Residue Number System to Speed Up Sign Detection in Digital Logic Circuits using FPGA

Mehul Kumar Pandya[1], Prof. Sachin Bandewar[3]
[1]M-Tech Research Scholar, [2] HOD & Research Guide,
Department of Electronics & Communication Engineering, SSSCE, Bhopal

Abstract- The sign detection algorithms are sort of thing get done using various algorithms and the digital logic can be synthesized on different synthesis tools. The digital algorithm designed for the sign detection is evaluated based on the area and delay profiles. The detection speed is must higher for faster calculations. In this paper a fast sign detection algorithm is proposed and the algorithm utilizes residue number system to detect sign. The proposed algorithm is designed to process 32-bits of information processing as in existing work with lower delay. The Unit Gate delay calculated of the proposed architecture is 0.196ns which is 80% better than the existing work and overall delay is 15.57ns.

Keywords - Sign Detection, Residue Number System, Delay, Unit Gate Model.

I. INTRODUCTION

In general, numbers may be signed, and for binary digital arithmetic there are three standard notations that have been traditionally used for the binary representation of signed numbers. These are sign-and-magnitude, one's complement, and two's complement. Of these three, the last is the most popular, because of the relative ease and speed with which the basic arithmetic operations can be implemented. Sign-and-magnitude notation has the convenience of having a sign-representation that is similar to that used in ordinary decimal arithmetic. And one's complement, although a notation in its own right, more often appears only as an intermediate step in arithmetic involving the other two notations.

The sign-and-magnitude notation is derived from the conventional written notation of representing a negative number by pretending a sign to a magnitude that represents a positive number. For binary computer hardware, a single bit success for the sign: a sign bit of 0 indicates a positive number, and a sign bit of 1 indicates a negative number. For example, the representation of the number positive-five in six bits is 000101, and the corresponding representation of negative-five is 100101. Note that the representation of the sign is independent of that of the magnitude and takes up exactly one bit; this is not the case both with one's complement and two's complement notations.

Sign-and-magnitude notation has two representations, 000. . . 0 and 100. . . 0, for the number zero; it is therefore redundant. With one exception (in the context of floating-point numbers) this existence of two representations for zero can be a nuisance in an implementation. Addition and subtraction are harder to implement in this notation than in one's complement and two's complement notations; and as these are the most common arithmetic operations, true sign-and-magnitude arithmetic is very rarely implemented.

In one's complement notation, the representation of the negation of a number is obtained by inverting the bits in its binary representation; that is, the 0s are changed to 1s and the 1s are changed to 0s. For example, the representation of the number positive-five in six bits is 000101 and negative- five therefore has the representation 111010. The leading bit again indicates the sign of the number, being 0 for a positive number and 1 for a negative number. We shall therefore refer to the most significant digit as the sign bit, although here the sign of a negative number is in fact represented by an infinite string of 1s that in practice is truncated according to the number of bits used in the representations and the magnitude of the number represented.

It is straightforward to show that the n-bit representation of the negation of a number N is also, when interpreted as the representation of an unsigned number, that of $2n - 1 - N$. (This point will be useful in subsequent discussions of basic residue arithmetic.) The one's complement system too has two representations for zero—00. . . 0 and 11. . . 1—which can be a nuisance in implementations. We shall see that a similar problem occurs with certain residue number systems. Addition and subtraction in this notation are harder to implement than in two's complement notation (but easier than in sign-and-magnitude notation) and multiplication and division are only slightly less so. For this reason, two's complement is the preferred notation for implementing most computer arithmetic.

Negation in two's complement notation consists of a bit-inversion (that is, a translation into the one's complement)

followed by the addition of a 1, with any carry from the addition being ignored. Thus, for example, the result of negating 000101 is 111011. As with one's complement notation, the leftmost bit here too indicates the sign: it is 0 for a positive number and 1 for a negative number; but again, strictly, the sign is actually represented by the truncation of an infinite string. For n-bit representations, representing the negation of the number N may also be viewed as the representation of the positive number $2n - N$.

In contrast with the first two conventional notations, the two's complement has only one representation for zero, i.e. 00. . . 0. The two's complement notation is the most widely used of the three systems, as the algorithms and hardware designs required for its implementation are quite straightforward. Addition, subtraction, and multiplication are relatively easy to implement with this notation, and division is only slightly less so.

