MODULE-4

IMPLEMENTATION OF BASIC DSP ALGORITHMS

❖ LEARNING OBJECTIVES

- ➢ Introduction, The Q-notation,
- ➢ FIR Filters,
- ➢ IIR Filters,
- ➢ Interpolation and
- ➢ Decimation Filters

5.1 Introduction:

In this unit, we deal with implementations of DSP algorithms & write programs to implement the core algorithms only. However, these programs can be combined with input/output routines to create applications that work with a specific hardware.

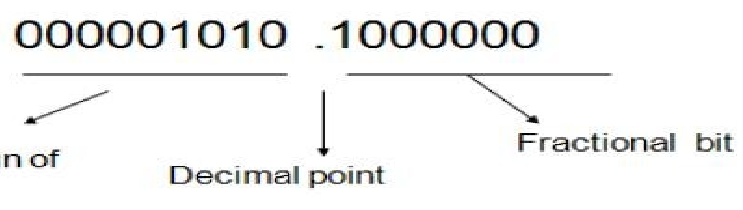
- ➢ Q-notation
- ➢ FIR filters
- ➢ IIR filters
- ➢ Interpolation filters
- ➢ Decimation filters

5.2 The Q-notation:

DSP algorithm implementations deal with signals and coefficients. To use a fixed point DSP device efficiently, one must consider representing filter coefficients and signal samples using fixed- point2's complement representation. Ex: N=16, Range: -2N-1 to +2N-1 -1(-32768 to 32767).Typically, filter coefficients are fractional numbers.

To represent such numbers, the Q-notation has been developed. The Q-notation specifies the number of fractional bits.

Ex: Q7



A commonly used notation for DSP implementations is Q15. In the Q15 representation, the least significant 15 bits represent the fractional part of a number. In a processor where 16 bits are used to represent numbers, the Q15 notation uses the MSB to represent the sign of the number and the rest of the bits represent the value of the number.

In general, the value of a 16-bit Q15 number N represented as:

$b_{15}\ldots\ldots\ldots\ldots b_1b_0$

$N = -b_{15} + b_{14}2^{-1} + \ldots\ldots\ldots\ldots + b_0 2^{-15}$

Range: -1 to $1 - 2^{-15}$

PROBLEM NO 5.1: What values are represented by the 16 bit fixed point number N=4000h is the Q15 and Q7 , Q0 notation?

SOLUTION:

4000h = 0100 0000 0000 0000

Q15 notation = 0.100 0000 0000 0000

$=2^{-1}$

N= .5

Q7 notation = 0100 0000 0.000 0000

$=2^{7}$

=128.0

Q0 notation = 0100 0000 0000 0000.

$=2^{14}$

=16384

Q9 notation = 0100 000.0 0000 000.

$=2^{5}$

=32

PROBLEM NO 5.2: What values are represented by the 16 bit fixed point number N=2000h is the Q15 and Q7 , Q0 notation?

SOLUTION:

2000h = 0010 0000 0000 0000

Q15 notation = 0.010 0000 0000 0000

$=2^{-2}$

N= .25

Q7 notation = 0010 0000 0.000 0000

$=2^{6}$

=64.0

Q0 notation = 0010 0000 0000 0000.

$=2^{13}$

=8192

PROBLEM NO 5.3: What values are represented by the 16 bit fixed point number N=4400h is the Q15 and Q7 , Q0, Q1 notation?

SOLUTION:

4400h = 0100 0100 0000 0000

Q15 notation = 0.100 0100 0000 0000

$=2^{-1} + 2^{-5}$

N= 0.53125

Q7 notation = 0100 0100 0.000 0000

$=2^{3} + 2^{7}$

=136

Q0 notation = 0100 0100 0000 0000.

$=2^{10} + 2^{14}$

=

Q1 notation = 0100 0100 0000 000.0

$=2^{9} + 2^{13}$

=8704

PROBLEM NO 5.3: Determine the value of each of the following numbers represented using the given Q notation?

i) -0.1958 as a Q15 number

ii) 136 as a Q7 number

iii) D0B5h as a Q15 number

iv) 4400h as a Q7 number

SOLUTION:

i) -0.1958 as a Q15 number

0011 0001 0101 1000 = +0.1958

1100 1110 1010 0111 compliment

1 .1100 1110 1010 0111 = -0.1958

Q15 = 11.100 1110 1010 0111 = -1.625

ii) 136 as a Q7 number

136 = 10001000

Q7 = 1.0001000

N = 1.0625

iii) D0B5h as a Q15 number

D0B5h = 1101 0000 1011 0101

Q15 notation = 1.101 0000 1011 0101

iv) 4400h as a Q7 number

Q7= 0100 0100 0000 0000

= 0100 0100 0.000 0000

=+128.0

PROBLEM NO 5.4: Determine the value of each of the following numbers represented using the given Q notation?

i) -352 as Q0 number

ii) 3.125 as Q7 number

iii) BDAFh in Q7 and Q15 number

iv) -0.160123 as Q15 number.

v) 4400h as Q0 number.

SOLUTION:

i) -352 as Q0 number

N =352 × 2^0 = 352 = 160 h

160h = 0001 0110 0000

1110 1001 1111 compliment due to minus sign(1 s compliment)

N = E 9 F h

ii) 3.125 as Q7 number

N = 3.125 × 2^7 = 400 = 190 h

190 h = 0001 0101 0000

Q7 = 0001 0.101 0000

iii) BDAFh in Q7 and Q15 number

N=1011 1101 1010 1111

Q7 = 1011 1101 1.010 1111

= 0100 0011 0.101 0000 compliment(1 s comp)

N = -132.625

Q15=1.011 1101 1010 1111

N = 0.100 0010 0101 0000 compliment (1 s comp)

N = -0.518066406

iv) -0.160123 as Q15 number.

N=0.160123 × 2^{15} = 5247 = 147F h

N=147Fh

N=0001 0100 0111 1111

N=1110 1011 1000 0000 compliment due to -tive sign

N=EB80h

Q15=1.110 1011 1000 0000

vi) 4400h as Q0 number.

N= 0100 0100 0000 0000.

N= 2^{14} + 2^{10}

N= 17508

<u>PROBLEM NO 5.4:</u> Determine the value of each of the following numbers represented using the given Q notation?

- **i) 4400h as Q7 number.**
- **ii) 4400h as Q0 number.**
- **iii) 2CCCh Q15 number**
- **iv) E6EFh as Q15 number.**
- **v) 4400h as Q9 number.**

SOLUTION:

i) 4400h as Q7 number

N= 0100 0100 0000 0000

Q7= 0100 0100 0.000 0000

$N=2^3 +2^7=72$

ii) 4400 h as Q0 number

N= 0100 0100 0000 0000

Q0=0100 0100 0000 0000

N=17408

iii) 2CCCh as Q15 number

N=0010 1100 1100 1100

N=7372

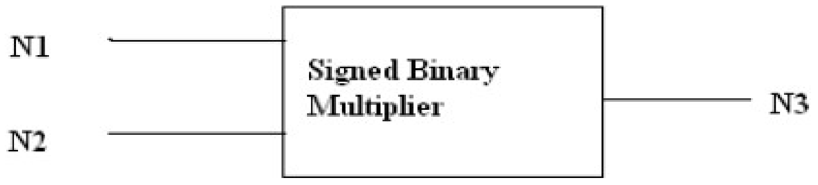
iv) E6EFh as Q15 number

N=1110 0101 1110 1111

N=0001 1010 0001 0000 compliment

$N=2^4+2^{9+}2^{11} +2^{12} = -6672$

Multiplication of numbers represented using the Q-notation is important for DSP implementations. Figure 5.1(a) shows typical cases encountered in such implementations.



N1(16 bit)	N2(16 bit)	N3(16 bit)
Q0	Q0	Q0
Q0	Q15	Q15
Q15	Q15	Q30

Figure 5.1Multiplication of numbers represented using Q-notation

Program to multiply two Q15 numbers

i.e N1×N2 = N1*N2

Where
N1 &N2 are 16-bit numbers in Q15 notation
N1×N2 is the 16-bit result in Q15 notation

```
                        .mmregs                 ; memory mapped registers
                        .data                   ; sequential locations
N1:                     .word    4000h          ; N1=0.5 (Q15 numbers)
N2:                     .word    2000h          ; N2=0.25 (Q15 numbers)
N1×N2                   .space   10h            ; space for N1×N2
                        .text
                        .ref            _c_int00
                        .sect           ".vectors "
RESET:          b       _c_int00                ; reset vector
                nop
```

5.3 FIR Filters:

A finite impulse response (FIR) filter of order N can be described by the difference

$$y[n] = \sum_{m=0}^{m=N-1} h(m)x(n-m)$$

equation

The expanded form is y(n)=h(N-1)x(n-(N-1))+h(N-2)x(n-(N-2))+ ...h(1)x(n-1)+h(0)x(n)

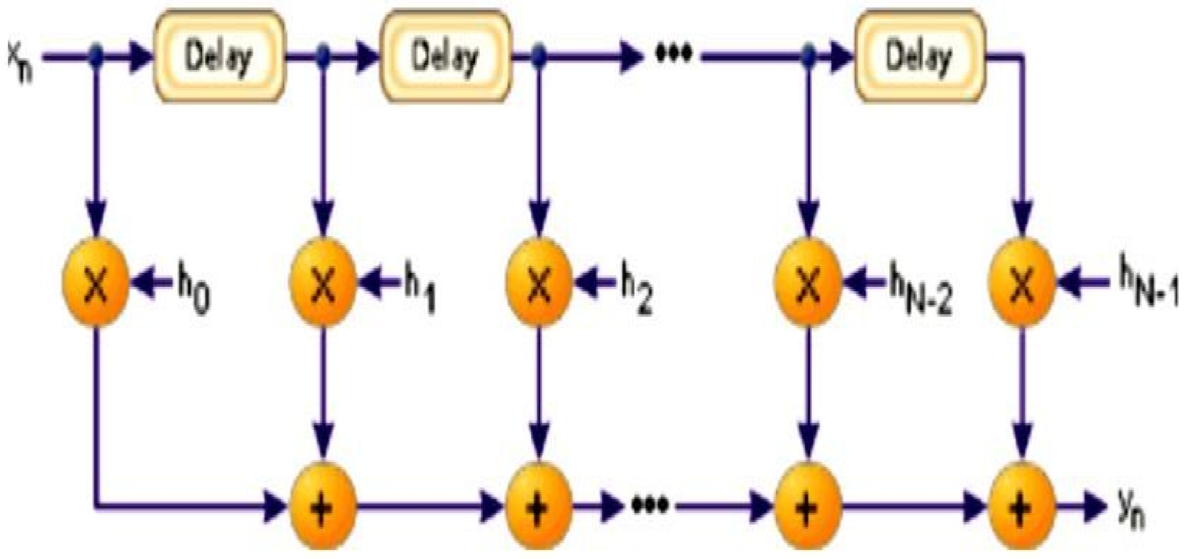


Figure 5.2 A FIR filter implementation block diagram

The implementation requires signal delay for each sample to compute the next output, y(n+1), is given as y(n+1)=h(N-1)x(n-(N-2))+h(N-2)x(n-(N-3))+ ...h(1)x(n)+h(0)x(n+1) Figure 5.3 shows the memory organization for the implementation of the filter. The filter Coefficients and the signal samples are stored in two circular buffers each of a size equal to the filter. AR2 is used to point to the samples and AR3 to the coefficients. In order to start with the last product, the pointer register AR2 must be initialized to access the signal sample x(2-(N-1)), and the pointer register AR3 to access the filter coefficient h(N-1). As each product is computed and added to the previous result, the pointers advance circularly. At the end of the computation, the signal sample pointer is at the oldest sample, which is replaced with the newest sample to proceed with the next output computation.

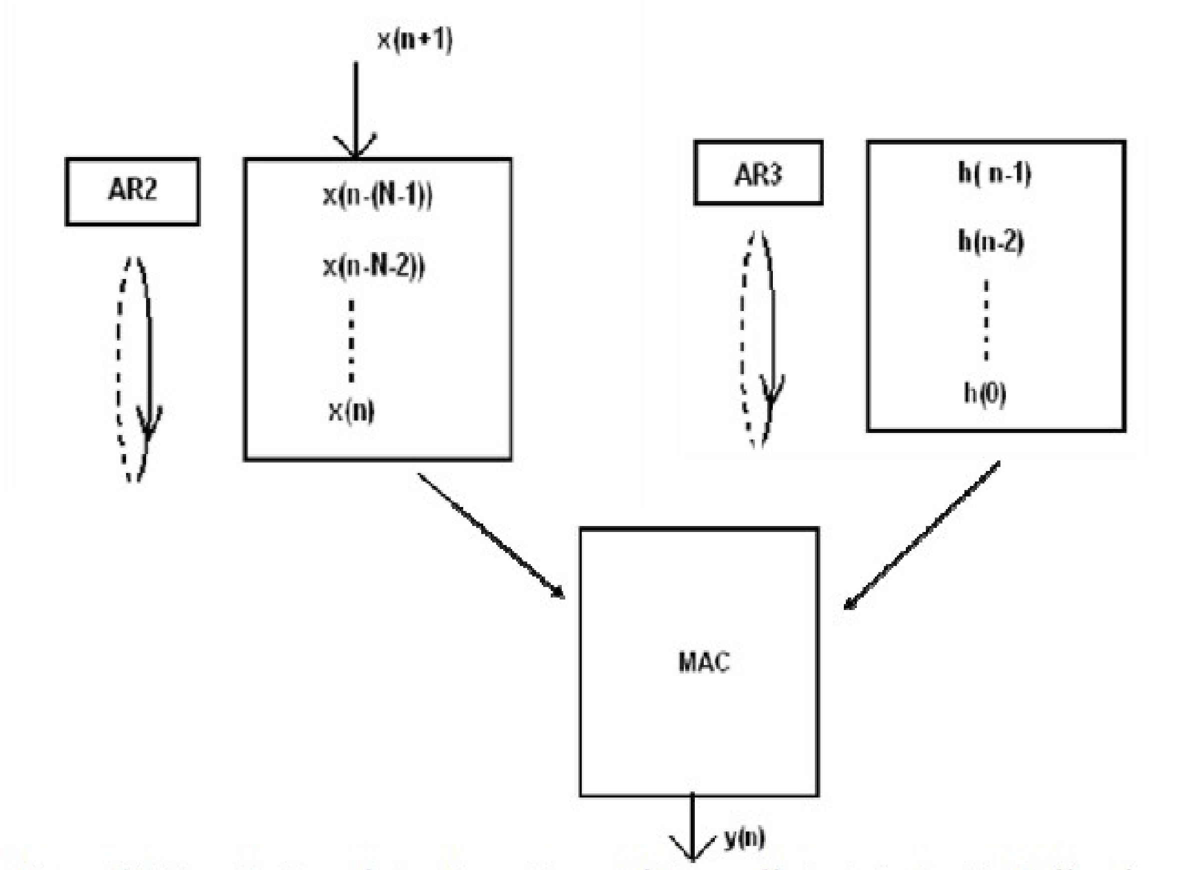


Figure 5.3 Organization of signal samples and filter coefficients in circular buffers for a FIR filter implementation.

