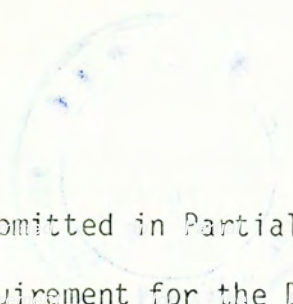


Implementing IIR Filters
via Residue Number Systems

by

Tai Leong Charn



A Thesis Submitted in Partial Fulfilment
of the Requirement for the Degree of
Master of Philosophy in Electronics.

The Chinese University of Hong Kong

May 1983.

443759

thesis
TK
7872
F5T34



Contents

Abstract

Acknowledgement

1. Introduction	p.1
2. Approximation for recursive filters using Bilinear-transformation method	p.6
2.1 Digital filter realization	p.6
2.2 Brief review of Bilinear-transformation	p.8
3. Introducing residue number system	p.12
3.1 Encoding and decoding of residue numbers	p.13
3.2 Modular arithmetic	p.17
4. Residue number techniques for recursive digital filtering	p.19
4.1 Scaling for IIR filters	p.19
4.1.1 Specialized residue classes	p.20
4.1.2 Scale factor being a modulus	p.22
4.1.3 Polarity ambiguity	p.27
4.2 Overflow detection in redundant residue number system	p.30
4.2.1 Redundant residue number system	p.30
4.2.2 Overflow detection	p.33
4.2.3 Numerical examples	p.38
4.2.4 Hardware considerations of redundant modulus $m_0 = 2$	p.42
4.3 Overflow suppression	p.45
4.4 A versatile residue system for recursive filtering.	p.49

5. Discussion

p.53

6. Conclusion

p.61

References

Appendix - A

APL programs to simulate the proposed residue number system for recursive digital filtering.

Appendix - B

Hardware implementation of second order IIR filters using APPLE II micro-computer.

Abstract

After briefly reviewing the bilinear transformation technique for digital filter design we introduce the structure of a residue number system and some notations. Then, a new algorithm to detect overflow via redundant residue number system will be presented. The approach eliminates the time-consuming conversion of mixed-radix digits but at the same time requires a polarity shift operation to handle signed numbers. Because the scaling process which converts the fractional coefficients into integer values is unavoidable in recursive digital filtering, attention is given on this part for efficient implementation of residue number decoding. The decoder is realized by table look-up technique. It is well known that the overflow oscillation can be suppressed by changing the overflow characteristics. A similar operation is derived for the residue number system. Finally, a residue number system combining the above features is established. Although the illustrative example is a second order section, it can easily be extended to higher order IIR filters. Programs are written to simulate the system and the results are presented to demonstrate the principles.

Acknowledgement

I would like to express my sincere gratitude to Prof. C.F. Chen, my supervisor, for his helpful guidance and counsel during the research.

Also, I want to acknowledge the Croucher Foundation who offers a studentship for supporting the work.

1. Introduction

A digital filter is a digital system that can be used to filter discrete-time signals which may be real time or recorded signals. It can be realized by the use of special purpose hardware, or by the implementation of iterative sequencing of software instructions (computer programs) executed on a processor. The former gets more of our attention because we are often dealing with the real time signals.

It is well known that digital filters have certain advantages over their analogue counterparts. These include high accuracy, high reliability and the capability of handling low frequency signals. A very important additional advantage is that filter characteristics can be changed easily by modifying the filter parameters in memory. The digital filters tend to replace analog filters in many applications not only because of the above merits, but also due to the tremendous advancement of very large-scale integrated (VLSI) circuit technology. Recently, the drop of component cost facilitates the hardware implementation of digital filters.

VLSI devices such as bit-slices and microprocessors have

been employed to realize digital filters[1-4]. Stand-alone microprocessor implementation has the problem of restricted bandwidth, which is a parameter assessed from the sampling rate of the filter. This comes from the fact that the instruction sets of microprocessors are for general purpose applications. Multiplication instructions are usually not available and even if they do, a long processing time is needed. To overcome the time-consuming multiplications which are required in a given difference equation, a separate fast multiplier is brought to co-operate with the processor. Such approach reaches a sampling rate of 555 kHz for a 2nd order low-pass filter [1]. As a comparison, the speed of microprocessor-based implementation is only 0.625 kHz [2] which is much lower and has limited applications.

Subject to the bottleneck of multiplications, many attempts have been made to find other ways of realizing the arithmetic operations in a digital filter. A noteworthy approach using distributed arithmetic technique is discussed by Peled and Liu[5]. It is applicable to the implementation of second-order sections with fixed coefficients represented in fix-point notation. According to this method, the values will be pre-calculated and stored in read-only-memory (ROM). Consequently, only add and shift operations are required in computing the difference equation. At the absence of multiplication, the operation speed is thus greatly

improved. A second-order section utilizing this algorithm is claimed to operate in real time on a signal with a 10 MHz bandwidth[5].

... Another approach appearing to compete favourably with the conventional filters is the implementation of residue arithmetic. For many years, residue number coding has been recognized as a system which provides the capability of high speed multiplication and addition[6]. Such a system usually consists of several sub-systems having shorter word-lengths. Each subsystem evaluates the difference equation individually and produces the corresponding output simultaneously. Because the word-length is not large, it is possible to execute the multiplication with a look-up table stored in ROM. This technique is known to be a fast operation of which the processing time is dependent on the time to access the ROM. Besides, there are no truncation or rounding errors arising in performing the arithmetic operations since residue system is composed of integers only.

The use of residue number coding in realizing digital filters requires a fractional-to-integer conversion. For a stable filter, it usually has fractional coefficients in the difference equation. Multiplied by a suitable factor, we will get a set of integer coefficients. No problem will

arise in this step for the implementations of FIR filters because no feedback term is involved in the difference equation. The implementation of FIR filters using residue technique has been reported in[7]. For IIR filters, a scale-down process is required to recover the actual result before feeding back in calculating the difference equation. Unfortunately, division as well as sign detection[13] and magnitude comparison are difficult to handle in residue systems. Scaling, or fixed constant division, is then a requisite operation during implementation. For efficient recursive digital filter realization, some specialized residue classes are designed[8,10]. Given these special classes, the scaling is easy to handle by slightly modifying the Chinese Remainder Theorem, which provides a means for translating the residues back to natural numbers. It is noted that the scaling and residue decoding are carried out simultaneously. In the following chapters, a technique which separates the above two operations is discussed and shown to have the advantage of saving hardwares.

Another problem, large-scale limit cycles which is introduced by arithmetic overflow, has urged researchers to pay attention on it. For a second order section implemented by 2's complement arithmetic, the output oscillations can be suppressed by modifying the overflow characteristic[11,12]. This method is applicable provided that the system itself is

capable of detecting arithmetic overflow. Residue number systems are lack of this capability but not for redundant residue number systems of which the dynamic range is larger than required. Error detection and correction using redundant residue number system has been discussed in[9]. From which, a method based on mixed radix conversion is described for overflow detection. The conversion generates the mixed radix digits sequentially and thus is time-consuming. Utilizing the same redundancy technique, an overflow detection scheme which gets rid of the conversion is discussed in the following chapters. No special operation is required except the polarity shift. The detection is accomplished by comparing the redundant residue with the one decoded from the set of non-redundant residues. Such approach is fast and easy to implement but incapable of correcting errors.

This research suggests various approaches to solve the problems encountered in implementing recursive digital filters using residue number system. All the results are verified by program simulations.

2. Approximation for recursive filters using Bilinear-transformation method

2.1 Digital filter realization

A digital filter is often described by a difference equation as shown in (1), where $\{X_n\}$ is the input sequence, $\{Y_n\}$ the output sequence, and $\{a_i\}$, $\{b_i\}$ are the filter coefficients.

$$Y(n) = \sum_{i=0}^{N-1} a_i X(n-i) + \sum_{i=1}^M b_i Y(n-i) \quad (1)$$

The corresponding Z-domain transfer function of the filter is as follows:

$$H(z) = \frac{\sum_{i=0}^{N-1} a_i z^{-i}}{1 - \sum_{i=1}^M b_i z^{-i}}$$

By manipulating the transfer function, there are several methods to realize the digital filter. Three of them, namely direct, parallel and cascade realizations, are usually employed. The cascade realization requires the transfer function to be factored into a product of 2nd order transfer functions as

$$H(z) = \prod_{i=1}^L H_i(z)$$

$$\text{where } H_i(z) = \frac{a_{0i} + a_{1i} z^{-1} + a_{2i} z^{-2}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}}$$

Alternatively, the transfer function can be expanded into partial fractions as

$$H(z) = \sum_{i=1}^L H_i(z)$$

$$\text{where } H_i(z) = \frac{a_{0i} + a_{1i} z^{-1}}{1 + b_{1i} z^{-1} + b_{2i} z^{-2}}$$

This gives the parallel form realization. Another form of realization will be called direct canonic realization as shown in Fig. 2.1. The number of unit delays employed in this method is equal to the order of the transfer function.

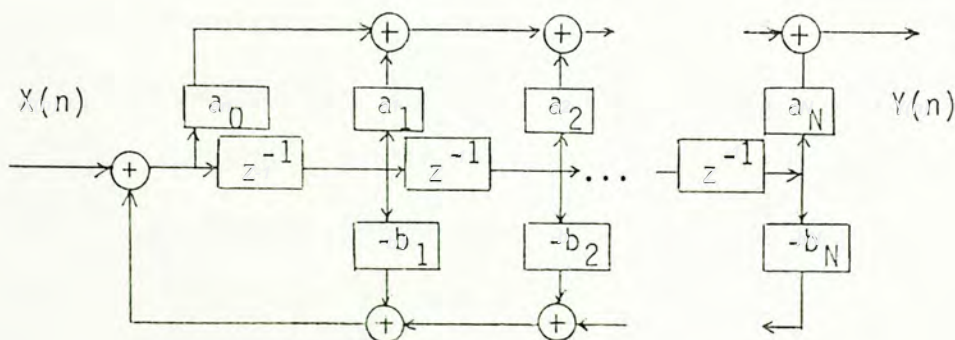


Fig. 2.1 Direct form realization of digital filter.

According to the value of M in eqn.(1), digital filters can be classified into two types. If $M=0$, then there will be no feedback terms (coefficient $b_i's=0$) and the filter is defined as a nonrecursive or finite impulse response (FIR) filter. When $M>0$, the filter is named as recursive or an infinite impulse response (IIR) filter.

2.2 Brief review of Bilinear-transformation

The task of designing a digital filter is mainly to find the coefficients satisfying some prescribed specifications. Although these specifications may be stated in time domain or frequency domain, we traditionally use the latter one. For IIR filter, one of the design technique starts from an analog filter having the required characteristic, and use various transformation methods to get the corresponding digital counterpart. The reason for using this approach is that the design methods of analog filters are well established. Several techniques[18] are presented to perform the transformation. In the following, a second-order IIR filter is designed by utilizing Bilinear-transformation method. We devote to the low-pass section only since other standard types namely, bandpass, high-pass or band-stop

filters, can be obtained from the low-pass filter by the well known frequency transformation technique.

The essence of the Bilinear-transformation method comes from the mapping described by eqn.(2), which transforms those in the left-half s-plane into the interior of unit circle in the z-plane.

$$s = (z-1)/(z+1) \quad (2)$$

Substitute $s=jv$ and $z=\exp(jwT)$ into eqn.(2) to find the relationship between the frequencies in both domains. The result is given by eqn.(3), where v and w are respectively the frequencies of continuous and discrete cases. Clearly, the function governed by (3) is not a linear mapping which leads to some distortions.

$$v = \tan(wT/2) \quad (3)$$

As an illustrative example, let the specifications for a digital filter be: second-order, low-pass with pass-band cut-off frequency, $f_c=100\text{Hz}$ and sampling frequency, $f_s=1\text{kHz}$ (sampling period= 1ms). For simplicity, the prototype analog filter will be chosen from the well-known Butterworth class. Eqn.(4) shows the transfer function of a second-order Butterworth low-pass filter, where v_c is the cut-off

frequency. From (3), $\omega_c=628.32$ rad/sec for the desired filter.

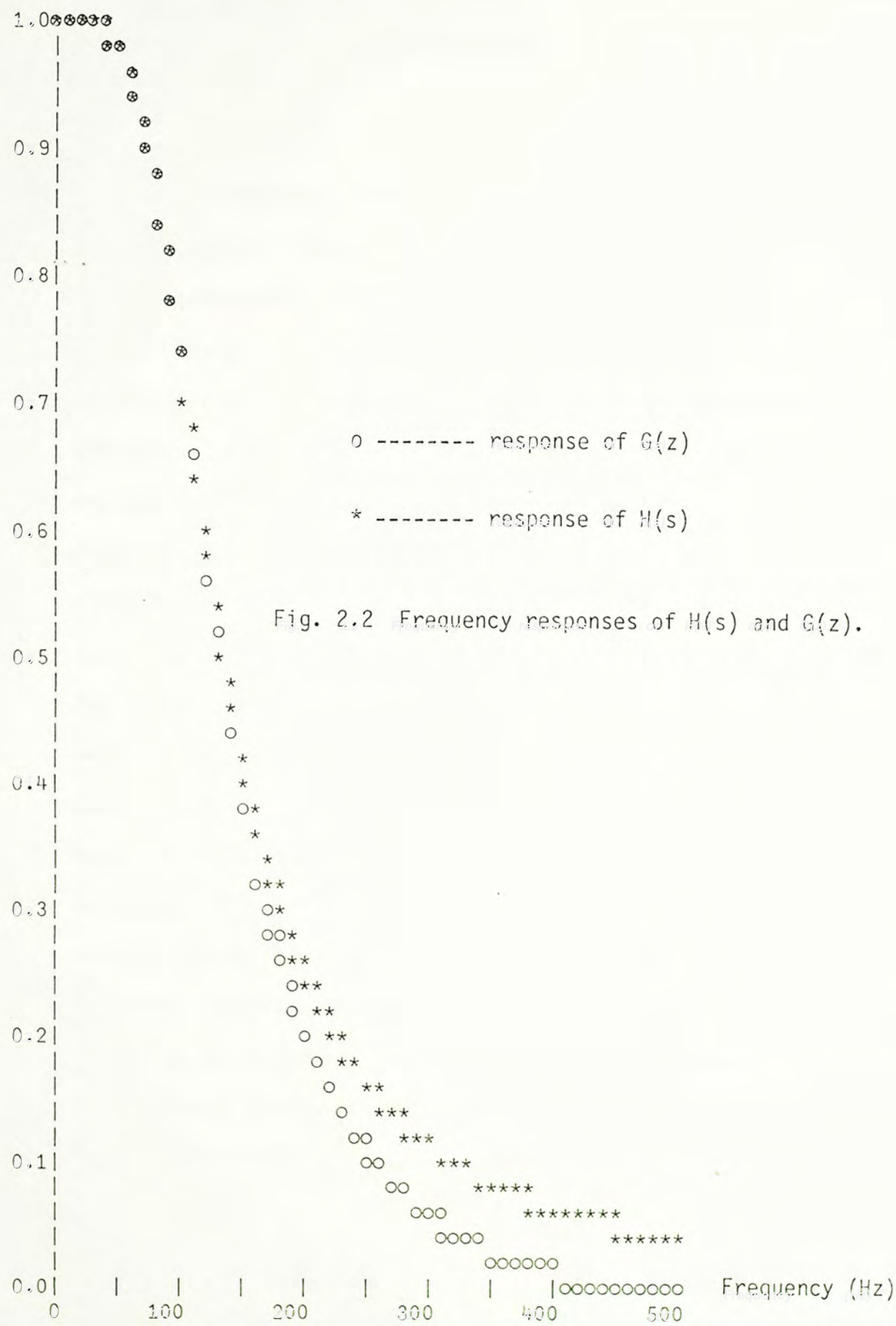
$$H(s) = \frac{1}{(s/\omega_c)^2 + 2(s/\omega_c) + 1} \quad (4)$$

The corresponding transfer function of the digital filter can be calculated using eqn.(2) either by direct substitution or matrix manipulation[19].

$$G(z) = \frac{0.0675 z^2 + 0.1349 z + 0.0675}{z^2 - 1.1430 z + 0.4128} \quad (5)$$

To demonstrate the differences between an analog filter and the digital one derived from it using Bilinear-transformation, the frequency responses of $H(s)$ and $G(z)$ are evaluated and plotted in Fig. 2.2. The result shows a good matching at the low frequency range and large deviation in the vicinity of the folding frequency.

