from the addition in each cycle and are initialized to 0’s and final product is held in two registers FP1 and FP2, with FP2 an shift register. Each consists of the following steps.

1) The three least significant bits of the multiplier are used to select multiplicand multiple.
2) The selected multiple, current product partial sum and current product partial carry are added in carry save adder, to produce new partial sum and partial carry.
3) The two least significant bits of the partial sum/ partial carry output make up the corresponding bits of the final product. Accordingly they are assimilated in a 2-bit carry propagate adder and inserted into FP2, whose contents are then shifted two bit places to right.
4) The content of MR are shifted two bits places to right. At the end of n cycles, the lower n bits of the final product will be in FP2, and the upper n bits are obtained by using a carry propagate adder to assimilate the final partial sum and partial carry bits.

3. Extension to the Applicability of CSA

- Condition 1: A must be one of addition and subtraction operations.
- Condition 2: B must be one of addition, subtraction, and multiplication operations.
- Condition 3: Output of B must be single fanout. That is, B drives only an input of A.
- Condition 4: There should be no \leak" of data values through the connections from B to A up-to the output bit-width of root of the tree.

Transformation Many signal processing and data-intensive computations frequently contain an operation in which only the upper m bits among n bits of its output are fed input to a descendent operation. We call this partial use of an output the lower-bit truncation problem. More- over, many designs often contain an operation whose output feeds several other operations. We call this the multiple fan out problem. In fact, the two problems correspond to the violations of Condition 4 and Condition 3, respectively. In the following, we provide solutions to the problems. We begin with an example to demonstrate how our approach handles operations with multiple fan out. Suppose that is a partial structure of a design that we are going to transform into CSAs. Note that operation O3 has multiple fan out. Consequently, two CSAs and two adders (one for each CSA tree) are used. Suppose that a design contains a chain of operations in which n of them have multiple fan out. Then, our algorithm will identify n operation trees having each operation with multiple fan out as root. Consequently, n adders will be allocated on the chain of the transformed CSA trees. Here, our objective is that we want to reduce the number of adders from n to 1 and replace the remaining n-1 adders with CSAs. We accomplish this by introducing the concept of CSA transformation without adder. However, when an operation with multiple fan out is encountered (i.e., Condition 3), we mark the operation as a root of another operation tree and continue to expand the operation tree. Once we collect all the operation trees by crossing over operations with multiple fan out, we topologically sort the operation trees from input boundary of design to output boundary. We then transform the operation trees on the list one by one. For those trees with multiple fan out root, we do not allocate n adders in the resultant CSA trees. Instead, we generate two outputs of each CSA tree. The two outputs are then used in two ways:

(1) Both of them are used as input operand of the operations trees which depend on the operation tree corresponding to the transformed CSA tree without adder and
(2) A new adder is created and they are used as input of the adder. (we refer this process to as operation-duplication.) The output of adder is then used for the fan out of the root of operation tree corresponding to the CSA tree. Consequently, only one adder will be created at the end of the last operation tree of the sorted list of trees. Note that the resultant transformation contains three CSAs and one adder in the two CSA trees and generates a faster timing than the one. However, the total area increases because one additional adder is created outside of the CSA trees.

4. CARRY FREE ADDITION USING REDUNDANT BINARY SIGNED DIGIT

The redundant binary representation (RBR) is a numeral system that uses more bits than needed to represent a single binary digit so that each number will have several representations. RBR is a place-value notation system. In RBR digit set will have more digits than the radix and digits are pairs of bits, that is, for every place RBR uses a pair of bits. RBR is unlike usual binary numerical systems, including two’s complement, which use single bit for each digit.

5. DESIGN AND IMPLEMENTATION OF RBSD BASED ALU

5.1 Design of one digit RBSD based ALU

The one digit ALU is designed by VHDL and is shown in Figure 5.1.

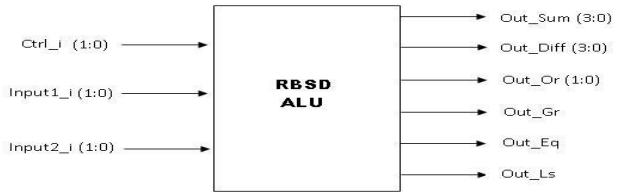

Figure 5.1: *One digit RBSD ALU pin out*

One digit (two bits) ALU has two input bit vectors Input1_i and Input2_i having width of two. It has one control input Ctrl_i also having width of two. Control input is used to select the required arithmetic operation. The designed ALU has adder, subtractor, OR gate and comparator. If the control is „00" adder unit will be activated. If it is „01" subtractor unit will be activated. If it is „10" OR gate will be activated. If it is „11" comparator will be activated. One digit ALU contains six outputs. Out_Sum is the output of the adder having the width of four. Out_Diff is the output of the

