Digital Signal Processors

Architecture, Programming and Applications

B Venkataramani
M Bhaskar

Tata McGraw-Hill

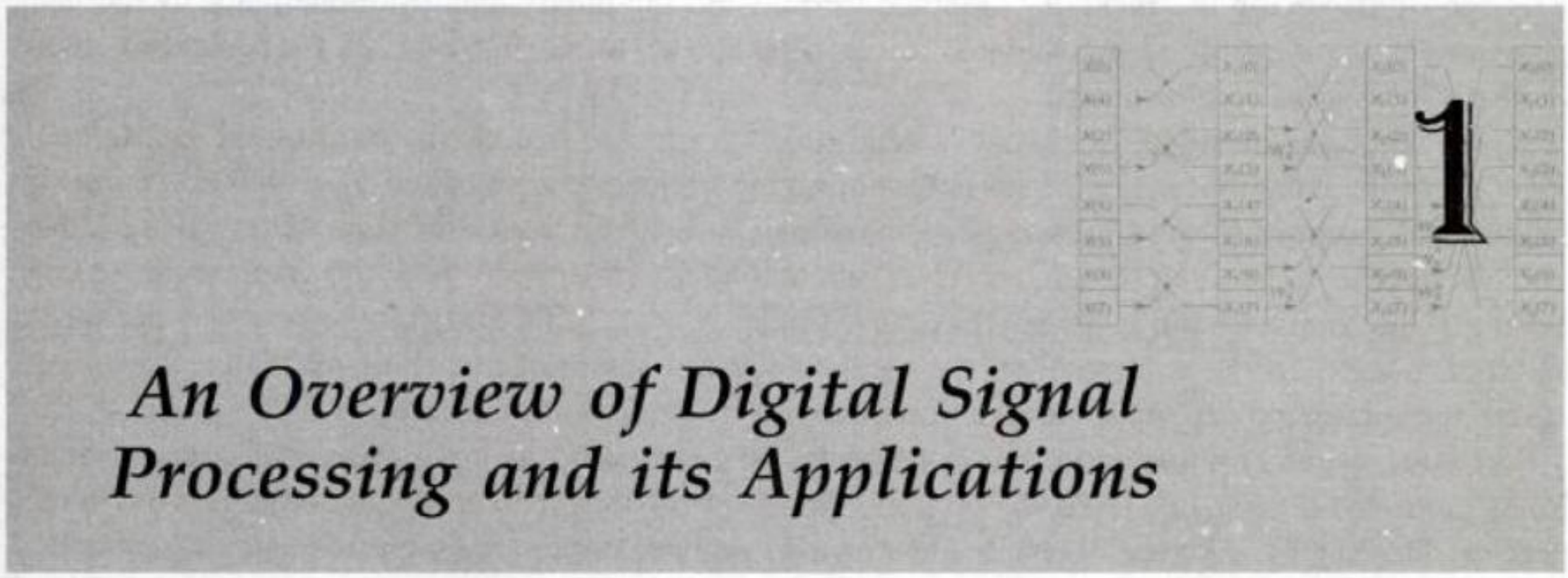
Fourteenth reprint 2008
RCLYYDRXRCLLD

ISBN-13: 978-0-07-047334-8
ISBN-10: 0-07-047334-X

Published by Tata McGraw-Hill Publishing Company Limited,
7 West Patel Nagar, New Delhi 110 008, typeset at Script Makers 19,
A1-B DDA Market, Pashchim Vihar, New Delhi 110 063 and printed at
Sai Print-O-Pack, New Delhi 110 020

The McGraw-Hill Companies

Contents



1 An Overview of Digital Signal Processing and its Applications

1.1 SIGNALS AND THEIR ORIGIN

A signal refers to any continuous function $f(\ )$ that is a function of one or more variables such as time, space, frequency, etc. Some common examples of signals are the voltage across a resistor; the velocity of a vehicle; the light intensity of an image; temperature; and pressure inside a system, etc., as a function of time, space or any other independent variable. These signals are processed in order to either monitor or control one or more parameters of a system. Detection of the average, RMS or peak values of a parameter, separation between adjacent peaks or zero crossings are examples of some processing carried out on the signal. In some applications, the processing may be done in order to produce another signal that has better characteristics than those of the original signal. The processing of the signal is carried out efficiently and with ease if these signals are converted to equivalent electrical voltages or currents with the use of transducers. Hence, the emphasis in this book will be restricted to processing signals in electrical form.

1.2 NOISE

Processing of the signal is made complex by the presence of other signals called *noise*. The noise signals generated from manmade and natural objects, Electrical appliances/machinery, lightening and thunderstorms are some of the sources of noise. In addition to this, any signal that interferes with the detection of the desired signal may be called an *interference* or *noise*. The first step in signal processing is to combat the effect of noise. When the noise and the desired signal have different characteristics, the signal can be completely separated from the noise before processing the signal further. Even though this step is an overhead this may be mandatory. Let us consider the following example, to illustrate this.

1.3 FILTERS AND NOISE

Frequency shift keying is a technique adopted for transmitting binary data from one place to another. For example, the transmitted sinusoidal signal may be chosen to be 1025 or 1225 Hz depending upon whether 0 or 1 is to be transmitted.

On the receiver, the signal frequency is determined in order to decode the transmitted data to be 0 or 1. One of the techniques adopted for determining the frequency (also called *frequency detection*) is to count the number of zero crossings of the signal in a given period of time. This can be done efficiently if the transmitted signal is received without noise. However, one of the common noise that appears at the receiver input is the power supply ripple. A noise of 60/50 Hz frequency may appear at the input to the receiver. Alternately if the receiver site has any high frequency oscillator, it may get leaked through the power supply lines and will appear at the input to the receiver.

Both these types of noise will make the detector take wrong decisions. However, its performance can be improved if these two types of noise are removed from the received signal before it is fed to the detector. This can be achieved by using a low pass filter for removing the high frequency signal and a high pass filter for removing the low frequency power supply ripple. The ideal characteristics of low pass, high pass and band pass filters are shown in Fig. 1.1. These filters may be constructed using discrete elements and their frequency response do not have the ideal characteristics. The ideal filters pass the signals in the pass band without any attenuation. The signals in the stop band have infinite attenuation. The transition from pass band to stop band occurs instantaneously as the frequency of the signal is swept. The frequency at which this occurs is called the cutoff frequency f_c. These filters by themselves qualify to be called signal processors as they remove the unwanted frequency components. Additional processing may be required to carry out a particular task such as frequency detection.

1.4 CORRELATORS

However, separation of the signal from the noise cannot always be achieved using simple filters shown in Fig. 1.1. The desired signal and the interfering noise may both lie in the same frequency range (bandwidth). In this case, the signal may be retrieved from noise by exploiting its behaviour in the time domain. The effect of the interference signal may be minimised by multiplying the received signal with the replica of all possible transmitted signals with suitable delay and then integrating them over one period (e.g., 1-bit duration in the case of FSK) of the transmitted signal. This operation is called *correlation*.

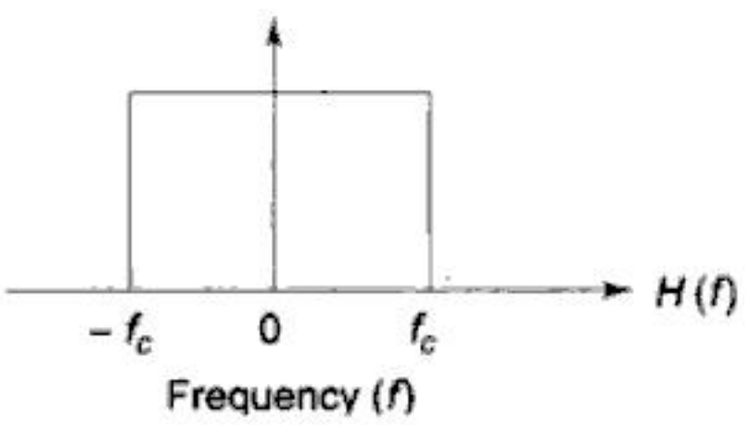


Fig. 1.1(a) *Low pass filter response*

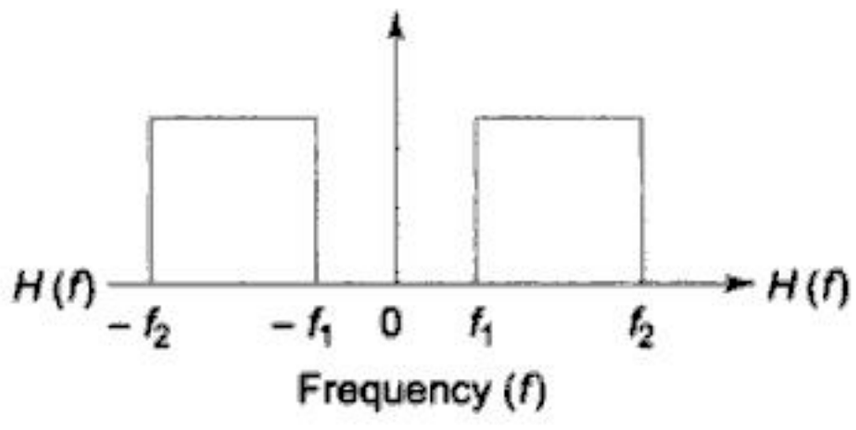


Fig. 1.1(b) *Bandpass filter response*

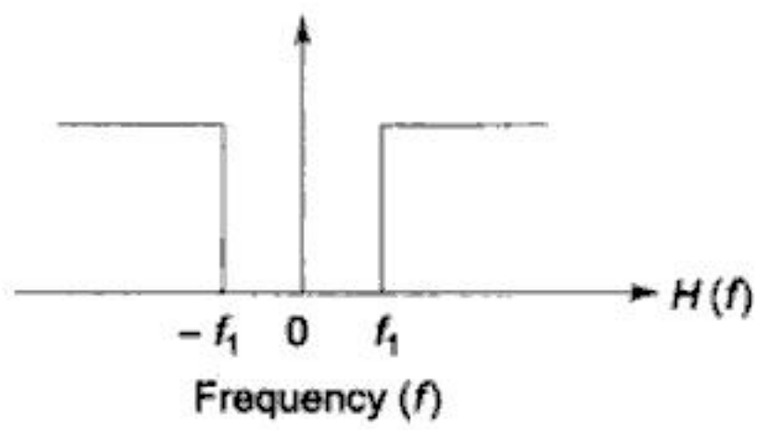


Fig. 1.1(c) *Highpass filter response*

1.5 CONVOLUTION AND INVERSE FILTERING

On its transit to the receiver, the transmitted signal may pass through several systems and each of these systems may modify it. The output $y(t)$ of a linear time invariant (LTI) causal system can be expressed as the convolution of the input $x(t)$, with the impulse response $h(t)$ and is given by the expression

$$y(t) = \int_0^t x(\tau)\, h(t-\tau)\mathrm{d}\tau = x(t) * h(t) \tag{1.1}$$

Here * denotes the convolution operation.

A system is said to be linear if the superposition principle is true. In other words if $y_1(t)$ is the response to the input $x_1(t)$ and $y_2(t)$ is the response to the input $x_2(t)$, then for any scalar α, β let the response of the system to the input $\alpha x_1(t) + \beta x_2(t)$ be $y(t)$. Then if the system is linear

$$y(t) = \alpha y_1(t) + \beta y_2(t) \tag{1.2}$$

A system is said to be time invariant if a shift in the input causes a corresponding shift in the output. In other words if $y(t)$ is the response of the system to $x(t)$, then for a time invariant system the response to the time shifted input $x(t-\tau)$ is given by $y(t-\tau)$.

A system is said to be causal if the output at any instant is determined only by its present input and its past inputs/outputs but not by its future inputs. Only causal system can be realised in practice and this requires the impulse response of the system $h(t)$ to be zero for $t < 0$.

For simplicity and to a good accuracy, the systems can be modelled to be LTI and causal in majority of signal processing applications. However, there are applications such as image processing system where causality is not true.

In order to nullify the effect of a system on the transmitted signal, the received signal may be passed through another system whose transfer function (Laplace transform of the impulse response) is the inverse of the original system. Inverse filters and equalisers use this principle.

1.6 FOURIER TRANSFORM AND CONVOLUTION THEOREM

The signals may be processed either in the time domain or in the transform domain. For example, computation of the output $y(t)$ with the use of the convolution integral given by (1.1) is an example of

time domain processing. The output may also be computed using transform domain techniques. This can be explained as follows: A signal is said to be of finite energy if

$$\int_{-\infty}^{\infty} |f(\tau)|^2 \, d\tau < \infty \tag{1.3}$$

Any finite energy signal $f(t)$ can be represented using the Fourier transform $F(\omega)$ given by

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \, e^{-j\omega t} dt \tag{1.4}$$

$f(t)$ can be obtained from $F(\omega)$ using the inverse Fourier transform given by

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \, e^{j\omega t} d\omega \tag{1.5}$$

When the lower limit in (1.4) and (1.5) is 0 and $j\omega$ is replaced by s, we get the expression for the Laplace transform of $f(t)$ and inverse Laplace transform of $F(s)$, respectively.

Convolution theorem relates $h(t)$, $x(t)$ and the convolved output $y(t)$ to their Fourier and Laplace transforms. Let the Fourier transform and the Laplace transform of $(x(t), h(t), y(t))$ be $(X(\omega), H(\omega), Y(\omega))$ and $(X(s), H(s), Y(s))$, respectively.

Then if $y(t) = x(t) * h(t)$ then $Y(\omega) = X(\omega) H(\omega)$ and $Y(s) = X(s) H(s)$.

Similarly if $Y(\omega) = A(\omega) * B(\omega)$ then $y(t) = a(t) b(t)$

and $Y(s) = A(s) * B(s) \Rightarrow y(t) = a(t) b(t)$.

As mentioned earlier * denotes the convolution operation. $H(s)$, the Laplace transform of the impulse response $h(t)$ of a causal LTI system is called the transfer function of the system and is given by

$$H(s) = \frac{Y(s)}{X(s)} \tag{1.6}$$

where $X(s)$ and $Y(s)$ are the Laplace transforms of $x(t)$ and $y(t)$ respectively. Laplace transforms have a number of applications such as studying the stability of the system, solving the differential equations and finding the initial value and final value of a system.

1.7 SAMPLING THEOREM AND DISCRETE TIME SYSTEM

Various signal processing operations explained above can be carried out either directly on the continuous signal or indirectly using the samples of the input signal and impulse responses. Accordingly they are called continuous time signal processing and discrete time signal processing, respectively. The discrete time signal offers a number of advantages. Firstly, it permits a number of such signals to be transmitted using the same channel by sending them at disjoint time intervals. This technique is called the *time division multiplexing*. However, in order to transmit the information without any loss of information the discrete time signal should satisfy the Nyquist's sampling theorem, which states that, any signal bandlimited to a maximum frequency of f_m can be perfectly reconstructed from its samples if the sampling rate f_s is greater than or equal to $2f_m$. If f_s is equal to $2f_m$, then it is called the *Nyquist rate*.

If the sampling rate is less than $2f_m$, then any signal component f_h which is greater than $f_s/2$ by Δf (i.e. $f_h = f_s/2 + \Delta f$) gets mapped to a frequency $f_s/2 - \Delta f$ after sampling and appears as a low frequency signal. This is called *aliasing*. To avoid this, either the sampling rate should be chosen to be above the Nyquist rate or the sampler should be preceded by a low pass filter with $f_c = f_s/2$.

1.8 LINEARITY, SHIFT INVARIANCE, CAUSALITY AND STABILITY OF DISCRETE TIME SYSTEMS

Some of the properties of the continuous time system discussed in Section 1.5 can be extended for the discrete time system as follows: A discrete time system is said to be LSI if the superposition property holds and a shift in the input causes a corresponding shift in the output.

In other words if $y_1(n)$ is the response of a discrete time system to the input $x_1(n)$ and $y_2(n)$ is the response to the input $x_2(n)$, then for any scalar α, β let the response of the system to the input $\alpha x_1(n) + \beta x_2(n)$ be $y(n)$. Then if the system is linear

$$y(n) = \alpha y_1(n) + \beta y_2(n) \tag{1.7}$$

The discrete time system is said to be shift invariant if a shift in the input causes a corresponding shift in the output. In other words if $y(n)$ is the response of the system to $x(n)$, then for a shift invariant system the response to the input $x(n-k)$ is given by $y(n-k)$. A LSI system can also be called as LTI system if the shift in the index (e.g. k in $n-k$) corresponds to a different sampling instant.

A discrete time system is causal if the impulse response sequence $h(n) = 0$ for $n < 0$. The system is stable if a bounded input sequence $x(n)$ i.e. a sequence for which $|x(n)| < \eta$ for all n and any arbitrary η, results in bounded output sequence $y(n)$. This is achieved if

$$\sum_{k=0}^{n} |h(k)| < \infty \tag{1.8}$$

1.9 *Z* TRANSFORM

Analogous to the Laplace transform for the continuous time system, for the discrete time system Z transform is defined. The Z transform of a sequence $x(n)$ is denoted as $X(z)$ and is given by

$$X(z) = \sum_{n=0}^{\infty} x(n) z^{-n} \tag{1.9}$$

Here z denotes a complex variable.

The Z transform $H(z)$ of the impulse response sequence $h(n)$ of a causal LSI system is called the system function. $H(z)$ can be used to examine the stability of a discrete time system. For a stable system, the magnitude of the poles of $H(z)$ should be <1. In other words, the magnitude of the points at which the denominator of $H(z)$ becomes zero should be less than 1. For example, in a system with system function $H(z)$ given by

$$H(z) = \frac{1}{(1 - az^{-1})} \tag{1.10}$$

$|a|$ should be < 1 for the system to be stable.

Additional Properties of Z transform.

1. *Linearity:* If $x(n)$ and $X(z)$ are Z transform pairs (denoted as,

$$x(n) \leftrightarrow X(z)\ ) \text{ and } y(n) \leftrightarrow Y(z) \text{ then}$$

$$(\alpha\, x(n) + \beta\, y(n)\ ) \leftrightarrow (\alpha\, X(z) + \beta\, Y(z))$$

2. *Multiplication by exponential sequence:* If $x(n) \leftrightarrow X(z)$, then $a^n x(n) \leftrightarrow X(a^{-1} z)$.

3. *Initial and final value theorems:* The initial value $x(0)$ and final value $x(\infty)$ can be evaluated using the Z transform $X(z)$ as follows:

$$x(0) = \lim_{z \to \infty} X(z) \tag{1.11}$$

$$x(\infty) = \lim_{z \to 1} (z - 1)\, X(z) \tag{1.12}$$

4. *Differentiation of $X(z)$*: If $x(n) \leftrightarrow X(z)$, then $nx(n) \leftrightarrow -z\dfrac{d}{dz} X(z)$.

5. *Convolution:* The convolution sum $y(n)$ of two sequences $x(n)$ and $h(n)$ denoted as $x(n) * h(n)$ is given by

$$y(n) = \sum_{k=0}^{n} x(k)\, h(n - k) \tag{1.13}$$

If $x(n) \leftrightarrow X(z)$, $h(n) \leftrightarrow H(z)$ and $y(n) \leftrightarrow Y(z)$, then,

$$(x(n) * h(n)) \leftrightarrow X(z)\, H(z) = Y(z)$$

Similarly $$(a(n)\, b(n)) \leftrightarrow (A(z) * H(z))$$

6. *Shifting property:* If $X(z)$ is the Z transform of the sequence $x(n)$, then the Z transform of the sequence $x(n - k)$ is given by $X(z)z^{-k}$. This property can be used for solving a difference equation. For example, consider the output sequence $y(n)$ of a LSI system expressed in terms of the input sequence $x(n)$ by the difference equation

$$y(n) = x(n) + ay(n - 1) \tag{1.14}$$

Taking the Z transform of each of the terms of (1.14) and using the shifting property, $Y(z)$ is given by

$$Y(z) = X(z) + az^{-1}\, Y(z)$$

Rearranging the terms we get

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{(1 - az^{-1})} \tag{1.15}$$

The impulse response of the system can be found by taking the inverse Z transform of (1.15). In general Z transform can be used to convert a difference equation into an algebraic equation involving the Z transforms of the sequences.

7. The inverse Z transform of a sequence can be obtained either by using the power series expansion of $X(z)$ or by using partial fraction expansion.

For example, if $X(z)$ is given by (1.10) its power series expansion is given by

$$X(z) = 1 + (az)^{-1} + (az)^{-2} + (az)^{-3} + \cdots$$

$$= \sum_{n=0}^{\infty} (az)^{-n} \tag{1.16}$$

Comparing this with (1.9) it can be concluded that $x(n) = a^{-n}$.
Similarly if

$$H(z) = \frac{1}{(1-az^{-1})(1-bz^{-1})} \tag{1.17}$$

Using partial fraction expansion, $H(z)$ can be rewritten as

$$H(z) = \frac{a}{(a-b)} \frac{1}{(1-az^{-1})} + \frac{b}{(b-a)} \frac{1}{(1-bz^{-1})} \tag{1.18}$$

Comparing (1.18) with (1.10) it can be concluded that

$$h(n) = \frac{a}{(a-b)} a^{-n} + \frac{b}{(b-a)} b^{-n} \tag{1.19}$$

Z transform can be used to arrive at efficient schemes for implementation of discrete time systems. This is considered in Section 1.13.

1.10 FREQUENCY RESPONSE OF LTI DISCRETE TIME SYSTEM

The response $y(n)$ of a LTI discrete time system with impulse response $h(n)$ for a complex exponential sequence given by

$$x(n) = e^{jn\omega} \text{ for } -\infty < n < \infty$$

is given by

$$y(n) = \sum_{m=-\infty}^{\infty} h(m)\, e^{j(n-m)\omega} \tag{1.20}$$

Let $H(e^{j\omega})$ be defined as

$$H(e^{j\omega}) = \sum_{m=-\infty}^{\infty} h(m)\, e^{-jm\omega} \tag{1.21}$$

Using (1.21) in (1.20), $y(n)$ is given by

$$y(n) = e^{jn\omega} \sum_{m=-\infty}^{\infty} h(m)\, e^{-jm\omega} = e^{jn\omega} H(e^{j\omega}) = x(n)\, H(e^{j\omega}) \tag{1.22}$$

Hence the complex exponential sequence is the eigenfunction of the discrete LTI system. Since it corresponds to the sampled sinusoid of frequency ω, the response $H(e^{j\omega})$ is called the *frequency response* of the system.

Properties

1. $H(e^{j\omega})$ can be obtained from the transfer function $H(z)$ by evaluating it at $z = e^{j\omega}$. This can be verified by comparing (1.21) with (1.9).
2. $H(e^{j\omega})$ is periodic in ω with period equal to 2π. In the sampled data system, $y(n)$ actually denotes the value $y(t)$ at the nth sampling instant nT_s. The dependence of the sampling frequency on the frequency response can be made clear by replacing ω by ωT_s. Hence for the sampled data system with sampling interval T_s, the frequency response is given by $H(e^{j\omega T_s})$ and it is periodic with a period of $(2\pi/T_s)$. Since ω is equal to $2\pi f$, when the frequency response is plotted as a function of f, it is periodic with a period of $(1/T_s)$.
3. As $H(e^{j\omega})$ is periodic in ω with period of 2π, (1.21) can be viewed as the Fourier series expansion for $H(e^{j\omega})$ with the Fourier coefficients $h(n)$ given by

$$h(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(e^{j\omega})\, e^{j\omega n}\, d\omega \tag{1.23}$$

4. For real values of $h(n)$, $|H(e^{j\omega})|$, the magnitude of the frequency response is symmetric in the interval $(0, 2\pi)$ and the phase response is antisymmetric in this interval.

1.11 DIGITAL SIGNAL PROCESSING

As mentioned in Section 1.7, one of the advantages of discrete time signal is time division multiplexing. The second advantage of discrete time signal is that they can be digitised and can be processed either by using digital hardware or by using software. For the digitisation, first, the discrete time signal, which can take any value in the range $(-A_m, A_m)$ where A_m denotes the maximum amplitude of the signal, is approximated to one of the $2N$ levels that is closest to the signal. The approximation error is called the quantisation error and it can be made small by choosing N to be large.

The $2N$ levels may be chosen to be at equal intervals in the range $(-A_m, A_m)$ in which case the signal is said to be uniformly quantised. The next step is to represent each of these $2N$ levels by an n-bit number. The number of bits, n, is given by $\log_2 2N$. Processing the n-bit numbers corresponding to the various samples of the signal is called digital signal processing. It offers a number of advantages compared to processing the continuous time signal directly. The latter approach is also called analog signal processing if the signal amplitude range is also continuous.

1.12 ADVANTAGES OF DIGITAL SIGNAL PROCESSING (DSP)

1.12.1 Ease of Processing

One of the requirements in signal processing is to delay the signal by a particular duration. For example, in moving target indicator radar, a number of pulses are transmitted one after another and the received signal corresponding to adjacent pulses is to be subtracted. This requires the received signal to be delayed by one pulse repetition interval. For analog signal processing, this is achieved using acoustic delay line. Increasing or decreasing the delay requires the delay line length to be changed, which is cumbersome. On the other hand, for DSP, the samples of the received signal can be stored in memory and can be read after one pulse repetition interval. Delay can be easily changed without switching in or switching out cables.

1.12.2 Thermal Drift and Reliability

For analog signal processing, circuits consisting of analog components such as resistors capacitors and op amps are used and their characteristics change with temperature. DSP circuits use digital hardware such as adders, multipliers and shift registers whose characteristics show no variation with temperature throughout their operating range. Component aging alters the performance of the analog circuit. For example, the dielectric material of capacitors is particularly prone to aging, which changes the impedance and alters the performance. DSP circuits have the same performance throughout their life time.

1.12.3 Repeatability

Component tolerance makes the analog circuit to have different characteristics with different set of components of same nominal value. For example, a 100-Ω resistor with 10% tolerance can have any value in the range 90–110 Ω. Accordingly the circuit behaviour cannot be exactly predicted. This problem can be minimised by using components with smaller tolerance. But this increases the system cost as these components are costly. Typical capacitors have a tolerance of 20% or worse. This makes the characteristics of an analog circuit to be poorly repeatable with a new set of components with the same nominal values. Digital circuits, however, are inherently repeatable.

1.12.4 Immunity to Noise

In analog signal processing, the signals are allowed to take any value within a particular range and noise can easily alter the magnitude. In DSP, the signals take binary values and for altering a 1 to 0 and vice versa a noise voltage of a large magnitude is required. In digital data transmission, the effect of noise can be completely eliminated by putting repeaters at suitable intervals. However, a repeater used with an analog signal will amplify both signal and noise and will be ineffective in combating the noise.

1.12.5 Programmability

DSP functions can be implemented either by using microprocessors/microcontrollers or by using programmable digital signal processors. This enables the same hardware configuration to be reprogrammed to perform a very wide variety of signal processing tasks by loading in different software. In analog case this would call for rewiring/resoldering.

1.12.6 Some Special Signal Processing Techniques

There are some signal processing techniques that cannot be performed by analog systems. Examples of these techniques are linear phase filters, notch filters, adaptive filters, data compression and error control coding. In control systems, an example is a deadbeat controller used when a very rapid settling time is required.

1.13 DSP IN THE SAMPLE AND TRANSFORM DOMAIN

For a discrete time system, the convolution integral given by Eq. (1.1) reduces to the convolution sum given by

$$y(n) = \sum_{k=0}^{n} x(k)\, h(n-k) \tag{1.24}$$

where $y(n)$, $x(n)$ and $h(n)$ are the nth sample of output, input and impulse response of the LTI causal discrete time system and $x(n)$ and $h(n)$ are assumed to be 0 for $n < 0$. If $x(t)$ and $h(t)$ are of finite duration they may be represented faithfully using M and N samples, respectively. In this case the output is also of finite duration and can be represented using $M + N - 1$ samples. In other words convolution of two sequences of length M and N results in a sequence of length $M + N - 1$. For finding $y(n)$, $n + 1$ multiplications and n additions are required. For large values of n, this may call for significant computational effort; $y(n)$ may be alternately computed using an indirect approach by using the transforms of the input and the impulse response. One of the transforms that may be used is the Discrete Fourier Transform (DFT). The DFT $X(k)$ of a sequence $x(n)$ of finite length M, for $k = 0, \cdots,$ etc. $M - 1$, is defined as

$$X(k) = \sum_{n=0}^{M-1} x(n)\, e^{-j(2\pi/M)\,kn} \tag{1.25}$$

The computation of the M DFT coefficients $X(0), \cdots, X(M-1)$ requires M^2 complex multiplications. However, using the Fast Fourier Transform (FFT) algorithm, the number of multiplications required is reduced to $M \log_2 M$.

The inverse DFT (IDFT) is used to find $x(n)$ from $X(k)$ using the expression given by

$$x(n) = \frac{1}{M} \sum_{k=0}^{M-1} X(k)\, e^{j(2\pi/M)kn} \tag{1.26}$$

IDFT can also be computed using FFT.

The convolution can be done using DFT as follows: Let the sequences $x(n)$ and $h(n)$ be of length M, N respectively. New sequences $x'(n)$ and $h'(n)$ are formed by appending $N-1$ and $M-1$ zeros to $x(n)$ and $h(n)$, respectively at their end. The DFT of both of these sequences are found and the corresponding DFT coefficients are multiplied and a new set of coefficients are obtained. That is $X'(0)$ is multiplied with $H'(0)$, $X'(1)$ is multiplied with $H'(1)$ and so on. The inverse DFT of these coefficients gives the convolution of the sequence $x(n)$ with $h(n)$. If FFT is used both for DFT and IDFT, the total number of multiplications required for convolution becomes $(M + N - 1)\,[\,3\log_2(M + N - 1) + 1]$. Using the direct approach the number of multiplications required is $(M + N - 1)M$. For small values of M and N the direct approach is computationally efficient. For large values, the DFT approach is efficient. For example, for M=N=8, the number of multiplications required using direct and DFT approaches are 120 and 185, respectively. For M=N=64, they are 8128 and 2794, respectively.

1.14 FAST FOURIER TRANSFORM (FFT)

The direct computation of DFT of a sequence of length N requires N^2 complex multiplications. The number of multiplications can be reduced to $N \log_2 N$ by using the FFT algorithm. There are two popular FFT algorithms: decimation in time (DIT) algorithm and decimation in frequency (DIF) algorithm. These two algorithms are dealt in detail in the literature. For the sake of brevity, one of the algorithms, DIT, is presented here.

1.14.1 Decimation In Time Algorithm

The first step in this algorithm is to rearrange the sequence in the bit reversed order. In the bit reversed number representation, the binary pattern corresponding to a particular decimal number is obtained by writing the natural binary equivalent of the number in the reverse order so that the most significant bit of the natural binary number becomes the least significant bit of the bit reversed number and vice versa. Using this rule, it can be verified that the binary equivalent of the numbers 0–15 is as shown in Table 1.1.

Table 1.1 Natural binary numbers and their bit reversed equivalents

Decimal number	*Natural binary number*	*Bit reversed number*
0	0000	0000
1	0001	1000
2	0010	0100
3	0011	1100
4	0100	0010
5	0101	1010
6	0110	0110
7	0111	1110
8	1000	0001
9	1001	1001
10	1010	0101
11	1011	1101
12	1100	0011
13	1101	1011
14	1110	0111
15	1111	1111

Let the input sequence be $x(0), x(1), \cdots, x(N-1)$. Let the bit reversed sequence be $x_1(0), x_1(1), \cdots, x_1(N-1)$. The computation of FFT involves $\log N$ stages of computation.

In the first stage, 2 point DFT of blocks of every consecutive two elements of $x_1(0), \cdots, x_1(N-1)$ is computed and the resulting sequence is denoted as $X_1(0), X_1(1), \cdots, X_1(N-1)$ as shown in Fig. 1.3. The manner in which $X_1(k)$ is obtained from $x_1(l)$, for $k, l = 0, 1, \cdots, N-1$ can be explained using the FFT butterfly diagram shown in Fig. 1.2. Let W_N be given by

$$W_N = e^{-j(2\pi/N)} \tag{1.27}$$

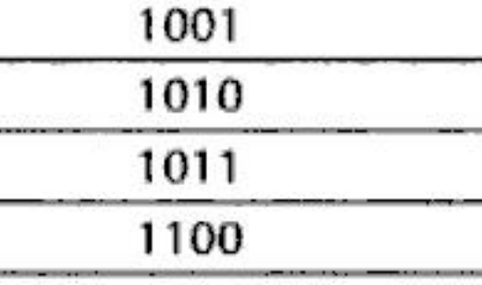


Fig. 1.2 *FFT butterfly with a twiddle factor of* W_N^k

The multiplication factor W_N^k is called the *twiddle factor*. For the 2 point DFT, the value of N is k and hence the twiddle factor is equal to one.

In the second stage, every consecutive block of four elements of $X_1(0)$, $X_1(1)$, $\cdots$, $X_1(N-1)$ is combined using two FFT butterflies. For $n = 0, 1, \cdots, (N/4 - 1)$, the first butterfly is formed between the $4n$th element and $(4n + 2)$th element. The second butterfly is formed between the $(4n + 1)$th element and $(4n + 3)$th element. The twiddle factor for the first butterfly is W_N^0. The twiddle factor for the second butterfly is $W_N^{N/2}$.

For example, for $N = 8$, the second twiddle factor becomes W_8^4. Let the output of the butterflies at the second stage be $X_2(0), X_2(1), \cdots, X_2(N-1)$.

In the third stage, every consecutive block of eight elements of $X_2(0), X_2(1), \cdots, X_2(N-1)$ is combined using four FFT butterflies. For $n = 0, (N/8 - 1)$, the first butterfly is formed between the $8n$th element and $(8n + 4)$th element. The second butterfly is formed between the $(8n + 1)$th element and $(8n + 5)$th element. The third butterfly is formed between the $(8n + 2)$th element and $(8n + 6)$th element. The fourth butterfly is formed between the $(8n + 3)$th element and $(8n + 7)$th element. The twiddle factor for the kth butterfly (for $k = 1, 2, 3, 4$) is obtained as $W_N^{(k-1)N/4}$. For example, for $N = 8$, the four twiddle factors are W_8^0 , W_8^2 , W_8^4 , W_8^6 respectively. Let the resulting output of the butterflies be $X_2(0), X_2(1), \cdots, X_2(N-1)$.

Table 1.2 Twiddle factors for the FFT bufferflies for $N = 4, 8, 16$

	$N = 4$	$N = 8$	$N = 16$
Stage 2 Twiddle factor	$1, W_4^2$	$1, W_8^4$	$1, W_{16}^8$
Stage 3 Twiddle factor		$1, W_8^2$ W_8^4, W_8^6	$1, W_{16}^4$ W_{16}^8, W_{16}^{12}
Stage 4 Twiddle factor			$1, W_{16}^2$ W_{16}^4, W_{16}^6 W_{16}^8, W_{16}^{10} W_{16}^{12}, W_{16}^{14}

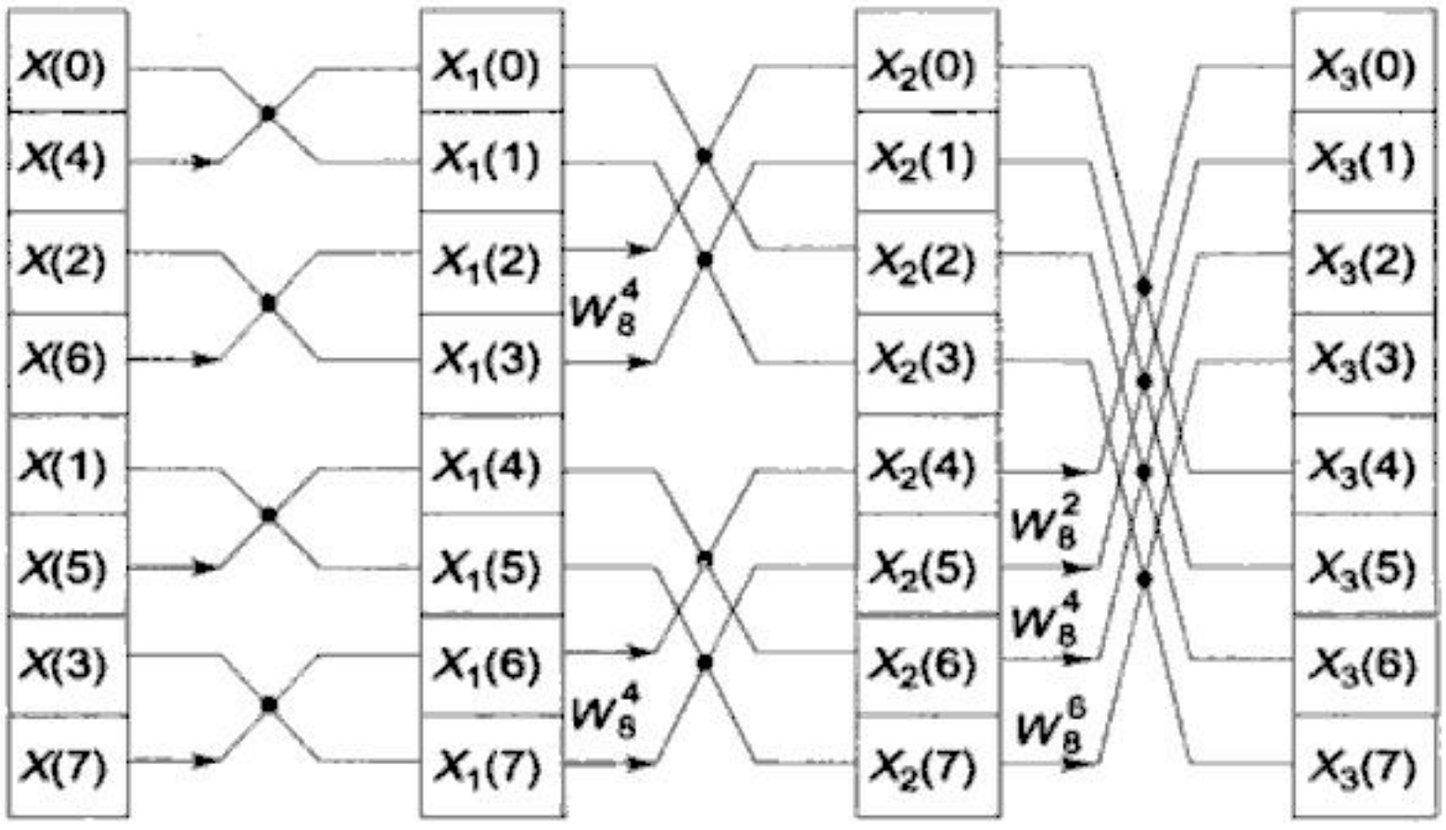


Fig. 1.3 *Decimation-in-time FFT flow diagram for 8-point FFT*

The procedure can be generalised as follows: At the rth stage of FFT computation, every block of consecutive 2^r elements of the output of the previous butterflies $X_{r-1}(0), X_{r-1}(1), \cdots, X_{r-1}(N-1)$

are combined using 2^{r-1} butterflies. For $m = 0, 1, \cdots, (N/2^r - 1)$, the kth butterfly is formed between the $(2^r m + k)$th element and $(2^r m + k + 2^{r-1})$th element.

The twiddle factor for the kth butterfly is given by,

$$W_N^{(k-1)N/p}, \text{ where } p = 2^{r-1}.$$

Twiddle factors at various stages of FFT computation for $N = 4, 8, 16$ are given in Table 1.2. The complete FFT butterfly flow diagram for $N = 8$ is given in Fig. 1.3.

