CHAPTER I
1. INTRODUCTION

1.1 FIXED POINT ARITHMETIC

In general, an n-bit fixed-point binary number can be expressed as

\[ N = \sum_{i=-\infty}^{\infty} b_i 2^i = b_{n-1} 2^{n-1} + b_{n-2} 2^{n-2} + \ldots + b_1 2^1 + b_0 2^0 + b_{-1} 2^{-1} + \ldots + b_{-m} 2^{-m} \]

\[ = (b_{n-1}\ldots b_0.b_{-1}b_{-2}\ldots b_{-m})_2 \]  \hspace{1cm} \ldots \ldots \ldots \ldots \ldots (1)

where \( b_i \) is either zero or one.

The bit \( b_{n-1} \) is termed as the most significant bit (MSB) and \( b_{-m} \) is termed as least significant bit (LSB). The integer portion of the number, \( b_n b_{n-1} \ldots b_0 \), is separated from the fractional portion, \( b_{-1} b_{-2} \ldots b_{-m} \), by the binary point or radix point.

In the binary representation, each bit can be either a zero or a one. In fixed point arithmetic, numbers are always normalized as binary fractions of the form \( b_0.b_1 b_2 \ldots b_c \) where \( b_0 \) is the sign bit. The C+1 bit normalized number is stored in a register as shown in fig. (1), where the sign is C-bit number by a fictitious binary point. The binary point is fictitious because it does not occupy any bit location in the register. The word length \( C \), is defined as the number of bit locations in the register to the right of the binary point.

There are three commonly used methods for representing signed numbers:

a. signed-magnitude

b. two's complement

c. one's complement

Let us consider the C+1 bit binary fraction \( b_0.b_1 b_2 \ldots b_c \) where \( b_0 \) is the sign bit. In the sign magnitude representation, the fractional number is positive if \( b_0 \) is zero, and it is negative if \( b_0 \) is one.

In the sign magnitude representation, binary numbers can be converted to decimal numbers using the relation.

\[ N = (-1)^{b_0} \sum_{i=1}^{C} b_i 2^{-i} \]  \hspace{1cm} \ldots \ldots \ldots \ldots \ldots (2)
The two’s complement representation of positive numbers is the same as sign magnitude representation. The two’s complement representations of negative numbers, however, are obtained by complementing all the bits of positive numbers and adding one to the LSB of the complement number. A decimal number can be recovered from its two’s complement representation using the relationship

\[ N = -b_0 + \sum_{i=1}^{C} b_i 2^{-i} \]......(3)

The one’s complement representation of fractional numbers is the same as the two’s complement without the addition of one to the LSB. The one’s complement representation can be shown via the relationship

\[ N = b_0 (2^C - 1) + \sum_{i=1}^{C} b_i 2^{-i} \]......(4)

The two’s complement representation of binary numbers has several advantages over the signed magnitude and one’s complement representations [1] and therefore is more popular. In general, the sum of two normalized C-bit numbers using fixed point arithmetic is a C-bit number while the product of two C-bit numbers is a 2C-bit number. Hence, if the register word length is fixed to C-bits, a quantization error will be introduced in multiplication but not in addition. The product is quantized either by rounding or by truncation.

The one’s complement representation is less efficient than the two’s complement because it requires the addition of the overflow carry to the result (end-around carry). In addition, in two’s complement representation intermediate overflows during the execution of arithmetic operations do not cause an error when the final result is within range, and zero has a unique representation. So the two’s complement representation is widely used in computer arithmetic. However, the addition of overflow carry complicates multiplier implementation. Booth’s algorithm (Booth 1951) [2] is used for software implementation, requiring considerably more steps than the algorithm for the multiplication of positive numbers. On the other hand, the hardware implementation of two’s complement multipliers does not yield simple modular circuit according to authors De Mori and Sera 1972 [3], Gipson and Gibbard 1975 [4], and
Pekmestzi and Papadopoulos 1981 [5]. Besides the designs are not expandable in modular fashion, except those structures that include programmable cells [6]. This results from the inherent asymmetry of the two’s complement representation.

### 1.2 FLOATING-POINT ARITHMETIC

A major disadvantage of fixed-point arithmetic is the limited range of numbers that can be represented with a given word length. Another type of arithmetic for which the same number of bits, has a much larger range of numbers is Floating-point arithmetic. In general, a Floating-point number can be expressed as

\[ N = M \times 2^E \]

\[ \text{(5)} \]

Where \( M \) and \( E \), both expressed in binary form, are termed as the mantissa and the exponent of the number respectively. In binary floating-point representation, numbers are always normalized by scaling \( M \) as a fraction whose decimal value lies in the range \( 0.5 \leq M < 1 \). Fig. (2) shows a floating point number stored in a register. The register is divided into the mantissa and the exponent. Both the mantissa and the exponent have fictitious binary points to separate the sign bit from the numbers. In floating point arithmetic, negative mantissas and negative exponents are coded in the same way as in fixed point arithmetic using two’s complement signed magnitude or one’s complement. The product of two floating point numbers is

\[ (M_1 \times 2^{E_1}) \times (M_2 \times 2^{E_2}) = (M_1 \times M_2) \times 2^{(E_1 + E_2)} \]

\[ \text{(6)} \]

Thus if mantissa is limited to \( C \) bits the product \( (M_1 \times M_2) \) must be rounded or truncated to \( C \) bits. The sum of two floating point numbers is performed by shifting the bits of the mantissa of the smaller number to the right and increasing the exponent until the two components are equal. Then the two mantissas are added and if necessary, normalized to satisfy eq. (5). The shifted mantissa may exceed its limited range and thus must be quantized.
1.3 DIFFERENT REPRESENTATIONS OF BINARY SYSTEM

1.3.1. NEGABINARY SYSTEM

Negabinary stands for negative binary. This is a binary system having base (-2). The decimal number in this system is given by the equation

\[ x_{-2} = \sum_{k=1}^{n} x_k (-2)^k \]  

(7)

Where \( x_k \) is either 0 or 1. It contains no special sign digit. The sign is inherent in all digits. The possibility of representing numbers in base -2 was suggested some years ago [7,8], and a number of machines operating in this mode have been constructed [9,10]. The advantages of negative base arithmetic are extensively discussed in [11].

