Combined System-Level Redundancy and Modular Arithmetic for Fault Tolerant Digital Signal Processing

W. K. Jenkins, B. A. Schnaufer, and A. J. Mansen
Department of Electrical and Computer Engineering and
The Coordinated Science Laboratory
1308 West Main St.
Urbana, Illinois 61801
U.S.A.

ABSTRACT

This paper proposes combining system-level modular redundancy with the arithmetic modularity of residue number system (RNS) arithmetic to achieve fault tolerance in high speed digital signal processing (DSP) systems. Double, triple, and quadruple modular redundancy, system-level concepts which are frequently used in commercial fault tolerant computers, are combined with RNS modularity for realizing important DSP computational kernels. The discussion includes the development of the serial-by-modulus (SBM) RNS architecture in which residue digits are processed sequentially in circuits that handle only one modular operation at a given time, thereby sacrificing speed for circuit simplicity. As a potential application of the SBM concept, a variable word length sum-of-products signal processing kernel is developed based on a serial-by-modulus RNS architecture. Due to the fact that the RNS is not a weighted number representation, if the instantaneous dynamic range requirement can be estimated it may be possible to perform the computation with only enough residue digits to provide the necessary dynamic range. The variable word length concept is demonstrated through computer simulation.

1. INTRODUCTION

It has been shown in earlier work that RNS designs can be effective for realizing high speed sum-of-products DSP kernels, such as finite convolution/correlation that is characteristic of FIR digital filters [1]. The modular structure of RNS arithmetic induces a natural modularity in the hardware, resulting in a system architecture that can be partitioned conveniently to achieve good parallelism and testability in VLSI integrated circuits. To achieve fault tolerance, redundancy can be incorporated into the RNS arithmetic to facilitate error detection, location, and correction [2, 3]. In recent years RNS arithmetic has been used more for DSP applications that are dominated by repetitive multiply-accumulate cycles than for general-purpose computing where many types of diverse operations tend to be equally important.

2. ERROR CONTROL RNS CODING

Redundant RNS coding achieves error control by providing redundant RNS modules and a general RNS error checker. With this approach the complete power of RNS error detection/correction theory is brought to bear on the problem of detecting erroneous residue digits, locating faulty modules through subspace projection, and correcting errors through base extension. This approach is very general in that the mathematics provides powerful error control capabilities, including error detection, error location, error correction and softly degraded performance in the presence of repeated hardware failures. It may also result in the minimal hardware redundancy within the VLSI modules themselves. The disadvantage of this approach is that a rather complex error checker is required to fully utilize the benefits of the powerful algebraic properties provided by redundant RNS coding.

A RNS code is constructed as a direct sum of many simple modular structures (either fields or rings) which have moduli that are pairwise relatively prime integers. If $\{m_1, m_2, \ldots, m_L\}$ is the set of moduli and $M = m_1 \cdot m_2 \cdots m_L$, then the interval [0, M-1] is called the legitimate range of the RNS because it represents the useful computational range of the number system. RNS arithmetic is performed as

$$(x_1, \ldots, x_L) * (y_1, \ldots, y_L) = (z_1, \ldots, z_L),$$

where $z_i = (x_i * y_i) \bmod m_i$ and * denotes addition, subtraction, or multiplication. Since z_i is determined entirely from x_i and y_i, RNS arithmetic is carry-free in the sense that there is no propagation of information from the i-th channel to the j-th channel, $i \neq j$. The lack of carry propagation means that errors in one digit cannot be propagated into other digits during operations involving addition, subtraction, or multiplication. The non-weighted structure of the RNS code is another property that makes RNS arithmetic useful in the design of fault tolerant processor structures. A faulty module can be identified by RNS error checking techniques and disconnected without affecting other modules. If the original RNS contains enough dynamic range, the reduced processor can continue to function with a reduced dynamic range (word length).