subtractor having the width of four. Out_Or is the output of the OR gate having the width of two. Out_Gr, Out_Eq, Out_Ls, are the three outputs of comparator. Simulation is done using ModelSim XE III 6.2g Simulator Figure 5.2 shows the simulation result for adder unit of one digit RBSD ALU. Input1_i = -1, Input2_i = -1,0,1 Input1 will be -1. As shown in table 1 redundant digit -1 is represented as 00 Input2 is (i) -1 (00), (ii) 0 – (01 or 10) (iii) 1 – (11)

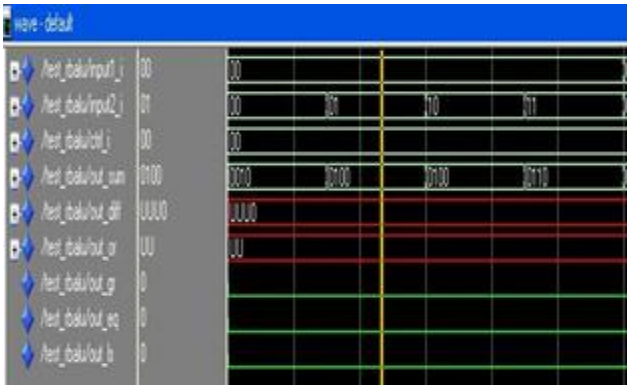

Figure 5.2: *Simulated Result for adder Input1_i=-1 Input2_i =-1, 0, 1*

Figure 5.3 shows the simulation result for subtractor unit of one digit RBSD ALU. Input1_i = -1, Input2_i = -1, 0, 1

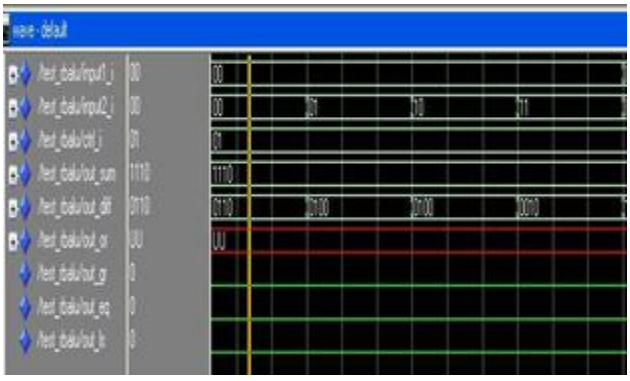

Figure 5.3: *Simulated Result for subtractor Input1_i=-1 Input2_i =-1, 0, 1*

Figure 5.4 shows the simulation result for OR gate of one digit RBSD ALU. Input1_i = -1, Input2_i = -1, 0, 1

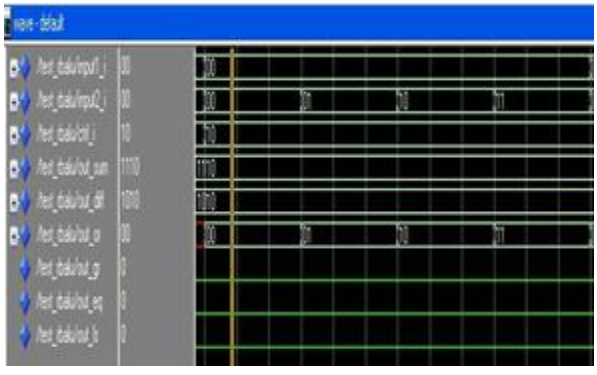

Figure 5.4: *Simulated Result for OR gate Input1_i=-1 Input2_i =-1,0,1*

Figure 5.5 shows the simulation result for comparator of one digit RBSD ALU. Input1_i = -1, Input2_i = -1,0,1