The algorithm prescribed for addition of two numbers represented by

\[ a = \sum_{i=0}^{k} a_i (-2)^i \] and \[ b = \sum_{i=0}^{k} b_i (-2)^i \] is as follows.

The sum bit \[ S_i = (a_i + b_i + c_i + d_i) \mod 2 \]  

(8)

and twin-carry bits are

\[
\begin{align*}
    c_{i+1} &= 1 \\
    d_{i+2} &= 1
\end{align*}
\]

when \( a_i + b_i \geq 2 \)  

(9)

\[
\begin{align*}
    c_{i+1} &= 0 \\
    d_{i+2} &= 0
\end{align*}
\]

when \( a_i + b_i < 2 \)  

(10)

The twin-carry bits \( c_{i+1}, d_{i+2} \) arise, since alternate positions in negative base number are negatively weighted. It is easy to see that shifting \( x_{-2} \) left 1 bit amounts to multiply it by \(-2\). It will then require an addition to the unshifted \( x_{-2} \) to produce \(-x_{-2}\). Because it is possible to use addition to define negation and hence subtraction, the above authors have suggested that one does not need subtraction at all. However, since we have seen that it requires little more hardware to build add- subtract unit over an add only unit, it is worthwhile to do so and speed up subtraction by eliminating a shift and an add. One can produce the negative of a number by subtracting it from 0, reducing negation to one operation rather than two. The negation of number can be obtained by adding it
to itself first and then shift right, since a right shift corresponds to dividing by -2. As shown by Zohar and others [12-14], the use of base -2 arithmetic permits the construction of radically new hardware for special purposes. This comes about because of the absence of apparent difference between positive and negative numbers represented using this base. In general circumstances, however, it is at present difficult to see base-2 as a strong competitor against positive base arithmetic. Arithmetic operations are usually more complex, and fast adders are difficult to design. The range of base-2 is not symmetrical: either there are twice as many positive numbers as negative ones, or vice-versa. One might also note that base-2 D/A conversion involve subtraction of reference voltages, which could magnify relative error because of cancellation. It might thus be reasonable to expect that base-2 will be used only in special circumstances while base +2 systems remain predominant. If this becomes true, then it is important that interfacing between the two kinds of systems should be simple so that we could use each type to do so the part of job for which it is best suited.

1.3.2. MIXED BINARY REPRESENTATION

This system is proposed by Yuen [15]. In this system the positive as well as negative base i.e. +2 and -2 have been used to represent decimal number. This is suitable for BCD numbers. In BCD number each decimal digit requires four bits for unique coding. Let us call these bits u, x, y and z. The arithmetic value of each decimal digit is given by

\[ D = (8z - 4y - 2x + u) \]

The largest value a digit can take is 9 and the smallest is -6. From speed and space point of view this system is useful in BCD arithmetic numbers. Given a 4-bit pattern the use of eqn. (11) immediately yields the decimal value. Let us convert (11) into the form

\[ D = z (2) (-2) + y (2) (-2) + x (-2) + u \]

We divide D by -2. The quotient is \( D' = z (2) (-2) + y (2) + x \) and the remainder is just u. Now, we divide D' by 2 to give quotient.

\[ D'' = z (-2) + y \]
and remainder x. Finally, dividing $D''$ by -2 again produces remainder y and quotient z. In other words, the bit pattern is produced by repeated division as in any other number representation. For example, take $D = 6$. The following results ensure:

$D' = -3, \quad u = 0$;

$D'' = -2, \quad x = 1$;

$z = 1, \quad y = 0$.

Thus, 6 has the representation 1010 as required.

1.3.3. RESIDUE NUMBER SYSTEM

A brief review of Residue number system (RNS) theory is represented as

Let $P = (p_1, p_2, p_3, \ldots, p_L)$ be a set of relatively prime integers (moduli), and let $x$ be any integer in $[0, M-1]$, where $M = \prod_{i=1}^{L} p_i$, then by the Euclidean algorithm, there exists integers $k_i, x_i$ such that

$$x = k_1 p_1 + x_1, \quad i = 1, 2, \ldots, L$$

\[\text{(12)}\]

The quantity $x_i$ is called the $i$th residue of $x$ and is usually denoted as $(x)p_i$ or $x \mod p_i$. If $x \in [0,M-1]$, $x$ be uniquely determined by $L$-tuple $(x_1, x_2, \ldots, x_L)$. In this case $x \equiv (x_1, x_2, \ldots, x_L)$. Inversion of residue $L$-tuple can be performed through the use of so-called Chinese Remainder Theorem or the mixed radix conversion algorithm [16]. The number $x$ can likewise be mapped onto $[-(M-1)/2, (M-1)/2]$ by the isomorphism

$$Y = \begin{cases} -(M - x); & \text{if } x > (M - 1)/2 \\ x_1; & \text{if } x \leq (M - 1)/2 \end{cases} \quad \text{..............(13)}$$

The major attribute and most dramatic feature of RNS is the manner in which it performs fixed point arithmetic. Let $x, y \in Z_m$ and $x \equiv (x_1, x_2, \ldots, x_L), y \equiv (y_1, y_2, \ldots, y_1)$ then $z = x \circ y = (z_1, z_2, \ldots, z_L)$, where $z_i = (x_i \circ y_i) \mod p_i$, for $i = 1, 2, \ldots, L$ and ‘$\circ$’ denotes the operation ‘$\times$’, ‘$+$’ or ‘$-$’. It is clear that the sub-operation within each modulus is independent of the other. That is, no carry information need be passed between moduli. The absence of carry requirements means that the concept of most and least significant digits
is void. Thus parallel architecture can be designed to process all modular partial sum and product concurrently. This parallelism can be translated into a speed-up of arithmetic. The ability to realize fast arithmetic has been demonstrated by authors [17-22]. In addition, the arithmetic is exact and therefore free of round-off error. However, it is this exactness that is often considered an RNS debit.