Normalized amplitude



3. Introducing residue number system

A residue number system consists of a set of pairwise relatively prime moduli $\{m_1, m_2, \dots, m_L\}$. The dynamic range which represents the useful computational range of the number system is $[0, M)$, where M is the product of all moduli. i.e. $M = \prod_{i=1}^L m_i$. In order to handle signed numbers, the dynamic range will be divided into positive and negative regions. If M is odd, the range of the residue representation is $[-(M-1)/2, (M-1)/2]$; if M is even, the range is $[-M/2, (M/2)-1]$. Each natural integer in the above ranges is uniquely coded by a sequence of L residue digits. Any number not in the range will then be classified as in overflow range. To guarantee the result is correct, the maximum and minimum values during intermediate calculations must be set within the dynamic range. Based on this criterion, the number (L) of moduli can be suitably chosen. During arithmetic calculation, each residue digit is evaluated independently. The operation can be very fast by selecting small moduli. Final result is then recovered from all residue digits through decoding using the Chinese Remainder Theorem.

3.1 Encoding and decoding of residue numbers

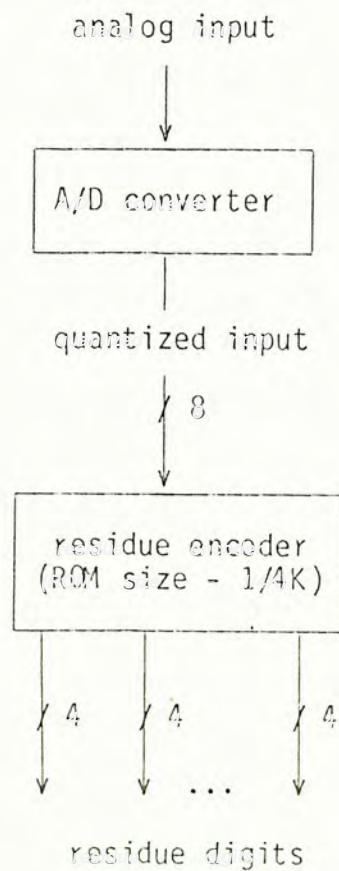
Fast operations of residue systems are achieved at the expense of an additional overhead cost of translating data into and out of the system. These two processes are respectively defined as encoding and decoding operations. During encoding process, a natural integer X is converted into a sequence of residue digits $\{x_1, x_2, \dots, x_L\}$ according to (6), where $|X|_{m_i}$ denotes the positive remainder of the division X/m_i for a certain integral quotient. Of course, the remainder is always less than m_i .

$$x_i = \langle X \rangle_{m_i} = \begin{cases} |X|_{m_i} & , X > 0 \\ m_i - |X|_{m_i} & , X < 0 \end{cases} \quad (6)$$

For example, $\langle 15 \rangle_{13} = 2$, $\langle -15 \rangle_{13} = 11$.

If a modulus has the form of 2^k , where k stands for any integer, the encoding is easy to implement. For binary digit representation, the task simply extracts the k least significant bits from the number being encoded. For moduli other than the above form the encoding is rather complex and requires arithmetic operations. However, since the input samples of filters are taken from an analog-to-digital converter which usually has 8-bit word length, it is

possible to encode these samples based on the technique of table look-up. The tables may be implemented by read-only-memory (ROM) which has an acceptable size of 256 words. This approach not only reduces the complexity but also provides a high-speed operation. If the system has L moduli, a total number of L residue encoders (ROMs) are required.



Decoding, an inverse operation of encoding, requires the derivation of a natural number from a set of residue digits $\{x_1, x_2, \dots, x_L\}$. The Chinese Remainder Theorem given by (7) can be used to carry out this operation. Denote M_i^{-1} as the

multiplicative inverse of M_i such that $\langle M_i(M_i^{-1}) \rangle_{m_i} = 1$.

$$X = \langle \sum_{i=1}^L M_i M_i^{-1} x_i \rangle_M, \quad M_i = M/m_i \quad (7)$$

As an illustration, suppose $m_1=2$, $m_2=3$, $m_3=5$ then eqn. (7) becomes

$$X = \langle 15 x_1 + 10 x_2 + 6 x_3 \rangle_{30}$$

The value represented by the residue set $\{x_1, x_2, x_3\} = \{1, 2, 1\}$ will be

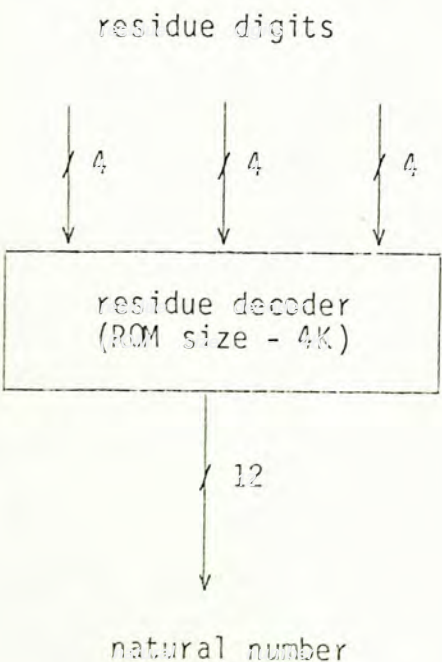
$$X = \langle 41 \rangle_{30} = 11$$

The value M is generally a large composite integer because it is the product of all moduli. Consequently, the mod M multiplication, as shown in (7), is costly to implement with hardware. Because of this, a technique based on the Peled and Liu structure is proposed in [7] to implement the Chinese Remainder algorithm. This approach reduces the mod M multiplication into a relatively common mod M addition.

Another solution to the implementation is using mixed-radix conversion process as discussed in [15]. This technique gives a shorter word length in ROM. A similar approach but with the combination of residue-to-binary and digital-to-analog operations is proposed in [14]. The method is particularly useful when it is desired to translate the

residue samples directly into analog form.

In addition to the above approaches, it is feasible to implement (7) using ROMs by directly feeding the residue digits into the ROM's input if the number and sizes of the moduli are small enough. For instance, the above illustration requires 1 bit for m_1 , 2 bits for m_2 and 3 bits for m_3 to hold residue digits. If the decoder is implemented by ROM, then the table must have an entry of 6-bit. This is definitely possible since nowadays a ROM having 16-bit entry is not very unusual. Typical size of modulus, however, is around 4- to 5 bits. This suggests that a residue number system having 3 moduli is suitable.



3.2 Modular arithmetic

The modular arithmetic is different from the conventional arithmetic in that it needs not take care of carry. During an arithmetic operation, two N-bit operands will produce a N-bit result. There is no truncation or rounding error even for multiplications. The so-called residue operations are defined by (8), where * denotes either addition, subtraction or multiplication.

$$x_1 x_2 \cdots x_L * y_1 y_2 \cdots y_L = z_1 z_2 \cdots z_L \quad (8)$$

$$z_i = \langle x_i * y_i \rangle_{m_i}, \quad i = 1, 2, \dots, L$$

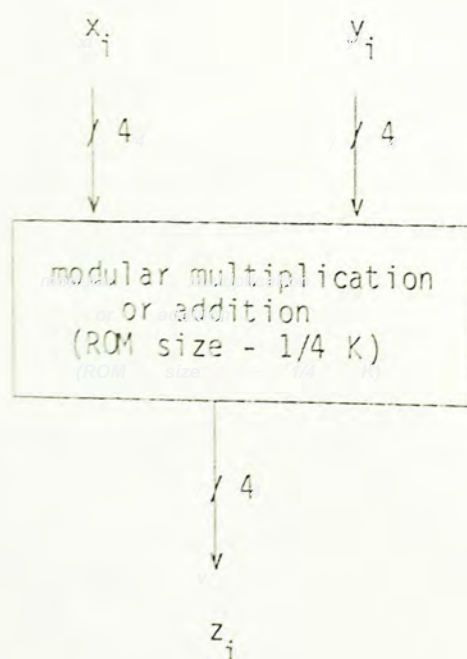
General division is excluded from the residue operations because it may produce fractional result which is not allowed in residue systems. However, fixed constant division is possible if its multiplicative inverse exists and the dividend is divisible. For example, consider the following two divisions $\langle 4/2 \rangle_5$ and $\langle 3/2 \rangle_5$, where the multiplicative inverse of 2 is 3.

$$(i) \quad \langle 4/2 \rangle_5 = \langle 4 \times 2^{-1} \rangle_5 = \langle 12 \rangle_5 = 2.$$

$$(ii) \quad \langle 3/2 \rangle_5 = \langle 3 \times 2^{-1} \rangle_5 = \langle 9 \rangle_5 = 4.$$

Clearly, (i) gives the correct result while (ii) does not.

Although conventional 2's complement multipliers and adders are still applicable to implement residue arithmetic operations, a residue encoding process is required to correct the results. This will hinder the filtering speed. To achieve faster operation, it inevitably comes back to the table look-up technique. According to this method, the multiplication and addition will be realized by ROMs. Since we have the problem of size limitation on ROM the residue digits must be as small as possible. When the sizes of residues cannot be further reduced, then it is possible to use a square-law multiplier as introduced in[16], or to compress the tables for modular arithmetics as discussed in[17].



4. Residue number techniques for recursive digital filtering

4.1 Scaling for IIR filters

Residue number systems have the merit of fast multiplication and addition. However, a difference equation with fractional coefficients can take no advantage of it. This is because residue number system only allows integers. For a FIR filter, the coefficients can be converted to integers by scaling. Such method however is not applicable to IIR filters. In order to utilize modular arithmetic, an appropriate scale factor (A) is multiplied to the original difference equation to produce integral coefficients as shown in eqn.(9a). The final result is then recovered by performing a constant division which is illustrated in eqn. (9b).

$$Y'(n) = \sum_{i=0}^{N-1} A a_i X(n-i) + \sum_{i=1}^M A b_i Y(n-i) \quad (9a)$$

$$Y(n) = Y'(n) / A \quad (9b)$$

As mentioned above, residue number system has no general

division. This requires us to evaluate (9) in three sequential steps. First of all, from (9a) a set of residue digits representing $Y'(n)$ are calculated through modular arithmetic. The second step will then involve the Chinese Remainder Theorem which determines the natural value of $Y'(n)$. Finally, the actual result is obtained by performing a scaling operation as indicated in (9b). In order not to hinder the filtering speed by these three steps, efficient scaling method must be devised.

4.1.1 Specialized residue classes

Appropriate choice of moduli in a residue number system can simplify the scaling operation[8,10]. This approach combines the last two steps and thus speed up the filtering. A special class is presented in the following to illustrate the principle of operation. The residue number system is supposed to have two moduli of the form $m_1 = m$ and $m_2 = m-1$ with a scale factor $A=m$.

From eqn. (7),

$$Y(n) = (M_1 M_1^{-1} y'_1 + M_2 M_2^{-1} y'_2 - k M) / A$$

where y'_1, y'_2 - residue digits of $Y'(n)$,

$$M_1 = m-1, M_2 = m, M = m(m-1),$$

$$0 < k < M_1^{-1} + M_2^{-1}.$$

This gives

$$Y(n) = (m-1)y'_1 - y'_1 + (y'_1 / m) + y'_2 - k(m-1)$$

Because $Y(n) < (m-1)$, this implies $Y(n) = \langle Y(n) \rangle_{m-1}$, so

$$Y(n) = \langle y'_2 - y'_1 \rangle_{m-1} \quad (10)$$

The above equation eliminates the term (y'_1 / m) so that a scaling error is arised, where $0 \leq (y'_1 / m) < 1$. It also demonstrates that the scaling and residue decoding for this special system can be replaced by a modular subtraction which is rather simple to implement.

4.1.2 Scale factor being a modulus

This approach requires that a modulus in the residue number system has same size as the scale factor. The set of residue digits for a natural value y' in this system will be denoted by $\{y'_1, y'_2, \dots, y'_L, y'_A\}$, where y'_A is the residue digit of the specific modulus which has same size as the scale factor. According to the property of residue, the result of the operation $(y' - y'_A)$ is divisible by the scale factor (A) . Consequently, as discussed in section 3.2, $(y' - y'_A)/A$ can be evaluated by modular arithmetic because the multiplicative inverse A^{-1} is defined and existing in any moduli except the specific modulus. The original scaling operation, y'/A , will now be replaced by $(y' - y'_A)/A$. This produces an error $0 \leq (y'_A/A) < 1$.

A residue number system employing this technique is shown in Fig. 4.1. As the system is used for recursive filtering, x and y are respectively the input and output samples. From the residue set of input sample $\{x_1, x_2, \dots, x_L, x_A\}$, we get another residue set $\{y'_1, y'_2, \dots, y'_L, y'_A\}$ by evaluating the difference equation. According to the result of y'_A , a set of modifying parameters $\{c_1, c_2, \dots, c_L\}$ will be generated through the block "encoding", where $c_i = \langle -y'_A \rangle_{m_i}$. The scaling operation is then carried out with the aid of these

parameters. i.e.

$$y_i = \langle (y'_i + c_i) A^{-1} \rangle_{m_i} \quad (11)$$

From this step, the residue set for the actual output sample $\{y_1, y_2, \dots, y_L\}$ can be obtained. Note that during the residue decoding, the residue digit y'_A is eliminated as indicated in Fig. 4.1. The reduction of residue digit is especially good for residue decoder that is realized by ROM. A smaller table size is required for this case. In Fig. 4.1, the block labelled as "scaling" and "encoding" can be implemented by ROMs.

As a comparison, a system having same dynamic range is sketched in Fig. 4.2. This system when performing residue decoding requires all residue digits $\{y'_1, y'_2, \dots, y'_L, y'_A\}$. The ROM size will be increased which is proportional to the additional residue digit y'_A . Such approach is suitable for those residue number system that are composed of special residue classes. i.e. Both scaling and decoding are carried out simultaneously.