1.15 DIGITAL FILTERS

In Section 1.3, the uses of low pass and high pass filters are illustrated. The flatness of the amplitude response in the pass band and the sharpness of the transition from pass band to stop band determines the complexity of the filter or the number of components required for building the filter. Digital filters use the digital hardware such as adder, multiplier and shift registers. The design of digital filters using the transfer function/impulse response of the analog filters has been dealt in detail in several textbooks. (see, e.g., Rabiner, Oppenheim). The digital filters can be classified broadly into two types: finite impulse response (FIR) filters and infinite impulse response (IIR) filters. FIR and IIR filters described below are examples of linear shift invariant system (LSI).

1.15.1 FIR Filters

The output of the FIR filter at the nth sampling instant $y(n)$, can be expressed as the function of the present as well as the past $(M-1)$ input samples and the impulse response sequence $h(k)$ as follows:

$$y(n) = \sum_{k=0}^{M-1} x(n-k)\, h(k) = \sum_{k=0}^{M-1} x_{n-k} h_k \tag{1.28}$$

This is a difference equation of order M. It is also referred to as tapped delay line filter with M taps or FIR filter with M taps. Comparing (1.28) with (1.13), it can be seen that FIR filter is a convolver where the present as well as past $M-1$ input samples are convolved with M impulse response coefficients. Hence, it can also be called a convolver. It may be noted that the past $M-1$ samples can be obtained by storing them in $M-1$ shift registers. Each shift register introduces a delay equal to 1 sampling interval and has a transfer function of z^{-1}. With this observation a structure for the implementation of the FIR filter can be obtained as shown in Fig. 1.4. This is also referred to as the direct implementation scheme as it uses M multipliers and $(M-1)$ adders for M products. In this case the sampling interval T_s is given by,

$$T_S = T_M + MT_A$$

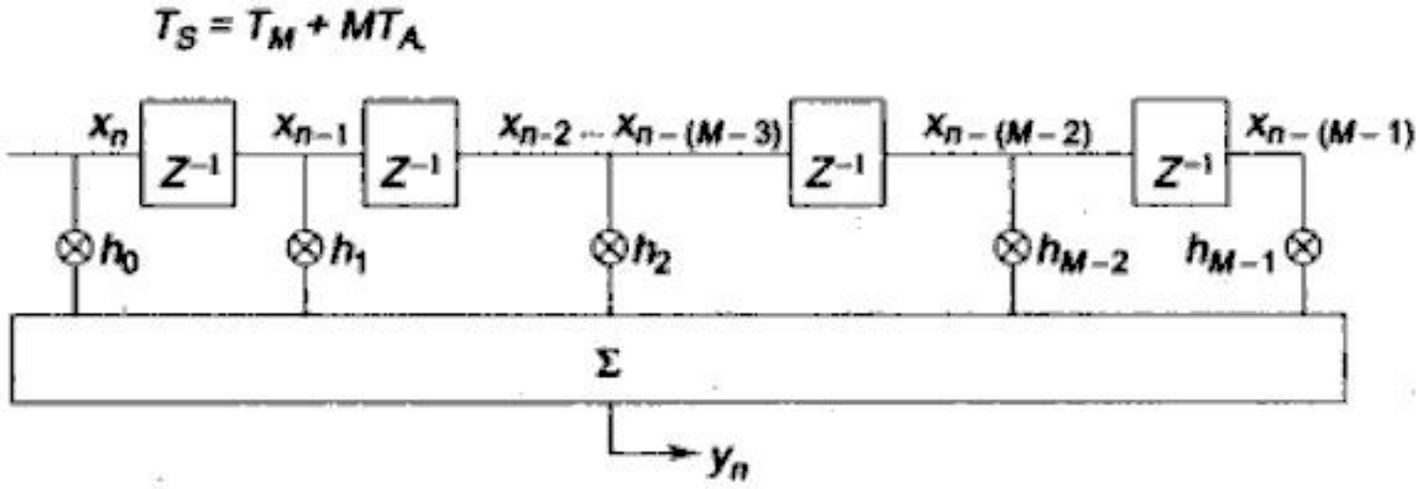


Fig. 1.4 *Direct implementation scheme for FIR filter*

where T_M and T_A are the time required for one multiplication and addition, respectively. The sampling time can be reduced to be $T_M + T_A$ by using the transpose of the filter structure given in Fig. 1.4. The transpose of a filter/network is obtained as follows: The direction of the signal flow in each branch of the filter is reversed. The branching points are replaced by adders and adders are replaced by branching nodes. The nodes at which the input is fed and the output is tapped are interchanged. The transfer function of the transpose structure is the same as that of the original structure. However, their other characteristics such as the effect of inaccuracy due to finite number of bits used for representing the numbers in a filter (referred to as the finite word length effect) will be different. The transpose of the filter given in Fig. 1.4 is given in Fig. 1.5. It can be verified that for this filter the minimum sampling period can be $T_M + T_A$. This structure can also be used for serial/parallel convolution as in this case the input data can be fed serially bit by bit and the output can also be obtained serially.

For implementation either in software or hardware another scheme that can be used is shown in Fig.1.6. This consists of two circulating registers for storing and circulating M numbers, a multiplier and an accumulator consisting of a register and an adder.

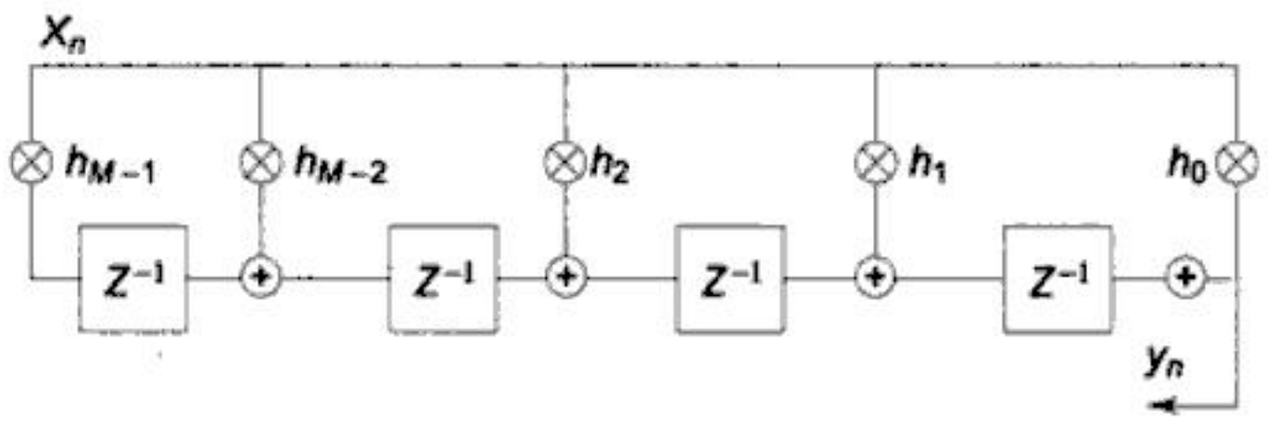


Fig.1.5 *Serial/Parallel convolver*

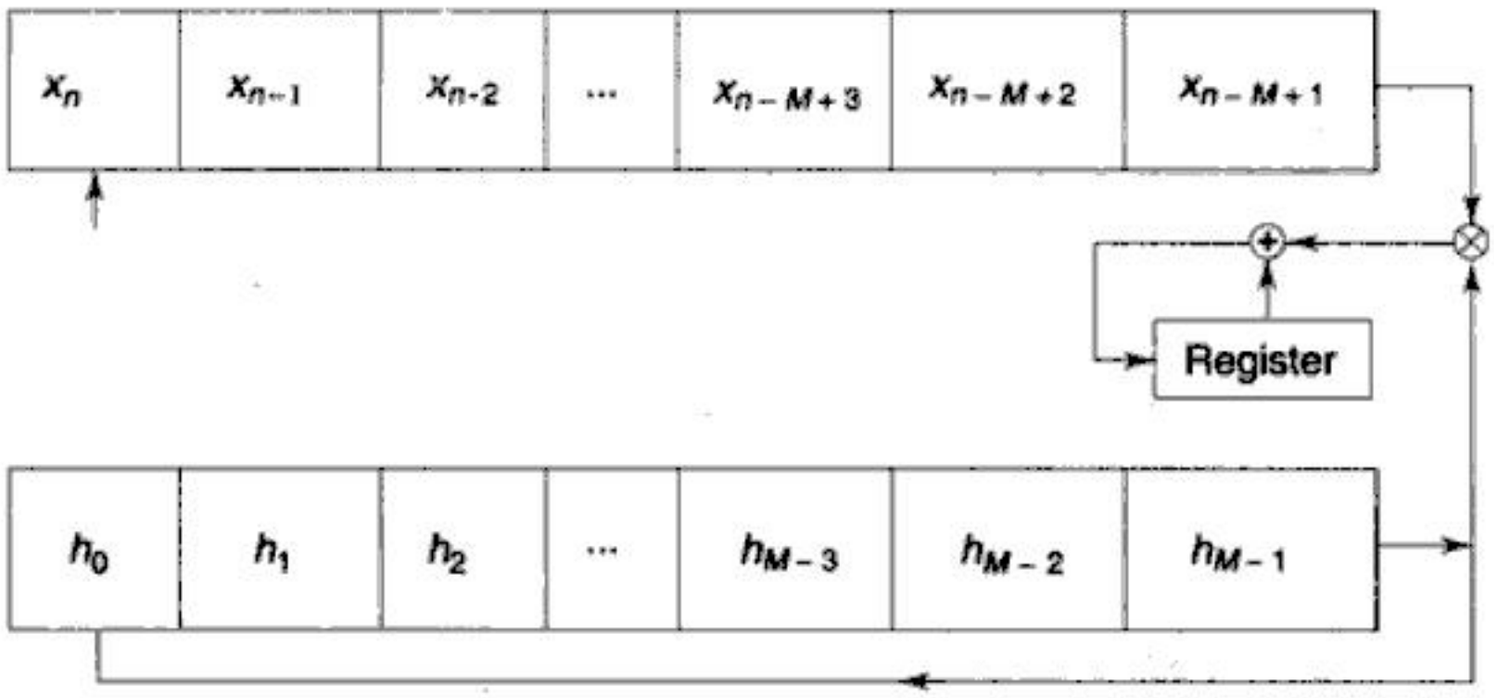


Fig.1.6 *Direct implementation scheme with single multiplier/adder*

1.15.2 Design of FIR Filters

The FIR filter may be specified by the frequency response of the filter. This in turn can be specified by the magnitude response and the desired phase response of the filter. A number of techniques have been developed for finding the filter coefficients, and computer programs have also been developed to facilitate this. Overview of two of the methods, the window method and frequency sampling method, is presented next.

1.15.3 Window Method

Using the periodicity property of the frequency response, it was shown in Section 1.10 that $h(n)$ the impulse response coefficients can be obtained as the Fourier coefficients given by Eq. (1.23). However, this requires an infinite sequence and the filter will be non-causal as $h(n)$ is non-zero for $n < 0$. This series may be truncated to have N coefficients. However this will result in overshoot and ripples in both passband and stopband and is referred to as *Gibb's oscillation* or phenomena. Another problem with the fourier series technique is its slow convergence. Choosing larger values of N does not automatically reduce the magnitude of the ripples in the transition region from passband to stop band significantly. This problem is overcome by choosing a window sequence $\omega(n)$ and multiplying it with $h(n)$ obtained using the Fourier series approach. $\omega(n)$ is chosen so as to obtain a smooth transition from the passband to the stopband and to be non-zero for $0 \le n \le N-1$. The resulting $h_\omega(n)$ is made causal by shifting it by $(N-1)/2$ towards right. Some of the window functions used are the rectangular window, Von Hann (also referred to as Hanning) window, Hamming window, Blackman window and Kaiser window. The expressions for the first three windows can be specified using the parameter α as follows:

$$\omega(n) = \alpha + (1-\alpha)\cos(2\pi n/N-1) \quad \text{for } 0 \le n \le N-1 \qquad (1.29)$$
$$= 0 \quad \text{otherwise.}$$

The value of α is 1, 0.5, 0.54 respectively, for rectangular, Hanning and Hamming windows. The Blackman window function is also of similar form and is given by

$$\omega(n) = 0.42 - 0.5\cos(2\pi n/N-1) + 0.08\cos(4\pi n/N-1) \quad \text{for } 0 \le n \le N-1$$
$$= 0 \quad \text{otherwise.} \qquad (1.30)$$

The frequency response of the window functions have a main lobe and a number of side lobes. The ripple ratio defined as the ratio of the first side lobe amplitude to the main lobe amplitude in the frequency response of the window function increases with α and the main lobe width decreases with α.

1.15.4 Frequency Sampling Method

In this method the value of the frequency response at M points in the interval $(0, (2\pi/T_s))$ is taken. In other words $H(e^{j\omega T_s})$ is evaluated for $\omega = (2\pi k/MT_s)$ for $0 \le k \le M-1$ and are taken to be M DFT coefficients. The inverse DFT of this sequence is taken and is treated as M impulse response samples $h(n)$. Out of these M samples N samples $(N < M)$ are taken to be the impulse response sequence for the FIR filter with N taps. To minimise the Gibb's oscillations, this is combined with one of the window functions discussed in the last section. The choice of the value of N may be done iteratively. Starting with a value of N, the filter frequency response may be evaluated after windowing. If it differs significantly from the desired frequency response, the value of N may be changed. This step is repeated till a satisfactory response is obtained. Several variations of the frequency sampling method is used in practice. For example, instead of choosing all the M DFT coefficients using the frequency response, some of the coefficients in the transition region from pass band to the stop band may be chosen by using optimisation techniques in order to achieve the required ripple as well as undershoot/overshoot characteristics.

FIR filters have a number of advantages: They are stable and can be made always realisable with suitable delays. The inaccuracy in the output due to the number of bits used for representation of the input samples and the impulse response sequence is smaller in FIR filters than in IIR filters. The

inaccuracy that results in both FIR and IIR filters due to the number of bits used for the representation is referred to as *finite wordlength effect*. For speech processing and data transmission, it is required to have frequency selective filters whose magnitude response is flat and the phase response is linear with frequency in the pass band. Such filters are referred to as *linear phase filters* and they can be realised using FIR structure. In this case, the impulse response has even symmetry. For example, the impulse response of the N-tap linear phase FIR filter satisfies

$$h(n) = h(M - 1 - n) \quad \text{for } 0 \leq n \leq (M/2) \tag{1.31}$$

For example, in an 8-tap linear phase FIR filter, the impulse response coefficients satisfy the following conditions:

$$h(7) = h(0),\ h(6) = h(1),\ h(5) = h(2),\ h(4) = h(3) \tag{1.32}$$

This reduces the number of multiplications required for computing (1.28) by half. In hardware implementation of the filter the number of multipliers required is reduced by half.

Another advantage of the FIR filter is that an FIR filter can be designed using well proven techniques for any arbitrary frequency response. IIR filters can be designed only for four basic types such as Low pass (LP), High pass (HP), Band pass (BP), Bandstop (BS) and few others. More general filters such as multiband filters are difficult to design using IIR filters but can be designed as FIR filters.

One of the limitations of FIR filters is the need for large number of taps for obtaining filters with sharp cutoff characteristics. IIR filters require less number of taps.compared to that required by FIR filters. More number of taps require more hardware resources or more computation time. Another disadvantage is that the linear phase filter may result in non-integral number of sample delays with frequency and it may not be acceptable in some applications.

1.15.5 IIR Filters

The output of the IIR filter at the nth sampling instant $y(n)$ can be expressed as the function of the present input and past $M - 1$ input samples as well as the past N output samples and is given by:

$$y(n) = \sum_{k=0}^{M-1} x(n-k)\, a(k) - \sum_{k=1}^{N} y(n-k)\, b(k) \tag{1.33}$$

where $a(k)$ and $b(k)$ are weight factors for the inputs and past outputs. For ease of notation, the samples are represented using suffixes wherever convenient. In terms of this notation, (1.33) can be rewritten as

$$y_n = \sum_{k=0}^{M-1} x_{n-k}\, a_k + \sum_{k=1}^{N} y_{n-k}\, b_k \Rightarrow \sum_{k=0}^{N} x_{n-k}\, a_k + \sum_{k=1}^{N} y_{n-k}\, b_k \tag{1.34}$$

Since the present output depends on the past outputs, IIR filter is also called a *recursive filter*. The Z transform of (1.34) can be used to arrive at different structures for implementation of the filter. For $M = N$ and $b(0) = 1$, the transfer function $H(z)$ of the filters given by (1.33) and (1.34) is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{A(z)}{B(z)}$$

Structure for implementation of the filter given by (1.33) – (1.34) is shown in Fig.1.7. It is called the direct implementation structure. It can be verified that the top set of adders and delay units with transfer function z^{-1} realises the function $A(z)$ and the bottom units realise the function $1/B(z)$. Transpose of this filter is shown in Fig.1.8. Both of these filter structures require $2N$ delay units, $2N$

adders and $2N + 1$ multipliers. The number of delay units can be reduced by realising $1/B(z)$ first and feeding the output of this filter to the filter with transfer function $A(z)$ as shown in Fig. 1.9. Since the delay units in both of these filters have the same inputs, a single delay unit can be shared by both of the filters with the transfer function $1/B(z)$ and $A(z)$. The resulting filter structure is given in Fig.1.10. A digital filter is said to be canonic if the number of delay units is equal to the order of the transfer function. Since in the above case the order of the transfer function is N and the number of delay units used in Fig. 1.10 is N, the filter given by Fig.1.10 represents a canonic realisation. Several other realisations can be obtained by factorising $H(z)$ or writing it as a sum of a number of tranfer functions.

Accordingly a cascade realisation and parallel realisation of the filters are obtained. This is facilitated by writing (1.33) in the form given by (1.35).

$$= \frac{(a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_{N-1} z^{-(N-1)} + a_N z^{-N})}{(1 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_{N-1} z^{-(N-1)} + b_N z^{-N})} \tag{1.35}$$

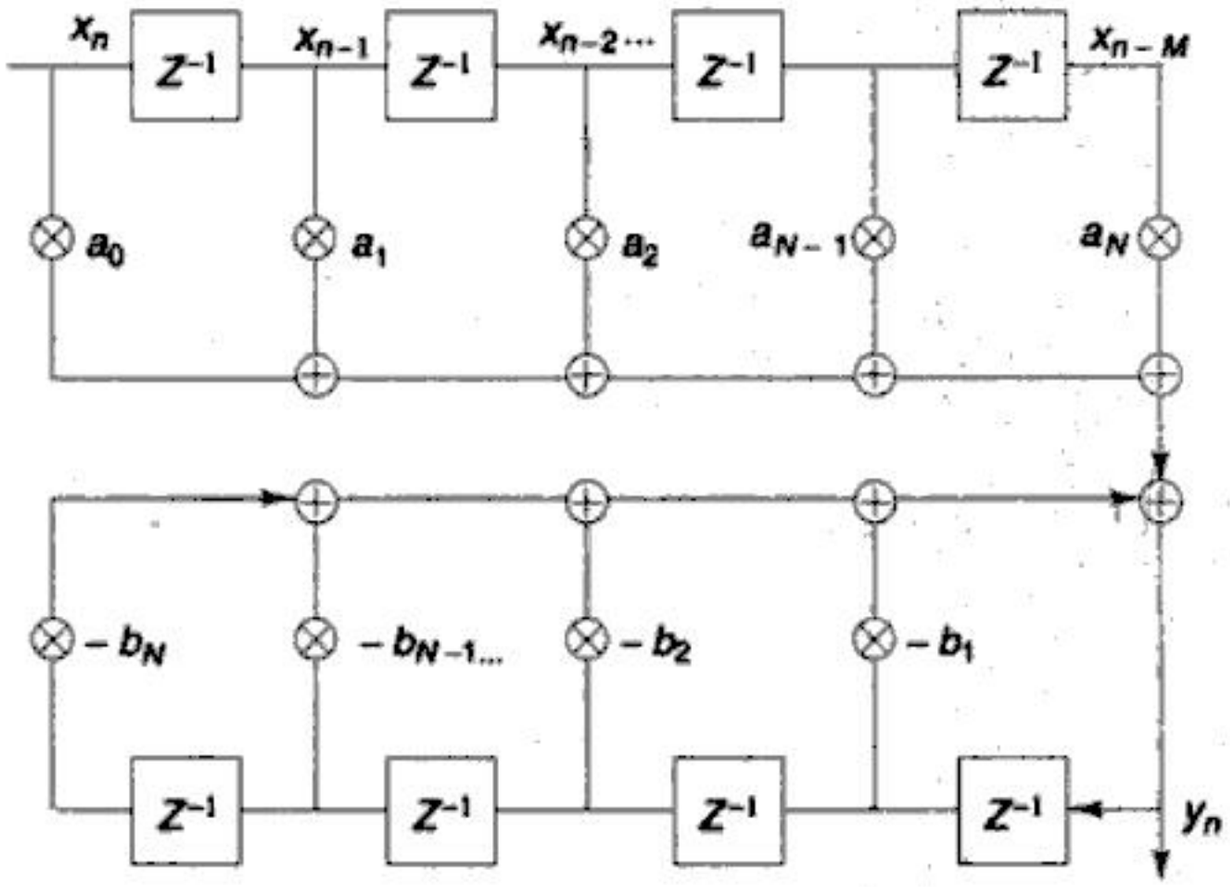


Fig. 1.7 *Direct implementation structure for an IIR filter*

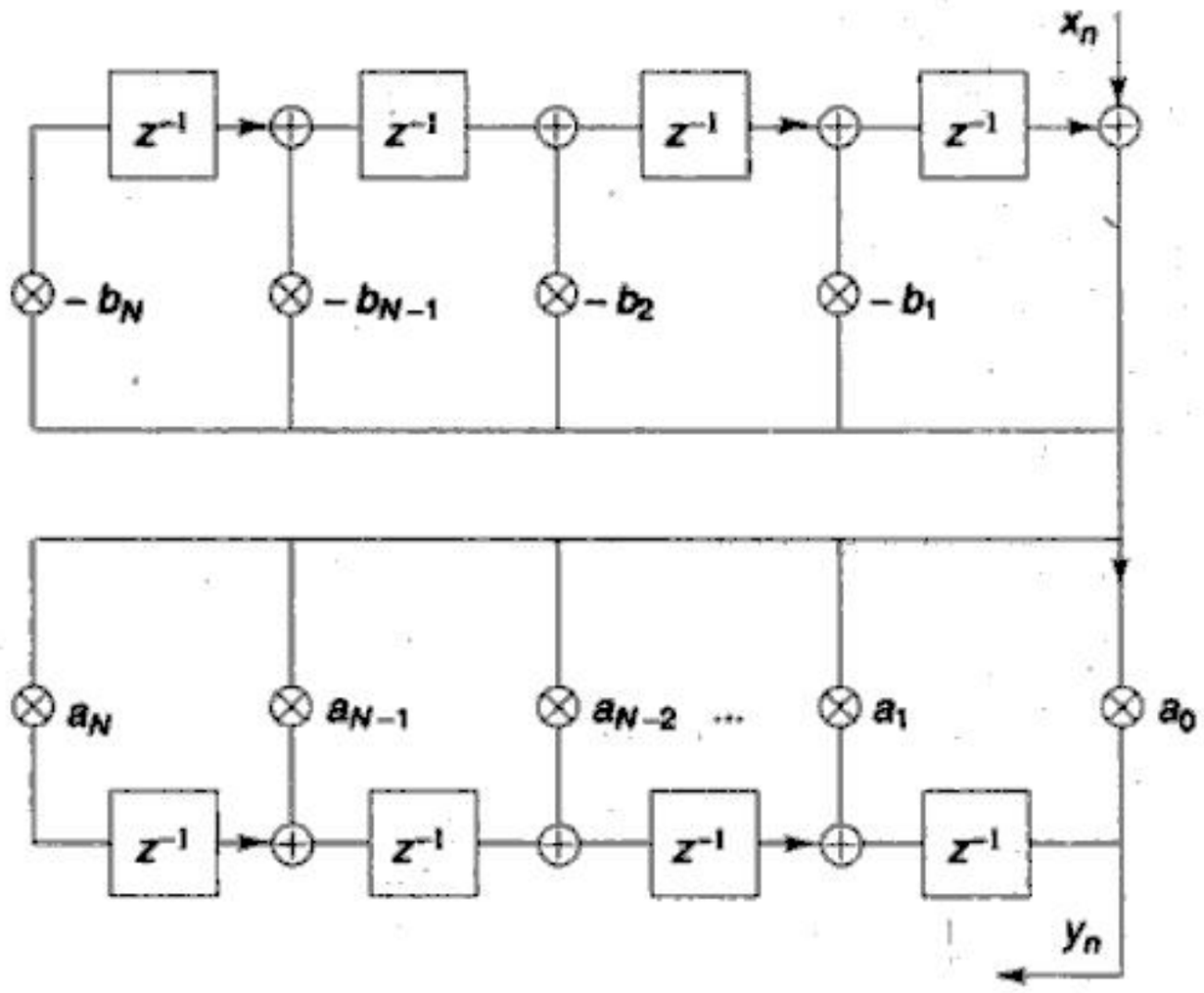


Fig. 1.8 *Direct implementation scheme for the IIR filter using the transpose structure*

As mentioned earlier an IIR requires less hardware and computational effort compared to an FIR filter for the same sharpness and flatness in the filter. However, these filters may not always be realisable and may become unstable and are affected more by the finite wordlength effects. When an IIR scheme exists for an FIR filter, then it requires less memory and less computation time compared to the latter. For example, consider an FIR filter with M coefficients in which the nth coefficient is given by

$$h(n) = a^{-n} \tag{1.36}$$

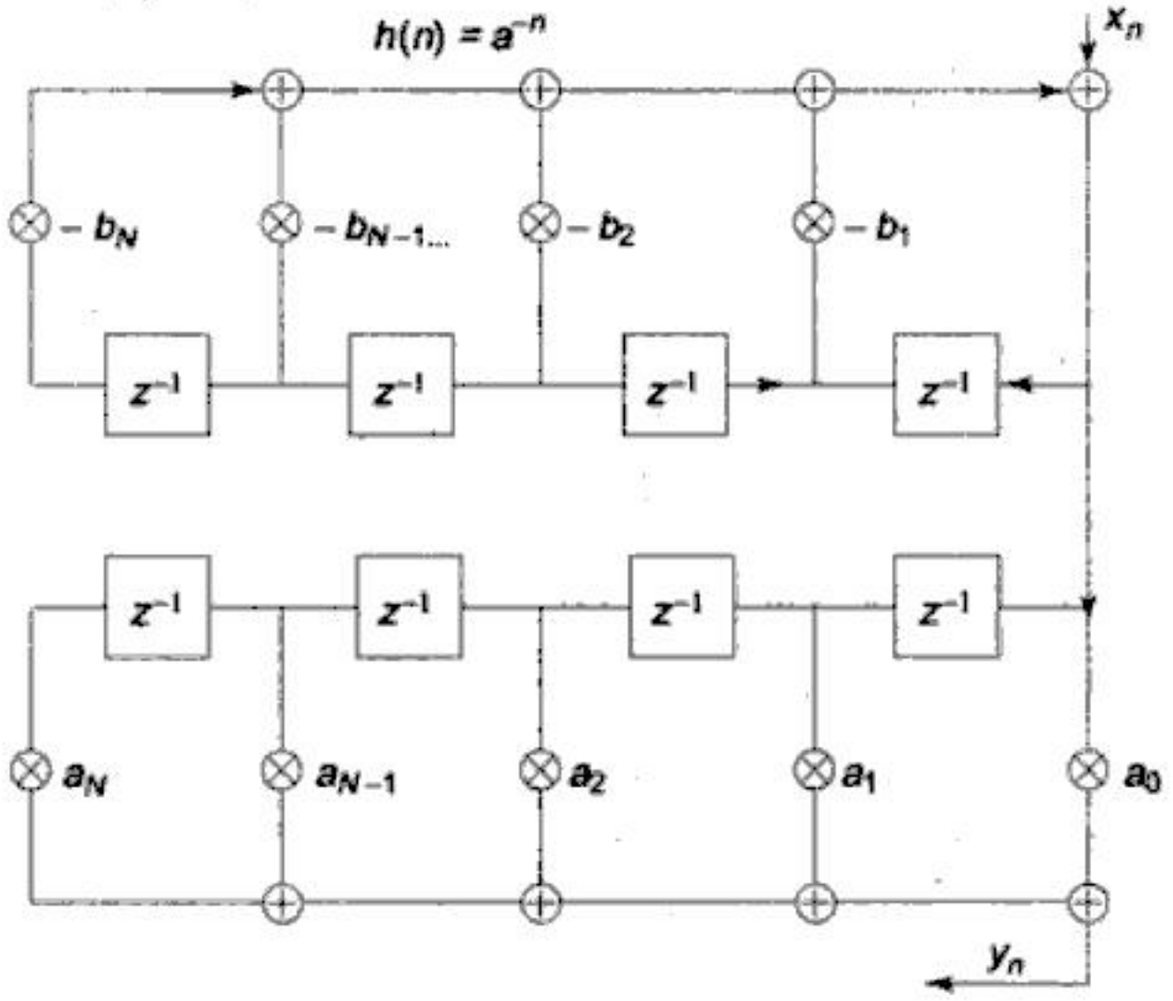


Fig. 1.9 *An alternate implementation scheme for the IIR filter*

The transfer function of this filter can be written as

$$H(z) = 1 + (az)^{-1} + (az)^{-2} + (az)^{-(M-1)} + (az)^{-M} \tag{1.37}$$

$$= \frac{1-(az)^{-(M+1)}}{1-(az)^{-1}} \tag{1.38}$$

Equation (1.37) represents an FIR filter and it requires M delay units, M adders as well as multipliers. Equation (1.38) represents an equivalent IIR filter. This requires $M + 2$ delay units, two adders and two multipliers.

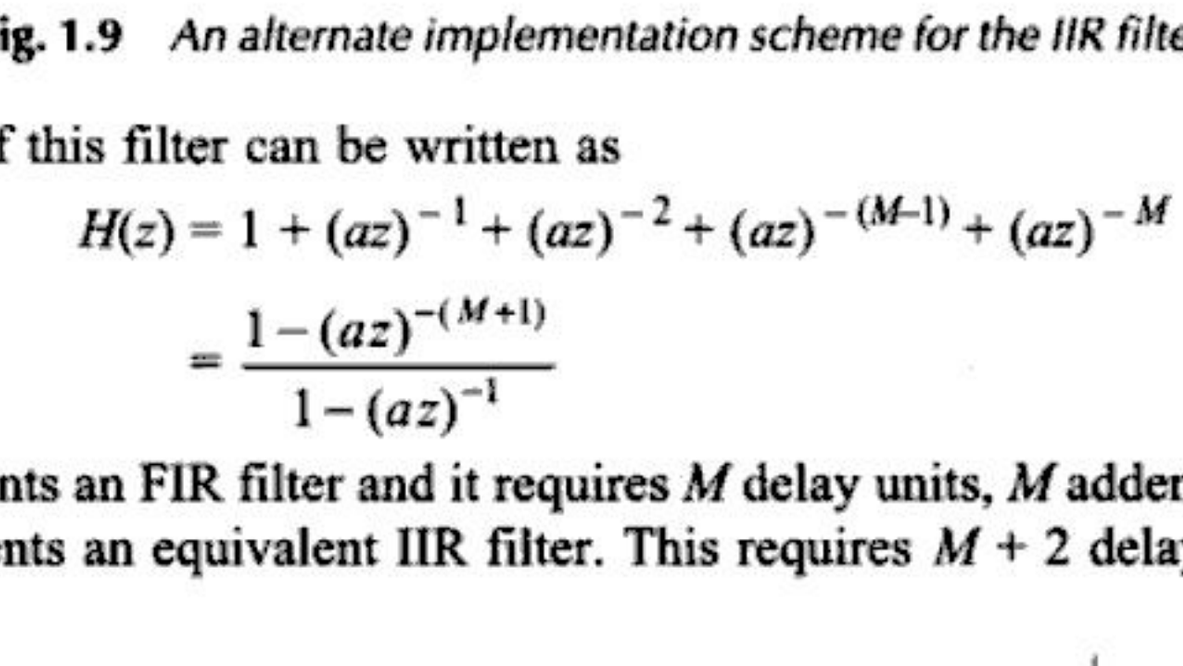


Fig. 1.10 *Canonic realisation scheme for the IIR filter*

1.15.6 Design of IIR Filters

Design of analog filters for LP, HP, BP and BS has been studied in detail in the past and has been implemented in a large number of communication and consumer electronic circuits. The impulse response of the IIR filters can be obtained from the corresponding analog circuits commonly using one of the following techniques: impulse invariance method and the bilinear transform method. These two techniques are discussed in brief next.

1.15.7 Impulse Invariance Method

In this method a digital IIR filter with impulse response $h(n)$ is obtained from the analog filter with impulse response $h_a(t)$ by evaluating $h(n)$ as

$$h(n) = h_a(t) \quad \text{for } t = nT_s\,; \quad n = 0,1,2\,\cdots \tag{1.39}$$

where T_s is the sampling interval. However, this method is applicable only if the analog filter response is bandlimited such that the frequency response $H(j\omega)$ is given by

$$H(j\omega) = 0 \quad \text{for } |\,\omega\,| > 1/2T_s \tag{1.40}$$

If this condition is not satisfied, aliasing occurs; that is, the signal with frequency f, in the frequency range $1/2T_s$ to $1/T_s$ gets mapped to $(f - 1/2T_s)$. Because of the periodic nature of the frequency response of the LTI digital filters, for m=1, 2, …, all the signals in the frequency range $(m/T_s, (2m + 1)/2T_s)$ gets mapped to the frequency range $(0, 1/2T_s)$ in the digital filters. The signal with frequency f in the range $((2m + 1)/2T_s, (2m + 2)/2T_s)$ appears as $f - 1/2T_s$ due to aliasing. Hence, even though this method is simple it is not applicable for filters whose response is not bandlimited. For example, this method is not suitable for the design of high HP filter. In practice, this method can be adopted if

$$H(j\omega) < 0.01\ H_{\max} \quad \text{for } |\,\omega\,| > 1/2T_s \tag{1.41}$$

where $H_{\max}$ denotes the maximum amplitude of the frequency response of the analog filter in the frequency range $(0, 1/2T_s)$.

1.15.8 Bilinear Transform Method

In this method the transfer function $H(z)$ of the digital filter is obtained from the corresponding analog filter transfer function $H(s)$ by using the mapping given by

$$H(z) = H(s) \text{ with}$$

$$s = \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}} \tag{1.41}$$

The relationship between the digital frequency ω and the analog frequency Ω can be obtained by substituting $s = j\Omega$ and $z = e^{j\omega T_s}$ in (1.41) as follows:

$$j\Omega = \frac{2}{Ts}\frac{1-e^{-j\omega T_s}}{1+e^{-j\omega T_s}} \tag{1.42}$$

$$\Omega = (2/T_s)\tan(\omega T_s/2) \tag{1.43}$$

In this case the upper half of the imaginary axis (or in other words the frequency range $(0, \infty)$) is mapped uniquely within the unit circle corresponding to the Z transform. Hence aliasing does not

occur in this case. However, this method introduces distortion in the frequency response of the digital filter designed. Specifically the phase response of this filter is distorted and hence a filter with linear phase characteristics cannot be obtained using the bilinear transform method.

This is because the bilinear transform results in non-linear mapping between the digital frequency ω and the analog frequency Ω as given by (1.43). This is called *frequency warping*.

To overcome this non-linearity, compensation has to be applied to the filter. One of the technique adopted is to prewarp the cutoff frequencies of the filter. This is called *prewarping*. For example, if a digital filter with the cutoff frequencies in the pass band as $\omega 1$, $\omega 2$, $\omega 3$, $\omega 4$ is desired, an analog filter with cutoff frequencies given by

$$\Omega_i = (2/T_s) \tan(\omega_i T_s/2) \quad \text{for } i = 1, 2, 3, 4 \tag{1.44}$$

is designed. After the warping given by (1.43) a digital filter with the desired cutoff frequencies is obtained. The prewarping can only compensate for the amplitude non-linearities and the phase non-linearities cannot be overcome.

1.16 FINITE WORDLENGTH EFFECT IN DIGITAL FILTERS

Because of the finite size of the registers used for storing the output of the processing elements such as adders and multipliers, overflow can occur in FIR filters if the results exceed the capacity of the registers. These results will normally be rounded off and this gives rise to round off noise. Overflow can be avoided by using floating point arithmetic. However, in filters using fixed-point computations, the overflow can be minimised by scaling the impulse response coefficients suitably.

In the case of IIR filters, because of the feedback connection, finite precision arithmetic can lead to oscillations that make the output to keep alternating between two values. Such oscillations are called *limit cycle oscillations*. This can occur when the input to the filter is zero, for example, during silence periods in voice communication. In this case the output keeps oscillating between two small values and is referred to as zero input limit cycles. Another type of oscillation called overflow limit cycle oscillation occurs when overflow occurs at the output due to the finite precision arithmetic. In this case output can keep oscillating with amplitudes equal to the full supply voltage. Another effect of finite wordlength is the quantisation noise. It can arise due to inadequate number of bits for representing the input to the filter, output of the filter or the output of the processing elements such as multipliers and adders.

1.17 MULTIRATE SIGNAL PROCESSING

Multirate signal processing refers to the techniques adopted for processing a digital signal by sampling it further either by using a higher rate than the rate at which the digital samples were obtained or by using a lower sampling rate. The sampler used for this purpose is accordingly called *upsampler* and *downsampler* respectively. The downsampler is also referred to as a *decimator* and a decimator that downsamples the input by a factor of M is denoted by the symbol given in Fig.1.11a. The upsampler is also referred to as an *interpolator* and an interpolator that increases the sampling rate by a factor of M is denoted by the symbol given in Fig. 1.11b. If $y(n)$ denotes the output of an M factor decimator of an input signal $x(n)$, the $y(n)$ can be written as

$$y(n) = x(Mn) \quad \text{for } n = 1, 2, 3, \cdots \tag{1.45}$$

Similarly if $y'(n)$ denote the output of an M factor interpolator of a signal $y(n)$, $y'(n)$ is given by

$$y'(n) = y(n/M) \quad \text{for } n = 1, 2, 3, \cdots \qquad (1.46)$$
$$= 0 \qquad \text{otherwise}$$

Fig. 1.11a *Downsampler*

Fig.1.11b *Upsampler*

Processing a signal after downsampling or upsampling has a number of advantages. To appreciate it let us consider the example, of a voice coder/decoder. When an analog signal is digitised using the sampler followed by the quantiser, the minimum sampling rate required is determined by the maximum frequency component in the signal and the number of bits used for quantisation is independent of the relative importance of the various frequency components that constitute the signal. Non-uniform quantisers allocate different step sizes for different amplitude ranges of the signal taking into account their probability of occurrence. Low amplitude signals occur more frequently and are more important. They are allocated small step sizes. Large amplitude signals are quantised with a large step size.

1.17.1 Subband Coding

Quantisation noise and the bit rate required for digitising an analog signal can be reduced further by examining their frequency content and allocating the number of bits per sample depending upon their importance. For example, the energy in the speech signal is concentrated more in the low frequency region of 0–1 kHz and is of decreasing concentration as the frequency is increased. Hence, more number of bits may be allocated in frequency bands where there is more energy concentration and less number of bits may be allocated where there is less energy concentration. This technique is called the *subband coding*. The first step required for subband coding is to separate the incoming signal into separate bands using LP, BP and HP filters. This may be done in the digital domain; that is, the analog signal may first be sampled and quantised without any regard to the frequency. The resulting digital signal may then be passed through the digital filters to separate the different band of frequencies. The individual band of frequencies may be individually processed further. However, they need not be processed at the rate at which the analog signal was sampled. This can be be verified using the sampling theorem for the BP signal.

