A VLSI RESIDUE ARITHMETIC MULTIPLIER WITH FAULT DETECTION CAPABILITY

V. Bobin and D. Radhakrishnan

Department of Electrical Engineering
University of Idaho
Moscow, ID 83843

ABSTRACT

A programmable Residue Arithmetic based multiplier is presented in this paper. The modulo-m multipliers are implemented using a set of identical multiplexer modules designed in pass logic. This makes the design simple, easily expandable, and ideally suited for VLSI implementation. In addition, the multiplier presented has single residue fault detection capability.has single residue fault detection capability.

1 Introduction

The Residue Number System (RNS) has been receiving a lot of attention lately, mainly because digital computational systems structured into residue arithmetic units promise speed, modularity, and high performance. The RNS performs the arithmetic operations of addition, subtraction, and multiplication as a set of concurrent operations. RNS-based systems are ideally suited for VLSI implementation because of the large number of identical modules and the absence of global interconnects in their layout.

The use of RNS in a wide variety of signal processing applications has been shown to provide a substantial reduction in both circuit complexity and design effort [4]. The inherent independence among the residue digits prevents the propagation of an error in one residue to the other during subsequent operations. This provides a basis for fault tolerance, which is built into the basic algebraic structure, and can be incorporated into the hardware architecture. In addition, the non-weighted code structure permits the addition of redundant residue digits to provide fault detection and correction.

2 The Residue Number System

The residue number system is of particular interest in digital systems because of the parallel nature of its arithmetic [1,2]. The theory of the residue number systems is based on a fundamental theorem in arithmetic called the Chinese Remainder Theorem.

In RNS, an integer X is uniquely represented by an n-tuple of integers $(x_1, x_2, \ldots, x_n)$, called the residue representation of X. The integers x_i, $i = 1, 2, \ldots, n$ are called the residues and are obtained as the remainders when the number X is divided by a set of distinct, and relatively prime integers, m_i, $i = 1, 2, \ldots, n$, called the moduli of the residue number system. Thus $x_i = X \ mod \ m_i$, denoted by $|X|_{m_i}$, where $0 \leq x_i < m_i$.

It follows from the Chinese Remainder Theorem that for any given n-tuple $(x_1, x_2, \ldots, x_i, \ldots, x_n)$ satisfying the above relationships there exists one and only one integer X such that $0 \leq X < \prod_{i=1}^{n} m_i$. Hence, for a given set of moduli $\{m_i\}$, $i = 1, 2, \ldots, n$, the number of integers that can be represented uniquely is M, where $M = \prod_{i=1}^{n} m_i$. The number X can be evaluated from the n tuple $(x_1, \ldots, x_n)$ using the equation: $X = (\sum_{i=1}^{n} \frac{M}{m_i} \alpha_i x_i) \ mod \ M$, where $\frac{M}{m_i} \alpha_i \equiv 1 \ mod \ m_i$.

Negative numbers can be represented by partitioning the set of all uniquely represented integers into two sets, negative and positive. A common assignment is that all numbers X; $0 \leq X \leq (M/2) - 1$, are considered positive and the rest are considered negative.

A Redundant Residue Number System (RRNS) is simply a Residue Number System with r redundant moduli [11]. RRNS has $n+r$ relatively prime moduli $(m_1, m_2, \ldots, m_n, m_{n+1}, \ldots, m_{n+r})$. $(m_1, \ldots, m_n)$ are the non-redundant moduli, and $(m_{n+1}, \ldots, m_{n+r})$ are the redundant moduli. Hence a number is represented by $n+r$ residue digits $(x_1, x_2, \ldots, x_n, \ldots, x_{n+r})$, where $(x_1, \ldots, x_n)$ are the non-redundant residue digits and $(x_{n+1}, \ldots, x_{n+r})$ are the redundant residue digits. The total range $[0, M_T)$ is divided into $[0, M)$ and $[M, M_T)$, where $M_T = \prod_{i=1}^{n+r} m_i$ and $M = \prod_{i=1}^{n} m_i$. They are referred to as legitimate and illegitimate ranges respectively.