In 2’s complement representation: care must be taken to prohibit register overflow. In a weighted numbering system this is usually accomplished through rounding. This is however a nontrivial problem in a residue based system since overflow detection is extremely difficult. Overflow in the RNS is promoted by scaling partial products and sums so that even under “worst case” conditions, system variables will not exceed their allotted dynamic range. Scaling, like division, is also difficult in RNS. Even with these difficulties, the potential of RNS is difficult to deny.

1.3.4. REDUNDANT BINARY REPRESENTATION

Redundant binary representation [23-25] contains digits of the three values 0, +1 and -1 where the arithmetic circuits use radix two. Redundant digit sets that are variants of the well known signed digit (SD) and carry save (CS) representation. These digits sets are defined as

\[ D^{(SD)} = \{-1, 0, +1\}, \quad D^{(CS2)} = \{0, 1, 2\}. \]

\[ D^{(SD\;\;1)} = \{-2, -1, 0, 1\}, \quad D^{(SD\;\;1)} = \{-1, 0, 1, 2\} \]

\[ D^{(CS\;\;3)} = \{0, 1, 2, 3\}. \]

There are two types of number systems based on each of these digit sets, a fully redundant system and a partially redundant system. A fully redundant number system is one in which all digit positions of a number are redundant. In a partially redundant number system, only some of digit positions are redundant. The positional radix-\(\beta\) number system represents an \(n\)-digit value \(V\) as a string of digits.

\[ (d_{n-1}, d_{n-2}, \ldots, d_0), \text{ where} \]

\[ \sum_{i=0}^{n-1} d_i \beta^i = V \]

\[ \sum_{i=0}^{n-1} d_i \beta^i = V \] \[ \sum_{i=0}^{n-1} d_i \beta^i = V \]
The value that each digit $d_i$ can assume is determined by the digit set for the position $D_i$, such that $d_i \in D_i$. In conventional representations, the digit set is same for all positions and is defined by $D = \{d \mid 0 \leq d \leq \beta - 1\}$. A number system is redundant if there is some value which does not have a unique representation. In other words a given number system is redundant if there exists an n-digit value $V$ which satisfies $V = \sum_{i=0}^{n-1} d_i \beta^i = \sum_{i=0}^{n-1} d'_i \beta^i$; where $d_i, d'_i \in D_i$ and there is at least one position $j$ where $d_j \neq d'_j$. This implies that for some digit position $k$, the cardinality of digit set $D_k$ satisfies $|D_k| > \beta$. We call such a position, a redundant digit position. In arithmetic process of redundant number system like multiplication and others, the conversion of redundant form into two’s complement form is needed. Conversion logic can be illustrated as

A carry signal used must be calculated from LSB to MSB recursively. The conversion logic equation is given by

$$C_{\text{out}+1} = C_{\text{in}} \overline{C_i} + s_i$$ ..........................(15)

The output carry signal $C_{\text{out}}$ will be set if the redundant binary bit $q_i = -1$ or if the input carry $C_{\text{in}}$ is +1 and the redundant binary digit $\neq +1$. This means that -1 initialises a carry signal and a +1 cancels a carry signal. The recursive equation, following eqn. (15), we have

$$C_{\text{out}+2} = s_{i+3} + \overline{V_{i+3}} (s_{i+2} + \overline{V_{i+2}} (s_{i+1} + \overline{V_{i+1}} (s_{i+0} + \overline{V_{i+0}} C_{\text{in}})))$$ ..........................(16)

Table (1) shows the connection between the redundant binary representation $q_i$, represented by $V_i$ (value) and $S_i$ (sign), the two’s complement result bit $b_0$ and the carry signals $C_{\text{in}}$ and $C_{\text{out}}$. The redundant binary addition multiplication and division have been studied by Takagi, Phatak and others [26-28]. The disadvantages of redundant arithmetic are the following. The redundancy doubles the storage requirements for data values. The representation can become ill conditioned, especially, as a result of iterated multiplication, and division and square root operations are more difficult to implement. Redundant representations play an important role in high speed computer arithmetic. Such representations support carry free additions i.e. addition in a small, constant time
and independent of operand widths. Certain weighted encodings of redundant number systems, using two valued digits (twits or generalized bits) have been used for high speed multiplier [29]. Twits are exemplified by a unibit which represents a value in \{-1,1\}, with -1 encoded as logic 0 and 1 encoded as logic 1. Such encodings, lead to efficient representations for redundant number systems and make it possible to realize arithmetic circuits based on a widely available optimized full and half adders, compressors and carry acceleration cells. Furthermore, they allow faithful representation of digit sets including those not having zero as a member, leading to enhanced encoding efficiency in some cases.

**1.3.5. THE (+1,-1) NUMBER REPRESENTATION**

The (+1,-1) Number representation was proposed by Pekmestzi in 1989 [30] for digital signal processing. In this representation low level of the signal is represented by -1 in place of zero in conventional binary while high level is represented by +1. Zero can not be represented in this system. Following the notation of [30], the analog fractional number A can be written as

\[
A = \sum_{k=1}^{n} (a_k)2^{-k}, \quad \text{where } (a_k) = +1 \text{ or } -1 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \q
or \( A = 2 \sum_{i=0}^{n} (a_i) 2^{-i} - [2^{-1} + 2^{-2} + .......(2^{-1} (2^{-2} + 2^{-3} + ....+ 2^{-n}))] \)

or \( A = 2 \sum_{i=0}^{n} a_i 2^{-i} + 2^{-n} - 1 \)

or \( A = 2A_n + 2^{-n} - 1 \) \hspace{1cm} \cdots \cdots \cdots \hspace{1cm} (21) \)

Where \( \text{A}_n \) represents a positive binary fractional number in conventional binary system using the same digit of (+1,-1) system. In this way the eq. (21) give the relation between the representation of same fractional numbers in both binary systems. The main feature of this representation is that the positive and negative decimal numbers can be converted into binary number in a unified way. There is no need of extra bit for sign. No extra hardware needed like one’s and two’s complement number for converting all 0’s into 1’s and vice-versa and extra addition bit one for addition to LSB in case of two’s complement.