1.17.2 Sampling Theorem For Band pass Signals And Its Application

For a BP signal with lower and upper cutoff frequencies of f_1, f_2, the Nyquist sampling rate f_s is $2(f_1 - f_2)$ if either f_1 or f_2 is a harmonic of f_s. Hence, if one of the subband is of bandwidth 0.5 kHz, it need not be sampled at 8 kHz for further processing but a sampling rate of 1 kHz may be just adequate. In practice, one may choose 2 kHz as the sampling rate. When the voice signal is sampled at 8 kHz rate, at the output of the filter corresponding to a subband, the samples depart at 8 kHz rate, but all of these samples are not required for processing. In the above case only 1/4th of the samples may be retained and the rest of them may be discarded. This is achieved using a downsampler.

At the receiver, the decoder has to combine the outputs from the individual subbands and deliver a single contiguous band. Even though the individual bands generate samples at a rate lower than 8 kHz the signal has to be delivered at the 8 kHz rate to the play out device. This is achieved by sampling the

output of the subband at a rate higher than the rate at which the sample arrive at its input. This is achieved by inserting zeros between the actual sample values. This process is achieved by the upsampler.

1.17.3 Spectrum of the Sampled Signal at the Output of the Downsampler and Upsampler

More insight into the downsampling and upsampling process can be gained by looking at the spectrum of the sampled signal and comparing it with the spectrum of the original signal. Let $H(\omega)$, denote the frequency spectrum of a signal bandlimited to f_m. $H_s(\omega)$ denotes the frequency spectrum at the output of sampler that samples the analog input at the rate of f_s samples/s. $H_d(\omega)$ denotes the frequency spectrum at the output of the downsampler that samples the output of the above sampler at the rate of f_d samples/s. In general $f_d = f_s/M$. For the case where $M = 2$, the spectrums of $H(\omega)$, $H_s(\omega)$ and $H_d(\omega)$ are shown in Figs 1.12a, 1.12b and 1.12c, respectively.

In Fig. 1.12b the frequency spectrum given in Fig.1.12a gets replicated at intervals of $f_s = 2f_d$. In Fig. 1.12c, it gets replicated at intervals of $f_d = f_s/2$. It can be verified from this figure that the original spectrum $H(\omega)$ and its replicas will be non-overlapping only if $f_m < f_d/2$. If $f_m > f_d/2$, then all the frequency components of $H(\omega)$ in the range $(f_d/2, f_m)$ gets converted to the frequency range $(0, f_m - f_d/2)$. In other words aliasing occurs and the original signal cannot be faithfully reproduced using the downsampler. To avoid aliasing either f_d should be greater than $2f_m$ or an antialiasing filter that bandlimits the input signal to $f_d/2$ should be used at the input to the downsampler. The antialiasing filter is also referred to as the *decimator filter*.

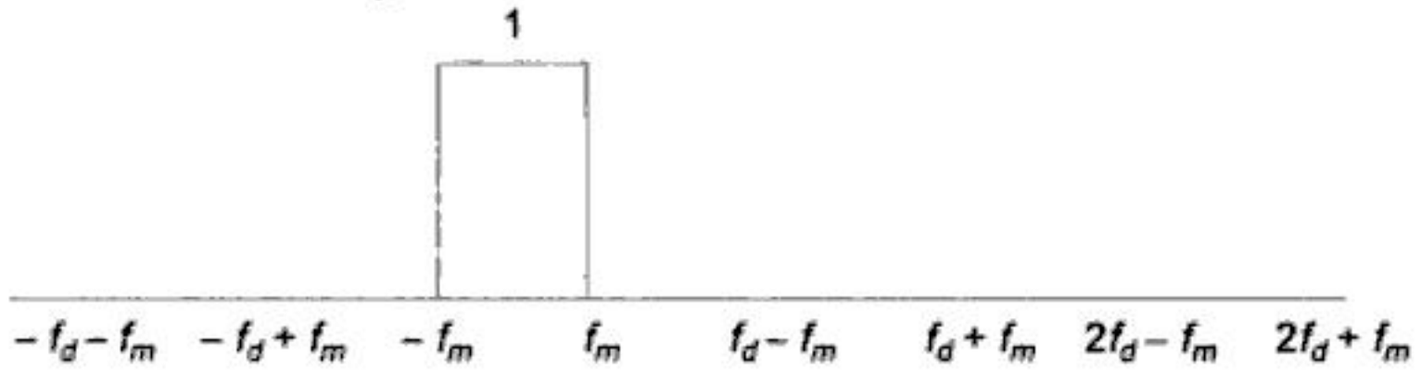


Fig. 1.12.a *Frequency spectrum $H(\omega)$ of the analog signal*

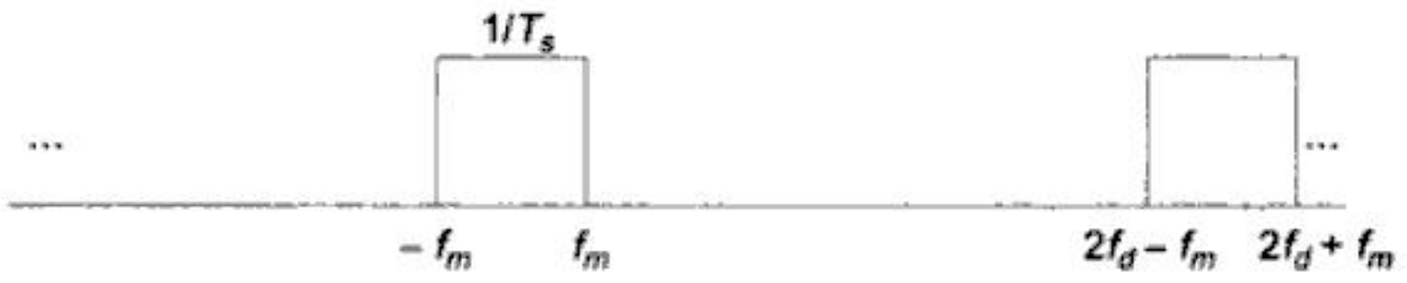


Fig. 1.12.b *Frequency spectrum $H_s(\omega)$ of the sampled signal*

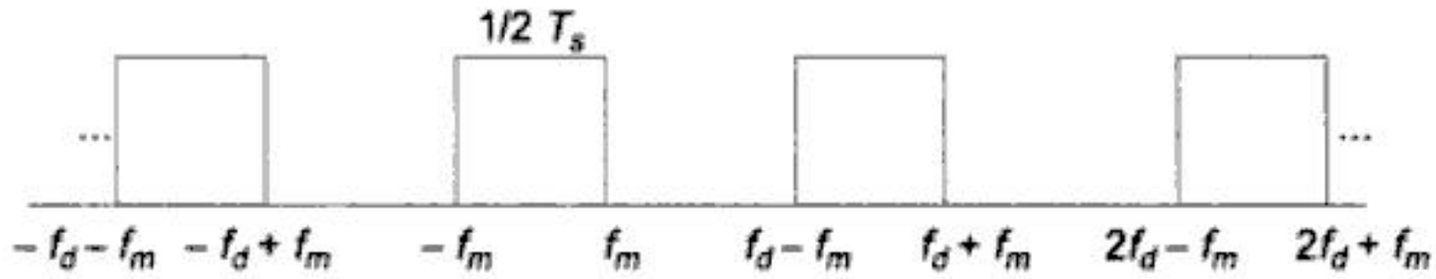


Fig. 1.12.c *Frequency spectrum $H_d(\omega)$ of the downsampled signal*

As mentioned earlier, the upsampler generates more samples by inserting zeros in between the actual samples. Insertion of zeros does not alter the frequency spectrum. Hence, the spectrum of the sampled signal at the output of the upsampler is the same as that at its input. Hence if the input is originally sampled at the rate of f_s, the spectrum at the output of the upsampler is the same as that

given in Fig. 1.12b. In order to regenerate the original analog signal, the upsampler should be followed with a LP filter with cutoff frequency of f_m where f_m denotes the highest frequency component in the analog signal. The LP filter used for this purpose is called the interpolator.

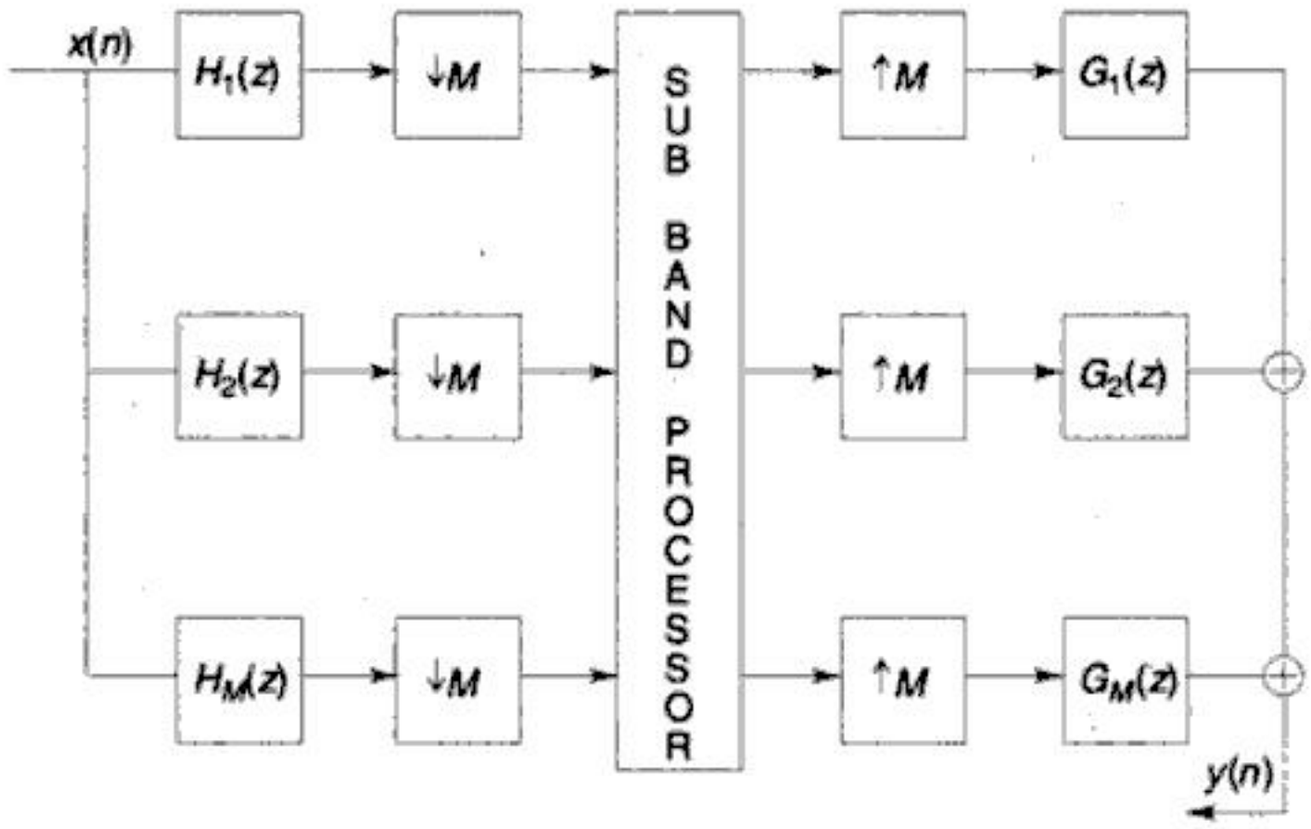


Fig.1.13 *A subband processor with filter banks*

1.17.4 Block Diagram of a Subband Processing System and Filter Banks

The block diagram of a subband processing system that processes M subbands individually using decimation by M and then upsampling by M is shown in Fig.1.13. The filters $H_1(z)$, $H_2(z)$, $\cdots$, $H_M(z)$ denote the filters used for separating the M bands and are called analysis filter banks. When the bandwidth of each of the filter band is the same, each of them can be decimated by a factor of M or lower without causing aliasing. In this case the filter banks are called the decimated filter banks. If the decimation factor is M, then they are called maximally decimated filter banks and they result in the maximum computational efficiency. The filters $G_1(z)$, $G_2(z)$, $\cdots$, $G_M(z)$ are called the *synthesis filter banks*. The analysis and synthesis filter banks, shown in Fig.1.13, require narrow BP filters. The design of the analysis filter banks can be simplified by using the tree structure given in Fig.1.14. The corresponding tree structure for the synthesis filter bank is shown in Fig.1.15. These structures use only the LP and HP filters. The cutoff frequency of the LP filter is increased as the tree is traversed

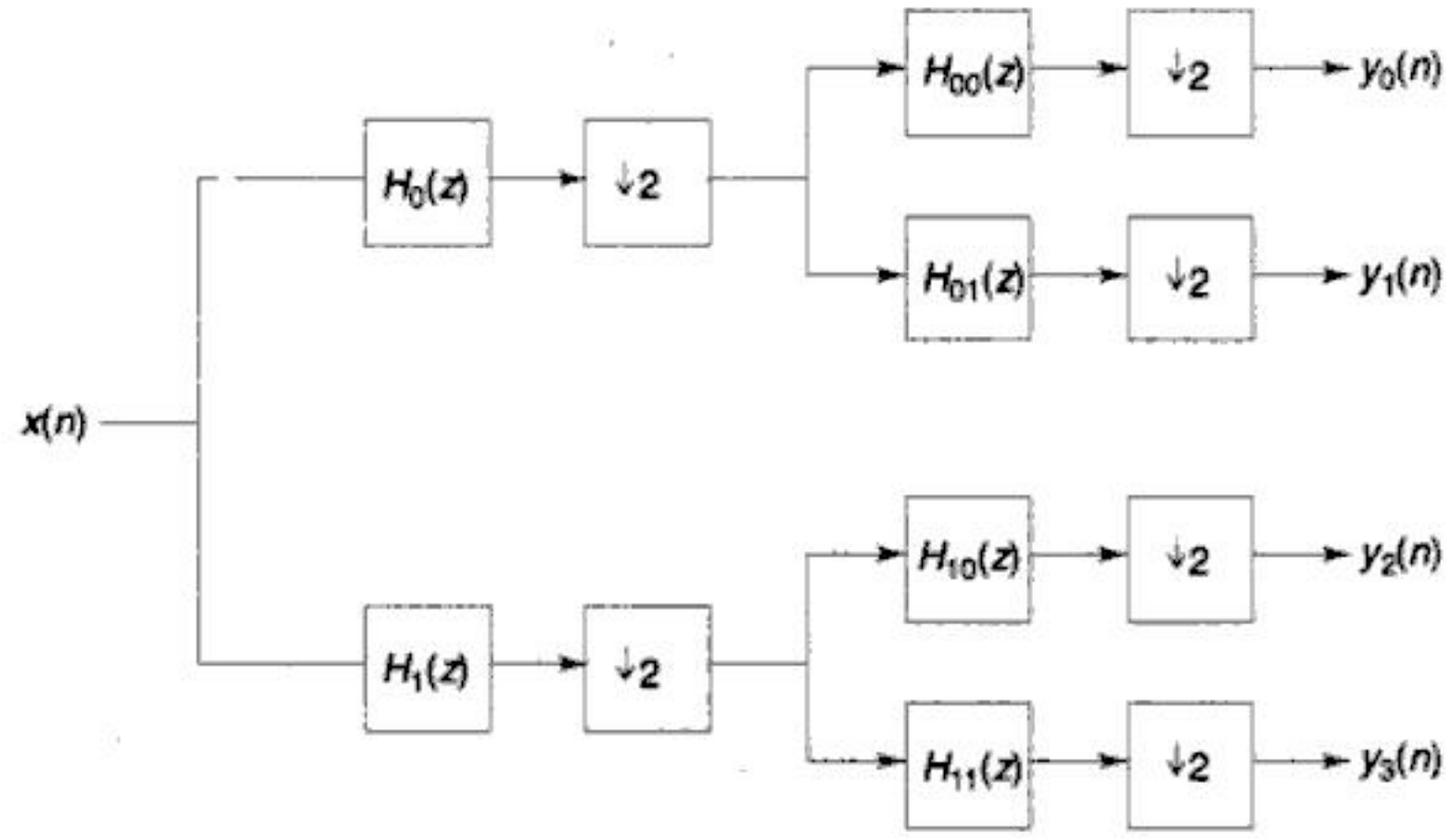


Fig.1.14 *Tree structured analysis filter bank*

towards the right. In Fig. 1.14 $H_0(z)$, $H_1(z)$ denote the transfer function of the LP and HP filter respectively and it can be verified that

$$H_0(z) = H_1(-z) \tag{1.47}$$

The two channel filter bank that satisfy (1.47) is referred to as a *quadrature mirror filter* (QMF) bank.

The tree structure is obtained by splitting every branch into two. However, it may not always be required to split each branch into two for sub band processing. For example, in the subband coding scheme proposed by Crochiere [1983], the analog speech signal is sampled at 8 kHz and split into four bands as shown in Fig. 1.13. Each of these bands has a bandwidth of 1 kHz. Next the band 0–1 kHz is split into two bands and a pruned tree with five bands: 0–0.5, 0.5–1, 1 – 2, 2 – 3 and 3 – 4 kHz is coded individually. Quantisation of the first two bands with 5 bits/sample, the next two bands with 4 bits/sample and the last band with 3 bits/sample gives a bit rate of 32 Kbps. Quantisation of the first two bands with 4 bits/sample, the next two bands with 2 bits/sample and the last band with 0 bits/sample gives a bit rate of 16 Kbps. With the 8 kHz sampling.

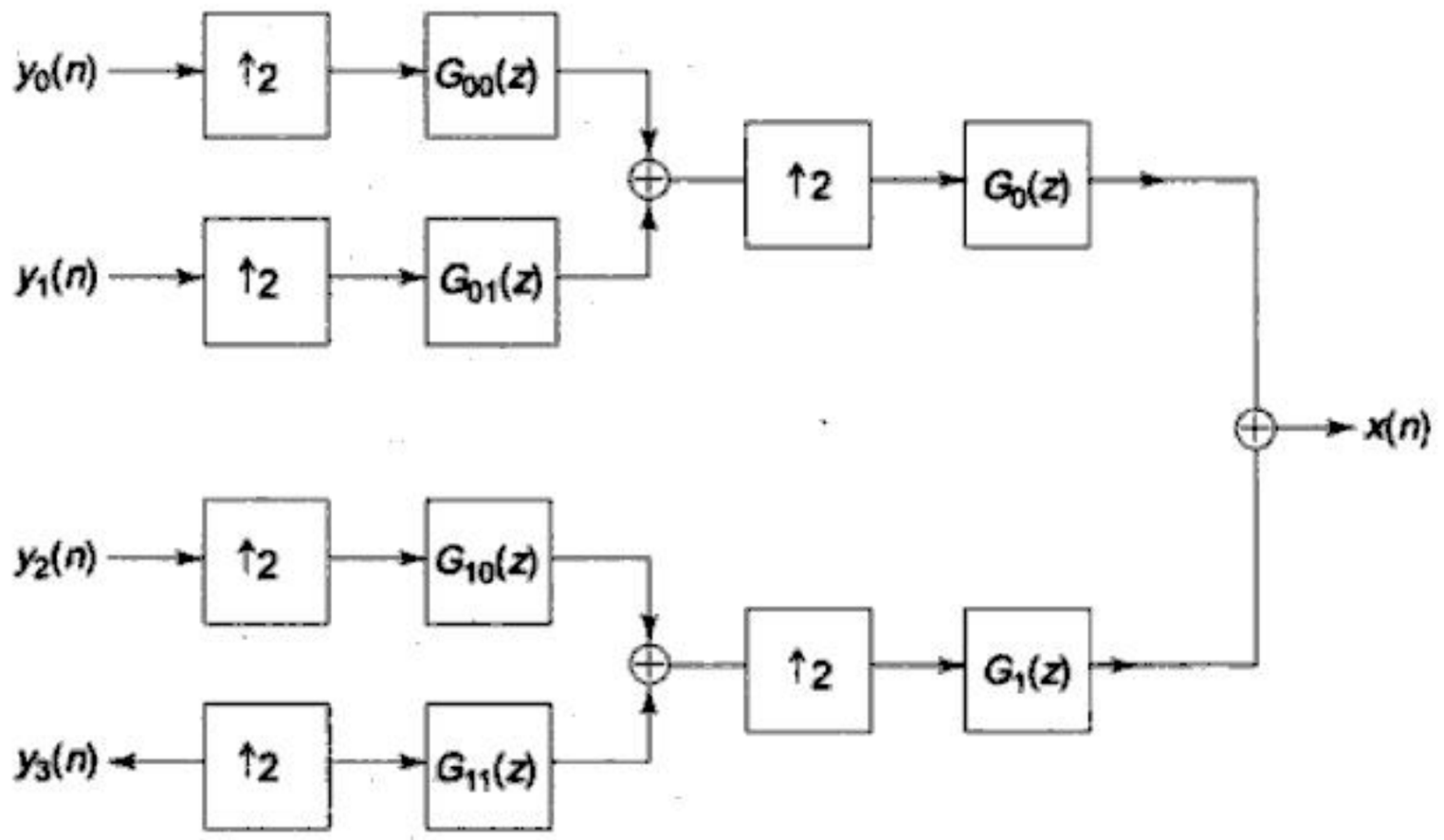


Fig. 1.15 *Tree structured synthesis filter bank*

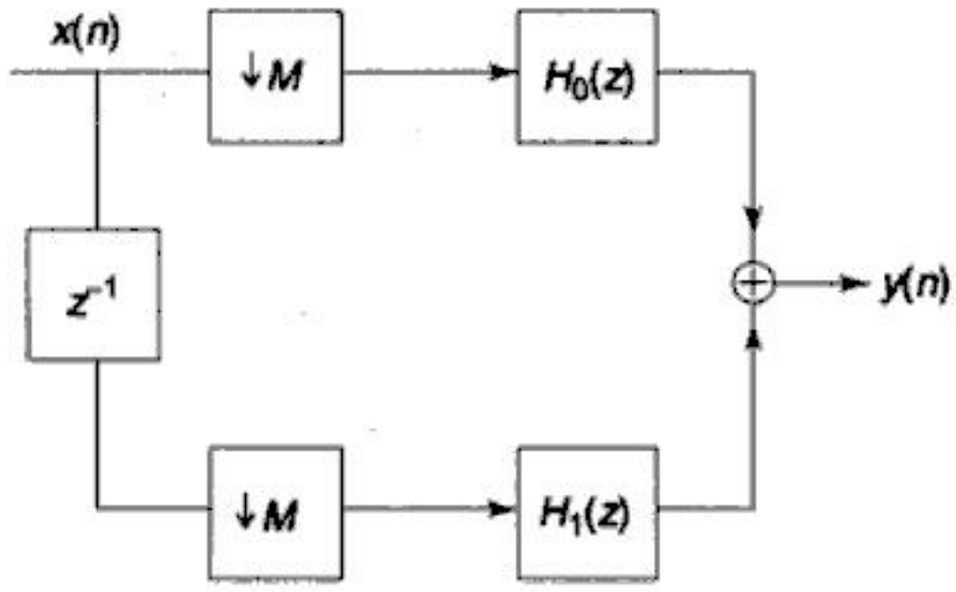


Fig. 1.16 *Linear filtering with two polyphase filters*

8 bits/sample, the bit rate required is 64 Kbps. Hence the decimation/interpolation has facilitated the data compression by a factor of 2–4.

1.17.5 Polyphase Filters

Another application of multirate signal processing is design of polyphase filter structures for speeding up the computation. Consider a LTI filter with N filter coefficients whose output is given by

$$y(n) = \sum_{k=0}^{N-1} x(k)\ h(n-k) \tag{1.48}$$

where $y(n)$, $x(n)$ and $h(n)$ are the nth sample of output, input and the impulse response of the LTI causal discrete time system and $x(n)$ and $h(n)$ are assumed to be 0 for $n < 0$.
Equation (1.48) can be rewritten as

$$y(n) = \sum_{m=0}^{N/2-1} x(2m)h\ (n-2m) + \sum_{m=0}^{N/2-1} x(2m+1)\ h(n-2m-1) \tag{1.49}$$

Hence $y(n)$ can be obtained using two filters each having $N/2$ taps. One filter processes the even samples and the other filter processes the odd samples. Computational complexity of the filter is reduced by a factor of 2. Each filter uses a decimator by a factor of 2. Using a delay unit the odd and even samples can be processed using the filter structure given in Fig.1.16. In this figure $H_0(z)$ and $H_1(z)$ denote the two filters with $N/2$ taps respectively. The first one processes the even samples and the second one processes the odd samples. This technique can be extended further. Using $M-1$ delay units and M decimators with decimation factor of M, the computational complexity of the individual filters can be reduced by a factor of M. The M individual filters are called as the polyphase filters.

1.17.6 Sampling Rate Conversion

Another application of multirate signal processing is the conversion of an analog signal digitised at one sampling rate to another digital signal that is stored and retrieved at another sampling rate. For example, CD quality voice is sampled, digitised and stored in a CD at the rate of 44.1 K samples/s. When this signal is to be stored/read in/from a digital audio tape (DAT), the samples have to be processed at the rate of 48 K samples/sec. Hence for this application, sampling rate conversion 44.1/48 K samples is required. This is achieved by choosing an interpolator with a factor of 160 and a decimator with a decimation factor of 147 as shown in Fig. 1.17. In Fig. 1.17 M=160 and N =147. Similar technique can be adopted for playing out a video signal digitised at one sampling rate on a system where the sampling rate is different.

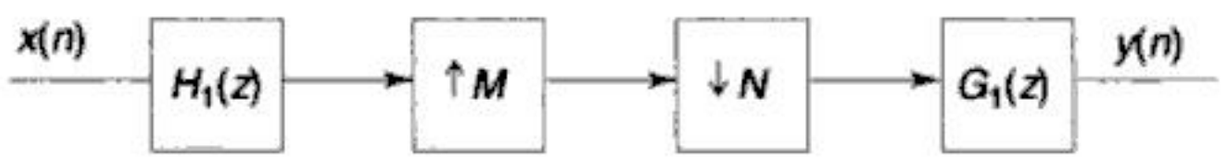


Fig.1.17 *Sampling rate conversion for CD to DAT data*

1.18 DISCRETE WAVELET TRANSFORM

Another application of multirate signal processing is the computation of wavelet transform and inverse wavelet transform coefficients. Wavelet transform is used for the compression of the speech and video signals. In wavelet analysis a signal is represented using a set of basis functions called the wavelets. These basis functions are obtained by shifting and dilating (scaling) a mother wavelet sequence $h(n)$. If a signal $x(k)$ is represented using m basis functions given by

$$hi(2^{i+1}\, n - k) \quad \text{for } (0 \le i \le m-1, -\infty < k < \infty)$$

the one-dimensional discrete wavelet transform (DWT) is defined as

$$y_i(n) = \sum_{k=-\infty}^{\infty} x(k)\, h_i\,(2^{i+1}\, n - k) \quad \text{for } 0 \le i \le m-2 \tag{1.50}$$

$$y_{m-1}(n) = \sum_{k=-\infty}^{\infty} x(k)\, h_{m-1}\,(2^{m-1}\, n - k) \quad \text{for } i = m-1 \tag{1.51}$$

$y_i(n)$ are called the *wavelet coefficients*. The inverse discrete wavelet transform (IDWT) is computed by the expression

$$x(n) = \sum_{i=0}^{M-2} \sum_{k=-\infty}^{\infty} y_i(k)\, f_i(n - 2^{i+1}k) + \sum_{k=-\infty}^{\infty} y_{m-1}(k)\, f_{m-1}\,(n - 2^{m-1}k) \tag{1.52}$$

where $f_i\,(n - 2^{i+1}k)$ are designed such that (1.52) perfectly reconstructs the original signal $x(n)$. It may be noted that the computation of the DWT and IDWT coefficients are similar to convolution operations. They can be calculated recursively as a series of convolutions and decimations using filter banks considered in Section 1.17.4.

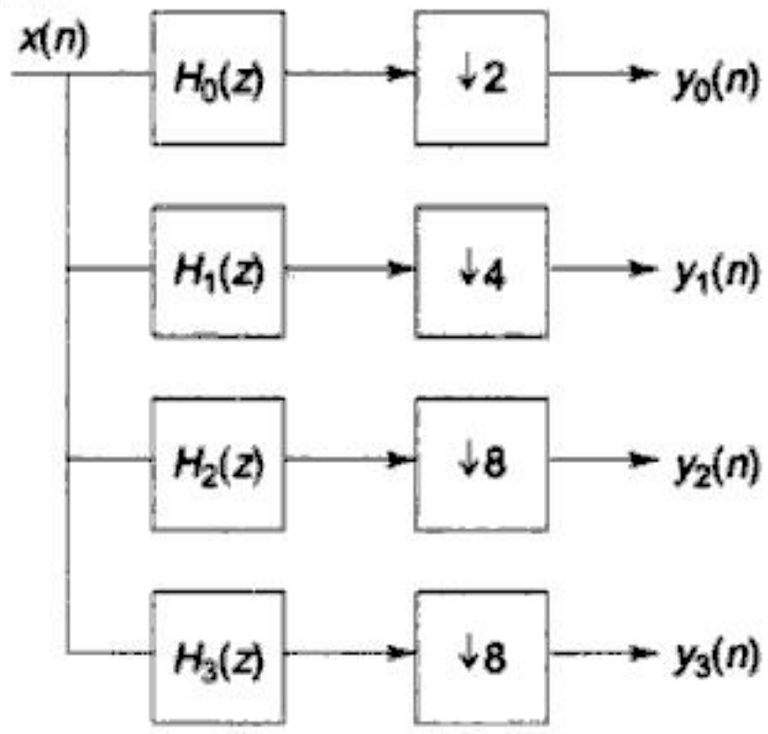


Fig.1.18 *Analysis filter banks for discrete wavelet transform*

For example, let us consider the computation of the DWT for $m = 4$ using filter banks. The wavelet coefficients are given by

$$y_0(n) = \sum_{k=-\infty}^{\infty} x(k)\, h_0\,(2n - k) \tag{1.53}$$

$$y_1(n) = \sum_{k=-\infty}^{\infty} x(k)\, h_1\,(4n - k) \tag{1.54}$$

$$y_2(n) = \sum_{k=-\infty}^{\infty} x(k)\, h_2\,(8n - k) \tag{1.55}$$

$$y_3(n) = \sum_{k=-\infty}^{\infty} x(k)\, h_3\,(8n - k) \tag{1.56}$$

They can be computed using the analysis filter banks with the decimators given in Fig.1.18.

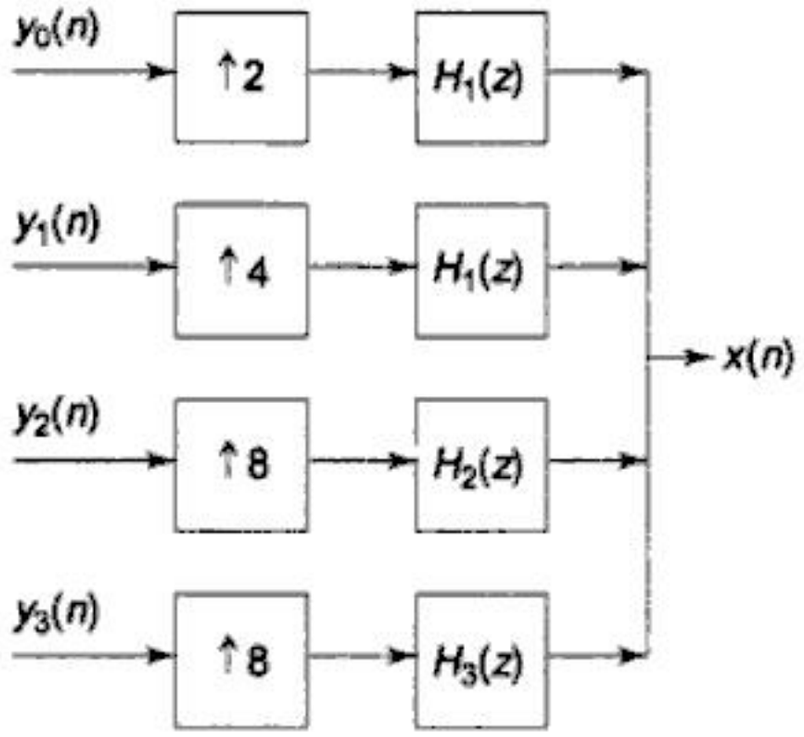


Fig.1.19 *Synthesis filter bank for inverse DWT*

The signal $x(n)$ can be reconstructed by computing the IDWT using the interpolators and synthesis filter bank as shown in Fig.1.19. DWT processes M samples at a time and generates M transform coefficients. When M is chosen to be of the form $M = 2^m$ the wavelet coefficients can be computed using the tree structured filter bank shown in Fig.1.14. The tree has m nodes or level at which a LP filter and a HP filter is used. As the tree is traversed from the root towards the child nodes, the cutoff frequency of the LP filter increases. However, the tree structure for the computation of DWT differs from that given in Fig.1.14. In this case in the tree, the branch corresponding to the output of the LP filter alone is split into branches; the HP output is not split further. Such a special kind of pruned-tree filter bank is called the *octave band filter bank*. The resulting tree structure for $m = 3$ is shown in Fig.1.20.

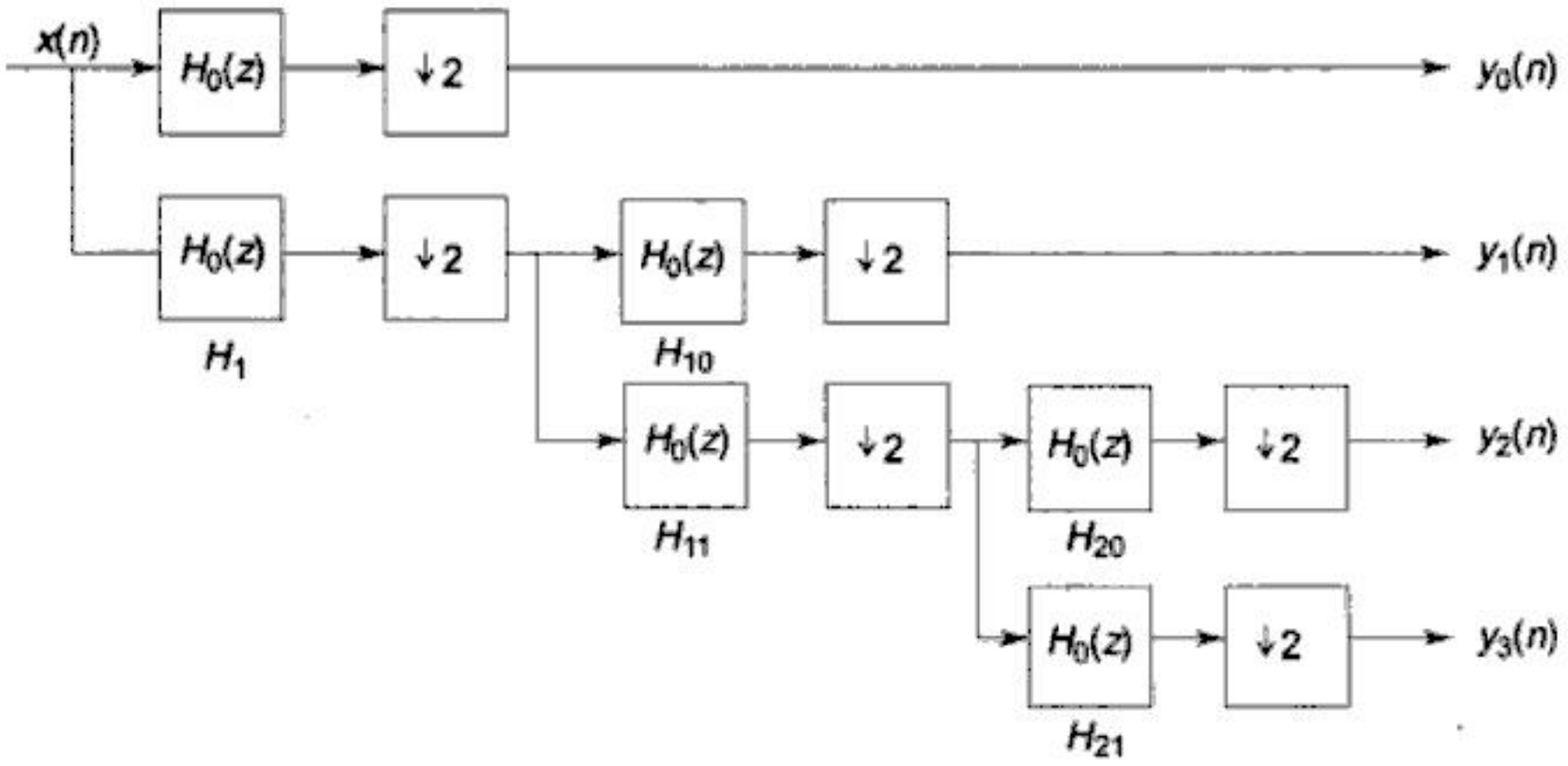


Fig.1.20 *Octave band filter bank for DWT*

1.19 ADAPTIVE FILTERS

As mentioned in Section 1.3, inverse filters and equalisers are used in a number of applications. The impulse response coefficients of these filters may have to be varied with respect to time as the original degradation mechanism itself may vary with respect to time. For example, the charactersitics of the

channel through which the signal is transmitted and reaches the receiver may change with time. A filter in which the filter coefficients are adapted to ensure that the desired signal is obtained as faithfully as possible is called an adaptive filter. If $d(n)$ is the desired signal and $y(n)$ is the output of the filter, the error $e(n)$ at the output of the filter at the nth sampling instant is given by

$$e(n) = d(n) - y(n)$$

One of the methods used for choosing the filter coefficients of the adaptive filter is to choose the coefficients so as to minimise the mean square error $E[e^2(n)]$; $e(n)$ is a random variable that can take any value between -2^{N-1} to 2^{N-1}, where N is the number of bits used for representing a number in the filter. Knowing the probability distribution of $d(n)$ and $y(n)$, the mean square error (MSE) can be computed. One of the popularly used technique for adaptive filter uses the FIR structure given in Fig.1.21 and is called the *Wiener filter*. The filter coefficients are adapted so as to minimize the MSE. If $h^{(i)}{}_n$ denotes the nth filter coefficient at the ith iteration, then its value at the $(i + 1)$th iteration $h^{(i+1)}{}_n$ is given by

$$h^{(i+1)}{}_n = h^{(i)}{}_n + e(n)\alpha \times (n)$$

The value of α should be chosen to be in the range, $0 < \alpha \leq [1/(NE(x^2(n)))]$ where $x(n)$ is the input to the adaptive filter and $E(x^2(n))$ denotes the average input signal power. The algorithm used above for the adaptation is called the least mean square (LMS) algorithm.

