1 a)

DSP is a technique of performing the mathematical operations on the signals in digital domain. As real time signals are analog in nature we need first convert the analog signal to digital, then we have to process the signal in digital domain and again converting back to analog domain. Thus ADC is required at the input side whereas a DAC is required at the output end. A typical DSP system is as shown in figure 1.3.

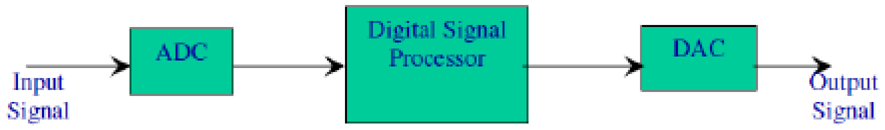


Figure 1.3 : A typical DSP system

A computer or a processor is used for digital signal processing. Antialiasing filter is a LPF which passes signal with frequency less than or equal to half the sampling frequency in order to avoid Aliasing effect. Similarly at the other end, reconstruction filter is used to reconstruct the samples from the staircase output of the DAC (Figure 1.4).

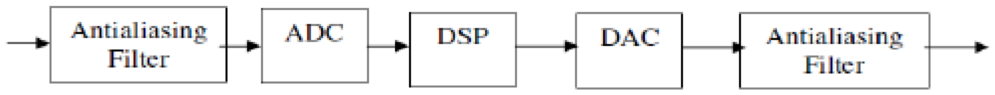


Figure 1.4: The block diagram of a DSP system

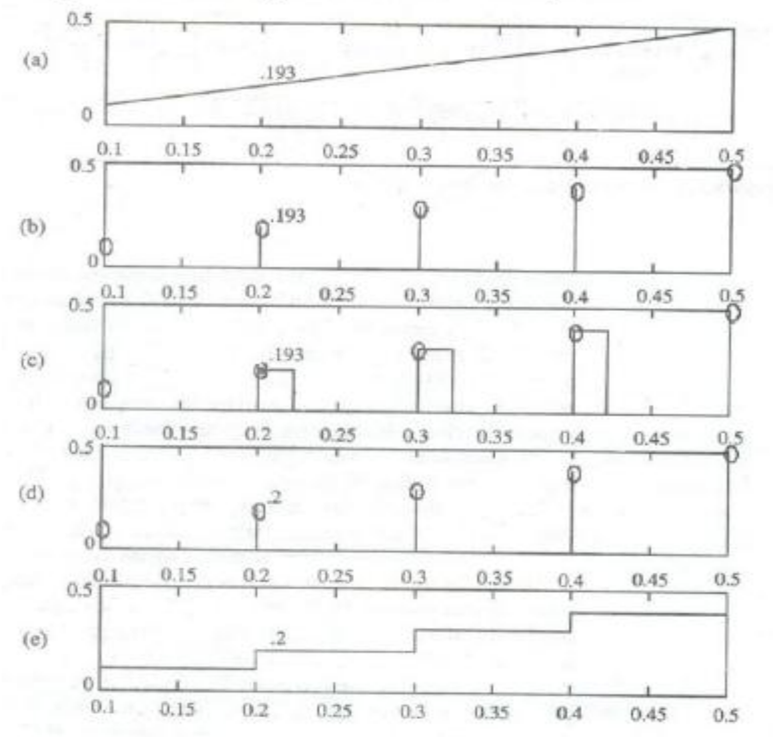


Figure 1.5 : Signals that occur in a DSP system

1b) y (n)= 0.9y (n-1)+0.1x (n)

Taking Z transformation both sides

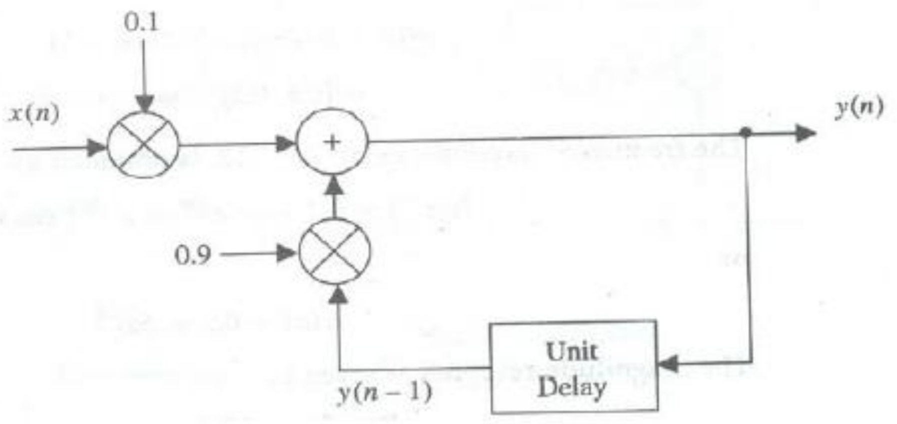
Y (Z) = 0.9 Z-1 Y (Z) + 0.1 X (Z)

Y (Z) [1- 0.9 Z-1] = 0.1 X (Z)

The transfer function of the system is given by the expression,

H (Z)= Y(Z)/X(Z) = 0.1/ [1- 0.9 Z-1]

Realization of the IIR filter with the above difference equation is as shown in figure.



1c) x(n) = {0, 2, 4, 6, 8 }

L=2 bk ={0.5, 1, 0.5}

Insert L-1 = 2-1 =1 zero ➔ w(n) = {0, 0, 2, 0, 4, 0, 6, 0, 8, 0}

y(m) = w(n) * bk

= {0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 4, 0}

2a) **Basic Architectural Features**

A programmable DSP device should provide instructions similar to a conventional microprocessor. The instruction set of a typical DSP device should include the following,

a. Arithmetic operations such as ADD, SUBTRACT, MULTIPLY etc

b. Logical operations such as AND, OR, NOT, XOR etc

c. Multiply and Accumulate (MAC) operation

d. Signal scaling operation

In addition to the above provisions, the architecture should also include,

a. On chip registers to store immediate results

b. On chip memories to store signal samples (RAM)

c. On chip memories to store filter coefficients (ROM)

2b) Shifters are used to either scale down or scale up operands or the results. The following scenarios give the necessity of a shifter

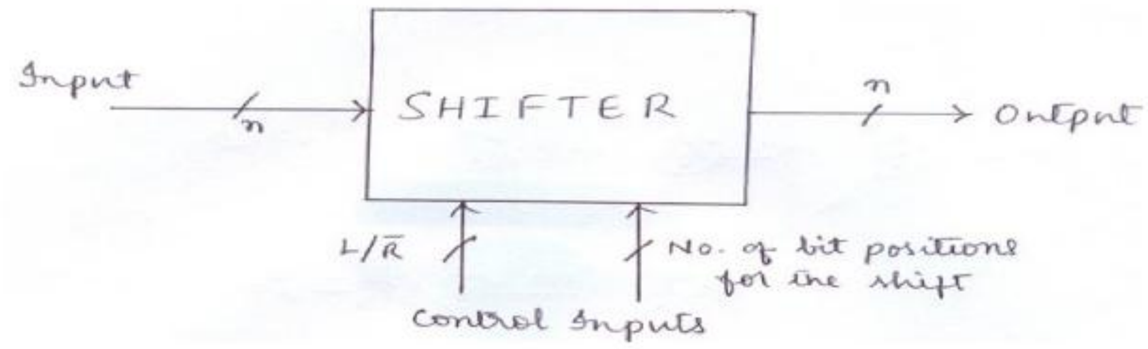
a. While performing the addition of N numbers each of n bits long, the sum can grow up to n+log2 N bits long. If the accumulator is of n bits long, then an overflow error will occur. This can be overcome by using a shifter to scale down the operand by an amount of log2N.