Program to implement an FIR filter:
It implements the following equation;

y(n)=h(N-1)x(n-(N-1))+h(N-2)x(n-(N-2))+ ...h(1)x(n-1)+h(0)x(n) Where N = Number of filter coefficients = 16.h(N-1), h(N-2),...h(0) etc are filter coefficients (q15numbers) .
The coefficients are available in file: coeff_fir.dat.
x(n-(N-1)),x(n-(N-2),...x(n) are signal samples(integers). The input x(n) is received from the data file: data_in.dat. The computed output y(n) is placed in a data buffer.

```
                    .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
```

FIR Filter Initialization Routine

```
; This routine sets AR2 as the pointer for the sample circular buffer
; AR3 as the pointer for coefficient circular buffer.
; BK = Number of filter taps - 1.
; AR0 = 1 = circular buffer pointer increment


fir_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 - 1                      ; Clear circular sample buffer
        ST #0h,*AR2+%
        STM #1,AR0                  ; AR0 = 1 = CB pointer increment
        RET
        nop
        nop
        nop
```

FIR Filter Routine

; Enter with A=the current sample x(n)-an integer, AR2 pointing to the location for the current sample x(n),andAR3pointingtotheq15coefficienth(N-1). Exit with A = y(n) as q15 number.

```
fir_filter:

            STL A, *AR2+0%              ; Place x(n)in the sample buffer
            RPTZ A, #Nm1                      ; A = 0
            MAC *AR3+0%,*AR2+0%,A       ; A = filtered sum (q15)
            RET
            nop
            nop
            nop
            .end
```

5.4 IIR Filters:

An infinite impulse response (IIR) filter is represented by a transfer function, which is a ratio of two polynomials in z. To implement such a filter, the difference equation representing the transfer function can be derived and implemented using multiply and add operations. To show such an implementation, we consider a second order transfer function given by

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}}$$

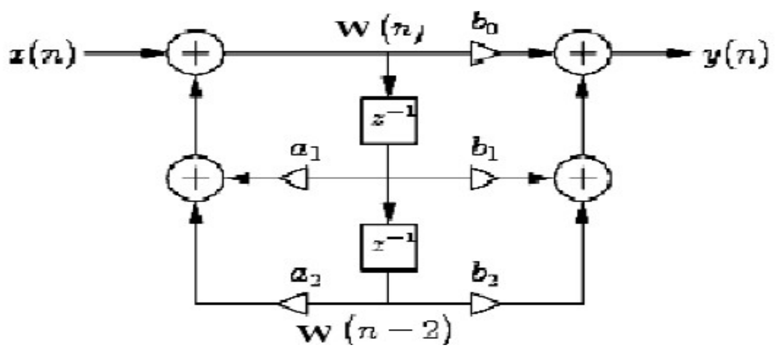


Figure5.4 Block diagram of second order IIR filter

$$\begin{aligned} w(n) &= x(n) + a_1 w(n-1) + a_2 w(n-2) \\ y(n) &= b_0 w(n) + b_1 w(n-1) + b_2 w(n-2) \end{aligned}$$

Program for IIR filter:
The transfer function is

H(z) = [b0 + b1.z(-1)+ b2.z**(-2)]/[1- a1.z**(-1)- a2.z**(-2)]**

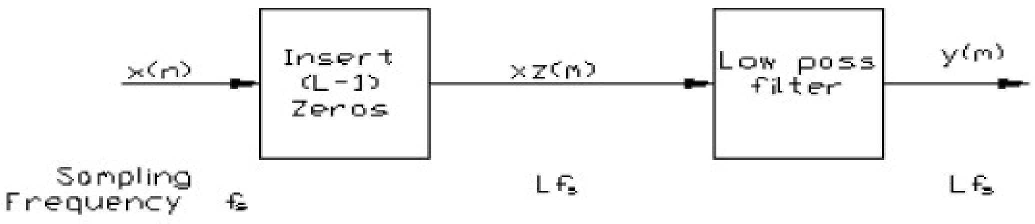
Which is equivalent to the equations: w(n) = x(n) + a1.w(n-1) + a2.w(n-2)
y(n) = b0.w(n) + b1.w(n-1) + b2.w(n-2)
Where w(n), w(n-1), and w(n-2) are the intermediate variables used in computations (integers).a1, a2, b0, b1, and b2 are the filter coefficients (q15 numbers). x(n) is the input sample (integer). Input samples are placed in the buffer, In Samples, from a data file, data_in.dat y(n) is the computed output (integer). The output samples are placed in a buffer, OutSamples.

```
                        .mmregs
                        .def _c_int00
                        .sect "samples"
InSamples               .include "data_in.dat"      ; Allocate space for x(n)s
OutSamples              .bss y,200,1                ; Allocate buffer for y(n)s
SampleCnt               .set 200                    ; Number of samples to filter

; Intermediate variables (sequential locations)
wn              .word 0                         ;initial w(n)
wnm1  .word 0                           ;initial w(n-1) =0
wnm2  .word 0                           ;initial w(n-2)=0


                  .sect "coeff"
; Filter coefficients (sequential locations)
b0              .word 3431                      ; b0 = 0.104
b1              .word -3356             ; b1 = -0.102
b2              .word 3431                      ; b2 = 0.104
a1              .word -32767            ; a1 = -1
a2              .word 20072             ; a2 = 0.612


                .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

 loop:
        LD *AR5+,15,A                   ; A = next input sample (q15)
        CALL iir_filter                 ; Call Filter Routine
        STH A,1,*AR6+                   ; Store filtered sample (integer)
        BANZ loop,*AR4-         ; Repeat till all samples filtered
        nop
        nop
        nop
```

```
IIR Filter Subroutine
; Enter with A = x(n) as q15 number
; Exit with A = y(n) as q15 number
; Uses AR2 and AR3

iir_filter:
            SSBX SXM    ; Select sign extension mode
        ;w(n)=x(n)+ a1.w(n-1)+ a2.w(n-2)

        STM #a2,AR2                         ; AR2 points to a2
        STM #wnm2, AR3                  ; AR3 points to w(n-2)
        MAC *AR2-,*AR3-,A                   ; A = x(n)+ a2.w(n-2)
                                            ; AR2 points to a1 & AR3 to w(n-
1)
        MAC *AR2-,*AR3-,A                   ; A = x(n)+ a1.w(n-1)+ a2.w(n-2)
                                            ; AR2 points to b2 & AR3 to w(n)
        STH A,1,*AR3                        ; Save w(n)

;y(n)=b0.w(n)+ b1.w(n-1)+ b2.w(n-2)

        LD #0,A                 ; A = 0
        STM #wnm2,AR3           ; AR3 points to w(n-2)
        MAC *AR2-,*AR3-,A       ; A = b2.w(n-2)
                                ; AR2 points to b1 & AR3 to w(n-1)
        DELAY *AR3              ; w(n-1) -> w(n-2)
        MAC *AR2-,*AR3-,A  ; A = b1.w(n-1)+ b2.w(n-2)
                                ; AR2 points to b0 & AR3 to w(n)
        DELAY *AR3              ; w(n) -> w(n-1)


        MAC *AR2,*AR3,A ; A = b0.w(n)+ b1.w(n-1)+ b2.w(n-2)
        RET                      ; Return
        Nop
        Nop
        Nop
        .end
```

5.5 Interpolation Filters:

An *interpolation filter* is used to increase the sampling rate. The interpolation process involves inserting samples between the incoming samples to create additional samples to increase the sampling rate for the output. One way to implement an interpolation filter is to first insert zeros between samples of the original sample sequence. The zero-inserted sequence is then passed through an appropriate lowpass digital FIR filter to generate the interpolated sequence. The interpolation process is depicted in Figure 5.5



Example:

X(n) = [0 2 4 6 8 10] ;input sequence

Xz(n) = [0 0 2 0 4 0 6 0 8 0 10 0] ;zero inserted sequence

h(n) = [0.5 1 0.5] ;impulse sequence

Y(n) = [0 0 1 2 3 4 5 6 7 8 9 10 5 0] ;interpolated sequence y(n)

Figure 5.5:The interpolation process

The kind of interpolation carried out in the examples is called *linear interpolation* because the convolving sequence h(n) is derived based on linear interpolation of samples. Further, in this case, the h(n) selected is just a second-order filter and therefore uses just two adjacent samples to interpolate a sample. A higher-order filter can be used to base interpolation on more input samples. To implement an ideal interpolation. Figure 5.6 shows how an interpolating filter using a 15-tap FIR filter and an interpolation factor of 5 can be implemented. In this example, each incoming samples is followed by four zeros to increase the number of samples by a factor of 5.

The interpolated samples are computed using a program similar to the one used for a FIR filter implementation. One drawback of using the implementation strategy depicted in Figure 5.7 is that there are many multiplies in which one of the multiplying elements is zero. Such multiplies need not be included in computation if the computation is rearranged to take advantage of this fact. One such scheme, based on generating what are called *poly-phase sub-filters*, is available for reducing the computation. For a case where the number of filter coefficients N is a multiple of the interpolating factor L, the scheme implements the interpolation filter using the equation. Figure 5.7 shows a scheme that uses poly-phase sub-filters to implement the interpolating filter using the 15-tap FIR filter and an interpolation factor of 5. In this implementation, the 15 filter taps are arranged as shown and divided into five 3-tap sub filters. The input samples x(n), x(n-1) and x(n-2) are used five times to generate the five output samples. This implementation requires 15 multiplies as opposed to 75 in the direct implementation of Figure 5.7.

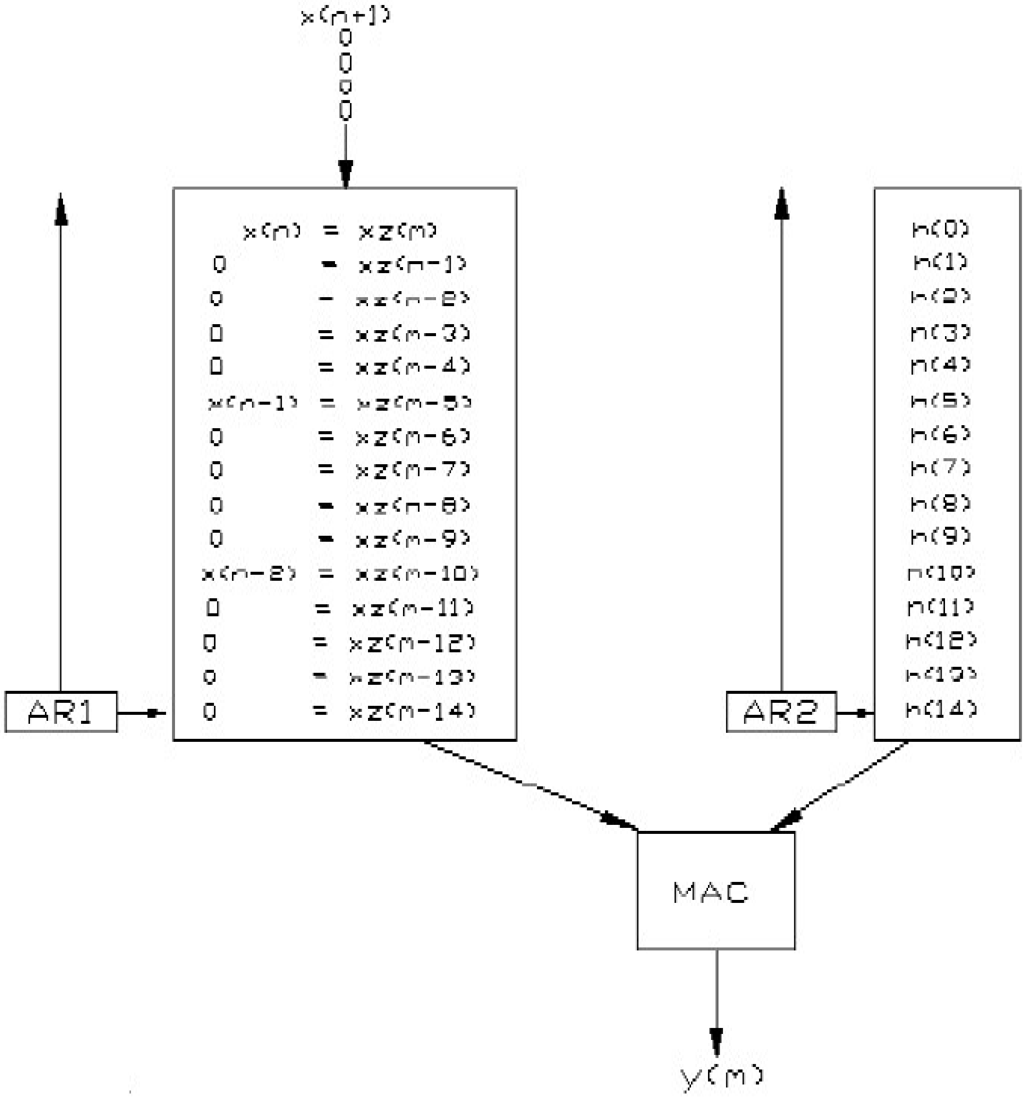


Figure 5.6 interpolating filter using a 15-tap FIR filter and an interpolation factor of 5

