International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:2682
Vol.4, No.4, April 2015 DOI:10.15693/ijaist/2015.v4i2.325-329

Efficient radix 2^k FFT Architecture

Mrs.R.Pavithra[1], Assistant Professor,KLNCE
S.Manoj Prabhakar[2], M.Marichamy[3],L.S.Karthik[4], KLNCE.

***Abstract*—The Fast Fourier Transform (FFT) is one of the most important algorithms in the field of digital signal processing. It is used to calculate the discrete Fourier transform (DFT) efficiently. The aim of the project is to design the radix-2^k FFT architecture. In feed forward architectures radix-2^k can be used for any number of parallel samples which is a power of two. Furthermore, both decimation in frequency (DIF) and decimation in time (DIT) decompositions can be used. In addition to this, the designs can achieve a very high throughput, which makes them suitable for the most demanding applications. Indeed ,the proposed radix-feed forward architectures require fewer hardware resources.**

I. INTRODUCTION

Application using frequency analysis of discrete-time signals in digital signal processor is the most convenient method especially in general-purpose digital computer or specially designed digital hardware. Frequency analysis is performed on a discrete-time signal {x(n)} by converting the time-domain sequence to an equivalent frequency-domain presentation

A straight forward representation of the Fourier transform is illustrated in figure 1.1. the figure, it shows the essence of the Fourier transform of a waveform is to decompose or separate the waveform into a sum of sinusoids of different frequency. Thus the pictorial representation of the Fourier transforms is a diagram which displays the amplitude and frequency of each of the determined sinusoids.

The Fourier transform identifies or distinguishes the different frequency sinusoids and their respective amplitudes which combine to form an arbitrary waveform. Mathematically , this relationship is stated as,

$$S(f) = \int_{-\infty}^{\infty} s(t)e^{-j2\pi ft}\, dt$$

The representation of finite Fourier Transform is given in {X(n)} of the sequence {x(n)}. Since {X(n)} is continuous function in frequency domain Fourier Transform, it is not computationally convenient to be represented by the sequence of {x(n)}. However the sequence of {x(n)} can be represented by sampling the spectrum {X(ω)} . This frequency domain representation leads to the discrete Fourier Transform (DFT), which is an important algorithm for performing frequency analysis of discrete-time signal. Discrete Fourier Transform (DFT) based signal processing is widely used and plays a significant role in digital signal processing algorithm.

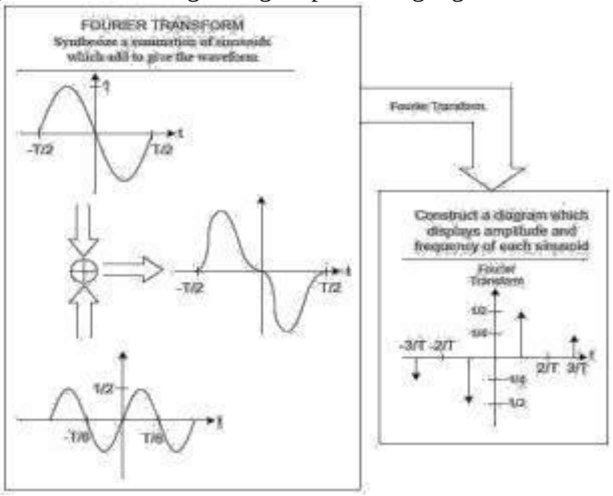

FOURIER TRANSFORM
Synthesize a summation of sinusoids which add to give the waveform
-T/2
T/2
t
-T/2
T/2
t
1/2
-T/6
T/6
t
Fourier Transform
Construct a diagram which displays amplitude and frequency of each sinusoid
Fourier Transform
1/2
1/4
-3/T -2/T
2/T 3/T
f

Fig .1.1 Interpretation of the Fourier transform

FAST FOURIER TRANSFORM

Fast Fourier Transform is a high efficient algorithm to compute the DFT. For the given round of error the FFT algorithm results in an equivalent data representation with the calculated DFT computation. The basic idea of this approach is to decompose the N-point DFT into successively smaller DFT. Eventually, this approach leads to a family of highly efficient computation of FFT algorithm.