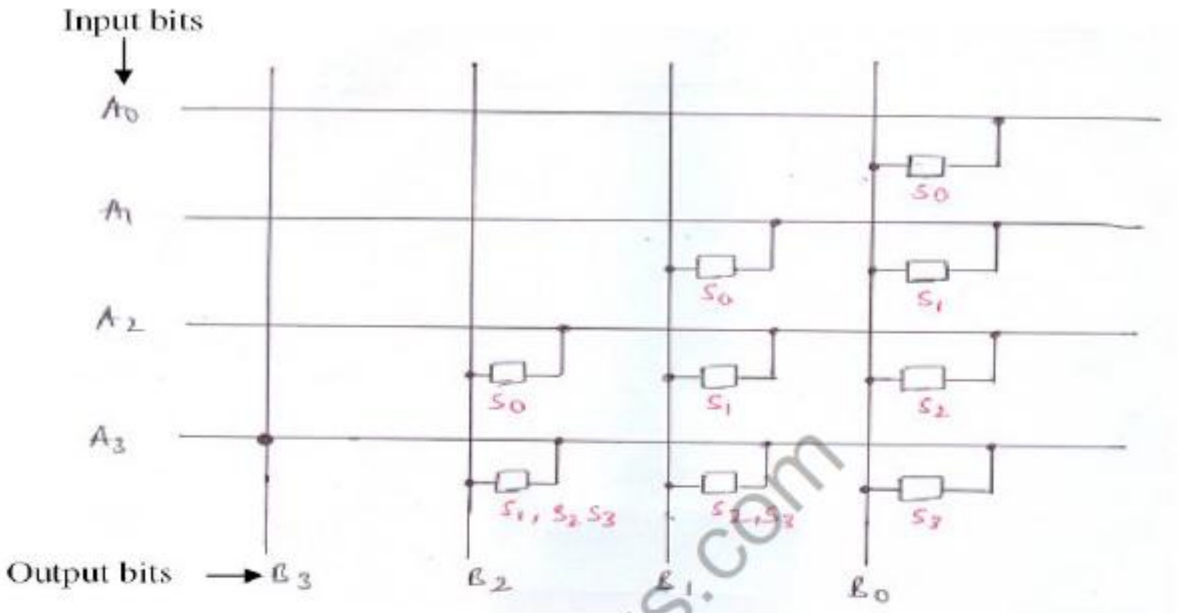
b. Similarly while calculating the product of two n bit numbers, the product can grow up to 2n bits long. Generally the lower n bits get neglected and the sign bit is shifted to save the sign of the product.

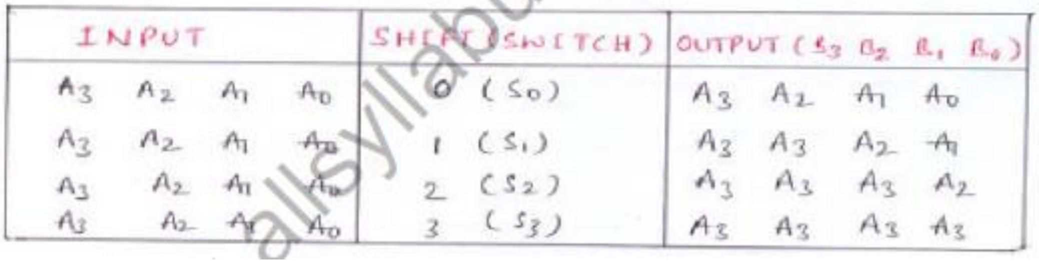
c. Finally in case of addition of two floating-point numbers, one of the operands has to be shifted appropriately to make the exponents of two numbers equal.

From the above cases it is clear that, a shifter is required in the architecture of a DSP.

The block diagram of a typical barrel shifter is as shown in figure







INPUT				SHIFT (SWITCH)	OUTPUT (B_3 B_2 B_1 B_0)			
A_3	A_2	A_1	A_0	0 (S_0)	A_3	A_2	A_1	A_0
A_3	A_2	A_1	A_0	1 (S_1)	A_3	A_3	A_2	A_1
A_3	A_2	A_1	A_0	2 (S_2)	A_3	A_3	A_3	A_2
A_3	A_2	A_1	A_0	3 (S_3)	A_3	A_3	A_3	A_3

Figure depicts the implementation of a 4 bit shift right barrel shifter. Shift to right by 0, 1, 2 or 3 bit positions can be controlled by setting the control inputs appropriately.

2c) **Address Generation Unit**

The main job of the Address Generation Unit is to generate the address of the operands required to carry out the operation. They have to work fast in order to satisfy the timing constraints. As the address generation unit has to perform some mathematical operations in order to calculate the operand address, it is provided with a separate ALU.

Address generation typically involves one of the following operations.

a. Getting value from immediate operand, register or a memory location

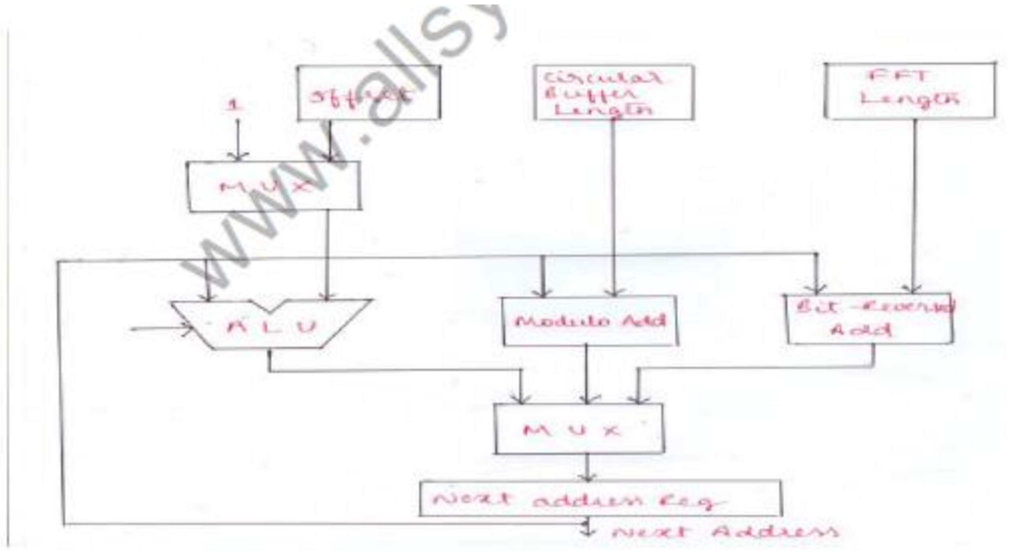
b. Incrementing/ decrementing the current address

c. Adding/subtracting the offset from the current address

d. Adding/subtracting the offset from the current address and generating new address according to circular addressing mode

e. Generating new address using bit reversed addressing mode

The block diagram of a typical address generation unit is as shown in figure



3a)