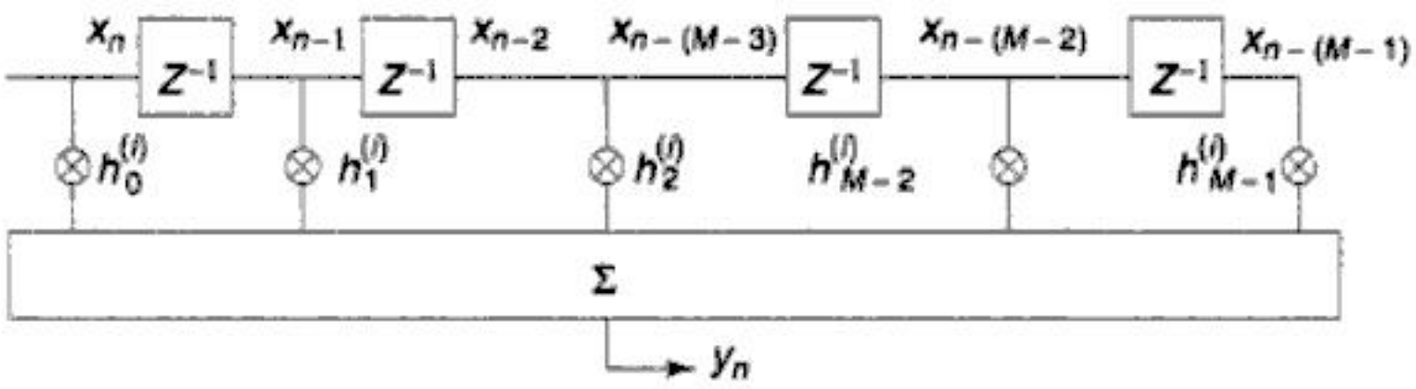


Fig.1.21 *Adaptive FIR filter*

In addition to the adaptive equalisers and inverse filters that may require adaptive filters, the adaptive filters are used in a number of applications. For example, an application may require the ambient noise present to be cancelled in preference to the signal. This, for example, could be used in an audio system used in an automobile. Since the background noise depends on the objects present, the noise generated would also change with time. The LMS algorithm can be used to adjust the filter coefficients to reduce the interference. The noise can be modelled as autoregressive or autoregressive moving average process. They are equivalent to outputs of FIR and IIR filters respectively.

1.20 DISCRETE COSINE TRANSFORM AND IMAGE DATA COMPRESSION

One of the popular methods used for compression of image data is using a combination of sample domain and transform domain techniques. An image may be considered to be consisting of a large number of picture elements called *pixels*. The image intensity corresponding to each of these pixels is digitised and processed further for either storage or transmission to another point. The number of pixels that constitutes an image depends on the image resolution used. One of the standards used is to assume the image to have 512×512 pixels/frame. On the other hand, the VGA standard assumes a resolution of 640×480 pixels/frame. Real time transmission of images requires 25 frames to be

transmitted per second. For monochrome images with a resolution of 512×512 this requires a transmission rate of about 50 Mbps assuming 8-bit accuracy/pixel. For colour images, the transmission rate becomes 150 Mbps. This calls for a large bandwidth and hence, the image data has to be compressed for transmitting it using a moderate band width. For compression of images two commonly used standards are the Joint Photographic Expert Group (JPEG) and Motion Picture Expert Group (MPEG). JPEG is used for still images and MPEG is used for moving images. They both make use of discrete cosine transform for image data compression.

An orthogonal transform is said to be efficient if it is able to pack the energy in the signal in the form of a set of samples in the first few transform coefficients. This is useful for image transmission as well as storage as only less number of coefficients need to be considered compared to the original signal. For the image data that can be approximated by a first-order Markov process to a good accuracy, DCT is an efficient transform. $X(k)$, the kth transform coefficient of 1 dimensional DCT of a sequence $x(n)$ of length M is given by

$$X(k) = \alpha(k) \sum_{n=0}^{M-1} x(n) \cos\ [(2n + 1)k\pi/2M] \tag{1.57}$$

Similarly $x(n)$, the nth data in the sampled sequence, can be expressed in terms of the DCT coefficients as follows:

$$x(n) = \frac{1}{M} \sum_{k=0}^{M-1} \alpha(k)X(k) \cos\ [(2n + 1)\ k\pi/2M] \tag{1.58}$$

where
$$\alpha(k) = 1/\sqrt{2} \quad \text{if } k = 0$$
$$= 1 \qquad k \neq 0$$

The M-point DCT and IDCT can be computed using $2M$-point DFT of a sequence $y(n)$ obtained as follows:

$$y(n) = x(n) \qquad \text{for } 0 \leq n < M$$
$$= x(2M - n - 1) \qquad \text{for } M \leq n < 2M \tag{1.59}$$

Let the kth DFT coefficient of $y(n)$ be $Y(k)$. The DCT coefficient $X(k)$ is given by

$$X(k) = \alpha(k)\ Y(k)\ e^{-j(k\pi/2M)} \qquad \text{for } 0 \leq k < M \tag{1.60}$$
$$= 0 \qquad \text{otherwise}$$

The M-point IDCT of the transform coefficients $X(k)$ may be obtained by computing the inverse DFT of the $2M$-DFT coefficients $Y(k)$ given by

$$Y(k) = X(k) \qquad \text{for } 0 \leq k < M$$
$$= 0 \qquad \text{for } k = M \tag{1.61}$$
$$= -X(2M - 1 - k) \qquad \text{for } M + 1 \leq k < 2M$$

If $y(n)$ denotes the inverse DFT coefficients of $Y(k)$, then $x(n)$ is obtained from $y(n)$ as follows:

$$x(n) = y(n) \qquad \text{for } 0 \leq n < M \tag{1.62}$$

Since DFT can be computed using FFT, the computational complexity for DCT and IDCT are of the order of $2M \log_2 2M$.

Image data is normally an $M \times N$ 2D array. This may be converted to an 1D array and the 1D DCT may be computed for data compression. Alternately, $X(k, l)$, the MN 2D DCT coefficients of the 2D image data array $x(m, n)$ may be computed using the 2D DCT equation given by

$$X(k, l) = \alpha(k, l) \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m, n) \cos[(2m + 1)k\pi/2M] \cos[(2n + 1)l\pi/2N] \qquad (1.63)$$

The 2D image data array elements $x(m, n)$ can be obtained from the 2D DCT coefficients $X(k, l)$ using the inverse DCT and are given by

$$x(m, n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X(k, l)\ \alpha(k, l) \cos[(2m + 1)\ k\pi/2M] \cos\ [(2n + 1)l\pi/2N] \qquad (1.64)$$

where
$$\alpha(k, l) = 1/\sqrt{2} \text{ if } \quad k = 0 \text{ and } l = 0 \qquad (1.65)$$
$$= 1 \qquad k \neq 0 \text{ and } l \neq 0$$

To reduce the computational and storage requirements, the $M \times N$ image data array is split into square blocks of size $N \times N$ where ($N < N$). DCT for each of these square blocks may be computed using either the 2D DCT or the 1D DCT. This also permits the computation of DCT using a number of processors simultaneously. DCT coefficients may be truncated to be zero when their value is below a threshold for data compression. For this purpose either threshold coding or zonal coding may be used. See Jain [1995] for more details.

Further, the DCT coefficients matrix becomes almost like an upper triangular matrix with almost identical elements along the off diagonals. This property is used to achieve further data compression using run length encoding. In this scheme, a sequence of data that has identical value is encoded into two n-bit numbers where the first number denotes the frequency of repetition of the number and the second number denotes the actual value of the number.

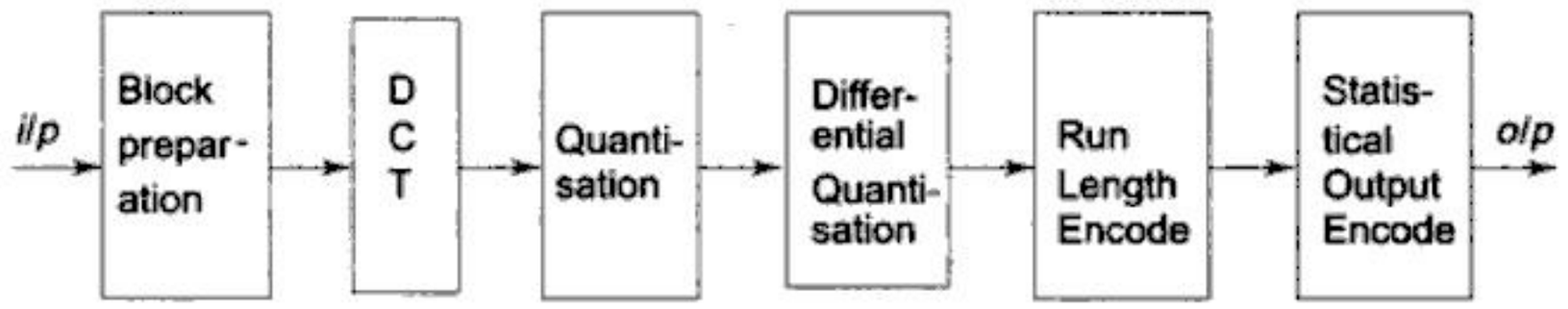


Fig. 1.22 *Block diagram of the JPEG coding scheme*

Block diagram of the JPEG coding scheme is shown in Fig.1.22. Let the image resolution be 640 × 480 pixels. The function performed by each of the block is as follows:

Block preparation: The given colour image matrix of size 640 × 480 is split into three matrices: one into 4800 luminance (Y) matrices each of size 8×8, and the remaining into two 1200 matrices each of size 8 × 8. They are called the *in phase* (I) and *Quadrature* (Q) *matrices*. The Red, Green and Blue (RGB) components are mapped into Y, I and Q components using the equation given in (1.66).

$$Y = 0.30R + 0.59G + 0.11B \qquad (1.66)$$
$$I = 0.60R - 0.28G - 0.32B$$
$$Q = 0.21R - 0.52G + 0.31B$$

This is done in order to achieve data compression. Our eye has poor resolution to colour (chrominance) information compared to the intensity (Y) information. Hence the full image matrix is

mapped into Y matrix. However, the I and Q matrices representing the chrominance information is obtained by reducing the image matrix size by a factor of 2 along the X and Y coordinates.

Discrete Cosine tranform and Quantisation: DCT is computed for each of the 8×8 matrices corresponding to the Y, I and Q components. The DCT has very good energy compaction. Hence in the DCT coefficients of an 8×8 matrix, only the elements corresponding to the left hand top corner has significant values as shown in Fig.1.23. The higher order coefficients along the rows and columns correspond to high frequency components and they need to be reproduced with only less number of levels. This is because our eye has a poor high frequency response. Different coefficients are allocated different number of bits based on their importance. One such allocation scheme is shown in Fig.1.24. Using this allocation scheme, the image matrix coefficients are modified as shown in Fig.1.25. For coding this image either the zonal coding or

160	90	50	24	16	16	32	0
102	86	46	20	8	16	0	0
62	48	36	16	16	16	0	0
24	20	16	12	24	32	0	0
16	16	16	0	0	0	0	0
16	16	32	32	0	0	0	0
32	32	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 1.23 *DCT coefficients of a 8 x 8 matrix*

8	8	7	6	5	4	3	2
8	8	7	6	5	4	3	2
7	7	7	6	5	4	3	2
6	6	6	6	5	4	3	2
5	5	5	5	5	4	3	2
4	4	4	4	4	4	3	2
3	3	3	3	3	3	3	2
2	2	2	2	2	2	3	2

Fig.1.24 *Quantisation table for the DCT coefficients*

160	90	25	6	2	1	1	0
102	86	23	5	1	1	0	0
31	24	18	4	2	1	0	0
6	5	4	3	3	2	0	0
2	2	2	0	0	0	0	0
1	1	2	2	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Fig. 1.25 *Quantised DCT coefficients*

the threshold coding can be employed. The number of bits required for transmission is reduced further by encoding only the difference in the amplitude corresponding to the DC component in the successive blocks along the vertical and horizontal direction. This corresponds to the top left corner element of the 8×8 transform matrix. This is called the *differential quantisation* and is used only for the DC coefficients.

Runlength encoding: The quantised matrix is converted into a linear vector using the zig-zag pattern. This helps to combine the 0s using run length encoding explained above.

Source coding: The resulting vector is further compressed using Huffman coding, which allocates more number of bits for those amplitudes whose probability of occurrence is less and vice versa.

1.21 LINEAR PREDICTIVE CODER AND SPEECH COMPRESSION

Another application of digital signal processing is the data compression using the linear predictive coder for the speech signal. For this purpose the human vocal tract is modelled as a linear time varying filter. The voiced sounds (e.g., vowels) are produced by exciting this filter with quasi periodic impulses. These impulses are called *quasi periodic* as periods of the impulses are not exactly equal and the amplitude of the impulses are not exactly equal. The unvoiced sounds (e.g., S, SH) are produced by exciting the above filter with white noise. The time varying filter is normally approximated by a linear time invariant filter over short time intervals. One of the common models used for the vocal tract is the linear predictive coder, which is a LTI filter over a short-term interval where the output at the nth sample, $y(n)$, is predicted based on the past p samples and is given by

$$y(n) = a_1 y(n-1) + a_2 y(n-2) + \cdots + apy(n-p) + u(n)$$

The error $u(n)$ in turn may be obtained by exciting another linear LTI filter fed with another exciting signal. In the European standard for cellular telephone GSM 6.10, the output rate of the speech coder used is 13 kbps and the input to the coder is voice signal sampled at the rate of 8 Kilo samples with 13 bits/sample. Hence the compression factor is 8. To achieve this, samples corresponding to every 20 ms are used to compute the LPC model parameters and the model parameters corresponding to excitation signal, including the LTI filter used if any, for generating the excitation. Additional details on estimating the LPC model parameters may be obtained from Rabiner and Juang [1993].

Review Questions

1.1 Find the output $y(n)$ of a linear shift invariant system with unit sample (impulse) response $h(n)$ given by $h(0) = 3$, $h(1) = 2$, $h(2) = 1$ and $h(n) = 0$ for all other value of n if it is fed with an input $x(n)$ that is non-zero only for $n = 0$ and 1. $x(0) = 2$ and $x(1) = 1$.

1.2 Show that the output $y(n)$ of a causal linear time invariant discrete time system for the input $x(n)$ given by
$x(n) = e^{j\omega n}$ is $y(n) = x(n)\, H(e^{j\omega})$ where $H(z)$ is the system (transfer) function of the LTI system.

1.3 Find the frequency response of the LSI causal system given by
$y(n) = x(n) + a\, y(n-1)$, for $a < 1$, and sketch the magnitude and the phase response of this system. Find also its impulse response. Show that the phase response of this system denoted as $\angle H(e^{j\omega})$ is given by $\angle H(e^{j\omega}) = \omega - \tan^{-1}(\sin\omega/(a - \cos\omega))$.

1.4 Find the frequency response of the LSI causal system given by

$y(n) = x(n) + a\,y(n-2)$, for $a < 1$. Find also its impulse response.

1.5 The magnitude of the frequency response of the LSI causal system given by $y(n) = x(n) - ax(n-1) + by(n-1)$ is independent of frequency and $a \neq b$. Find the relationship between a and b. Such a system is called an all pass system.
Hint: Assume the magnitude of the numerator of the frequency response to be k times the magnitude of the denominator.

1.6 Determine the impulse response coefficients of a digital LP filter with $H(e^{j2\pi fT_s}) = 1$, for $f < f_c$, and,

$= 0$, for $f_c < f < f_s/2$,

where $f_s = 1/T_s$ is the sampling rate of the digital filter.

1.7 Determine the impulse response coefficients of a digital filter with $H(e^{j2\pi fT_s})$ given in Fig. 1.26.

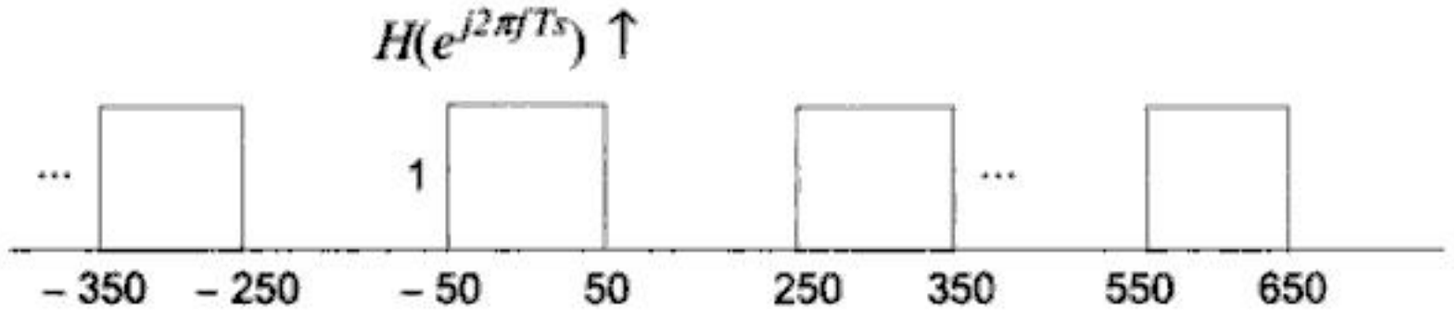


Fig. 1.26 *Frequency response of the filter*

1.8 Determine the impulse response coefficients of a digital LP filter with $H(e^{j\omega}) = 1$, for $\omega < \omega_c$, and,

$= 0$, for $\omega_c < \omega < \pi$.

1.9 Determine the impulse response coefficients of a digital filter with $H(e^{j\omega})$ given in Fig. 1.27.

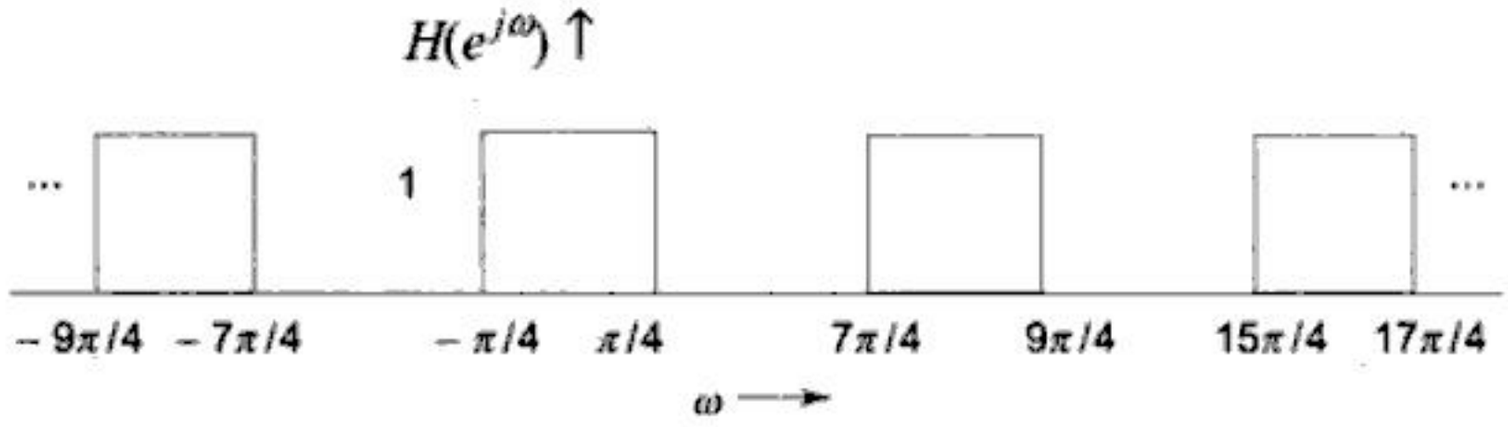


Fig. 1.27 *Frequency response of the filter*

1.10 Sketch the frequency spectrum of $f(t) = \sin 2\pi(1250)t$. Determine the minimum sampling rate required to reconstruct the signal from its samples. What should be the cutoff frequency of the reconstruction (smoothing) filter? If the signal is sampled at the rate of 2000 samples/s what would be the output of the smoothing filter?

1.11 Sketch the frequency spectrum of
$f(t) = \sin 2\pi(1000)t + \sin 2\pi(1250)t$.
Determine the minimum sampling rate required to reconstruct the signal from its samples. What should be the cutoff frequencies of the reconstruction (smoothing) filter? If the signal is sampled at the rate of 2000 samples /s what would be the output of the smoothing filter?

1.12 Let $f(t)$ be $\sin 2\pi(1000)t + \sin 2\pi(1250)t$ and $y(t) = [f(t)]^2$. Determine the minimum sampling rate required to reconstruct $y(t)$ from its samples. What should be the cut-off frequencies of the reconstruction (smoothing) filter?

1.13 A continuous time signal is to be filtered to remove frequency components in the range 10 kHz $< f <$ 25 kHz. The maximum frequency component present in the signal is 50 kHz. If the signal is sampled at the Nyquist rate, what range of digital frequencies should be rejected by the filter? Sketch the magnitude characteristics of the ideal digital filter required for this purpose.

1.14 Find the impulse response of a system whose system function is given by

$$X(z) = \frac{2}{(1-0.5z^{-1})(1+0.5z^{-1})}.$$

1.15 Find the impulse response of a system whose system function is given by

$$X(z) = \frac{(3-z^{-2})}{(1-0.25z^{-1})(1-0.75z^{-1})}.$$

1.16 The impulse response of a discrete time system is given by
$h(n) = a^{-n}$, for $0 \le n \le 15$, and 0, otherwise. Find the system function of this filter.

1.17 The impulse response of a discrete time system is given by
$h(n) = [\ 1, 0.2, 0.04, 0.0016, 0, 0, 0, \cdots]$. Find the system function of this filter.

1.18 (a) Find the system function of the following LTI causal systems
(i) $y_1(n) = x_1(n) + 0.25\ x_1(n-1) + 0.5\ y_1(n-1)$
(ii) $y_2(n) = x_2(n) + 0.5\ x_2(n-1) + 0.75\ y_2(n-1)$.
(b) If $x_2(n) = y_1(n)$, find the system function of the LTI system whose input is $x_1(n)$ and the output is $y_2(n)$.
(c) If $x_1(n) = y_2(n)$, find the system function of the LTI system whose input is $x_2(n)$ and the output is $y_1(n)$.
(d) If the systems given in (a) are cascaded, find the system function of an equivalent LTI system.
(e) If the systems given in (a) are connected in parallel, find the system function of an equivalent system.

1.19 Find the 3-point DFT of the sequence given by

$$x(n) = 1, \text{ for } n = 0,1, 2.$$

1.20 Find the 8-point DFT of the sequence given by
$x(n) = 1$ for $n = 0,1, 2$ and $x(n) = 0$, for $n = 3, 4, \cdots, 7$.

1.21 Find the 8-point DFT of the infinite sequence given by $a(n) = (0.5)^n$ for $n = 0, 1, 2, \ldots$.

1.22 Find the 16-point DFT of the infinite sequence given by $h(n)=(0.25)^n$ for $n = 0, 1, 2, \ldots$.

1.23 Show that the computation of the DFT of a N-point sequence is equivalent to a multiplication of matrix by a vector. What is the (m, n)th element of this matrix.

1.24 It is required to design a LP digital filter with cutoff frequency of 0.5π.
(a) If the sampling period is 1s, what should be the cutoff frequency of the corresponding analog filter if the digital filter is designed using (i) impulse invariant technique and (ii) bilinear transform method?

(b) If the sampling period is 0.1 ms, what should be the cutoff frequency of the corresponding analog filter if the digital filter is designed using (i) impulse invariant technique and (ii) bilinear transform method?

1.25 It is required to design a LP digital filter with cutoff frequency of 0.5π. If the sampling period is 0.01 ms, what should be the cutoff frequency of the corresponding analog filter if the digital filter is designed using (i) impulse invariant technique and (ii) bilinear transform method?

1.26 Determine the impulse response coefficients of a digital filter whose frequency response is given by

$$H(e^{j\omega}) = 0 \quad \text{for } |\omega| < .85\pi$$
$$= 1 \quad \text{for } .85\pi < |\omega| < \pi$$

1.27 (a) Obtain the impulse response of the analog filter whose transfer function is given by,

$$H(s) = \frac{1}{(s-0.5)}.$$

(b) If the sampling rate used is $1/T$, what is the impulse response of the equivalent digital filter obtained using the impulse invariant method?

(c) What is the transfer function of the equivalent digital IIR filter?

1.28 The transfer function of an analog filter is given by,

$$H(s) = \frac{1}{(s-a)}.$$

(a) What is the transfer function of the equivalent digital IIR filter if bilinear transform is used to convert the analog filter to the equivalent digital filter? Assume the sampling rate to be $1/T$.

(b) What is the impulse response of the equivalent digital IIR filter?

1.29 Obtain the impulse response of the analog filter whose transfer function is given by

$$H(s) = \frac{1}{(s-0.75)}.$$

If the sampling rate used is 10,000 samples/s, what is the impulse response of the equivalent digital filter obtained using the impulse invariant method?

1.30 Determine the impulse response coefficients of a digital FIR filter whose frequency response is given by

$$H(e^{j\omega}) = 0 \quad \text{for } |\omega| < .75\pi$$
$$= 1 \quad \text{for } .75\pi < |\omega| < \pi$$

using a filter of order 8 and windowing using Black man window given by

$$\omega(n) = 0.42 - 0.5\cos(2\pi n/N-1) + 0.08\cos(4\pi n/N-1) \quad \text{for } 0 \le n \le N-1$$
$$= 0 \quad \text{otherwise.}$$

1.31 A digital FIR filter of order 8 is to be designed using the frequency sampling method. The frequency response of the filter is given by

$$H(e^{j\omega}) = 1 \quad \text{for } |\omega| < .3\pi$$

$$= 0 \quad \text{for } .3\pi < |\omega| < \pi$$

Determine the impulse response coefficients.

1.32 Show that the frequency response function $H(e^{j\omega})$ of a linear phase filter with M (odd) taps and with impulse response $h(n)$, $n = 0, \cdots, N-1$ symmetrical about $(N-1)/2$ is given by

$$H(e^{j\omega}) = e^{-j\,\omega(N-1)/2} \sum_{n=0}^{(N-1)/2} a(n) \cos\omega n$$

where $a(0) = h[(N-1)/2]$ and $a(n) = 2h[(N-1)/2 - n]$ for $n = 1, \cdots, (N-1)/2$.

1.33 Show that the frequency response function $H(e^{j\omega})$ of a linear phase filter with M (even) taps and with impulse response $h(n)$, $n = 0, \cdots, N-1$ symmetrical about $(N-1)/2$ is given by

$$H(e^{j\omega}) = e^{-j\omega(N-1)/2} \sum_{n=1}^{N/2} b(n) \cos[\omega(n - \tfrac{1}{2})]$$

where $b(n) = 2h[N/2 - n]$ for $n = 1, \cdots, N/2$.

1.34 Show that the frequency response function $H(e^{j\omega})$ of a linear phase filter with M (odd) taps and with impulse response $h(n)$, $n = 0, \cdots, N-1$ antisymmetrical about $(N-1)/2$ is given by

$$H(e^{j\omega}) = e^{-j\omega(N-1)/2}\, e^{j\,\pi/2} \sum_{n=1}^{(N-1)/2} c(n) \sin(\omega n)$$

where $c(n) = 2h[(N-1)/2 - n]$ for $n = 1, \cdots, (N-1)/2$ and $h[(N-1)/2] = 0$.

1.35 Show that the frequency response function $H(e^{j\omega})$ of a linear phase filter with M (even) taps and with impulse response $h(n)$, $n = 0, \cdots, N-1$, symmetrical about $(N-1)/2$ is given by

$$H(e^{j\omega}) = e^{-j\omega(N-1)/2}\, e^{j\,\pi/2} \sum_{n=1}^{N/2} d(n) \sin[\omega(n - \tfrac{1}{2})]$$

where $b(n) = 2h[N/2 - n]$ for $n = 1, \cdots, N/2$

1.36 Realise the system given by the transfer function

$$X(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}$$

in direct form I and direct form II.

1.37 Realise the system given by the transfer function

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_4 z^{-4}}{1 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + b_4 z^{-4}}$$

in (a) direct form I, (b) the transpose of direct form I and,
(c) canonic form.

1.38 Show how DCT of an 8-point sequence can be computed using a $2N$-point FFT.

1.39 Show how the DCT coefficients of an 8-point sequence can be computed using a bank of filters.

1.40 Show how the computational complexity of an FIR filter of order 21 can be reduced by using three polyphase filters.

Self-test Questions

1.41 $y(n)$ denotes the output of a linear shift invariant system with unit sample (impulse) response $h(n)$ given by $h(0) = 3$, $h(1) = 2$, $h(2) = 1$ and $h(3) = 0.5$. $h(n) = 0$ for all other values of n. This system is fed with an input $x(n)$ that is non-zero only for $n = 0$, 1 and 2 with $x(0) = 2$, $x(1) = 1$ and $x(2) = 0.5$. The minimum value of n ($n > 0$) for which $y(n) = 0$ is.

(a) 8 (b) 7 (c) 6 (d) 5

1.42 Which of the following systems are causal?

(a) $y(n) = x(n) + 0.25\ x(n-1) + 0.5\ x(n-2)$

(b) $y(n) = x(n) + 0.25\ x(n-3) + 0.75\ y(n-1)$

(c) $y(n) = x(n) + 0.5\ x(n-1) + 0.5\ y(n-1) + 0.8\ y(n-2)$

(d) $y(n) = x(n+1) - 0.5\ x(n-1) + 0.8\ x(n-1)$

1.43 The minimum sampling rate required to reconstruct the signal $f(t) = \sin 2\pi(1000)t$ from its samples is ——— Hz.

(a) 500 (b) 1000 (c) 2000 (d) none of the above

1.44 The cutoff frequency of the reconstruction (smoothing) filter for a signal sampled at a frequency of f_s is ————————.

(a) $f_s/2$ (b) f_s

(c) $2f_s$ (d) f_m, i.e. the signal maximum frequency

1.45 The signal $f(t) = \sin 2\pi(1000)t$ is sampled at the rate of 1800 samples/s. The frequency of the signal at the output of the smoothing filter is ——— Hz.

(a) 1000 (b) 900 (c) 1100 (d) 800

1.46 Speech is digitised using a sampling rate of 8 kHz. An antialiasing filter with cutoff frequency of 3.4 kHz is preceded by the sampler. The loss of the speech signals in the frequency range 3.4 – 20 kHz due to antialiasing filter introduces a degradation in the signal quality. On the other hand, sampling without the antialiasing filter also introduces degradation in the signal quality. Which of the following statements are true.

(a) Degradation with antialiasing filter is less

(b) Degradation without antialiasing filter is less

(c) Degradation with or without the antialiasing filter is the same

1.47 The number of stages of FFT computations required for the computation of the DFT of a 512-point sequence is _____.

(a) 9 (b) 8 (c) 7 (d) 6

1.48 At the fifth stage of FFT computation of a 512-point FFT, the number of distinct twiddle factors used is _____.

(a) 2 (b) 3 (c) 4 (d) 5

1.49 Which of the the following properties are true for an IIR filter designed using bilinear transform method.

(a) Requires the use of antialiasing filter

(b) Requires prewarping the filter cutoff frequencies

(c) Not suited for the design of HP filters

(d) Results in unique mapping from analog to digital frequencies

1.50 Which of the the following properties are true for an IIR filter designed using impulse invariant technique.
(a) Requires the use of antialiasing filter
(b) Requires prewarping the filter cutoff frequencies
(c) Not suited for the design of HP filters
(d) Results in unique mapping from analog to digital frequencies

1.51 Which of the following properties are true for the linear phase FIR filter with even number of coefficients and symmetric impulse response.
(a) Not suited for HP filter but has real magnitude response
(b) Suited for HP filter and has real magnitude response
(c) Has imaginary magnitude response
(d) Suited for differentiators and hilbert transformers

1.52 Which of the following properties are true for the linear phase FIR filter with odd number of coefficients and symmetric impulse response.
(a) Not suited for HP filter but has real magnitude response
(b) Suited for HP filter and has real magnitude response
(c) Has imaginary magnitude response
(d) Suited for differentiators and hilbert transformers

1.53 Which of the following characteristics are true for a half band filter.
(a) In a filter with M taps, the value of the coefficients of the odd taps is zero.
(b) The frequency response of the filter is antisymmetric with respect to the frequency $f_s/4$, where f_s is the sampling frequency.
(c) The ripple in the pass band is equal to that in the stop band.
(d) These filters are used in multirate systems.

Table 1.3 Impulse response coefficients of 4 filters

Filter no.	$h(0)$	$h(1)$	$h(2)$	$h(3)$	$h(4)$	$h(5)$	$h(6)$
I	2	6	7	6	2	0	0
II	2	5	8	–5	-2	0	0
III	2	3	4	4	3	2	0
IV	2	0	3	0	2	0	0

1.54 Among the filters given in Table 1.3, the filter that has imaginary magnitude response and is suited for differentiators and hilbert transformers is.
(a) I (b) II (c) III (d) IV

1.55 Among the filters given in Table 1.3, the filter that is not suited for HP filter but has real magnitude response is.
(a) I (b) II (c) III (d) IV

1.56 Among the filters given in Table 1.3, the filter that is suited for HP filter and has real magnitude response is.
(a) I (b) II (c) III (d) IV

1.57 Among the filters given in Table 1.3, the filter whose frequency response is antisymmetric with respect to the frequency *fs*/4 is.
(a) I (b) II (c) III (d) IV

1.58 The number of multiplications required for performing the convolution of two sequences with identical length 8 using the direct method is.
(a) 256 (b) 120 (c) 128 (d) 64

1.59 The number of multiplications required for performing the convolution of two sequences with identical length 8 using the indirect method using FFT is.
(a) 256 (b) 120 (c) 192 (d) 185

1.60 A multirate system is required for converting the sampling rate from 48 K samples to 42.1 K samples. The interpolation factor, decimation factor to be used is.
(a) 147, 160 (b) 160,147 (c) 480, 421 (d) 421,480 .

2

Introduction to Programmable DSPs

The programmable digital signal processors (P-DSPs) are designed with features that are specifically required for digital signal processing applications. The conventional microprocessors are meant for general purpose applications and hence they do not have these features. However, an advanced microprocessor or a RISC processor may use some of the techniques adopted in P-DSPs or may even have instructions that are specifically required for DSP applications. They may have performances close to that of a P-DSP for certain operations. For example, the DEC Alpha 21064 computes a 1024 point complex FFT in 480 μs, as compared to the Analog device ADSP 21060 that takes about 460 μs to carry out the same operation. However in terms of low power requirement, cost, real time I/O capability and availability of high speed on-chip memories, the P-DSPs have an advantage over the advanced microprocessors and the RISC processors. In this chapter some of the features specifically required for performing digital signal processing operations efficiently are discussed in detail.

2.1 MULTIPLIER AND MULTIPLIER ACCUMULATOR (MAC)

One of the most common operations required in digital signal processing applications is array multiplication. For example, convolution and correlation require array multiplication. In Chapter 1, it was shown how the array multiplication can be done using a single multiplier and adder. The implementation scheme is reproduced in Fig. 2.1. One of the important requirements of these array multipliers is that they have to process the signals in real time. Before the next sample of the input signal arrives at the input to the array, the array multiplication should be completed. This requires the multiplication as well as accumulation to be carried out using hardware elements. There are two approaches to solve this problem. A dedicated MAC unit may be implemented in hardware, which integrates multiplier and accumulator in a single hardware unit. This approach is adopted by the Motorola DSP processor DSP5600X. The other approach is to have multiplier and accumulator separate. For example, in the Texas Instruments DSP processor, 320C5X, the output of the multiplier is stored into the product register. The content of this in turn can be added to accumulator register ACC in the central ALU. In

both of the above approaches, the MAC operation can be completed in one clock cycle. The presence of H/W multipliers and/or multiplier accumulator is one of the mandatory requirements of a P-DSP.

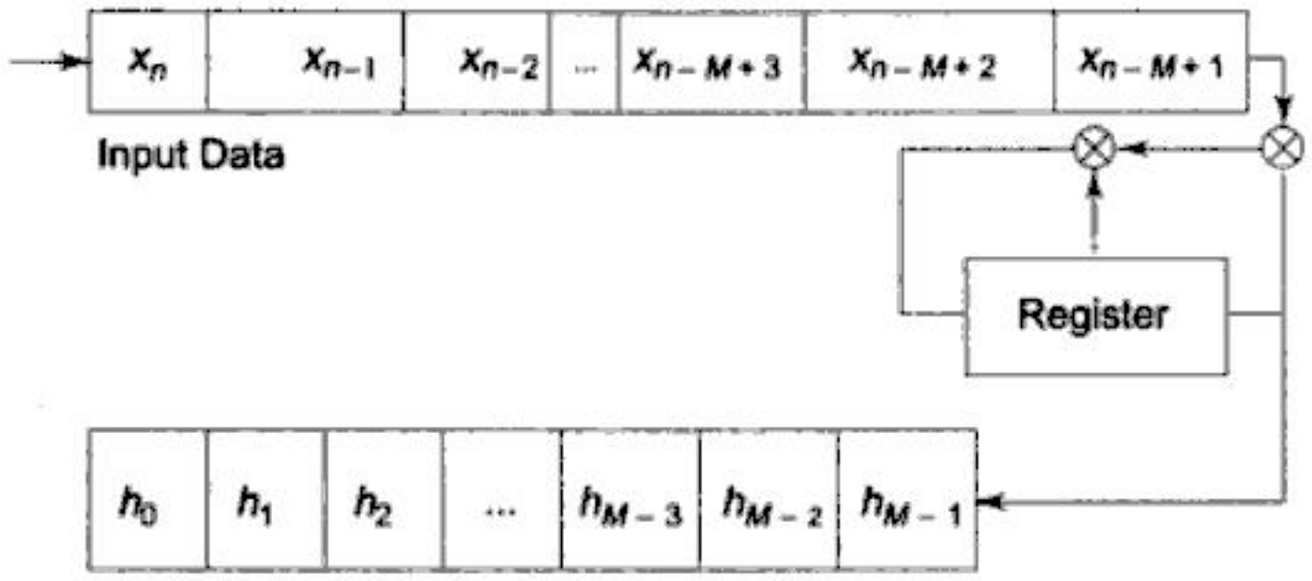


Fig. 2.1 *Implementation of convolver with single multiplier/adder*

In Fig. 2.1, y_n , the output at the nth sampling instant, is obtained by multiplying the array $\boldsymbol{x}_n = [x_n\ x_{n-1}\ x_{n-2} \cdots, x_{n-M+3}\ x_{n-M+2}\ x_{n-M+1}]$ corresponding to the present and the past $M - 1$ samples of the input with the array $\mathbf{h} = [h_0\ h_1\ h_2 \ldots h_{M-3}\ h_{M-2}\ h_{M-1}]$ corresponding to the impulse response sequence. To obtain y_{n+1}, the input signal array $\boldsymbol{x}_{n+1}$ is multiplied with the array $\mathbf{h}$. The vector $\boldsymbol{x}_{n+1}$ is obtained by shifting the array $\boldsymbol{x}_n$ towards right so that the $(n + 1)^{\text{th}}$ sample of the input data x_{n+1} becomes the first element and all the elements of $\boldsymbol{x}_n$ are shifted towards right by 1 position so that the i^{th} element of $\boldsymbol{x}_n$ becomes the $(i + 1)^{\text{th}}$ element of $\boldsymbol{x}_{n+1}$. Instead of shifting the elements of $\boldsymbol{x}_n$ towards right all at a time after finishing the vector multiplication, each of the elements may be shifted separately soon after the MAC operation that uses these elements is over. For example, after obtaining the product $x_{n-M+1}\,h_{M-1}$, the element x_{n-M} may be made to be equal to x_{n-M+1}. Similarly, after obtaining the product $x_{n-M+2}\,h_{M-2}$, the element x_{n-M+1} may be made equal to x_{n-M+2} and so on.

This is achieved in P-DSP by using a special instruction called MACD multiply accumulate with data shift. For example, TMS320C5*X* has the instruction MACD pgm, dma, which multiplies the content of the program memory pgm with the content of the data memory with address dma and stores the result in the product register. The content of product register is added to the accumulator before the new product is stored. Further, the content of dma is copied to the next location whose address is dma + 1.