A redundant RNS is formed by adding extra moduli that are not used to increase the computational range, but rather provide additional degrees of freedom needed for error detection and correction. If r redundant moduli are added to create L+r total moduli, all L+r moduli must be pairwise relatively prime to ensure a unique representation for each state in the system.

1063-6889/93 $03.00 © 1993 IEEE

A simple example illustrates the principles of RNS error detection and correction. Consider a redundant RNS defined by $m_1 = 3$, $m_2 = 4$, $m_3 = 5$, and $m_4 = 7$, where the legitimate (computational) range is provided by m_1 and m_2, and where m_3 and m_4 provide the redundancy necessary for error checking. All numerical quantities must be scaled so that the results of all calculations always remain within the legitimate range [0, 11]. The illegitimate range [12, 419] is used only when a single digit error occurs. Note that a legitimate state can be correctly represented by any three of the four residues (as well as by any two of the residues). For example, $x = 5 \approx (2, 1, 0, 5)$ is correctly represented by (2, 1, 0) in the reduced RNS defined by $\{m_1, m_2, m_3\}$, by (2, 1, 5) in $\{m_1, m_2, m_4\}$, by (2, 0, 5) in $\{m_1, m_3, m_4\}$, or by (1, 0, 5) in $\{m_2, m_3, m_4\}$. Therefore if one of the original residue digits is erroneous, and if one digit is discarded from the original set of four residues, a correct representation results if and only if the erroneous digit is discarded. Once an erroneous digit is located, the correct value can be found by base extension using the subset of correct digits.

The mechanism by which a single digit error transforms a legitimate number into an illegitimate one is easily explained using the Chinese Remainder Theorem. Whenever an error occurs in the i-th digit, the resulting illegitimate number x' can be expressed as the correct number plus an error, $x' = (x + e_i) \bmod M_T$. The minimum e_i always causes x' to fall outside the legitimate range, whereas the maximum e_i is small enough so that large errors cannot wrap-around into the legitimate range. For the example above, the Chinese Remainder Theorem takes the form

$$(x) \bmod 420 = \big(140(2x_1)\bmod 3 + 105(1\cdot x_2)\bmod 4 + 84(4x_3)\bmod 5 + 60(2x_4)\bmod 7\big)\bmod 420$$

from which it is seen that the minimum possible error is 60, while the maximum is 360. Therefore, a single error always transforms x into an x' that falls into the illegitimate range [12, 419].

The error detection/correction procedure requires three basic steps: 1) It is first determined if an error has occurred by checking if the number is legitimate or illegitimate, 2) the erroneous digit is found by discarding the digits one at a time until a legitimate reduced representation is found, and 3) the correct digit is produced by a base extension using the reduced set of digits found to be legitimate in step 2. There are many variations on this procedure, and many different approaches to carrying out the details of each step which are described in the literature. In general, the error control approach described in this section has not received any widespread application in practice, largely because the error checker itself is relatively complicated and is not well suited for real time operation.

Much of the effort in designing for RNS fault tolerance involves determining that an error has occurred, and locating the module responsible for the incorrect result. The approach described above might be called an *external* error checking scheme, since the entire collection of residue digitals is taken into a global error checker which then determines if an error occurred and where it originated. Another approach, which is simply mentioned here for the sake of completeness, carries the *local* error checking philosophy to a greater degree. In this approach some simple local error checking techniques are applied within each RNS module to internally check for RNS digit errors. Recently Beckman et. al. [4] published work on a group-theoretic framework for fault-tolerant computation that provides a rather compete theory of analysis and design for error correcting codes that can operate within a specific RNS module. Although a complete discussion of this promising approach is beyond the scope of this paper, such techniques may eventually lead to fault tolerant designs with less redundancy and less overhead power consumption by eliminating the need for complicated error checkers and the extensive replication of hardware.