Each RNS is associated with a mixed radix number system that is useful during output conversion [2]. The mixed radix representation consists of n mixed radix digits, denoted by $(a_1, a_2, \ldots, a_n)$. For each number X, the mixed radix representation is given by: $X = \sum_{k=1}^{n} a_k \prod_{i=1}^{k-1} m_i$, where $0 \leq a_i < m_i$ for all $i = 1, 2, \ldots, n$. Here, $\prod_{i=1}^{0} m_i$ is defined to be 1.

2.1 Residue Arithmetic

Since their introduction, the residue number systems (RNS) have been considered a promising way to provide very fast arithmetic because of their modular nature. Addition of two numbers in the RNS is performed by adding the residues modulo m_i of the two numbers. That is, $X+Y = (|X|_{m_1}, |X|_{m_2}, \ldots, |X|_{m_n}) + (|Y|_{m_1}, |Y|_{m_2}, \ldots, |Y|_{m_n})$
$= (|Z|_{m_1}, |Z|_{m_2}, \ldots, |Z|_{m_n}) = Z$, where $|Z|_{m_i} = ||X|_{m_i} + |Y|_{m_i}|_{m_i}$. If $X + Y > M - 1$, then $(|Z|_{m_1}, |Z|_{m_2}, \ldots, |Z|_{m_n}) = |X + Y|_M$. This produces an overflow and hence the sum modulo M is obtained.

In a similar manner, multiplication is done by taking the product of the corresponding residues modulo m_i. Subtraction of Y from X can be performed by adding the negative of Y to X. In the following discussion, only positive integers in the range $[0, M)$, where $M = \prod_{i=1}^{n} m_i$, will be assumed. However, the same results can be obtained for the range $[(-M/2), (M/2-1)]$, ie., if negative integers are also considered.

Division is a complex operation in the RNS [3]. However, if Y is divisible by X and X is relatively prime to M, the residue representation of $Y/X = Z$ can be found easily.

2.2 Overflow Detection using RRNS

Determination of the magnitude of a result is not as easy in RNS as it is in positional number systems. Hence overflows are relatively difficult to detect. However, in RRNS overflows can be detected by checking whether the result is legitimate or illegitimate [5]. This in turn can be determined easily by a simple mixed radix conversion. The $n + r$ mixed radix digits corresponding to the RRNS are given by: $(a_1, a_2, \ldots, a_n, a_{n+1}, \ldots, a_{n+r})$, where $(a_1, a_2, \ldots, a_n)$ are the non-redundant mixed radix digits, and $(a_{n+1}, a_{n+2}, \ldots, a_{n+r})$ are the redundant mixed radix digits.

Etzel and Jenkins have shown that for a RRNS where each redundant modulus is larger than each non-redundant modu-

lus, an integer X lies in the legitimate range if and only if the redundant mixed radix digits are all zero [5]. That is, if $m_{n+j} > m_i$ for all $j = 1,...,r$ and $i = 1,...,n$, then, $a_{n+j} = 0$ for all j. Hence overflow can be detected by simply checking for non-zero redundant mixed radix digits.

2.3 Error Detection and Correction using RRNS

For a RRNS to have error detection and correction capability, the operands and results must be constrained to lie within the legitimate range. Barsi and Maestrini have described a scheme for RNS error detection by determining whether a number $(x_1,...,x_{n+r})$ is legitimate or illegitimate [7]. Assume a RRNS with $r \geq 1$ and each redundant modulus larger than each non-redundant modulus. A single residue error occurs when a legitimate number $X = (x_1, x_2, ..., x_i, ..., x_{n+r})$ changes to a different number $\overline{X} = (x_1, x_2, ..., \overline{x}_i, ..., x_{n+r})$, due to an error in the i^{th} residue digit, where $i = 1,...,n+r$. This number $\overline{X}$ will be an illegitimate number. Thus a single error can be detected in exactly the same manner as an overflow, assuming that an overflow does not occur simultaneously. A mixed radix conversion is performed to determine if the result is in the legitimate or illegitimate range. If the result is in the illegitimate range, the residue that is in error can be identified and corrected provided, the RRNS has atleast two redundant moduli, each larger than the non-redundant moduli. In order to do this, the m_i-projections of the result have to be found [7].

