VLSI implementation of a novel algorithm for binary-negabinary code conversion

Ritu Bajpai
Mona Zaghloul
Department of Electrical and Computer Engineering
The George Washington University
Washington DC, USA
ritu_ece@gwu.edu

S. K. Balasubramanian
Department of Electronics Engineering
Institute of Technology, Banaras Hindu University
Varanasi, India

Abstract—A code converter establishes compatibility between two systems using different codes. Negabinary number system is adopted for unique applications where it provides certain definite advantages over binary number system. In this paper, we have presented VLSI implementation of a novel algorithm that can be used for the conversion of numbers from binary to negabinary number system and vice-versa. This algorithm is faster, consumes less power and more generic than all the algorithms which have been proposed earlier.

I. INTRODUCTION

Negabinary number system has the advantage of unique representability for both positive and negative numbers. Arithmetic circuits have already been proposed for negabinary adders and subtracters [1], [2]. Also circuit designs for negabinary A/D conversion and for complicated arithmetic operations like division and square rooting have been proposed [3], [4]. These days negabinary number system is finding use in digital optical computing [5], [7] and designing low power circuits [6]. The use of negabinary number system leads to an evident need for an efficient binary to negabinary code converter. A few algorithms are already available for code conversion between various number systems [2], [8], [9], [10] but a fast algorithm for conversion between binary and negabinary number system has never before been implemented using VLSI design techniques.

In [2] conversion algorithms between negative and positive bases have been presented but it has also been remarked that if binary number with more than 5 digits is sent into the circuit, the result may be incorrect. The convertor presented in [8] is a serial convertor and hence has a slow speed of operation. Reference [9] presents repetitive division as a general algorithm for radix conversion but no hardware implementation has been suggested. In [10] four algorithms have been proposed for negabinary to binary conversion and converse four algorithms have been proposed for binary to negabinary conversion. Out of these eight algorithms, four algorithms use a binary adder or subtracter as an intermediate stage. This in a way tends to defeat the objective of taking advantage of inherent property of negabinary numbers in performing addition and subtraction. Out of the remaining four algorithms, three involve negabinary subtracter as an intermediate stage. This will involve extra arithmetic before a negabinary adder can be used as a negabinary subtracter. Furthermore, the four algorithms that use negabinary arithmetic in [10] do not apply to negative numbers.

We have developed a common algorithm for the interconversion of binary and negabinary number systems and have implemented it using VLSI design techniques. As compared to earlier algorithms, our algorithm is a parallel algorithm which can be applied for both positive and negative numbers. Also, in our proposed design we use only negabinary arithmetic.

II. NEGABINARY NUMBER SYSTEM

The term nega-binary refers to a number system representing real numbers using integer powers of the base –2 and assigning known fixed weights to different digit positions. An n-digit negabinary number A can be expressed as

\[ A = \sum_{i=0}^{n-1} A_i \cdot (-2)^i, \]

where the coefficients \( A_i \in (0, 1) \). We can see that in the negabinary number system, the sign of alternate digits is opposite and the polarity of any digit at \( i^{th} \) position (counting from least significant digit) depends on whether the value of ‘i’ is odd or even.
III. ALGORITHM FOR CODE CONVERSION FROM BINARY TO NEGABINARY

If we consider positive binary number and its corresponding negabinary number, while bits at the even positions contribute the same in both the number systems, they are the non-zero bits at the odd positions that create a difference. Similarly for the negative numbers, the bits at the odd positions contribute the same in both the number systems whereas the non-zero bits at the even positions create a difference. By the observation of the above mentioned property of the binary and negabinary numbers we came up with the following algorithm for binary to negabinary code conversion:

A. Positive number conversion
1) Start from the LSB.
2) Keep the bits unchanged until a non-zero bit is encountered at an odd position.
3) Once a 1 is encountered at an odd position, complement all the subsequent bits till a 0 is complemented to 1 at an even position.
4) Stop the process if there is no subsequent bit, else starting from the next bit repeat the steps 2 to 4.

B. Negative number conversion
1) Start from the LSB.
2) Keep the bits unchanged until a non-zero bit is encountered at an even position.
3) Once a 1 is encountered at an even position, complement all the subsequent bits till a 0 is complemented to 1 at an odd position.
4) Stop the process if there is no subsequent bit, else starting from the next bit repeat the steps 2 to 4.

IV. ALGORITHM FOR CODE CONVERSION FROM NEGABINARY TO BINARY

The converse algorithm for negabinary to binary code conversion is as follows:

A. Positive number conversion
1) Determine the sign of the negabinary number (if the first occurrence of 1 starting from MSB is at even position then the number is positive else it is negative).
2) Start from LSB.
3) Keep the bits unchanged until a 1 is encountered at an odd position.
4) Once a 1 is encountered at an odd position complement all the subsequent bits till a 1 is complemented to 0 at an even position.
5) Stop the process if there is no subsequent bit, else starting from the next bit repeat the steps 3 to 5.

B. Negative number conversion
1) Determine the sign of the negabinary number.
2) Start from the LSB.
3) Keep the bits unchanged until a 1 is encountered at an even position.
4) Once a 1 is encountered at an even position complement all the subsequent bits till a 1 is complemented to 0 at an odd position.
5) Stop the process if there is no subsequent bit, else starting from the next bit repeat the steps 3 to 5.

On the basis of the observations made from the above algorithms, the following single algorithm can be arrived at for conversion of both binary to negabinary and negabinary to binary numbers.