3. RNS MODULAR REPLICATION

The questions of complexity and reliability of a complicated error checker motivate the search for alternative error checking mechanisms that avoid complicated error checking procedures. One of the best known schemes to achieve fault tolerance in general computer systems is to simply provide three completely independent computer systems and to compare their results at various stages of a computational task. As long as two of the computers agree, the result is declared to be correct. If two out of three systems consistently agree, but the third consistently disagrees, the latter is declared "faulty" and is removed from further consideration. Reliability is maintained by continuing to compare the results of the remaining two properly functioning systems. If the two functioning systems eventually cease to agree, then a second failure has occurred, and some type of off-line diagnostics can be invoked to determine which of the two has failed. Once the faulty system is identified and removed from further voting, the computational process can continue on the remaining functioning computer. This fault tolerant scheme, illustrated in Fig. 1, is well known as triple modular redundancy (TMR).

Quadruple modular redundancy (QMR) is a related concept in which every important component is replicated four times. In one popular implementation of QMR, each circuit board that contains a critical subsystem is provided with two copies on the same board. In addition, every circuit board is duplicated, so that the bus interfaces with four identical copies of each subsystem. The outputs of subsystem pairs are compared locally on each board before the results are placed on the bus. As long as there is agreement at the board level a *correct* signal is passed to the system controller. However, when the local board level comparison fails to obtain an agreement between the subsystems on that board, a signal is sent to the system controller indicating that an error has occurred on the

corresponding board. This result is not used, but the correct result is produced by the duplicate board-pair, which presumably achieves agreement between its subsystems during this time. This concept is called quadruple modular redundancy (QMR), which is illustrated in Fig. 2.

With QMR, there is generally no need to perform error correction on-line, but rather enough redundancy is built into the design so that error-free results continue to be produced in redundant modules even while the system is undergoing physical board replacement. However, this type of fault tolerant system is inefficient with respect to hardware redundancy and power consumption. It is not very attractive in situations where the computer system is inaccessible to a technician, such as in computers carried on unmanned spacecraft.

A similar type of fault tolerance, illustrated in Fig. 3, can be achieved with an RNS design that does not require an elaborate error checker, but relies on double modular redundancy (RNS-DMR) and simple comparator circuits. For example, suppose an RNS architecture is chosen that has five information digits and one redundant digit, i.e., the system has approximately 17% redundancy in terms of excess word length. Suppose also that two copies each RNS module are built into the hardware and corresponding RNS digits are compared at various check points during the computation. If a particular RNS digit pair does not agree, that digit is declared erroneous. An erroneous RNS digit can either be discarded, or it can be reconstructed using the RNS correction mechanism discussed previously in Section 2. If the corresponding RNS module pair consistently fails to agree, then the module is declared faulty and permanently removed from the system, or simply removed until a technician can perform a module swap. In an environment where technician servicing is impractical or impossible, it may be possible to provide spare modules within the hardware so that numerous failures can be discarded before the system catastrophically fails. This concept is a form of soft system degradation in which the effective word length (dynamic range) becomes incrementally shorter as failures occur, until eventually the word length is too short for effective computation. The fault tolerant architecture described above can handle both hard (permanent) and soft (transient) errors. The only difference between the two cases is that a faulty module will typically be put back into operation on the next cycle if soft errors are suspected, whereas it will be permanently removed from operation if it develops a history of faulty performance that is indicative of hard failure.

In the above design the modular redundancy and pairwise voting (comparing) is similar to QMR, although the inherent modularity of the RNS architecture allows more flexibility and increased efficiency. While the QMR requires a factor of 4.0 in hardware redundancy, the RNS design described above requires a hardware redundancy factor of 2.34 (a factor of 2.0 comes from the DMR and the 0.34 comes from the 17% excess word length in each module) to achieve a similar degree of fault tolerance. This approach attempts to creatively use RNS redundancy with a simple distributed error checking scheme to avoid the difficulties of a complicated error checker that cannot be distributed effectively among the individual VLSI modules [5].