In two’s complement system symmetry is not maintained during quantization because the number of two’s complement is greater by one in negative side than the positive side. In digital signal processing multiplication is very important from arithmetic point of view. During multiplication process in two’s complement, the sign bit is always checked, so that addition or subtraction may be performed using add shift technique. Finally, arithmetic shift is required to preserve the sign bit. Thus a single multiplier cell in two’s complement multiplication can not be introduced and hence, modular circuits can not be developed which is essential for VLSI. To make it modular, specific software are required.

Whereas the multiplication process in (+1,-1) binary system is straight, simple, and modular. Hence, it is helpful in VLSI design for DSP. The brief review of work performed in (+1, -1) representation is as follows:

In (+1, -1) representation the quantizer is represented by the relation

\[ Q[X] = \sum_{k=1}^{n} (x_k) 2^{-k} \] \hspace{1cm} \cdots \cdots \cdots \hspace{1cm} (22) \)

Where \( x_k = +1 \) or -1.
The quantity \( Q \mid X \) takes \( 2^n \) values equally spaced in the range \([-1+2^{-n}, +1-2^{-n}]\) for all combination of \((x_k)\). The distance between the successive values is \((2^{-n+1})\). The analog value is \( X \) represented by the discrete \( Q \mid X \). The step size of quantizer is twice in comparison of conventional binary system. The quantization levels are symmetric with respect to zero and representation of zero analog value is not possible. Quantization error is half of the step size. The mean square value is

\[
\sigma_q^2 = \frac{\Delta^2}{12}
\]

\[\text{.........(23)}\]

Where \( \Delta \) is the step size. The mean square error is large in comparison of conventional binary due to large step size.

Signal to noise ratio is given by

\[
10 \log_{10} (\text{SNR})_0 = 10 \log_{10} \left( \frac{3}{4 \times 2^R} \right)
\]

where \( R \) is the number of bits per sample.

In this system truncation and rounding produce the same result so, they are equivalent. Let us investigate through an example. Let a fractional number be \(0.111\bar{1} = \frac{27}{132} \) where \( \bar{1} = -1 \). The right most bit is one which is an extra bit and hence it should be truncated. After truncation, the number becomes \(0.11\bar{1} = \frac{13}{16} \). The error of truncation is \( \frac{1}{32} \). When other number is taken as \(0.111\bar{1} = \frac{29}{132} \). After truncated \( \bar{1} \), the number becomes \(0.1111 = \frac{15}{16} \). Now the truncation error is \( \bar{1} = \frac{-1}{32} \).

For rounding the number either \( 1 \) or \( \bar{1} \) is added to LSB. A number which is equivalent to the complement of the last bit i.e. \( \frac{-1}{32} = 0.\bar{1}1111 \) is required to be added because only full addition is possible. Addition of number is as follows
\[ 0.11\overline{1} = \frac{27}{32} \]
\[ 0.11\overline{1} = -\frac{1}{32} \]

\[ \text{extra bit} \]

1. \[ \overline{1} \overline{1} \overline{1} \overline{1} = 0.11\overline{1} \]

\[ = \frac{27}{32} \]

Since extra bit one is added to LSB and therefore it should be subtracted to get the correct result. Hence, the result will be \[ \frac{27}{32} - \frac{1}{32} = \frac{26}{32} = \frac{13}{16} = 0.11\overline{1}. \]

Similarly, for other number \[ 0.11\overline{1} = \frac{29}{32} \], a number which is equivalent to the complement of the last bit i.e. \[ 0.1\overline{1} = \frac{1}{32} \]. Adding \[ \frac{29}{132} = 0.111\overline{1} \]

and \[ \frac{1}{32} = 0.1\overline{1} \] in similar way, we have \[ 0.11\overline{1} = \frac{31}{32} \]

Hence the correct result is

\[ \frac{31}{32} - \frac{1}{32} = \frac{30}{32} = \frac{15}{16} = 0.11\overline{1} \]

Thus, we find truncation and rounding produces the same result. The number before truncation or rounding may be greater or smaller depending on LSB being 1 or \( \overline{1} \) respectively. In this system, the truncation of number from \( n \) digits to \( b \) digits causes an error \( E \) which satisfies the following inequality.

\[ -2^{-b} + 2^{-n} < E < 2^{-b} - 2^{-n} \]

1st order digital filter is computed by the difference equation

\[ y(n) = x(n) + a y(n-1) \]

\[ \text{.........(24)} \]

Where \( a \) is the 1st order coefficient, \( x(n) \) is the input data \( y(n-1) \) is previous output data

Let us take an example

\[ x(n) = -\frac{3}{16} = 0.1\overline{1} \]

\[ \text{.........(25)} \]

\[ a y(n-1) = \frac{59}{128} = 0.1\overline{1} \overline{1} \overline{1} \]

\[ \text{.........(26)} \]
where \( a = 0.111 \frac{7}{16} \) is coefficient.

\[ y(n-1) = \overline{111} = \frac{3}{128} \] is delayed data.