1) Determine the sign of the number.
2) If the number is negative left shift the number by one bit.
3) If the conversion is from negabinary to binary complement all bits at even positions.
4) Start from the LSB.
5) Keep the bits unchanged until a non-zero bit is encountered at an even position.
6) Once a 1 is encountered at an odd position, complement all the subsequent bits till a 0 is complemented to 1 at an even position.
7) If there is a subsequent bit, then starting from the next bit repeat the steps 5 to 7.
8) If the conversion is from negabinary to binary complement all bits at even positions.
9) If the number is negative then truncate the LSB of the converted number.

V. HARDWARE SCHEME

It can be seen from Sections III and IV that the algorithms developed by us are ripple algorithms which have slow speed of operation. To overcome this drawback we replaced the ripple complementation step of the above algorithms by 2-bit binary to negabinary conversion followed by weight adjustment and negabinary addition. Table-1 shows all possible 2-bit binary numbers and their corresponding values in both decimal and negabinary form.

Consider the signed binary number, \( I = S I_1 I_0 \), where \( S \) (MSB) is the sign bit (\( S = 1 \) corresponds to a negative...
number and $S = 0$ corresponds to a positive number) and $I_1$ and $I_0$ represent the magnitude of the number. The negabinary number corresponding to the binary number of 2-bit magnitude, would be a 4-bit number $X = X_3X_2X_1X_0$. Considering only the positive values for $I$, its corresponding decimal values would vary from 0 to 3.

The values of $X$’s can be reduced in terms of $I_1$ and $I_0$ which can be given by $X_3 = 0$, $X_2 = I_1$, $X_1 = I_1$ and $X_0 = I_0$. Thus the negabinary number corresponding to a binary number $I = SI_1I_0$, with $S = 0$ can be directly represented as $X = 0I_1I_1I_0$. Here we see that splitting a number in pairs of two for acceleration does not involve any extra hardware and hence delay. Therefore 2-bit wise conversion is preferred over more bitwise conversion.

For a positive binary number with $n$ magnitude bits, the conversion is carried out by splitting the number in pairs of two bits each. The corresponding 4-bit negabinary numbers are added by left shifting each subsequent negabinary number by two bits. The block schematic representation of 4 bit positive binary to negabinary conversion is shown in fig. 1(a) and 1(b). Fig. 1(c) shows the conversion steps for an 8-bit number. Similar conversion steps can be used for data longer than 8 bits.

Further, the same hardware which is proposed for the conversion of a binary to negabinary number can be used for the conversion of a negabinary to binary number by complementing either the even or the odd position bits of the negabinary number depending on its sign. Complementing is done before the split operation and the result is re-complemented after the addition step is performed.

It can be verified that the same hardware that is used for the conversion of positive numbers can also be used for the conversion of negative numbers by left shifting the negative numbers by one bit before conversion and then truncating the LSB of the result by right shifting by one bit.

It is necessary to know the sign of the number for performing the conversion. The sign of a binary number is fed in as a single bit. The sign of a negabinary number depends on whether the MSB is at even or at odd position.

VI. COMPARISON WITH EXISTING ALGORITHMS

The proposed algorithm was simulated using cadence SpectreS. As the algorithm proposed in [2] works only for maximum 5 bit data length and the algorithm proposed in [8] is a serial algorithm, they were not selected for a comparison with the proposed algorithm. Algorithm for negabinary to binary conversion from [10] was simulated and compared with the proposed algorithm for negabinary to binary conversion. Same negabinary adder module was used for both the algorithms. The simulation results are listed in table 2. The proposed algorithm gives better speed and power results.

<table>
<thead>
<tr>
<th>Table 1: Binary and Negabinary Representation of Decimal Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Comparison of Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negabinary Input</td>
</tr>
<tr>
<td>------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>1100011</td>
</tr>
<tr>
<td>1110010111111</td>
</tr>
<tr>
<td>1010101101</td>
</tr>
<tr>
<td>1001101</td>
</tr>
</tbody>
</table>

Figure 1 (a) Obtaining operands to be fed to negabinary adder for conversion of a positive 4 bit binary number to negabinary. (b) Performing the negabinary addition on the obtained operands. (c) Conversion steps for an 8 bit positive binary number to negabinary number.
VII. IMPLEMENTATION AND TESTING

The proposed design was simulated and laid out in cadence using AMI 0.5u technology. The area of the proposed design is 229620 \( \text{um}^2 \). A negabinary carry-look-ahead adder proposed in [1] was used to implement the adder needed for the conversion algorithm. D-flip flops were added at the input and the output to synchronize the input and the output data. Fig. 2 shows the complete layout of the 8 bit radix converter and fig. 3 shows the picture of the chip taken under the microscope. The fabricated chip was tested at 50 MHz clock speed with a high level test station. Some of the test results are shown in fig. 4 and the test values are listed in table 3. As mentioned earlier, due to the D flip- flops at the input and the output, the output is seen at the second rising clock edge after the input is entered. Also radix 0 implies binary number and radix 1 implies a negabinary number. Signin is the sign of the input number, 0 for positive and 1 for negative. For negabinary number signin is always 0 as the number itself contains the sign information. Signout is the sign of the output number but has complemented notation compared to signin. Signout is 1 for positive number and 0 for a negative number.

VIII. Conclusion

We have proposed a novel algorithm for inter-conversion of binary and negabinary number system. The proposed algorithm has also been implemented using the VLSI design techniques. The fabricated chip has been tested and the results have been verified. The algorithm presented in this paper is faster, consumes less power than the existing algorithms and has been presented with complete layout scheme for the first time.

| TABLE-3: TEST RESULTS | | | |
|---|---|---|
| Input | Output | Decimal value |
| (11111111 \(_b\) | (0100000011 \(_2\) | 255 |
| (11111111 \(_b\) | (-0001010101 \(_2\) | -95 |
| (01010101 \(_b\) | (0001010101 \(_2\) | 95 |
| (-01010101 \(_b\) | (0011111111 \(_2\) | -95 |

REFERENCES