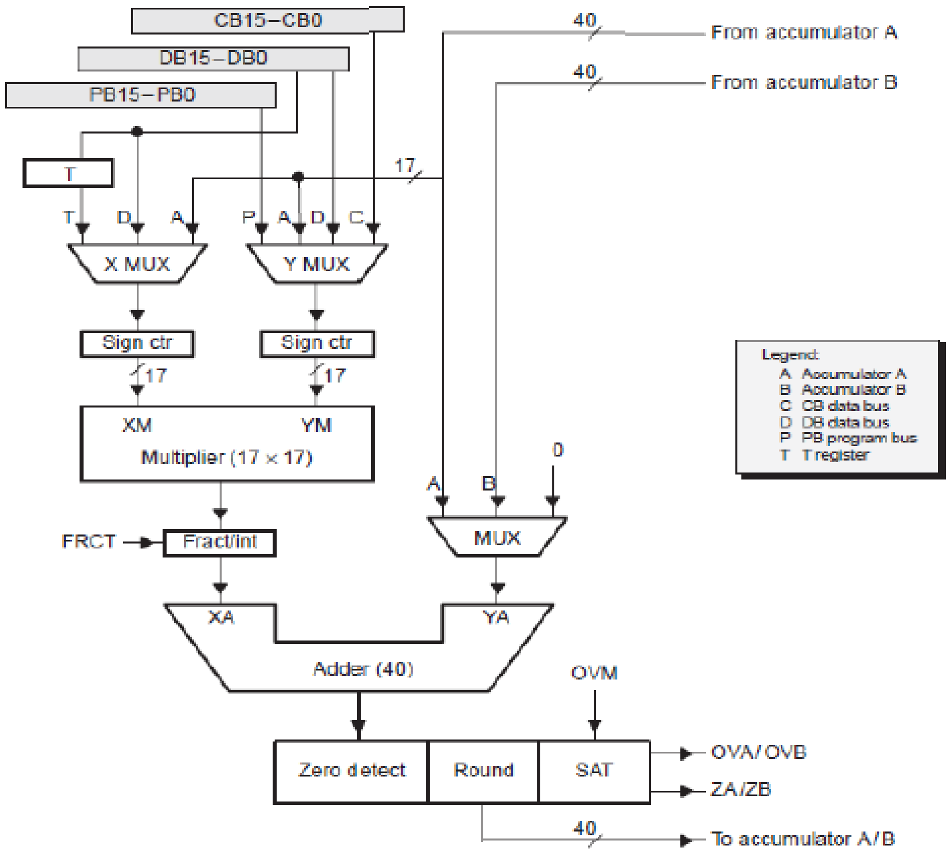


Figure 3.4. Functional diagram of the multiplier/adder unit of TMS320C54xx processors.

Multiplier/adder unit: The kernel of the DSP device architecture is multiplier/adder unit. The multiplier/adder unit of TMS320C54xx devices performs 17 x 17 2's complement multiplication with a 40-bit addition effectively in a single instruction cycle. In addition to the multiplier and adder, the unit consists of control logic for integer and fractional computations and a 16-bit temporary storage register, T. Figure 3.4 show the functional diagram of the multiplier/adder unit of TMS320C54xx processors. The compare, select, and store unit (CSSU) is a hardware unit specifically incorporated to accelerate the add/compare/select operation. This operation is essential to implement the *Viterbi* algorithm used in many signal-processing applications. The exponent encoder unit supports the EXP instructions, which stores in the T register the number of leading redundant bits of the accumulator content. This information is useful while shifting the accumulator content for the purpose of scaling.

3b) A programmable DSP device should provide the programming capability involving branching, looping and subroutines. The implementation of repeat capability should be hardware based so that it can be programmed with minimal or zero overhead. A dedicated register can be used as a counter. In a normal subroutine call, return address has to be stored in a stack thus requiring memory access for storing and retrieving the return address, which in turn reduces the speed of operation. Hence a LIFO memory can be directly interfaced with the program counter.

Program Control

Like microprocessors, DSP also requires a control unit to provide necessary control and timing signals for the proper execution of the instructions. In microprocessors, the controlling is micro coded based where each instruction is divided into microinstructions stored in micro memory. As this mechanism is slower, it is not applicable for DSP applications. Hence in DSP the controlling is hardwired base where the Control unit is designed as a single, comprehensive, hardware unit. Although it is more complex it is faster.

Program Sequencer

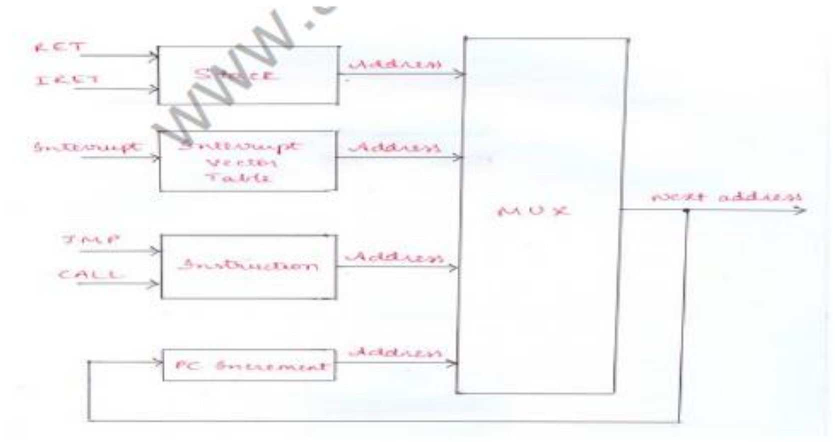
It is a part of the control unit used to generate instruction addresses in sequence needed to access instructions. It calculates the address of the next instruction to be fetched. The next address can be from one of the following sources.

a. Program Counter
b. Instruction register in case of branching, looping and subroutine calls
c. Interrupt Vector table
d. Stack which holds the return address

The block diagram of a program sequencer is as shown in figure

Program sequencer should have the following circuitry:

a. PC has to be updated after every fetch
b. Counter to hold count in case of looping
c. A logic block to check conditions for conditional jump instructions
d. Condition logic-status flag



3c)

i) *AR3-0 → AR3 → AR3 - AR0;
AR3 = 200h - 20h = 1E0h

ii) *AR3+ ---→AR3 → AR3 + 1;
AR3 = 200h + 1 = 201h

iii) *AR3 --→AR3 is not modified.
AR3 = 200h

iv) *+AR3(50h) → AR3 → AR3 + 50h;
AR3 = 200 + 50h = 250h

v) *AR3+0% →AR3→AR3+AR0

AR3=200+40h

vi) *AR3-0B ←27Fh

vii) *AR3+0B ←240h

viii) *AR3+0 ← AR3=200+40h

4a)

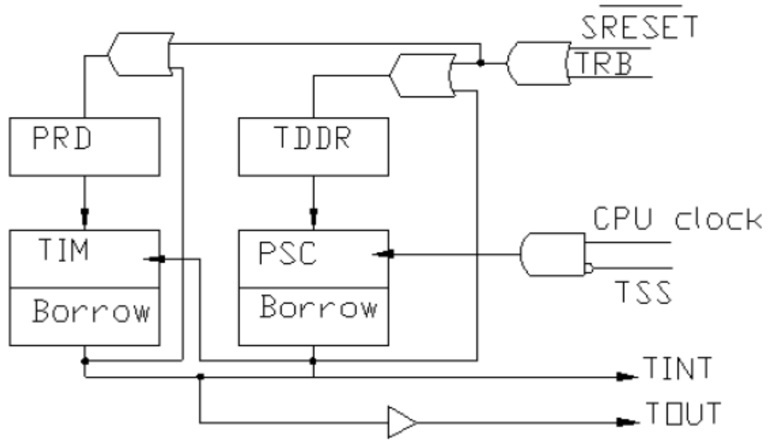
```
.mmregs
.global _c_int000
.text
._c_int00:
          STM   #10h, AR2                 :initialize counter     AR2=10h
          STM   #410h, AR2          :Initialize Pointer AR2=410h
          LD    #0h,  A                   :Initialize sum A=0
          SSBX  SXM                  :Select sign extension mode
START:
          ADD     *AR1+,   A         :Add the next data value
          BANZ    START,  *AR2-      :Repeat if not done
          NOP                             :No  operation

          .end
```

4b)

Hardware Timer

• An on chip down counter
• Used to generate signal to initiate any interrupt or any other process
• Consists of 3 memory mapped registers:
_ The timer register (TIM)
_ Timer period register (PRD)
_ Timer controls register (TCR)
• Pre scaler block (PSC).
• TDDR (Time Divide Down ratio)
• TIN &TOUT