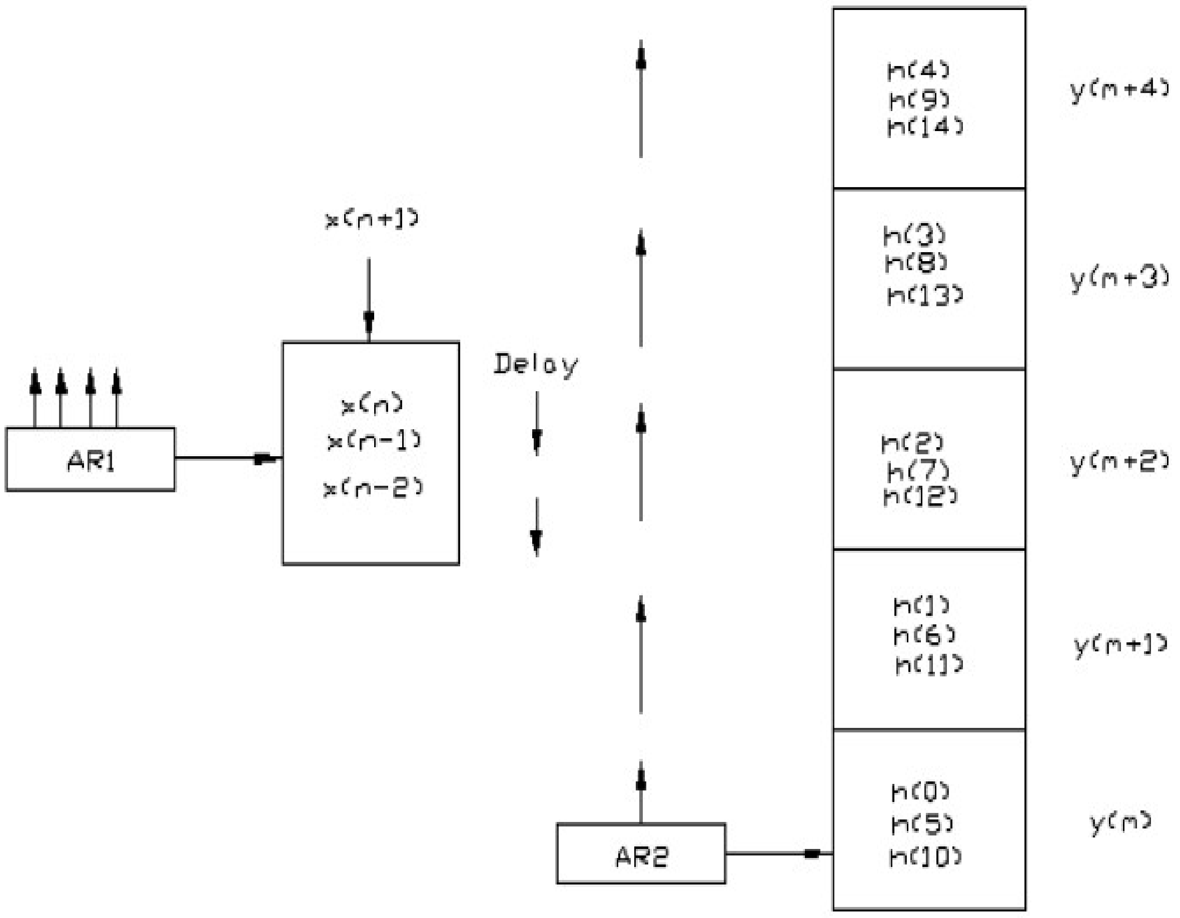


Figure5.7: A scheme that uses poly-phase sub-filters to implement the interpolating filter

Using the 15-tap FIR filter and an interpolation factor of 5

$$y(m+i) = \sum_{K=0}^{N/L-1} h(KL+i)x(n-K)$$

Where i = 0,1,2,…. (L-1) and m = nL.

5.6 Decimation Filters:

A decimation filter is used to decrease the sampling rate. The decrease in sampling rate can be achieved by simply dropping samples. For instance, if every other sample of a sampled sequence is dropped, the sampling the rate of the resulting sequence will be half that of the original sequence. The problem with dropping samples is that the new sequence may violate the sampling theorem, which requires that the sampling frequency must be greater than two times the highest frequency contents of the signal.

To circumvent the problem of violating the sampling theorem, the signal to be decimated is first filtered using a low pass filter. The cutoff frequency of the filter is chosen so that it is less than half the final sampling frequency. The filtered signal can be decimated by dropping samples. In fact, the samples that are to be dropped need not be computed at all. Thus, the implementation of a decimator is just a FIR filter implementation in which some of the outputs are not calculated.

Figure 5.8 shows a block diagram of a decimation filter. Digital decimation can be implemented as depicted in Figure 5.9 for an example of a decimation filter with decimation factor of
3. It uses a low pass FIR filter with 5 taps. The computation is similar to that of a FIR filter. However, after computing each output sample, the signal array is delayed by three sample intervals by bringing the next three samples into the circular buffer to replace the three oldest samples.

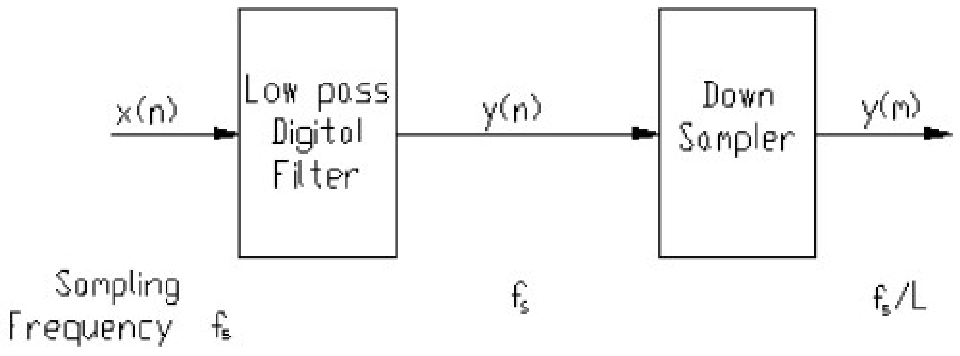


Figure 5.8: The decimation process

$$y(m) = y(nL) = \sum_{K=0} h(K)x(nL-K):$$

Where n = 0,1,2,.....
L=decimation factor
N=filter size

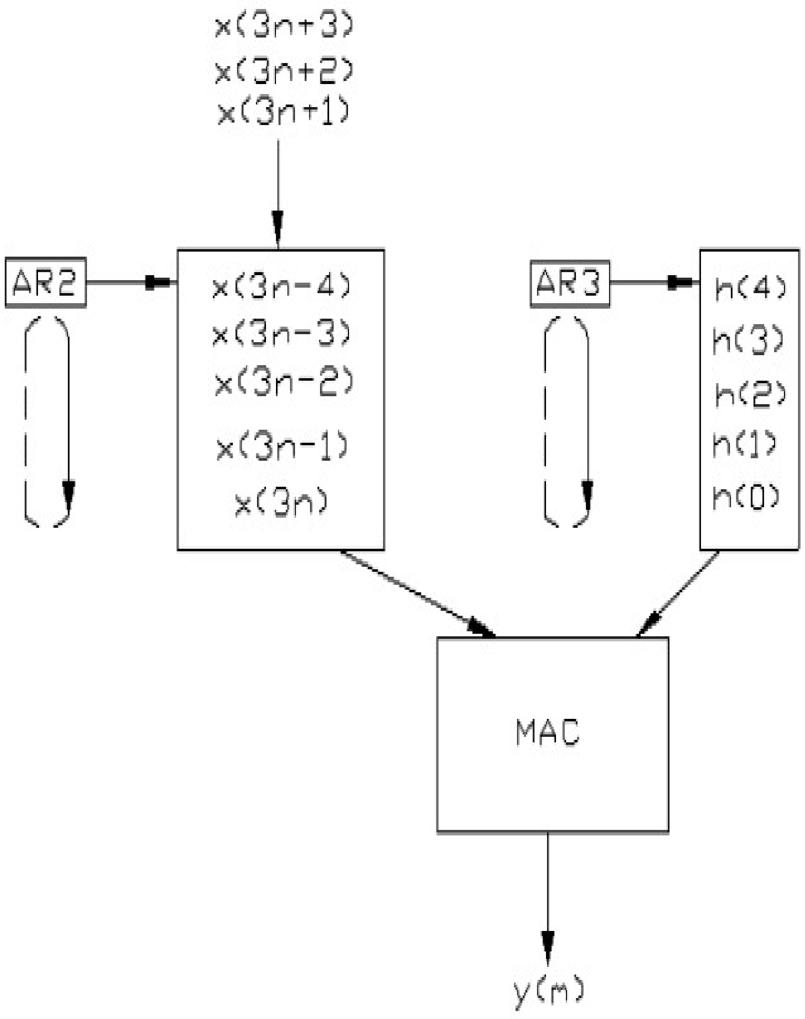


Figure 5.9: Implementation of decimation filter

Implementation of decimation filter

It implements the following equation:
y(m) = h(4)x(3n-4) + h(3)x(3n-3) + h(2)x(3n-2) + h(1)x(3n-1) + h(0)x(3n) followed by the equation
y(m+1) = h(4)x(3n-1) + h(3)x(3n) + h(2)x(3n+1) + h(1)x(3n+2) + h(0)x(3n+3)
and so on for a decimation factor of 3 and a filter length of 5.

```
                  .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
```

FIR Filter Routine

Enter with A = x(n), AR2 pointing to the circular sample buffer, and AR3 to the circular coeff buffer. AR0 = 1.

Exit with A = y (n) as q15 number.

```
dec_filter :
          LD *AR5+,A                       ; Place next 3 input samples
          STL A, *AR2+0%             ; into the signal buffer
          LD *AR5+,A
          STL A, *AR2+0%
          LD *AR5+,A
          STL A, *AR2+0%
          RPTZ A, #Nm1               ; A = 0
          MAC *AR3+0%,*AR2+0%,A         ; A = filtered signal
          RET                             ; Return
          nop
          nop
          nop
        .end
```

Problems:

1. What values are represented by the 16-bit fixed point number N=4000h in Q15 & Q7 notations?

Solution:

Q15 notation: **0.100 0000 0000 0000 (N=0.5)**

Q7 notation: **0100 0000 0.000 0000 (N=+128)**

ASSIGNMENT NO:5 (Recommended Questions)

1. Determine the value of each of the following 16- bit numbers represented using the given Q- notations:
   (i) 4400h as a Q0 number
   (ii) 4400h as a Q7 number
   (iii) 4400h as a Q15 number
   (iv) 4400h as a Q1 number
   (v) 0.3125 as a Q15 number
   (vi) - 0.3125 as a Q15 number. **(DEC,2012.4m)**
2. Write an assembly language program for TMS320C54XX processors to multiply two Q15 numbers to produce Q15 number result. **(Dec 12 , 6 marks)(July 11, 6m)(June/July2012, 4m)**
3. What is an interpolation filter? Explain the implementation of digital interpolation using FIR filter and poly phase sub filter.**(Dec 12, 8 marks.DEC 2015,08M)**
4. Describe the importance of Q-notation in DSP algorithm implementation with examples. What are the values represented by 16- bit fixed point number N=4000h in Q15, Q9, Q7 notations? **(DEC20 11, 6m, june,2014.10M)**
5. Explain how the FIR filter algorithms can be implemented using TMS320c54xx processor.**(DEC 2012,6m.MAY-JUNE 10, 10marks. June 2014,10m)**
6. Explain with the help of a block diagram and mathematical equations the implementation of a second order IIR filter. No program code is required.**(June/July2011, 11m)**
7. Determine the value of each of the following numbers represented using the given Q- notations:
   i) -0.1958 as a Q15 number
   ii) 136 as a Q7 number
   iii) D0B5H as a Q15 number
   iv) 4400H as a Q7 number **(June/July2012, 4m)**
8. With the help odd block diagram explain the implement of an FIR filter in TMS320C54XX processor. Show the memory organization for the filter implementation.**(June/July2012, 12m)**
9. How do you obtain the product of two Q15 number in Q15 representation. **(June/July2012, 2m.DEC 2015.06M.)**
10. **What** is an interpolation filter? Explain the implementation of digital interpolation using FIR filter and poly phase subfilter. Write the program. [DEC 2015,08M]
11. What is the drawback of using linear interpolation for implementing of an FIR filter in TMS320C54XX processor? Expain a scheme that overcomes this drawback. **(DEC 2012, 6m.june 2015,06M.)**
12. Briefly explain IIR filters. **(DEC 2011, 4m)**