Adding (25) and (26)

\[ x(n) = 0.1 \overline{11} \]

\[ |ay(n-1)| = 0.1\overline{11} \]

\[ 1 \leftrightarrow \overline{11} \text{ (rounding error)} = -\frac{5}{128} \]

\[ y(n) = 0.1\overline{11} \]

Now the truncated bits are \( \overline{11} = \frac{3}{128} \). MSB (1) is used as carry bit i.e. \( \frac{1}{16} \). Now

\[ \frac{3}{128} - \frac{1}{16} = -\frac{5}{128} \]

So, carry bit does not change the rounding error.

Second order filter algorithm is given by

\[ y(n) = x(n) + ax(n-1) + b x(n-2) - cy(n-1) - d y(n-2) \]

where \( a, b, c, \) and \( d \) are filter coefficients. Higher order filters are realized by combining 1st and 2nd order filters in cascade.

1.4 LINEAR PREDICTION CODING

Linear predictive coding represents a completely different approach to the problem of source encoding. In LPC the source is modeled as a linear system (filter) which, when excited by an appropriate input signal results in the observed source output. In stead of transmitting the samples of source wave form to receiver, the parameters of the linear system are transmitted along with the appropriate excitation signals. Linear predictive coding for speech signals have been studied by authors [31-35] in detail.

To be specific, suppose the source output is sampled at a rate to or exceeding the Nyquist rate and let the samples taken over some interval be denoted as \( x_n, n = 0, 1, \ldots N-1 \). In LPC the sampled sequence is assumed to have been generated by an all pole (discrete-time) filter having the transfer function.
H(z) = \frac{G}{1 - \sum_{k=1}^{p} a_k z^{-k}} ................................(27)

Appropriate excitation functions are an impulse, a sequence of impulses, or a sequence of white noise with unit variance. In any case, suppose that the input sequence is modeled is denoted as v_n, n = 0, 1, 2……. Then the output sequence of the all pole model satisfies the difference equation.

\[ x_n = \sum_{k=1}^{p} a_k x_{n-k} + Gv_n \] .................(28)

In general, the observed source output x_n, n = 0, 1, 2……. N-1 does not satisfy the difference eq. (24), but only its model does. If the input is a white-noise sequence or an impulse, we may form an estimate (or prediction) of x_n by the weighted linear combination.

\[ \hat{x}_n = \sum_{k=1}^{p} a_k x_{n-k} \quad n > 0 \] .................(29)

the difference between x_n and \( \hat{x}_n \) is given by

\[ e_n = x_n - \hat{x}_n \]

\[ = x_n - \sum_{k=1}^{p} a_k x_{n-k} \] .................(30)

represents the error between the observed value x_n and estimated (predicted) value \( \hat{x}_n \). The filter coefficient \{a_k\} can be selected to minimize the mean square value of this error.

### 1.5 Warped Linear Prediction Coding (WLPC)

Warped linear predictive coding (WLPC) is a clear step forward in the utilization of characteristics of human hearing in designing coding algorithms since a WLPC system can be adjusted such that the spectral resolution closely approximates the frequency resolution of human hearing. After pioneering work by Strube [36], it has not been widely used. A group of researchers [37-39] studied and exploited WLPC techniques in speech analysis and coding applications as a part of their mel-generalized cepstral coding methodology.

The related techniques with Laguerre filters have been used by many authors especially in the field of automatic control [40]. For wideband audio
applications it was first used by Laine et al [41]. Recently, the current authors have used WLPC techniques specially in low-delay wideband audio coding [42-44]. Basically, all conventional techniques for parameters spectral estimation and linear filtering can be warped in a straightforward way.

1.5.1. FREQUENCY WARping

The transfer function of first-order allpass (AP) filter is given by

\[ D(z) = \frac{z^{-1} - \lambda}{1 - \lambda z^{-1}} \]  

(31)

By definition, the magnitude response of the filter is a constant. The turning point frequency \( f_p \) defines the frequency where warping does not affect the frequency resolution i.e. where the group delay is one. A convenient expression for \( f_p \) as a function of \( \lambda \) and sampling frequency \( f_s \) is given by

\[ f_p = \pm f_s/2\pi \text{ arc } \cos (\lambda) \]  

(32)

The frequency resolution of a warped system with \( \lambda \geq 0 \) is higher below \( f_p \) and lower above \( f_p \), than in a conventional system with uniform frequency resolution. The difference in frequency resolution between a warped and conventional signal processing system would be small at low sampling rates.

The standard method of transforming a discrete signal processing system to a warped system involves replacing the unit delays of the original system by first-order allpass filters.

1.5.2. WARPED FILTER STRUCTURES

A prediction error filter, or the inverse filter, given by

\[ A(z) = 1 - \sum_{k=1}^{N} a_k D(z)^k \]  

(33)

is a warped FIR type filter (WFIR). The output of prediction error signal is usually called the residual signal \( r(n) = x(n) - \hat{x}_n \). Warped direct form FIR filters were introduced by the author [45] for a review. A warped FIR type lattice filter may be derived directly by replacing the unit delay of conventional filter structure with first order allpass elements [46]. The reflection coefficient of a warped structure can be computed from the estimated coefficients of a direct-
from filter as in the case of conventional filter. It is also possible to implement a
synthesis filter given by
\[ A^{-1}(z) = \frac{1}{1 - \sum_{k=1}^{N} a_k D(z)^k} \] .............(34)

1.6 THE LEVINSON - DURBIN ALGORITHM

The Levinson - Durbin [47] algorithm is an order-recursive method for
determining the solution in the set of linear equations.
\[ \Phi_p a_p = \phi_p \] .............(35)
where \( \Phi_p \) is a \( p \times p \) Toeplitz matrix, \( a_p \) is the vector of predictor coefficient
expressed as
\[ a_p^* = [a_{p1}, a_{p2}, \ldots, a_{pp}] \] .............(36)

Where \( a_p^* \) is transpose of \( a_p \) and \( \phi_p \) is a \( p \)-dimensional vector with elements
\[ \phi_p^* = [\phi(1)\phi(2)\ldots\phi(p)] \] .............(37)

Where \( \phi_p^* \) is transpose of \( \phi_p \).