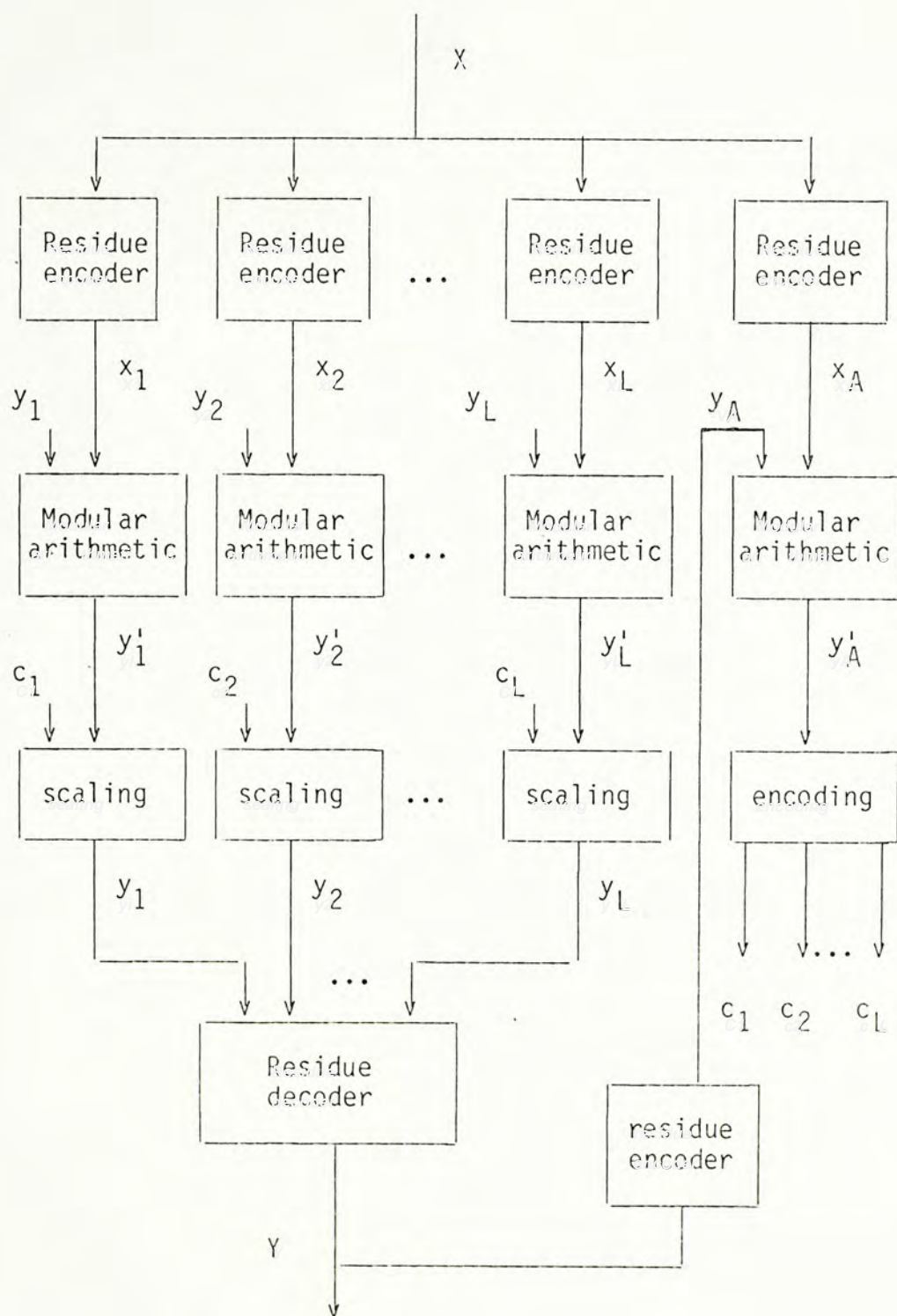


Fig. 4.1 Residue number system with one modulus being the scale factor.

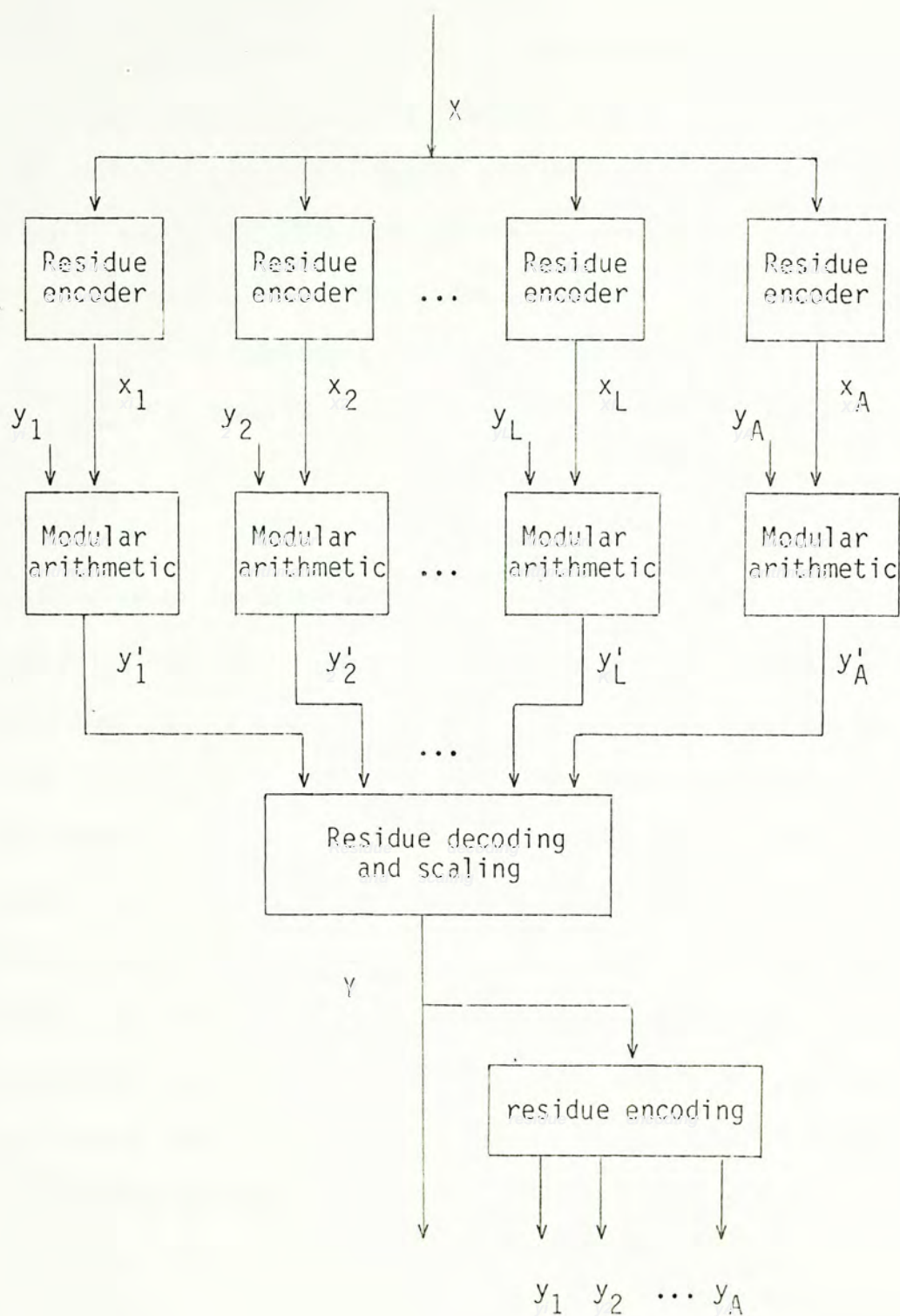


Fig. 4.2 A general residue number system for IIR filtering.

The scale factor must be large enough to convert fractional coefficients into integers with acceptable error. For example, a scale factor of 100 will preserve two significant digits after the decimal point. This requires 7 bits. If a modulus has such word-length, it is not profitable to implement modular arithmetic by ROM. Two operands will totally occupy 14 bits and this requires a 16 K ROM.

Because of large table size we must find another way of realizing modular arithmetic. It is known from section 3.1 that the residue encoding is simple for a modulus having the form of 2^k . As encoding is a step involved in modular arithmetic, it is reasonable to choose a scale factor of which the value is a power of 2. Instead of using ROM the modular addition now can be implemented by 2's complement adder. For modular multiplication, we can use the square-law multiplier as discussed in [16]. Three adders and two ROMs are needed for this method. Since only one operand is fed to the ROM, the table size is reduced significantly.

4.1.3 Polarity ambiguity

The scaling operation is a many-to-one mapping. A scaled result may come from several different inputs. When these inputs are around the boundary separating positive and negative ranges, it arises a problem of polarity change. The phenomenon is that a positive (negative) number after scaling will fall in the negative (positive) range. The sign is not preserved during scaling.

To analyze the cause of this problem, let A and M be respectively the scale factor and the dynamic range after scaling. The original dynamic range without scaling will thus be $M \times A$ (the scale factor being one of the moduli). Since an appropriate scale factor is an even value (power of 2), M must be odd to satisfy the condition of "pairwise relatively prime". The product of M and A is undoubtedly an even number. Refer to section 3.1, the positive and negative regions for the original system and the one after scaling can be classified as following.

	positive range	negative range
original system	$[0, (M \times A)/2 - 1]$	$[(M \times A)/2, (M \times A) - 1]$
after scaling	$[0, (M - 1)/2]$	$[(M + 1)/2, (M - 1)]$

The scaling operation of the largest positive and negative numbers are defined as

$$P = \text{INT} [(MxA)/2 - 1 + K] / A]$$

$$N = \text{INT} [(MxA)/2 + K] / A]$$

where $\text{INT} [X]$ denotes the integer part of X
and K is an offset to be determined.

In order not to have polarity ambiguity,

$$P \leq (M-1) / 2 \quad (12)$$

$$N \geq (M+1) / 2 \quad (13)$$

These imply $(1/2 + 1/A) > K/A \geq 1/2$.

For any value of A , $K = A/2$ will satisfy the above condition. With this, we conclude that the polarity ambiguity can be removed by adding an offset $A/2$ to the original data before scaling. For example, if a residue number system consists of the moduli $m_1 = 5$ and $m_A = 2$, the corresponding parameters will be $A = 2$, $M = 5$ and offset = 1. Refer to Fig. 4.3, a polarity error appears at the row where $X = 5$. The value $X = 5$ is regarded as a negative number in the original system. However, the scaling without offset produces a result of 2 which is considered as a

positive number and ensue the change of polarity. Correct scaled value 3, as shown in the rightmost column, is obtained if we add the offset. It is noticed that the dynamic ranges of the system before and after scaling are respectively $[-5,4]$ and $[-2,2]$.

original system		scaled results			
X		$\text{INT}[X/2]$		$\text{INT}[(X+\text{offset})/2]$	
0	(0)	0	(0)	0	(0)
1	(1)	0	(0)	1	(1)
2	(2)	1	(1)	1	(1)
3	(3)	2	(2)	2	(2)
4	(4)	2	(2)	2	(2)
5	(-5)	2	(2)	3	(-2)
6	(-4)	3	(-2)	3	(-2)
7	(-3)	3	(-2)	4	(-1)
8	(-2)	4	(-1)	4	(-1)
9	(-1)	4	(-1)	0	(0)

() = the corresponding signed values,

scale factor = 2.

Fig. 4.3 Elements of a residue number system before and after scaling.

4.2 Overflow detection in redundant residue number system

4.2.1 Redundant residue number system

Based on an existing residue number system we can set up a redundant residue number system. Let the set of moduli for the existing system be $\{m_1, m_2, \dots, m_L\}$, from which we define $M = \prod_{i=1}^L m_i$. This means that there are totally M states that can be handled by the system. As signed numbers are involved, half of the states are used to represent negative data. According to eqn. (6), the negative numbers will map onto the upper part of the interval $[0, M)$. Consequently, the dynamic range of the system is $[-(M-1)/2, (M-1)/2]$ for odd M , and $[-M/2, (M/2)-1]$ for even M . To obtain the redundant residue number system (RRNS), extra moduli $\{m_{L+1}, m_{L+2}, \dots, m_{L+r}\}$ will be added to the fundamental set. With the additional r moduli the number of states is extended from M to MT where $MT = \prod_{i=1}^{L+r} m_i$. Clearly, $MT > M$. If we keep the computational range of the RRNS the same as above, then redundancy appears. In other words, we have the redundant interval $[M, MT)$ if the operands and results of operations carried out in the RRNS are constrained to the range $[0, M)$. This is similar to the case of using five or more bits to represent a 4-bit number.

0 ----- M ----- MT

$[0,M)$ permissible range of original system

$[0,MT)$ extended range for the RRNS

$[M,MT)$ redundant interval

It is seen from above that there is no difference between a residue number system and a redundant residue number system except that the latter possesses redundant states. Using the upper range to represent negative numbers also applies to the RRNS. This implies that the mapping results would unavoidably fall in the redundant interval $[M,MT)$. For the reason that those states greater than M cannot be controlled by the original system, it is important to bring the mapping results back to the range $[0,M)$. A polarity shift [9] which is actually a Mod MT addition is performed to accomplish this. For M odd, a constant value of $(M-1)/2$ is added to the data; for M even, $M/2$ is added. After performing the circular shift, the positive and negative numbers are respectively mapped onto the upper and lower parts of the range $[0,M)$ in the RRNS indicated in Fig. 4.4. The purpose of the polarity shift is for range indication. Actual result must be recovered by subtracting a constant,

especially for those feedback data which occur in recursive digital filtering.

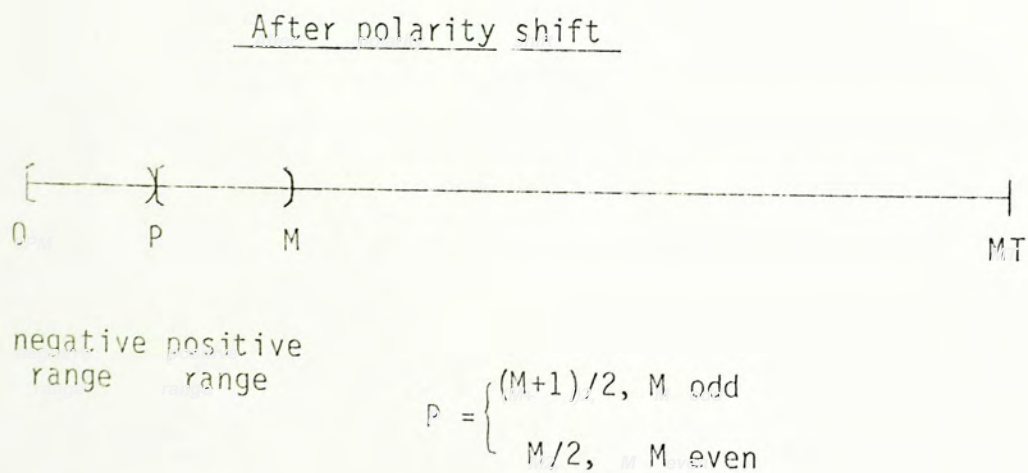
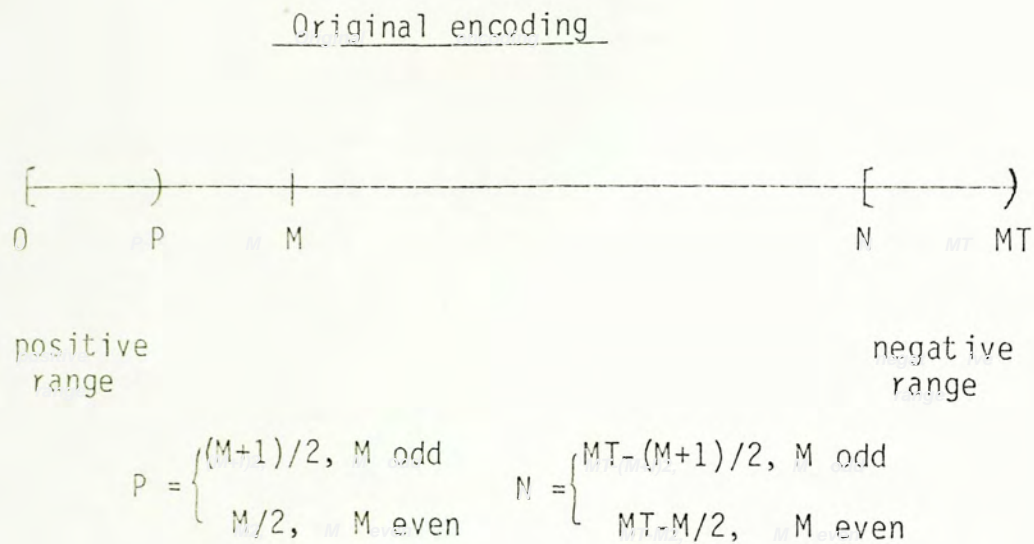


Fig. 4.4 Illustration of RNS intervals before and after polarity shift.