The timer register (TIM) is a 16-bit memory-mapped register that decrements at every pulse from the prescaler block (PSC). The timer period register (PRD) is a 16-bit memory-mapped register whose contents are loaded onto the TIM whenever the TIM decrements to zero or the device is reset (SRESET). The timer can also be independently reset using the TRB signal. The timer control register (TCR) is a 16-bit memory-mapped register that contains status and control bits. Table shows the functions of the various bits in the TCR. The prescaler block is also an on-chip counter. Whenever the prescaler bits count down to 0, a clock pulse is given to the TIM register that decrements the TIM register by 1. The TDDR bits contain the divide-down ratio, which is loaded onto the prescaler block after each time the prescaler bits count down to 0. That is to say that the 4-bit value of TDDR determines the divide-by ratio of the timer clock with respect to the system clock. In other words, the TIM decrements either at the rate of the system clock or at a rate slower than that as decided by the value of the TDDR bits. TOUT and TINT are the output signal generated as the TIM register decrements to 0. TOUT can trigger the start of the conversion signal in an ADC interfaced to the DSP. The sampling frequency of the ADC determines how frequently it receives the TOUT signal. TINT is used to generate interrupts, which are required to service a peripheral such as a DRAM controller periodically. The timer can also be stopped, restarted, reset, or disabled by specific status bits.



4c)

Cycles	Prefetch	Fetch	Decode	Access	Read	Exe& Write	AR3	A
1	LD						85	
2	ADD	LD					85	
3	STL	ADD	LD				85	
4		STL	ADD	LD			86	
5			STL	ADD	LD		86	5h
6				STL	🟩	LD	87	5h
7					STL	ADD	87	1005h
8						STL	87	1005h

5a)

i) .3125 as Q15 number

Q15 notation= .3125 x 2^{15} = 10,240 in decimal=2800h

ii)) -.3125 as Q15 number

Q15 notation= .3125 x 2^{15} = 10,240 in decimal=2800h

Since the number is negative (-.3125), hexadecimal equivalent of 10,240 (i.e) 2800h should be subtracted from FFFFh to get D7FFh

Hence -.3125 = - (10,240 decimal)= -(2800h) = FFFF-2800 = D7FFh

iii) 3.125 as Q7 number

3.125 X 2^7 = 190 h

5b)

```
                        .mmregs
                        .def _c_int00
                        .sect "samples"
InSamples               .include "data_in.dat"       ; Allocate space for x(n)s
OutSamples              .bss y, 200,1             ; Allocate space for y(n)s
SampleCnt               .set 200                  ; Number of samples to
                                                               filter
                        .bss CoefBuf, 16, 1           ; Memory for coeff circular
                                                                buffer
                        .bss SampleBuf, 16, 1   ; Memory for sample  circular buffer
                .sect "FirCoeff"              ; Filter coeff (seq locations)
FirCoeff              .include "coff_fir.dat“
Nm1                   .set 15                            ; N – 1


                        .text
 _c_int00:
                        STM #OutSamples, AR6     ; clear o/p sample buffer
                        RPT #SampleCnt
                        ST #0, *AR6+
                        STM #InSamples, AR5      ; AR5 points to InSamples buffer
                        STM #OutSamples, AR6     ; AR6 points to OutSample buffer
                        STM #SampleCnt, AR4    ; AR4 = Number of samples to
                                                               filter

                        CALL fir_init                       ; Init for filter calculations
                        SSBX SXM                 ; Select sign extension mode

loop:
                        LD *AR5+,A            ; A = next input sample (integer)
                        CALL fir_filter                  ; Call Filter Routine
                        STH A,1,*AR6+                  ; Store filtered sample (integer)
                        BANZ  loop,*AR4-   ; Repeat till all samples filtered
                         nop
                         nop
                         nop
```

5c)

```
                    .mmregs
                         .def _c_int00

                         .sect "samples"
    InSamples        .include "data_in.dat"            ; Allocate space for x(n)s
    OutSamples       .bss y,80,1                 ; Allocate space for y(n)s
    SampleCnt            .set 240                        ; Number of samples to decimate

                         .sect "FirCoeff"        ; Filter coeff (sequential)
    FirCoeff      .include "coeff_dec.dat"
    Nm1           .set 4                         ; Number of filter taps - 1

                           .bss CoefBuf, 5, 1    ; Memory for coeff circular buffer



                        .bss SampleBuf, 5, 1 ; Memory for sample circular buffer
                        .text
 _c_int00:
                        STM #OutSamples, AR6      ; Clear output sample buffer
                        RPT #SampleCnt
                        ST #0, *AR6+

                        STM #InSamples, AR5        ; AR5 points to InSamples buffer
                        STM #OutSamples, AR6       ; AR6 points to OutSample buffer
                        STM #SampleCnt, AR4      ; AR4 = Number of samples to filter
                        CALL dec_init            ; Init for filter calculations
 loop:
                CALL dec_filter               ; Call Filter Routine
                        STH A,1,*AR6+               ; Store filtered sample (integer)
                        BANZ loop,*AR4-             ; Repeat till all samples filtered
                        nop
                        nop
                        nop
```

Decimation Filter Initialization Routine
This routine sets AR2 as the pointer for the sample circular buffer, and AR3 as the pointer for coefficient circular buffer.
BK = Number of filter taps. ; AR0 = 1 = circular buffer pointer increment.

```
  dec_init :
                ST #CoefBuf,AR3                      ; AR3 is the CB Coeff Pointer
                ST #SampleBuf,AR2            ; AR2 is the CB sample pointer
                STM #Nm1,BK                          ; BK = number of filter taps
                RPT #Nm1
                MVPD #FirCoeff, *AR3+%               ; Place coeff in circular buffer
                RPT #Nm1                      ; Clear circular sample buffer
                ST #0h,*AR2+%
                STM #1,AR0;                  ; AR0 = 1 = CB pointer increment
                 RET                                 ; Return
                 nop
                 nop
                 nop
```

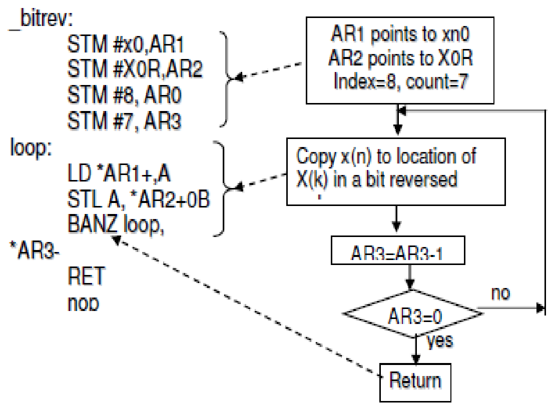
6a)

DITFFT algorithm requires input in bit reversed order. The input sequence can be arranged in bit reverse order by reverse carry add operation. Add half of DFT size (=N/2) to the present bit reversed index to get next bit reverse index. And employ reverse carry propagation while adding bits from left to right. The original index and bit reverse index for N=8 is listed in table 6.3

Table 6.3: Original & bit reverse indices

Original Index	Bit Reversed Index
000	000
001	100
010	010
011	110
100	001
101	101
110	011
111	111

Consider an example of computing bit reverse index. The present bit reversed index be
110. The next bit reversed index is
1 1 0
1 0 0 (N/2=4)
----
0 0 1
There are addressing modes in DSP supporting bit reverse indexing, which do the computation of reverse index. Subroutine for bit reversed index generation:


```
_bitrev:
        STM #x0,AR1
        STM #X0R,AR2
        STM #8, AR0
        STM #7, AR3
loop:
        LD *AR1+,A
        STL A, *AR2+0B
        BANZ loop,
*AR3-
        RET
        nop
```

6b)

i) $\log_2 N = 7$

ii) $\frac{N}{2} = 64$

iii) $\frac{N}{2}\log_2 N = 7 \text{ x } 64 = 448$

iv) Nil

6c)

