VLSI efficient binary-to-residue converter for DSP applications

T. Srikanthan and D. Radhakrishnan
Nanyang Technological University
Nanyang Avenue, Singapore 639798
Telephone: (65)799-5044 Fax: (65)792-6559
e-mail: astsrikan@ntu.edu.sg

An area efficient binary-to-residue converter suitable for pipeline applications is proposed in this paper. This architecture uses multiple access on a ROM to generate the residues of a given number. It has been found that more than 25% reduction in the ROM area and a saving of one adder delay can be achieved at the expense of an extra ROM access to generate the residues of a 32 bit binary number.

1. Introduction

Residue number systems (RNS) are becoming very popular in many computation intensive applications, such as digital signal processing [1]. An RNS is defined by a set of relatively prime integers \( m_1, m_2, \ldots, m_s \) called the moduli of the number system. Such a system provides unique representation of numbers from 0 to \( M-1 \) where \( M=\prod_{i=1}^{s} m_i \). Each integer \( X \) is represented by an \( r \) tuple \( (x_1, x_2, \ldots, x_r) \), where each residue \( x_i = X \mod m_i \), defined as the least remainder when \( X \) is divided by the moduli \( m_i \).

In RNS, arithmetic operations on large integers are done by converting them into smaller residues and performing the operations independently and all in parallel, thereby speeding up the whole operation. In present systems an analog signal is first converted into binary and then a binary-to-residue converter is used to generate the residues. The residue results produced at the end of processing are finally reconverted back to binary. These conversions from binary-to-residue (forward conversion) and from residue-to-binary (reverse conversion) are complex operations for a general moduli set.

In this paper we present a novel technique by which the residue generation can be simplified by using multiple access technique applied on a small ROM.

2. Binary-to-residue conversion

A number of binary-to-residue conversion schemes are available in the literature [2-6]. A good majority of them makes use of ROM tables. In [3] a VLSI computing architecture is proposed for forward and reverse conversion of an integer \( N \), represented in the weighted number system, to and from a residue code based on moduli \( m_i \), \( 1 \leq i \leq s \). The residue corresponding to the \( j \)th bit of an \( n \) bit binary number with respect to modulus \( m_i \) is generated and serially stored in a register of size \( \lceil \log_2 m_i \rceil \). Two such registers storing residues of adjacent bits are combined to form a processing element (PE). A total of \( n/2 \) PEs are used to generate the residues corresponding to each and every bit in the \( n \) bit binary word. For a given binary number, depending on the value of bit \( b_j \) either the register contents or zeros will be output. The two outputs from each PE are added in a modulo \( m_i \) adder, thereby using \( n/2 \) modulo adders in the first level. Afterwards the outputs from these modulo adders are again added using additional levels of modulo adders, while cutting down the number of modulo adders at each level by a factor of 2. A modification to the above scheme was proposed in [4], which is more suitable for pipeline applications. Here the \( n \) bit binary word is partitioned into \( n/q \) groups of \( q \) adjacent bits. The modulo \( m_i \) values corresponding to each group of \( q \) bits are stored in a ROM. Then the \( n/q \) values read from these ROMs are added by means of \( n/q \) processing elements, to generate the residue.

For special moduli sets of the form \( \{2^{n-1}, 2^n, 2^n+1\} \), simple conversion schemes are available using only adder elements [5,6].

In this paper we propose an elegant architecture for forward conversion which is especially suitable for pipeline applications and can be used with any moduli set. Compared to earlier approaches, our method uses reduced ROM storage and a fewer number of modulo adders.
3. Residue generation by multiple access technique

In multiple access technique, as the name implies, the values stored in a single ROM are successively accessed a number of times to generate the residues. As in earlier approaches, the n bit binary word is partitioned into n/q fields of q bits each. This results in a number with a higher radix, which is equal to the qth power of 2. Unlike in earlier approaches the modulo values corresponding to each field are generated independently using the same ROM. They are then added together using modulo adders to generate the residue of the input integer. This results in considerable reduction in ROM size and can easily be incorporated in any processor chip.

For convenience let us assume that the length n of the binary word is an integer multiple of the field width q. Hence n = pq, where p is an integer. This is shown in Figure 1. The q bit fields are numbered from 0 to p-1 starting from LSB.

The decimal equivalent of the number X stored in the n bit binary word is given by:

\[ X = \sum_{i=0}^{n-1} b_i 2^i = \sum_{i=0}^{p-1} x_i r^i \]

where the radix r = 2^q, and \( x_i \) denotes the decimal value corresponding to the \( i^{th} \) \( q \) bit field. A residue table is generated corresponding to the \( q \) bit field 1. This table will have \( 2^q \) entries corresponding to the integers \( x_i \), for \( 0 \leq x_i \leq 2^q - 1 \). It is programmed with the value \( \lfloor i \cdot r \rfloor_m \) as the entry at index \( I \) for any modulus \( m \). The size of this table would be \( 2^q \times \lceil \log_2 m \rceil \).