13. What are the values represented by 16- bit fixed point number N=2000h in Q15, Q10, Q7 notations? **(MAY-JUNE 10, 9m)**
14. **Write a** TMS320C54XX processor to implement an FIR filter of length 15 and interpolation factor 5. **[DEC 2011,8M]**
15. Define Q Notation. Explain Q7 and Q15 Notation with example.[june 2013,5M]
16. Realize and write a program for a second order IIR filter on TMS320C54XX **[june 2013,10M]**
17. Explain with necessary block diagram, memory oraganisation for implementating FIR filter of order N**. [june 2013,5M]**
18. Brifly explain IIR filters. With the help of block diagram,explain second order IIR filter. **[DEC 2013,06M]**
19. **Write a** TMS320C54XX processor that illustrates the implementation of an interpolating FIR filter of length 15 and interpolating factor 5. **[DEC 2013,06M]**
20. What are the values represented by 16- bit fixed point number N=4000h in Q5, Q10, Q7 notations**? (DEC20 11, 6m)**
21. What is the significance of Q notation in DSP? **[DEC,2014.04M]**
22. Explain with the help of block diagram and mathematical implementation of decimation filter on TMS320C54XX processor**.[DEC 2014,10M]**
23. Repeat both N1=0.5 and N2=0.25 in Q15 notation and write assembly language program for TMS320C54XX processor to multiply obtained Q15 number**.[june 2015.06M]**
24. Show the memory organization for digital interpolation 15 tap FIR filter with interpolation factor 5**.[june,2015.04M]**
25. Determine the value of each of the following 16- bit numbers represented using the given Q- notations:
    i) 4400h as a Q7 number
    ii) 4400h as a Q0 number
    iii) 2ccch as a Q15 number
    iv) E6EFh as a Q 15 number **[june,2015.04M]**
26. Represent each of the following in desired Quotation format
    i) -352 as Q0 number
    ii) 3.125 as Q7 number
    iii) BDAFG h in Q7 and Q15 number
    iv) -0.160123 as Q15 number **(DEC2014, 6m)**
27. What do you meant by Q notation used In DSP algorithm implementation? Determine the value of each of the following 16- bit numbers represented using the given Q- notations:
    i) 4400h as a Q7 number
    ii) 4400h as a Q15 number **[DEC 2013,06M]**

IMPLEMENTATION OF FFT ALGORITHMS

❖ LEARNING OBJECTIVES

- Introduction,
- FFT algorithm for DFT computations
- Butterfly computation
- Overflow & scaling
- Bit reversed index generation
- 8 point FFT implementation on TMS320C5XX
- Computation of the signal spectrum

6.1 Introduction:

The N point Discrete Fourier Transform (DFT) of x(n) is a discrete signal of length N is given by eq(6.1)

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad ; \qquad k = 0 \ldots N-1 \qquad (6.1)$$

$W_N^{kn} = e^{-j2\pi kn/N}$ is the twiddle factor

By referring to eq (6.1) and eq (6.2), the difference between DFT & IDFT are seen to be the sign of the argument for the exponent and multiplication factor, 1/N. The computational complexity in computing DFT / I DFT is thus same (except for the additional multiplication factor in IDFT). The computational complexity in computing each X(k) and all the x(k) is shown in table 6.1.

The Inverse DFT (IDFT) is given by eq(2)

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}; \quad n = 0 \ldots N-1 \qquad (6.2)$$

In a typical Signal Processing System, shown in fig 6.1 signal is processed using DSP in the DFT domain. After processing, IDFT is taken to get the signal in its original domain. Though certain

Table 6.1: computational complexity in DFT/ IDFT		
Computation of each term, X(k) or x(n)	N complex number multiplications	N-1 complex number additions
Computation of all the terms X(k) or x(n)	N2 complex number multiplications	N(N-1) complex number additions
Complexity is of the order of N^2		

amount of time is required for forward and inverse transform, it is because of the advantages of transformed domain manipulation, the signal processing is carried out in DFT domain. The transformed domain manipulations are sometimes simpler. They are also more useful and powerful than time domain manipulation. For example, convolution in time domain requires one of the signals to be folded, shifted and multiplied by another signal, cumulatively. Instead, when the signals to be convolved are transformed to DFT domain, the two DFT are multiplied and inverse transform is taken. Thus, it simplifies the process of convolution.

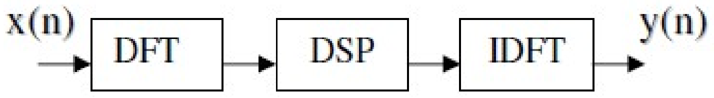


Fig 6.1: DSP System

6.2 An FFT Algorithm for DFT Computation:

As DFT / IDFT are part of signal processing system, there is a need for fast computation of DFT / IDFT. There are algorithms available for fast computation of DFT/ IDFT. There are referred to as Fast Fourier Transform (FFT) algorithms. There are two FFT algorithms: Decimation-In-Time FFT (DITFFT) and Decimation-In-Frequency FFT (DIFFFT). The computational complexity of both the algorithms are of the order of log2(N). From the hardware / software implementation viewpoint the algorithms have similar structure throughout the computation. In-place computation is possible reducing the requirement of large memory locations. The features of FFT are tabulated in the table 6.2.

Table 6.2: Features of FFT

Features	DITFFT	DIFFFT
Sequence which is decimated by factor 2	Time domain sequence	DFT sequence
Input sequence	Bit reversed order	Proper order
Output sequence	Proper order	Bit reversed order

6.2.1 Two point DFT Computation:

Consider an example of computation of 2 point DFT. The signal flow graph of 2 point DITFFT Computation is shown in fig 6.2. The input / output relations is as in eq (6.3) which are arrived at from eq(6.1)

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad ; \qquad k = 0 \ldots N-1 \qquad (6.1)$$

$W_N^{kn} = e^{-j2\pi kn/N}$ is the twiddle factor

$$\begin{aligned} X(0) &= x(0)W_2^0 + x(1)W_2^0 = x(0) + x(1) \\ X(1) &= x(0)W_2^0 + x(1)W_2^1 = x(0) - x(1) \end{aligned} \qquad (6.3)$$

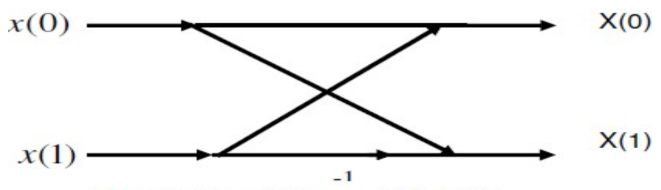
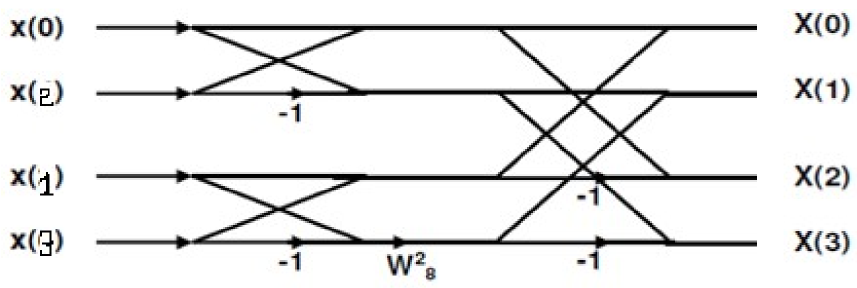


Fig 6.2: Signal Flow graph for N=2

Similarly, the Butterfly structure in general for DITFFT algorithm is shown in fig. 6.3.

6.2.2 FOUR POINT DFT COMPUTATION:



- We require total of 4 butterflies in two stages of computation
- The first stage has 2 butterflies one operating on x(0) other on x(2) and other butterflies one operating on x(1), x(3).
- In second stage the first butterfly operates on upper output of the first stage butterflies and other operates on then lower output of the first stage butterflies.
- The lower output of the second butterfly of the first stage needs to be multiplied with twiddle factor W_4.

- The i/p sample x(0) through x(3) are required to rearrange in the order x(0), x(2), x(1), x(3).
- The natural occurring i/p sample indices 0,1,2,3 are represented by their binary equivalent 00,01,10,11 and their binary number are reversed we get 00,10,01,11 and their binary numbers.
- This process of rearrangement of indices for DFT computation is called BIT REVERSING.

6.2.3 **EIGHT POINT DFT COMPUTATION:**

The signal flow graph for N=8 point DITFFT is shown in fig. 6.4. The relation between input and output of any Butterfly structure is shown in eq (6.4) and eq(6.5).

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad ; \qquad k = 0 \ldots N-1 \qquad (6.1)$$

$W_N^{kn} = e^{-j2\pi kn/N}$ is the twiddle factor

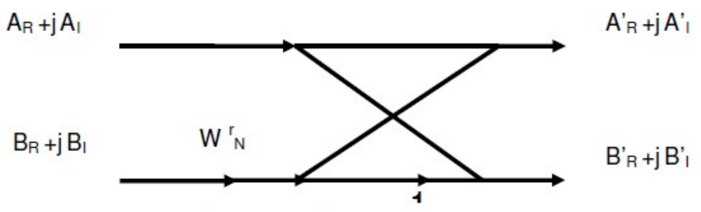


Fig 6. 3: Butterfly structure for N point DITFFT Computation

$$A'_R + jA'_I = A_R + jA_I + (B_R + jB_I)(W_R^r + jW_I^r) \qquad (6.4)$$
$$B'_R + jB'_I = A_R + jA_I - (B_R + jB_I)(W_R^r + jW_I^r) \qquad (6.5)$$

Separating the real and imaginary parts, the four equations to be realized in implementation of DITFFT Butterfly structure are as in eq(6.6).

$$\left\{\begin{aligned} A'_R &= A_R + B_R W_R^r - B_I W_I^r \\ A'_I &= A_I + B_I W_R^r + B_R W_I^r \\ B'_R &= A_R - B_R W_R^r + B_I W_I^r \\ B'_I &= A_I - B_I W_R^r - B_R W_I^r \end{aligned}\right\} \qquad (6.6)$$

Observe that with N=2^M, the number of stages in signal flow graph=M, number of multiplications = (N/2)log2(N) and number of additions = (N/2)log2(N). Number of Butterfly Structures per stage = N/2. They are identical and hence in-place computation is possible. Also reusability of hardware designed for implementing Butterfly structure is possible. However in case FFT is to be computed for a input sequence of length other than 2^M the sequence is extended to N=2^M by appending additional zeros. The process will not alter the information content of the signal. It improves frequency resolution. To make the point clear, consider a sequence whose spectrum is shown in fig. 6.5.

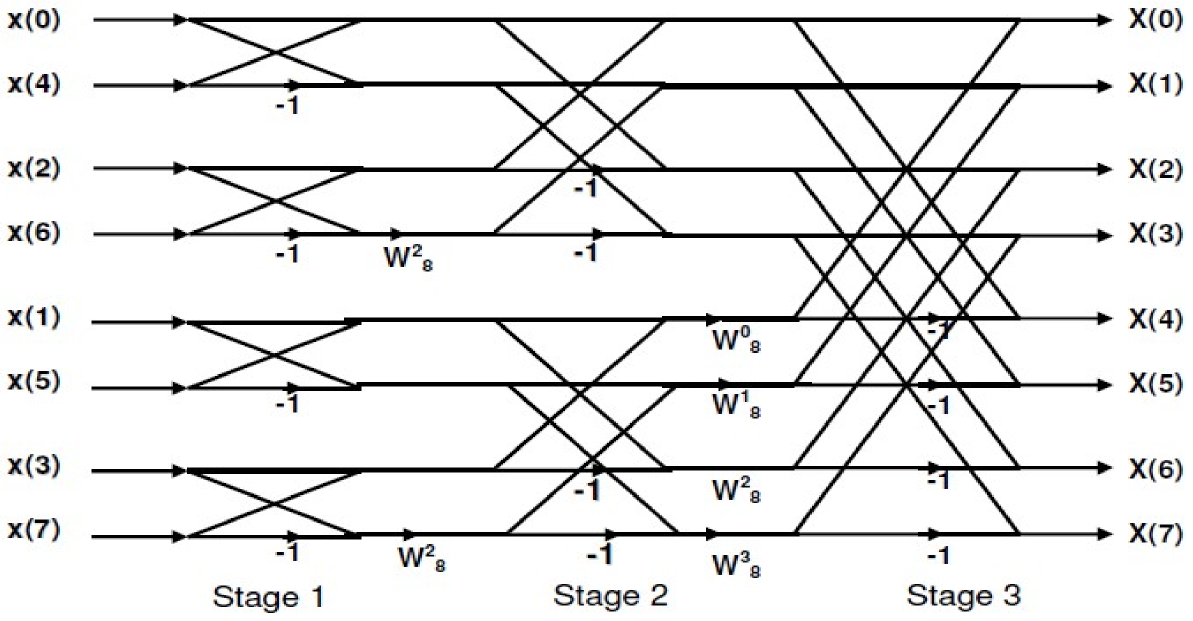


Fig 6.4. Signal flow graph of 8 point DITFFT Computation

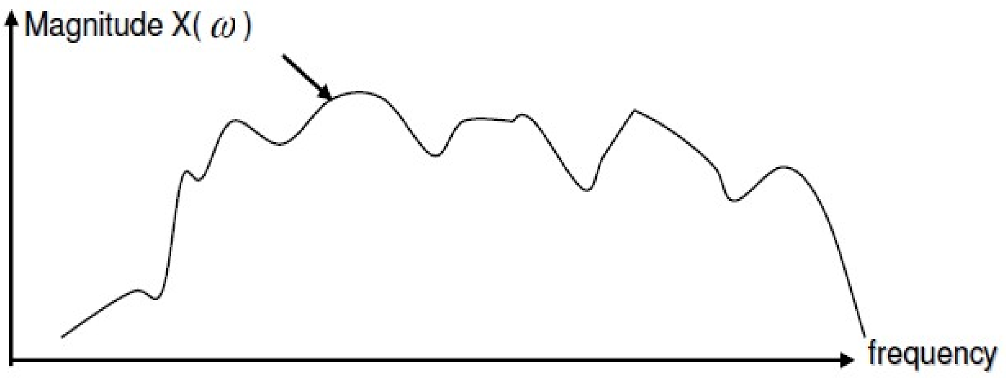


Fig 6.5: Spectrum of x(n)

The spectrum is sampled to get DFT with only N=10. The same is shown in fig 6. The variations in the spectrum are not traced or caught by the DFT with N=10. For example, dip in the spectrum near sample no. 2, between sample no.7 & 8 are not represented in DFT. By increasing N=16, the DFT plot is shown in fig. 6.7. As depicted in fig 6.7, the approximation to the spectrum with N=16 is better than with N=10. Thus, increasing N to a suitable value as required by an algorithm improves frequency resolution.