2.2 MODIFIED BUS STRUCTURES AND MEMORY ACCESS SCHEMES IN P-DSPs

It may be noted that the MAC operation with data move (i.e. the MACD instruction) requires four memory accesses per instruction cycle. (An instruction cycle is the time that elapses since an instruction is fetched till the particular instruction completes execution including the time taken for writing the result into a register or memory. Many of the instructions in P-DSPs including the MACD instruction require only one processor clock period/instruction cycle. In the conventional microprocessors one instruction cycle corresponds to several clock periods.) The four memory accesses/clock period required for the MACD instructions are as follows:

1. Fetch the MACD instruction from the program memory
2. Fetch one of the operands from the program memory
3. Fetch the second operand from the data memory

4. Write the content of the data memory with address dma into the location with the address dma + 1

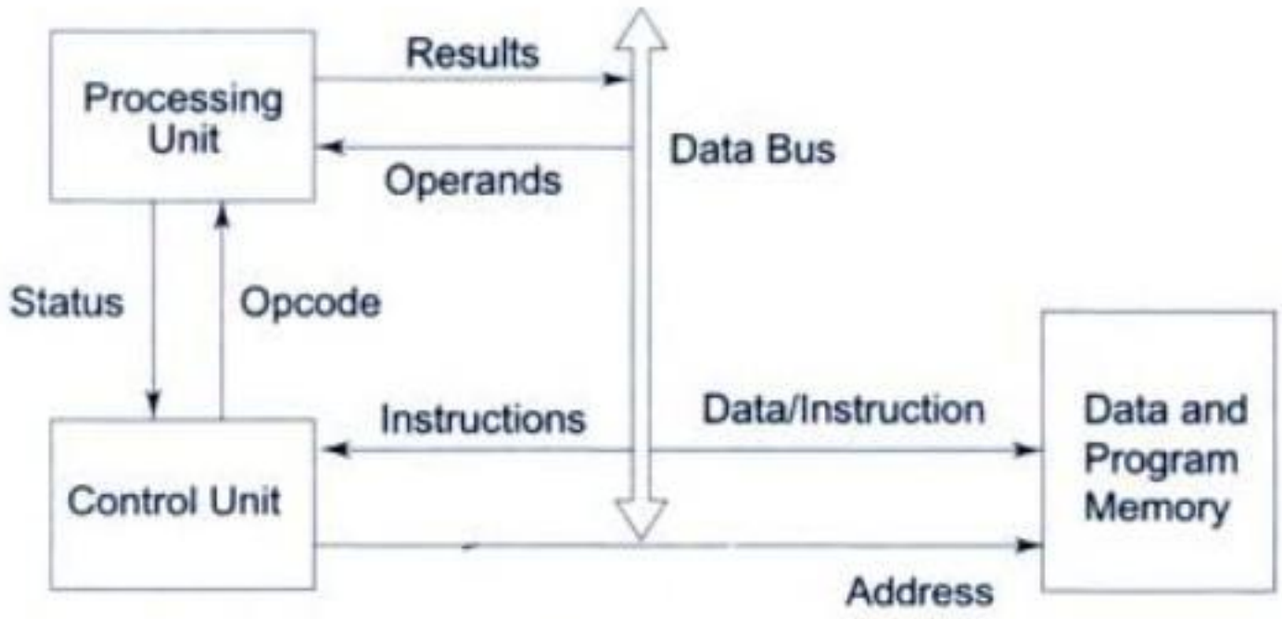


Fig. 2.2 *Von Neumann architecture*

The relatively static impulse response coefficients are stored in the program memory and the samples of the input data are stored in the data memory. If the MACD instruction is to be executed in a machine with *Von Neumann architecture*, it requires four clock cycles. This is because in the Von Neumann architecture shown in Fig. 2.2 there is a single address bus and a single data bus for accessing the program as well as data memory area. One of the ways by which the number of clock cycles required for the memory access can be reduced is to use more than one bus for both address and data. For example in the *Harvard architecture* shown in Fig. 2.3, there are two separate buses for the

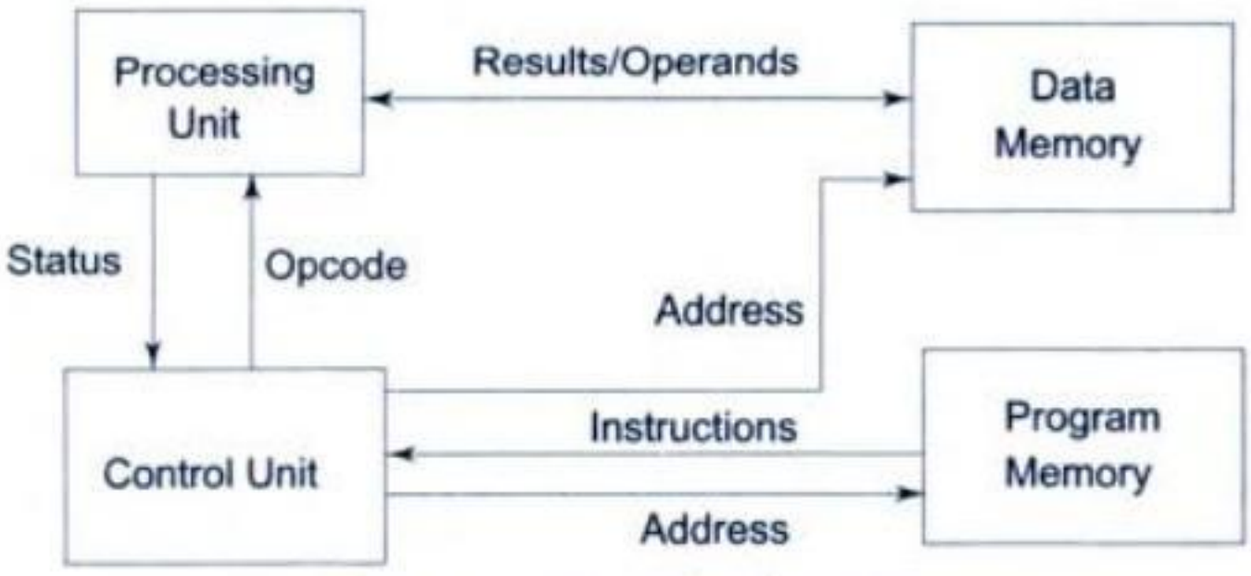


Fig. 2.3 *Harvard architecture*

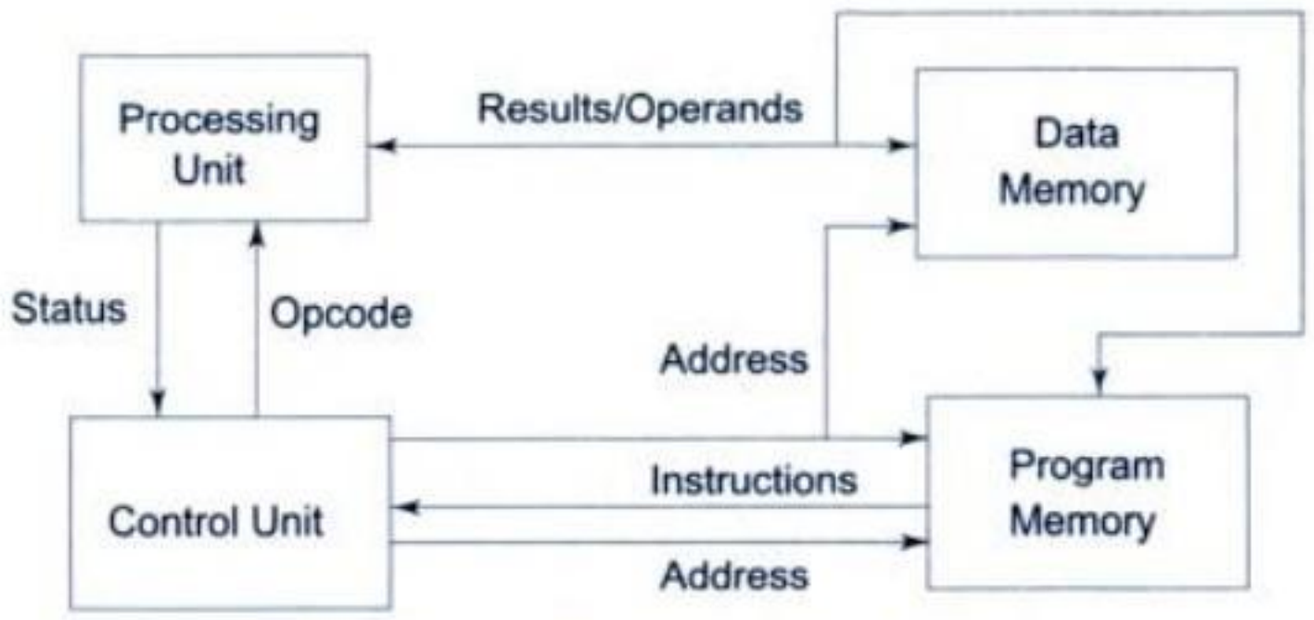


Fig. 2.4 *Modified harvard architecture*

program and data memory. Hence the content of program memory and data memory can be accessed in parallel. The instruction code can be fed from the program memory to the control unit while the operand is fed to the processing unit from the data memory. The processing unit consisting of the registers and processing elements such as MAC units, multiplier, ALU, shifter, etc., are also referred to as data path. The P-DSPs follow the modified Harvard architecture shown in Fig. 2.4. One set of bus is used to access a memory that has both program and data and another that has data alone. Data can also be transferred from one memory to another. The modified Harvard architecture is used in several P-DSPs, for example P-DSPs from Texas Instruments and Analog devices.

With the Harvard architecture, the number of memory accesses/clock cycle was shown to be two. This can be increased further by using more number of buses. For example, by using three separate address and data buses, the number of memory accesses/clock cycle can be increased to three. Motorola DSP5600X, DSP96002, etc. have three separate buses. TMS320C54X has four address buses.

Since the cost of an IC increases with the number of pins in the IC, extending a number of buses outside the chip would unduly increase the price. Hence the P-DSP's use multiple buses only for connecting the on-chip memory to the control unit and data path. For accessing off-chip memory only a single bus is used for accessing both the program memory and data memory. Because of this, any operation that involves an off-chip memory is slow compared to that using the on-chip memory.

2.3 MULTIPLE ACCESS MEMORY

The number of memory accesses/clock period can also be increased by using a high speed memory that permits more than one memory access/clock period. For example, the DARAM, the dual access RAM, permits two memory access/clock period. Multiple access RAM may be connected to the processing unit of the P-DSP by using the Harvard architecture. For example DARAM connected to a P-DSP with two independent data and address buses can be used to achieve four memory accesses/clock period.

2.4 MULTIPORTED MEMORY

Another technique that is adopted for increasing the number of accesses/clock period is to use multiported memory. For example the dual port memory has two independent data and address buses as shown in Fig. 2.5 and hence two memory accesses can be achieved in a clock period. Multiported memories dispense with the need for storing the program and data in two different memory chips in order to permit simultaneous access to both program and data memory. However, one of the major limitations of the dualported memory is the increase in the cost compared to two single port memory of the same total capacity. This is because of the increased number of pins and larger chip area required for the dualported memory. Larger number of I/O pins require a larger and more expensive package and a larger die size.

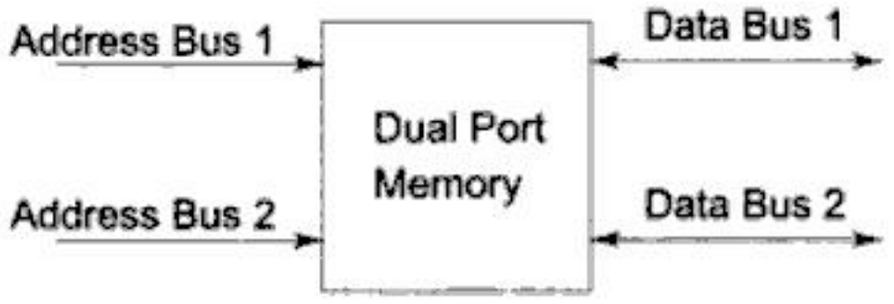


Fig. 2.5 *Block diagram of a dualported memory*

Some P-DSPs combine the modified Harvard architecture with the dualported memories. For example, the Motorola DSP 561XX processors have a singleported program memory and a dualported data memory. Hence one program memory access and two data memory accesses can be achieved per clock period.

2.5 VLIW ARCHITECTURE

Another architecture used for P-DSPs, for example in TMS320C6X, is the very long instruction word (VLIW) architecture. These P-DSPs have a number of processing units (data paths). In other words, they have a number of ALUs, MAC units, shifters, etc. The VLIW is accessed from memory and is used to specify the operands and operations to be performed by each of the data paths. As shown in Fig. 2.6, the multiple functional units share a common multiported register file for fetching the operands and storing the results. Parallel random access by the functional units to the register file is facilitated by the read/write cross bar. Execution of the operations in the functional units is carried out concurrently with the load/store operation of data between a RAM and the register file.

The performance gains that can be achieved with VLIW architecture depends on the degree of parallelism in the algorithm selected for a DSP application and the number of functional units. The throughput will be higher only if the algorithm involves execution of independent operations. For example, in Fig. 2.1, by using eight functional units, the time required for convolution can be reduced by a factor of 8 compared to the case where a single functional unit is used.

However, it may not always be possible to have independent stream of data for processing. Further the number of functional units is also limited by the hardware cost for the multiported register file and cross bar switch.

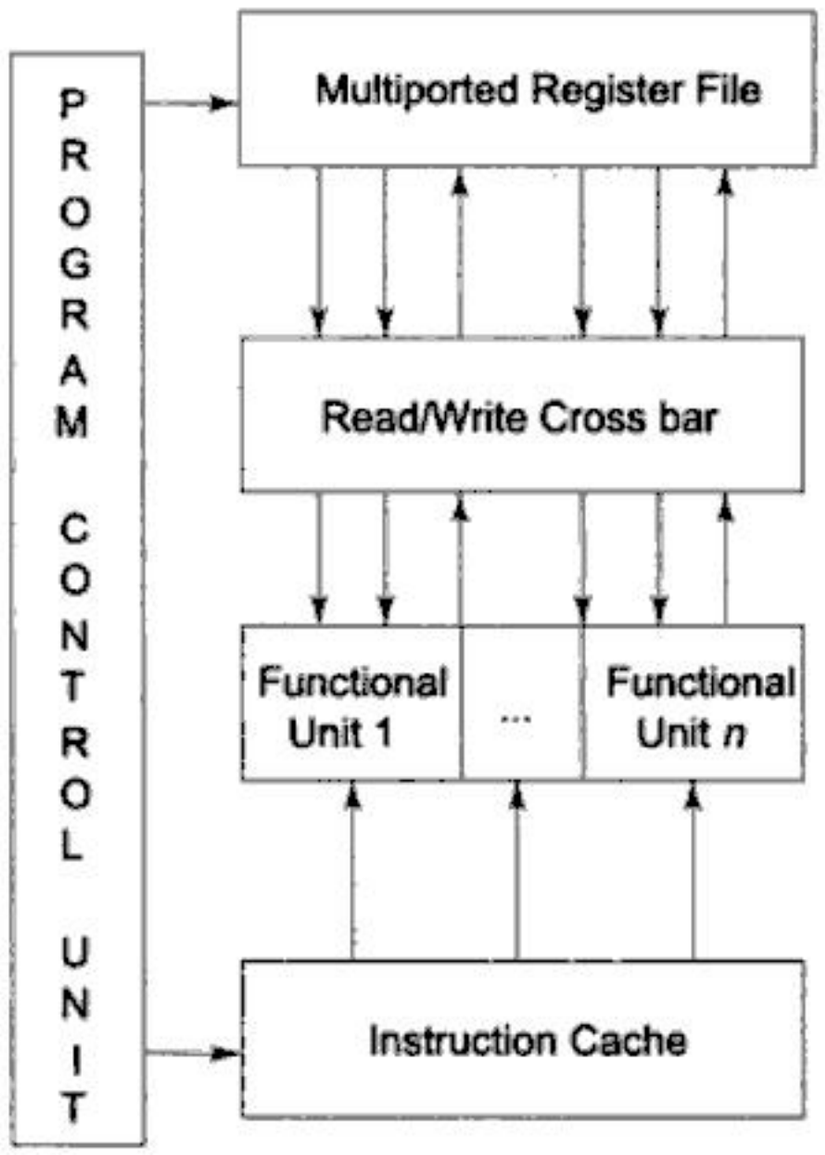


Fig. 2.6 *Block diagram of the VLIW architecture*

2.6 PIPELINING

One of the approaches adopted for increasing the efficiency of the advanced microprocessors as well as P-DSPs is instruction pipelining. An instruction cycle starting with the fetching of an instruction and ending with the execution of the instruction including the time storage of the results can be split into a number of microinstructions. Execution of each of the microinstructions is also referred to as one phase of an instruction. For example, an instruction cycle requiring four microinstructions can be said to be in four phases as follows:

1. Fetch phase in which the instruction is fetched from the program memory
2. Decode phase in which the instruction is decoded
3. Memory read phase in which the operand required for the execution of the instruction may be read from the data memory
4. Execution phase in which execution as well as the storage of the results in either one of the registers or memory is carried out

Each of the above microinstructions may be carried out separately by four functional units. Let us assume that each of the above four phases take equal time for completion. In this case in a conventional microprocessor with no pipelining, each of the functional units is busy only 25% of the time. This is because only one instruction is processed at the CPU at a time. Figure 2.7 shows when each of the functional unit is busy when a program containing three instructions I1,I2, I3 is executed.

Value of *T*	Fetch	Decode	Read	Execute
1	I1			
2		I1		
3			I1	
4				I1
5	I2			
6		I2		
7			I2	
8				I2
9	I3			
10		I3		
11			I3	
12				I3

Fig. 2.7 *Instruction cycles of processor with no pipelining*

The functional units can be kept busy almost all the time by processing a number of instructions simultaneously in the CPU. For example, in a machine with four functional units, four instructions I1, I2, I3 and I4 can be processed simultaneously as shown in Fig. 2.8. When I1 enters the decode phase, I2 can enter the opcode fetch phase. When I1 enters the operand read phase, I2 enters the decode phase and I3 enters the opcode fetch phase. When I1 enters the execute phase, I2 enters the operand read phase, I3 enters the decode phase and I4 enters the opcode fetch phase. The pipeline is fully loaded now and all the functional units have useful work to do. The instructions that follow I4 keep the functional units busy till the program is exited. Let *T* denote the time required for each phase of the

instruction. One clock cycle of the processor corresponds to T. In a period of $12T$ only three instructions can be executed in a machine without pipelining. In the same period nine instructions can be executed as shown in Fig. 2.8. Hence the throughput is increased by a factor of 3 in this case.

Value of T	Fetch	Decode	Read	Execute
1	I1			
2	I2	I1		
3	I3	I2	I1	
4	I4	I3	I2	I1
5	I5	I4	I3	I2
6	I6	I5	I4	I3
7	I7	I6	I5	I4
8	I8	I7	I6	I5
9	I9	I8	I7	I6
10		I9	I8	I7
11			I9	I8
12				I9

Fig.2.8 *Instruction cycles of a processor with pipelining*

It may be noted that the initial latency of a machine with four phases is $4T$. Hence for executing a program with N instructions, the time required for execution is $(N + 4)T_s$. With a non-pipelined machine, the time required for executing N instructions is $4NT$.

Instruction pipeline shown in Fig. 2.8 corresponds to a highly optimistic case. In the case of processors requiring single clock cycle for execution for each of the instructions in the program, the throughput shown in Fig. 2.8 can be achieved. This is normally achieved with restricted instruction set computers (RISC). However in complex instruction set computers (CISC), there are also instructions with multiple word requiring multiple clock cycles for execution. In this case all the functional units cannot be kept busy all the time. For example, in the case of call and branch instructions of a P-DSP, four phases or T states are required for the call/branch instruction to exit execution phase. By that time two more single word instructions or one double instruction enters the instruction pipeline. These instructions should not be executed. Hence two words have to be flushed out of the instruction pipeline before the instructions are fetched starting from the new program address.

To overcome this problem, some of the P-DSPs have special branch/call and return instructions called as delayed branch/call/return instructions. When the delayed branch instruction is executed, the program branches to the new program address only after the two 1-word instructions or the single 2-word instruction following the branch instruction are completely executed. Similarly, when the delayed call instruction is executed, the program calls to the subroutine only after the two 1-word instructions or the single 2-word instruction following the call instruction are completely executed. When the delayed call/branch/return instructions are executed, there is no need for flushing the pipeline and maximum throughput is obtained. Examples of pipeline operation of delayed as well as undelayed branch/call instructions are given in Chapter 4.

The throughput efficiency of the pipeline may also be reduced because of conflicts between the instructions in the instruction pipeline in different phases. This happens if the same memory is used to store the data and program and there is only a single address bus for addressing both the program and

data memory. This is true in the case of off-chip memory. For example, an instruction in fetch phase may try to fetch the instruction code from a memory chip that is also accessed by another instruction that is in the operand read phase. To avoid the conflict, the operand read phase will be done first and the opcode fetch phase will be repeated till there is no conflict again.

The number of instructions that are processed simultaneously in the CPU, also referred to as depth of the instruction pipeline, differs in different families of P-DSPs. The pipeline depths of some of the P-DSPs are given in Table 2.1.

Table 2.1 Instruction pipeline depth of some P-DSPs

P-DSP Name/family	*Pipeline Depth*
Analog devices	2
Motorola DSP5600X	3
TI TMS320C5X	4
TI TMS 320C54X	5

2.7 SPECIAL ADDRESSING MODES IN P-DSPs

In addition to the addressing modes such as direct, indirect and immediate supported by the conventional microprocessors, P-DSPs have special addressing modes that permit single word/instruction format and thereby speed up the execution by making effective use of the instruction pipelining. Further there are also special addressing modes such as cyclic addressing and bit reversed addressing that are specifically tailored for DSP applications. The details of these addressing are presented next.

2.7.1 Short Immediate Addressing

This permits the operand to be specified using a short constant that forms part of a single word instruction. The length of the short constant depends on the instruction type and the P-DSP. For example in the case of TI TMS320C5X, an 8-bit constant can be specified as one of the operands in the single word instructions for addition, subtraction, AND, OR, XOR, etc.

2.7.2 Short Direct Addressing

This permits the lower order address of the operand of an instruction to be specified in the single word instruction. In the TI TMS320 DSPs, the higher order 9 bits of the memory are stored in the data page pointer and only the lower 7 bits are specified as a part of the instruction. Each contiguous block of 128 words is referred to as one page in the TI DSPs. The argument in the instruction specifies only the location within the current page. In the Motorola DSP5600X, short direct memory addressing permits a 6-bit address to be specified in the instruction.

2.7.3 Memory-mapped Addressing

The CPU registers and the I/O registers of the P-DSPs are also accessible as memory location. This is achieved by storing them in either the starting page or the final page of the memory space. For example, in TMS320C5X, page 0 corresponds to the CPU registers and I/O registers. In the case of Motorola DSP5600X, the last page of the memory space containing 64 locations is used as the memory map for the CPU and I/O registers. When these registers are accessed using memory mapped address-

ing modes, the higher address bits are not taken from the data page pointer and instead made to be 0 in the case of TI DSPs and made to be 1 in Motorola DSPs.

2.7.4 Indirect Addressing

In P-DSPs this addressing mode has a number of options. This permits an array of data to be processed in P-DSP to be efficiently fetched and stored. The address of the operands can be stored in one of the registers called *indirect address registers*. In the case of TI processors, the indirect address registers are called *auxiliary registers* ARS. Any of these registers can be updated when the operand fetched using these registers are being executed. This is made possible by having an additional ALU in the CPU core specifically for the indirect address registers or ARs. The ARs may be incremented or decremented either in steps of 1 or in steps specified by the content of an offset register. In the case of TI processors, the offset register is called an *INDX register*. In the P-DSPs from analog devices it is called the *modifier register*. The content of the indirect address registers may also be updated by a constant using bit reversed addressing mode explained in the next section. In the TI 5X processors the new address computed by the auxiliary ALU is not used for fetching the operand for the current instruction that is being decoded and is executed. It is used for fetching the operand that uses the indirect addressing mode next with this particular AR. For this reason, the indirect addressing mode used in TI 5X P-DSPs is called *indirect addressing mode* with post-increment/decrement. In Motorola DSP563XX, the updated indirect address register content may also be used to fetch the operand for the current instruction. Hence this mode is called the indirect addressing mode with pre-increment/decrement. In TI TMS320C54X processors both post-increment/decrement and pre-increment/decrement operations are supported.

2.7.5 Bit Reversed Addressing Mode

The bit reversed number representation is explained in Section 1.14. The binary pattern corresponding to a particular decimal number is obtained by writing the natural binary equivalent of the number in the reverse order so that the most significant bit of the natural binary number becomes the least significant bit of the bit reversed no and vice versa.

For the computation of the FFT, the data is to be arranged in the bit reversed order and 2-point DFT of the resulting sequence is to be computed first. In the bit reversed addressing mode, when a 16-point FFT is to be computed, 2-point DFT of X(0) and X(8) is to be found. Similary 2-point DFT of X(4) and X(12) and so on. It may be noted from Table 1.1 that the value 0, 8, 4, 12 corresponds to the consecutive numbers in the bit reversed number representation. In the bit reversed addressing mode, the address is incremented/decremented by the number represented in the bit reversed form.

2.7.6 Circular Addressing

In real time processing of signals, the input signal is continuously stored in the memory. The processed data is stored in another memory space continuously and may be written onto the output device. In this case input as well as output program will be simple. However, since the input as well as output memory space will be finite in size, the entire memory space would be exhausted after processing the input signal for some time, if the data is written into the memory by using linear addressing mode. One way to overcome this problem is to keep checking whether the range of either the input or the output memory space is exceeded. In that case, the new data is to be stored starting from the beginning of the

particular memory space. However, checking this condition is an overhead that can be overcome using the circular addressing mode. In this mode, the memory can be organised as a circular buffer with the beginning memory address and the ending memory address corresponding to this buffer defined by the programmer. In the circular addressing mode, when the address pointer is incremented, the address will be checked with the ending memory address of the circular buffer. If it exceeds that, the address will be made equal to the beginning address of the circular buffer.

2.8 ON-CHIP PERIPHERALS

The P-DSPs have a number of on-chip peripherals that relieve the CPU from routine functions. Further they also help to reduce the chip count on the DSP system based around P-DSP. Some of the on-chip peripherals in the P-DSPs and their functions are as follows.

2.8.1 On-chip Timer

Two of the common applications of the timers are generation of periodic interrupts to the P-DSPs and generation of the sampling clocks for the A/D converters. The timer mode can be programmed by the P-DSPs. The timers can generate a single pulse or a periodic train of pulses. They can also generate a single square wave or a periodic square wave. The period of the timer is also made programmable.

2.8.2 Serial Port

This enables the data communication between the P-DSP and an external peripheral such as A/D converter, D/A converter or an RS232 C device. These ports normally have input and output buffers so that the P-DSP writes or reads from the serial port in parallel form and the serial port sends and receives data to the peripherals in serial form. They also generate interrupts when the serial port output buffer is empty or the input buffer is full. These devices have parallel to serial and serial to parallel converter inbuilt into them. The shift clock can be fed either from the P-DSP or an external device can supply it. The serial ports can operate either in the asynchronous mode or in the synchronous mode. In the asynchronous mode, the transmit data and receive data lines alone are used for communication and bit clock is

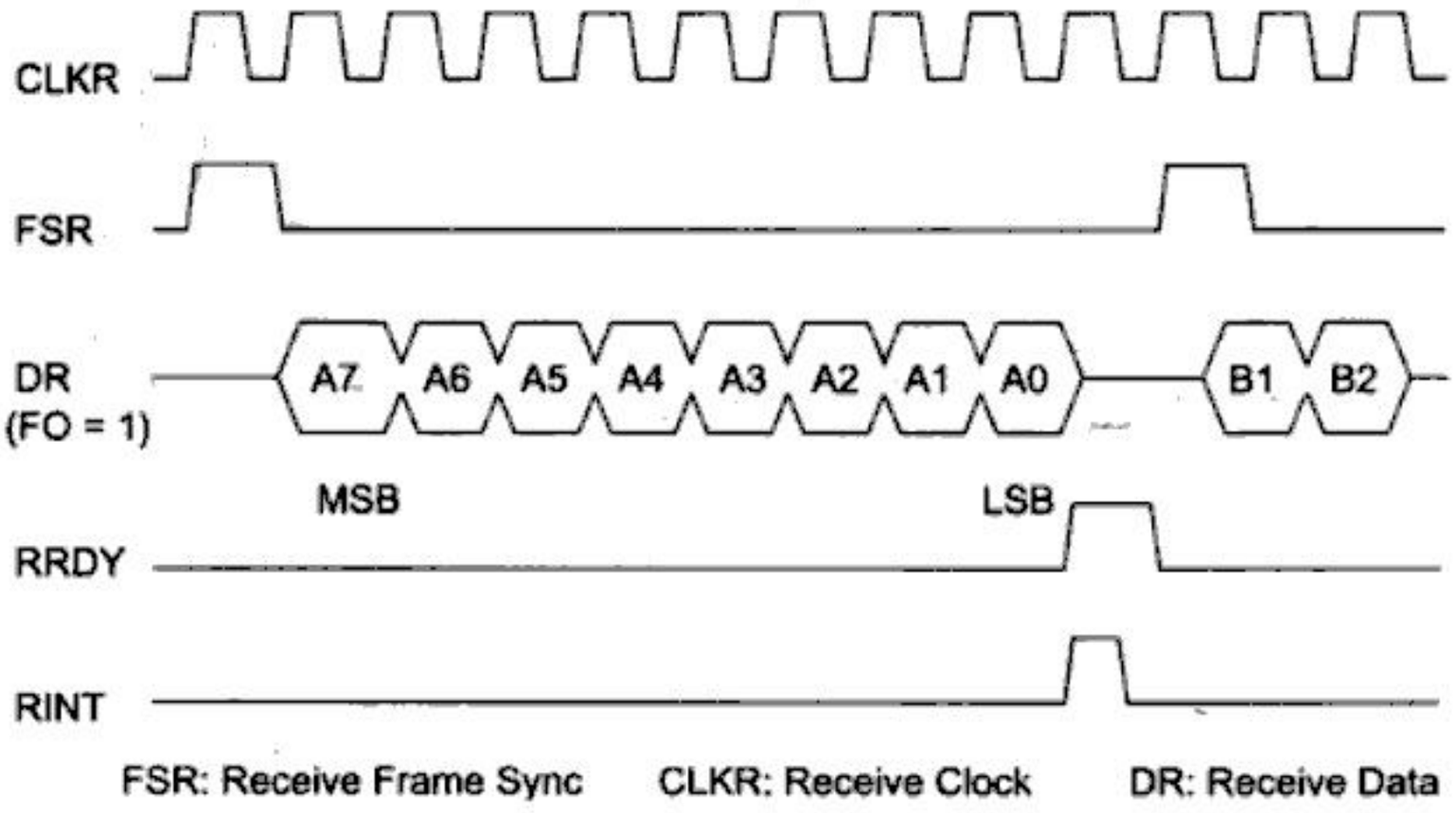


Fig. 2.9 *Burst mode serial port receive operation*

transmitted from either end. In the case of synchronous mode, both bit clock and a frame sync signal that indicates the beginning of the first bit of the data transmitted using synchronous mode is transmitted from the serial port to the I/O device and also from I/O port to the serial port. Example, of the two signals with respect to the transmitted data is shown in Fig. 2.9.

2.8.3 TDM Serial Port

The P-DSPs have a special serial port called TDM serial port. This permits a P-DSP to communicate with other devices or P-DSPs by using time division multiplexing (TDM). One of the devices can generate the frame sync pulse that indicates the beginning of a TDM frame and bit clock, the duration for which a bit is to be transmitted. As shown in Fig. 2.10 the TDM frame is split into a number of equal slots and each slot can be allotted for one of the devices.

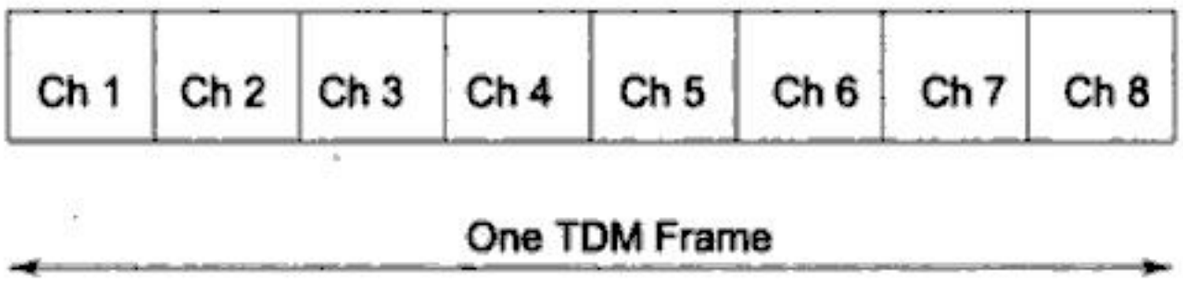


Fig. 2.10 *TDM frame with 8 time slots*

For example, in Fig. 2.10, there are 8 slots/frame and is referred to as a TDM with eight channels. In each of the slots, a number of bits may be transmitted by a channel. The TDM serial port normally uses four lines for the purpose of serial communication. They are

TFRM: the frame sync signal
TClock: the bit clock
TADD: The address of the serial device that is outputting data in a particular TDM slot
TDAT: The data transmitted into the TDM channel by the authorised device

The signals TADD and TDAT are bidirectional and are tristate controlled so that only one of the devices transmit the data and address in these lines at a time. Any one of the devices can generate the TFRM and clock signals and they are used by the other devices as a reference. A scheme where eight devices are interconnected using the TDM serial port is shown in Fig. 2.11. An example of how TI TMS320C5X can be configured to be one of the devices is shown in Fig. 2.12. An example, of each of the devices outputting a 16-bit data (D15 - D0) in its slot and also the address of the device (A0-A15), which is supposed to receive this data, is shown in Fig. 2.13.

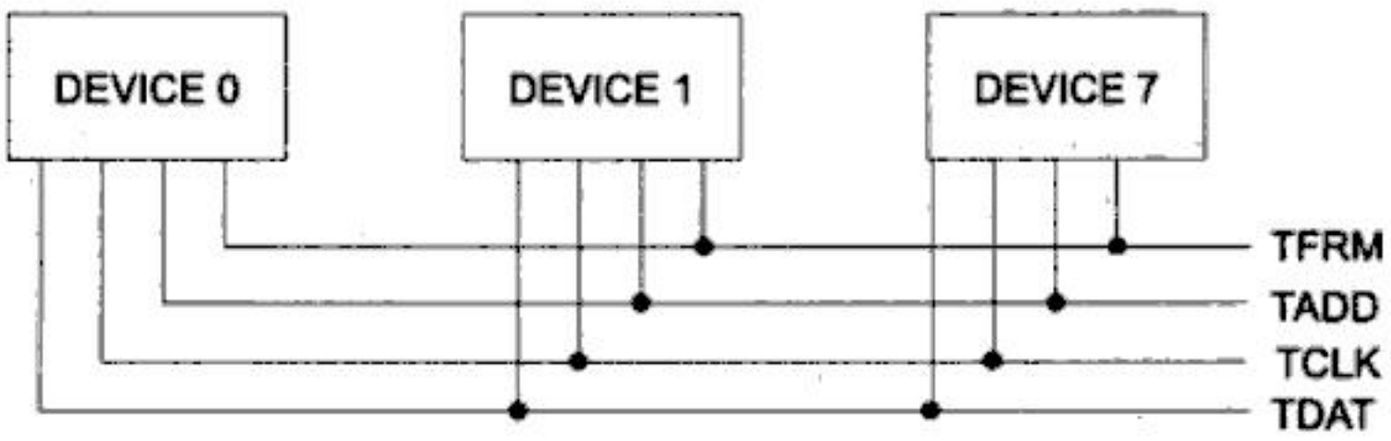


Fig. 2.11 *Interconnecting 8 devices using TDM serial using 4-bit-bus*

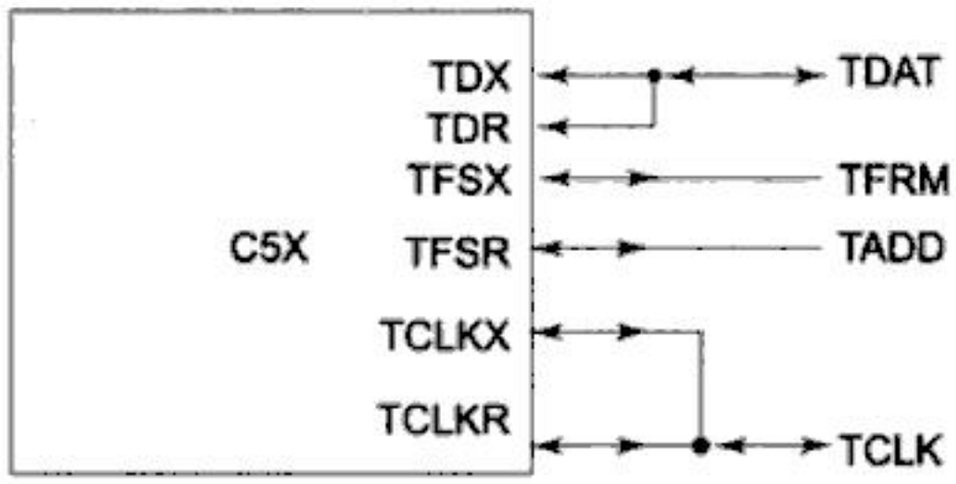


Fig. 2.12 *TMS320C5X configured to be one of TDM devices*

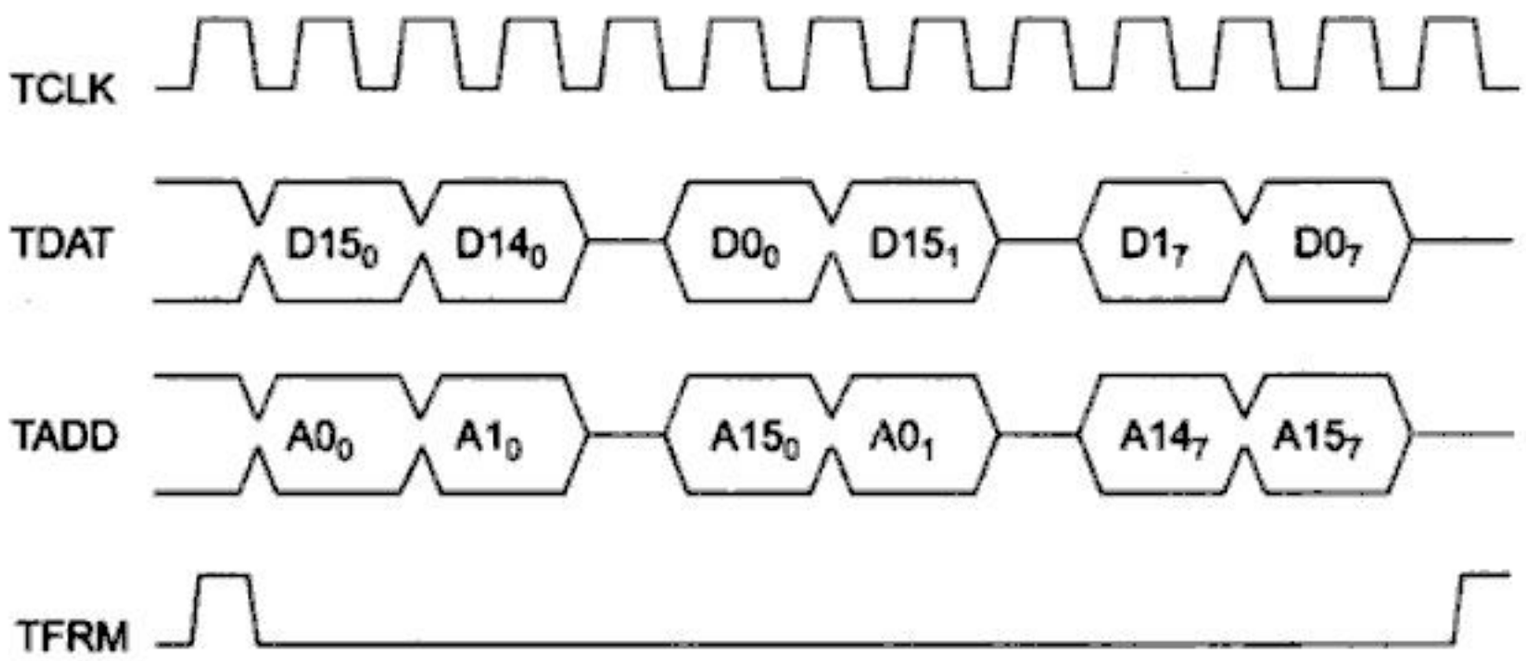


Fig. 2.13 *Data transfer using TDM channel*

2.8.4 Parallel Port

Parallel ports enable communication between the P-DSP and other devices to be faster compared to the serial communication by using a number of lines in parallel. In addition, they also have additional lines, which are for strobing or for handshaking purposes. The P-DSPs have two approaches for assigning lines for parallel port. In one approach used by the TI, the data bus itself is used for parallel ports. This is achieved by allocating a specific address space for I/O and whenever this address space is addressed using the I/O instructions, the parallel port signals including the handshaking signals are sent over the data bus. In another approach, separate lines are dedicated for parallel ports including the handshaking signals.