In any processing system, number of bits per data in signal processing is fixed and it is limited by the DSP processor used. Limited number of bits leads to overflow and it results in erroneous answer. InQ15 notation, the range of numbers that can be represented is -1 to 1. If the value of a number exceeds these limits, there will be underflow / overflow. Data is scaled down to avoid overflow. However, it is an additional multiplication operation. Scaling operation is simplified by selecting scaling factor of 2^-n. And scaling can be achieved by right shifting data by n bits. Scaling factor is defined as the reciprocal of maximum possible number in the operation. Multiply all the numbers at the beginning of the operation by scaling factor so that the maximum number to be processed is not more than 1. In the case of DITFFT computation, consider for example,

$$\begin{aligned} A_I' &= A_I + B_I W_R^r + B_R W_I^r \\ &= A_I + B_I \cos\theta + B_R \sin\theta \qquad (6.7) \end{aligned}$$

$$\text{where } \theta = 2\pi kn/N$$

To find the maximum possible value for LHS term, Differentiate and equate to zero

$$\frac{\partial A_I'}{\partial \theta} = -B_I \sin\theta + B_R \cos\theta = 0$$

$$\Rightarrow B_I \sin\theta = B_R \cos\theta \qquad (6.8)$$

$$\Rightarrow \tan\theta = B_R / B_I$$

$$\therefore \sin\theta = \frac{B_R}{\sqrt{B_R^2 + B_I^2}} \qquad \text{Similarly,} \qquad \cos\theta = \frac{B_I}{\sqrt{B_R^2 + B_I^2}}$$

Substituting them in eq(6.7),

$$A_I' = A_I + \sqrt{B_R^2 + B_I^2}$$

$$A_{I,\max}' = 1 + \sqrt{2} = 2.414$$

Thus scaling factor is 1/2.414=0.414. A scaling factor of 0.4 is taken so that it can be implemented by shifting the data by 2 positions to the right. The symbolic representation of

Butterfly Structure is shown in fig. 6.8. The complete signal flow graph with scaling factor is shown in fig. 6.9.

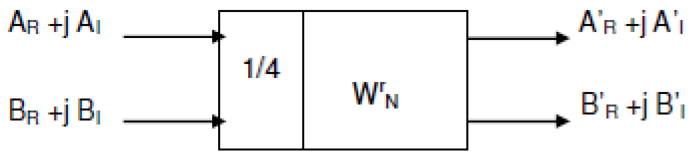
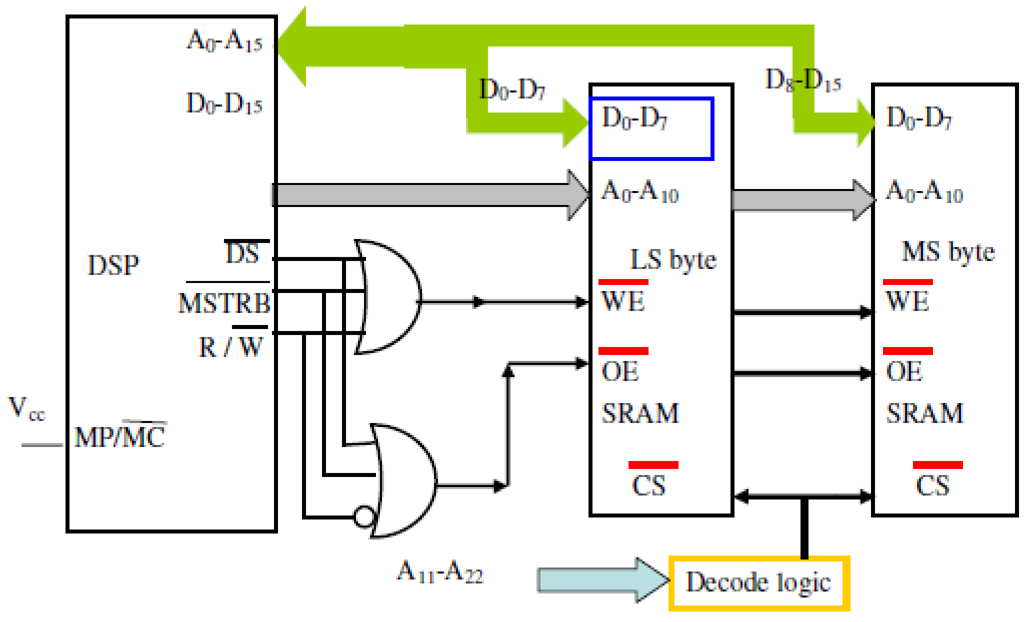
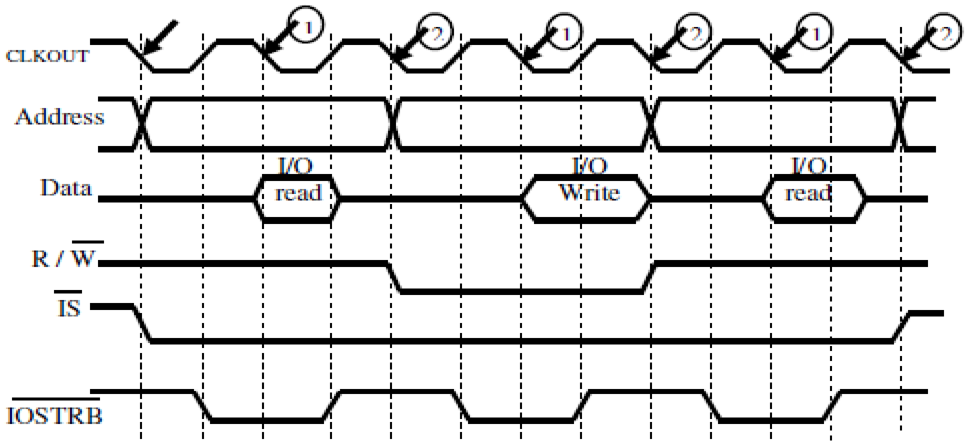


Fig 6.8: Symbolic representation of Butterfly structure with scaling factor

7a)



7b)



7c)

Initialize processor with respect to desired speed, internal registers: PMST, BSCR, SWSR. Initialize internal timer for sampling period of ADC. Apply analog input signal. Send start conversion through TOUT. Continue with any other program execution. ADC interrupts DSP after conversion on INT1. DSP reads 10 bit data and converts to 8 bit by shifting to right. It then processes the sample and sends this data to DAC. DAC converts the data back to analog. The corresponding program is as follows.