4.2.2 Overflow detection

A redundant residue number system having a redundant modulus m_0 is depicted in Fig. 4.5. In spite of the redundant modulus, the original residue number system which consists of the moduli $\{m_1, m_2, \dots, m_L\}$ produces an output Y_M . However, as an overall view, the RRNS itself would give an output Y_{MT} . It is noted that $M = \prod_{i=1}^L m_i$, and $MT = M \times m_0$. As indicated in Fig. 4.5, y_0, y_1, \dots, y_L are the residue codes representing the results after modular arithmetic operations. Using the decoding formula (7), the residue set $\{y_0, y_1, \dots, y_L\}$ would generate the output Y_{MT} , and by eliminating y_0 , the remaining residue digits will give Y_M . Obviously, $0 \leq Y_M < M$ and $0 \leq Y_{MT} < MT$.

As long as the calculated results are within the fundamental range $[0, M)$, Y_{MT} and Y_M should be equal because y_0 is redundant for this case. Direct comparison of Y_{MT} and Y_M , however, is not efficient. It requires a large residue decoder (mod MT) to find Y_{MT} . By taking the following operation, we can save one residue decoder. The parameter y_0 is immediately available at the redundant modulus system.

$$\langle Y_M \rangle_{m_0} = \langle Y_{MT} \rangle_{m_0} = y_0, \quad \text{if } 0 \leq Y_{MT} < M. \quad (14)$$

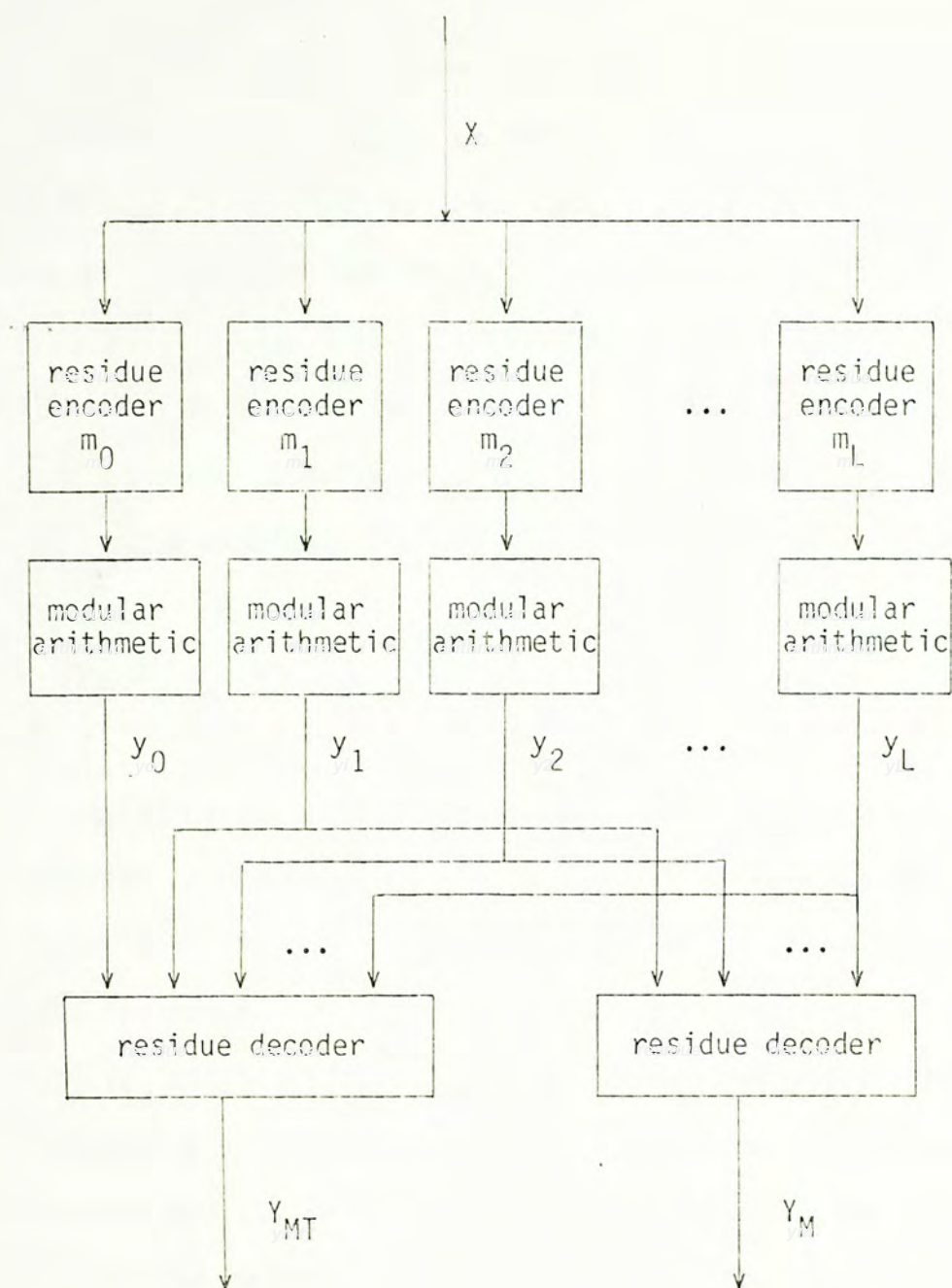


Fig. 4.5 Insertion of a redundant modulus m_0 .

Since the RRNS has a permissible range $[0, MT)$, it is possible for the results to fall in the redundant interval $[M, MT)$. When this is the case, Y_M would produce an incorrect data but not for Y_{MT} . In fact, we have $0 \leq Y_M < M$ and $M \leq Y_{MT} < MT$. Because they are both derived from the same residue set $\{y_1, y_2, \dots, y_L\}$, their residue codes in redundant modulus must be different to satisfy the one-to-one mapping. i.e.

$$\langle Y_M \rangle_{m_0} \neq \langle Y_{MT} \rangle_{m_0} = y_0, \text{ if } M \leq Y_{MT} < MT. \quad (15)$$

Having established the above criterion, we can proceed to overflow detection. It is known that after the polarity shift, the whole valid computational range will be mapped onto the range $[0, M)$. If a result lies in the range $[M, MT)$, then it can be classified as overflow. The capability of determining a result in $[0, M)$ or $[M, MT)$ has already been provided by (14) and (15). Eventually, we reach the purpose of overflow detection.

Let the residue set $\{k_0, k_1, \dots, k_L\}$ represent the constant (K) required for the polarity shift.

$$z_i = \langle y_i + k_i \rangle_{m_i}, \quad i = 0, 1, \dots, L. \quad (16)$$

From (16), the actual output $\{y_0, y_1, \dots, y_L\}$ is changed to

$\{z_0, z_1, \dots, z_L\}$. To distinguish whether a data is in the redundant interval, we can now compare $\langle Z_M \rangle_{m_0}$ and z_0 as described above, where Z_M is decoded from the residue set $\{z_1, z_2, \dots, z_L\}$. We conclude that overflow occurs only if $\langle Z_M \rangle_{m_0} \neq z_0$.

The utilization of an RRNS for overflow detecting is illustrated in Fig. 4.6. Because the polarity shift is a modular addition, ROM instead of binary adder implementation is preferred. An additional residue encoder will be required for the comparison. After determining there is no overflow, the real result Y_M can be obtained by subtracting the constant K from Z_M . The operation is no longer a modular subtraction. With reference to Fig. 4.4, the valid values of Z_M is in the range $[0, M-1]$. A 2's complement subtractor will exactly generate the sign of the number after subtracting the constant as illustrated in the following.

	constant K	Z_M	$Y_M = Z_M - K$
M odd	$(M-1)/2$	$[0, M-1]$	$[-(M-1)/2, (M-1)/2]$
M even	$M/2$	$[0, M-1]$	$[-M/2, M/2-1]$

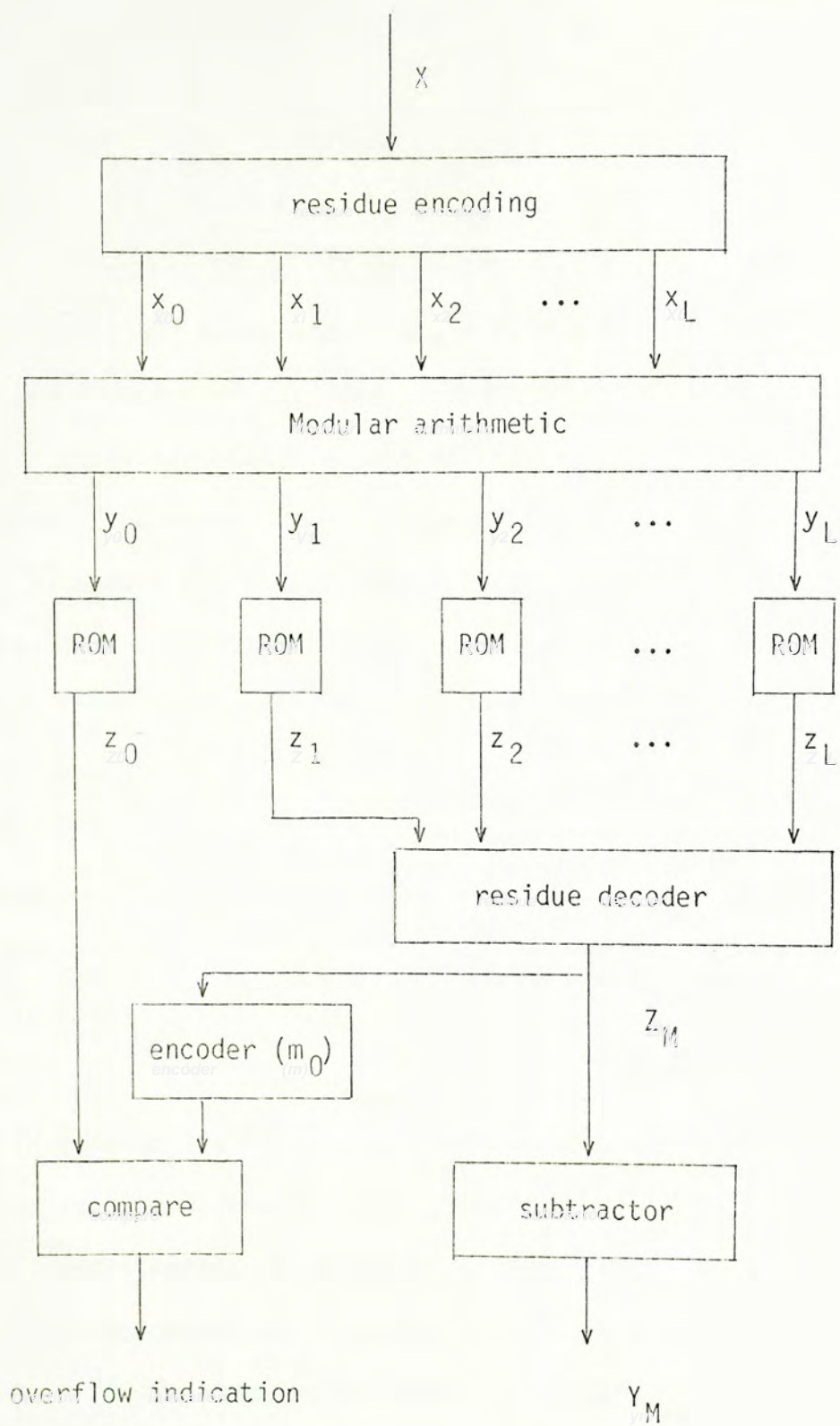


Fig. 4.6 A RRNS for overflow detection.

4.2.3 Numerical examples

For demonstration, a redundant residue number system consisting of three moduli $m_0 = 2$, $m_1 = 3$ and $m_2 = 5$ is chosen, where m_0 is the redundant modulus. Table 4.7 illustrates the possible sets of the residue codes for this system. The computational range of the RRNS is confined to $[-7,7]$ though the actual dynamic range is $[-15,14]$. As $M=15$ is an odd value, the constant associated with the polarity shift for this example will be 7 of residue codes (1,1,2). Considering the field of "signed numbers" in Table 4.7, those results after polarity shift are listed in the right column. For instance, given a state of value 4, it is interpreted as a positive number 4, but after the operation, it is regarded as a negative number -3.

Three examples involving addition and multiplication are given in the following. The operations are carried out by modular arithmetic, from which we demonstrate the principle of overflow detection. With the aid of Table 4.7, it is easy to perform the encoding and decoding operations.

states		signed numbers		residue codes		
	y_{MT}		(p)	y_0	y_1	y_2
0 -->	0	0	-7	0	0	0
	1	1	-6	1	1	1
	2	2	-5	0	2	2
	3	3	-4	1	0	3
	4	4	-3	0	1	4
	5	5	-2	1	2	0
	6	6	-1	0	0	1
	7	7	-0	1	1	2
	8	8	1	0	2	3
	9	9	2	1	0	4
	10	10	3	0	1	0
	11	11	4	1	2	1
	12	12	5	0	0	2
	13	13	6	1	1	3
	14	14	7	0	2	4
M -->	15	-15	8	1	0	0
	16	-14	9	0	1	1
	17	-13	10	1	2	2
	18	-12	11	0	0	3
	19	-11	12	1	1	4
	20	-10	13	0	2	0
	21	-9	14	1	0	1
	22	-8	-15	0	1	2
	23	-7	-14	1	2	3
	24	-6	-13	0	0	4
	25	-5	-12	1	1	0
	26	-4	-11	0	2	1
	27	-3	-10	1	0	2
	28	-2	-9	0	1	3
	29	-1	-8	1	2	4
MT -->	30					

p - polarity shift

Table 4.7 Residue sets of a special RRNS.

Case 1 : $y = 5 + (-2)$

Modular arithmetic : $(y_0, y_1, y_2) = (1, 2, 0) + (0, 1, 3) = (1, 0, 3)$

Polarity shift : $(z_0, z_1, z_2) = (1, 0, 3) + (1, 1, 2) = (0, 1, 0)$

From Table 4.7 : $Z_M = 10$

$$z'_0 = \langle Z_M \rangle_{m_0} = 0$$

Conclusion : No overflow because $z'_0 = z_0$

$$\text{Result : } y = z_M - 7 = 10 - 7 = 3$$

Case 2 : $y = 5 \times (-2)$

Modular arithmetic : $(y_0, y_1, y_2) = (1, 2, 0) \times (0, 1, 3) = (0, 2, 0)$

Polarity shift : $(z_0, z_1, z_2) = (0, 2, 0) + (1, 1, 2) = (1, 0, 2)$

From Table 4.7 : $Z_M = 12$

$$z'_0 = \langle Z_M \rangle_{m_0} = 0$$

Conclusion : Overflow for the reason $z'_0 \neq z_0$

($y = -10$ after multiplication)

Case 3 : $y = (-5) \times (-5)$

Modular arithmetic : $(y_0, y_1, y_2) = (1, 1, 0) \times (1, 1, 0) = (1, 1, 0)$

Polarity shift : $(z_0, z_1, z_2) = (1, 1, 0) + (1, 1, 2) = (0, 2, 2)$

From Table 4.7 : $Z_M = 2$

$$z'_0 = \langle Z_M \rangle_{m_0} = 0$$

Conclusion : $z'_0 = z_0$ implies no overflow. (Wrong!)