2.8.5 Bit I/O Ports

The P-DSPs have additional I/O ports that are single bit wide. These port bits may be individually set, reset or read. These bits are normally used for control purposes but they can also be used for data transfer. There are no handshaking signals for these I/O ports. Some of these bits are also used for conditional branching or calls. For example, in TI processors there are instructions such as branch if I/O zero.

2.8.6 Host Port

The P-DSPs also have a special parallel port normally 8-bit or 16-bit wide called the *host port* that enables them to communicate with a microprocessor or PC, which is called as a host. In addition to data communication, the host can generate interrupts and also cause the P-DSP to load a program from ROM to the RAM on reset. Almost all the P-DSPs including the ones from Analog devices, Motorola and TI have host ports.

2.8.7 Comm Ports

These are parallel ports that are used for interprocess communication between a number of identical P-DSP in a multiprocessor system. For example, a multiprocessor system may be built using a number of TI TMS320C4X. For the purpose of communication of the data between these processors six comm ports each of width 8 bits is provided. Since the data to be processed may be 32 or more number of bits, the P-DSPs have provision for splitting the data in streams of 8 bits and also assemble the 8 bits into words of 32 bits. Analog devices DSP ADSP 2106X has 6 comm ports each of which is 4 bits wide.

2.8.8 On-Chip A/D and D/A Converters

Some of the P-DSPs targeted towards voice applications such as cellular telephones and tapeless answering machines have A/D and D/A converters inside the P-DSP. For example, Motorola DSP 561XX and Analog devices ADSP 21MSP5X both have the A/D and D/A on chip and permit effective sampling rates of about 8 kHz.

2.8.9 P-DSPs with RISC and CISC

P-DSPs may be implemented using either the RISC processor or the CISC processor. For example, TI TMS320C6X P-DSPs uses RISC processor and a large number of P-DSPs from Analog devices, Motorola and TI make use of CISC. For example, TI TMS32054X and the Motorola DSP563XX and analog devices ADSP 2100X make use of CISC. TI TMS320C8X has a RISC and four P-DSPs with CISC in a single core. The relative advantages of each of these processors are as follows:

RISC Advantages

The chip area dedicated to the realisation of the control unit is considerably reduced because of the reduced number of instructions. About 20% of the chip area may be used for the control unit in RISC. In CISC processors about 30 – 40% of the chip area is used up for the control unit. Therefore in a RISC there is more area available for incorporating other features.

As a result of considerable reduction in the control area, the CPU registers and the data paths (processing units) can be replicated and the throughput of the processor can be increased by applying pipelining and parallel processing.

In a RISC, all the instructions are of uniform length and take the same time for execution. Hence the dummy periods or hold periods in the instruction pipeline is reduced to the minimum. This increases the computational speed.

A simpler and smaller control unit in RISC has fewer gates. This reduces the propagation delay and increases the speed. Reduced number of instructions, formats and addressing modes result in simpler and smaller decoder, which, in turn, increase the speed.

In RISC processors, the delayed branch and call instructions can be effectively used and they improve the speed.

HLL support: Writing the programs in C and C++ relieves the programmer from learning the instruction set of a P-DSP and instead concentrate on the application. It increases the throughput of the programmer. Since RISC has a smaller number of instructions, the compiler for any HLL is shorter and simpler. The availability of a relatively large number of CPU registers permits a more efficient code optimisation by maximising the use of CPU registers over slower memories.

CISC Advantages:

Some of the advantages of RISC also turn out to be disadvantages when viewed from a different perspective. The CISC processors have a very rich instruction set that even support high level language constructs similar to "if condition true then do", "for " and "while". The P-DSPs with CISC also have instructions specifically required for DSP applications such as MACD, FIRS, etc. This makes the application program written in the assembly language to be shorter and easy to follow. Since RISC has a smaller number of instructions, implementation of a single CISC instruction might require a number of instructions in RISC. This increases the memory required for storing the program and the traffic between CPU and memory is increased. This is on the one hand increases the computation time and on the other hand makes the program difficult to debug.

The HLL compilers are costly by several orders of magnitude compared to the P-DSPs themselves. For P-DSP with RISC architecture, compilers are essential. For most of the low cost applications, DSP platforms without the compilers are preferred. Hence a majority of P-DSPs are CISC based. The P-DSP manufacturers have tried to keep the codes for the new processors upward compatible with the older processors. This makes the learning curve steeper.

The relative disadvantages of each of these architectures are diminishing. By making the RISC processors applicable for larger and larger applications, the cost of the chip per se and the compiler costs are being brought down. The HLL compilers for the CISC processors are also becoming as efficient as hand assembly and the costs are coming down. Hence the distinction between the two in terms of cost and debugging efficiency is likely to narrow down further. The code composer studio from TI permits the programming in HLL as well as assembly language in a single development environment so that the best features of both the HLL and assembly language programming can be used by the programmer.

Review Questions

2.1 Explain why a MAC operation is implemented in hardware in programmable DSPs.
2.2 Explain how convolution is performed using a single MAC unit.
2.3 Explain the difference between a MAC instruction and MAC with data shift instruction. When is the latter instruction preferred?
2.4 Explain the difference between Von Neumann and Harvard architecture for the computer. Which architecture is preferred for DSP applications and why?
2.5 Explain why the P-DSPs have multiple address and data buses for internal memory and peripherals but have only a single address and data bus for the external memory and peripherals?
2.6 Explain the different techniques adopted for increasing the number of memory accesses/ instruction cycle.

2.7 Explain how a higher throughput is obtained using the VLIW architecture. Give an example, of a DSP that has VLIW architecture.
2.8 Explain what is meant by instruction pipelining. Explain with an example, how pipelining increases the throughput efficiency.
2.9 Explain how delayed branch/call instructions are superior to the undelayed branch/call instructions.
2.10 Explain the memory mapped addressing mode used in P-DSPs.
2.11 What are the different ways in which the operand for instructions can be specified using indirect addressing mode.
2.12 What is meant by bit reversed addressing mode? What is the application for which this addressing mode is preferred?
2.13 What is meant by circular addressing mode? What is the application for which this addressing mode is preferred?
2.14 Mention some applications of on-chip timer in P-DSPs.
2.15 Distinguish between the synchronous and asynchronous mode of operation of serial ports.
2.16 Explain the operation of TDM serial ports in P-DSPs.
2.17 What is the use of host ports in P-DSPs? How do they differ from the comm ports?
2.18 List the relative merits and demerits of RISC and CISC processors.

Self-test Questions

2.19 The features in which PDSP is superior to advanced microprocessors is ————————.
(a) Low cost (b) Low power
(c) Computational speed (d) Real time I/O capability
2.20 In modified Harvard architecture for fetching the content of program and data memory, a separate bus is used for ———————— memory and a single bus is used for ———————— memory.
2.21 Number of memory accesses/clock /period that can be achieved using on chip DARAM of a P-DSP is ————————.
(a) 1 (b) 2 (c) 3 (d) 4
2.22 VLIW architecture differs from conventional P-DSP in which of the following aspects:
(a) Instruction cache
(b) Number of functional units
(c) Use pipelining
(d) A single word fetched from memory has a number of instructions
2.23 A P-DSP has four pipeline stages and uses four phase clock. The number of clock cycles required for executing a program with 25 instruction is ————————.
(a) 29 (b) 28 (c) 25 (d) 26
2.24 The number of instruction cycles required for executing a program in a microprocessor with no pipelining is————————.
(a) 1 (b) 2 (c) 3 (d) 4
2.25 The addressing mode that is convenient for FFT computation is————————.
(a) Indirect addressing (b) Circular mode
(c) Bit reversed addressing (d) Memory mapped

2.26 The addressing that permits the content in internal register of the CPU & I/O to be accessed as memory location is————————.

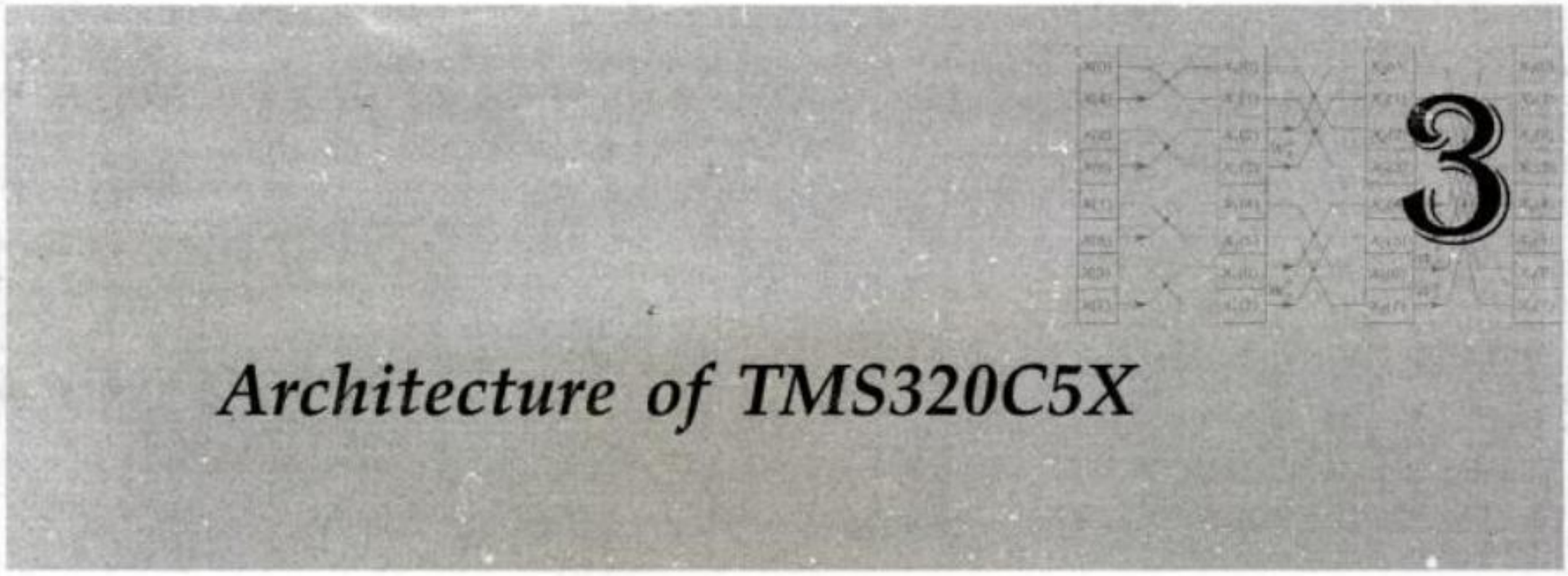
(a) Indirect addressing (b) Circular mode
(c) Bit reversed addressing (d) Memory mapped

2.27 The serial port that permits the data from a number of I/O devices to be sent using a single serial port is called ————————.

(a) Comm port (b) Host port
(c) Time division multiplexing (d) Bit I/O port

2.28 Which of the following characteristics are true for a RISC processor?

(a) Smaller control unit (b) Small instruction set
(c) Short program length (d) Less traffic between CPU & memory



3

Architecture of TMS320C5X

3.1 INTRODUCTION

Leading manufacturers of integrated circuits such as Texas Instruments (TI), Analog Devices and Motorola manufacture the digital signal processor (DSP) chips. These manufacturers have developed a range of DSP chips with varied complexity. The underlying concepts are broadly the same. Some of these concepts are discussed in Chapter 2. In order to give a feel for the design of systems with DSP chips, in this chapter, some details on the design of systems using the TMS320C5X DSP chip (denoted in brief as 5X) manufactured by TI are given.

The TMS320 DSP family consists of two types of single-chip DSPs: 16-bit fixed-point and 32-bit floating-point. These DSPs possess the operational flexibility of high-speed controllers and the numerical capability of array processors. Combining these two qualities, the TMS320 processors are inexpensive alternatives to custom fabricated VLSI and multichip bit-slice processors. TMS320C5X belongs to the fifth generation of the TI's TMS320 family of DSPs. The first five generations of TMS320 family are C1X, C2X, C3X, C4X and C5X. The C1X, C2X, C2XX and C5X are 16-bit fixed-point processors. Instruction sets of the higher generation fixed-point processors are upward compatible to the lower generation fixed-point processors. For example C5X can execute the instructions of both C1X and C2X. The 54X is upward compatible with 5X. C3X and C4X are 32-bit floating-point processors and C4X is upward compatible with C3X instruction set. The sixth generation C6X devices feature VelociTI™, an advanced very long instruction word (VLIW) architecture developed by TI and can execute 1600 MIPS. The eighth generation C8X devices, have, on a single piece of silicon, a number of advanced DSPs (ADSPs) and a RISC master processor. Typical application of the above families of TI DSPs are as follows:

C1X, C2X, C2XX, C5X, C54X: toys, hard disk drives, modems, cellular phones and active car suspensions

C3X: filters, analysers, hi-fi systems, voice mail, imaging, bar-code readers, motor control, 3D graphics or scientific processing

C4X: parallel-processing clusters in virtual reality, image recognition telecom routing, and parallel-processing systems.

C6X: wireless base stations, pooled modems, remote-access servers, digital subscriber loop systems, cable modems and multichannel telephone systems

C8X: video telephony, 3D computer graphics, virtual reality and a number of multimedia applications

The TI DSP chips have IC numbers with the prefix TMS320. If the next letter is C (e.g. TMS320C5X), it indicates that CMOS technology is used for the IC and the on-chip non-volatile memory is a ROM. If it is E (e.g. TMS320E5X) it indicates that the technolgy used is CMOS and the on-chip non-volatile memory is an EPROM. If it is neither (e.g. TMS3205X), it indicates that NMOS technology is used for the IC and the on-chip non-volatile memory is a ROM. Under C5X itself there are three processors, 'C50, 'C51 and 'C5X, that have identical instruction set but have differences in the capacity of on-chip ROM and RAM. The characteristics of some of the TMS320 family DSP chips are given in Table 3.1.

The instruction set of TMS320C5X and other DSP chips is superior to the instruction set of conventional microprocessors such as 8085, Z80, etc., as most of the instructions require only a single cycle for execution. The multiply accumulate operation used quite frequently in signal processing applications such as convolution requires only one cycle in DSP.

Table 3.1 Characteristics of some of the TMS320 family DSP chips

	'C15	'C25	'C30	'C50	'C541
Cycle time (ns)	200	100	60	50	25
on chip RAM	4K	4K	4K	2K	5K
Total memory	4K	128K	16M	128K	128K
Parallel ports	8	16	16M	64K	64K

Architecture of TMS320C5X DSPs

The block diagram of the internal architecture of C5X is shown in Fig. 3.1. The 320C5X DSPs are said to have advanced Harvard architecture because they have separate memory bus structures for program and data and have instructions that enable data transfer between the program and data memory area.

3.2 BUS STRUCTURE

Separate program and data buses allow simultaneous access to program instructions and data, providing a high degree of parallelism. For example, while data is multiplied, a previous product can be loaded into, added to or subtracted from the accumulator and, at the same time, a new address can be generated. Such parallelism supports a powerful set of arithmetic, logic and bit-manipulation operations that can all be performed in a single machine cycle. In addition, the 'C5X includes the control mechanisms to manage interrupts, repeated operations and function calling.

The 'C5X architecture has four buses and their functions are as follows:

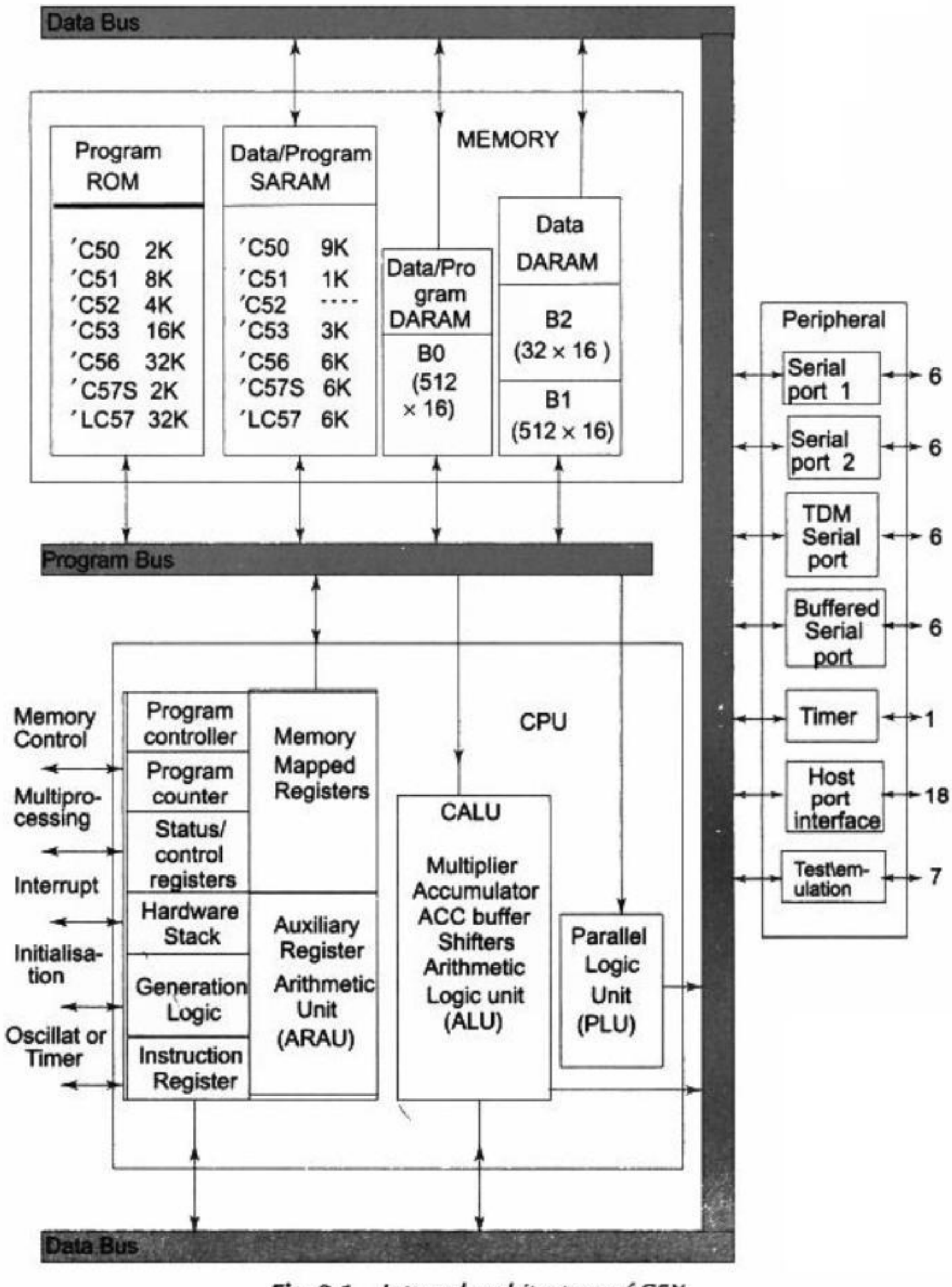


Fig. 3.1 *Internal architecture of C5X*

Program bus (PB): It carries the instruction code and immediate operands from program memory space to the CPU.

Program address bus (PAB): It provides addresses to program memory space for both reads and writes.

Data read bus (DB): It interconnects various elements of the CPU to data memory space.

Data read address bus (DAB): It provides the address to access the data memory space. The program and data buses can work together to transfer data from on-chip data memory and internal or external program memory to the multiplier for single-cycle multiply/accumulate operations.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

ARB Auxiliary Register Buffer

This 3-bit field holds the previous value contained in the ARP in ST0. Whenever the ARP is loaded, the previous ARP value is copied to the ARB, except when using the LST #0 instruction. When the ARB is loaded using the LST #1 instruction, the same value is also copied to the ARP. This is useful when restoring context (when not using the automatic context save) in a subroutine that modifies the current ARP.

CNF On-chip RAM configuration control bit. This 1-bit field enables the on-chip dual-access RAM block 0 (DARAM B0) to be addressable in data memory space or program memory space. The CNF bit can be modified by the LST #1 instruction. If CNF is 0, the on-chip DARAM block 0 is mapped into data memory space. The CNF bit can be cleared by a reset or the CLRC CNF instruction. When CNF is 1, the on-chip DARAM block 0 is mapped into program memory space. The CNF bit can be set by the SETC CNF instruction.

TC Test/control flag bit. This 1-bit flag stores the results of the ALU or parallel logic unit (PLU) test bit operations. The status of the TC bit determines if the conditional branch, call and return instructions are to be executed.

SXM Sign-extension mode bit. This 1-bit field enables/disables sign extension of an arithmetic operation. The SXM bit does not affect the operations of certain arithmetic or logical instructions; the ADDC, ADDS, SUBB or SUBS instruction suppresses sign extension, regardless of SXM.

C Carry bit. This 1-bit field indicates an arithmetic operation carry or borrow in the ALU. The single-bit shift and rotate instructions affect the C bit.

HM Hold mode bit. This 1-bit field determines whether the central processing unit (CPU) stops or continues execution when acknowledging an active HOLD signal.

XF pin status bit. This 1-bit field determines the level of the external flag (XF) output pin.

PM Product shift mode bits. This 2-bit field determines the product shifter (P-SCALER) mode and shift value for the PREG output into the ALU. Table 3.2 gives the PM bits and the function performed.

Table 3.2 PM bits and the function performed

PM bits	*Function*
b1 b0	P-SCALER mode for PREG output
0 0	No shift
0 1	Left-shifted 1 bit; LSB zero-filled
1 0	Left-shifted 4 bits; 4 LSBs zero-filled
1 1	Right-shifted 6 bits; sign extended; 6 LSBs lost. The product is always sign extended, regardless of the value of the SXM bit

3.13 ON-CHIP MEMORY

The 'C5X architecture contains a considerable amount of on-chip memory to aid in system performance and integration:

Program Read-Only Memory (ROM)
Data/Program Dual-Access RAM (DARAM)
Data/Program Single-Access RAM (SARAM)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Review Questions

3.1 Mention few applications of each of the families of TI DSPs
3.2 What are the different buses of TMS320C5X and their functions?
3.3 List the functional units in CALU of 5X and explain the source and destination of operands of each of these units.
3.4 List the various registers used with the ARAU and their functions.
3.5 What is meant by memory mapped register? How is it different from a memory?
3.6 List status register bits of 5X and their functions.
3.7 Distinguish between the dual-access RAM and single-access RAM used in the on-chip memory of 5X.
3.8 List the on-chip peripherals in 5X and their functions.
3.9 What are the various interrupt types supported by 5X?
3.10 Draw the internal architecture diagram of 5X and indicate the various blocks.

Self-test Questions

3.11 The 320C5X DSPs are said to have advanced Harvard architecture because
(a) they have separate memory bus structures for program and data
(b) they have instructions that enable data transfer between the program and data memory area
(c) they have same memory bus structures for program and data
(d) the contents of program memory cannot into the data memory or vice versa

3.12 The central ALU of C5X DSP processors have —— bit ALU and one of the operands for the ALU operation comes from ——.
(a) 32, ACC (b) 16, ACC (c) 32, ACCB (d) 16, ACCB

3.13 The result of operations performed in central ALU are stored in — .
(a) ACC (b) ACCB (c) TREG0 (d) PREG

3.14 The ALU register whose either higher order word or lower order word can be loaded from memory is.
(a) ACC (b) ACCB (c) TREG0 (d) PREG

3.15 The ——bit register used for temporary storage of accumulator is — .
(a) 32, PREG (b) 32, ACCB (c) 16, TREG0, (d) 32, ACC

3.16 The —— permits execution of logical operations on data without affecting the contents of ACC.
(a) parallel logic unit (b) auxiliary ALU (c) central ALU

3.17 The hardware multiplier unit in the C5X processors perform multiplication of — times—bit represented in — complement form.
(a) 16, 16, 1s (b) 8,8 1s (c) 16, 16, 2s (d) 8, 8, 2s

3.18 ——— holds the result of multiplication and is —— bit wide.
(a) PREG, 32 (b) PREG, 16 (c) TREG0, 16 (d) TREG0, 32

3.19 The register in which the multiplicand is stored before multiplication is performed is —— and is —— bit wide.
(a) PREG, 32 (b) PREG, 16 (c) TREG0, 16 (d) TREG0, 32

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

$0 \le lk \le 65535$ *lk*: Long Constant
ind: {* *+ *– *0+ *0– *BR0+ *BR0–}

Operands can be constants or assembly-time expressions that refer to memory, I/O ports, register addresses, pointers, shift counts and a variety of other constants.

The complete list of the mnemonics of the various instructions supported by 5X and a brief description of each of these instructions are given in Appendix A4. More detailed explanation of these instructions with examples is given in the TMS30C5X user reference manual. In this chapter, in Section 4.2 the various addressing modes supported by C5X are discussed. In Sections 4.3–4.5 some of the most commonly used C5X instructions are explained individually with examples. In chapter 6 application programs which make use of the instructions explained in the above sections are presented.

4.2 ADDRESSING MODES

C5X processors can address 64K words of program memory and 96K words of data memory. C5X supports the following six addressing modes:
Direct addressing
Memory-mapped register addressing
Indirect addressing
Immediate addressing
Dedicated-register addressing
Circular addressing
The details of each of these addressing modes are considered next.

4.2.1 Direct Addressing

The data memory used with C5X processors is split into 512 pages each of 128 words long. The data memory page pointer (DP) in ST 0 holds the address of the current data memory page. In the direct addressing mode of C5X, only lower-order 7 bits of the address are specified in the instruction. The upper 9 bits are taken from the DP as shown in Fig. 4.1.

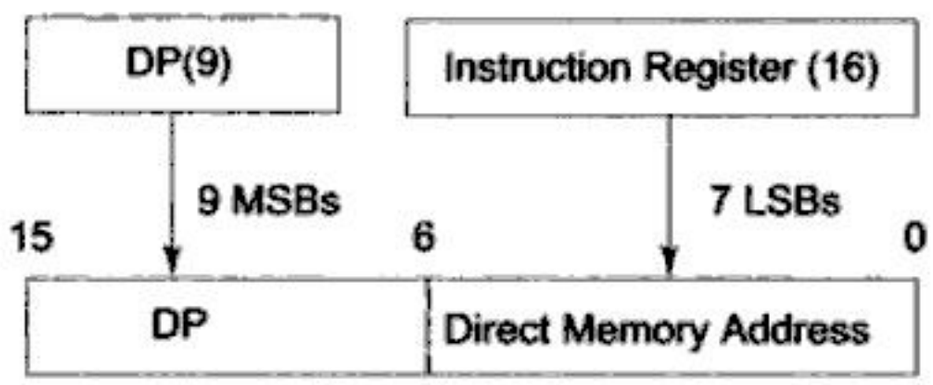


Fig. 4.1 *16-bit data memory address bus (DAB)*

4.2.2 Memory-Mapped Register Addressing

The RAM area in page 0 is used for storing some of the registers, interrupt vector addresses and so on. These locations can be accessed by specifying the actual address or by the register name. (e.g., the AR0 can either be denoted by the actual memory location (16h) used for storing its value or by the symbol AR0). Since these memory locations can be interchangeably used with the register names, the registers corresponding to page 0 are referred to as *memory-mapped registers* (MMRs).

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

the physical location of a data value. When INDX is added to the current AR, using bit-reversed addressing, addresses are generated in a bit-reversed fashion.

Example 4.5 Assume that the ARs are eight bits long, that AR2 represents the base address of the data in memory ($0110\ 0000_2$) and that INDX contains the value $0000\ 1000_2$. When the MAC (Multiply Accumulate) instruction MAC 0FF00h, *BR0+ is repeatedly executed eight times, the value of AR2 is modified as given in Table 4.2.

Table 4.2 Bit-reversed addresses example 4.5

Instruction executed	*Value of AR2*
MAC 0FF00h,*BR0+ ;	AR2 = 0110 0000 (0th value)
MAC 0FF00h,*BR0+ ,	AR2 = 0110 1000 (1st value)
MAC 0FF00h,*BR0+ ;	AR2 = 0110 0100 (2nd value)
MAC 0FF00h,*BR0+ ;	AR2 = 0110 1100 (3rd value)
MAC 0FF00h,*BR0+ ;	AR2 = 0110 0010 (4th value)
MAC 0FF00h,*BR0+ ;	AR2 = 0110 1010 (5th value)
MAC 0FF00h,*BR0+ ;	AR2 = 0110 0110 (6th value)
MAC 0FF00h,*BR0+ ;	AR2 = 0110 1110 (7th value)

4.2.6 Immediate Addressing

In immediate addressing, the instruction word(s) contains the value of the immediate operand. The 'C5X has both 1-word (8-bit, 9-bit and 13-bit constant) short immediate instructions and 2-word (16-bit constant) long immediate instructions. Table 4.3 lists the instructions that support immediate addressing.

Table 4.3 Instructions that support immediate addressing

Short Immediate (1-word)			
8-bit constant	*9-bit constant*	*13-bit constant*	
ADD ADRK LACL LAR RPT SBRK SUB	LDP	MPY	
Long immediate (2-word) 16-bit constant			
ADD	AND	APL	CPL
LACC	LAR	MPY	OPL
OR	RPT	RPTZ	
SPLK	SUB	XOR	XPL

4.2.6.1 Short Immediate Addressing

In short immediate instructions, the operand is contained within the instruction machine code.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

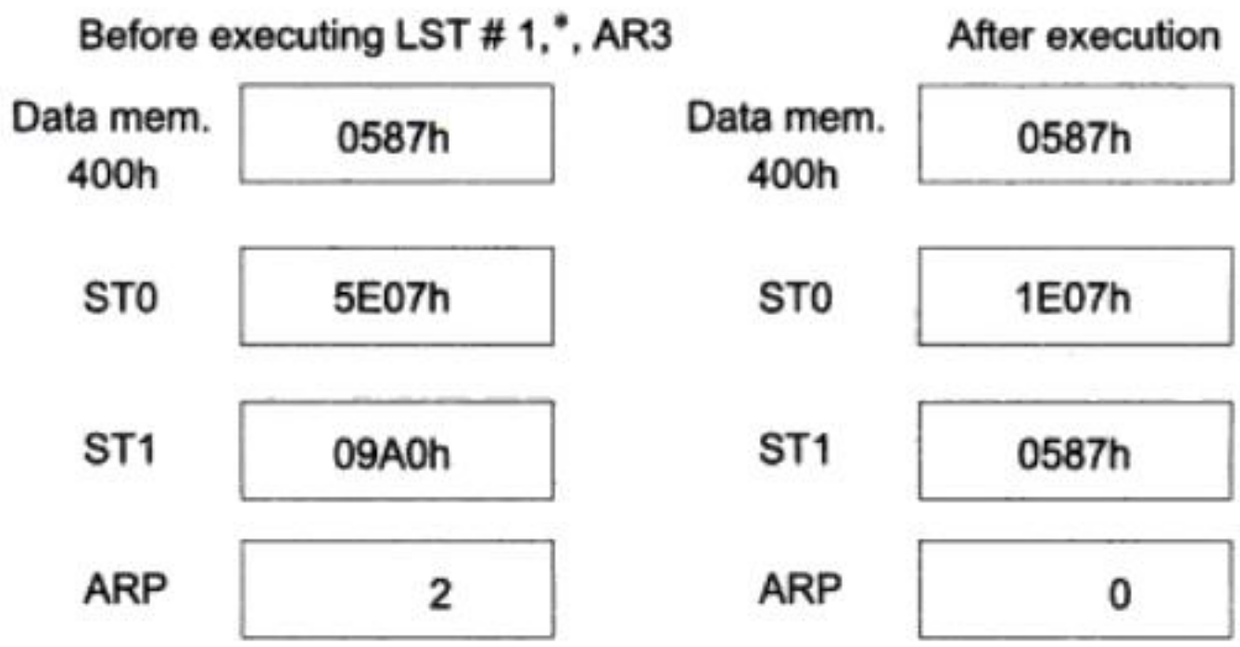


Fig. 4.8 *Loading ARP using indirect addressing*

The ARP can be modified using the MAR *+, ARn instruction. In this case, content of the current AR is incremented and the new value of ARP becomes *n*.

Example 4.14 Let the present value of ARP be 5. When the Instruction MAR *+, AR3 is executed, the value of ARP becomes 3. The content of AR5 is incremented.

4.4 ADDITION/SUBTRACTION INSTRUCTIONS

In the addition/subtraction instructions of C5X, one of the operand is ACC. The other operand can be PREG, ACCB or the content of memory fetched using one of the addressing modes. For the ADD and SUB instructions alone, the number fetched from memory, using dma, indirect addressing and immediate addressing with long constant, can be shifted left in the scaling shifter by 0–16 before performing the required operation.

Example 4.15 Let the initial content of ACC be 1234h. After executing the instruction ADD #2345h, 2, the content of ACC is as shown in Fig. 4.9.

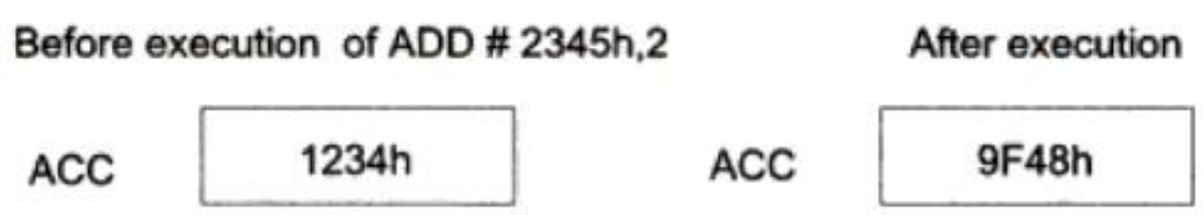


Fig. 4.9 *Addition using immediate addressing, example 4.14*

In this case the data 2345h is left shifted by two positions before it is added to ACC.

In the case of indirect addressing mode, for the ADD/SUB instructions, the third arguement (if it is present) denotes the new value to which ARP is to be updated after executing the instruction.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

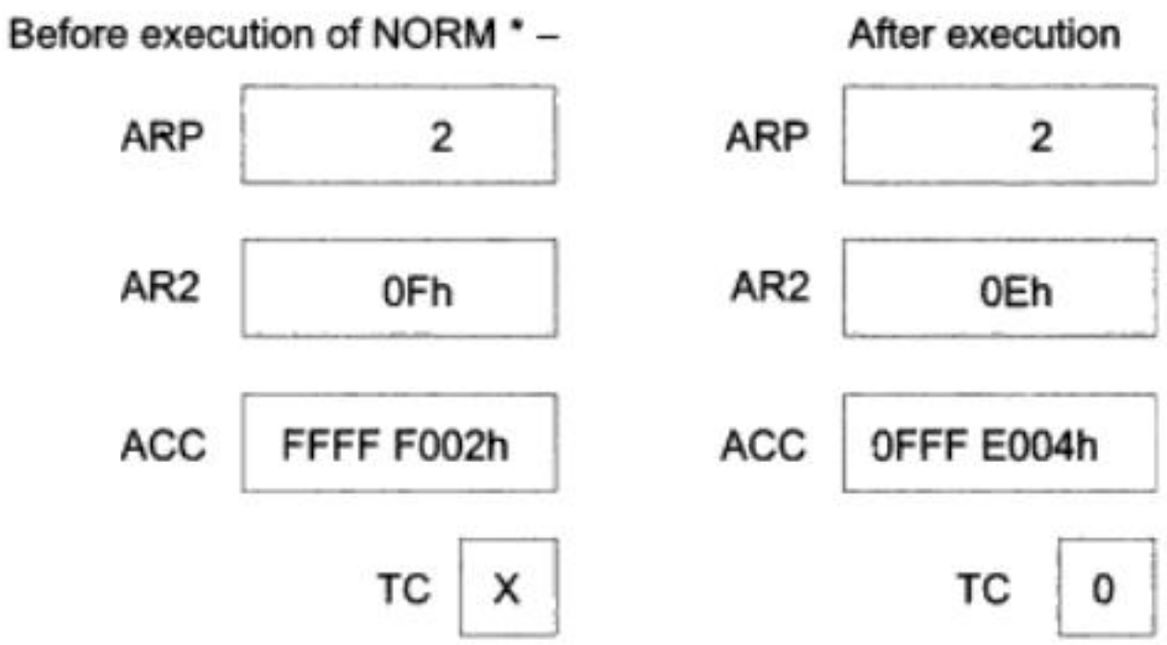


Fig. 4.19 *NORM instruction decrementing AR if ACC has extra sign bits*

4.8 PROGRAM CONTROL INSTRUCTIONS

4.8.1 Branch and Call Instructions

The C5X instruction set has both conditional and unconditional branch and call instructions. The branch and call instructions of C5X permit more than one condition to be tested using a single instruction. Branching occurs only if all the conditions are satisfied. The conditions which can be tested and the corresponding condition codes are given in Table 4.6.