**Theorem 1**

The residue \( \lfloor x_i r^i \rfloor_m \) corresponding to the \( i^{th} \) field equals the value accessed at the \( i^{th} \) successive access of the residue table for field 1, using the previously accessed value from the same table as the new index to this table, starting with an initial index of \( x_i \).

**Proof:** The residue \( \lfloor x_i r^i \rfloor_m \) can be expressed as: \( \lfloor x_i r^i \rfloor_m = \lfloor x_i r \rfloor_m \cdot r^{i-1} \rfloor_m \). But \( \lfloor x_i r \rfloor_m \) equals the value stored at index \( x_i \) of the residue table for field 1. Let \( \lfloor x_i r \rfloor_m = y_j \). Hence \( \lfloor x_i r^i \rfloor_m = \lfloor y_j r^{i-1} \rfloor_m = \lfloor y_j r \rfloor_m \cdot r^{i-2} \rfloor_m \). Now \( \lfloor y_j r \rfloor_m \) corresponds to the value stored at index \( y_j \) of the residue table for field 1. It follows by induction that by accessing the residue table for field 1 successively for \( i \) times using the newly accessed values as the index to the table, produces the residue \( \lfloor x_i r^i \rfloor_m \). QED

From Theorem 1 it follows that the residues corresponding to each and every field from 1 to \( p-1 \) of the n bit binary word can be generated using a single residue table for field 1. When multiple access technique is used in this manner, the number of accesses to the residue table depends on the total number of fields to which the n bit word is partitioned. For a \( p \) field partition, the number of table accesses is \( p(p+1)/2 \) using a single table for field 1. The access time can be reduced by adding more residue tables. The reduction in time is in direct proportion to the square of the number of tables added.

A comparison to illustrate the effect of multiple access on ROM table reduction and the delay in residue generation is presented in Table 1. This table shows the total ROM size used and the delay in residue generation for a 32-bit binary word, divided into 8 fields each of 4-bits width. All moduli used are assumed to be 5-bits (17, 19, 23, 27, 29, 31, 32). The residue generation for field 0 does not require any ROM. The first row in the table corresponds to the case where no multiple access is used. Hence this needs individual ROMs for each of the 7 fields. The third row in Table 1 (shown in italics in the table) represents a reasonable compromise between hardware and speed. In this case 5 individual ROMs are used for 5 fields. The other two fields generate their residues by multiple access technique. It is interesting to note that for this case the total memory size has been reduced by more than 25% and the number of adder delays reduced from 3 to 2 with a nominal increase of one extra ROM access.

In the case of 5-ROM architecture (row3 in Table1) the steps for residue generation are as follows. In the first cycle, field 7 accesses the residue table of field 4, field 6 accesses the residue table of field 5 and fields 1,2,3 access their respective residue tables. The residues corresponding to fields 0,1,2, and 3 are
added by using two modulo adders. The residue generation for fields 6 and 7 needs one more ROM access. During this time residues for fields 4 and 5 are also generated. These four residues are circulated through the same two modulo adders while the previous outputs of these adders are added by another modulo adder. The final residue will be generated in two subsequent clock cycles. The conversion delay in this architecture can be encountered by using a pipeline architecture.

Table 1. Hardware complexity and delay due to multiple access for a 32 bit word

<table>
<thead>
<tr>
<th>ROM size</th>
<th>Total time for residue generation</th>
<th>No. of modulo adders used</th>
</tr>
</thead>
<tbody>
<tr>
<td>672 X 5</td>
<td>1 ROM access + 3 5-bit modulo addition</td>
<td>42</td>
</tr>
<tr>
<td>576 X 5</td>
<td>2 ROM access + 2 5-bit modulo addition</td>
<td>36</td>
</tr>
<tr>
<td>480 X 5</td>
<td>2 ROM access + 2 5-bit modulo addition</td>
<td>24</td>
</tr>
<tr>
<td>384 X 5</td>
<td>3 ROM access + 2 5-bit modulo addition</td>
<td>18</td>
</tr>
<tr>
<td>288 X 5</td>
<td>4 ROM access + 2 5-bit modulo addition</td>
<td>18</td>
</tr>
</tbody>
</table>

4. Conclusions

The main objective of this research was to find an area-time efficient architecture for the generation of residues from a binary number. It is shown that the ROM storage used for forward conversion can be reduced by using multiple access techniques, but this in turn increases the conversion delay. Different choices of ROM storage Vs conversion delay indicate that a reasonable compromise is obtained by using 300 bytes of storage and 24 5-bit modulo adders for a 32-bit binary-to-residue converter. The conversion speed can be increased by using pipeline architecture.

ACKNOWLEDGEMENTS

The authors wish to thank Mr. Bimal Gisuthan for his help during the preparation of this paper.

REFERENCES