In case (3), the wrong conclusion is reached though it appears not to be. The above multiplication will give the value 25 which is obviously out of the range $[-7,7]$. The failure comes from the fact that the value 25 is not only an invalid number to the original RNS but also to the RRNS of which the permissible range is $[-15,14]$. Such a defect will become less harmful when the redundant range is extended. By setting the redundant modulus greater than $M/2$, which is the maximum value of operand, it is possible to get rid of the fault completely.

4.2.4 Hardware Considerations of Redundant Modulus $m_0=2$

An RRNS with the capability of overflow detection has been established. So far, we have not investigated the hardware complexity when an RNS is modified to an RRNS. Such a modification mainly involves the insertion of a redundant modulus system of which the internal structure is similar to the other fundamental modulus systems. It is known that the hardware cost is proportional to the word length of the modulus system. In order to save hardware during the insertion, the size is chosen to be as small as possible. Clearly, $m_0 = 2$ will be the optimal one.

A modulus system with word length of 2 has special advantages over the others. It is noticed that such a system is composed of two residues, 0 and 1. One bit is sufficient to represent all residues. Consequently, the hardware needed to configure this system will use logic gates as the major components. Having built such a system, it is easy to find the residue code of a natural number because the remaining job is simply to extract the least significant bit of the number. For modular arithmetic, the multiplication and addition can be realised by AND gates and Exclusive-OR gates respectively.

operands		x	+
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Furthermore, it is possible to replace the ROM by an Exclusive-OR gate for the polarity shift. Its residue code is 0 if the constant K is even; and 1 if K is odd. The following table shows the possible combinations of y_0, k_0 and z_0 . It is readily seen that z_0 is the result of EX-OR operation between k_0 and y_0 .

y_0	k_0	$z_0 = \langle y_0 + k_0 \rangle_2$
0	0	0
0	1	1
1	0	1
1	1	0

The redundant modulus system implemented by logic gates is depicted in Fig. 4.8, where a 2nd order IIR filter is used as an example. Such a system not only has simple structure but also can operate at very high speed. The result is produced after the delay of two logic gates.

Difference equation of a 2nd order IIR filter:

$$Y(n) = a_0 X(n) + a_1 X(n-1) + a_2 X(n-2) + b_1 Y(n-1) + b_2 Y(n-2)$$

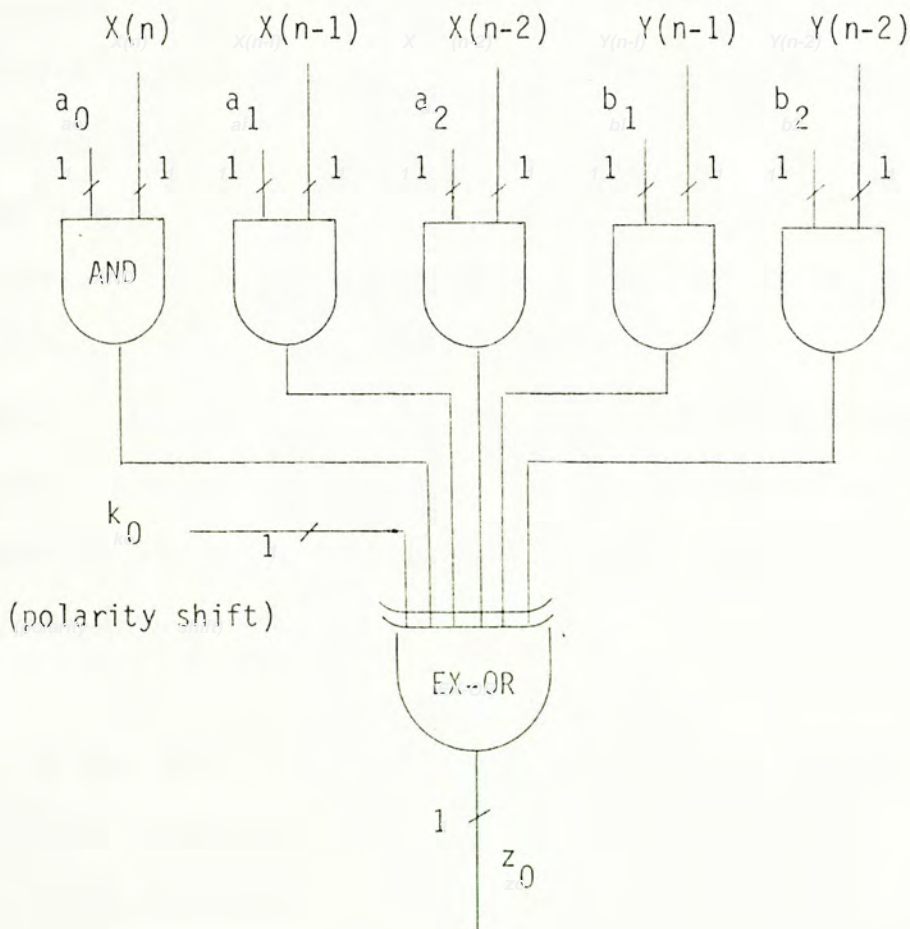


Fig. 4.8 Internal structure of a redundant modulus system having 1-bit word length.

4.3 Overflow suppression

Overflow of 2's complement arithmetic would introduce undesirable full-scale oscillations[12] which persist regardless of what input sequence is subsequently applied to the filter. The character of the oscillation has been analyzed in detail by [11]. It is proven that if the overload characteristic is modified to certain patterns, then no self-oscillations will be present. We have found empirically that for a stable second order IIR filter, the overflow of residue arithmetic can similarly produce self-oscillations as indicated in Fig. 4.9. The figure, which is the filter's step response, shows that oscillations appear after an overflow occurs at the sixth sampled point. There is no way to stop the oscillations even if input is set to zero.

To cope with the self-oscillation, we first consider the overflow characteristic of the residue arithmetic. As described previously, negative numbers of a residue system are located at the upper part of state range according to the complement technique. This is similar to 2's complement arithmetic in which positive numbers come first and then negative numbers as we count in ascending order.

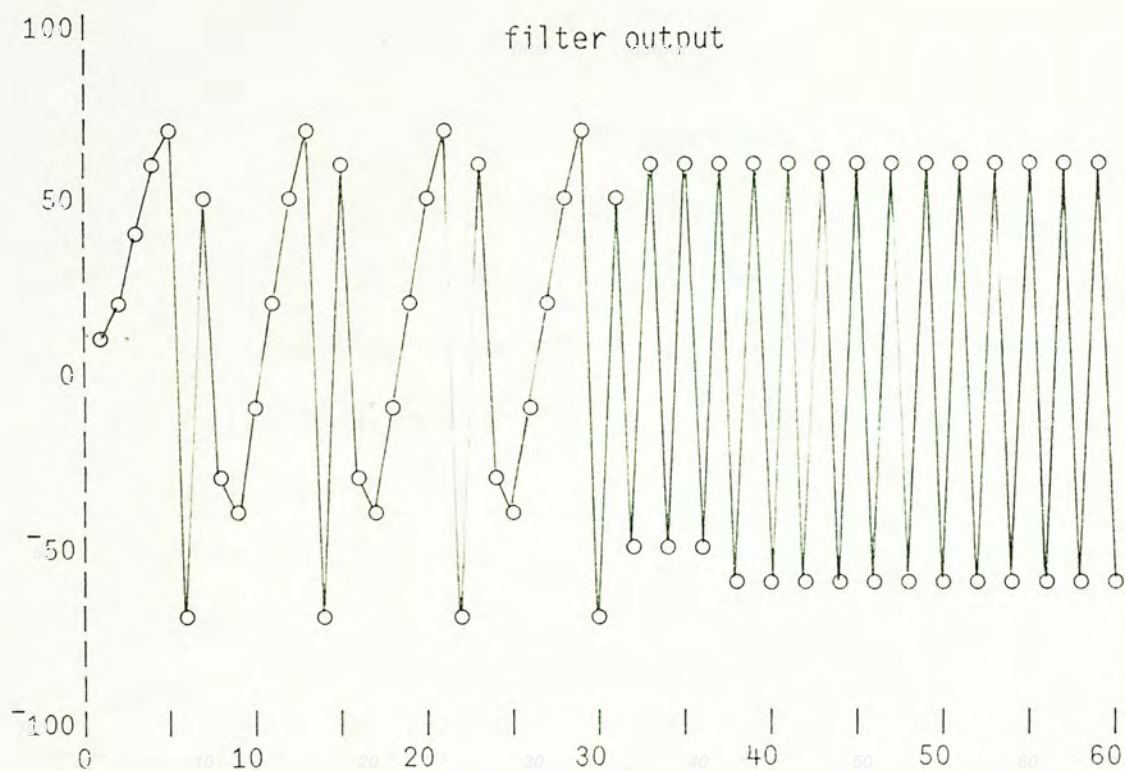


Fig. 4.9 Step response of a 2nd order IIR filter showing self-oscillations.

Consequently, the result of the addition of two positive numbers becomes negative if it overflows. For instance, the residue system utilized to implement the above second order filter has a dynamic range $[-71, 71]$ while its state range is $[0, 142]$. Negative numbers $[-71, -1]$ map onto the upper state range $[72, 142]$. Because 71 is considered as the largest positive number, overflow will certainly arise for the addition $71+1$. The result is undoubtedly 72 but will be regarded as -71 by the residue system for the reason that it falls in negative region. Such kind of overflow characteristic, which is depicted in Fig. 4.10(a), is exactly same as that for 2's complement arithmetic.

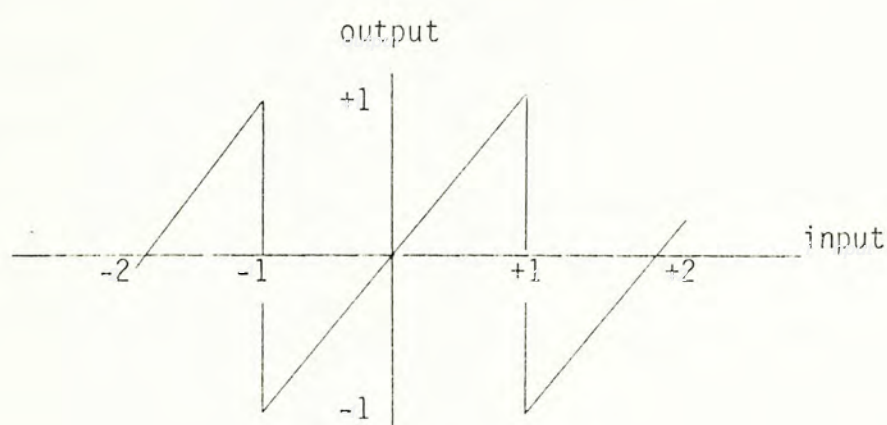
According to the analysis described in [11], the oscillations would be suppressed by modifying the overflow characteristic to the one as shown in Fig. 4.10(b). To achieve the specific characteristic, results which are out of range must be complemented. If the residue system has totally M states, then the complement of a value Y is defined by eqn.(17a).

$$\bar{Y} = M - Y \quad (17a)$$

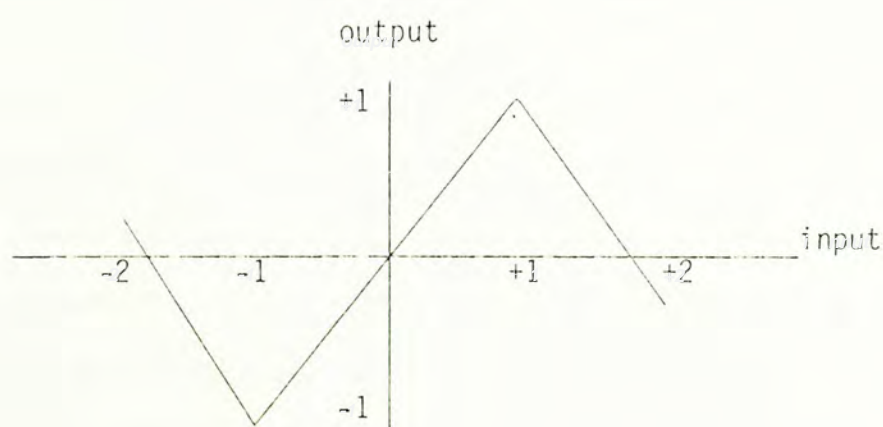
$$\bar{y}_i = m_i - y_i, \quad \bar{y}_i = \langle \bar{Y} \rangle_{m_i}, \quad y_i = \langle Y \rangle_{m_i}. \quad (17b)$$

With the same example as illustrated above, we have $M = 143$ and $Y = 72$, then the complemented result $\bar{Y} = 71$. That

means, we can still preserve the sign of overflow result but lose the magnitude information after performing the complement. Eqn.(17b), which directly finds the complemented residue digits, is the actual operation carried out by a residue number system. It is not difficult to verify that both expressions are equivalent.



(a)



(b)

Full scale dynamic range normalized to ± 1

Fig. 4.10 Overflow characteristic.

4.4 A versatile residue system for recursive filtering

Having discussed the various problems encountered during recursive filtering, we now proceed to obtain a summary. First of all, a residue number system which includes the features of scaling, overflow detection and suppression is established. The system is flexible and easily adapted to any order of IIR filters by modifying the structure of "modular arithmetic". Then, as an illustration, a second order low-pass filter is implemented. Typical functions such as step and impulse are input to the system to see the responses. The results are compared with those obtained by infinite precision arithmetic.

Fig. 4.11 illustrates the organization of a suggested residue number system which is set up according to previous discussions. There are four moduli, two of them (m_1, m_2) are classified as the fundamental modulus, one of them is the redundant modulus (m_0) and the remaining one (m_A) is associated with the scale factor. As the system is used for recursive filtering, X and Y_n are respectively representing the input and output samples.