Table 4.3 Condition codes for 5X

Conditions	*Condition codes*
ACC = 0	EQ
ACC ≠ 0	NEQ
ACC < 0	LT
ACC ≤ 0	LEQ
ACC > 0	GT
ACC ≥ 0	GEQ
C = 0	NC
C = 1	C
OV = 0	NOV
OV = 1	OV
TC = 0	NTC
TC = 1	TC
BIO low	BIO
Unconditionally	UNC

Example 4.31 BCND PGM1FFh, LEQ, OV In this case branch occurs to program memory address 1FFh only if ACC ≤ 0 and OV =1. Otherwise branching does not occur. The 'C5X performs speculative fetching by reading two additional instruction words. Hence if the conditions are not met, the fetched instructions would be executed. Otherwise these two instruction words are discarded.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

MADS	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; multiply data memory value by value specified in BMAR and store result in PREG
MPY	Multiply data memory value by TREG0 and store result in PREG Multiply short immediate by TREG0 and store result in PREG Multiply long immediate by TREG0 and store result in PREG
MPYA	Add PREG, with shift specified by PM bits, to ACC; multiply data memory value by TREG0 and store result in PREG
MPYS	Subtract PREG, with shift specified by PM bits, from ACC; multiply data memory value by TREG0 and store result in PREG
MPYU	Multiply unsigned data memory value by TREG0 and store result in PREG
PAC	Load PREG, with shift specified by PM bits, to ACC
SPAC	Subtract PREG, with shift specified by PM bits, from ACC
SPH	Store PREG high byte, with shift specified by PM bits, in data memory location
SPL	Store PREG low byte, with shift specified by PM bits, in data memory location
SPM	Set product shift mode (PM) bits
SQRA	Add PREG, with shift specified by PM bits, to ACC; load data memory value to TREG0; square value and store result in PREG
SQRS	Subtract PREG, with shift specified by PM bits, from ACC; load data memory value to TREG0; square value and store result in PREG
ZPR	Zero PREG

A4.5 Branch and Call Instructions

Mnemonic	Description
B	Branch unconditionally to program memory location
BACC	Branch to program memory location specified by ACCL
BACCD	Delayed branch to program memory location specified by ACCL
BANZ	Branch to program memory location if AR not zero
BANZD	Delayed branch to program memory location if AR not zero
BCND	Branch conditionally to program memory location
BCNDD	Delayed branch conditionally to program memory location
BD	Delayed branch unconditionally to program memory location
CALA	Call to subroutine addressed by ACCL
CALAD	Delayed call to subroutine addressed by ACCL
CALL	Call to subroutine unconditionally
CALLD	Delayed call to subroutine unconditionally
CC	Call to subroutine conditionally
CCD	Delayed call to subroutine conditionally
INTR	Software interrupt that branches program control to program memory location
NMI	Nonmaskable interrupt and globally disable interrupts (INTM = 1)
RET	Return from subroutine

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

4.24 The 16-bit register used for incrementing/decrementing the ARn in steps larger than 1 is ——————.

(a) ARCR (b) ARP (c) ARB (d) INDX

4.25 The contents of ARs are decremented /incremented using ——————.

(a) central ALU (b) auxiliary ALU (c) PLU

4.26 When an operand for an instruction is accessed using the indirect addressing mode and after the data is fetched, the content of the AR used for accessing the data is to be decremented by the number in INDX register, the addressing mode is specified by the symbol ——————.

(a) *0+ (b) *0- (c) *BR0+ (d) *BR0-

4.27 When an operand for an instruction is accessed using the indirect addressing mode and after the data is fetched, the content of the AR used for accessing the data is to be incremented by the number in INDX register with reverse carry propagation, the addressing mode is specified by the symbol ——————.

(a) *0+ (b) *0- (c) *BR0+ (d) *BR0-

4.28 The AR ALU (ARAU) performs —————— arithmetic on —————— numbers.

(a) unsigned, 16 (b) signed, 16 (c) signed, 32 (d) unsigned, 32

4.29 The symbol used to indicate the immediate address mode for the operand is ——————.

(a) $ (b) * (c) # (d) *- (e) *+

4.30 In the dedicated register addressing mode, the register whose contents are used if an immediate operand is unspecified is ——————.

(a) ARCR (b) BMAR (c) DBMR (d) ACCB

4.31 Before the instruction SBRK #5H is executed, the contents of ARP, AR3 and AR5 are 3H, 1058H and 1000H, respectively. After the execution of the instruction, the content of AR3 is ——————.

(a) 5H (b) 1053H (c) 1058H (d) 1000H

4.32 Assume that the contents of ACC, ARP, AR3 and locations 0045H, 40C5H are 1000H, 3, 40C5H and 2400, 2300H, respectively, initially. When the instruction LAMM * is executed, the content of ACC is ——————.

(a) 2400H (b) 2300H (c) 40C5H (d) 0003H

4.33 The mnemonic for the instruction used to move a word from data memory to program memory is ——————.

(a) BLDD (b) BLDP (c) BLPD (d) TBLR
(e) TBLW

4.34 The mnemonic for the instruction used to move a word from data memory to program memory and in which the program memory address is contained in ACC lower order word is ————.

(a) BLDD (b) BLDP (c) BLPD (d) TBLR
(e) TBLW

4.35 The mnemonic for the instruction which multiplies two 16-bit numbers represented in 2's complement form is ——————.

(a) MPYA (b) MPY (c) MPYU (d) MPYS
(e) MADD (f) MADS

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

From Table 5.2 it can be verified that the decode, read and execute functional units are doing unproductive operations in cycles 3, 4 and 5 respectively.

5.3.2 Pipeline Operation with Branch and Call Instructions

The branch instruction requires two cycles when branching does not occur and four cycles when branching occurs; one for the B instruction to enter the execute phase, one for fetching the branch address, two more for flushing out the one two-word or two 1-word instructions which enter the instruction pipeline after the branch instruction. The same is true with the call instructions. They require two cycles when the subroutine is not called. When the subroutine is called they also require four cycles out of which the last two cycles are required for flushing out the pipeline.

However, the delayed branch and call instructions permit the call and branching to be carried out in two clock cycles. The one 2-word instruction or two 1-word instructions following the delayed branch/call instruction are fetched from program memory and executed before the branch/call is carried. Hence instruction pipeline need not be flushed out after the call/branch instruction and the execution can resume from the branch address. Examples 5.3–5.6 illustrate these facts.

Example 5.3 When the following program

```
        ZAP
        B PGM1250h
        ADD *
        SACL *+
        MAC 4500h, 25h
PGM1250h: LACC *+
```

is executed, the instruction pipeline is loaded in different cycles as shown in Table 5.3.

Table 5.3 Instruction pipeline for branch instruction, Example 5.3

Cycle	*PC*	*Fetch*	*Decode*	*Read*	*Execute*
1	[B]	ZAP			
2	[1250h]	B	ZAP		
3	[ADD*]	1250h	B	ZAP	
4	[SACL*+]	ADD *	Dummy	B	ZAP
5	[LACC *+]	SACL*+	Dummy	Dummy	B
6		LACC*+	Dummy	Dummy	Dummy
7			LACC*+	Dummy	Dummy
8				LACC*+	Dummy
9					LACC*+

In Table 5.3, [B], [1250h], etc., denote the program memory (PGM) address where the branch instruction, the next operand, etc., are stored.

When the instruction B enters the execute phase, the ADD and SACL instructions enter the decode and fetch phases, respectively. However, since these two instructions are not to be executed, they are dummy phases and these instructions have to be flushed out of the instruction pipeline. This requires two cycles. Hence branch requires four cycles (one for the B instruction to enter the execute, one for fetching the branch address and two more for flushing out the unwanted instructions.)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

When the instruction LACC + enters the decode phase in cycle 6 the value of AR1 is 167h and this is used for accessing the data in its data read phase in cycle 6. Further even though LACC *+ enters the instruction pipeline after the SAMM instruction, it modifies AR1 to be 168h in its decode phase in clock cycle 6 itself. SAMM modifies the value of AR1 only in its execute phase in clock cycle 7.

In 7th clock cycle the ADD *+ enters the decode phase. At that time the value of AR1 is 168h and hence it uses this address when it enters the data read phase in cycle 8. It also modifies AR1 to 169h in cycle 7. But SAMM also tries to modify ARi to 164h. In this conflict only SAMM succeeds.

Example 5.7 Assume that the contents of the memory location 164h, 165h, 166h, 167h, 168h are as follows:
(164h) = 90h, (165h) = 80h, (166h) = 60h,
(167h) = 40h, and (168h) =30h.
When the following program

```
LAR AR1, #167h
LACC #164h
SAMM AR1
NOP
NOP
LACC *+
ADD *+
```

is executed the instruction pipeline at various clock cycles is as shown in Table 5.8.

Table 5.8 Instruction pipeline with SAMM followed by illegal instructions

Cycle	*PC*	*Fetch*	*Decode*	*Read*	*Execute*	*ACC*	*AR1*
1	[LACC]	LAR				XX	XX
2	[164h]	LACC	LAR			XX	XX
3	[SAMM]	164h	LACC	LAR		XX	XX
4	[NOP]	SAMM	Dummy	LACC	LAR	XX	167h
5	[NOP]	NOP	SAMM	Dummy	LACC	164h	167h
6	[LACC]	NOP	Dummy	SAMM	Dummy	164h	168h
7	[ADD]	LACC	Dummy	Dummy	SAMM	164h	164h
8		ADD	LACC	Dummy	Dummy	164h	165h
9			ADD	LACC	Dummy	164h	166h
10				ADD	LACC	90h	166h
11					ADD	110h	166h

In Example 5.7 what was expected was to add the content of location 164h, i.e., 90h, with that of 165h, i.e., 80h, and store the result in ACC. This actually happens as the SAMM instruction is followed immediately by two NOP instructions which do not make use of AR1. Expected results are obtained because of the the following reasons:

When the instruction LACC + enters the decode phase in cycle 8, the value of AR1 is 164h and this is used for accessing the data in its data read phase in cycle 9. The SAMM instruction modifies AR1 to 164h in its execute phase in clock cycle 7 itself. In this case since the decode phase of LACC *+ occurs after the execute phase of SAMM, no problem arises. In the decode phase of LACC *+, the value of AR1 is modified to 165h.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

6.2.2 Direct Addressing Mode

In direct addressing mode, the instruction word contains the location of the operand in a particular data page. The starting address of the operand page is indicated by the data page pointer (DP). It is loaded using the LDP instruction. Program 6_3.asm gives an example illustrating how the operations such as load accumulator and ARs, store the operands in data memory, add, subtract and multiply can be performed in direct addressing mode. It is also possible in some instructions to left shift the data memory content defined by the shift code in the instruction and the respective operations can be performed.

```
Label       Mnemonic            Comments
            .mmregs             ;includes memory map registers
            .ps 0a00h           ;origin of the program 0a00h
            .entry              ;program counter initialised
            LDP #20h            ;the data page no. 20h (32) is loaded into
                                ;  the data page pointer (DP)
            LACC 10h            ;content of 20h (32) page 10 th location
                                (i.e. ;content of data memory address
                                (dma) ;(1010h) is loaded into ACCU
            LACC 5h,2           ;content of dma 1005h is left shifted by 2
                                ;bits and then loaded into ACCU
            LDP #22h            ;DP loaded with 22h (dma page starting
                                ;address 1100h)
            LAR AR0,15h         ;AR0 loaded with content of dma 1115h
            SACL 15h            ;ACCU low byte stored into dma 1115h
            SACL 20h,3          ;ACCU low byte left shifted by 3 bits &
                                ;stored into dma 1120h
            SAMM AR7            ;ACCU low byte stored into AR7 in
                                ;page 0. DP remains unaffected
            LDP #12h            ;the data page no. 12h loaded into DP
            ADD 25h             ;the content of dma 0925h added to
                                ;ACCU & the result stored into ACCU
            ADD 7h,2            ;the content of dma 0907h left shifted by
                                ;2 bits & added to ACCU
            SUB 10h             ;the content of 0910h is subtracted from
                                ;the content of ACCU
            SUB 12h,2           ;the content of 0912h is left shifted by 2
                                ;bits & subtracted from ACCU
            SPLK #10h,TREG0     ;constant 10 stored into TREG0
            MPY 15h             ;content of 0915h is multiplied with
                                ;TREG0 & the result stored into PREG
            .end                ;program end
```

Program 6_3.asm Direct addressing mode

6.2.3 Indirect Addressing Mode

The address of the operand in indirect addressing mode is the content present in the current AR. The current AR is indicated by the auxiliary register pointer (ARP). Program6_4.asm gives an example for

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The value of n is stored in one of the ARs (AR0). First the series is generated and stored by using a loop. Using another loop the sum is calculated. Each time the loop is executed once, the AR having the value of n is decremented once its value decrements to zero the execution completes.

6.3.2 Generation of Fiboracci Series

The generation of the numbers corresponding to Fiboracce series is achieved using Program 6_8.asm. If the first two numbers of the sequence are assumed, the next number in the series is the sum of the previous two numbers. For example, $x(0) = 0$, $x(1) = 1$ then $x(i) = x(i\text{-}1) + x(i\text{-}2)$, where $i>1$. The length of the sequence 'n' is stored into one of the AR (AR1). The previous two numbers are needed for the generation of the next number. One number is stored in ACCU and another number is stored into the accumulator buffer ACCB. The sum

```
Label       Mnemonic              Comments
            .mmregs
            .ds 1000h
            .ps 0a00h
            .entry
            LAR  AR0,#1000h       ;starting dma (1000h) stored into AR0
            LAR  AR1,#10h         ;sequence length (N =10) loaded into AR1
            LACC #0h              ;zero ACCU
            EXAR                  ;contents of ACCU & ACCB exchanged
            LACC #01h             ;load the constant 1 into ACCU
loop:       MAR *,AR0             ;modify ARP to AR0
            SACL *,0              ;store ACCU low byte in dma pointed by AR0

            ADDB                  ;the content of ACCB added to ACCU
            EXAR                  ;contents of ACCU & ACCB exchanged
            SACL  *+,0,AR1        ;store the ACCU low byte in dma pointed
                                  ;by AR0, increment the content of AR0 and
                                  ;ARP points to AR1
            BANZ  loop,AR0        ;branch to loop if AR1 non-zero AR1
                                  ;decremented & ARP points to AR0
            .end                  ;program end
```

Program 6_8.asm Generating the Fibonacci series numbers

of these two registers give the next number in the series. Each time the next number is calculated, the content of AR1 is decremented, once the content of AR1 decrements to zero the generation of the additional numbers in the sequence is stopped.

6.3.3 Convolution Using MAC and MACD Instructions

The convolution of the following two sequences of length N and M is performed using MAC and MACD instructions in the following examples.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

6.4.1 The On-Chip Timer in C5X and Programming its Mode

The timer is an on-chip down counter that can be used to periodically generate CPU interrupts. The timer mode is programmed using the timer control register (TCR). The TCR diagram is shown in Fig. 6.4. The significance of some of the frequently used bits of the TCR is shown in Table 6.3. The remaining bits of TCR may be chosen to be 0.

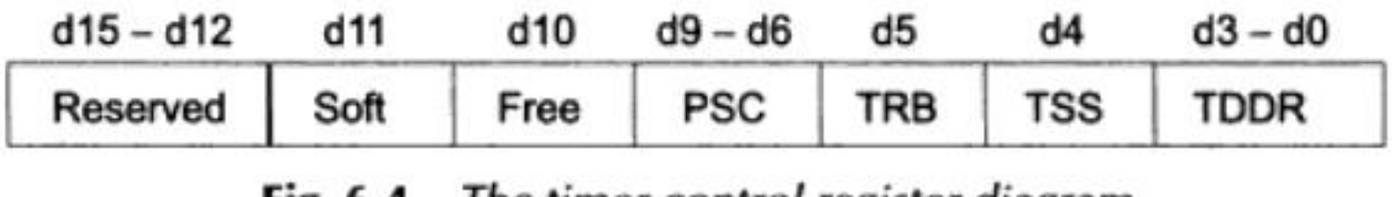


Fig. 6.4 *The timer control register diagram*

The timer is decremented by one at every CLKOUT1 cycle. A timer interrupt (TINT) is generated each time the counter decrements to zero. The timer provides a convenient means of performing periodic I/O or other functions. The timer interrupt rate, $TINT_{rate}$, is given by

$$TINT_{rate} = [t_c * (TDDR + 1) * (PRD + 1)]^{-1}$$

where t_c is period of CLKOUT1 of C5X. TDDR is a 4-bit register and its value is loaded by writing into the lower order 4 bits of TCR. PRD is a 16-bit memory-mapped register. For the timer interrupt to cause the DSP to branch to the interrupt service routine, the mask bit corresponding to TINT should be unmasked in the IMR. An example program for programming the on-chip timer is given in Program 6.11.asm.

Table 6.3 Significance of some of the Bits of the TCR

TCR bits	*Parameter name*	*Value on reset*	*Description*
d9–d6	PSC	—	Timer prescaler counter bits. These bits specify the count for the on-chip timer. When the PSC is decremented past 0 or the timer is reset, the PSC is loaded with the contents of the TDDR, and the TIM is decremented
d5	TRB	—	Timer reload bit. This bit resets the on-chip timer. When the TRB is set, the TIM is loaded with the value in the PRD and the PSC is loaded with the value in the TDDR. The TRB is always read as a 0
d4	TSS	0	Timer stop status bit. This bit stops or starts the on-chip timer. At reset, the TSS bit is cleared and the timer immediately starts timing. TSS = 0 The timer is started TSS = 1 The timer is stopped
d3–d0	TDDR	0000	Timer divide-down register bits. These bits specify the timer divide-down ratio (period) for the on-chip timer. When the PSC bits are decremented past 0, the PSC is loaded with the contents of the TDDR

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The initialisation of timer, serial port and the delay routine are given in separate files in order to improve the readability of the program. They are inserted in Program6_13.asm by the include statements.

6.4.3.8 Interfacing the DSP and AIC

The interfacing diagram of the TMS320C50 and TLC320C40 is given in Fig. 6.14. The timer clock out signal is fed as master clock to AIC and the shift clock signal generated by the AIC is applied to clock receive and transmit pins of DSP. The receive and transmit frame synchronisation signals from DSP and AIC are interconnected. Similarly the DX and DR pins are interconnected.

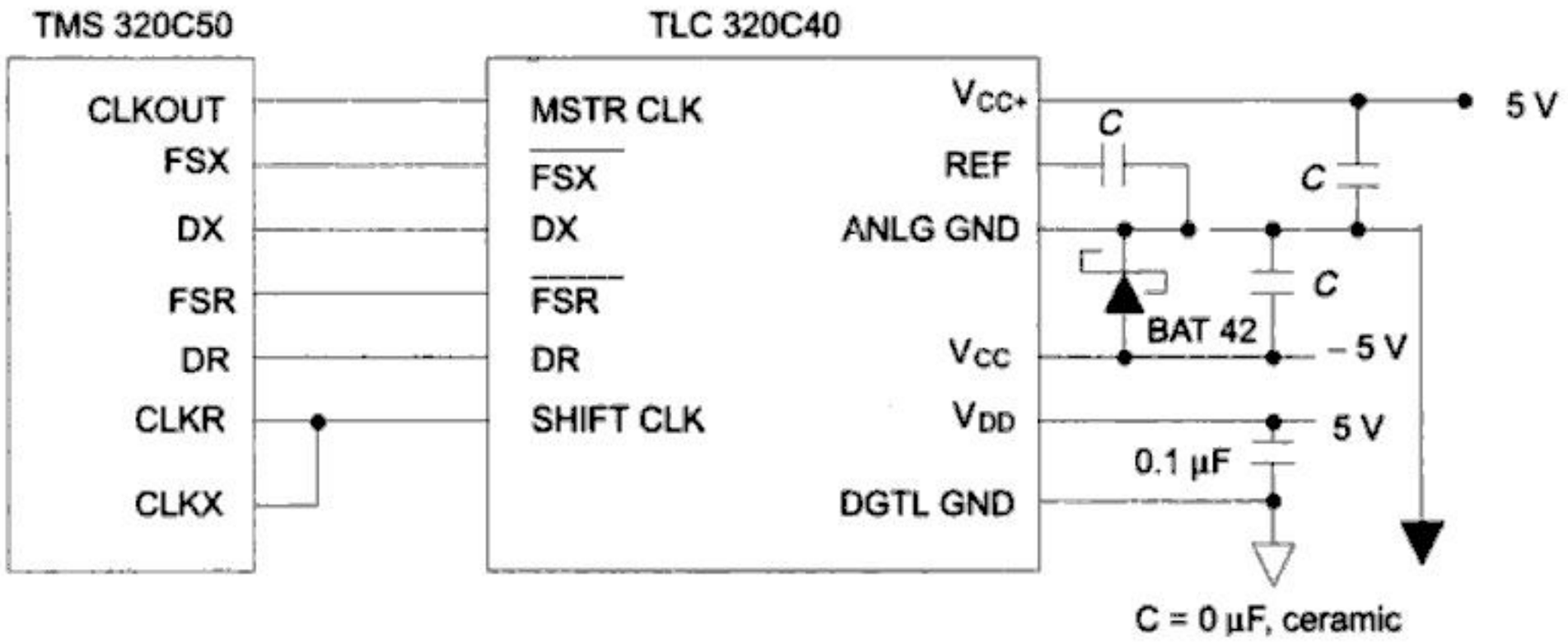


Fig. 6.14 *AIC interface to TMS320C50*

Waveform Generation using AIC Programs for generating various **waveforms** such as square, ramp, triangular and trapezoidal are given in this section.

6.4.3.9 Square Wave

For square wave generation the positive and negative amplitudes and T_{on} and T_{off} time periods are to be fixed. For this the following symbol declaration is used. The values are set for these symbols using .set assembler directive. In the source program it is enough to mention only these symbols wherever required.

Amp+ve: positive amplitude — T_{on}: on period of the wave
Amp–ve: negative amplitude — T_{off}: off period of the wave

Table 6.11 Parameters for generating different square waves

Amp -ve	*Amp +ve*	*Ton*	*Toff*	*Type of square wave generated*
9fffh	7fffh	0f00h	0f00h	Sqr wave with 50% duty cycle & 0 dc component
9fffh	7fffh	7f00h	5f00h	Sqr wave with duty cycle >50% & 0 dc component
9fffh	7fffh	3f00h	7f00h	Sqr wave with duty cycle <50% & 0 dc component
0000h	7ffeh	0f00h	0f00h	+ve clamped sqr wave with 50% duty cycle
9ffeh	0000h	0f00h	0f00h	-ve clamped sqr wave with 50% duty cycle

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
          splk  #0h,cwsr
          clrc intm
          call  aicint
loop1     lamm drr                  ;content of drr register loaded into
                                    ;accumulator
          idle
loop      sfr                       ;right shift the content of accumulator
          exar                      ;exchange content of ACCU with ACCB
          Bcnd  loop3,nc            ;if there is no carry branch to loop3,
                                    ;else execute next instruction
          lar  ar0,#1000h           ;load the starting address of the look up
                                    ;table 1 (1000h) in AR0
          lar ar1,#53               ;the no. of sample values to be
                                    ;transmitted (54) is loaded in AR1
          mar *,ar0
loop2     lacc  *+,0,ar1            ;loop2 transmits 1 kHz sine wave
                                    ;samples to output
          and #0fffch
          samm dxr
          idle
          Banz  loop2,ar0
          lacb                      ;load the ACCB content back to ACCU
          Bcnd  loop1,eq            ;branch to loop1 if ACCU content is zero,
                                    ;to get new sample from drr
          b loop                    ;branch to loop to know the next bit
                                    ;information of ACCU
loop3     lar  ar2,#1200h           ;load the starting address of the look ;up
                                    table 2 (1200h) in AR2
          lar ar3,#26               ;the no. of sample values to be
                                    ;transmitted (27) is loaded in AR3
          mar *,ar2
loop4     lacc  *+,0,ar3            ;loop4 transmits 2 kHz sine wave sample
                                    ;to output
          and #0fffch
          samm dxr
          idle
          Banz  loop4,ar2
          lacb
          Bcnd  loop1,eq
          b loop
aicint:   .include  "aic_int.asm"   ;program for AIC initalisation inserted
                                    ;here
          .end
```

Program6_20.asm FSK generation program

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
                .WORD   0,0,0,0,0,0,0,0,0,0
```

pad.dat

6.4.3.18 Study of Periodic Frequency Response of the Digital Filters

In Chapter 1, it was mentioned that the digital filters have periodic frequency response with period equal to the sampling frequency. In the LP filter considered in Section 6.4.3.17, assume that the continuous time switched-capacitor BP filter is not included (aic_ctr = 30h) at the input to the AIC. In this case if the input to the AIC is greater than fs/2 (i.e. 4.5 kHz) aliasing would occur. Since the LP filter designed has a cutoff frequency of 1 kHz, only those aliased components which lie within ±1 kHz from nf_s would be passed by the LP filter. Hence if the frequency of input signal to the filter is increased, it will have a periodic response with period equal to fs (9 kHz in the above example). Hence if an input signal of frequency 108 kHz is fed to the digital LP filter discussed above, the output is not zero, it is as strong as the input itself. This is because of aliasing. The periodicity property of the digital filter may also be used to determine the sampling rate of the AIC for different values of Ta and Tb. With the filter coefficients chosen in Section 6.4.3.18, vary the values of Ta and Tb. The LP filter cutoff frequency will not be 1000 Hz as the sampling rate is changed. The output will be zero for frequency > fX. As the input frequency is increased further, the output would start rising again at fx1. As the frequency is increased further, the output amplitude rises and reaches the full amplitude. With further increase in frequency, the output would become zero again at $fx2$. The sampling frequency f_s is then given by ($fx1$+$fx2$)/2. This is because

$$f_{x1} = f_s - f_x \quad \text{and} \quad f_{x2} = f_s + f_x$$

If the switched-capacitor BP filter is included at the input to the AIC, aliasing would not occur. A LP filter in this case really behaves like a LP filter. In this case for all frequencies greater than f_c, the cutoff frequency of the LP filter, the output would be zero.

Appendix 6.1

ASSEMBLER DIRECTIVES

Mnemonic and syntax	Description
(a) Directives that define sections	
.data	Assemble into data memory
.ds [address]	Assemble into data memory (initialise data address)
.entry [address]	Initialise the starting address of the program counter when loading a file