All of the notations above can be readily extended to non-binary radices. The extension of binary sign-and-magnitude to an arbitrary radix, r, involves representing the magnitude in radix-r and using 0 in the sign digit for positive numbers and $r - 1$ for negative numbers. An alternative representation for the sign is to use half of the permissible values of the sign digit (that is, 0 . . . $r/2 - 1$, assuming r is even) for the positive numbers and the other half (that is, $r/2$ . . . $r - 1$, for an even radix) for the negative numbers.

The generalization of one's complement to an arbitrary radix is known as diminished-radix complement, the name being derived from the fact that to negate a number in this notation, each digit is subtracted from the radix diminished by one, i.e. from $r - 1$.

Alternatively, the representation of the negation may also be viewed as the result of subtracting the number from $rn - 1$, where n is the number of digits used in the representations. Thus, for example, the negation of 01432 in radix-8 is 76345, i.e. 77777 − 01432. The sign digit will be 0 for a positive number and $r - 1$ for a negative number. The generalization of two's complement to an arbitrary radix is known as radix complement notation.

In radix complement notation, the radix-r negation of a number is obtained, essentially, by subtracting from rn , where n is the number of digits used in the representations. Alternatively, negation may also be taken as the formation of the diminished-radix complement followed by the addition of a 1. Thus, for example, the radix-8 negation of 01432 is 76346, i.e. 100000 − 01432 or 77777 − 01432 + 1. The determination of sign is similar to that for the radix-r diminished-radix complement.

II. RESIDUE NUMBER SYSTEM

Residue number systems are based on the congruence relation, which is defined as follows. Two integers, a and b are said to be congruent modulo m if m divides exactly the difference of a and b; it is common, especially in mathematics tests, to write $a \equiv b \pmod{mi}$ to denote this. Thus, for example, $10 \equiv 7 \pmod 3$, $10 \equiv 4 \pmod 3$, $10 \equiv 1 \pmod 3$ and $10 \equiv -2 \pmod 3$. The number m is a modulus or base, and we shall assume that its values exclude unity, which produces only trivial congruences. For unsigned numbers, that range is $[0,M-1]$. Representations in a system in which the moduli are not pairwise relatively prime will not be unique; two or more numbers will have the same representation.

Advantages :

Most important advantage of residue arithmetic over conventional arithmetic is the absence of carry propagation, in the two main operations of addition and multiplication, and the relatively low precisions required ranging to individual prime or co-prime number of the moduli set, which enables LUT implementations in various operations. In practice, these may make residue arithmetic worthwhile, even though in terms of practical applications such arithmetic has little fields over conventional arithmetic to cover. However the concept of 'break and process' can be very useful in places where integer arithmetic is predominant. However, the fields of Communication and Signal Processing are yet to be explored thoroughly for application on RNS

Basic advantages of residue arithmetic are:

- High Speed
- Low Power
- Superior Fault Tolerance
- Reduction of Computational Load

Limitations and Constraints in Residue Arithmetic On completion of extensive literature survey the following problem statements were de- fined:

- Magnitude Calculation
- Sign Detection
- Overflow Detection and Correction

III. PROPOSED METHODOLOGY

For sign detection a trite technique was used. In this a sign bit was introduced as 32nd bit. This results in a signed bit representation of a number. Operating on floating numbers is a difficult task in residue arithmetic. Hence we convert the floating point number to an integer in a range ±Z. The mapping is done such that the operating range Z < R. If the floating point number ±A is in range, and A < Z, then

$$P_{sn} = Z\ A \tag{3.1}$$

$$X = x\ P_{sn} \tag{3.2}$$

where P_{sn} is precision which decides the incorporation of digits after decimal points.

If an adding circuit is to compute the sum of three or more numbers it can be advantageous to not propagate the carry result. Instead, three input adders are used, generating two results: a sum and a carry. The sum and the carry may be fed into two inputs of the subsequent 3-number adder without having to wait for propagation of a carry signal. After all stages of addition, however, a conventional adder (such as the ripple carry or the look ahead) must be used to combine the final sum and carry results.