For first order (\( p=1 \)) predictor, we have the solution
\[ \phi(0) a_{11} = \phi(1) \]
\[ a_{11} = \frac{\phi(1)}{\phi(0)} \] .............(38)

The residual mean square error (MSE) for the first order predictor is
\[ E_1 = \phi(0) - a_{11} \phi(1) \]
Or \[ = \phi(0) - a_{11}^2 \phi(0) \]
Or \[ = \phi(0) (1 - a_{11}^2) \] .............(39)

In general we may express the solution for the coefficients of an \( m \)th order
predictor in terms of the coefficients of the \( (m-1) \)st order predictor. Thus we
express \( a_m \) as the sum of two vectors, namely.
\[ a_m = [a_{m1}, a_{m2}, \ldots, a_{mm}] = [a_{m-1}, d_{m-1}, 0] \]
\[ = [a_{m-1}, 0] + [d_{m-1}, k_m] \] .............(40)
Where the vector $d_{m-1}$ and scalar $k_m$ are to be determined. Also, $\Phi_m$ may be expressed as

$$\phi_m = \begin{bmatrix} \phi_{m-1} \\ \phi'_m \\ \phi(0) \end{bmatrix}$$

$$\ldots \ldots (41)$$

Where $\phi'_m$ is just the vector $\phi_{m-1}$ in reverse order.

Now,

$$\begin{bmatrix} \phi_{m-1} & \phi'_m \\ \phi'_m & \phi(0) \end{bmatrix} \begin{bmatrix} a_{m-1} \\ d_{m-1} \end{bmatrix} + \begin{bmatrix} 0 \\ k_m \end{bmatrix} = \begin{bmatrix} \phi_{m-1} \\ \phi_m \end{bmatrix}$$

$$\ldots \ldots (42)$$

From eq. (42) we obtain the two equations. The first is the matrix equation

$$\Phi_{m-1} a_{m-1} + \Phi_{m-1} d_{m-1} + k_m \phi'_m = \phi_{m-1}$$

$$\ldots \ldots (43)$$

but $\Phi_{m-1} a_{m-1} = \phi_{m-1}$. Hence equations (43) simplifies to

$$\Phi_{m-1} d_{m-1} + k_m \phi'_m = 0$$

$$\ldots \ldots (44)$$

This equation has the solution

$$d_{m-1} = -k_m \Phi_{m-1}^{-1} \cdot \phi'_m$$

$$\ldots \ldots (45)$$

But $\phi'_m$ is just $\phi_{m-1}$ in reverse order. Hence, the solution in equation (45) is simply $a_{m-1}$ in reverse order multiplied by $(-k_m)$ i.e.

$$d_{m-1} = -k_m \begin{bmatrix} a_{m-1} & a_{m-2} & \ldots & a_{m-1} \end{bmatrix}$$

$$\ldots \ldots (46)$$

The second equation obtained from (42) is the scalar equation

$$\Phi'^{m-1} a_{m-1} + \Phi'^{m-1} d_{m-1} + \phi(0) k_m = \phi(m)$$

$$\ldots \ldots (47)$$

We eliminate $d_{m-1}$ from (47) by use of (46). The resulting equations give us $k_m$

i.e.

$$k_m = \frac{\phi_m - \phi'_m a_{m-1}}{\phi(0) - \phi'_m \phi'_m}$$

$$\ldots \ldots (48)$$

where $E_{m-1}$ is the residual MSE given as

$$E_{m-1} = \phi(0) - a_{m-1} \phi_{m-1}$$

$$\ldots \ldots (49)$$
By substituting (46) for \( d_{m-1} \) in (40), we obtain the order recursive relation

\[
a_{mk} = a_{m-1,k} - k_m a_{m-1,m-k}
\]

where \( k = 1,2, \ldots, m-1 \) \hspace{1cm} (50)

and \( a_{mm} = k_m \)

The minimum MSE may also be computed recursively. We have

\[
E_m = \phi(0) - \sum_{k=1}^{m} a_{mk} \phi(k)
\]  \hspace{1cm} (51)

From equations (50) and (51), we obtain

\[
E_m = \phi(0) - \sum_{k=1}^{m} a_{m-1,k} \phi(k) - a_{mm} \left[ \phi(m) - \sum_{k=1}^{m-1} a_{m-1,m-k} \phi(k) \right]
\]  \hspace{1cm} (52)

But the term in the brackets in (52) is just the numerator of \( k_m \) in (48).

Hence

\[
E_m = E_{m-1} - a_{mm} E_{m-1}
\]

\[
= E_{m-1} (1 - a_{mm}^2)
\]  \hspace{1cm} (53)

Equation (53) determines the prediction coefficient.

### 1.7 DIFFERENT TYPES OF ADDERS

#### 1.7.1. CARRY SAVE ADDER

A look ahead carry save adder can add two \( n \)-bit numbers in \( O(\log n) \) time adding three \( n \)-bit numbers takes only a constant additional amount of time. The method is to reduce the problem of adding three numbers to the problem of adding two numbers.

Given three \( n \)-bit numbers are \( x = \{x_{n-1}, x_{n-2}, \ldots, x_0 \} \),
\( y = \{y_{n-1}, y_{n-2}, \ldots, y_0 \} \) and \( z = \{z_{n-1}, z_{n-2}, \ldots, z_0 \} \),

and \( n \) bit carry – save adder produce an \( n \) – bit.

Number \( u = \{u_{n-1}, u_{n-2}, \ldots, u_0 \} \) and an \( (n+1) \) bit number \( v = \{v_{n-1}, v_{n-2}, \ldots, v_0 \} \) such that

\[
u + v = x + y + z
\]  \hspace{1cm} (54)

as shown in fig.(3).