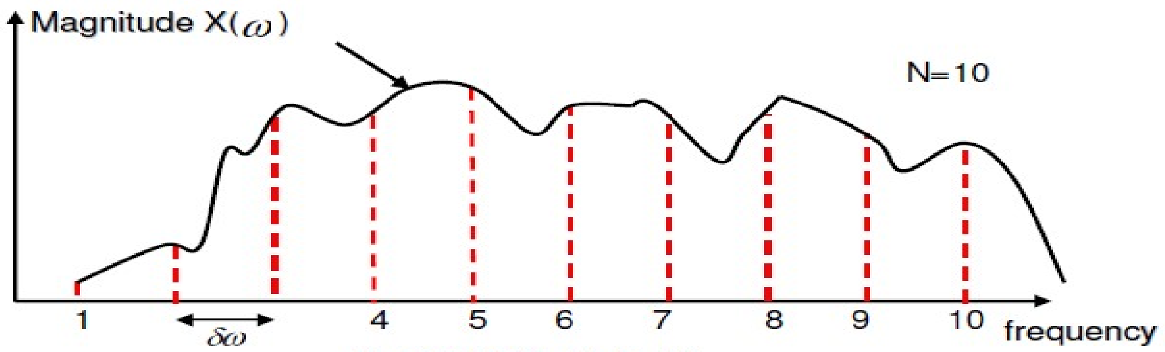


Fig 6.6: DFT with N=10

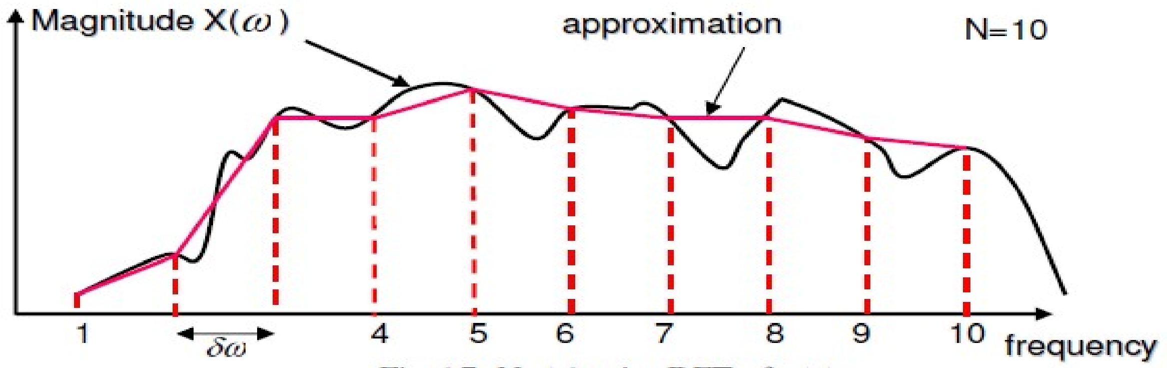


Fig 6.7: N=16 point DFT of x(n)

N= 2^M Point FFT COMPUTATION:

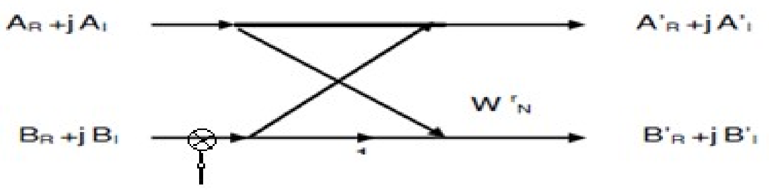
- **DFT computation can be extended to N points where N is power of 2 yields $\log_2 N$ stages of computation with each stage requiring N/2 butterflies.**
- **This computation structure is the Fast Fourier transform.**

ZERO PADING:

- **At time, the sequence to be transformed is appended with zero before computing the DFT. This can be done to satisfy the condition that the FFT algorithm requires that the number of points to be a power of 2.**
- **Another objective of zero padding is to increase the transformed points to decrease the frequency intervals between adjacent point represented by the X(K) sequence.**

A BUTTERFLY COMPUTATION:

Butterfly structure for DIFFFT: The input / output relations are



$$A'_R + jA'_I = A_R + jA_I + B_R + jB_I$$
$$B'_R + jB'_I = (A_R + jA_I - (B_R + jB_I))(W^r_R + jW^r_I)$$

Separating the real and imaginary parts,

$$\therefore\ A'_R = A_R + B_R \quad \& \quad A'_I = A_I + B_I$$
$$B'_R = (A_R - B_R)W^r_R - (A_I - B_I)W^r_I$$
$$B'_I = (A_R - B_R)W^r_I + (A_I - B_I)W^r_R$$

How many add/subtract and multiply operations are needed to implement a general butterfly of DITFFT?

Solution: Referring to 4 equations required in implementing DITFFT Butterfly structure, ADD/SUBSTRACT operations are 06 and Multiplication operations are 04

Derive the optimum scaling factor for the DIFFFT Butterfly structure.

Solution: The four equations of Butterfly structure are Differentiating 4th relation and setting it to zero, (any equation may be considered)

$$\frac{\partial B'_I}{\partial \theta} = (A_R - B_R)\cos\theta - (A_I - B_I)\sin\theta = 0$$
$$\Rightarrow (A_R - B_R)\cos\theta = (A_I - B_I)\sin\theta$$
$$\therefore\ \tan\theta = \frac{A_R - B_R}{A_I - B_I} \qquad \text{P6.5.2}$$

$$A'_R = A_R + B_R$$
$$A'_I = A_I + B_I$$
$$B'_R = (A_R - B_R)W^r_R - (A_I - B_I)W^r_I$$
$$B'_I = (A_R - B_R)W^r_I + (A_I - B_I)W^r_R \qquad \text{P6.5.1}$$

$$\sin\theta = \frac{(A_R - B_R)}{\sqrt{(A_R - B_R)^2 + (A_I - B_I)^2}}$$
$$\& \quad \cos\theta = \frac{(A_I - B_I)}{\sqrt{(A_R - B_R)^2 + (A_I - B_I)^2}}$$

$$\therefore\ B'_{I.\max} = \sqrt{(A_R - B_R) + (A_I - B_I)}$$
$$= \sqrt{2} \qquad \text{P6.5.3}$$

Thus scaling factor is 0.707. To achieve multiplication by right shift, it is chosen as 0.

PROBLEM 6.1: Determine the following for a 128-point FFT computation:

a) **Number of stages**
b) **Number of butterflies in each stage**
c) **Number of butterflies needed for the entire computation**
d)

SOLUTION:

i) **No of butterflies in each stage N/2=128/2=64**

ii) **Number of stages = $\log_2 N = \log_2 64 = 7$**

iii) **No of butterflies for entire computation**

= no of stages × no of butterflies in each stage

$= \log_2 N \times N/2$

$= 7 \times 64$

$= 448$

PROBLEM NO 6.2: What minimum size FFT must be used to compute a DFT of 220 points in radix 2 algorithm? Determine the number of complex multiplication and addition needed?

SOLUTION: Minimum size = 256

a) **No of stages** $= \log_2 N = \log_2 220 = 7.78 \Rightarrow 8$

$\log_2 N = 8$

i.e $N = 2^8$

$N = 256$

b) **Number of butterfly structure = number of stages × N/2**

$= \log_2 256 \times 256/2$

$= 1024$

$= 1024$

c) **Number of butterfly complex multiplication = 1024**

d) **Number of butterfly complex addition = 2 × 1024 = 2048**

PROBLEM NO 6.3: Determine the following for a 512-point FFT computation:

- a) **Number of stages**
- b) **Number of butterflies in each stage**
- c) **Number of butterflies needed for the entire computation**
- d) **Number of butterflies that need no twiddle factors**
- e) **Number of butterflies that require real twiddle factors**
- f) **Number of butterflies that require complex twiddle factors**

SOLUTION:

a) **Number of stages = $\log_2 N = \log_2 512 = 9$**

b) **Number of butterflies in each stage = N/2=512/2=256**

c) **Number of butterflies needed for the entire computation**

= no of stages × no of butterflies in each stage

$=\log_2 N \times N/2$

$= 9 \times 256$

$= 2304$

d) **Number of butterflies that need no twiddle factors =N-1=512-1=511**

e) **Number of butterflies that require real twiddle factors =N-1=512-1=511**

f) **Number of butterflies that require complex twiddle factors=2304-511=1793**

PROBLEM 6.4:

- i) **What minimum size FFT must be used to compute DFT of 40 samples**
- ii) **How many stages are required for FFT computation?**
- iii) **How many butterflies there per stage**
- iv) **How many butterflies are needed for the entire computation**

SOLUTION:

i) No of stages $= \log_2 N = \log_2 40 = 5.32 \Rightarrow 6$

ii) $\log_2 N = 6$

i.e $N = 2^6$

no of samples $N = 64$

iii) Butterfly per stage = N/2 =64/2=32

iv) Butterfly need for entire computation = no of stages × no of butterflies in each stage

$= 6 \times 32$

=192

PROBLEM 6.5: What minimum size FFT must be used to compute a DFT of 40 points? What must be done to samples before the chosen FFT is applied?

Solution:

Minimum size FFT for a 40 point sequence is 64 point FFT. Sequence is extended to 64 by appending additional 24 zeros. The process improves frequency resolution from

a) **No of stages** **= $\log_2$ N= $\log_2$ 40 =5.32=>6**

$\log_2$ N=6

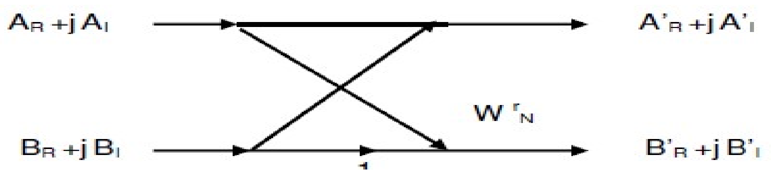
i.e N=2^6

N =64

Additional zeros = 64-40 =24

Zero padding must be done to the sample before the chosen FFT is applied.

$\delta\omega = 2\pi/40$ to $\delta\omega = 2\pi/64$ (P6.1)



PROBLEM 6.6: What minimum size FFT must be used to compute a DFT of 500 points? What must be done to samples before the chosen FFT is applied?

Solution:

Minimum size FFT for a 40 point sequence is 64 point FFT. Sequence is extended to 64 by appending additional 24 zeros. The process improves frequency resolution from

a) **No of stages** **= $\log_2$ N= $\log_2$ 500 = 8.9 => 9**

$\log_2$ N=9

i.e N=2^9

N =512

Additional zeros = 512-500 =12 must be added.

Zero padding must be done to the sample before the chosen FFT is applied.

PROBLEM NO 6.7: A time domain sequence of 73 elements is to be convolved with time domain sequence of 50 elements using DFT to transform the two sequences multiplying them and then doing the IDFT to obtain the resulting time domain sequence to implement DFT or IDFT, the DIT FFT algorithm is to be used. Determine the total number of complex multiplication needed to implement the conditions. Assume that each butterfly computation requires one complex multiplication.

SOLUTION: **X(1) be the length of =73**

X(2) be the length of =50

Length of convolved sequence = 73 + 50 = 122

Length of DFT or IDFT = 2^n= 128

Two DFT and one IDFT each of length 128 are to be determined

No of butterfly structure per stage = N/2=128/2=64

No of stages = $\log_2 N = \log_2 128 = 7$

Total no of complex multiplication =64×7×3=1344

PROBLEM NO 6.3: Determine the following for a 1024-point FFT computation:

g) **Number of stages**
h) **Number of butterflies in each stage**
i) **Number of butterflies needed for the entire computation**
j) **Number of butterflies that need no twiddle factors**
k) **Number of butterflies that require real twiddle factors**
l) **Number of butterflies that require complex twiddle factors**

SOLUTION:

i) **Number of stages = $\log_2 N = \log_2 1024 = 10$**

ii) **Number of butterflies in each stage = N/2=1024/2=512**

iii) **Number of butterflies needed for the entire computation**

= no of stages × no of butterflies in each stage

$=\log_2 N \times N/2$

= 10 × 512

= 5120

iv) **Number of butterflies that need no twiddle factors =N-1=1024-1=1023**

v) **Number of butterflies that require real twiddle factors =N-1=1024-1=1023**

vi) **Number of butterflies that require complex twiddle factors=5120-1023=4097**

6.3 Overflow and Scaling:

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,

$$\begin{aligned} A_I' &= A_I + B_I W_R^r + B_R W_I^r \\ &= A_I + B_I \cos\theta + B_R \sin\theta \qquad (6.7) \\ &\text{where } \theta = 2\pi kn/N \end{aligned}$$

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

$$\begin{aligned} \frac{\partial A_I}{\partial \theta} &= -B_I \sin\theta + B_R \cos\theta = 0 \\ &\Rightarrow B_I \sin\theta = B_R \cos\theta \qquad (6.8) \\ &\Rightarrow \tan\theta = B_R / B_I \end{aligned}$$

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

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.