4. SERIAL-BY-MODULUS ARCHITECTURES

In many DSP applications the high speed of a fully parallel implementation may not be essential, in which case an interesting alternative is a serial-by-modulus (SBM) RNS structure. In a SBM-RNS architecture the residue digits $x_1, \ldots, x_L$ are transmitted through the processor serially, i.e., x_1 arrives first, then x_2, etc. Assume that the moduli are b-bit integers, so that there are "b" parallel bits throughout the entire processor. In rough terms, the SBM RNS architecture operates L times slower than a fully parallel design, although the hardware complexity is somewhat reduced. This reduction in hardware complexity is not proportional to the speed reduction, because RNS arithmetic is ultimately implemented by some form of table look-up, and it remains necessary to have appropriate ROM-based tables for every modulus in the RNS. However, there will be some reduction in complexity due to common adders and busses for all moduli. Another attractive feature is that the shorter word length of the signal path facilitates more effective testing of circuits during manufacture. Serial-by-modulus arithmetic, similar in principle to bit-serial binary arithmetic that was popular in the early days of digital filters, represents a compromise between the slower speed of bit-serial binary arithmetic and the high cost of bit parallel binary arithmetic [6]. The motivation is to reduce the hardware complexity of the implementation by going to a SBM-RNS design, and then to provide fault tolerant DMR by duplicating each SBM module in the hardware. Obviously a SBM-RNS-DMR design does not provide much protection against hard errors because a failure in one module is likely to be catastrophic with respect to all the RNS digits it must produce. However, the scheme appears to be effective for soft errors, as well as for certain types of memory errors in the RNS arithmetic look-up tables.

An SBM-RNS architecture using prime moduli requires an efficient realization for both mod (p_i-1) logarithmic addition (multiplication) and mod p_i addition (accumulation). A typical design uses a standard 2's-complement adder whose output addresses a ROM correction table, which produces the correct modular sum from the binary sum. When the adder implements an index addition the inverse logarithmic mapping is encoded into the same correction table so translation from index form to residue form does not require any additional hardware. Consider an RNS defined by the four 5-bit moduli 31, 29, 23, and 19. When two residues are added in a 2's-complement adder, the result may require as many as six bits, including the high order carry-out bit. Furthermore, since there are four moduli, it will take two "label" bits to uniquely identify each residue so that it can be unambiguously identified with the correct timing interval. In total each residue, together with its timing label and carry-out buffer, can be encoded with 8-bits, as illustrated

in the block diagram of Fig. 4. The ROM in Fig. 4 consists of four independent correction tables corresponding to the four different m_i, i = 1, 2, 3, and 4. The 2-bit timing label, appended to the corresponding residue by two extra bits, serves as the two highest order bits of the address, so the correct look-up table is automatically addressed. The circuit of Fig. 4 contains *zero detect* and *force zero* circuits, which are necessary to handle the case of a zero operand when the module is used as a multiplier that requires mod(p_i-1) addition. When the module is used for simple mod p_i addition the special zero handling capability is turned off.

Since the label bits are attached to each residue digit (as opposed to being generated locally by a synchronized clock), simple comparators can be used at strategic locations to check that no timing skew has occurred and to guarantee that digit synchronization is properly maintained throughout the processor. A design based on the four moduli listed above realizes more than 18 bits of effective word length in a module that has only an 8-bit word length. Not only has the total hardware requirement of the SBM-RNS structure been reduced as compared to the completely digit-parallel form, but the module can be more easily tested due to its short word length. Of course, the effective speed of the processor is reduced by an approximate factor of four. Note that two modules similar to the one shown in Fig. 4 (one for multiplication and one for addition) form the basis of a sum-of-products DSP kernel. The sum-of-products concept is illustrated in Fig. 5 where the ROM in the standard IA (index-add) cell is programmed to correct for both mod(p_i-1) addition and to implement the inverse index mapping. The standard A (add) cell is programmed for simple mod p_i addition.