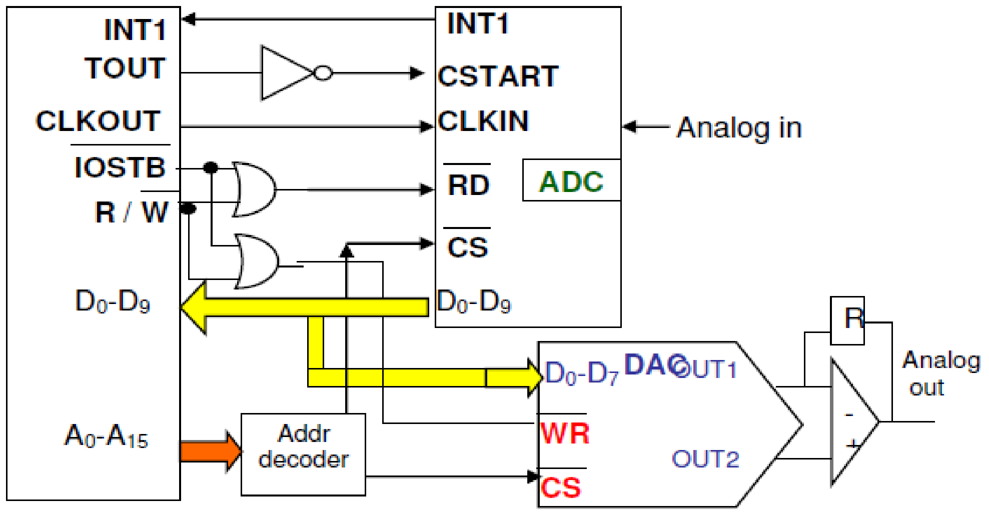
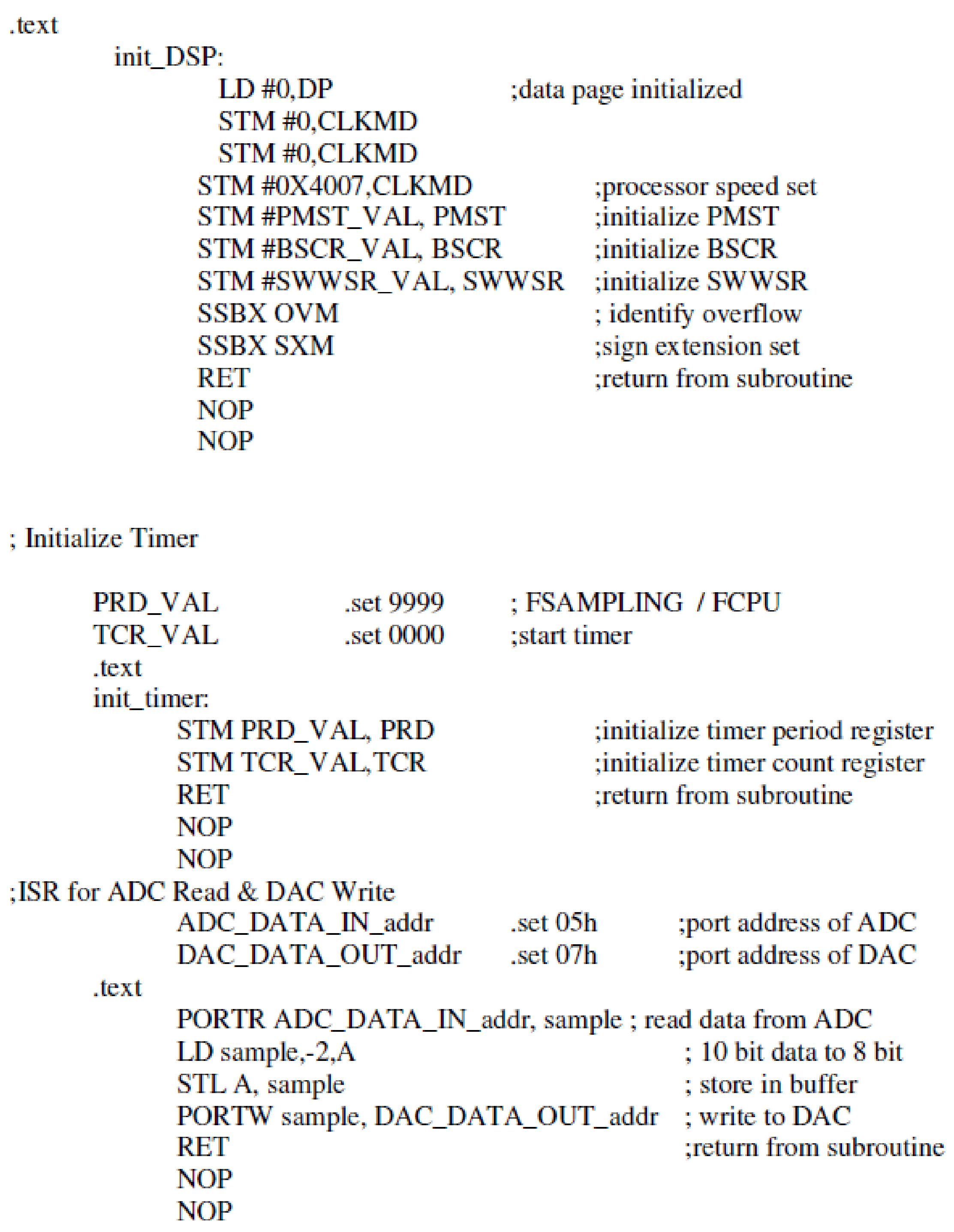


Figure 7.3 : Interfacing DSP to ADC and DAC

```
buffer: .bss sample,1                  ;data buffer
     •   text
_c_int00:

            STM #0X0500,SP             ;initialize Stack Pointer
            SSBX INTM                  ; disable all interrupts
            CALL init_DSP              ; initialize DSP processor
            CALL init_timer            ;initialize timer
            STM #0XFFFF,IFR            ;pending interrupts cleared
            ORM #0002H,IMR             ;INT 1 unmasked
            RSBX INTM                  ; enable all interrupts
;   Initialize DSP Processor
;init_DSP
            PMST_VAL              .set  00A0h; MC & OVLY, interrupt vector is set
            BSCR_VAL              .set  0000h ; bank switching reset, 64K only
            SWWSR_VAL              .set  2000h ;s/w wait, 2 clock wait states
```



```
.text
          init_DSP:
                    LD #0,DP                    ;data page initialized
                    STM #0,CLKMD
                    STM #0,CLKMD
                  STM #0X4007,CLKMD             ;processor speed set
                  STM #PMST_VAL, PMST           ;initialize PMST
                  STM #BSCR_VAL, BSCR           ;initialize BSCR
                  STM #SWWSR_VAL, SWWSR         ;initialize SWWSR
                  SSBX OVM                      ; identify overflow
                  SSBX SXM                      ;sign extension set
                  RET                           ;return from subroutine
                  NOP
                  NOP


; Initialize Timer

        PRD_VAL          .set 9999      ; FSAMPLING  / FCPU
        TCR_VAL          .set 0000      ;start timer
        .text
        init_timer:
                STM PRD_VAL, PRD                ;initialize timer period register
                STM TCR_VAL,TCR                 ;initialize timer count register
                RET                             ;return from subroutine
                NOP
                NOP
;ISR for ADC Read & DAC Write
                ADC_DATA_IN_addr       .set 05h         ;port address of ADC
                DAC_DATA_OUT_addr      .set 07h         ;port address of DAC
        .text
                PORTR ADC_DATA_IN_addr, sample ; read data from ADC
                LD sample,-2,A                          ; 10 bit data to 8 bit
                STL A, sample                           ; store in buffer
                PORTW sample, DAC_DATA_OUT_addr         ; write to DAC
                RET                                     ;return from subroutine
                NOP
                NOP
```

8 a.

Synchronous Serial Interface: There are certain I/O devices which handle transfer of one bit at a time. Such devices are referred to as serial I/O devices or peripherals. Communication with serial peripherals can be synchronous, with processor clock as reference or it can be synchronous. Synchronous serial interface (SSI) makes communication a fast serial communication and asynchronous mode of communication is slow serial communication. However, in comparison with parallel peripheral interface, the SSI is slow. The time taken depends on the number of bits in the data word.

CODEC Interface Circuit: CODEC, a coder-decoder is an example for synchronous serial I/O. It has analog input-output, ADC and DAC. The signals in SSI generated by the DSP are DX: Data Transmit to CODEC, DR: Data Receive from CODEC, CLKX: Transmit data with this clock reference, CLKR: Receive data with this clock reference, FSX: Frame sync signal for transmit, FSR: Frame sync signal for receive, First bit, during transmission or reception, is in sync with these signals, RRDY: indicator for receiving all bits of data and XRDY: indicator for transmitting all bits of data. Similarly, on the CODEC side, signals are FS*: Frame sync signal, DIN: Data Receive from DSP, DOUT: Data Transmit to DSP and SCLK: Tx / Rx data with this clock reference. The block diagram depicting the interface between TMS320C54xx and CODEC is shown in fig. 8.1. As only one signal each is available on CODEC for clock and frame synchronization, the related DSP side signals are connected together to clock and frame sync signals on CODEC. Fig. 8.2 and fig. 8.3 show the timings for receive and transmit in SSI, respectively.