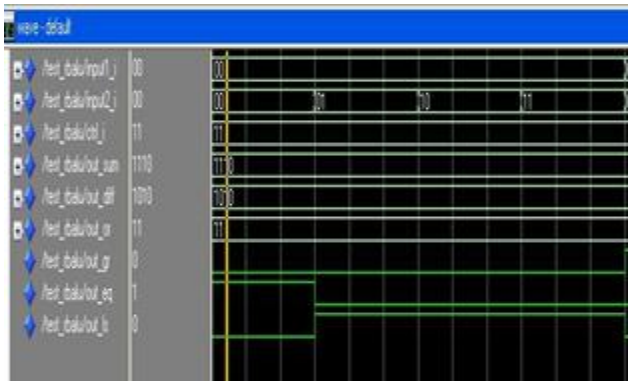

Figure 5.5: *Simulated Result for Comparator Input1_i=-1 Input2_i =-1, 0, 1*

6. Design of two digit RBSD based ALU

The two digit ALU is designed by VHDL[10] and is shown in Figure 7

Figure 6.1: *Two Digit RBSD ALU Pin Out*

Two digit (four bits) ALU has two input bit vectors Input1_i and Input2_i having width of four. It has one control input Ctrl_i also having width of two. Control input is used to select the required arithmetic operation. The designed ALU has adder, subtractor, OR gate and comparator. If the control is „00" adder unit will be activated. If it is „01" subtractor unit will be activated. If it is „10" OR gate will be activated. If it is „11" comparator will be activated Two digit ALU contains six outputs. Out_Sum is the output of the adder having the width of six. Out_Diff is the output of the subtractor having the width of six. Out_Or is the output of the OR gate having the width of four. Out_Gr, Out_Eq, Out_Ls, are the three outputs of comparator. The block diagram of two digit RBSD ALU is as shown in Figure 6.2.

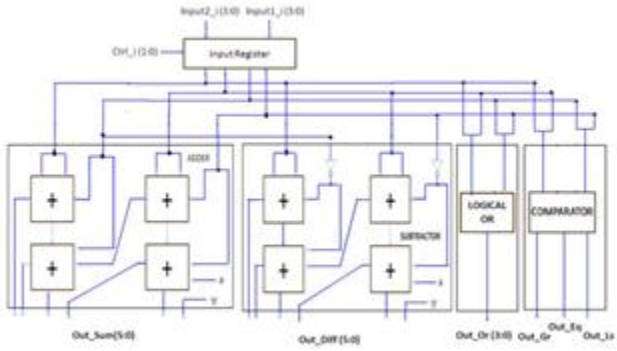

Figure 6.2: *Block Diagram of Two Digit RBSD ALU*

The two digit RBSD ALU contains adder, subtractor, logical OR and comparator block. Adder unit contains four full adders. Two digit RBSD ALU means four bit ALU. Because each RBSD digit is represented by two bits. The output of adder is six bits Subtractor unit contains four full adders and two NOT gates. NOT gates are used to find the additive inverse of one of one of the operands. After finding the additive inverse it will be added to the other operand. The output of subtractor is six bits The comparator unit will have two inputs each input of four bits. It has three outputs. It will compare two inputs and depending on the status of the inputs one of the output will be at logic high state. Simulation is done using ModelSim XE III 6.2g Simulator. The simulation results for two digit ALU are verified with its corresponding logic diagram and are shown below. Figure 6.3 shows the simulation result for adder unit of two digit RBSD ALU. Input1_i = -3, Input2_i =-3,-2,-1,0,1,2,3

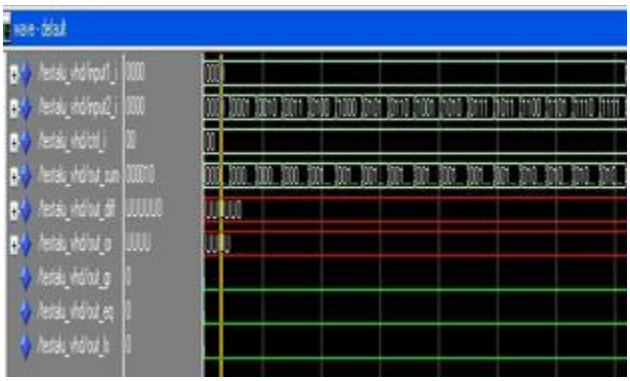

Figure 6.3: *Simulated Result for Adder of Two Digit ALU Input1_i=-3 Input2_i =-3,-2,-1, 0,1,2,3*

Figure 6.4 shows the simulation result for subtractor unit of two digit RBSD ALU. Input1_i = -3, Input2_i =-3,-2,-1, 0,1,2,3

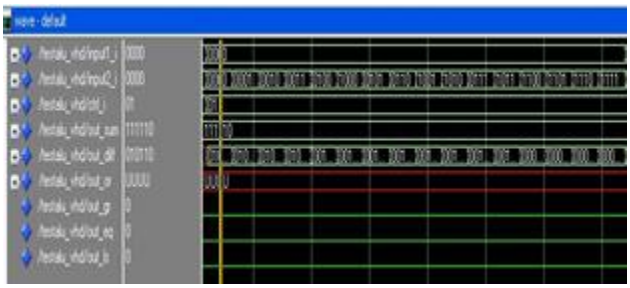

Figure 6.4: *Simulated Result for Subtractor of Two Digit ALU Input1_i=-3 Input2_i =-3,-2,-1,0,1,2*

7. Conclusion

We have seen by using RR-4 sequential multiplier performed addition, division subtraction, multiplication multi operand operation with carry propagate adder using Redundant Signed Digit and Carry Save technique.