Fast Fourier Transform is popularized by J. W. Cooley of IBM and John W. Tukey of Princeton University when they published a paper in 1965 which describes the fast computation of DFT. Several architectures have been proposed based on Cooley-Tukey algorithm to further reduce the computational complexity, including radix-2, radix-4 and split radix.

DECIMATION IN TIME

Decimation is the process of breaking down something into it's constituent parts. Decimation in time involves breaking down a signal in the time domain into smaller signals, each of which is easier to handle. This process of decimating the signal can easily be visualized. The first breakup into two N/2 point DFT can be shown as:

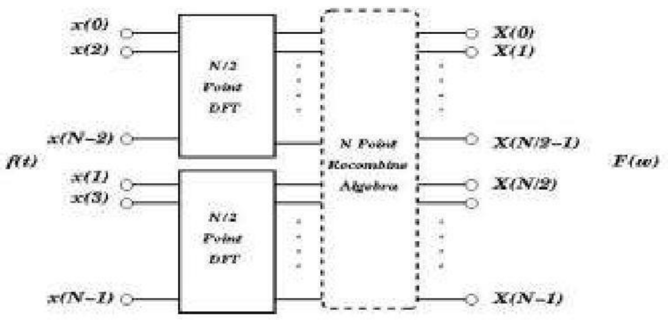


Fig.1.2 Breakup into two N/2 point DFT

The Recombine Algebra mentioned in the diagram is just used to combine the samples again in the correct order. This is repeated again and again until you reach a series of two point DFTs e.g. For an 8 sample signal:

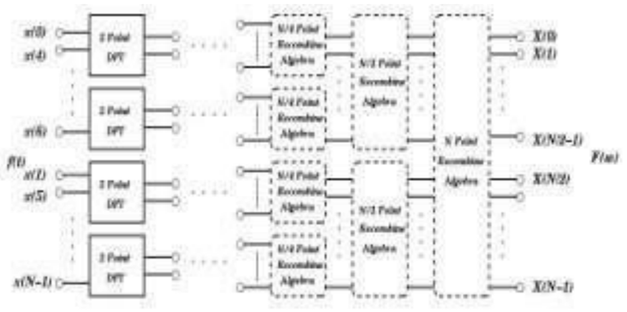
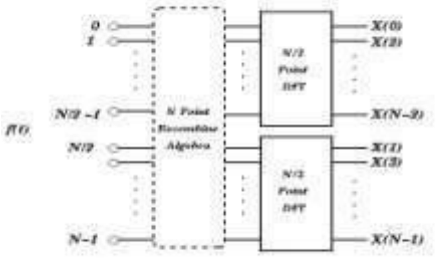

x(0)
x(4)
x(6)
f(t)
x(1)
x(5)
x(N-1)
2 Point DFT
N/4 Point Recombine Algebra
N/2 Point Recombine Algebra
N Point Recombine Algebra
X(0)
X(1)
X(N/2-1)
X(N/2)
X(N-1)
F(w)

Fig 1.3 Breakup for an 8 sample signal

Since the recombination algebra takes N complex multiplications and there are log2 (N) stages, the approx number of complex multiplications is N log2(N).This means that this decimation approach has reduced the number of complex multiplications from N squared (N2) to N log2 (N). At high values of N (i.e., large signals) this is a massive saving.

DECIMATION IN FREQUENCY

So far you've seen the FFT implemented by decimating the signal in the time domain. It is also possible to implement the FFT by decimating the signal in the frequency domain and recombine the signal in the time domain. Apart from the difference it follows the same pattern as the decimation in time method. The first stage is breaking the N point signal down into two N/2 point DFTs.