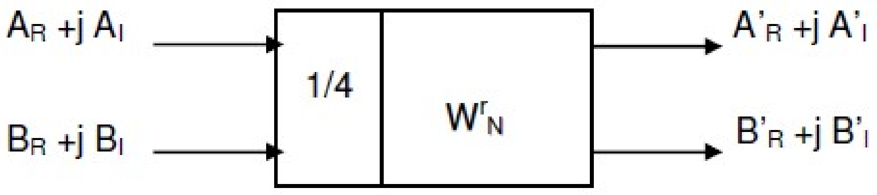


Fig 6.8: Symbolic representation of Butterfly structure with scaling factor

6.4 Bit-Reversed Index Generation:

As noted in table 6.2, 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 ndex 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

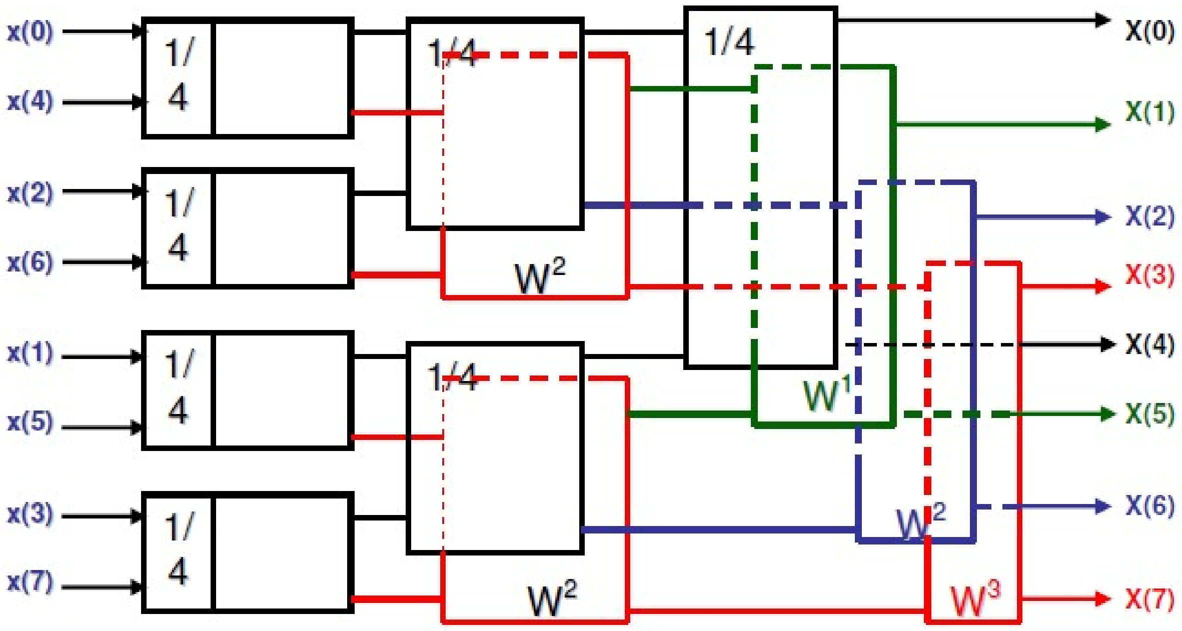


Fig 6.9: Signal flow graph with Scaling

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.

6.5 Implementation of FFT on TMS32OC54xx:

The main program flow for the implementation of DITFFT is shown in fig. 6.10. The subroutines used are _clear to clear all the memory locations reserved for the results. _bitrev stores the data sequence x (n) in bit reverse order. _butterfly computes the four equations of computing real and imaginary parts of butterfly structure. _spectrum computes the spectrum of x (n). The Butterfly subroutine is invoked 12 times and the other subroutines are invoked only once.

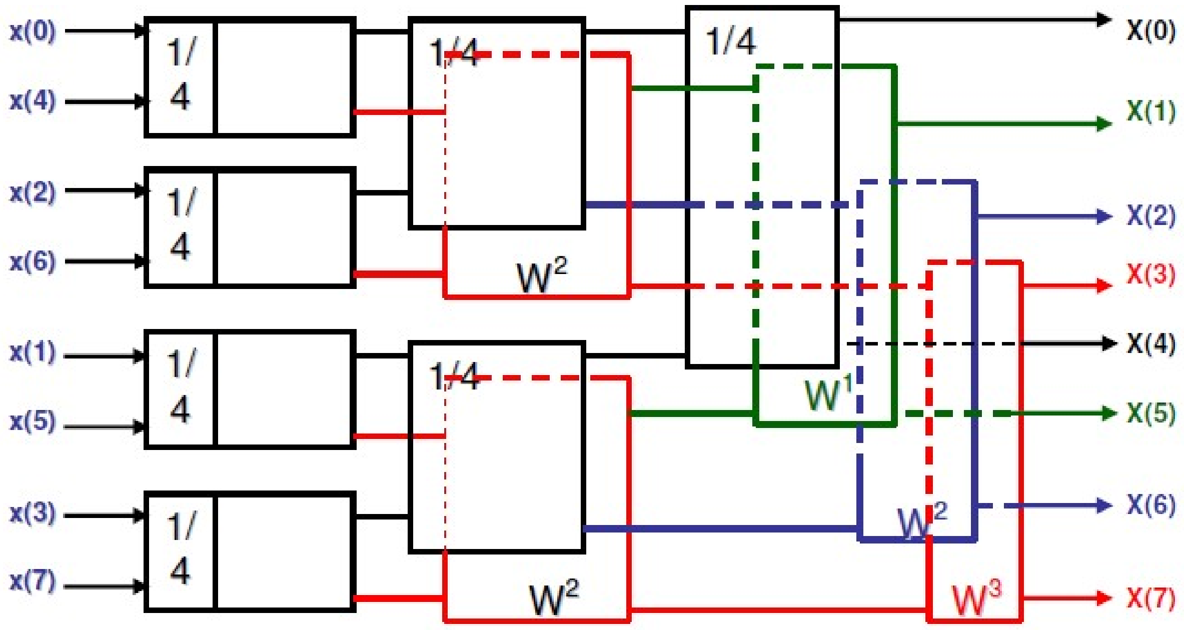


Fig 6.9: Signal flow graph with Scaling

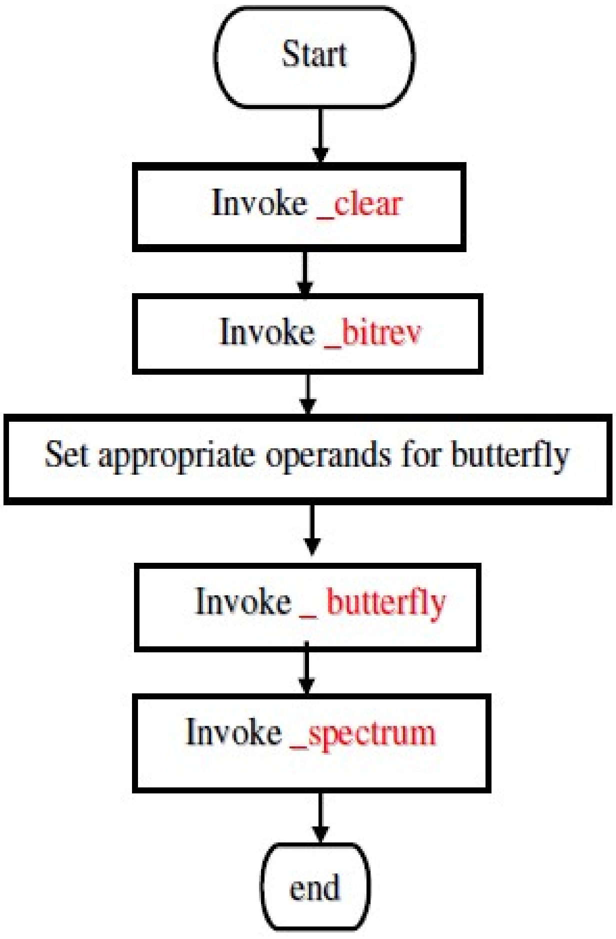


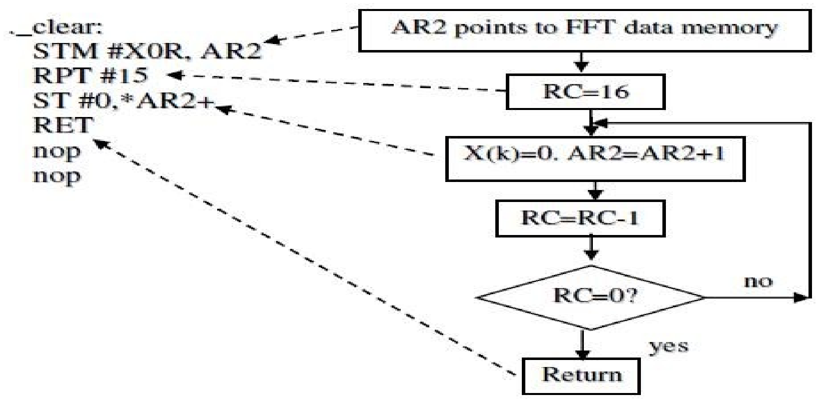
Fig. 6.10: Main Program Flow

The program is as follows

```
.mmregs
.def _c_int00
.data
; Reserve 8 locations for x(n)
;x(n)                    Q15 notation    decimal value
```

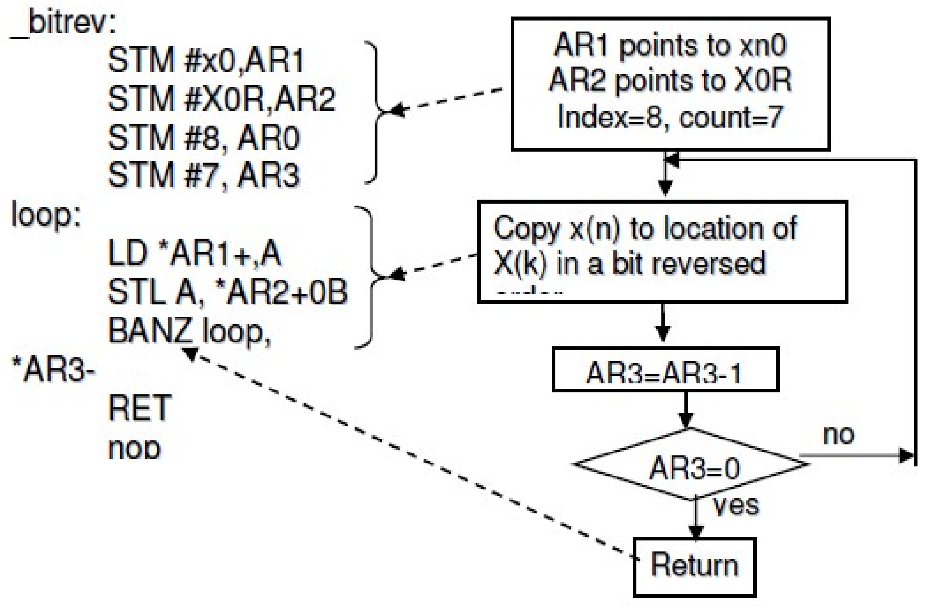
```
xn0     .word  0                ; 0h                 0.0
xn1     . word 16384            ; 4000h              0.5
xn2     .word 23170             ; 5A82h              0.707
xn3     . word - 24576          ; E000h             -0.25
xn4     .word 12345     ; 3039h             0.3767
xn5     .word 30000             ; 7530h              0.9155
xn6     .word 10940     ; 2ABCh             0.334
xn7     .word 12345             ; 3039h              0.3767

; Reserve 16 locations for X(k)
X0R      .word 0                 ;real part of X(0) =0
X0Im     .word 0         ;imaginary part of X(0) =0
X1R     .word 0
X1Im    .word 0
X2R      .word 0
X2Im    .word 0
X3R     .word 0
X3Im    .word 0
X4R     .word 0
X4Im    .word 0
X5R     . word 0
X5Im    .word 0
X6R     .word 0
X6Im    .word 0
X7R     .word 0
X7Im    .word 0

; 8 locations for W08 to W38, twiddle factors
W08R          .word   32767         ;cos(0)=1
W08Im         .word 0               ;-sin(0)=0
W18R          .word   23170         ;cos(pi/4)= 0.707
W18Im         .word -23170          ;-sin(pi/4)= -0.707
W28R          .word   0             ;cos(pi/2)= 0
W28Im         .word -32767          ;-sin(pi/2)= -1
W38R          .word -23170          ;cos(3pi/4)= -0.707
W38Im         .word -23170          ;-sin(3pi/4)= -0.707

; 8 locations for Spectrum
S0      .word   0               ;Frequency content at 0
S1      .word   0               ;Frequency content at fs/8
S2      .word   0               ;Frequency content at 2fs/8
S3      .word   0               ;Frequency content at 3fs/8
S4      .word   0               ;Frequency content at 4fs/
S5      .word   0               ;Frequency content at 5fs/8
S6      .word   0               ;Frequency content at 6fs/8
S7      .word   0               ;Frequency content at 7fs/8


;temporary locations
TEMP1       .word 0
TEMP2       .word 0
;MAIN PROGRAM
. text
_c_int00:
        SSBX SXM    ; set sign extension mode bit of ST1
        CALL  _clear
        CALL  _bitrev
```

Clear subroutine is shown in fig. 6.11. Sixteen locations meant for final results are cleared. AR2 is used as pointer to the locations. Bit reverse subroutine is shown in fig. 6.12. Here, AR1 is used as pointer to x(n). AR2 is used as pointer to X(k) locations. AR0 is loaded with 8 and used in bit reverse addressing. Instead of N/2 =4, it is loaded with N=8 because each X(k) requires two locations, one for real part and the other for imaginary part. Thus, x(n) is stored in alternate locations, which are meant for real part of X(k). AR3 is used to keep track of number of transfers.



```
._clear:
  STM #X0R, AR2
  RPT #15
  ST #0,*AR2+
  RET
  nop
  nop
```