Here, \( u_i = \text{parity (} x_i, y_i, z_i \text{) } \) and

\( v_{i+1} = \text{majority (} x_i, y_i, z_i \text{) } \)

for \( i = 0,1, \ldots, n-1; \ v_0 \) always equals to 0.
The 8-bit carry save adder in fig. (4) consists of full adders \( FA_0, FA_1, \)
\( FA_2, \ldots, FA_7 \). For \( i = 0,1,\ldots,7 \), full adder \( FA_i \) is taken as \( x_i, y_i \) and \( z_i \). The sum-bit output of \( FA_i \) is taken as \( u_i \) and carry out of \( FA_i \) is taken as \( v_{i+1} \). Bit \( v_0 \) is hardwired to 0.

Since the computation of all \( 2n+1 \) output bits are independent, they can be
performed in parallel. Thus a carry save adder operates in \( \Theta(1) \) time and \( \Theta(n) \)
size. Although this method is not asymptotically better than the method of using
two carry – look ahead additions, it is much faster in practice.

1.7.2. CONDITIONAL SUM ADDER

In this scheme the sum \( S = A + B \) (\( A = A_{n-1}, \ldots, A_0 \);
\( B = B_{n-1}, \ldots, B_0 \)) is developed under both the 0 and 1 state of the input
carry. It is implemented in multiple steps. In the very first step, the sum and the
carry for the \( i \)th position is developed under both the conditions of 0 or 1 of input
carry to the \( i \)th full adder. Let these be marked as \( S^0_i, c^0_i \) for input carry state ‘0’
and \( S^1_i, c^1_i \) for input carry state ‘1’. In this next level sum and carry for \( (i+1) \)th
and \( i \)th positions are developed as \( S^0_{i+1}, c^0_{i+2}, S^0_i \) for carry input ‘0’ and \( S^1_{i+1}, c^1_{i+2}, S^1_i \) for carry input ‘1’. Four different situations may rise:

1. For \( c^0_i, c^1_i = 00 \), \( S^0_{i+1}, c^0_{i+2} \) is carried forward to the next level both for
   carry input 0 or 1, since irrespective of carry input carry for \( i \)th position,
   the sum and the carry for \( (i+1) \)th position are \( S^0_{i+1}, c^0_{i+2} \).

2. For \( c^0_i, c^1_i = 01 \) or 10, \( S^0_{i+1}, c^0_{i+2} \) for carry 0 and \( S^1_{i+1}, c^1_{i+1} \) for carry 1,
   are carried forward to next level.

3. In the case \( c^0_i, c^1_i = 11 \), \( S^1_{i+1}, c^1_{i+1} \) is carried forward to the next level
   both for carry 0 or 1, since irrespective of the input carry to the \( i \)th
   position, the sum and the carry for \( (i+1) \)th position are \( S^1_{i+1}, c^1_{i+2} \).

Conditional sum adder was first introduced by Sklansky [48] in
1960. Conditional sum adder suffers from fan out limitation since the number of
multiplexers that need to be driven increase exponentially. In the worst case, a
carry signal is used to select \( n/2 \) multiplexers in an \( n \)-bit adder. A modification
of Conditional sum adder has been represented by Becker and Kolla [49]. These
adders are called conditional carry adders since Conditional sum adder principle
applies only to the carry generation circuit. The number of fan-out for each multiplexer is constant internally. In high speed adder designs, fan out limitation seriously degrades estimated addition speed.

A fast binary adder with conditional carry generation is presented by Chung Lo [50] in 1997. He used static CMOS realization. This model has been compared to spanning tree carry-look ahead adder. In a 1-\(\mu\)m static CMOS realization, the proposed adder adds two 32 bit operands in a 3.28 ns. This delay is increased from the assertion of the input to the arrival to the slowest sum bit.

1.7.3. CARRY LOOK-AHEAD ADDER

An \(n\) bit adder is just a combinational circuit. It can therefore be written by a logic form whose form is a sum of products and can be completed by a circuit with two levels of logic. The formula for the \(i\)th adder can be written as

\[ S_i = a_i, b_i, c_i + a_i, b_i, c_i + a_i, b_i, c_i + a_i, b_i, c_i \]

Where \(c_i\) is both, the carry in the \(i\)th adder and the carry out from the \((i-1)\)st adder.

The problem with this formula is that although we know the values of \(a_i\) and \(b_i\), which are inputs to the circuit but we don’t know \(c_i\). So our goal is to write \(c_i\) in terms of \(a_i\) and \(b_i\). To accomplish this, we first write it

\[ c_i = g_{i-1} + p_{i-1}c_{i-1} \]

\[ g_i = a_{i-1}b_{i-1}, p_{i-1} = a_{i-1} + b_{i-1} \]

If \(g_{i-1}\) is true, then \(c_i\) is certainly true, so a carry is generated. Thus, \(g\) is for generate. If \(p_{i-1}\) is true, then if \(c_{i-1}\) is true, it is propagated to \(c_i\).

Equation (55) can be rewritten as:

\[ c_i = g_{i-1} + p_{i-1}g_{i-2} + p_{i-1}p_{i-2}g_{i-3} + \ldots + p_{i-1}p_{i-2}\ldots p_1p_0 + p_{i-1}p_{i-2}\ldots p_1p_0c_0 \]

An adder that computes carries using eqn (57) is called a carry look ahead adder, or CLA. A CLA requires one logic level to form \(p\) and \(g\), two levels to form the carries, and two for sum, for a grand total of five logic levels. This is a vast improvement over the 2n levels required for the ripple carry adder.