Compare this to the DFT diagram to see how the breakdown & recombination has been reversed. Just the same as the Decimation in Time method, the breakdown is continued until it reaches a series of two point DFTs such as this:

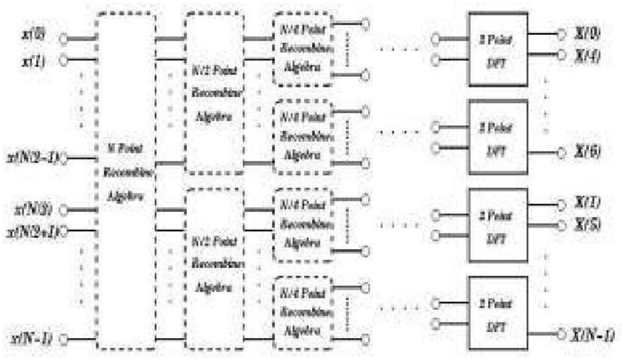

x(0)
x(1)
x(N/2-1)
x(N/2)
x(N/2+1)
x(N-1)
N Point Recombine Algebra
N/2 Point Recombine Algebra
N/4 Point Recombine Algebra
2 Point DFT
X(0)
X(4)
X(6)
X(1)
X(5)
X(N-1)

Fig 1.3 Breakup of series two point DFT

II. RADIX 2 DIF FFT

For radix-2,the DIF decomposition splits the output sequence into even and odd samples.The following equations are obtained

$$X[2r] = \sum_{n=0}^{\frac{N}{2}-1}\left(x[n] + x\left[n+\frac{N}{2}\right]\right)e^{-\frac{j2\pi rn}{\left(\frac{N}{2}\right)}}, r = 0,1,\ldots\ldots,\frac{N}{2}-1 \quad (2.1)$$

$$X[2r+1] = \sum_{n=0}^{\frac{N}{2}-1}\left(x[n] - x\left[n+\frac{N}{2}\right]\right)e^{-j2\pi/(Nn)}e^{-\frac{j2\pi rn}{\left(\frac{N}{2}\right)}}, r = 0,1,\ldots\ldots,\frac{N}{2}-1 \quad (2.2)$$

The N-point DFT is transformed into N/2 DFTs.Applying the procedure iteratively leads to the decomposition into 2-point DFTs.Fig 2.1 shows the flow graph of 8-point radix-2 DIF FFT.

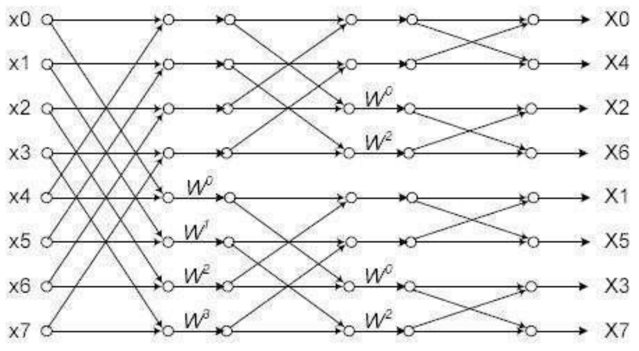


Fig. 2.1. Flow graph of a radix-2 8 point DIF-FFT

The graph is divided into 3 stages and each of them consists of set of butterflies and multipliers.The twiddle factor in between the stages indicate the multpilicaiton by W_n^k, where W_N denotes the *N*th root of unity, with its exponent evaluated modulo *N*. This algorithm can be represented as a data flow graph (DFG) as shown in Fig. 2.2.The nodes in the DFG represent tasks or computations.

,

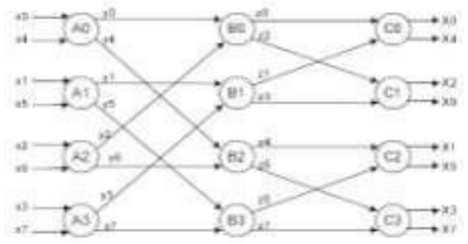


Fig.2.2.Data Flow graph (DFG) of a radix-2 8-point DIF FFT.