The m_i-projection of X, denoted by X_i, is defined as: $X_i = (x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_{n+r})$, ie., X with the i^{th} residue deleted. In a RRNS with $r = 2$ and the moduli satisfying the earlier condition, if an illegitimate number $\overline{X}$ is found such that for some $i, i = 1,...,n+r$, $\overline{X}_i$ is legitimate, then the other projections $\overline{X}_j$, for $j = 1,...,n+r$ and $j \neq i$, are all illegitimate [7]. Thus in order to isolate the faulty residue digit it is sufficient to find the $\overline{X}_i$ that is legitimate. It is immediately apparent that the legitimate $\overline{X}_i$ is the correct value X and that the error occurred in the i^{th} digit. Now the correct value of the residue digit x_i is the legitimate $\overline{X}_i$ mod m_i. Thus a single error correction is possible with 2 redundant moduli. The following result from [6] is a generalization of the above observation: a RRNS with r redundant moduli will detect r errors, and correct $\lfloor r/2 \rfloor$ errors, where $\lfloor x \rfloor$ is the largest integer such that $\lfloor x \rfloor \leq x$.

3 RNS-based Multiplier

A modular design of a residue arithmetic multiplier suitable for use in a large RNS-based computational system is presented in this paper. This multiplier consists of several modulo-m multipliers operating in parallel. The design uses identical multiplexer modules and is very general in the sense that the same basic structure can be used to implement a RNS-based adder, subtractor, or any such "two input" computational structure with similar input range and output range. The same structure is also used to build an error detector for the whole RNS-based system.

The selection of proper moduli is vitally important for the efficiency of the design. The moduli chosen in this design are 17, 19, 23, 25, and 29. The first four are the non-redundant moduli and the last viz., 29, is a redundant modulus. All five moduli are represented by 5 bits. Consequently, the cell sizes are all identical, an important attribute to minimize the design effort. At the same time, these moduli give the system a reasonably large legitimate range. The legitimate range in this case is $[0, 185725)$ which is approximately 17.5 bits, and the total range is $[0, 5386025)$.

3.1 Multiplier Design

Each modulo-m multiplier is designed using multiplexer (MUX) modules as the basic building blocks. It consists of m multiplexers arranged in two levels. The first level has $(m-1)$ multiplexers and the second level has one. Each multiplexer has n selector inputs and m inputs, each n bits wide where $n = \lceil log_2 m \rceil$. The schematic diagram of a modulo-m multiplier is shown in Fig. 1. The multiplicand (X) and the multiplier (Y) are applied to the selector inputs of the first and the second level multiplexers respectively. Depending on the binary values applied to the selector inputs, a particular data input is enabled to the output of the module. As shown in Fig. 1, the input D_1 of the second level multiplexer is connected to $|0|_m = 0$, since if $Y = 00...0$, $|XY|_m$ is 0, whatever the value of X. When $Y = 00..01$, $|X|_m$, has to be passed to the output of the multiplier. Hence the output of MUX 1 is connected to D_2 input of the second level multiplexer. The data inputs of MUX 1 are programmed such that $|X|_m$ is enabled to its output. This means that the input to $D_1 = |0|_m$, $D_2 = |1|_m$, and so on, and $D_m = |m-1|_m$. Similarly, MUX 2 will be programmed with the values corresponding to $|2X|_m$. This procedure is followed so that MUX m-1 would be programmed with the values corresponding to $|(m-1)X|_m$.

3.2 Multiplexer Design

The multiplexer module is implemented in pass logic [8]. It consists of a decoding logic which decodes the selector inputs, which then enables a set of nMOS pass transistors to pass the respective data to the output of the module. The structure of the multiplexer for a 5 bit modulus is shown in Fig. 2. The inputs B_{ij}, $i = 1,2,...,m$ and $j = 1,2,...,5$ have to be programmed depending on the modulo values used, and the position of the module in the multiplier structure.

For the first level MUX the decoding logic is the same for all the modules and hence the same decoder is used for all of them. The second level MUXs however require a separate decoder for each modulo-m channel.