It is evident from eqn (57) that a carry look ahead idea to build an adder that has about \(\log_2 n\) logic levels (substantially fewer than the 2n required by a
ripple carry adder), and has a simple regular structure. The idea is to build up the p’s and g’s in steps. We have already seen that
\[ c_i = g_0 + c_0 p_0 \] ..........(58)

This says that there is a carry out of 0 position (c_1). There is either a carry generated in 0th position or there is a carry into the 0th position and the carry propagates; similarly,
\[ c_2 = G_{01} + P_{01} c_0 \] ..........(59)

\( G_{01} \) means there is a carry generated out of the block consisting of the first two bits. \( P_{01} \) means that a carry through this block. \( P \) and \( G \) have the following logic equations.
\[ G_{01} = g_1 + p_1 g_0 \] ..........(60)
\[ P_{01} = p_1 p_0 \] ..........(61)

More generally, for any \( j \) with \( i < j, j + 1 < k \), we have the recursive relations
\[ c_{k+1} = G_{ik} + p_k c_i \] ..........(62)
\[ G_{ik} = G_{j+1,k} + p_{j+1,k} G_{ij} \] ..........(63)
\[ P_{ik} = P_{j} p_{j+1,k} \] ..........(64)

Equation (63) says that a carry is generated out of the block consisting of bits \( i \) through \( k \) inclusively either it is generated in the high-order part of the block (\( j+1, k \)) or it is generated in the low order part of the block (\( i, j \)) and then propagated through the high part. These equations will also hold for \( i \leq j \leq k \) if we have \( G_{ij} = g_i \) and \( P_{ii} = p_i \).

1.8 MULTIPLIERS

1.8.1. UNSIGNED MULTIPLIER

Compared with addition and subtraction, multiplication is a complex operation, whether performed in hardware or software. A wide variety of algorithms have been used in various computers. Firstly, we will discuss unsigned integer’s multiplication and secondly, the multiplication of two’s complement representation for signed numbers and finally the other multipliers. Fig. (5) illustrates the multiplication of unsigned binary integers, as might be carried out on paper and pencil. Several important observations can be made.
1. Multiplication involves the generation of partial products, one for each digit in the multiplier. These partial products are then summed to produce the final product.

2. The partial products are easily defined. When the multiplier bit is 0, the partial product is 0. When the multiplier bit is 1, the partial product is the multiplicand.

3. The total product is produced by summing the partial products. For this operation each successive partial product is shifted one position to the left relative to the preceding partial product.

4. The multiplication of two n-bit binary integer results in a product of up to 2n bits in length.

Compared with paper and pencil approach, there are several things we can do to make the operation more efficient. First, we can perform a running addition on the partial products rather than waiting until the end. This eliminates the need for storage of all the partial products, fewer registers are needed. Second, we can save some time on the generation of partial products. For each 1 on the multiplier, an add and shift operation are required, but for each 0, only a shift is required. Fig. (6) shows a possible implementation employing these measures. The multiplier and multiplicand are loaded into two registers Q and M respectively. A third register called A register, is also needed and is initially set to 0. There is also a 1-bit C register, initialized to 0, which holds a potential carry bit resulting from addition.

The operation of the multiplier is as follows. Control logic reads the bits of the multiplier one at a time. If \(Q_0\) is 1, then the multiplicand is added to the A register and the result is stored in the A register. Then, all of the bits of the C, A and Q registers are shifted to the right one bit, so that the C bit goes into \(A_{n-1}\), \(A_0\) goes into \(Q_{n-1}\) and \(Q_0\) is lost. If \(Q_0\) is 0, then no addition is performed, but shifting is done only. This process is repeated for each bit of the original multiplier. The resulting 2n bit product is contained in the A and Q registers. A flowchart of the operation is shown in fig. (7). On the second cycle, when the multiplier bit is 0, there is no add operation.
1.8.2. TWO'S COMPLEMENT MULTIPLIER

Two’s compliment multiplier using Booth’s algorithm [2] is depicted in fig (8) and can be described as follows. As before, the multiplier and multiplicand are placed in Q and M registers respectively. There is also a 1-bit register placed logically to the right of the least significant bit \( Q_0 \) of the Q register and designated as \( Q_{-1} \). The results of the multiplication will appear in A and Q registers. A and \( Q_{-1} \) are initialized to 0. Control logic scans the bits of the multiplier one at a time. Now, as each bit is examined, the bit to its right is also examined. If the two bits are same 1-1 or 0-0), then all of the bits of the A, Q and \( Q_{-1} \) registers are shifted to the right 1 bit. If the two bits differ; then the multiplicand is added to or subtracted from the A register according as the two bits are 0-1 and 1-0 respectively. Following the addition or subtraction, the right shift occurs. In either case the right shift is such that the left most bit of \( A_{n-1} \) is not only shifted into \( A_{n-2} \) but also remains in \( A_{n-1} \). This is required to preserve the sign of the number in A and Q. It is known as an arithmetic shift, since it preserves the sign bit.

Present work deals with the study of hardware realization of combinational sequential and digital signal processing circuits in new binary system (+1,-1). In combinational circuits, various gates were constructed and their characteristics have been studied. The results so obtained were compared to conventional (0,1) ECL gates and concluded that for DSP applications such gates are useful. In sequential circuits, digital clock, R-S flip-flop, clocked R-S flip-flop, D flip-flop, J-K flip-flop, TTL flip-flop and different shift registers were constructed and functions were tested by truth tables in the (+1,-1) representation. The hardware realizations are different from conventional system. Positive and negative clocks have been used everywhere. Positive and negative signals can be directly stored as data and shifted without going for any conversion as in (0, 1) system. The hardware realization of digital to analog and analog to digital conversions have been discussed. The circuits were experimentally tested and found suitable.
Carry look ahead adder and carry save add shift technique for multiplication process are being studied in detail. The speed of multiplier have been analyzed and compared with conventional two’s complement multiplier. The delay is reduced much in our system. The other advantages of multiplier have been also discussed. Finally, linear prediction (LP) and warped linear prediction (W LP) coding in (+1,-1) representation have been discussed in detail. It contains the study of digital filters and autocorrelation function. The speech signals are analyzed and synthesized using LPC and WLPC. Results have been compared with experimental values and found suitable. Result and discussion as well as conclusion of entire work is presented at the end.