In one particularly interesting configuration the SBM-RNS architecture could be used for the design of an elemental component in a sum-of-products processor that is required to operate in a noisy environment where the probability of soft errors is high. In such an environment the processor would continue to compute additional residues until it has processed enough error free digits to provide the dynamic range needed for the final output. Such a *compute-until-correct* fault tolerant concept is feasible because the digits in the RNS code are not weighted and have no relative significance. As long as a final number of error-free digits can cover the dynamic range of the output, and as long as the output translation device is given the proper information as to which digits are correct, the Chinese Remainder Theorem translation algorithm can be modified under program control to produce a valid output. The *compute-until-correct* fault tolerance concept could be used in noisy environments for systems that must keep functioning as long as possible without the benefit of technician servicing. An example of such a computing environments are those encountered by computing systems that operate in deep space probes. Although *compute-until-correct* fault tolerance is an interesting concept, it will not be further addressed in this paper.

In a second configuration the SBM-RNS architecture could be used as the basis of a variable word length processor. This concept is developed more fully in the next section, where a computer simulation is presented to illustrate the principles of operation.

5. VARIABLE WORD LENGTH FILTERS

The SBM-RNS architecture can be used as the basis for the design of a variable word length (VWL) processor [6,7]. In the following discussion it will be assumed that the overall processor has DMR to protect primarily against soft errors. Consider the situation in which a large amount of stored data must be processed at high speed by a sum-of-products operator, such as a digital correlator or an FIR digital filter. This scenario may occur in video processing, where frames are buffered and stored to give the processor working time to complete essential processing on each frame. Assume that the processor is designed to have a total of L moduli. When the output of the sum-of-products processor is small, some of the residue digits are redundant and, therefore, do not have to be computed. For simple RNS operations such as addition, subtraction, and multiplication, the residue digits corresponding to different moduli do not interact with each other. Therefore, redundant digits do not need to be computed since their results are not required. Which terms are in excess depend solely on the final output of the computation and not on intermediate results. Since the algebraic structure implemented by the RNS is an integer ring, for computations involving addition and multiplication, the size of intermediate results is not important in determining the output dynamic range. So, for a sum-of-products computation, only the size of the final result determines how many residues are necessary for that computation.

To implement a VWL filter, certain assumptions must be made in order to adaptively change the number of residue terms. Since the output of the current calculation is unknown, it is not certain how many digits are necessary; thus, the previous output(s) must be used. It is assumed that the output signal is slowly changing such that the current computation would use approximately the same number of terms that were needed in the previous calculation. A set of buffer residues must be added at times to make sure the output always remains valid. Since this can take away some of the efficiency of the algorithm, as small a buffer as possible should be used. Several different methods of determining the buffer are discussed in the following section.

5.1 Buffering Methods

In case the current output exceeds the range of the minimum set of moduli needed for the previous computation, one or more extra terms may be added to prevent overflow. The added buffer reduces efficiency but is necessary in some situations. This section outlines several buffering methods.

5.1.1. Constant Buffer

With a constant buffer, once the minimum number of terms is determined by the result of the previous calculation, one more residue term is added to prevent overflow in the next calculation. This creates a buffer range that is added to the current range so that the new result can exceed the current range and still remain valid. Even with a buffer, overflow can occur if the output increases too abruptly. If the signal is not changing slowly enough, then the current output could possibly jump over the buffer range and create an overflow condition. If two redundant terms are added, instead of only one, the probability of overflow is further reduced.

5.1.2. Adaptive Buffer

One drawback to the constant buffer method is that extra terms may not always be necessary and will only slow down the system. For instance, the output could be far from the end of a range and not changing fast enough to go into the buffer range. This means that no buffer term would be necessary until the output gets closer to the upper edge of the current range. If the constant buffer method were used, the extra term(s) would unnecessarily be computed. However, there are instances with any signal when a buffer is needed because the previous output is near the edge of the current range and, no matter how slowly the signal is fluctuating, the next may spill into a larger range. A method that could adapt to these situations should improve the efficiency of the filter by eliminating unwanted calculations.