In this case, all the nodes represent the butterfly computations of the radix-2 FFT algorithm. In particular, assume nodes A and B have the multiplier operation on the bottom edge of the butterfly. The folding transformation is used on the DFG in Fig. 2.2 to derive a pipelined architecture. To transform the DFG, we require a folding set, which is an ordered set of operations executed by the same functional unit. Each folding set contains *K* entries some of which may be null operations. *K* is called the folding factor, the number of operations folded into a single function unit. The operation in the *j*-th position within the folding set (where *j* goes from 0 to *K* - 1) is executed by the functional unit during the time partition *j*. The term *j* is the folding order, the time instance to which the node is scheduled to be executed in hardware.

For example, consider the folding set $A = (\varphi, \varphi, \varphi, \varphi, A_0, A_1, A_2, A_3)$ for $K = 8$. The operation A_0 belongs to the folding set *A* with the folding order 4. The functional unit *A* executes the operations A_0, A_1, A_2, A_3 at the respective time instances and will be idle during the null operations. We use the systematic folding techniques to derive the 8-point FFT architecture. Consider an edge *e* connecting the nodes *U* and *V* with *w*(*e*) delays. Let the executions of the *l*-th iteration of the nodes *U* and *V* be scheduled at the time units $Kl + u$ and $Kl + v$, respectively, where *u* and *v* are the folding orders of the nodes *U* and *V*. The folding equation for the edge *e* is

$$D_F(U \to V) = Kw(e) - P_U + v - u \quad (2.3)$$

where P_U is the number of pipeline stages in the hardware unit which executes the node

RADIX-2^2 DIF FFT

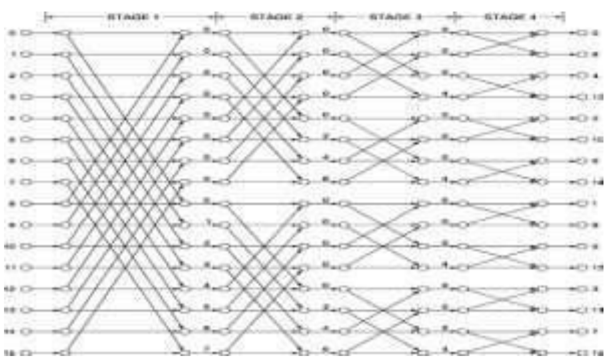
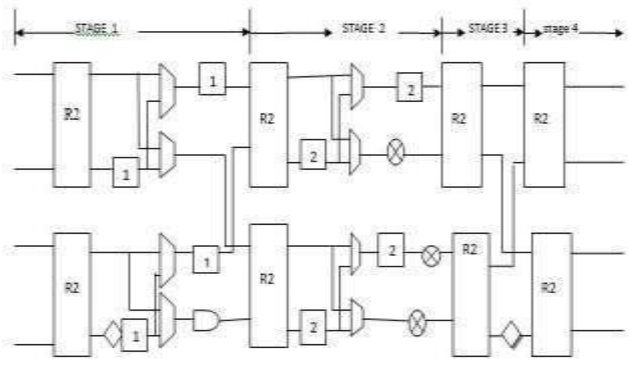


Fig.2.3.16 Point Radix-2^2 DIF-FFT architecture

After these two stages,full multipliers are required to compute the product of decomposed twiddle factors.The complete radix-2^2 algorithm can be derived by applying recursively.Fig.2.3 shows the flow graph of an N=16 point FFT decomposed according to decimation in frequency (DIF).The numbers at the input and output of the graph represent the index of input and output samples,respectively.The advantage of the algorithm is that it has the same multiplicative complexity as radix-4 algorithm,but still retains the radix-2 butterfly structures.We can observe that, only every other stage of the flow graph has non-trival mulplications. The –j notion represents the trival multiplication,which involves only real-imaginary swapping and sign inversion.