The decoder is designed as an nMOS pass network with the final output buffered by an inverter. The buffer helps to keep the signal propagation delays down, eliminates signal deterioration, and provides enough drive capability. The function table of the decoder for a modulo-25 multiplier is shown in Table 1. This table shows which output line of the decoder is high for each selector input combination. In this case, both the selector inputs and the data inputs are all 5 bits wide, since $\lceil log_2 25 \rceil = 5$.

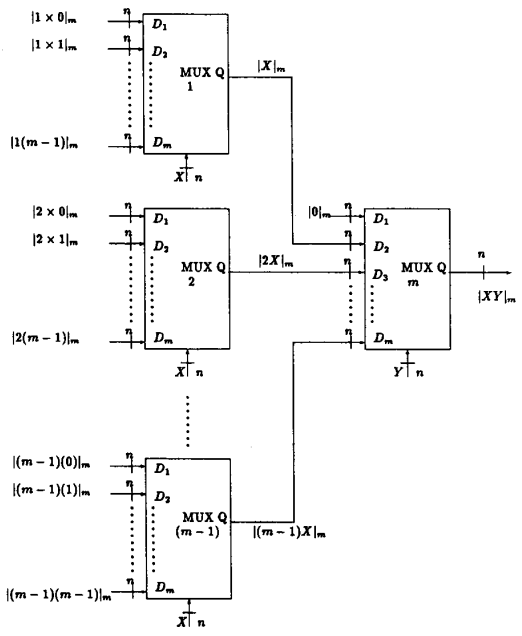

Figure 1: A Modulo-m Multiplier

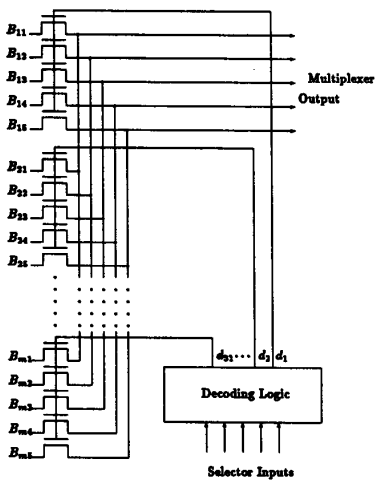

Figure 2: A Multiplexer for a 5 bit Modulus

a_1	a_2	a_3	a_4	a_5	Active
0	0	0	0	0	d_1
0	0	0	0	1	d_2
0	0	0	1	0	d_3
0	0	0	1	1	d_4
0	0	1	0	0	d_5
0	0	1	0	1	d_6
0	0	1	1	0	d_7
0	0	1	1	1	d_8
0	1	0	0	0	d_9
0	1	0	0	1	d_{10}
0	1	0	1	0	d_{11}
0	1	0	1	1	d_{12}
0	1	1	0	0	d_{13}
0	1	1	0	1	d_{14}
0	1	1	1	0	d_{15}
0	1	1	1	1	d_{16}
1	0	0	0	0	d_{17}
1	0	0	0	1	d_{18}
1	0	0	1	0	d_{19}
1	0	0	1	1	d_{20}
1	0	1	0	0	d_{21}
1	0	1	0	1	d_{22}
1	0	1	1	0	d_{23}
1	0	1	1	1	d_{24}
1	1	0	0	0	d_{25}

Table 1: Function Table for Modulo-25 Decoder

From the pass functions for $\overline{d_1}$ through $\overline{d_{25}}$, it is noticed that pairs of pass functions such as $d_1d_2, d_3d_4, ..., d_{23}d_{24}$ can be implemented as a single unit with substantial reduction in the number of pass transistors. This is illustrated in Fig. 3 for the pair d_1d_2.

3.3 Simulation and Layout

Both logic simulation and circuit simulation of the MUX modules were done using SHIVA and HPSPICE respectively. The parasitic capacitances were estimated from a layout of the modulo m multiplier which is shown in Fig. 4. The simulation using HPSPICE indicated a worst case delay of 160 nano seconds through the modulo-m multiplier.

The layout was done using MOSIS 1.2 micron design rules. It measures $1000\mu \times 720\mu$. The structure laid out is the most general one so that it can be adapted for any modulo-m computational structure with two 5 bit wide inputs. All modulo-m multipliers are implemented as identical structures [10].