An adaptive algorithm proposed in this work determines when to use a buffer by calculating how close the previous output is to the edge of the next larger range. If it far from the boundary then no buffer is added. If it is close, then two buffer terms are added. If it is between these extremes then only one buffer term is added. The current range is taken to be the minimum range necessary to represent the previous output. The closeness ranges are fixed in size and do not fluctuate when the current range changes.

5.1.3. Linear Prediction

The buffering methods described above depend on the assumption that the output is slowly changing so that the current output will not be far from the previous one. This is not necessarily a good assumption and may cause the output to jump over the buffer. Also, they do not take into account if the output is decreasing towards a smaller range. They only assume that the output will be larger than the previous and, therefore, will need a larger range. If a prediction algorithm is added to the above methods, then better tracking should occur. Some of the experiments documented in [7, 8]], as well as the example presented in the next section, use a simple two-term linear prediction method prior to computing the buffer. In this case, a linear prediction of the current output from previous outputs is used to estimate of the range requirement of current output.

5.2 Computer Experiments

Computer experiments were performed in software written in C that was designed to emulate an SBM FIR filter. The program takes a specified set of strictly prime moduli, computes the necessary correction tables, and stores them in arrays which act as the hardware ROM. The filter coefficients and the input data are read from files in binary form; binary-to-residue and residue-to-binary conversions are performed inside the program. By passing different parameters from the command line, the simulator is designed to perform any of the buffering algorithms previously discussed.

Data analysis was also performed by the simulator. For each output computed, a tabulation of the actual number of residues used was saved in a file. From this, the average number of residues for the entire batch of data was computed, along with a histogram showing the statistical distribution of the number residues were used. Furthermore, the locations of and the total number of overflow errors were tracked throughout the process. To show how an optimum VWL algorithm could perform, the minimum number of necessary residue terms was stored for each output computed. An average was computed to determine the optimum figure of merit.

Due to space limitations it is possible to present here results from only one experiment, although many experiments were performed and are fully documented in [7]. The results presented here are for the seven moduli {31, 29, 23, 19, 17, 13, 11}, an RNS that has a dynamic range of approximately 30 bits. When the number of moduli required is less than maximum, the subset chosen is taken from the largest modulus to the smallest so that the broadest sub-range is retained. For example, if only three terms are needed then {31, 29, 23} are used.

All the different buffering methods were simulated by first estimating the current output by simply using the previous output, or by linear prediction using the previous two output samples. For each estimated output, the minimum number of residues necessary to represent that number is computed, and the buffer terms are then added to that number. For the constant buffer case, the buffer size is defined to be either one or two, depending on the method chosen. With the adaptive buffer method, the two closeness ranges are passed as parameters (tabulated in the left column of Table 1) to the simulator for buffer selection.

The test data used as an input signal for processing by the VWL filer in this example was chosen to be an experimentally measured neural signal recorded from a single brain cell of a rat, with a record of 1000 samples. This particular test signal was chosen for this experiment because the signal contains spikes that require a large dynamic range, although there is considerable "idle" time between the spikes during which the VWL algorithm can reduce the computational range. The filter characteristic used in this VWL filter experiment was a low-pass FIR filter designed using the Hamming window.

One particular experiment is summarized in Table 1, where results are listed for a constant buffer algorithm with one term, a constant buffer algorithm with two terms, and an adaptive buffer algorithm with different choices for two closeness ranges, as listed. In this experiment the filter coefficients and the input were scaled so that the range of the output did not exceed the range of five moduli. However, the number of residue digits that are actually used can be as high as seven. The low-pass filter used in these comparisons had a normalized cutoff of 0.1 rads.

It can be seen in Table 1 that as the closeness ranges are reduced in size, the average number of moduli used by the adaptive algorithm decreases. However, as the number decreases, so does the average dynamic range, until the number of overflow errors begins to increase rapidly. Note that the linear estimation scheme is more conservative in the sense that for the same closeness ranges it maintains a lower error rate. But it also does not reduce the average number of residues as much. These results are not intended to be conclusive, but rather to illustrate how an adaptive word length design might function. More research will be needed to determine the effectiveness of VWL designs in practical problems.