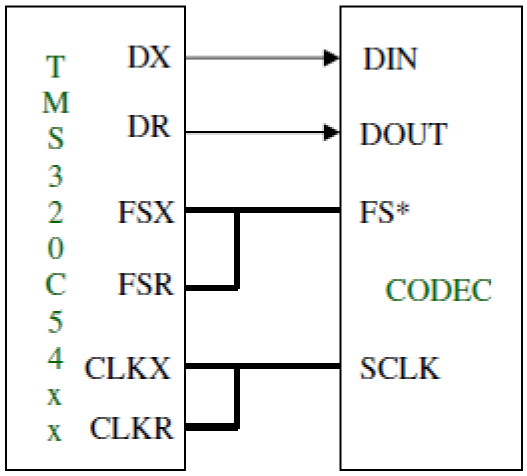


Fig. 8.1: SSI between DSP & CODEC

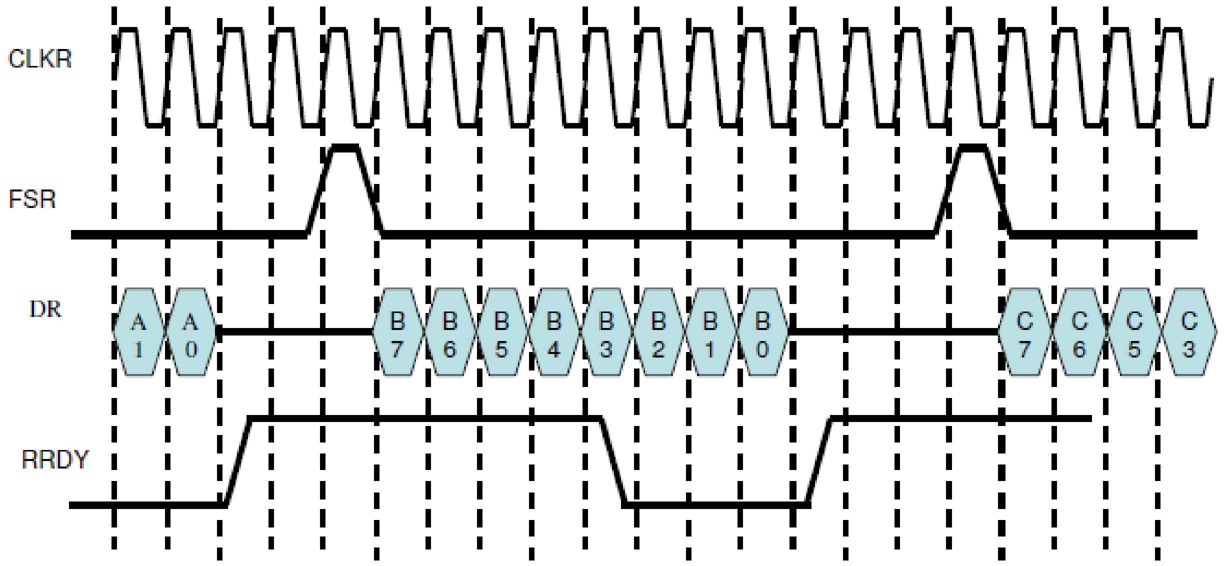


Fig. 8.2: Receive Timing for SSI

As shown, the receiving or transmit activity is initiated at the rising edge of clock, CLKR / CLKX. Reception / Transfer starts after FSR / FSX remains high for one clock cycle. RRDY / XRDY is initially high, goes LOW to HIGH after the completion of data transfer. Each transfer of bit requires one clock cycle. Thus, time required to transfer / receive data word depends on the number of bits in the data word. An example of data word of 8 bits is shown in the fig. 8.2 and fig. 8.3.

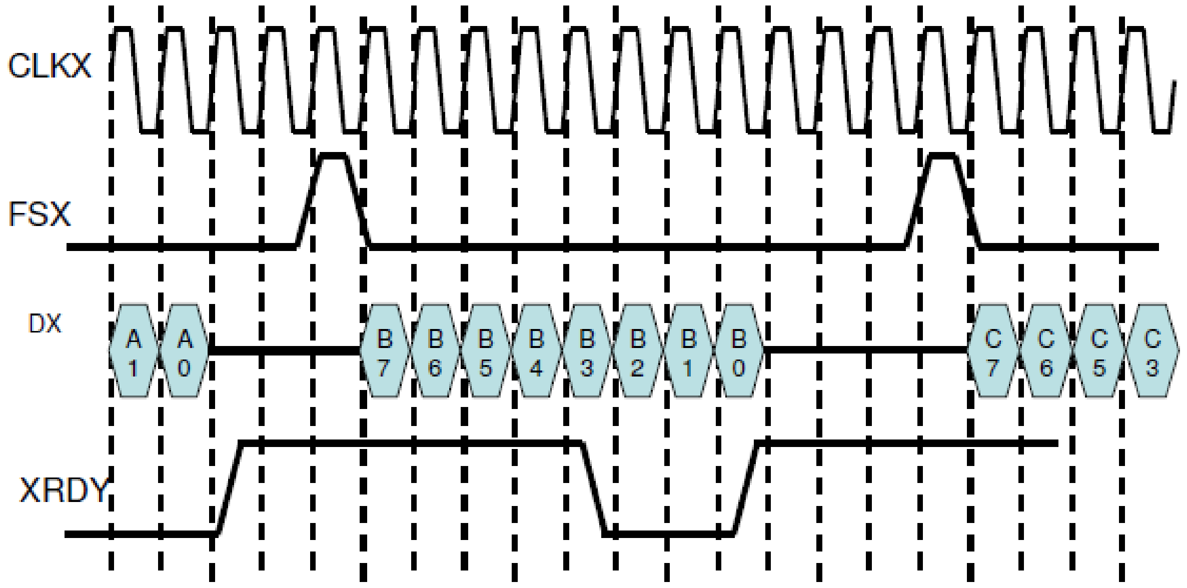


Fig 8.3: Transmit Timing for SSI

8 b.

DSP Based Biotelemetry Receiver:

Biotelemetry involves transfer of physiological information from one remote place to another for the purpose of obtaining experts opinion. The receiver uses radio Frequency links. The schematic diagram of biotelemetry receiver is shown in fig. below. The biological signals may be single dimensional signals such as ECG and EEG or two dimensional signals such as an image, i.e., X-ray. Signal can even be multi dimensional signal i.e., 3D picture. The signals at source are encoded, modulated and transmitted. The signals at destination are decoded, demodulated and analyzed.

An example of processing ECG signal is considered. The scheme involves modulation of ECG signal by employing Pulse Position Modulation (PPM). At the receiving end, it is demodulated. This is followed by determination of Heart beat Rate (HR). PPM Signal either encodes single or multiple signals. The principle of modulation being that the position of pulse decides the sample value. The PPM signal with two ECG signals encoded is shown in fig. below. The transmission requires a sync signal which has 2 pulses of equal interval to mark beginning of a cycle. The sync pulses are followed by certain time gap based on the amplitude of the sample of1st signal to be transmitted. At the end of this time interval there is another pulse. This is again followed by time gap based on the amplitude of the sample of the 2nd signal to be transmitted. After encoding all the samples, there is a compensation time gap followed by sync pulses to mark the beginning of next set of samples. Third signal may be encoded in either of the intervals of 1st or 2nd signal. With two signals encoded and the pulse width as tp, the total time duration is 5tp.