8. References

[1] G. Jaberipur, B. Parhami, M. Ghodsi, Weighted bit-set encodings for redundant digit sets: theory and applications, in: Proceedings of 36th Asilomar Conference on Signals, Systems and Computers, 2002, pp. 1629–1633.

[2] G. Jaberipur, B. Parhami, M. Ghodsi, Weighted two-valued digit-set encodings: unifying efficient hardware representation schemes for redundant number systems, IEEE Trans. Circuits Syst. I 52 (7) (2005) 1348–1357.

[3] G. Jaberipur, B. Parhami, Stored-transfer representations with weighted digit-set encodings for ultrahigh-speed arithmetic, IET Circuits Devices Syst. 1 (1) (2007) 102–110.

[4] G. Jaberipur, B. Parhami, Constant-time addition with hybrid-redundant numbers: theory and implementations, Integration VLSI J. 41 (1) (2008) 49–64.

[5] R.D. Kenney, M.J. Schulte, M.A. Erle, A high-frequency decimal multiplier, in: IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD), 2004, pp. 26–29

[6] R.D. Kenney, M.J. Schulte, High-speed multioperand decimal adders, IEEE Trans. Comput. 54 (8) (2005) 953–963.

[7] T. Lang, A. Nannarelli, A radix-10 combinational multiplier, in: Proceedings of Asilomar Conference on Signals, Systems, and Computers, 2006, pp. 313–317.

[8] T. Lang, A. Nannarelli, A radix-10 digit-recurrence division unit: algorithm and architecture, IEEE Trans. Comput. 56 (6) (2007) 727–739.

[9] G. Metze, J.E. Robertson, Elimination of Carry propagation in digital computers, in: Proceedings of International Conference on Information Processing, Paris, 1959, pp. 389–396.

[10] H. Nikmehr, B.J. Phillips, C.C. Lim, A decimal carry-free adder, in: Proceedings of SPIE Conference on Smart Materials, Nano-, Micro-Smart Systems, 2004, pp. 786–797.

[11] H. Nikmehr, B. Phillips, C.C. Lim, Fast decimal floating-point division, IEEE Trans. Very Large Scale Integration (VLSI) Syst. 14 (9) (2006) 951–961.

[12] B. Parhami, Generalized signed-digit number systems: a unifying framework for redundant number representations, IEEE Trans. Comput. 39 (1) (1990) 89–98.

[13] R.K. Richards, Arithmetic Operations in Digital Computers, Van Nostrand Comp., Inc., 1955.

[14] M. Schmookler, A. Weinberger, High speed decimal addition, IEEE Trans. Comput. C-20 (8) (1971) 862–866.

[15] S. Shankland, IBM's POWER6 gets help with math, multimedia, ZDNet News (2006).

[16] B. Shirazi, D.Y. Yun, C.N. Zhang, RBCD: redundant binary coded decimal adder, in: IEE Proceedings on Computer & Digital Techniques (CDT), vol. 36, no. 2, 1989.

[17] I.E. Sutherland, R.F. Sproull, D. Harris, Logical Effort: Designing Fast CMOS Circuits, Morgan Kaufmann, Los Altos, CA, ISBN 1558605576, 1999.

[18] A. Svoboda, Decimal adder with signed digit arithmetic, IEEE Trans. Comput. C-18 (3) (1969) 212–215.

[19] A. Vazquez, E. Antelo, P. Montuschi, A new family of high-performance parallel decimal multipliers, in: Proceedings of the 18th IEEE Symposium on Computer Arithmetic, 2007, pp. 195–204.

[20] L. Wang, M.J. Schulte, A decimal floating-point divider using Newton– Raphson iteration, J. VLSI Signal Process. Syst. 14 (1) (2007) 3–18.

[21] C.K. Yuen, A new representation for decimal numbers, IEEE Trans. Comput.C-26 (12) (1977) 1286–1288.