ACKNOWLEDGEMENTS

The research is supported by the National Science Foundation under grant number NSF MIP 91-00212.

REFERENCES

[1] M. A. Soderstrand, W. Kenneth Jenkins, Graham A. Jullien, and Fred J. Taylor, eds., *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, New York, NY, 1986.

[2] H. Krishna, K.-Y. Lin, and J.-D. Sun, "A coding theory approach to error control in redundant residue number systems - Part I: Theory and single error correction," *Transactions on Circuits and Systems*, vol. 39, no. 1, pp 8-17, January 1992.

[3] K. M. Elleithy and M. A. Bayoumi, "Fast and flexible architectures for RNS arithmetic decoding," *Transactions on Circuits and Systems*, vol. 39, no. 4, pp 226-235, April 1992.

[4] P.E. Beckman and B. R. Musicus, "A group-theoretic framework for fault-tolerant computation," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, Vol 5, pp 557-560, San Francisco, CA, March 1992.

[5] W. K. Jenkins and B. A. Schnaufer, "Fault tolerant architectures for efficient realization of Common DSP kernels," *Proceedings of the 35th Midwest Symposium on Circuits and Systems*, Washington, D. C., pp 1320-1323, August 1992.

[6] W. K. Jenkins and S. F. Lao, "The design of an integrated RNS digital filter module based on serial-by-modulus residue arithmetic," *Proceedings of the 1987 International Conference on Signals, Systems, and Computers*, Port Chester, New York, pp 634-638, October 1987.

[7] A. J. Mansen, "Variable Word-length DSP Using Serial-by-Modulus Residue Arithmetic, M.S. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana-Champaign, May 1993.

[8] W. K. Jenkins and A, J. Mansen, "Variable word length DSP using serial-by-modulus residue arithmetic, *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, April 27-30, Minneapolis, MN, to appear.

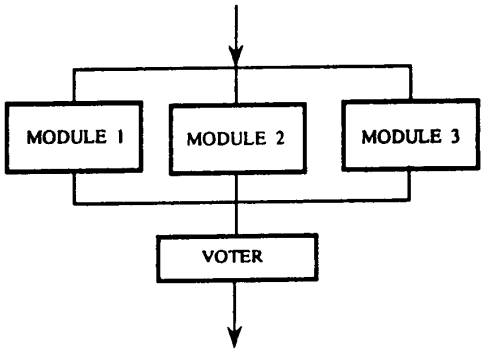

Fig 1. Conventional triple modular redundancy (TMR).

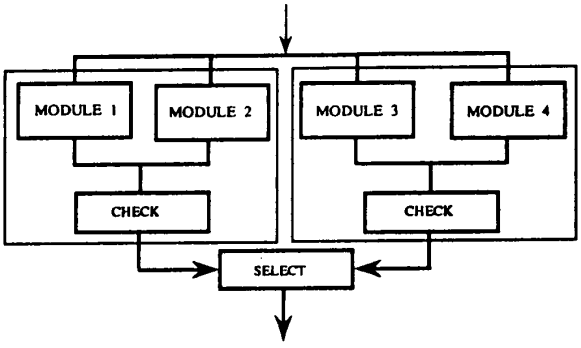

Figure 2. Quadruple modular redundancy (QMR).

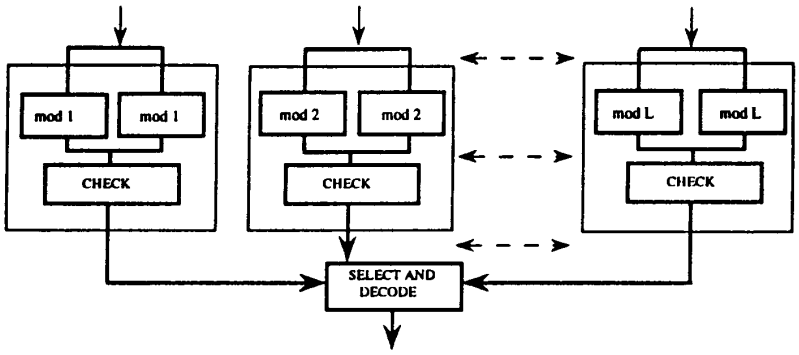

Figure 3. RNS double modular redundancy (DMR).