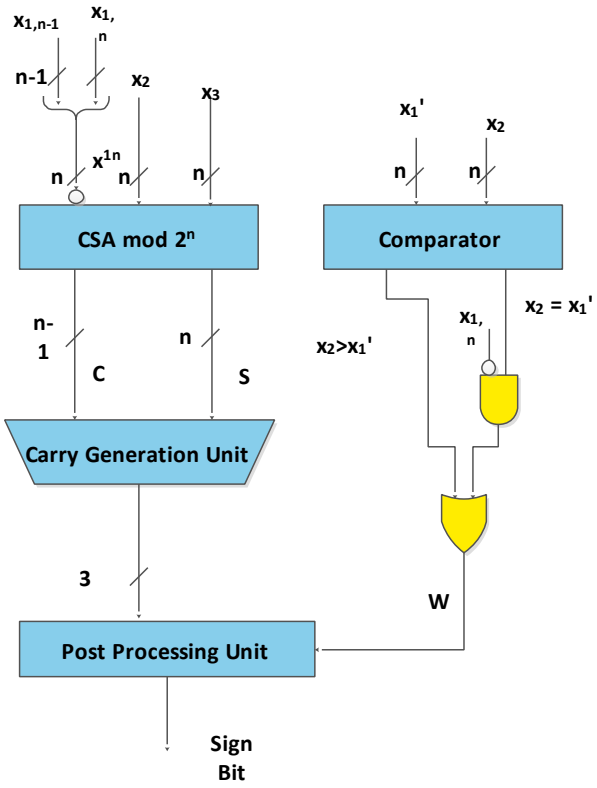

Fig. 3.1 Block Diagram of the Sign Detection Algorithm

IV. SYNTHESIS OUTCOMES

The sign detection algorithm is synthesized on XILINX ISE 13.1, considering the FPGA KINTEX 7 target family board. The proposed architecture is analysed for the delay improvements and found better than the existing work where maximum unit gate delay is about 1ns for 32-Bit design and 0.196ns of 32-Bit proposed architecture.

The comparison is shown in the Table 1 and Table 2 below.

Table 1: Comparison of Delay for Unit Gate Model

Design	Delay
32-Bit Existing Architecture	1 ns
32-Bit Proposed Architecture	0.196 ns

Table 2: Comparison of Overall Delay

Design	Delay
32-Bit Existing Architecture	19 ns
32-Bit Proposed Architecture	15.563ns

The timing summary of the synthesis model is given in the below table with unit gates and overall delay logic delay as well as route delay.

I. CONCLUSION AND FUTURE SCOPE

The sign detection is a integral part of the any digital logic circuit and should be faster in processing parallel with the logic calculations. To speed up sign detection process an algorithm is proposed in this paper which is based on the residue number system. This algorithm is the real factor to improve the detection speed. The outcomes of the proposed model also synthesized and found the improved delay profile than the existing work. The delay improvements are significant than the existing models.

In the future designs for speed improvements in sign detection technologies can be utilized better chip architectures which is designed on smaller nanometer architecture to reduce the delay of algorithm, some improvements in comparator design will also help to reduce the delay.

Table 3: Timing Details of the Proposed Architecture

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 290 / 1
-------------------------------------------------------------------------
Delay:               15.563ns (Levels of Logic = 37)
  Source:            x1<0> (PAD)
  Destination:       sign_bit (PAD)