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
.word   -551        18    -0.0168
.word   -584        17    -0.0178
.word   -383        16    -0.0117
.word   0           15    -0.0000
.word   438         14    0.0134
.word   763         13    0.0233
.word   827         12    0.0252
.word   557         11    0.0170
.word   0           10    0.0000
.word   -681        9     -0.0208
.word   -1240       8     -0.0378
.word   -1417       7     -0.0432
.word   -1022       6     -0.0312
.word   0           5     -0.0000
.word   1533        4     0.0468
.word   3307        3     0.1009
.word   4960        2     0.1514
.word   6131        1     0.1871
.word   6554        0     0.2000
.word   6131        1     0.1871
.word   4960        2     0.1514
.word   3307        3     0.1009
.word   1533        4     0.0468
.word   0           5     -0.0000
.word   -1022       6     -0.0312
.word   -1417       7     -0.0432
.word   -1240       8     -0.0378
.word   -681        9     -0.0208
.word   0           10    0.0000
.word   557         11    0.0170
.word   827         12    0.0252
.word   763         13    0.0233
.word   438         14    0.0134
.word   0           15    -0.0000
.word   -383        16    -0.0117
.word   -584        17    -0.0178
.word   -551        18    -0.0168
.word   -323        19    -0.0098
.word   0           20    0.0000
.word   292         21    0.0089
.word   451         22    0.0138
.word   431         23    0.0132
.word   255         24    0.0078
.word   0           25    -0.0000
.word   -236        26    -0.0072
```

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(e) RSR

6.47 The registers of on-chip serial port of C5X which shifts the data serially in or out are ———.

(a) SPC (b) DXR (c) DRR (d) XSR
(e) RSR

6.48 Writing the data into — of serial port of C5X generates the XINT.

(a) SPC (b) DXR (c) DRR (d) XSR (e) RSR

6.49 Writing the data into — of serial port of C5X generates the RINT.

(a) SPC (b) DXR (c) DRR (d) XSR
(e) RSR

6.50 The signals which initiate the serial shifting of the data in the on-chip serial port at the beginning of every frame in burst mode and for the first frame in the synchronous transfer mode for the shift registers used at the receive and transmit side respectively are —.

(a) CLKX (b) DX (c) FSX (d) CLKR
(e) RX (f) FSR

6.51 In the AIC, the registers which are used to obtain the switched capacitor frequency of 288 kHz are

(a) TX counter A, TX counter B (b) TX counter A, RX counter A
(c) RX counter A, RX counter B (d) TX counter B, RX counter A

6.52 In the AIC, the registers which are used to obtain the required sampling frequency are ——.

(a) TX counter A, TX counter B (b) TX counter A, RX counter A
(c) RX counter A, RX counter B (d) TX counter B, RX counter B

6.53 To program the AIC, the secondary communication is initiated by choosing the d0, d1 bits of the data transmitted through the DX pin in the primary communication mode as —, ——.

(a) 0, 0 (b) 1, 0 (c) 0, 1 (d) 1,1

6.54 256 samples of a sine wave of frequency 1000 Hz are stored in data memory organised as a circular buffer of size 256. The data is read one after another from this buffer and transmitted through the DX pin infinitely. The frequency at the output of the AIC is — Hz.

(a) 1000 (b) 500 (c) 2000 (d) 4000

6.55 256 samples of a sine wave of frequency 1000 Hz are stored in data memory organised as a circular buffer of size 256. The data is read one after another from this buffer and the alternate data is transmitted through the DX pin infinitely. The frequency at the output of the AIC is — Hz.

(a) 1000 (b) 500 (c) 2000 (d) 4000

6.56 256 samples of a sine wave of frequency 1000 Hz are stored in data memory organised as a circular buffer of size 256. The data is read one after another from this buffer and each data is transmitted twice through the DX pin infinitely. The frequency at the output of the AIC is — Hz.

(a) 1000 (b) 500 (c) 2000 (d) 4000

6.57 Sampling frequency of the AIC is programmed to be 8 kHz. A LP filter with cutoff frequency of 4 kHz is implemented using the kit. The antialiasing filter at the input section of AIC is not enabled. If a signal of frequency 9 kHz is fed to the AIC, the output of the LP filter has a frequency of —— kHz.

(a) 9 (b) 1 (c) 5 (d) 7

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The various CPU components of 'C3X processors are shown in Fig. 7.2.

7.4.1 CPU Internal Buses

The CPU unit consist of four buses, CPU1, CPU2, REG1 and REG2. The data bus DDATA carries data to the CPU over the CPU1 and CPU2 buses. The CPU1 and CPU2 buses can carry two data memory operands to the multiplier, ALU and the register file for every machine cycle. Internal to the CPU are the two buses REG1 and REG2, these buses can carry two data values from the register file to multiplier and ALU for every machine cycle. The CPU internal buses are shown in the Fig. 7.2.

7.4.2 Floating-Point/Integer Multiplier

The multiplier performs multiplications on 24-bit integer and 32-bit floating-point values in a single cycle. The 'C3X implementation of floating-point arithmetic allows for floating-point or fixed-point operations at speeds upto 33 ns per instruction cycle. To get further higher speed, parallel instructions can be used. The parallel instructions will perform a multiply and an ALU operation in a single cycle. When the multiplier performs floating-point multiplication, the inputs are 32-bit floating-point numbers and the result is a 40-bit point floating number, whereas in the case of integer multiplication, the input data is 24 bits and yields a 32-bit result.

7.4.3 Arithmetic Logic Unit (ALU) and Barrel Shifter

The ALU performs single-cycle arithmetic and logical operations. It performs operations on 32-bit integer, 32-bit logical and 40-bit floating-point data. It also performs integer and floating-point conversions in a single cycle. The results of the ALU are always maintained at 32-bit integer or 40-bit floating-point formats. The barrel shifter is used to shift the operands upto 32 bits left or right in a single cycle.

7.4.4 Auxiliary Register Arithmetic Unit (ARAU)

The 'C3X family processors consists of two ARAUs (ARAU0 and ARAU1). These two units can generate two addresses in a single cycle. The ARAUs operate in parallel with the multiplier and ALU and they are used for indirect addressing mode. The ARAUs support address displacement and the displacement can be indicated in the assembly code using the displacement field or the two index register (IR0 and IR1) contents. They also support circular and bit-reversed addressing modes.

7.5 CPU REGISTER FILE

The 'C3X processors consists of 28 registers in a multiport register. These registers are tightly coupled to the CPU. The list of the registers present in the CPU register file are given below.

(a) Extended-precision registers (R7–R0)
(b) Auxiliary registers (AR7–AR0)
(c) Data page pointer (DP)
(d) Index registers (IR1 and IR0)
(e) Block size register (BK)
(f) System stack-pointer (SP)
(g) Status register (ST)
(i) CPU/DMA interrupt-enable' register (IE)

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

operations such as load and store, as well as arithmetic and logical functions, these flag bits are set or cleared. The results could be zero, negative, carry, overflow, etc.

31–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	PRGW Sta.	INT Con.	GIE	CCC	CCE	CCF	X	RM	OVM	LUF	LV	UF	N	Z	V	C
	R	R/W	R/W	R/W	R/W	R/W		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Fig. 7.7(b) *The status register format ('C32)*

Note: 1 *(x – reserved bit, read as 0 and 2) R – read , W – write*

It is possible to read the information in the status register bits, it can be written as well. When the status register is loaded, the contents of the source operand replace the current contents of ST bit-for-bit, regardless of the state of any bits in the source operand. The source operand content can be written as such in the status register. This allows the status register to be saved and restored. At system reset, logic 0 is written to this register. Table 7.2 defines the status register bit names and their functions.

Table 7.2 Status register bits summary

Bit name	*Function*
C	Carry condition flag
V	Overflow condition flag
Z	Zero condition flag
N	Negative condition flag
UF	Floating-point underflow condition flag
LV	Latched overflow condition flag
LUF	Latched floating-point underflow condition flag
OVM	Overflow flag. The overflow mode flag affects only integer operations If OVM = 0, the overflow mode is turned off If OVM = 1, integer results overflowing in the positive direction are set to the most positive, 2s-complement number (7FFFFFFFh), and integer results overflowing in the negative direction are set to the most negative 32-bit, 2s-complement number (8000 0000h)
RM	Repeat mode flag If RM = 1, the PC is modified in either the repeat-block or repeat-single mode
CF	Cache freeze. Enables or disables the instruction cache. Set CF = 1 to freeze the cache (cache is not updated). When CF = 0, the cache is automatically updated by instruction fetches from external memory
CE	Cache enable. CE enables or disables the instruction cache. Set CE = 1 to enable the cache, Set CE = 0 to disable the cache
CC	Cache clear. CC = 1 invalidates all entries in the cache
GIE	Global interrupt-enable. If GIE = 1, the CPU responds to an enabled interrupt. If GIE = 0, the CPU does not respond to an enabled interrupt
INT	Interrupt configuration('C32 only)

Contd.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

7.6.1 Memory Internal Buses

The high performance of 'C3X processors is due to the internal buses and parallelism. The memory buses are program bus, data bus and DMA buses. The separate program bus consists of program address (PADDR) bus and program data (PDATA) bus. There are two data address buses (DADDR1 and DADDR2) and one data (DDATA) bus. The DMA bus contains DMA address (DMAADDR) bus and a DMA data (DMADATA) bus. These buses allow parallel instruction code fetch from program memory, two data accesses and DMA access. These buses connect on-chip memory, off-chip memory and on-chip peripherals. All address buses are 24 bits and data buses are 32bits in size.

The PC is connected to PADDR bus and the IR is connected to PDATA bus. These buses can fetch a single instruction word for every machine cycle. The two data address buses are connected to the AR unit, which generates the data memory addresses and the data bus is connected to the CPU over CPU1 and CPU2 buses, which carry the data to the CPU.

The DMA controller is connected with a 24-bit address bus (DMAADDR) and a 32-bit data bus (DMADATA). These buses allow the DMA to perform memory access in parallel with the memory accesses occurring from the data and program buses.

7.6.2 Memory Maps

The memory map of 'C3X devices depend on the logic level applied to the external pin MC/$\overline{\text{MP}}$ or MCBL/$\overline{\text{MP}}$. For logic 0 the processor runs in microprocessor mode, whereas for logic 1 it runs in microcomputer mode. The number of address lines for the memory is 24 bits and start and end address of the memory are 00000h-FFFFFFh.

Microprocessor mode In microprocessor mode, the 4K on-chip ROM is not mapped into the 'C3X memory map. The memory maps of 'C30, 'C31 and 'C32 devices are almost similar. The memory map for microprocessor mode of 'C30, 'C31 and 'C32 can be found in C3X User's guide [1996].

The external memory port with $\overline{\text{STRB}}$ signal active accesses the address locations 00000h-7FFFFFh (8.192M words). In this space, locations 0h-03Fh (192 words) consists of interrupt, trap vectors and reserved locations in 'C30 and 'C31, whereas in 'C32 location 0h alone is containing the reset vector. The locations 800000h-807FFFh (32K words) are reserved in 'C31 and 'C32. In 'C30 the total 32K size is partitioned into four 8K segments. The first and third 8K segments are mapped to the expansion bus and can be accessed when $\overline{\text{MSTRB}}$ & $\overline{\text{IOSTRB}}$ signals are active respectively. The second and fourth 8K segments are reserved locations. In all the three processors, locations 808000h-8097FFh (6K words) are memory-mapped for the peripheral bus registers.

The 'C30 and 'C31 processors have two 1K word RAM blocks starting from 809800h to 809FFFh, whereas in 'C32 it has two 256-word blocks starting from 87FE00h to 87FFFFh. The locations 80A000h-FFFFFFh (7.96M words) of 'C30 and 'C31 are mapped to external space and accessed when $\overline{\text{STRB}}$ signal is active. In 'C32 device, locations 809800h-80FFFFh (26K words) and 830000h-87FDFFh (319.5K words) are reserved. The locations 810000h-82FFFFh (128K words), 880000h-8FFFFFh (512K words) and 900000h-FFFFFFh (7.168M words) are mapped to external space and can be accessed when $\overline{\text{IOSTRB}}$, $\overline{\text{STRBO}}$ and STRB1 signals are active.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

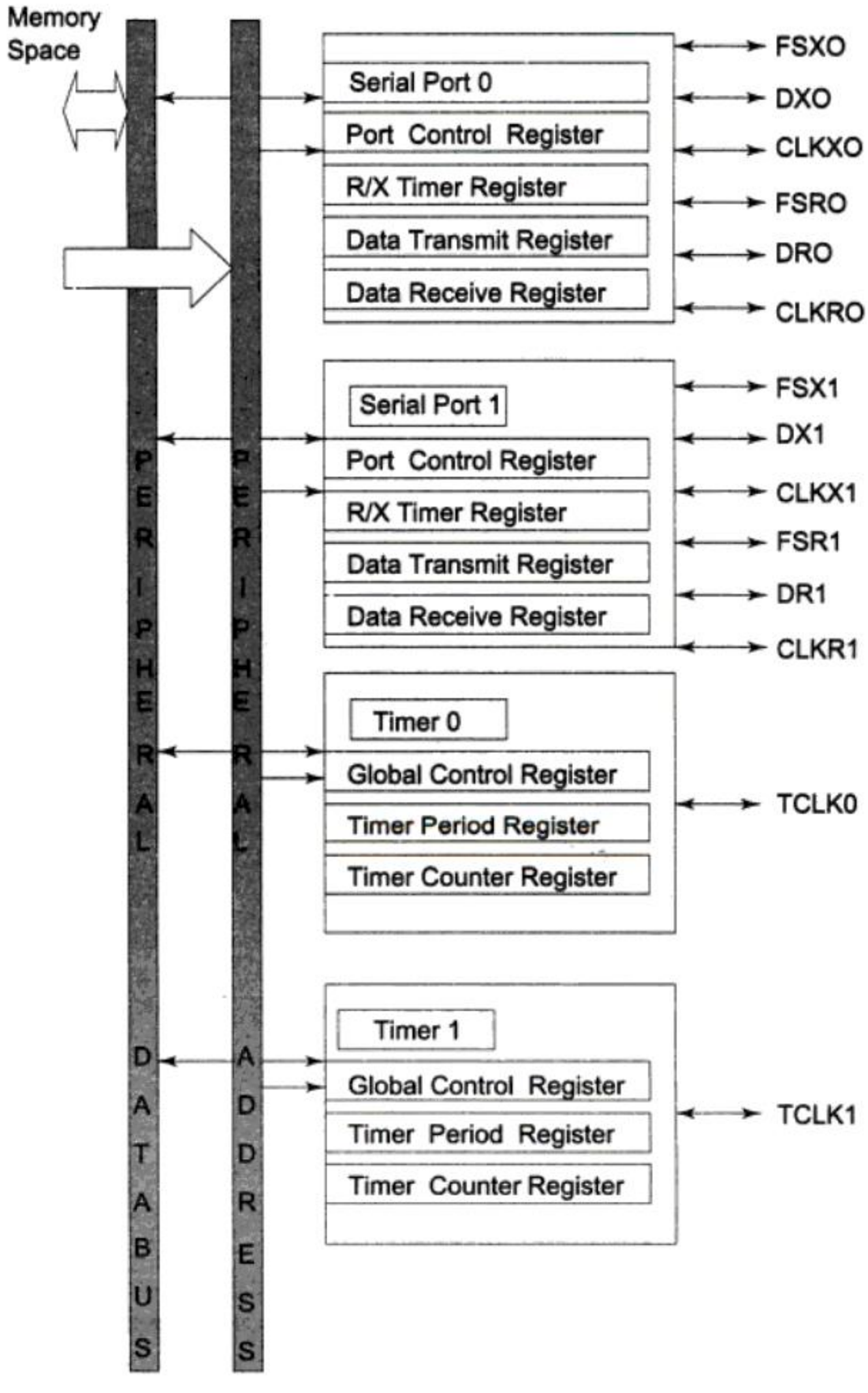


Fig. 7.12 *Block diagram of 'C3X peripherals*

the content of the period register. When the values are equal, the counter is zeroed. This causes the internal interrupt. The pulse generator present generates two types of external clock signals, either pulse or clock.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Self-test Questions

7.16 The 'C3X family processors are _____ bit floating-point processors.
(a) 8 (b) 16 (c) 24 (d) 32

7.17 The 'C3X family processor suitable for low power application is _____.
(a) 'C30 (b) 'C31 (c) `LC31 (d) 'C32

7.18 The on-chip RAM available in the 'C3X family processors is _____.
(a) 2K (b) 4K (c) 8K (d) 64K

7.19 The 'C3X family processors are different in _____.
(a) architecture (b) memory size (c) speed of operation

7.20 The 'C3X family processors have _____.
(a) only SARAM (b) only DARAM (c) both SARAM and DARAM

7.21 The no. of address buses present in the 'C3X family processors are _____.
(a) 5 (b) 6 (c) 3 (d) 2

7.22 The no. of data buses present in the 'C3X family processors are _____.
(a) 8 (b) 7 (c) 9 (d) 4

7.23 The multiplier in the 'C3X family processors can perform _____ multiplications.
(a) 24-bit integer
(b) 32-bit floating point
(c) both 24-bit integer and 32-bit floating-point

7.24 The size of the 'C3X processor ALU is _____.
(a) 24 bits (b) 16 bits (c) 32 bits (d) 40 bits

7.25 The no. of ARAUs in the 'C3X processors are _____.
(a) 2 (b) 3 (c) 1 (d) 4

7.26 The no. of index registers in the 'C3X processors are _____.
(a) 3 (b) 1 (c) 2 (d) 4

7.27 The 'C3X processor register file contains _____ registers.
(a) 12 (b) 24 (c) 28 (d) 34

7.28 The no. of data pages and location in each page of data memory of the 'C3X processor are _____.
(a) 512, 128K (b) 256, 64K (c) 256, 32K (d) 512, 8K

7.29 The no. of bits of the DP used for direct addressing mode are _____.
(a) LSB 9 bits (b) LSB 8 bits (c) LSB 12 bits (d) all bits of DP

7.30 The no. of ARs used for indirect addressing mode of the 'C3X processor are _____.
(a) 6 (b) 8 (c) 4 (d) 2

7.31 The no. of flag bits present in the status (ST) register of the 'C3X processor are _____.
(a) 4 (b) 6 (c) 9 (d) 10

7.32 Which bit present in the ST register of the 'C3X processor will globally disable and enable the interrupts?
(a) GIE (b) C (c) Z (d) OV

7.33 The purpose of I/O flag register is to
(a) to control the function of XF0 and XF1 pins
(b) to control the interrupt operation
(c) to control the read and write operations

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Table 8.1 Most, least positive and negative numbers in floating-point format

	Short floating-point format	*Single precision floating-point format*	*Extended precision floating-point format*
Most, positive	$(2\text{-}2^{-11})\times 2^{7}=$ 2.5594×10^{2}	$(2\text{-}2^{-23})\times 2^{127}=$ 3.4028234×10^{38}	$(2\text{-}2^{-23})\times 2^{127}=$ 3.4028234×10^{38}
Least, positive	$(1\times 2^{-7})=$ 7.18125×10^{-3}	$(1\times 2^{-127})=$ 5.8774717×10^{-39}	$(1\times 2^{-127})=$ $5.8774717541\times 10^{-39}$
Least, negative	$(-1-2^{-11})\times 2^{-7}=$ -7.8163×10^{-3}	$(-1-2^{-23})\times 2^{-127}=$ $-5.8774724\times 10^{-39}$	$(-1-2^{-23})\times 2^{-127}=$ $-5.8774717569\times 10^{-39}$
Most, negative	$(-2\times 2^{7})=$ -2.5600×10^{2}	$(-2\times 2^{127})=$ -3.4028236×10^{38}	$(-2\times 2^{127})=$ $-3.4028236691\times 10^{38}$

Single Precision Floating-Point Format In the single precision floating-point format, an 8-bit exponent field and a 24-bit mantissa with an implied most significant nonsign bit is used to represent the operands.

Extended Precision Floating-Point Format In the extended precision floating-point format the operands are represented with an 8-bit exponent field and a 32-bit mantissa with an implied significant nonsign bit.

The maximum positive, negative, minimum positive and negative numbers that can be represented using the three floating-point formats are given in Table 8.1.

8.2 ADDRESSING MODES

The TMS320C3X processor supports the following six addressing modes:

(1) Register addressing mode, (2) Direct addressing mode, (3) Indirect addressing mode, (4) Short-immediate addressing mode, (5) Long-immediate addressing mode and (6) PC-relative addressing mode. In this section, the various addressing modes are explained with examples.

8.2.1 Register Addressing

The TMS320C3X processor register file contains eight extended precision registers (R0–R7). These CPU registers contain the operand. The syntax of this addressing mode is

mnemonic src, dst

The *mnemonic* can be any assembly instruction code that support register addressing mode, *src* is the source register and *dst* is the destination register. The registers R0–R7 can be used both for source and destination registers.

Example 8.1 ADDI R3,R5—This instruction adds the two hexadecimal integer operands present in the registers R3 and R5. The result is stored in the register R5. The content of register R3 is unchanged.

Before execution		After execution	
R3	11223344	R3	11223344
R5	22334455	R5	33557799

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

R3	23459872	R3	23459872

Content of the location 00803013h

11223344	23459872

Example 8.6 STF R4,+—AR5(5)—The floating point operand present in the register R4 is stored in data memory. The data memory address is the subtraction of displacement (disp=5) from the content of AR5. After execution, the new content of AR5 is the subtraction of displacement from the old content AR5. Take the content of AR5 as 00602020h, then the memory address is 0060201Bh.

Before execution — After execution

AR5	00602020	AR5	0060201 B
R4	2.34353674e + 01	R4	2.34353674e + 01

Content of the location 0060201Bh

1.23425354e + 01	2.34353674e + 01

Example 8.7 SUBB *AR2++(4),R6—The integer operand present in the data memory location, pointed by AR2 is subtracted from the content of register R6 and the result is stored in register R6, The data memory address before execution is the content of AR2, the new content of AR2 is the sum of displacement and the old content AR2. Take the content of AR2 as 00701010h.

Before execution — After execution

AR2	00701010	AR2	00701014
R6	23459872	R6	23459872

Content of the location 00701010h

11223344	1223652E

Example 8.8 ADDF *AR6++(IR1),R0—The floating point operand present in the data memory address pointed by AR6 is added with the content of the register R0 and the result is stored in R0. The new content in AR6 is the sum of the old content in AR6 and the content of IR1.

Before execution — After execution

AR6	00800000	AR6	00801000
IR1	00001000		00001000

Content of the location 0080000h

	1.23425354e + 01		1.23425354e + 01
R0	2.34353674e + 02	R0	2.46696209e + 02

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

8.3 GROUPS OF ADDRESSING MODES

There are six types of addressing modes. Since some addressing modes are not appropriate for some instructions, the types of addressing modes are used in four groups as follows
1. General addressing modes (G); 2. Three operand addressing modes (T); 3. Parallel addressing modes (P); 4. Conditional branch addressing modes. This section explains about the groups of addressing modes.

1. General Addressing Modes The general addressing mode includes register addressing, direct addressing, indirect addressing and immediate addressing modes. The general addressing mode uses the general-purpose instructions. The bits 31-29 set 000 indicates the general addressing mode. The bits 22 and 21 of the instruction word specify the general addressing mode.

0 0 - register addressing
0 1 - direct addressing
1 0 - indirect addressing
1 1 - immediate addressing

2. Three-operand Addressing Modes In three-operand addressing modes two source operands and one destination operand are used. The instructions which use three operand addressing mode have the form ADDI3, LSH3, CMPF3, XOR3, etc. The three-operand addressing modes include register and indirect addressing modes. The bits 31-29 set to 001 indicates the three-operand addressing mode. The bits 22 and 21 of the instruction word specify the type of addressing mode to be used for two source operands

	SRC1	SRC2
0 0 -	register	register
0 1 -	indirect	register
1 0 -	register	indirect
1 1 -	indirect	indirect

3. Parallel Addressing Modes Some of the 'C3X instructions can occur in pair and those instructions will be executed in parallel. The parallel instructions are indicated with two vertical bars (||) and this includes register and indirect addressing modes. For parallel addressing mode four operands are needed. The instruction code format of the parallel addressing mode is shown in Fig. 8.5 and its field description is given in Table 8.5

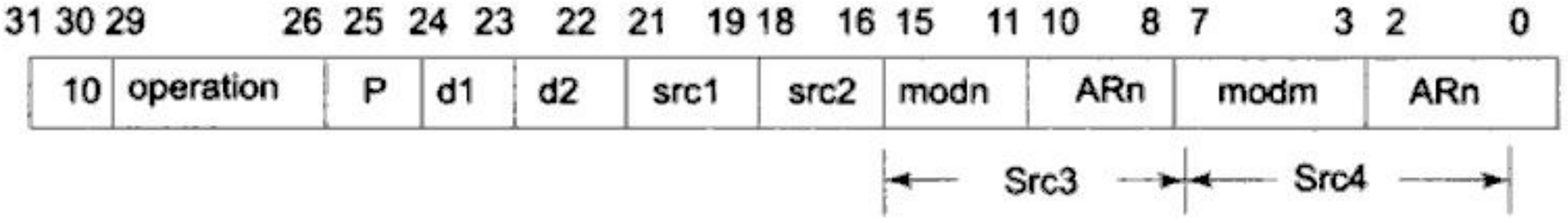


Fig. 8.5 *The instruction code format of parallel addressing mode*

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

8.4 ASSEMBLY LANGUAGE INSTRUCTIONS

The TMS320C3X assembly instructions set supports numeric-intensive, signal processing and general-purpose applications. There are 113 instructions organised in six groups. All instructions sets are one word and most of them require one cycle to execute. In the instructions set some instructions support floating-point and some use fixed-point operands. The six functional groups of addressing modes are as follows.

- Load and store
- Two–operand arithmetic/logical
- Three–operand arithmetic/logical
- Program control
- Interlocked operations
- Parallel operations

In this section the groups of addressing modes are discussed and some important instructions sets are explained with examples. The general syntax of assembly instructions is as follows.

1. Three operand instructions
   mnemonic src2,src1,dst

The source operands *src2* and *src1* can be accessed by the register and indirect addressing modes, whereas the destination operand *dst* should be accessed only by register addressing mode.

2. Other instructions
   mnemonic src,dst

The source operand *src* can be accessed by general addressing mode (G) and the destination operand *dst* should be accessed only by register addressing mode. But for store instructions the source and destination operand addressing modes are reverse.

Load and Store Instructions The 'C3X processors support 13 load and store instructions. Using these instructions a word can be loaded from memory into a register, stored from register into memory. Certain instructions are used to manipulate data on the system stack. Load instructions can also be conditional instructions.

Two-Operand Instructions There are 35 two-operand arithmetic and logical instructions. Out of this two-operands one is source and another is destination. The source operand can be a memory word, a register, or a part of the instruction word, where as the destination operand is always a register.

Three-Operand Instructions Some arithmetic and logical instructions have three operands. There are 17 such three operand instructions, which allow the processor to read two source operands from memory or from the CPU register file in a single cycle and store the results in destination, which is always a register.

Program-Control Instructions The program-control instruction group consists of repeat instructions, both standard and delayed branch, call and return instructions. There are 17 such instructions, which affect the program flow. Several program-control instructions support conditional operations.

Interlocked-Operation Instructions There are 5 instructions which support interlocked operations. These instructions are used for multiprocessor communication through the external signals (XF0 and XF1) to allow for powerful synchronisation mechanisms. The source address is accessed only through

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

about the properties of the result of arithmetic and logical instructions. The flag bits and its details are explained in Table 8.11. The conditional codes are based on the status of these flag bits. The list of conditions and its description are given in Table 8.12. The specified condition in the conditional field is true, the respective operation is performed, if it is false the next instruction is executed. The load, branch, call, return and trap instructions can be conditional instructions.

Table 8.11 The status register flags and its descriptions

Flag	*Description*
1. Latched Floating-point underflow conditional flag (LUF)	LUF is set whenever UF is set. LUF can be cleared by resetting the processor or by modifying it in the status register.
2. Latched overflow condition flag (LV)	LV is set whenever V is set. LV can be cleared by resetting the processor or by modifying it in the status register.
3. Floating-point underflow condition flag (UF)	Whenever the exponent of the result is less than or equal to –128, a floating point under flow occurs. UF is set for under flow, if not it is cleared. The output value is set to zero for under flow.
4. Negative condition flag (N)	For logical, integer and floating point operations N is set if the result is negative, and cleared otherwise. Zero is positive.
5. Zero condition flag (Z)	For logical integer and floating point operations, Z is set if the output is 0 and cleared otherwise.
6. Overflow condition flag(V)	For integers if the maximum positive (2^{32-1}) and negative (-2^{32}) numbers are obtained in the result V is set, otherwise it is cleared. For floating point operations, if the exponent of the result is greater than 127, V is set; otherwise it is cleared.
7. Carry flag (C)	When integer addition results a carry or in an integer subtraction, a borrow occurs to the MSB of the output, the C bit is set otherwise it is cleared.

Table 8.12 Flag conditions and descriptions

Condition	*Description*
Unconditional Compares	
U	Unconditional
Unsigned Compares	
LO	Lower than
LS	Lower than or same as
HI	Higher than
HS	Higher than or same as
EQ	Equal to
NE	Not equal to
Signed Compares	
LT	Less than
LE	Less than or equal to
GT	Greater than
GE	Greater than or equal to
EQ	Equal to
NE	Not equal to

Contd.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

The repeat mode flag bit (RM), which is present in the status register (ST), specifies whether the processor is running in the repeat mode. The S bit is internal to the processor and cannot be programmed but this bit is necessary to describe the single instruction repeat operation. If the RM flag bit is set, it specifies the repetitions of a black of code. If both RM bit and S bit are set, it indicates single instruction repeat operation. The maximum number of repetitions that is possible in 'C3X processor is 8000 0001h times.

8.4.9 Low-power Control Instructions

The low-power control instruction group consists of four instructions that can be used for low-power modes. The low-power instructions and its description are given in Table 8.15.

Table 8.15

Instruction	*Description*
IDLE	Idle until interrupt
IDLE2	Low-power idle
LOPOWER	Divide the clock by 16
MAXSPEED	Restore clock to regular speed

The IDLE instruction stops the CPU operations until an interrupt is received. The global interrupt bit is set. The IDLE2 instruction serves the same function as IDLE, but it removes the functional clock input from the internal devices. When LOPOWER instruction is used, the processor continues the execution of instructions at the reduced clock rate. The input clock rate is divided by 16 times. For MAXSPEED, it exits the LOPOWER power down mode and starts the execution with full speed.

Appendix 8

INSTRUCTION SET SUMMARY—FUNCTIONAL GROUPS

A8.1 Load, Store, Push and Pop Instructions

Mnemonic	Description
LDE	Load floating-point exponent
LDF	Load floating-point value
LDF cond	Load floating-point value conditionally
LDI	Load integer
LDI cond	Load integer conditionally
LDM	Load floating-point mantissa
LDP	Load data page pointer
STF	Store floating-point value
STI	Store integer
POP	Pop integer from stack
POPF	Pop floating-point value from stack

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

(b) Parallel Load Instructions

Mnemonic	Description
LDF ‖ LDF	Load floating-point value
LDI ‖ LDI	Load integer

(c) Parallel Multiply and Add/Subtract Instructions

Mnemonic	Description
MPYF3 ‖ ADDF3	Multiply and add floating-point value
MPYF3 ‖ SUBF3	Multiply and subtract floating-point value
MPYI3 ‖ ADDI3	Multiply and add integer
MPYI3 ‖ SUBI3	Multiply and subtract integer

INSTRUCTION SET SUMMARY—ALPHABETICAL ORDER

Mnemonic	Description
ABSF	Absolute value of a floating-point number
ABSI	Absolute value of an integer
ADDC	Add integers with carry
ADDC3	Add integers with carry
ADDF	Add floating-point values
ADDF3	Add floating-point values
ADDI	Add integers
ADDI3	Add integers
AND	Bitwise-logical AND
AND3	Bitwise-logical AND (3-operand)
ANDN	Bitwise-logical AND with complement
ANDN3	Bitwise-logical ANDN (3-operand)
ASH	Arithmetic shift
ASH3	Arithmetic shift (3-operand)
B cond	Branch conditionally (standard)
B condD	Branch conditionally (delayed)
BR	Branch unconditionally (standard)
BRD	Branch unconditionally (delayed)
CALL	Call subroutine
CALL cond	Call subroutine conditionally
CMPF	Compare floating-point values
CMPF3	Compare floating-point values (3-operand)
CMPI	Compare integers
CMPI3	Compare integers (3-operand)
DB cond	Decrement and branch conditionally (standard)
DB condD	Decrement and branch conditionally (delayed)
FIX	Convert floating-point value to integer

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

9

Application Programs in C3X

In this chapter, some application programs in TMS320C3X processors are given. To test these programs, assembly language programming tools and certain hardware accessories are needed. The programming tools are used to write the 'C3X assembly language programs and converted into machine language codes. The machine language codes are then loaded into the 'C3X DSP using the hardware accessories. It is necessary to supply the real time signals to the DSP for processing. The real time signals, which are analog in nature are to be digitised and after processing the signal digitally in DSP, they have to be converted back to analog signals. So, along with DSP, a minimum number of devices are to be connected for real time signal processing. In this chapter application program development using the TI's TMS320C3X starter kit is discussed.

9.1 TMS320C3X STARTER KIT (DSK)

9.1.1 Overview of TMS320C3X Starter Kit

The 'C3X starter kit (DSK) is a low-cost, simple, high-performance stand-alone application development board. The DSK has on-board, industry standard TMS320C31 floating-point processor. This allows us to verify 'C3X codes with full speed and experiment real time signal processing. The block diagram of DSK is shown in Fig. 9.1. The 50 MHz system clock makes the instruction cycle time of 'C31 as 40 ns (25 MHz) and this allows execution of instructions upto 50 MFLOPS and 25 MIPS. A standard or enhanced parallel port interface is used to connect the DSK to host PC and through this interface, the communication is carried out between 'C31 and PC.

The DSK has TLC320C40, an analog interface circuit (AIC). The AIC consists of variable rate analog-to-digital converter (ADC) and a digital-to-analog converter (DAC) with 14-bit dynamic range 20,000 samples per second. There are two standard RCA plug connectors, for analog input and outputs.

All the signals of 'C3X are routed to expansion connectors. The expansion connectors include four 32-pin headers, an 11-pin jumper block, and a 12-pin XDS510 header. This feature gives freedom to design new daughter boards, to create new software on a host PC and to run the software on the DSK board.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

```
                    ;content R3 is left shifted and it is right
                    ;shifted, if the data content is less than
                    ;zero. When left shifted, the LSBs are zero
                    ;filled and for right shift the MSBs are sign
                    ;extended
OR @9900h, R2       ;the bit  wise logical OR is performed between
                    ;the content of data memory location 9900h of
                    ;data page 80h and the content of R2
SUBI @9900h, R3     ;the integer operand present in location
                    ;9900h of data page 80h is subtracted from the
                    ;register content of R3 and the result is
                    ;stored in R3
MPYI @9910h, R0     ;the integer operand present in the location
                    ;9910h of data page 80h is multiplied with the
                    ;content of R0 and the result is stored in R0
STF  R2,@9902h      ;the floating-point operand present in R2 is
                    ;stored into the data memory location 9902h
                    ;of data page 80h
LDF @9902h, R6      ;the floating-point operand present in the
                    ;data memory location 9902h of data page 80h
                    ;is loaded into R6
ADDF @9902h, R2     ;the floating-point operand present in the
                    ;data memory location 9902h of data page 80h
                    ;is added to the content of register R2 and the
                    ;result is stored in R2
SUBF @9902h, R2     ;the floating-point operand present in the
                    ;data memory location 9902h of data page 80h
                    ;is subtracted from the content of R2 and the
                    ;result is stored in R2
MPYF @9902h, R2     ;the floating-point operand present in the
                    ;data memory location 9902h of data page 80h
                    ;is multiplied with the content of R2 and the
                    ;result is stored in R2

.end
```

Program 9.3 Example program on direct addressing mode in C3X

9.2.3 Register Addressing Mode

The 'C3X processor CPU has eight extended precession registers R0–R7. These registers are used for the various fixed-point and floating-point operations. In this addressing mode the operations are performed with the content of these eight registers. These registers accept fixed-point and floating-point operands. The example given in Program 9.4 illustrates some instructions which use the register-addressing mode. It is to be noted that the register-addressing mode supports three operand instructions. The two different operands in two different registers can be used for the arithmetic and logic operations. The three-operand instructions are valid for both fixed-point and floating-point operands.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Table 9.2 (Contd.)

Bit Name	*Function*
INV	Inverter control bit If an external clock source is used and INV = 1, the external clock is inverted as it goes into the counter. If the output of the pulse generator is routed to TCLK and INV = 1, the output is inverted before it goes to TCLK. If INV = 0, no inversion is performed on the input or output of the timer. The INV bit has no effect, regardless of its value, when TCLK is used in I/O port mode
TSTAT	Timer status bit. This bit indicates the status of the timer. It tracks the output of the uninverted TCLK pin. This flag sets a CPU interrupt on a transition from 0 to 1. A write has no effect

The timer can receive its input and send its output in several different modes, depending upon the setting of CLKSRC, FUNC and I/O. The four timer modes of operation are defined in Table 9.3. The timer output frequency depends on the input frequency and the count value in the period register. The following equations can be used to calculate the output frequency of the timer either in clock mode or in pulse mode.

$$f(\text{pulse mode}) = f(\text{timer input clock})/\text{period register}$$

$$f(\text{clock mode}) = f(\text{timer input clock})/(\ 2\text{X period register})$$

Table 9.3 Timer modes

CLKSRC	*FUNC*	*Timer mode*
1	0	The timer input comes from the internal clock. The internal clock is not affected by the INV bit in the global control register. In this mode, TCLK is connected to the I/O port control, and TCLK can be used as a general-purpose I/O pin
1	1	The timer input comes from the internal clock, and the timer output goes to TCLK. This value can be inverted using INV, and you can read in DATIN the value output on TCLK
0	0	The timer is driven according to the status of the I/O bit If I/O = 0, the timer input comes from TCLK If I/O = 1, TCLK is an output pin
0	1	TCLK drives the timer If INV = 0, all 0-to-1 transitions of TCLK increment the counter If INV = 1, all 1-to-0 transitions of TCLK increment the counter

9.5.4 Timer Initialisation

The on-chip timer can be used to generate clock signals for external devices. By programming the global control register, timer period register and timer counter register, the required frequency can be obtained from the timer output pin of the timer. In C3X starter kit the timer input is 25 MHz and it is from internal clock. The period register is programmed to divide this input clock by a factor of two. The initialisation routine for the timer is given in Program 9.13. First the on-chip timer memory-map register address values and the timer period register count value are set to variables. Then the control

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

10.6.2 Overflow Handling

The ALU saturation logic prevents a result from overflowing by keeping the result at a maximum (or minimum) value. This feature is useful for filter calculations. The logic is enabled when the overflow mode bit (OVM) in status register ST1 is set.

When a result overflows:

If OVM = 0, the accumulators are loaded with the ALU result without modification

If OVM = 1, the accumulators are loaded with either the most positive 32-bit value (00 7FFF FFFFh) or the most negative 32-bit value (0FF 8000 0000h), depending on the direction of the overflow

The overflow flag (OVA/OVB) in status register ST0 is set for the destination accumulator and remains set until one of the following occurs:

A reset is performed

A conditional instruction (such as a branch, a return, a call or an execute) is executed on an overflow condition.

The overflow flag (OVA/OVB) is cleared

The accumulator may also be saturated by using the SAT instruction, regardless of the value of OVM.

10.6.3 The Carry Bit

The ALU has an associated carry (C) bit that is affected by most arithmetic ALU instructions, including rotate and shift operations. The C bit supports efficient computation of extended-precision arithmetic operations. The C bit is not affected by loading the accumulator, performing logical operations or executing other nonarithmetic or control instructions, so it can be used for overflow management. Two conditional operands, C and NC, enable branching, calling, returning and conditionally executing according to the status (set or cleared) of the C bit. Also, the RSBX and SSBX instructions can be used to load the C bit. The C bit is set on a hardware reset.

10.6.4 Dual 16-Bit Mode

For arithmetic operations, the ALU can operate in a special dual 16-bit arithmetic mode that performs two 16-bit operations (for instance, two additions or two subtractions) in one cycle. This mode is selected by setting the C16 field of ST1. This mode is especially useful for the Viterbi add/compare/select operation (see Section 10.9, Compare, Select and Store Unit (CSSU)).

10.7 BARREL SHIFTER

The barrel shifter is used for scaling operations such as prescaling an input data-memory operand or the accumulator value before an ALU operation; performing a logical or arithmetic shift of the accumulator value; normalising the accumulator; postscaling the accumulator before storing the accumulator value into data memory. The SXM bit controls signed/unsigned extension of the data operands; when the bit is set, sign extension is performed. Some instructions, such as LDU, ADDS and SUBS operate with unsigned memory operands and do not perform sign extension, regardless of the SXM value. The shift count determines how many bits to shift. Positive shift values correspond to left

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Review Questions

10.1 What are the functional blocks present in 54X but not in 5X? Explain the function performed by each of them.
10.2 Compare the multiplier units in 54X and 5X.
10.3 Compare the address and data buses of 54X and 5X.
10.4 The data stored in data memory is 16 bits long. But the multiplier requires 17 bit dat(a). Explain how the MSB is generated?
10.5 Explain the two ways in which the accumlator may be loaded with the most positive value or most negative value when overflow occurs.
10.6 Explain how the rounding operation is carried by the adder/multiplier units of 54X.
10.7 What is the use of guard bits in accumlators?
10.8 What is the use of the following registers of 54X?
10.9 (a) T (b) TRN (c) BIC (d) XPC
10.10 Explain the operation of CSSU of 54X and explain its use with an application.
10.11 Explain the operation of the exponent encoder in 54X.

Self-test Questions

10.12 No. of Auxiliary Register ALU(s) in 54X is ———— and number of data buses which can be used for reading data from data memory is ————.
(a) 1, 1 (b) 1, 2 (c) 2, 1 (d) 2, 2
10.13 Which of the following are available in 54X but not in 5X?
(a) SP (b) 16 bit timer (c) XPC (d) 8-bit HPI
10.14 Number of data bus in 54X is —.
(a) 1 (b) 2 (c) 3 (d) 4
10.15 Number of address bus in 54X is —.
(a) 1 (b) 2 (c) 3 (d) 4
10.16 ———— is used to store the result of adder units in the ALU of 54X.
(a) Accumlator A (b) Accumlator B
(c) Either A or B (d) T Register
10.17 —— is used to store the result of multiplier units in the ALU of 54X.
(a) Accumlator A (b) Accumlator B
(c) Either A or B (d) PREG
(e) T register
10.18 In the LD || MAC parallel instruction, the register where the MAC result is stored is —.
(a) A (b) B (c) A or B (d) T register
10.19 Bits 32–16 of ——— can be used as an input to the multiplier in 54X.
(a) A (b) B (c) Either A or B (d) Neither A or B
10.20 Which of the following registers are present in 5X but not in 54X?
(a) PC (b) SP (c) PREG (d) INDX
10.21 The max no. of wait states that software wait-state generator can produce is ———.
(a) 1 (b) 4 (c) 7 (d) 8

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

address; thus, Xmem remains the same. Regardless of the condition, Xmem is always read and updated.

Syntax STRCD Xmem, cond

11.8 REPEAT INSTRUCTIONS OF '54X

Only some of the instructions of 54X are repeatable. The instructions which cannot be repeated are highlighted with † symbol (*Refer to the 54x instruction set summary*) in Table A10.1.

RPT—Repeat Next Instruction

The repeat counter (RC) is loaded with the number of iterations when RPT is executed. The number of iterations (n) is obtained from a 16-bit single data-memory operand or an 8- or 16-bit constant. The instruction following the RPT instruction is repeated $n + 1$ times. We cannot access RC while it decrements.

Syntax

1: RPT Smem
2: RPT #k
3: RPT #lk

RPTB[D]—Block Repeat

The RPTB[D] instruction allows a block of instructions to be repeated the number of times specified by the memory-mapped BRC. BRC must be loaded before the execution of an RPTB instruction. When the RPTB is executed, the Block-Repeat Start Address Register (RSA) is loaded with PC + 2 (PC + 4 if delayed) and the Block-Repeat.End Address Register (REA) is loaded with the program-memory address (pmad).

The RPTB instruction is interruptible. Single-instruction repeat loops (RPT and RPTZ) can be included as part of RPTB blocks. To nest instructions we need to ensure that the BRC, RSA and REA registers are appropriately saved and restored and the block repeat active flag (BRAF) is properly set.

In the RPTBD instruction, which specifies a delayed block repeat, the two 1-word instructions or the one 2-word instruction following the RPTB is fetched and executed before the execution of the RPTBD instruction.

Block repeat can be deactivated by clearing the BRAF bit.

Syntax RPTB[D] pmad

RPTZ—Repeat Next Instruction and Clear Accumulator

The RPTZ instruction clears the destination accumulator and repeats the instruction following RPTZ $n + 1$ times, where n is the value in the repeat counter (RC). The RC value is obtained from the long-immediate constant.

Syntax RPTZ dst, lk

11.9 INSTRUCTIONS FOR BIT MANIPULATIONS

BIT—Test Bit

The BIT instruction copies the specified bit of the dual data-memory operand into the TC bit of status register ST0.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

data path. There are two general purpose register files (A and B), one for each data path. Each of these files contains sixteen 32-bit registers (A0–A15 for file A and B0–B15 for file B). These register files can be used for data, data address pointers, or condition registers. The 'C62X register files support data ranging in size from packed 16-bit data through 40-bit fixed point.

Each functional unit present in the processor reads directly from and writes directly to the register files within its own data path, i.e., the .L1, .S1, .D1 and .M1units write to register file A and the .L2, .S2, .D2 and .M2 units write to register file B. There are two register file data cross paths (1X and 2X), through which the register files are connected to opposite side registers. These cross path allow functional units from one data path to access a 32-bit operand from the opposite side register file.

It is to be noted that out of eight units only six units in 'C62X can access the register file in the opposite side (M, S and L units, not D units), via a cross path. The M units and S units can access only src2 operand through cross path, whereas in L units, both src1 and src2 operands can be accessed through the cross path.

The 'C62X has two 32-bit paths, LD1 for register file A and LD2 for register file B, for loading data from memory to the register file, and two 32-bit paths, ST1 and ST2, for storing register values to memory from each register file. There are two data address paths DA1 and DA2, each of them connected to .D units in both data paths. This allows data address generated by any one path to access to or from any register.

13.5 FUNCTIONAL UNITS AND ITS OPERATIONS

The L units can perform 32/40-bit arithmetic, logic and compare operations. It is also capable of performing arithmetic operations on 8-, 16-bit operands. The S units are capable of doing 32-bit arithmetic, logic and shift operations. The M units are specially meant for multiply and rotate operations. The D units are used for load, store and address generation operations. The detailed list of the functional units and its operations are given in Table 13.1. Both L and S units are capable of doing data packing, unpacking and byte shift operations. It is to be noted that only .S2 unit is capable of accessing the control register file.

Table 13.1 Functional units and operations

Functional unit	*Operations*
.L unit (.L1 and .L2)	32/40-bit arithmetic and compare operations, leftmost 1 or 0 bit counting for 32 bits, Normalisation count for 32 and 40 bits, 32-bit logical operations, byte shifts, data packing/unpacking, 5-bit constant generation
.S unit (.S1 and .S2)	32-bit arithmetic operations, 32/40-bit shifts and 32-bit bit-field operations, 32-bit logical operations, branches, constant generation, register transfer to/from control registers (.S2 only), byte shifts, data packing/unpacking
.M unit (.M1 and .M2)	16 X 16 multiply operations, bit expansion, bit interleaving/deinterleaving, variable shift operations, rotation
.D unit (.D1 and .D2)	32-bit add, subtract, linear and circular address calculation, load and stores operations, 5-bit constant generation, 32-bit logical operations

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Power-down The power-down logic allows reduced clocking to reduce power consumption. Most of the operating power of CMOS logic dissipates during circuit switching from one logic state to another. By preventing some or all of the chip's logic from switching, significant power savings can be obtained without losing any data or operational context.

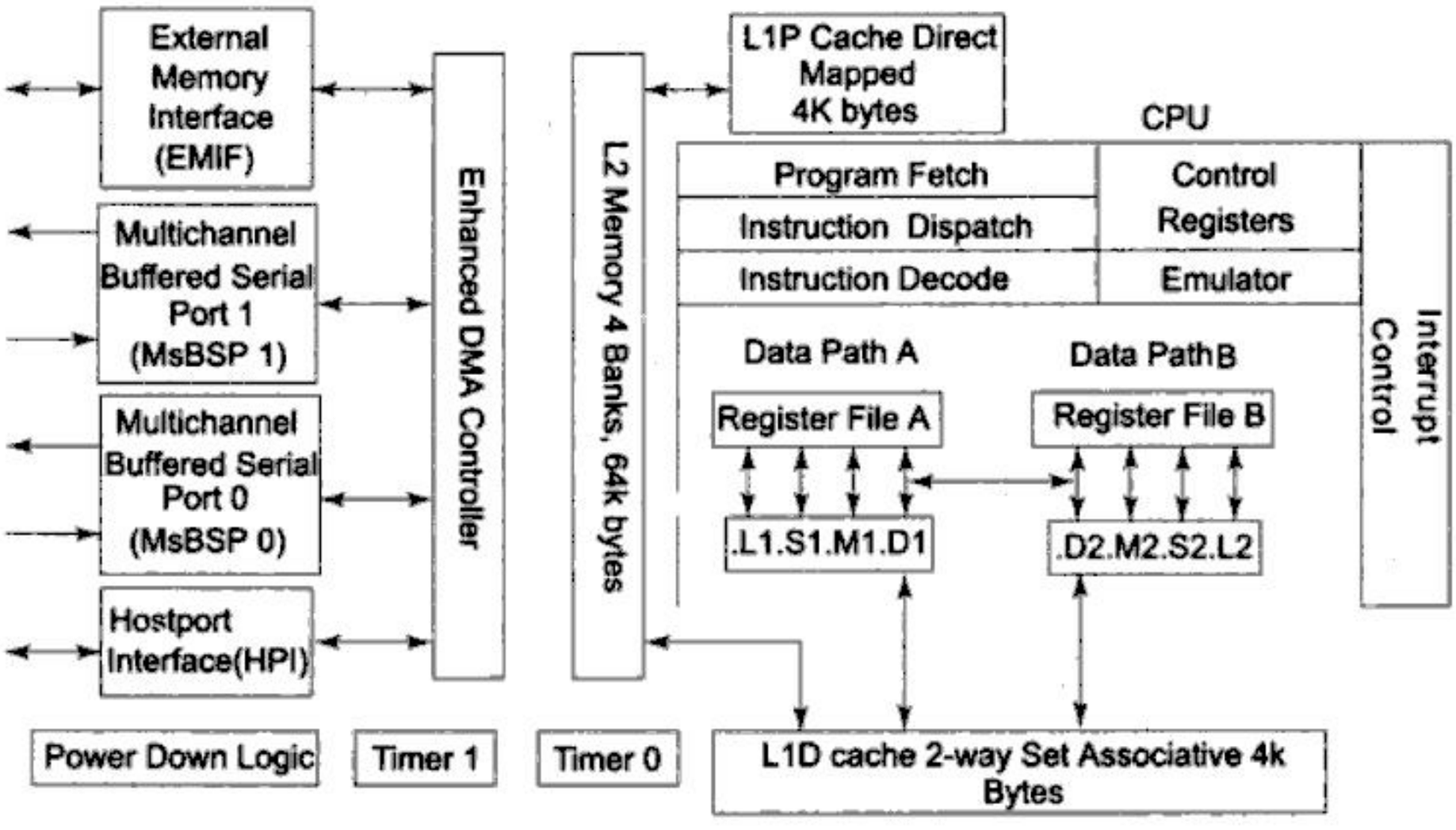


Fig. 13.5 *Block diagram of C6x with peripherals*

13.11 PROGRAM DEVELOPMENT

The C6X programs can be written in assembly language, in C or combinations of both using the code composer studio. All the features listed in Chapter 12 on CCS make the programming and debugging environment user friendly. The CCS feature is available for both the C6X starter kit and C6X EVM.

Review Questions

13.1 Explain how the C6X architecture differs from those of C5X and C54X.
13.2 List the functional units in C6X and explain the function performed by each of them.
13.3 Explain the addressing modes supported by C6X.
13.4 Explain the internal and external memory organisation in C6X.
13.5 Explain the operation of L2 cache controller.
13.6 Explain the use of the memory attribute registers in C6X.
13.7 What is the function of EMIF in C6X?
13.8 Explain the C6X pipeline operation.
13.9 What is meant by glueless interface?
13.10 What is the use of HPI in C6X?

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

You have either reached a page that is unavailable for viewing or reached your viewing limit for this book.

Digital Signal Processors

Architecture, Programming and Applications

The book is designed for a first course in Digital Signal Processors. It blends the Digital Signal Processing theory with its applications on systems using Digital Signal Processors.

Salient Features

- Exposure to DSP architectures and various commercial Digital Signal Processors
- Discusses a wide variety of Texas Instruments (TI) DSP chips including C3X, C5X, C563XX and C6X
- Detailed analysis and explanations of architecture, instructions and applications of each of these TI processors
- A chapter on other ways of realizing DSP design including FPGAs
- An overview of Motorola's DSP563XX processors
- Usage and application of the CODE COMPOSER STUDIO software for design and testing of DSP-based systems
- Pedagogy Rich
  - 243 Illustrations
  - 244 Review Questions
  - 280 Self-test questions.

Tata McGraw-Hill
A Division of The McGraw-Hill Companies

Visit us at: www.tatamcgrawhill.com

ISBN 0-07-047334-X