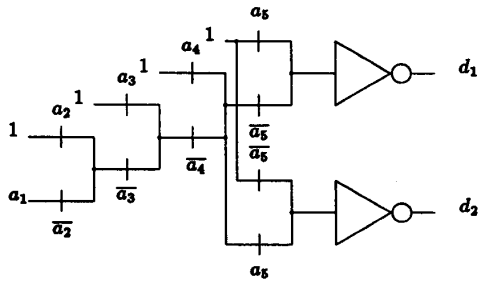

Figure 3: Decoding Logic for d_1 and d_2 of Fig. 2

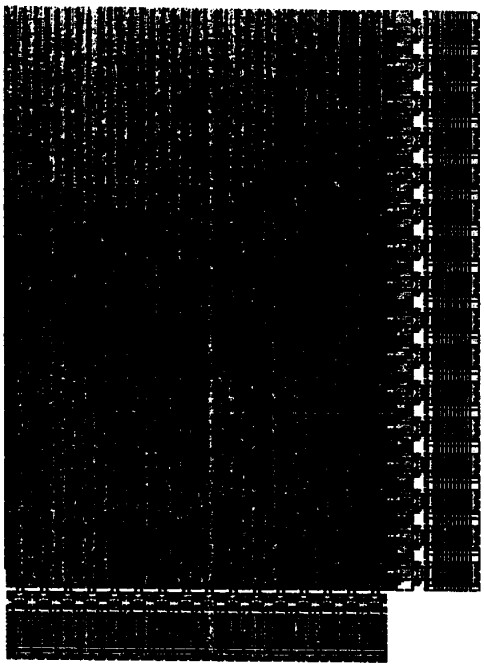

Figure 4: Layout of Modulo-m Multiplier

4 Fault Detector

The two most important properties attributable to fault tolerance are carry-free arithmetic and the lack of ordered significance among residues. The former makes the errors remain confined to their original residue positions, while the latter implies that any erroneous residue can be discarded without affecting the result, provided the RRNS leaves enough range in the reduced system to contain the result. These properties of RRNS are used here to build hardware fault detectors.

4.1 Array Architecture

In section 2 it was pointed out that in order to detect a single residue error, the RNS must have at least one redundant modulus that is larger than each non-redundant modulus. Then it is sufficient to check if the result falls in the legitimate range or not. In the remainder of this paper it is assumed that only one residue is in error.

The design of a modular error checker for a self-checking residue number arithmetic is given in [9]. This is a simple mixed radix converter and can be characterized as the upper left triangular portion of a square array. This architecture is based on a recursive procedure given below. Each cell in the array is characterised by the following recursive equation: $s_{p+1,k} = |(s_{p,k} - a_p)m_p^{-1}|_{m_k}$ for $p = 1, \ldots, n+r-1$, and $k = p+1, \ldots, n+r$, where $s_{1,k} = x_k$, $k = 1, \ldots, n+r$ and $a_j = s_{j,j}$, $j = 1, \ldots, n+r$. Here m_p^{-1} is the multiplicative inverse of m_p and is defined by the equation $|m_p \times |\frac{1}{m_p}|_{m_k}|_{m_k} = 1$.

The complete set of equations obtained from the recursion above are as follows:

$a_1 = x_1$

$a_2 = ((x_2 - a_1)m_1^{-1}) \ mod \ m_2$

$a_3 = (((x_3 - a_1)m_1^{-1} - a_2)m_2^{-1}) \ mod \ m_3$

$a_4 = ((((x_4 - a_1)m_1^{-1} - a_2)m_2^{-1} - a_3)m_3^{-1}) \ mod \ m_4$

$a_5 = (((((x_5 - a_1)m_1^{-1} - a_2)m_2^{-1} - a_3)m_3^{-1} - a_4)m_4^{-1}) \ mod \ m_5$

The corresponding array realization is shown in Fig. 5. The basic operation performed in each cell of the array can be represented by: $((u - v)m_i^{-1}) \ mod \ m_j$.