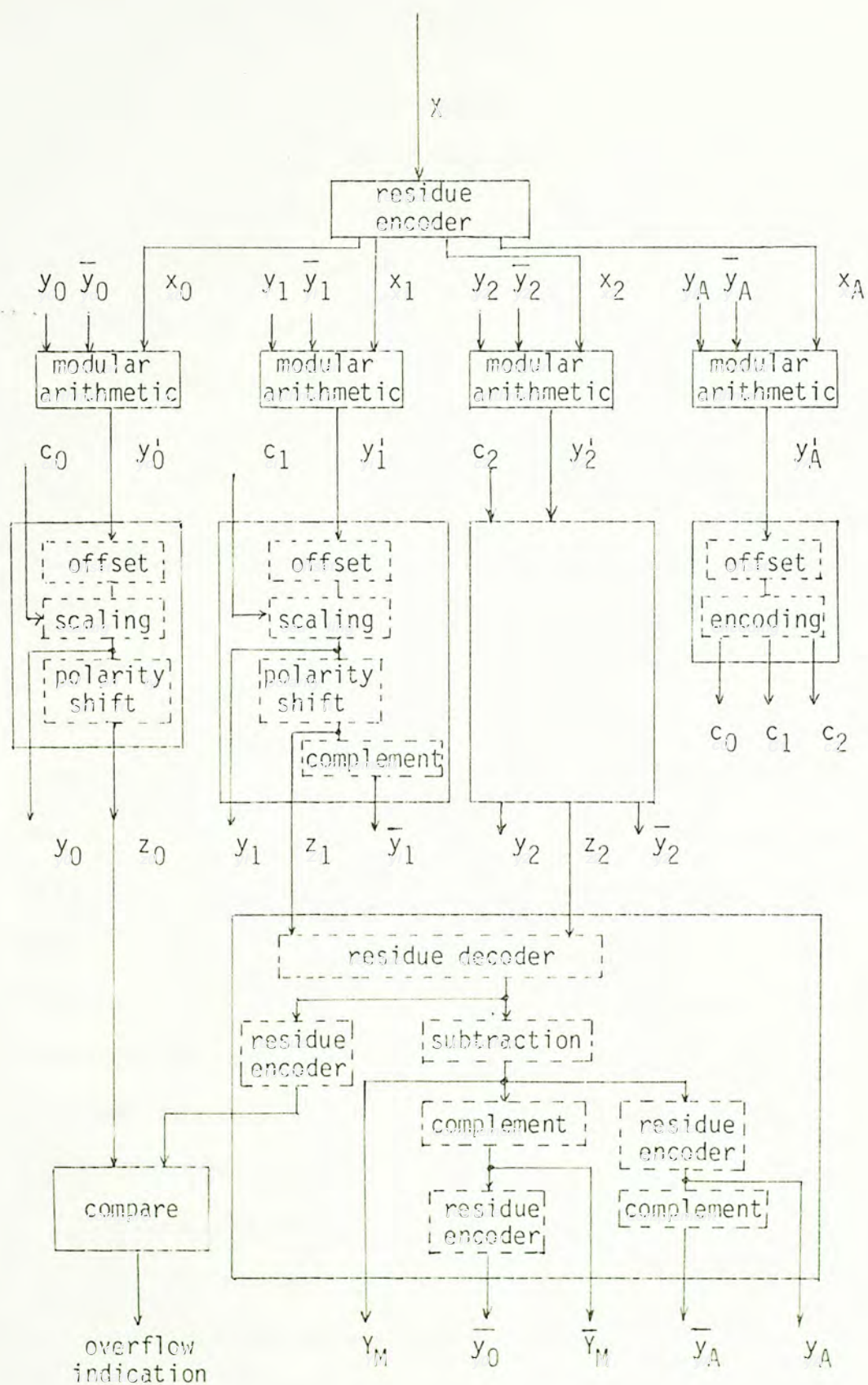


Fig. 4.11 A special RNS for recursive filtering.

At first sight, the system seems complicated. However, a careful investigation shows that several operations can be combined together. For instance, operations such as offset, scaling, polarity shift and complement are grouped as a single unit and actually realized by a ROM with input set $\{y'_i, c_i\}$ and output set $\{z_i, y_i, \bar{y}_i\}$. Such an arrangement not only reduces hardware complexity but also saves processing time because the operations must be carried out in series for direct realizations. Same technique can be found in output module in which residue decoding, encoding, subtraction and complement are put together. The ROM output, unlike the input, has no size limitation in the sense that additional ROMs can be connected in parallel with the original one to expand the output capacity. From Fig. 4.11, as Y_M and y_A are expected to be large values, two or more ROMs will be used to implement the output module. Finally, it is worth mentioning that if there is an overflow, the complemented results $\{\bar{y}_0, \bar{y}_1, \bar{y}_2, \bar{y}_A\}$ are selected, otherwise, the normal values $\{y_0, y_1, y_2, y_A\}$ are chosen for the operations concerned. Utilizing digital multiplexors, the selection is easy to implement. Depending on the filter's order, the internal organization of "modular arithmetic" may be varied. Fig. 4.12 shows the structure for a 2nd order IIR filter. For higher order, more registers are required to hold the sampled values and coefficients.

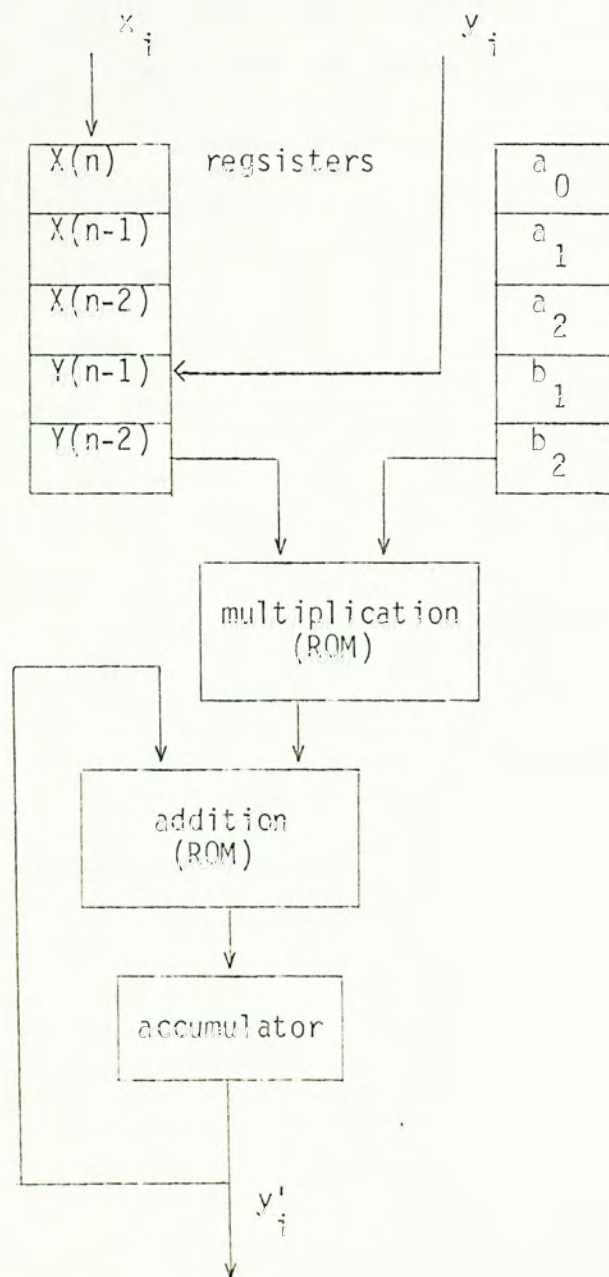


Fig. 4.12 Structure of "modular arithmetic"

With preference of small sizes, we assign $m_1 = 11$, $m_2 = 13$. Each of them would consequently occupy 4 bits. To

preserve two significant digits after decimal point of filter's coefficients, let the scale factor be 128. The value may be varied subject to the precision required. After determining the scale factor, we have $m_A = 128$. Although the optimal value of the redundant modulus is 2, it is not applicable to this system. To satisfy the condition of relatively prime, the redundant modulus will be 3 which is the smallest number next to the optimal one.

Based on the given numerical values, the dynamic range provided by the system is $[-71,71]$. For clarity, the parameters concerned with the polarity shift and scaling are listed below.

	value	residue digits
constant for polarity shift	71	{ 2,5,6 }
offset for scaling	64	{ 1,9,12 }

A program coded by APL language is written to simulate the residue system. Although many operations can be implemented by ROMs, no table is created during the simulation. The operations are directly simulated by their arithmetic expressions. In order to evaluate the performance of the residue system, a second order IIR filter having the difference equation shown in eqn.(5) is implemented. Fig. 4.13-4.14 show respectively the step and

impulse responses of the filter. The corresponding responses evaluated by infinite precision arithmetic are also present to illustrate the differences. To demonstrate the overflow suppression, the magnitude of input function is increased. Fig. 4.15 depicts the result. It is seen that by setting the input to zero, the oscillation will gradually die down.

Deviations, though not great, are observed between the results calculated by residue arithmetic and those by infinite precision arithmetic. The accuracy is mainly affected by two factors, quantization of filter's coefficients and truncation during scaling. A large scale factor can reduce the effect of the former one. For example, in the above illustration, the coefficient set used by infinite precision arithmetic is $\{0.0675, 0.1349, 0.0675, 1.143, -0.4128\}$. After the quantization (multiplied by the scale factor, round off and divided by the scale factor), the set becomes $\{0.0703, 0.1328, 0.0703, 1.1406, -0.414\}$ which are the actual parameters supplied for the RNS. Clearly, differences are found between the two coefficient sets which in turn affect the filter's responses.

sequence (n)	step input $X(n)$	residue arithmetic $Y_r(n)$	infinite precision arithmetic $Y_i(n)$	error $Y_r(n) - Y_i(n)$
1	50	4	3.375	0.625
2	50	15	13.98	1.022
3	50	29	23.08	0.9218
4	50	41	39.82	1.182
5	50	48	47.42	0.5832
6	50	51	51.26	-0.2553
7	50	52	52.51	-0.5062
8	50	52	52.35	-0.3514
9	50	51	51.66	-0.6581
10	50	50	50.93	-0.9295
11	50	50	50.38	-0.383
12	50	50	50.06	-0.05906
13	50	50	49.91	0.0856
14	50	50	49.88	0.1172
15	50	50	49.91	0.09365
16	50	50	49.95	0.05365
17	50	50	49.98	0.01766
18	50	50	50.01	-0.006957
19	50	50	50.02	-0.02024
20	50	50	50.03	-0.02527
21	50	50	50.03	-0.02552
22	50	50	50.02	-0.02374
23	50	50	50.02	-0.0216
24	50	50	50.02	-0.01989
25	50	50	50.02	-0.01882

Fig. 4.13 Step response evaluated by residue arithmetic.

sequence (n)	impulse input $X(n)$	residue arithmetic $Y_r(n)$	infinite precision arithmetic $Y_i(n)$	error $Y_r(n) - Y_i(n)$
1	50	4	3.375	0.625
2	0	11	10.6	0.3974
3	0	14	14.1	-0.1006
4	0	11	11.74	-0.7402
5	0	7	7.598	-0.5983
6	0	3	3.839	-0.8385
7	0	1	1.251	-0.2509
8	0	0	-0.1548	0.1548
9	0	0	-0.6933	0.6933
10	0	0	-0.7285	0.7285
11	0	0	-0.5465	0.5465
12	0	0	-0.3239	0.3239
13	0	0	-0.1447	0.1447
14	0	0	-0.03162	0.03162
15	0	0	0.02357	-0.02357
16	0	0	0.04	-0.04
17	0	0	0.03599	-0.03599
18	0	0	0.02462	-0.02462
19	0	0	0.01329	-0.01329
20	0	0	0.005023	-0.005023
21	0	0	0.0002567	-0.0002567
22	0	0	-0.00178	0.00178
23	0	0	-0.002141	0.002141
24	0	0	-0.001712	0.001712
25	0	0	-0.001073	0.001073

Fig. 4.14 Impulse response evaluated by residue arithmetic.

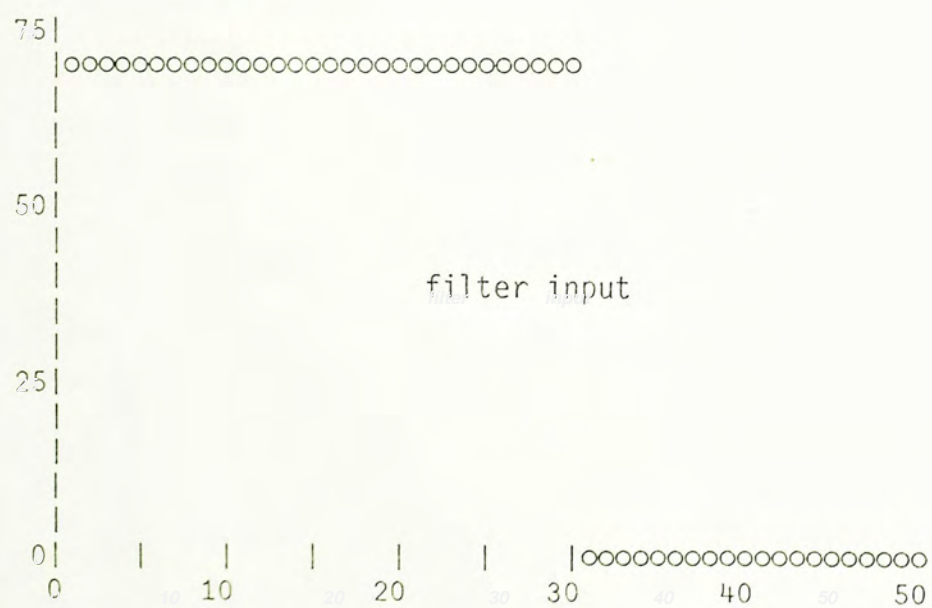
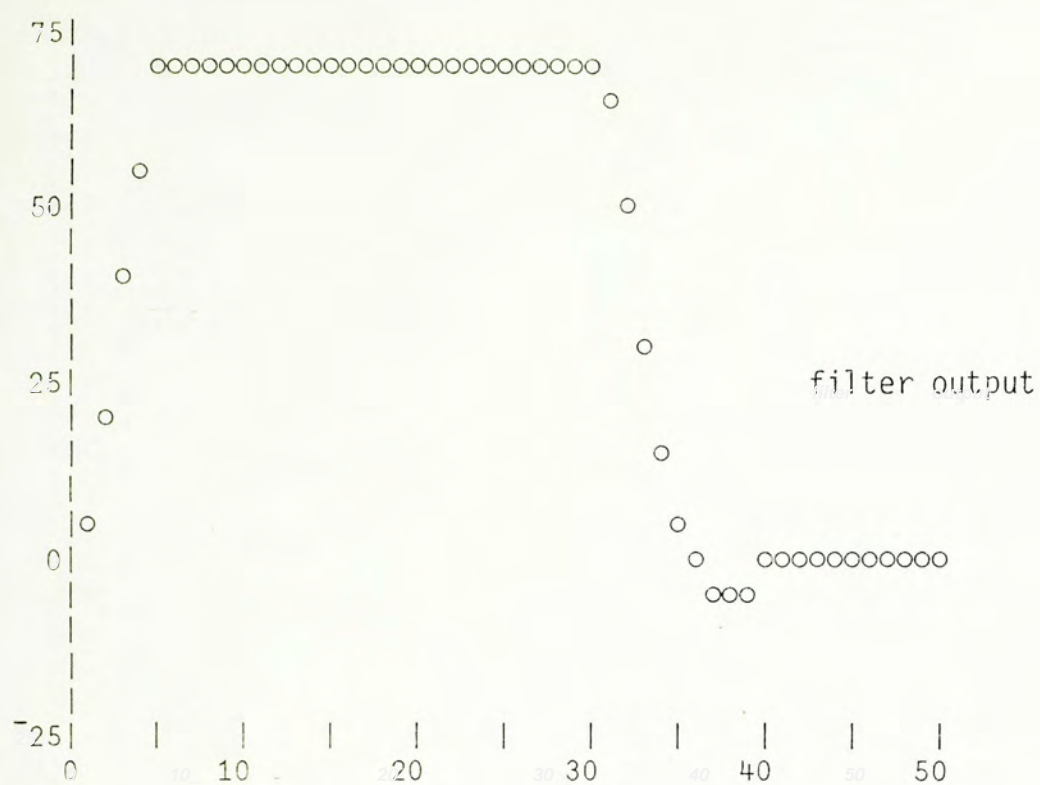


Fig. 4.15 Step response with overflow suppression.