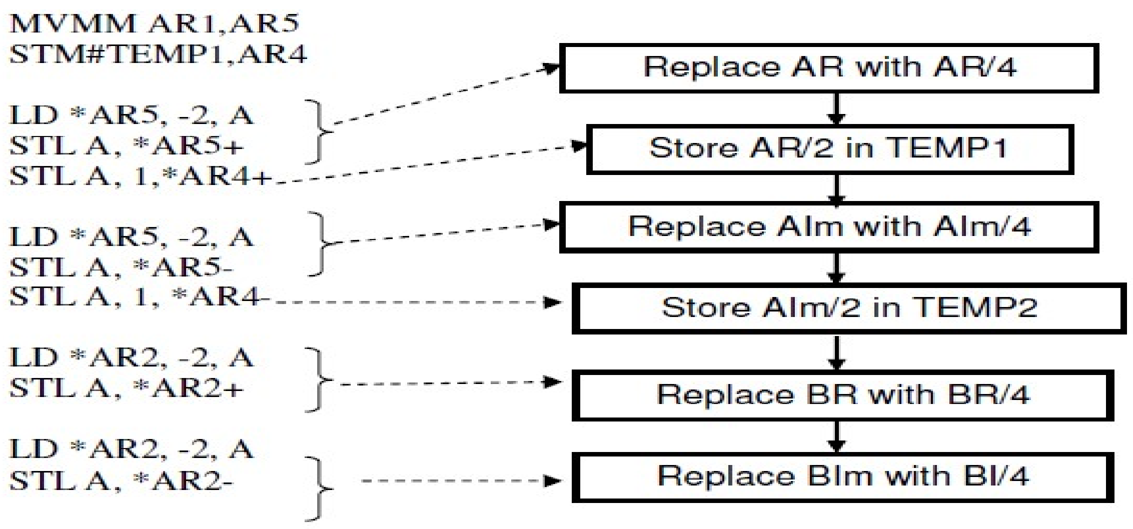
Fig. 6.11: Clear subroutine



```
_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
```

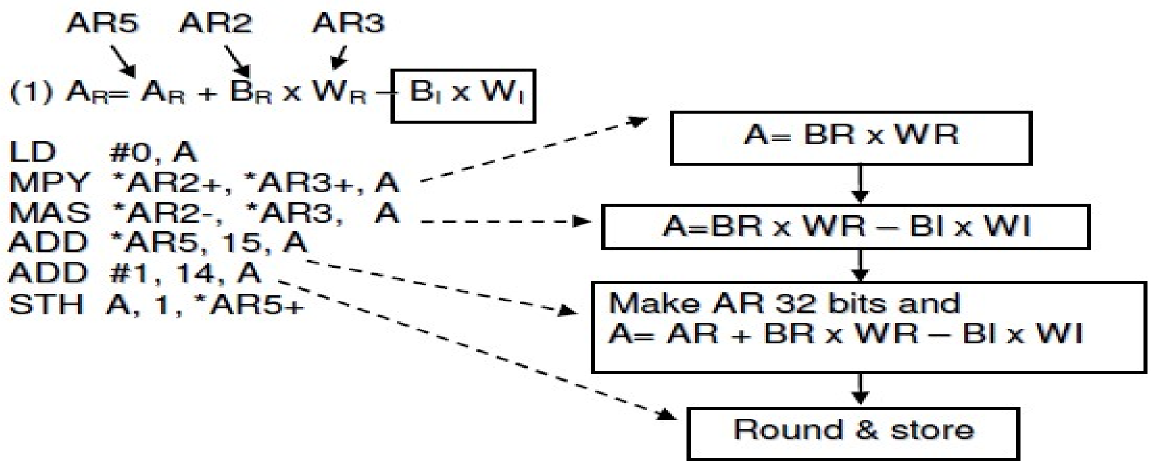
Fig. 6.12: Bit Reverse Subroutine

Butterfly subroutine is invoked 12 times. Part of the subroutine is shown in fig. 6.13. Real part and imaginary of A and B input data of butterfly structure is divided by 4 which is the scaling factor. Real part of A data which is divided by 2 is stored in temp location. It is used further in computation of eq (3) and eq (4) of butterfly. Division is carried out by shifting the data to the right by two places. AR5 points to real part of A input data, AR2 points to real part of B input data and AR3 points to real part of twiddle factor while invoking the butterfly subroutine. After all the four equations are computed, the pointers are in the same position as they were when the subroutine is invoked. Thus, the results are stored such that in-place computation is achieved. Fig. 6.14 through 6.17 show the butterfly subroutine for the computation of 4 equations.



```
MVMM AR1,AR5
STM#TEMP1,AR4

LD *AR5, -2, A
STL A, *AR5+
STL A, 1,*AR4+

LD *AR5, -2, A
STL A, *AR5-
STL A, 1, *AR4-

LD *AR2, -2, A
STL A, *AR2+

LD *AR2, -2, A
STL A, *AR2-
```

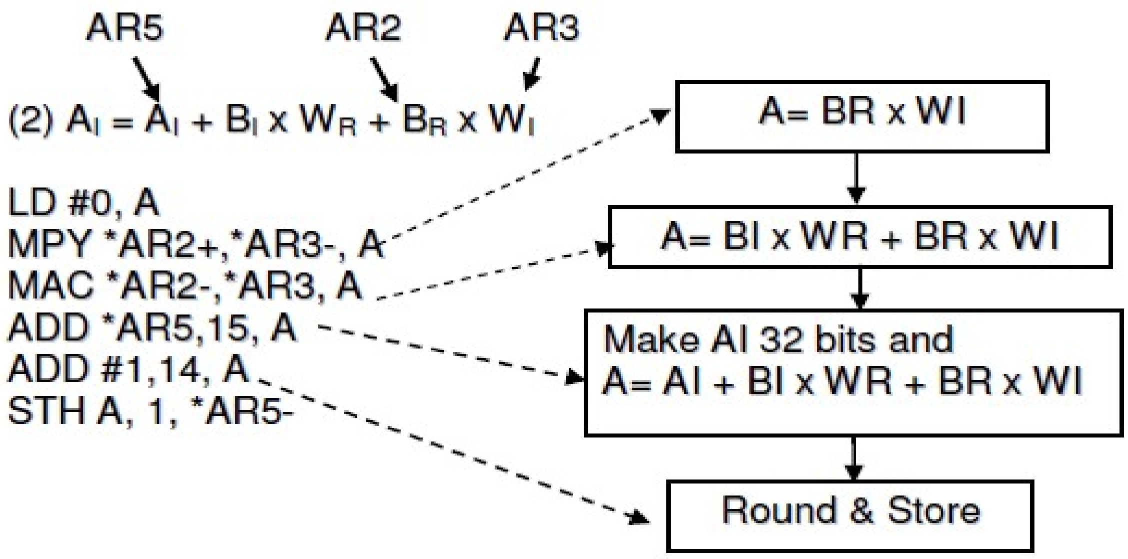
Fig. 6.13: Butterfly Subroutine



$$(1)\ A_R = A_R + B_R \times W_R - \boxed{B_I \times W_I}$$

```
LD    #0, A
MPY   *AR2+, *AR3+, A
MAS   *AR2-, *AR3,  A
ADD   *AR5, 15, A
ADD   #1, 14, A
STH   A, 1, *AR5+
```

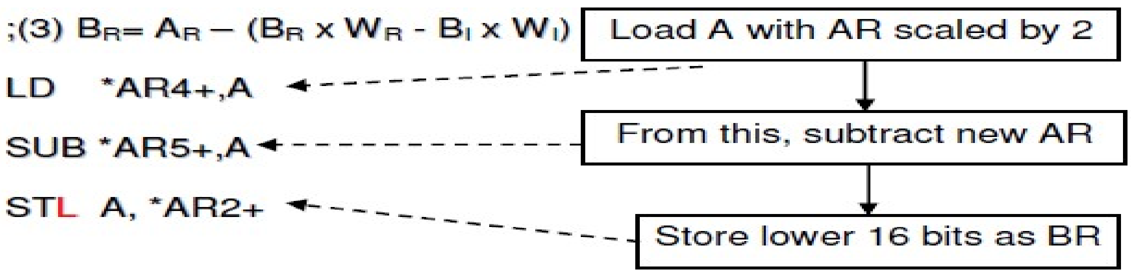
Fig. 6.14: Real part of A output of Butterfly



$$\overset{AR5}{\overset{\searrow}{}}$$ (2) $A_I = A_I + B_I \times W_R + B_R \times W_I$

```
LD #0, A
MPY *AR2+,*AR3-, A
MAC *AR2-,*AR3, A
ADD *AR5,15, A
ADD #1,14, A
STH A, 1, *AR5-
```

Fig. 6.15: Imaginary part of A output of Butterfly


Load A with AR scaled by 2
From this, subtract new AR
Store lower 16 bits as BR

;(3) $B_R = A_R - (B_R \times W_R - B_I \times W_I)$

```
LD    *AR4+,A
SUB *AR5+,A
STL   A, *AR2+
```

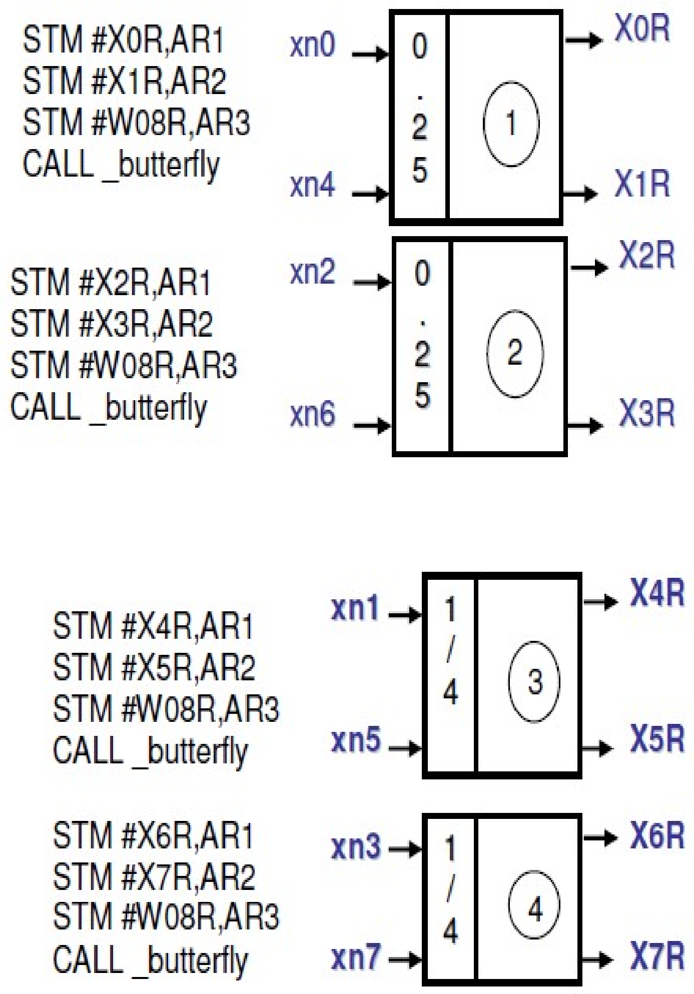
Fig. 6.16: Real part of B output of Butterfly

;(4) $B_I = A_I - (B_I \times W_R + B_R \times W_I)$

```
LD   *AR4 -, A
SUB *AR5-, A
STL  A, *AR2-

RET
nop
nop
```

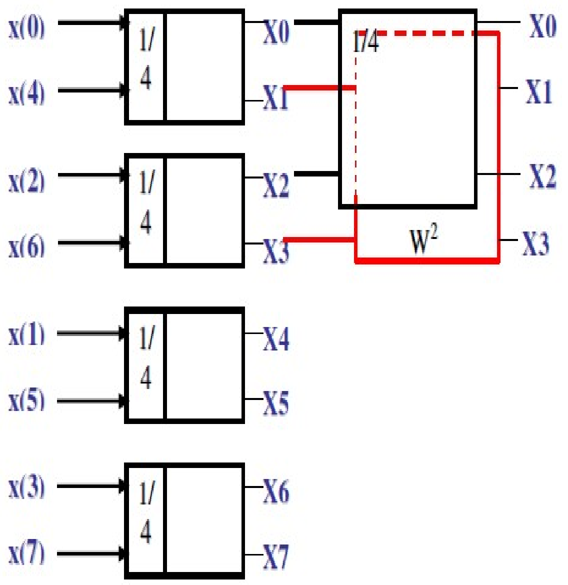
Fig. 6.17: Imaginary part of B output of Butterfly

Figure 6.18 depicts the part of the main program that invokes butterfly subroutine by supplying appropriate inputs, A and B to the subroutine. The associated butterfly structure is also shown for quick reference. Figures 6.19 and 6.20 depict the main program for the computation of 2nd and 3rd stage of butterfly.