Each cell is again designed using two levels of multiplexers. The data inputs to the first level multiplexers are programmed so that the proper $((u-v)m_i^{-1}) \ mod \ m_j$ is passed to the output. The selector inputs to the first level multiplexers are all fed with u and the selector input to the second level multiplexer is fed with v. This is identical to the modulo-m multiplier units described earlier. Thus the whole system (both multiplier and fault detector) can be implemented by the repetition of a single module and hence is best suited for VLSI implementation.

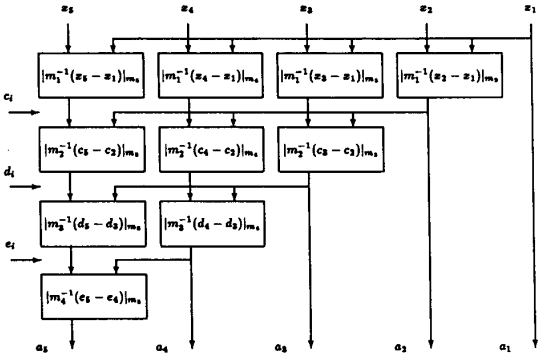

Figure 5: A Self Checking Mixed-Radix Converter Realization

5 Conclusions

The RNS-based multiplier presented in this paper consists of 5 modulo-m multipliers and an on-line fault detector all designed in nMOS pass logic. The complete system can be impelemented by the repetition of a single multiplexer module, thereby reducing the design effort to a minimum. The silicon area of a single modulo-m multiplier, estimated from its layout is $7.2 \times 10^5 \mu m^2$. The area of of the whole multiplier with an added self-checking fault detector will be about 15 times this area. Added to this is the area of the conversion circuitry. Even though all the hardware can be packed into a VLSI chip, the conversion overhead for binary-to-residue and residue-to-binary seems to be wasteful for a multiplier. This holds for the fault detector too, despite its self-checking property. However, these disadvantages vanish if the modulo-m multiplier is part of a larger RNS-based system such as a digital signal processor. The multiplier modules presented can be used to implement any "two input" computational structure where the inputs are 5 bits wide.

It is noticed that the hardware complexity increases exponentially with the number of bits in the moduli. This is a disadvantage when larger moduli are required. The 5-moduli RNS considered in this paper gives a reasonably large dynamic range without sacrificing simplicity of hardware.

References

[1] H. L. Garner, "The Residue Number System," IRE Trans. Electronic Computers, Vol. EL-8, No.6, June 1959, pp 140-147.

[2] N. S. Szabo and R.I. Tanaka, "Residue Arithmetic and its Applications to Computer Technology," NY: McGraw-Hill, 1967.

[3] Y. A. Keir, P. W. Cheney, and M. Tannenbaum, " Division and Overflow Detection in Residue Number Systems," IRE Trans. Electron. Comput., vol EC-11, pp 501-507, August 1962.

[4] R. D. Haggarty, B. L. Johnson, and E. A. Palo, "A New Design Method for VLSI Signal Processors," IEEE, pp 46.3.1-46.3.8, 1986.

[5] M. H. Etzel and W. K. Jenkins, " Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," IEEE Transactions on Acoustics, Speech, and Signal Processing, pp 538-545, October 1980.

[6] D. Mandelbaum, "Error Correction in Residue Arithmetic," IEEE Transactions on Computers, pp 538-545, June 1972.

[7] F. Barsi and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems," IEEE Transactions on Computers, pp 307-315, March 1973.

[8] D. Radhakrishnan, S. R. Whitaker, and G. K. Maki, "Formal Design Procedures for Pass Transistor Switching Circuits," JSSC, Vol. SC-20, No. 2, pp 531-536, April 1985.

[9] W. K. Jenkins, "The Design of Error Checkers for Self-Checking Residue Number Arithmetic," IEEE Transaction on Computers, April 1983.

[10] V. Bobin, "A VLSI Residue Arithmetic Multiplier With Fault Detection," M.S. thesis, University of Idaho, Moscow, Idaho, April 1989.

[11] R. W. Watson,"Error detection and correction and other residue interacting operations in a residue redundant number system," Ph.D. dissertation, Univ. of California, Berkeley, 1965.