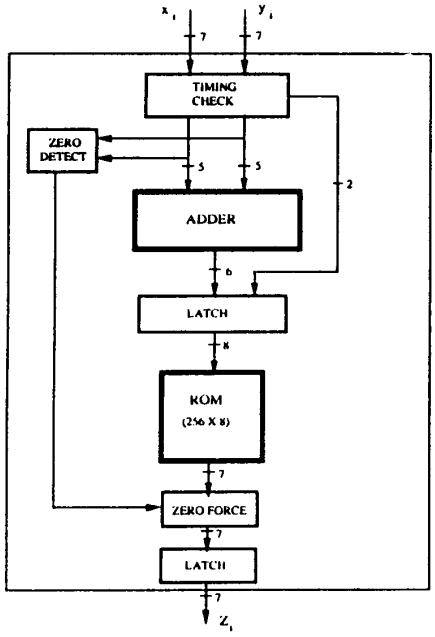

Figure 4. Serial-by-modulus RNS computational element.

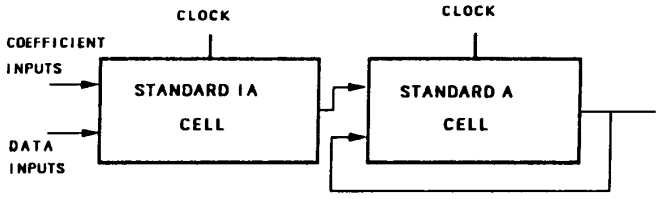

Figure 5. Two SBM computational elements forming a sum-of-products DSP module.

Table 1. Results of Using the 7-term VWL RNS Filter on Neural Data

	w/out Linear Estimation		w/ Linear Estimation	
Buffer Method	Errors	Ave. # of Res.	Errors	Ave. # of Res.
Constant Buffer				
1 Term	0%	5.204	0%	5.220
2 Term	0%	6.202	0%	6.215
Adaptive Buffer				
10^6, 10^7	0%	5.940	0%	5.951
10^5, 10^7	0%	5.479	0%	5.526
10^4, 10^7	0%	5.273	0%	5.298
10^3, 10^7	0%	5.214	0%	5.228
10^2, 10^7	0%	5.205	0%	5.219
10, 10^7	0%	5.204	0%	5.218
1, 10^7	0%	5.204	0%	5.218
10^5, 10^6	0%	5.219	0%	5.265
10^4, 10^6	0%	5.013	0%	5.036
10^3, 10^6	0%	4.953	0%	4.966
10^2, 10^6	0%	4.944	0%	4.957
10, 10^6	0%	4.943	0%	4.956
1, 10^6	0%	4.943	0%	4.956
10^4, 10^5	1%	4.558	0%	4.614
10^3, 10^5	2%	4.495	0%	4.542
10^2, 10^5	2%	4.481	0%	4.532
10, 10^5	2%	4.479	0%	4.531
1, 10^5	2%	4.479	0%	4.531
10^3, 10^4	17%	4.137	2%	4.322
10^2, 10^4	18%	4.121	2%	4.311
10, 10^4	18%	4.119	2%	4.310
1, 10^4	18%	4.119	2%	4.310
10^2, 10^3	48%	3.696	5%	4.245
10, 10^3	51%	3.649	6%	4.244
1, 10^3	51%	3.649	6%	4.244
10, 10^2	77%	3.154	7%	4.230
1, 10^2	80%	3.119	7%	4.230
1, 10	95%	2.437	7%	4.225