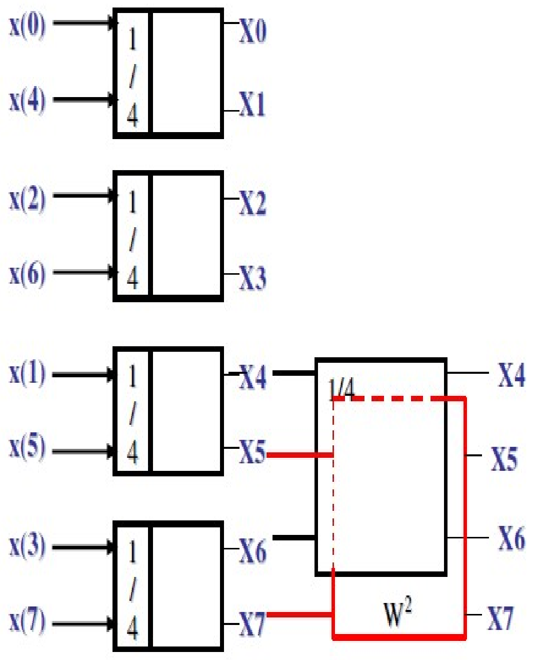
```
STM #X0R,AR1
STM #X1R,AR2
STM #W08R,AR3
CALL _butterfly

STM #X2R,AR1
STM #X3R,AR2
STM #W08R,AR3
CALL _butterfly

STM #X4R,AR1
STM #X5R,AR2
STM #W08R,AR3
CALL _butterfly

STM #X6R,AR1
STM #X7R,AR2
STM #W08R,AR3
CALL _butterfly
```

Fig. 6.18: First stage of Signal Flow graph of DITFFT



```
STM #X0R,AR1
STM #X2R,AR2
STM #W08R,AR3
CALL _butterfly
```

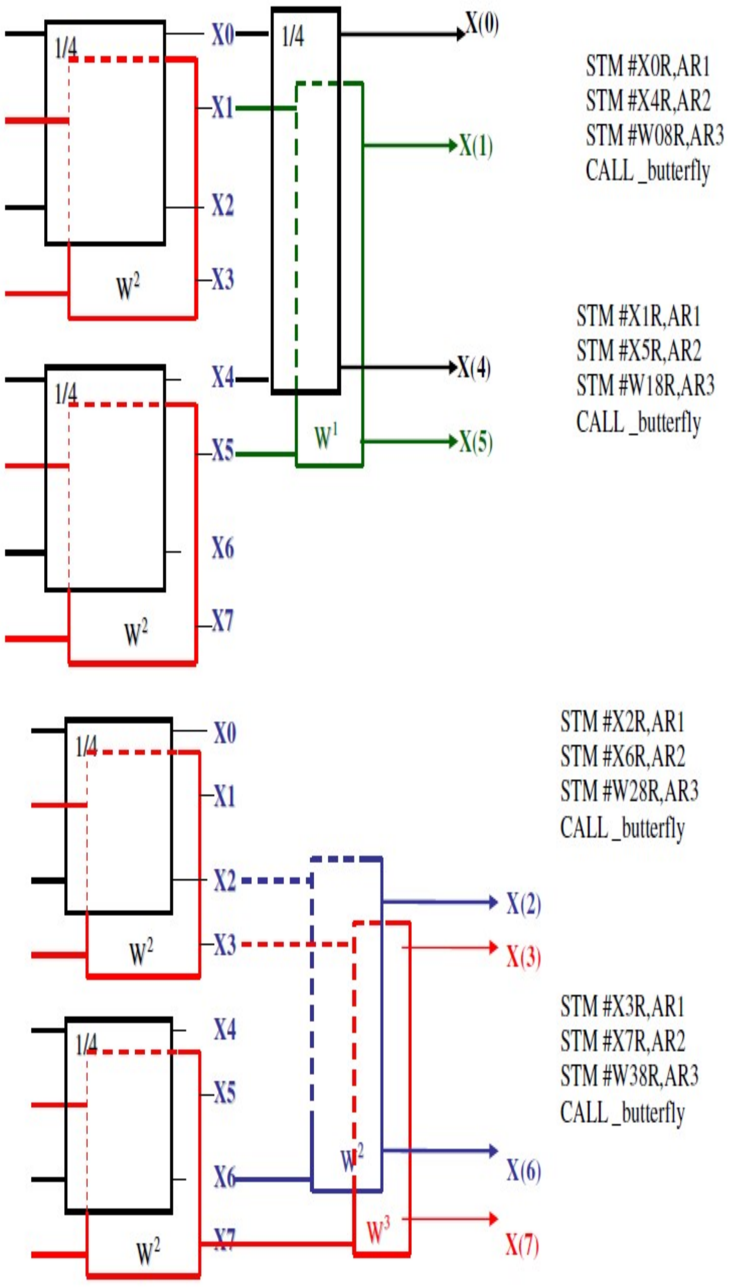
```
STM #X1R,AR1
STM #X3R,AR2
STM #W28R,AR3
CALL _butterfly
```



```
STM #X4R,AR1
STM #X6R,AR2
STM #W08R,AR3
CALL _butterfly
```

```
STM #X5R,AR1
STM #X7R,AR2
STM #W28R,AR3
CALL _butterfly
```

Fig. 6.19: Second stage of Signal Flow graph of DITFFT

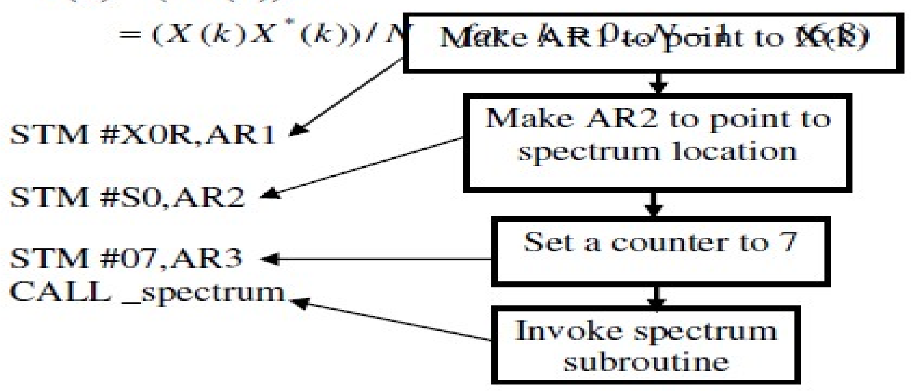


```
STM #X0R,AR1
STM #X4R,AR2
STM #W08R,AR3
CALL _butterfly

STM #X1R,AR1
STM #X5R,AR2
STM #W18R,AR3
CALL _butterfly

STM #X2R,AR1
STM #X6R,AR2
STM #W28R,AR3
CALL _butterfly

STM #X3R,AR1
STM #X7R,AR2
STM #W38R,AR3
CALL _butterfly
```

Fig. 6.20: Third stage of Signal Flow graph of DITFFT

After the computation of X(k), spectrum is computed using the equation(6.8). The pointer AR1 is made to point to X(k). AR2 is made to point to location meant for spectrum. AR3 is loaded with keeps track of number of computation to be performed. The initialization of the pointer registers before invoking the spectrum subroutine is shown in fig. 6.21. The subroutine is shown in fig. 6.22. In the subroutine, square of real and imaginary parts are computed and they are added. The result is converted to Q15 notation and stored.

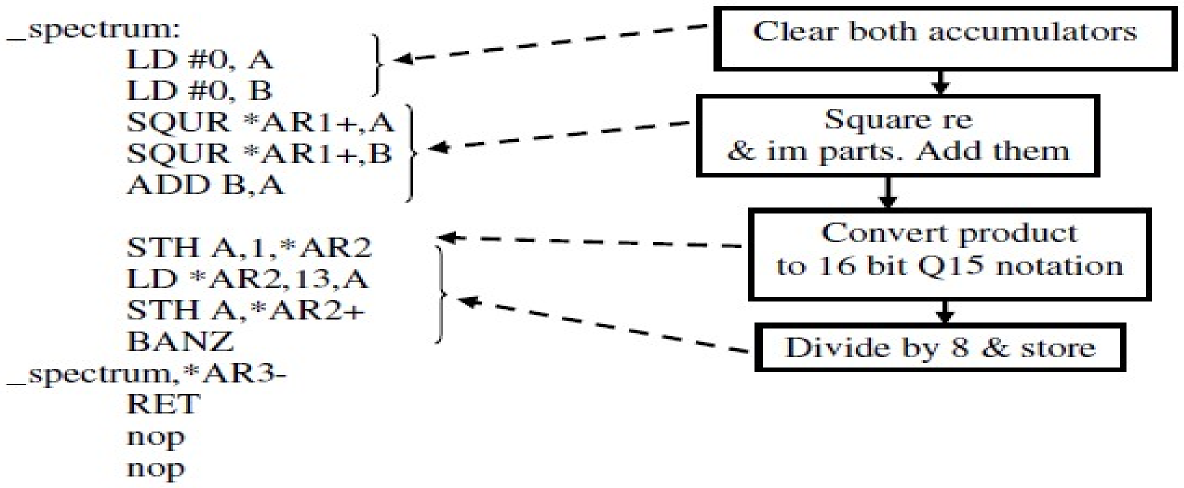
$$S(k) = (X^2(k))/N$$
$$= (X(k)X^*(k))/N$$



```
STM #X0R,AR1
STM #S0,AR2
STM #07,AR3
CALL _spectrum
```

Make AR1 to point to X(k)
Make AR2 to point to spectrum location
Set a counter to 7
Invoke spectrum subroutine

Fig. 6.21: Initialization for Spectrum Computation



```
_spectrum:
          LD #0, A
          LD #0, B
          SQUR *AR1+,A
          SQUR *AR1+,B
          ADD B,A

          STH A,1,*AR2
          LD *AR2,13,A
          STH A,*AR2+
          BANZ
_spectrum,*AR3-
          RET
          nop
          nop
```

Clear both accumulators
Square re & im parts. Add them
Convert product to 16 bit Q15 notation
Divide by 8 & store

Fig. 6.22: Subroutine for Spectrum Computation

Assignment no: 6 (Recommended Questions)

1. **What do you mean by bit-reversed index generation and how it is implemented in TMS320C54XX DSP assembly language?[june 2015,08M]**
2. **Write a subroutine program to find the spectrum of the transformed data using TMS320C54XX DSP.[june 2015,06M]**
3. **Explain and sketch a general DITFFT butterfly in place computation structure.[JUNE/JULY 2011,4M]**
4. **Determine the number of stages and number of butterflies in each stage and the total number of butterflies needed for the entire computation of 512 point FFT.[JUNE 2014,6M]**
5. **Explain how the bit reversed index generation can be done in 8 pt FFT. Also write a TMS320C54xx program for 8 pt DIT-FFT bit reversed index generation.[june/july 2013,6M,JUNE 2014,10M]**
6. **Determine the following for a 128-point FFT computation:**
    e) **number of stages**
    f) **number of butterflies in each stage**
    g) **number of butterflies needed for the entire computation**
    h) **number of butterflies that need no twiddle factors**
    i) **number of butterflies that require real twiddle factors**
    j) **number of butterflies that require complex twiddle factors.[JUNE 2012,6M]**
7. **Explain, how scaling prevents overflow conditions in the butterfly computation.[JUNE 2012,6M]**
8. **Explain, how scaling prevents overflow conditions in the butterfly computation.(June/July 2012, 6m)**
9. **With the help of the implementation structure, explain the FFT algorithm for DIT-FFT computation on TMS320C54XX processors. Use ¼ as a scale factor for all butterflies. (June/July 2012, Dec 2011, 12m.JUNE 2014,4M)**
10. **Derive the equation to implement a butterfly structure In DITFFT algorithm. (DEC 2011, 8M.DEC 2011,8M)**
11. **Determine the optimum scaling factor to prevent overflow.[JUNE/JULY 2011,10M. DEC 2014,8M]**
12. **i) Derive the equation to implement a Butterfly structure in DITFFT algorithm**
    ii)How many add/subtract and multiply operations are needed to compute the butterfly structure?
    iii)Determine the optimum scaling factor (DEC 2011, 6m.DEC 2015,08M)
13. **Write the subroutine for bit reversed address generation. Explain the same.**
14. **Why zero padding is done before computing the DFT? (DEC 2012, 2m. JUNE 2016,02M)**

15. **What do you mean by bit-reversed index generation and how it is implemented in TMS320C54XX DSp assembly language? (DEC 2012, 8m,DEC2015,06M)**
16. **What minimum size FFT must be used to compute a DFT of 220 point in redix 2 algorithm? Determine the number of butterfly structures needed for this algorithm and thereby determine number of complex multiplication and additions needed[DEC 2012,4M]**
17. **Write a subroutine program to find the spectrum of the transformed data using TMS320C54XX DSP. [DEC 2012, 6m]**
18. **With the help of the implementation structure, explain the FFT algorithm for DIT-FFT computation on TMS320C54XX processors. Use ¼ as a scale factor for all butterflies [DEC 2012, 6m]**
19. **Determine the following for a 1024-point FFT computation:**
    - **a) Number of stages**
    - **b) Number of butterflies in each stage**
    - **c) Number of butterflies needed for the entire computation**
    - **d) Number of butterflies that need no twiddle factors**
    - **e) Number of butterflies that require real twiddle factors**
    - **f) Number of butterflies that require complex twiddle factors. [MAY-JUNE 11]**
20. **Determine the following for a 512-point FFT computation:**
    - **m) Number of stages**
    - **n) Number of butterflies in each stage**
    - **o) Number of butterflies needed for the entire computation**
    - **p) Number of butterflies that need no twiddle factors**
    - **q) Number of butterflies that require real twiddle factors**
    - **r) Number of butterflies that require complex twiddle factors.[JUNE 2013,6M]**
21. **With the help of algorithm to generate bit reversed index for 16 bit DFT computation [DEC 2013,12M]**
22. **Write TMS320C54XX program for 8 point FFT implementation structure.[DEC 2013,08M]**
23. **What minimum size FFT must be used to compare 500 points DFT?What must be done to the samples before the chosen FFT is applied?[DEC 2014,04M]**
24. **Why scaling is required before or during butterfly computation?Draw the butterfly computation that uses 0.25 as scale factor[june 2015,4M]**
25. **i) What minimum size FFT must be used to compute DFT of 40 samples**
    ii) How many stages are required for FFT computation?
    iii) How many butterflies there per stage
    iv) How many butterflies are needed for the entire computation?[DEC 2015,06M]
26. **Write an assembly language program for implementing following on TMS320C54XX processors**

i)Bit reversed address generation. ii) Spectrum of the transformed data.[DEC 2014,8M]