III 4 PARALLEL RADIX-2^2 FEED FORWARD ARCHITECTURE

The feed forward architectures,also known as multi-path delay commutator (MDC) do not have feedback loops and each stage passes the processed data to the next stage.These architectures can also process several samples in parallel.In current real-time applications,the FFT has to be calculated at very high throughput rates,even in the range of Giga samples per second.The Radix-2^k was presented for the SDF FFT as an improvement on radix-2 and radix-4.Next,radix-2^3 and radix-2^4,which enable certain complex multipliers to be simplified,were also presented for the SDF FFT.An explanation of radix-2^k SDF architectures can be found in.Finally,the current need for high throughput has been met by the MDF,which includes multiple interconnected SDF paths in parallel.However,radix-2^k has not been considered for feed forward architecture until the first radix-2^2 feed forward FFT architectures were proposed.The paper shows that radix-2^k can be used for any number of parallel samples which is a power of two.Accordingly,radix-FFT architectures for 2,4, and 8 parallel samples are presented.These architectures are shown to be more hardware-efficient than previous feed forward and parallel feedback design in the literature.This makes them very attractive for the computation of the most demanding applications.4-parallel architecture can be derived using the following folding sets.



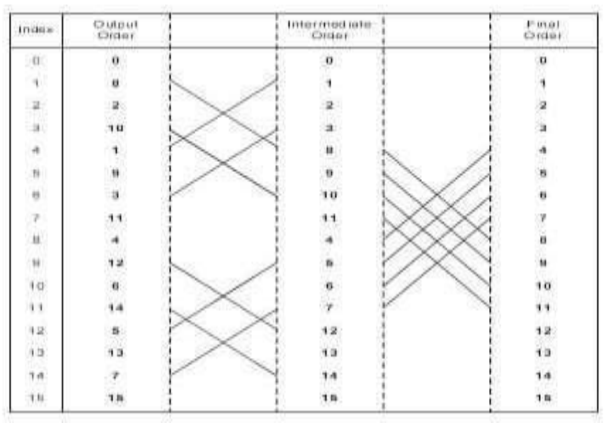
THIS IS proposed architecture 2^4 =16. 16 point input=output

Fig.3.1 Proposed 4-parallel (Architecture 2) for the computation of 16-point radix- 2 DIF FFT

The DFG shown in Fig is retimed to get the non-negative folded delays. The final architecture can be obtained following the proposed approach. For an *N*-point FFT, the architecture takes 4(*log*4*N* ¡ 1) complex multipliers and 2*N* ¡ 4delay elements. We can observe that hardware complexity is almost double that of the serial architecture and processes 4-samples in parallel. The power consumption can be reduced by 50% by lowering the operational frequency of the circuit.As for radix-2^2, these properties have been obtained directly from the flow graphs of the algorithms. The conditions for butterflies are the same for all stages of the FFT, whereas the conditions for rotations depend on the stage, S . Rotations are classified into trivial (T), non-trivial (NT), and rotations by W_8 or W_{16}. Rotations by and are not-trivial, but include a reduced set of angles. According the rotations by W_8 only consider angles that are multiples of ∏/4, whereas W_{16} only includes multiples of ∏/8. This allows for the simplification of the rotators that carry out the rotations. For this purpose, different techniques have been proposed in the literature. They include the use of trigonometric identities, the representation of the coefficients in canonical signed digit (CSD) and the scaling of the coefficients.