  Data Path: x1<0> to sign_bit
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             4   0.003   0.574  x1_0_IBUF (x1_0_IBUF)
     LUT5:I0->O            1   0.040   0.349  b3/p1/Y<1>33_SW0 (N01)
     LUT5:I3->O            1   0.040   0.467  b3/p1/Y<1>33_SW1 (N16)
     LUT6:I2->O            1   0.040   0.292  b3/p1/Y<1>33 (b3/p1/Y<1>33)
     LUT6:I5->O            1   0.040   0.467  b3/p1/Y<1>34 (b3/p1/Y<1>34)
     LUT6:I2->O            1   0.040   0.292  b3/p1/Y<1>35 (b3/p1/Y<1>35)
     LUT6:I5->O            1   0.040   0.467  b3/p1/Y<1>36 (b3/p1/Y<1>36)
     LUT6:I2->O            1   0.040   0.292  b3/p1/Y<1>37 (b3/p1/Y<1>37)
     LUT6:I5->O            1   0.040   0.467  b3/p1/Y<1>38 (b3/p1/Y<1>38)
     LUT6:I2->O            1   0.040   0.292  b3/p1/Y<1>39 (b3/p1/Y<1>39)
     LUT6:I5->O            1   0.040   0.467  b3/p1/Y<1>40 (b3/p1/Y<1>40)
     LUT6:I2->O            1   0.040   0.292  b3/p1/Y<1>41 (b3/p1/Y<1>41)
     LUT6:I5->O            1   0.040   0.467  b3/p1/Y<1>42 (b3/p1/Y<1>42)
     LUT6:I2->O            1   0.040   0.292  b3/p1/Y<1>43 (b3/p1/Y<1>43)
     LUT4:I3->O            1   0.040   0.292  b3/p1/Y<1>44_SW0 (N2)
     LUT5:I4->O            1   0.040   0.560  b3/p1/Y<1>44_SW1 (N14)
     LUT6:I1->O            1   0.040   0.434  b3/p1/Y<1>44 (b3/p1/Y<1>44)
     LUT6:I3->O            1   0.040   0.292  b3/p1/Y<1>46 (b3/p1/Y<1>46)
     LUT5:I4->O            1   0.040   0.560  b3/p1/Y<1>47_SW0 (N4)
     LUT6:I1->O            1   0.040   0.560  b3/p1/Y<1>47 (b3/p1/Y<1>47)
     LUT6:I1->O            1   0.040   0.000  b3/p1/Y<1>48_G (N19)
     MUXF7:I1->O           1   0.196   0.292  b3/p1/Y<1>48 (b3/p1/Y<1>48)
     LUT5:I4->O            1   0.040   0.560  b3/p1/Y<1>49_SW0 (N6)
     LUT6:I1->O            1   0.040   0.560  b3/p1/Y<1>49 (b3/p1/Y<1>49)
     LUT6:I1->O            1   0.040   0.000  b3/p1/Y<1>50_G (N21)
     MUXF7:I1->O           1   0.196   0.292  b3/p1/Y<1>50 (b3/p1/Y<1>50)
     LUT5:I4->O            1   0.040   0.560  b3/p1/Y<1>51_SW0 (N8)
     LUT6:I1->O            1   0.040   0.560  b3/p1/Y<1>51 (b3/p1/Y<1>51)
     LUT6:I1->O            1   0.040   0.000  b3/p1/Y<1>52_G (N23)
     MUXF7:I1->O           1   0.196   0.292  b3/p1/Y<1>52 (b3/p1/Y<1>52)
     LUT5:I4->O            1   0.040   0.560  b3/p1/Y<1>53_SW0 (N10)
     LUT6:I1->O            1   0.040   0.349  b3/p1/Y<1>53 (b3/p1/Y<1>53)
     LUT6:I4->O            1   0.040   0.292  b3/p1/Y<1>54 (b3/p1/Y<1>54)
     LUT5:I4->O            1   0.040   0.560  b3/p1/Y<1>55_SW0 (N12)
     LUT6:I1->O            1   0.040   0.349  b3/p1/Y<1>55 (b3/p1/Y<1>)
     LUT5:I3->O            1   0.040   0.279  b3/p1/Mxor_SIGN_BIT_xo<0>1
     OBUF:I->O                 0.002          sign_bit_OBUF (sign_bit)
    ----------------------------------------
    Total                     15.563ns (1.873ns logic, 13.690ns route)
                                       (12.0% logic, 88.0% route)
```

REFERENCES

[1] R. Zimmermann, "Efficient VLSI implementation of modulo (2n □} 1) addition and multiplication," in Proc. 14th IEEE Symp. Comput. Arithmetic, 1999, pp. 158–167.

[2] T. Tomczak, "Fast sign detection for RNS {2n − 1, 2n, 2n + 1}," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 55, no. 6, pp. 1502–1511, Jul. 2008.

[3] P. Mohan, "RNS-to-binary converter for a new three-moduli set {2n+1 − 1, 2n, 2n − 1}," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 54, no. 9, pp. 775–779, Sep. 2007.

[4] K. Furuya, "Design methodologies of comparators based on parallel hardware algorithms," in Proc. 10th ISCIT, Oct. 2010, pp. 591–596.

[5] T. V. Vu, "Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding," IEEE Trans. Comput., vol. 34, no. 7, pp. 646–651, Jul. 1985.

[6] Z. Ulman, "Sign detection and implicit-explicit conversion of numbers in residue arithmetic," IEEE Trans. Comput., vol. 32, no. 6, pp. 590–594, Jun. 1983.

[7] S. Bi and W. Gross, "The mixed-radix Chinese remainder theorem and its applications to residue comparison," IEEE Trans. Comput., vol. 57, no. 12, pp. 1624–1632, Dec. 2008.

[8] N. Szabo, "Sign detection in nonredundant residue systems," IRE Trans. Electron. Comput., vol. EC-11, no. 4, pp. 494–500, Aug. 1962.

[9] S. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," IEEE Trans. Comput., vol. 43, no. 1, pp. 68–77, Jan. 1994.

[10] E. Al-Radadi and P. Siy, "RNS sign detector based on Chinese remainder theorem II (CRT II)," Comput. Math. Appl., vol. 46, nos. 10–11, pp. 1559–1570, 2003.