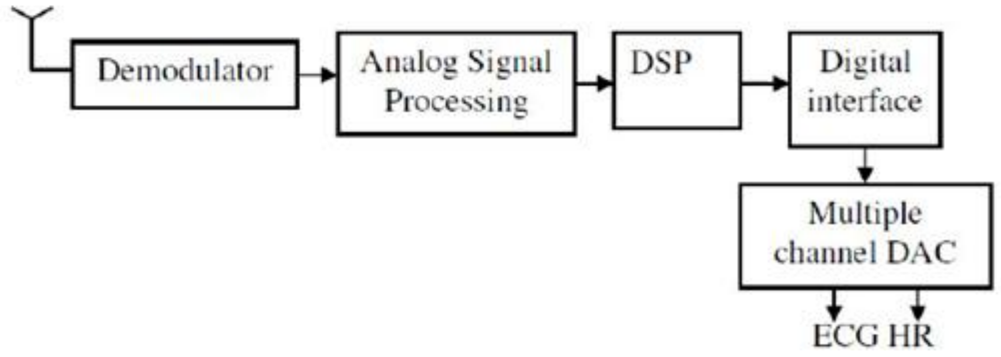
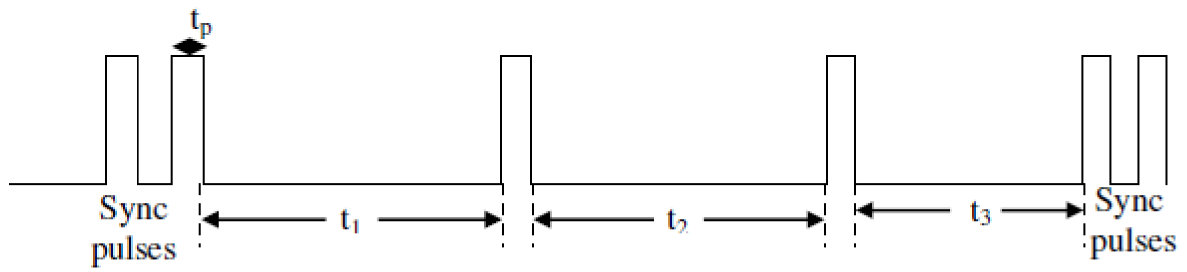


Figure 8.4 : Bio- telemetry receiver

An example of processing ECG signal is considered. The scheme involves modulation of ECG signal by employing Pulse Position Modulation (PPM). At the receiving end, it is demodulated. This is followed by determination of Heart beat Rate (HR). PPM Signal either encodes single or multiple signals. The principle of modulation being that the position of pulse decides the sample value. The PPM signal with two ECG signals encoded is shown in fig. 8 .5.The transmission requires a sync signal which has 2 pulses of equal interval to mark beginning of a cycle. The sync pulses are followed by certain time gap based on the amplitude of the sample of 1st signal to be transmitted. At the end of this time interval there is another pulse. This is again followed by time gap based on the amplitude of the sample of the 2nd signal to be transmitted. After encoding all the samples, there is a compensation time gap followed by sync pulses to mark the beginning of next set of samples. Third signal may be encoded in either of the

intervals of 1st or 2nd signal. With two signals encoded and the pulse width as tp, the total time duration is 5tp.



t_p: each pulse interval

t_1: pulse interval corresponding to sample value of 1st signal

t_2: pulse interval corresponding to sample value of 2nd signal

t_3: compensation time interval

Figure 8.5: A PPM signal with 2 ECG signals
8 c.

An Image Processing System: In comparison with the ECG or speech signal considered so far, image has entirely different requirements. It is a two dimensional signal. It can be a color or gray image. A color image requires 3 matrices to be maintained for three primary colors-red, green and blue. A gray image requires only one matrix, maintaining the gray information of each pixel (picture cell). Image is a signal with large amount of data. Of the many processing, enhancement, restoration, etc., image compression is one important processing because of the large amount of data in image.

To reduce the storage requirement and also to reduce the time and band width required to transmit the image, it has to be compressed. Data compression of the order of factor 50 is sometimes preferred. JPEG, a standard for image compression employs lossy compression technique. It is based on discrete cosine transform (DCT). Transform domain compression separates the image signal into low frequency components and high frequency components. Low frequency components are retained because they represent major variations. High frequency components are ignored because they represent minute variations and our eye is not sensitive to minute variations. Image is divided into blocks of 8 x 8. DCT is applied to each block. Low frequency coefficients are of higher value and hence they are retained. The amount of high frequency components to be retained is decided by the desirable quality of reconstructed image. Forward DCT is given by eq (1).

$$f_{v,u} = \frac{1}{4} c_v c_u \sum_{x=0}^{7} \sum_{y=0}^{7} f_{x,y} \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

------ (1)

Since the coefficients values may vary with a large range, they are quantized. As already noted low frequency coefficients are significant and high frequency coefficients are insignificant, they are allotted varying number of bits. Significant coefficients are quantized precisely, with more bits and insignificant coefficients are quantized coarsely, with fewer bits. To achieve this, a quantization table as shown in fig. 8.20 is employed. The contents of Quantization Table indicate the step size for quantization. An entry as smaller value implies smaller step size, leading to more bits for the coefficients and vice versa.

The quantized coefficients are coded using Huffman coding. It is a variable length coding Huffman Encoding. Shorter codes are allotted for frequently occurring long sequence of 1's & 0's. Decoding requires Huffman table and dequantization table. Inverse DCT is taken employing eq (3). The data blocks so obtained are combined to form complete image. The schematic of encoding and decoding is shown in fig. 8.6.

$$f_{x,y} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} c_u c_v f_{u,v} \cos(\frac{(2x+1)u\pi}{16})\cos(\frac{(2y+1)v\pi}{16})$$

------(2)

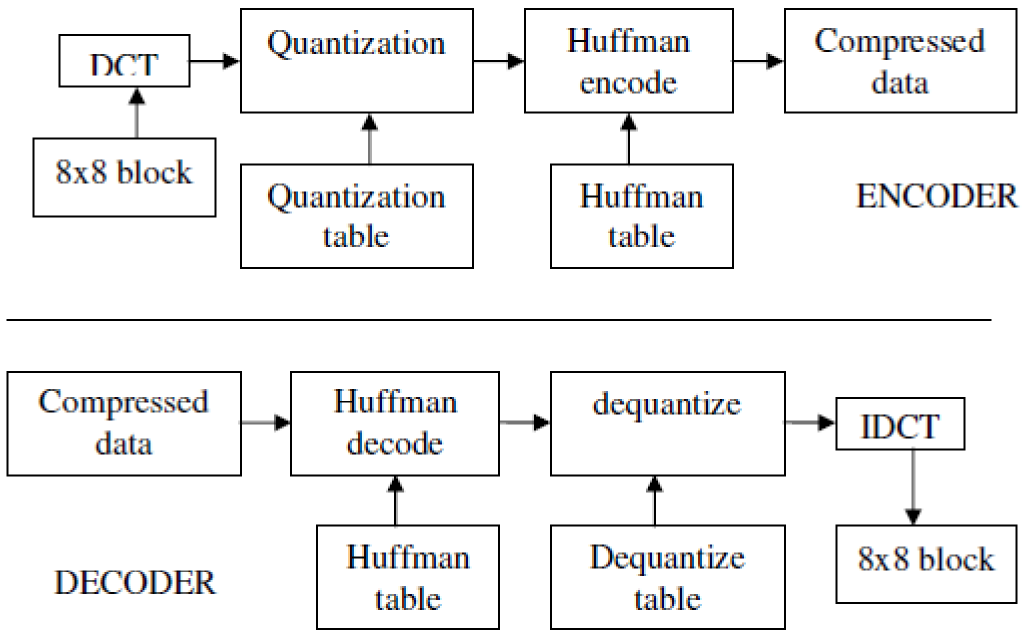


Figure 8.6 : JPEG encoder and decoder