Reordering of output samples

Reordering of the ouput samples is an inherent problem in FFT computation.The outputs are obtained in bit-reversal order in the serial architectures.In general the problem is solved by using a memory of size N.Samples are stored in memory in natural order using counter for the addresses and then they are read it in bit-reversal order by reversing the bits of the counter. In embedded DSP systems, special memory addressing schemes are developed to solve this problem. But in case of real-time systems, this will lead to an increase in latency and area. The order of the output samples in the proposed architectures is not in the bit-reversed order. The output order change for different architectures because of different folding sets/scheduling schemes. We need a general scheme for reordering these samples. One such approach is presented in this section. The approach is described using a 16-point radix-2 DIF FFT example and the corresponding architecture is shown in Fig.3.3. The order of output samples is shown in Fig. 3.2



Index	Output Order	Intermediate Order	Final Order
0	0	0	0
1	8	1	1
2	2	2	2
3	10	3	3
4	1	8	4
5	9	9	5
6	3	10	6
7	11	11	7
8	4	4	8
9	12	5	9
10	6	6	10
11	14	7	11
12	5	12	12
13	13	13	13
14	7	14	14
15	15	15	15

Fig 3.2 solution to the reordering of outuput samples

The first column (index) shows the order of arrival of the output samples. The second column (output order) indicates the indices of the output frequencies. The goal is to obtain the frequencies in the desired order provided the order in the last column. We can observe that it is a type of de-interleaving from the output order the final order. Given the order of samples, the sorting can be performed in two stages. It can be seen that the first and the second half of the frequencies are interleaved. The intermediate order can be obtained by de-interleaving these samples as shown in the table. Next, the final order can be obtained by changing the order of the samples. It can be generalized for higher number of points; the reordering can be done by shuffling the samples in the respective positions according to the final order required.

A shuffling circuit is required to do the de-interleaving of the output data. Fig.3.3 shows a general circuit which can shuffle the data separated by *R* positions. If the multiplexer is set to ”1” the output will be in the same order as the input, whereas setting it to ”0” the input sample in that position is shuffled with the sample separated by *R* positions.

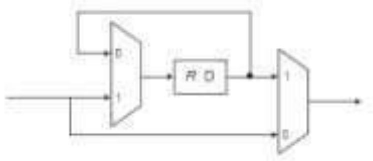


Fig 3.3 Basic circuit for data shuffling

IV.CONCLUSION

This architecture can be extended to radix- 2^k feedforward (MDC) FFT architectures. Indeed, it is shown that feedforward structures are more efficient than feedback ones when several samples in parallel must be processed.

In feedforward architectures radix -2^k can be used for any number of parallel samples which is a power of two. Indeed, the number of parallel samples can be chosen arbitrarily depending of the throughput that is required. Additionally, both DIF and DIT decompositions can be used.

REFERENCES

[1] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, “An efficient locally pipelined FFT processor,” *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 7, pp. 585–589, Jul. 2006.

[2] H. L. Groginsky and G. A. Works, “A pipeline fast Fourier transform,” *IEEE Trans. Comput.*, vol. C-19, no. 11, pp. 1015–1019, Oct. 1970.

[3] A. M. Despain, “Fourier transform computers using CORDIC itera-tions,” *IEEE Trans. Comput.*, vol. C-23, pp. 993–1001, Oct. 1974.

[4] S. He and M. Torkelson, “Design and implementation of a 1024-

point pipeline FFT processor," in *Proc. IEEE Custom Integr. Circuits Conf.*, 1998, pp. 131–134.

[5] M. A. Sánchez, M. Garrido, M. L. López, and J. Grajal, "Implementing FFT based digital channelized receivers on FPGA platforms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 44, no. 4, pp. 1567–1585, Oct. 2008.

[6] A. Cortés, I. Vélez, and J. F. Sevillano, "Radix [illegible] FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.

[7] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 414–426, May 1984.

[8] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 6, pp. 451–455, Jun. 2010.

[9] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix- [illegible] FFT/IFFT processor for MIMO-OFDM systems," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, 2008, pp. 834–8

[10] L. Liu, J. Ren, X. Wang, and F. Ye, "Design of low-power, 1 GS/s throughput FFT processor for MIMO-OFDM UWB communication system," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2007, pp. 2594–2597.