5. Discussion

Large scale factor is usually required to reduce the quantization error of filter coefficients. Typical word length of scale factor is around 8 bits. Previous sections assume that one modulus in a residue number system is equal to the scale factor. If so, look-up tables are no longer applicable to implement the corresponding residue arithmetic as the table size is quite large. However, being treated as a system, large modulus (or scale factor) can be further decomposed into small constituents, each of which are then considered as a small modulus. For instance, the following figure shows a scale factor (A) breaking down into two subsystems, A1 and A2. The decomposition requires an additional residue decoder to recover the value y'_A as indicated in Fig. 5.1. However, if we combine the 'encoding' and 'residue decoding' operations, that is, both are realized by a single table with input $\{y'_{A1}, y'_{A2}\}$ and output $\{c_0, c_1, \dots, c_L\}$, then the decoder can be saved.

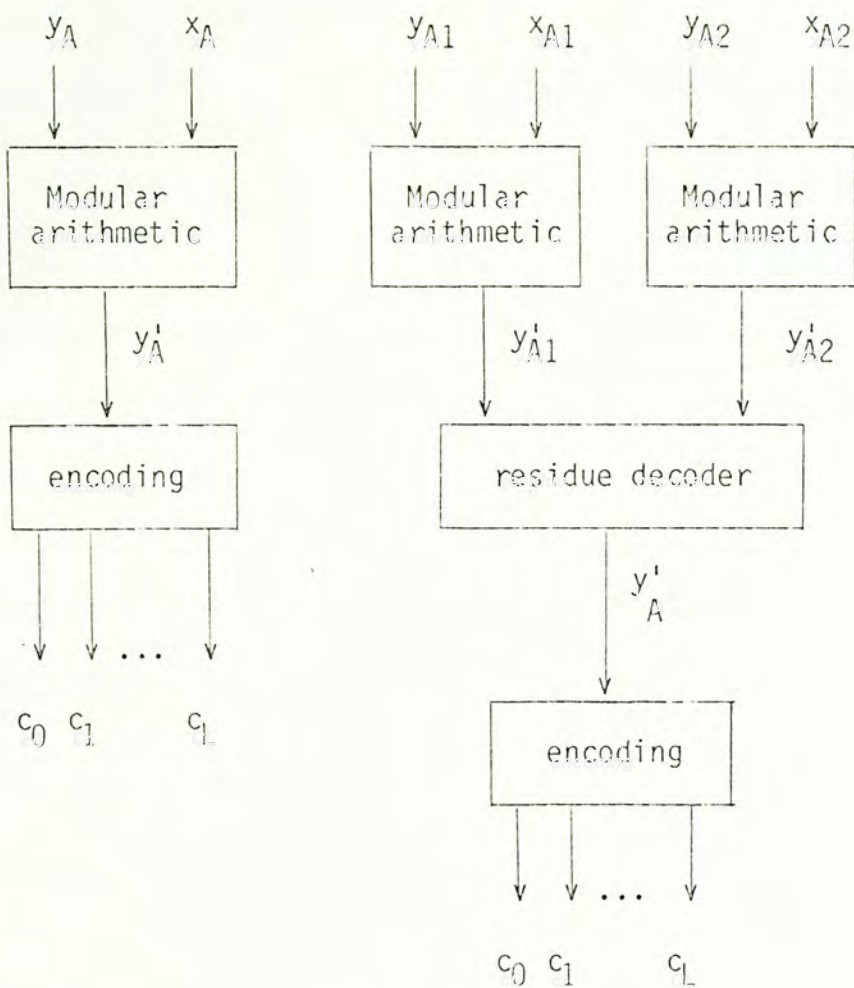


Fig. 5.1 Decomposition of a scale factor (A) into A1 and A2.

In order to satisfy the condition of relatively prime among the moduli, the scale factor must be an odd value if the redundant modulus is 2. Consequently, we have an odd scale factor (A) and an even scaled dynamic range (M). With similar polarity analysis as shown in section 4.1.3,

$$P = \text{INT} [((M \times A)/2 - 1 + K)/A] ,$$

$$N = \text{INT} [((M \times A)/2 + K)/A]$$

and

$$P \leq M/2 - 1 ,$$

$$N \geq M/2 .$$

By solving the inequality, the offset K has zero value. That means, direct scaling of an arbitrary signed number for this case would not cause polarity change.

Based on the above discussions, we suggest another residue number system which consists of the moduli {2,13,15,7,31}. They are, respectively,

redundant modulus - 2 ;

fundamental moduli - 13,15 ;

and constituents of scale factor - 7,31 .

(i.e. scale factor = 217)

The system has three advantages: it does not require offset operation, eliminates the use of large modulus and is possible to adopt the optimal redundant modulus ($m_0=2$).

6. Conclusion

New approaches to scaling and overflow detection in residue number system have been presented for use in recursive digital filtering. Under the assumption that the scale factor is equal to one modulus or a product of several moduli, scaled residue digits can be produced before performing residue decoding. By adding an appropriate offset value during scaling, the polarity ambiguity is completely eliminated. As scaling is carried out separately from residue decoding, less residue digits are input to the decoder, which makes it feasible to be realized by look-up tables.

Based on redundancy technique, a scheme is devised to detect residue arithmetic overflow. Redundancy is established by inserting a redundant modulus to the fundamental set of moduli of a given residue number system. A polarity shift operation, which is actually a modular addition, is required to accomplish the detection. It is shown that a redundant modulus of value 2 has the simplest hardware structure and is recommended for use. As the overflow characteristic of residue arithmetic is similar to 2's complement arithmetic, self-oscillations exist for residue implementation. The problem is solved by replacing

overflow results with their complemented values.

A 2nd order IIR filter is implemented based on the above approaches. In comparison with those calculated from infinite precision arithmetic, small deviations are observed.

References

- [1] P.J. Allen and A.G.J. Holt, "Implementation of a 2nd-order digital filter using AM2900 bit-slice devices and a fast multiplier", IEE Proc., vol.128, Pt.G, No.4, pp.216-219, August 1981.
- [2] R.J. Simpson and T.J. Terrell, "Microprocessor-based digital filters", The Radio and Electronic Engineer, vol.51, No.9, pp.423-428, September 1981.
- [3] J.G.M. Goncalves and R.M. Lea, "LSI module for the implementation of digital filters", IEE Proc., vol.128, Pt.F, No.6, pp.353-358, November 1981.
- [4] L.C. Tai, "Design of a microprocessor-based digital filter", A progress report presented to the Croucher Foundation, 1982.
- [5] A. Peled and B. Liu, "A new hardware realization of digital filters", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-22, No.6, pp.456-462, December 1974.
- [6] McClellan and Rader, "Number theory in digital signal processing", Prentice Hall.
- [7] W.K. Jenkins and B.J. Leon, "The use of residue number systems in the design of finite impulse response digital filters", IEEE Transactions on Circuits and Systems, vol. CAS-24, No.4, pp.191-201, April 1977.
- [8] W.K. Jenkins, "Recent advances in residue number techniques for recursive digital filtering", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-27, No.1, pp.19-30, Feb 1979.

- [9] 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, vol. ASSP-28, No.5, pp.538-544, October 1980.
- [10] M.H. Etzel and W.K. Jenkins, "The design of specialized residue classes for efficient recursive digital filter realization", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-30, NO.3, pp.370-380, June 1982.
- [11] P.M. Ebert, J.E. Mazo and M.G. Taylor, "Overflow oscillations in digital filters", Bell Syst. Tech. J., vol.48, pp.2999-3020, Nov. 1969.
- [12] S.L. Freeny, "Special-purpose hardware for digital filtering", Proceedings of the IEEE, vol.63, No.4, pp.633-648, April 1975.
- [13] D.K. Banerji and J.A. Brzozowski, "Sign detection in residue number systems", IEEE Transactions on Computers, vol. C-13, No.4, pp.313-320, April 1969.
- [14] W.K. Jenkins, "Techniques for residue-to-analog conversion for residue-encoded digital filters", IEEE Transactions on Circuits and Systems, vol. CAS-25, No.7, pp.555-562, July 1978.
- [15] A. Baraniecka and G.A. Jullien, "On decoding techniques for residue number system realizations of digital signal processing hardware", IEEE Transactions on Circuits and Systems, vol. CAS-25, No.11, pp.935-936, November 1978.
- [16] M.A. Soderstrand and C. Vernia, "A high-speed low-cost modulo P_i multiplier with RNS arithmetic applications", Proceedings of the IEEE, vol.68, No.4, pp.529-532, April 1980.

- [17] C.H. Huang and F.J. Taylor, "A memory compression scheme for modular arithmetic", IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-27, No.6, pp.608-611, December 1979.
- [18] C.T. Chen, "One-dimensional digital signal processing", New York, M. Dekker, 1979.
- [19] C.F. Chen and Y.T. Tsay, "A new formula for the discrete time system stability test", Proceedings of IEEE, pp.1200-1202, August 1977.

Appendix - A

APL programs to simulate the proposed
residue number system for recursive filtering

```

      ▽ SORNS;I;M;NCOEF;HM
[1]  P A 2ND ORDER IIR FILTER IMPLEMENTED
[2]  P BY RESIDUE NUMBER SYSTEM
[3]  P TO SHOW THE SPECIFIC SCALING
[4]  P AND OVERFLOW DETECTION
[5]  P
[6]  P MOD ← SET OF MODULI
[7]  P SFACTOR ← SCALE FACTOR
[8]  P ISF ← MULTIPLICATIVE INVERSE OF SFACTOR
[9]  P DECODE ← PARAMETERS IN DECODING FORMULA
[10] P CO ← FILTER COEFFICIENTS IN RESIDUE CODES
[11] P REG ← REGISTERS CONTAIN X0,X1,X2,Y1,Y2
[12] P HM ← HALF OF DYNAMIC RANGE (M)
[13] P HA ← HALF OF SCALE FACTOR
[14] MOD←(3,11,13,128)
[15] NMOD←ρMOD
[16] BASE←(2,3)
[17] M←*/MOD[BASE]
[18] SFACTOR←MOD[NMOD]
[19] ISF←(2,3,6)
[20] DECODE←(78,66)
[21] NCOEF←ρCOEF
[22] REG←(NMOD,NCOEF)ρ0
[23] YR←NMODρ0
[24] ZR←C←(NMOD-1)ρ0
[25] HM←[M:2
[26] HA←CO←PS←10
[27] P MAGNIFY, QUANTIZE AND ENCODE COEFFICIENTS
[28] I←1
[29] LOOP1:CO←CO,[0.5+MOD[I]]|SFACTOR*COEF
[30] HA←HA,MOD[I]|SFACTOR÷2
[31] PS←PS,MOD[I]|HM
[32] →(NMOD≥I←I+1)/LOOP1
[33] CO←(NMOD,NCOEF)ρCO
[34] P
[35] P BEGIN OF FILTERING
[36] P XIN, YY ← INPUT AND OUTPUT SAMPLES
[37] J←1
[38] P MODULAR ARITHMETIC
[39] START:MODA
[40] P SCALING OPERATION
[41] SCALE
[42] P OVERFLOW DETECTION
[43] OVDET
[44] YR[NMOD]←MOD[NMOD]|YY[J]←YM
[45] →(N≥J←J+1)/START
      ▽

```

```

      ▽ MOD4; I
[1]  R RIGHT SHIFT REG-(X0,X1,X2,Y1,Y2)
[2]  R AND EVALUATE THE DIFFERENCE EQN.
[3]  I←1
[4]  LOOP: REG[I;]←(MOD[I]|XIN[J]),4+REG[I;]
[5]  REG[I;4]←YR[I]
[6]  YR[I]←MOD[I]|+/CO[I;]×REG[I;]
[7]  →(NMOD≥I←I+1)/LOOP
      ▽

```

```

      ▽ SCALE; I
[1]  R ADD HALF OF THE SCALE FACTOR
[2]  I←1
[3]  LOOP1: YR[I]←MOD[I]|YR[I]+HA[I]
[4]  →(NMOD≥I←I+1)/LOOP1
[5]  R SCALE DOWN THE OUTPUT RESULTS
[6]  I←1
[7]  LOOP2: C[I]←MOD[I]|-YR[NMOD]
[8]  YR[I]←MOD[I]|ISF[I]×(YR[I]+C[I])
[9]  →(NMOD>I←I+1)/LOOP2
      ▽

```

```

      ▽ OVDET; I
[1]  I←1
[2]  LOOP1: ZR[I]←MOD[I]|YR[I]+PS[I]
[3]  →(NMOD>I←I+1)/LOOP1
[4]  ZM←M|+/DECODE×ZR[BASE]
[5]  YM←ZM-HM
[6]  →(ZR[1]=MOD[1]|ZM)/O
[7]  I←2
[8]  LOOP2: YR[I]←MOD[I]-YR[I]
[9]  →(NMOD>I←I+1)/LOOP2
[10]  YM←-YM
[11]  YR[1]←MOD[1]|YM
      ▽

```

Appendix - B

Hardware implementation of 2nd order IIR
filters using APPLE II micro-computer

Abstract - A project summarized from the report
"Design of a microprocessor-based digital filter"
is presented to illustrate the hardware aspect as
well as the distributed arithmetic of digital filters.

Introduction - Second order IIR filters are implemented according to the algorithm presented by Peled and Liu. The method utilizes distributed arithmetic technique to carry out the multiplications as required in a difference equation. In order to reduce the hardware complexity for the realization, a micro-computer having enough supporting tools such as program debugging routines, peripheral interfaces is chosen. The one we adopted is APPLE II micro-computer.

Distributed arithmetic - Based on the algorithm, filter coefficients as well as input and output samples are represented by fixed point notation. Since the processing unit is byte oriented, an 8-bit word length is used for the digital filter. The binary form of a value X will be written as $X^0.X^1X^2X^3X^4X^5X^6X^7$. i.e.

$$X = \sum_{j=1}^7 X^j 2^{-j} - X^0, \text{ where } X^j = 0 \text{ or } 1. \quad (1)$$

From (1), it is known that $-1 \leq X < 1$. By substituting (1) into the standard difference equation of a 2nd order IIR filter, we get

$$Y(n) = \sum_{i=0}^2 a_i \left(\sum_{j=1}^7 X^j(n-i) 2^{-j} - X^0(n-i) \right) + \sum_{i=1}^2 b_i \left(\sum_{j=1}^7 Y^j(n-i) 2^{-j} - Y^0(n-i) \right)$$

$$= \sum_{j=1}^7 F^j 2^{-j} - F^0 \quad (2)$$

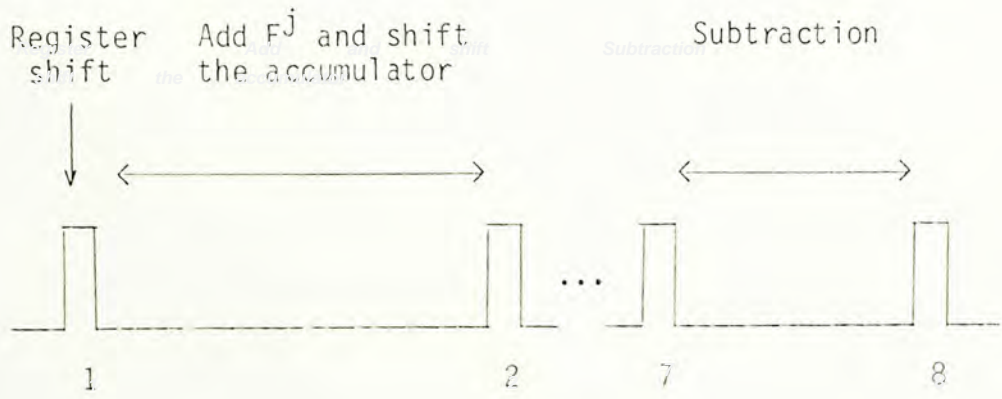
$$\text{where } F^j = \sum_{i=0}^2 a_i X^j(n-i) + \sum_{i=1}^2 b_i Y^j(n-i)$$

Given a set of coefficients, F^j can be pre-calculated and stored in a table for further accessing. The value is addressed using the following 5 bits, $X^j(n) X^j(n-1) X^j(n-2) Y^j(n-1) Y^j(n-2)$. From (2), multiplications are consequently replaced by bit shift and add operations.

Hardware realization - Fig. 1 illustrates the block diagram of the digital filter. The corresponding hardware circuit and timing diagram are respectively depicted in Fig. 2 and Fig. 3. From Fig. 1, five registers are used to store the input and output samples. The table is realized by read-only-memory (ROM). Real-time signals can be fetched in or sent out through the A/D and D/A converters. All control signals such as shift, reset are generated by the CPU through the peripheral interface device. Except the data acquisition module, all components are assembled on one circuit board to give neat appearance.

Circuit description - Quantized input sample produced by the analog-to-digital converter is stored in the X_n

register. At initial phase, a reset procedure is necessary for all registers so that they hold zero values. An I/O port namely PIA-6821 acts as an interface between the CPU and the registers. Each time a shift command is issued, the table is addressed and the output is fetched by the CPU for add and shift operations. After repeating the same procedures for seven times, we reach the final step which requires a subtraction. If the calculation signals overflow, then its result will be complemented, otherwise the result is directly passed to the Y_{n-1} register through the I/O port and at the same time, accessed by the digital-to-analog converter. Both input and output samples are displayed on the screen of an oscilloscope for examining. The original table requires 32 words but a 2K ROM is used, that means other types of 2nd order filters can also be accommodated until the whole ROM is completely occupied. Different types of filters are designed to be selected by a switch-band.



One phase for evaluating an output sample

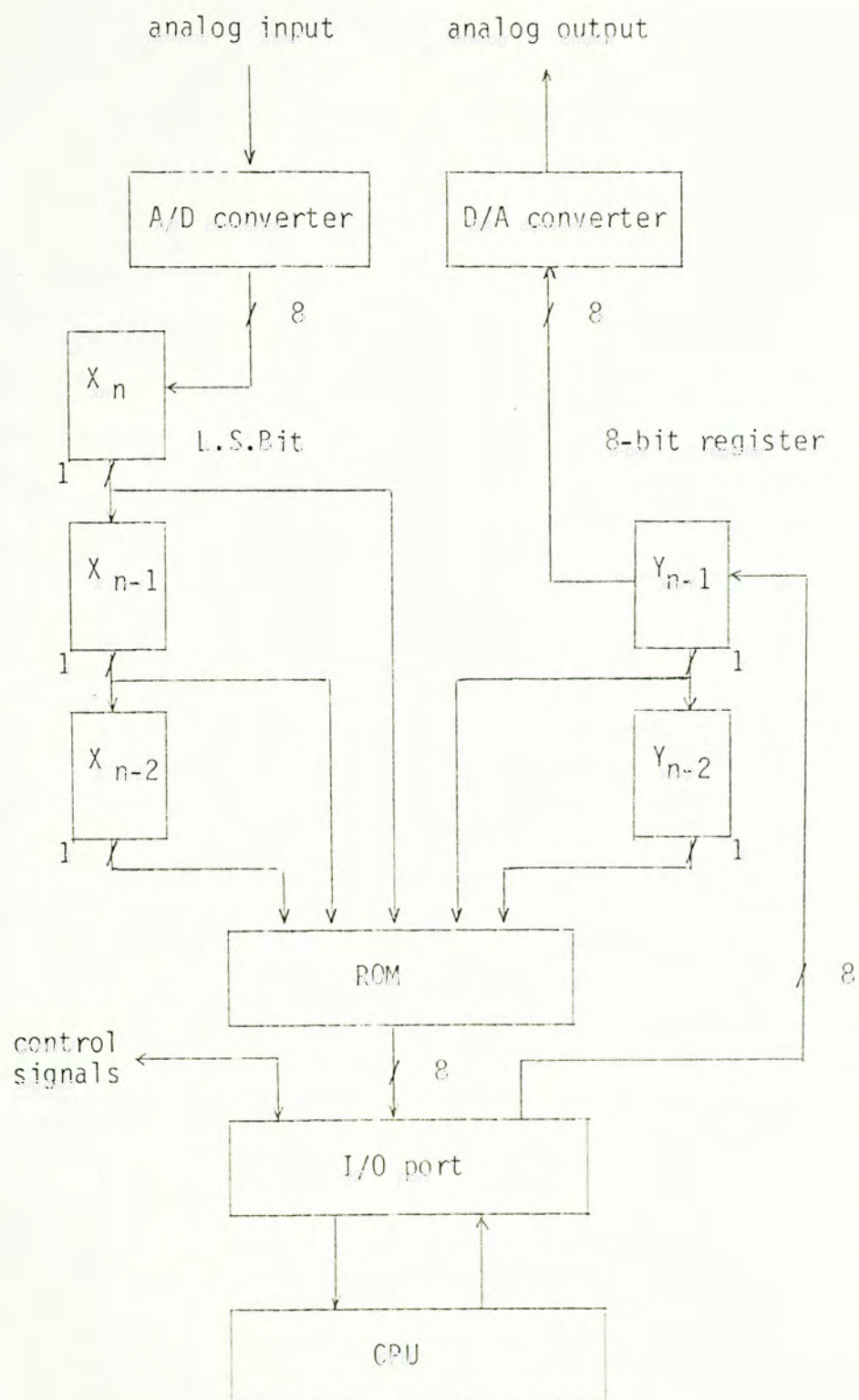


Fig. 1 Block diagram of the hardwired digital filter.

CB2 - PE



CA2



LOAD



SHIFT



STC

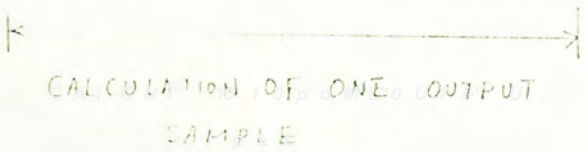


Fig. 3 Control signals.

Results - Two 2nd order IIR filters having transfer functions shown in the following are implemented. The tables calculated from the corresponding coefficient sets are given in Fig. 4.

- (1) Low-pass filter, cut-off frequency 100 Hz,
sampling frequency 1 KHz.

$$H_1(z) = \frac{0.0675 + 0.1349 z^{-1} + 0.0675 z^{-2}}{1 - 1.143 z^{-1} + 0.4128 z^{-2}}$$

- (2) Band-pass filter, lower and higher cutoff
frequencies 9.5 Hz, 10.5 Hz, sampling
frequency 100 Hz.

$$H_2(z) = \frac{0.1 - 0.1 z^{-2}}{1 - 1.5695 z^{-1} + 0.9391 z^{-2}}$$

By applying different inputs to the filters, we get the responses as illustrated in Fig. 5 and Fig. 6. The time required for evaluating one output sample is approximately 0.36 ms so it gives a sampling frequency of 2.7 KHz. It is

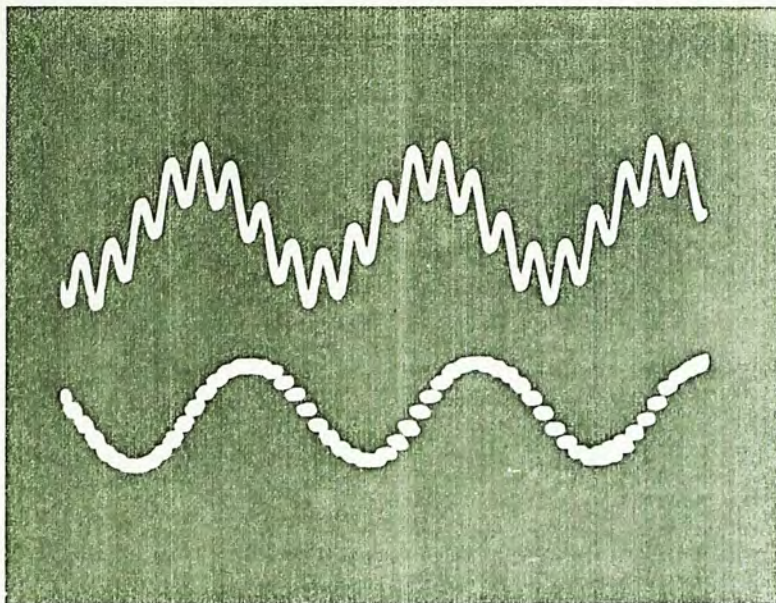
noted that the sampling frequency does not match with the two filters. The modification of sampling frequency, however, will affect the cutoff frequency as their ratio is fixed. Consequently, for $H_1(z)$, 100 Hz is changed to 270 Hz; and for $H_2(z)$, 9.5 Hz and 10.5 Hz are respectively changed to 256 Hz and 283 Hz.

From Fig. 5, the low-pass filter successfully retains the 120 Hz signal component and rejects the other component, 1 KHz which is out of the cutoff frequency.

For the band-pass filter, a square-wave is used for the testing. An output of sinusoidal waveform is obtained as illustrated in Fig. 6. The result can be explained to be the fundamental frequency (270 Hz) of the square-wave, which lies in the pass-band.

Address	Contents	
	$H_1(z)$	$H_2(z)$
00	00	00
01	E6	C4
02	49	64
03	2E	28
04	04	FA
05	EA	BE
06	4D	5E
07	33	21
08	08	00
09	EF	C4
0A	51	64
0B	37	28
0C	0C	FA
0D	F3	BE
0E	56	5E
0F	3B	21
10	04	06
11	EA	CB
12	4D	6A
13	33	2E
14	08	00
15	EF	C4
16	51	64
17	37	28
18	0C	06
19	F3	CB
1A	56	6A
1B	3B	2E
1C	11	00
1D	F7	C4
1E	5A	64
1F	40	28

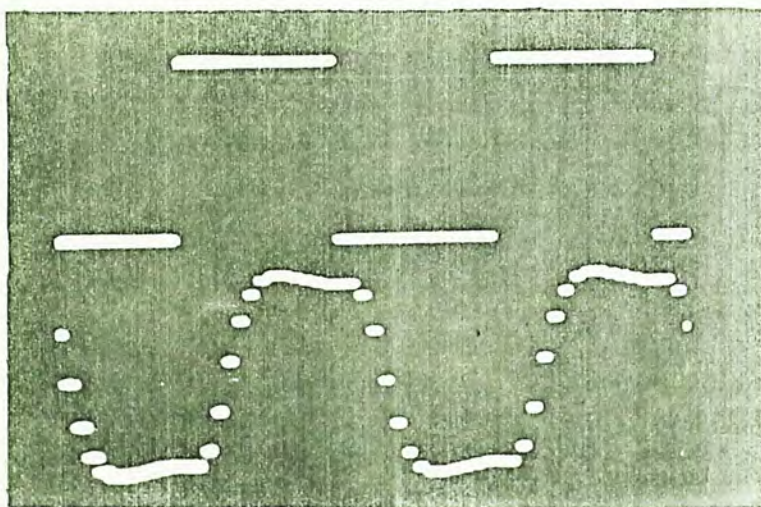
Fig. 4 Tables for the filters.



upper: input signal - $V \sin(2\pi f_1 t) + V \sin(2\pi f_2 t)$,

where $f_1 = 1 \text{ KHz}$, $f_2 = 120 \text{ Hz}$.

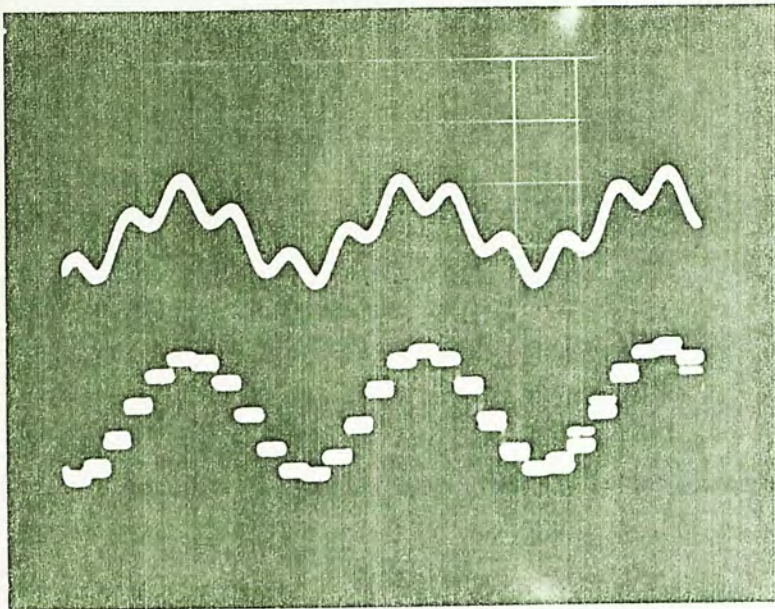
lower: output of low-pass filter $H_1(z)$.



upper: input signal - 100 Hz square wave.

lower: output of low-pass filter $H_1(z)$.

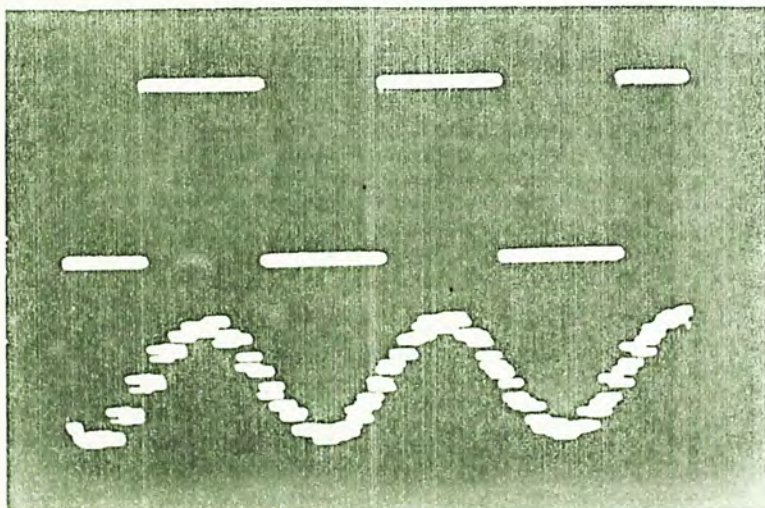
Fig. 5 Responses of $H_1(z)$.



upper: input signal - $V \sin(2\pi f_1 t) + V \sin(2\pi f_2 t)$,

where $f_1 = 1.2 \text{ KHz}$, $f_2 = 270 \text{ Hz}$.

lower: output of bandpass filter $H_2(z)$.



upper: input signal - 270 Hz square wave.

lower: output of bandpass filter $H_2(z)$.

Fig. 6 Responses of $H_2(z)$.

Discussion and conclusion - The assembly program for initialization and digital filtering has been stored in an EPROM located on the I/O interface card. With sophisticated design, the card can be plugged into any seven of the eight peripheral slots provided by the APPLE II micro-computer. To start the filtering in real-time, key in the command "PR#n" where n is the slot number.

Based on distributed arithmetic technique, two 2nd order IIR digital filters have been implemented. The method is suitable for those systems using general purpose processor which lacks multiplication instructions. A sampling frequency of 2.7 KHz is achieved for the implementation using APPLE II micro-computer. Real-time results are obtained which show both filters work correctly.



000443759