IMPLEMENTATION OF 2-D FAST FOURIER TRANSFORM FOR SUPER RESOLUTION

A Project Report
submitted in partial fulfillment of the
requirement for the Degree of

MASTER OF ENGINEERING

in

MICROELECTRONICS

by

Sqn Ldr K N Guruprasad

under the guidance of

Prof. S K Nandy

Department of Electronic Systems Engineering
Indian Institute of Science, Bangalore
Bangalore - 560012 (INDIA)
JUNE 2014

ACKNOWLEDGEMENT

First and foremost, I express my sincere and heart-felt gratitude to my adviser and guide, Prof. S K Nandy for his guidance and support throughout. His foresighted suggestions have always come to my help, whenever and wherever I stumbled.

I convey my sincere thanks to Gopinath Mahale for his involvement and support in the project work which helped me in understanding the subject well and carry out my tasks well. I offer my sincere thanks to Kala Nalesh, for her critical guidelines that she offered during this work. The Brainstorming sessions that I had with both Gopinath and Kala, proved to be very vital in understanding the topic as well as the work that I was expected to do. I express my special thanks to Hamsika Mahale for helping me out during the coding phase of the project. I wish them the best in all their future endeavors.

I owe my thanks to all the faculties of IISc for their insightful coursework. I thank the staff of DESE for their direct/indirect support, making my stay comfortable at IISc. I am grateful to this Alma-Mater for providing me an opportunity to interact with some of the brightest minds in India. This 2 year stay in this institute has improved my potential and changed the way I appreciate various technologies and related subjects. This would prove critical in my future activities in my parent organization.

I also offer my sincere thanks and best wishes to all my course mates for making these 2 years, the most wonderful time of my life. I wish this association grows stronger every passing day and brings in many more wonderful moments in all our lives.

I thank Indian Air Force, my parent organization, for providing me with this wonderful opportunity to do my post-graduation in this esteemed institute, which marks a very important turning point in my career and my life.

I thank my family for supporting me during difficult times in my life and motivating me to achieve many things during my stay in this esteemed institution.

K N Guruprasad
Squadron Leader
08 Jun 14

ABSTRACT

Constructing a higher resolution image from several low resolution images is called Super Resolution. In Image Processing applications, as data size increases, data processing becomes tedious and time consuming. In order to reduce the effect of these factors, we convert the image, which is in spatial domain, into frequency domain. Fourier Transform is one very well-known method of representing data in frequency domain. Fourier representation has been a topic of intense research for over half a century now. To calculate the Fourier Transform of images of bigger data sizes, various state of the art methods are available. 2 D FFT algorithm is one such method, which is used in this project work. In realizing this algorithm, we intend to make use of Row-Transpose-Column method. Accessing Column data increases the delay which is overcome by the Transpose operation. Our method eases the complexity, as it will be a Row operation, followed by a Transpose and finally another Row operation. But this operation requires repeated data access from memory. Further work is underway in this aspect.

CONTENTS

ACKNOWLEDGEMENT. i

ABSTRACT. iii

LIST OF EQUATIONS. . . . . vi.

LIST OF FIGURES. vii

1. INTRODUCTION 1

2. OVERVIEW OF FOURIER REPRESENTATION 3

2.1 Fourier Series. 3

2.2 1-D Fourier Transform 4

2.3 2-D Fourier Transform. 4

3. DISCRETE FOURIER TRANSFORM 7

3.1 Methods to compute DFT. 7

3.1.1 Brute Force method 7

3.1.2 Matrix multiplication method. 8

3.2 Fast Fourier Transforms 8

3.2.1 Radix 2 method 9

3.2.2 Radix 4 method. 9

3.2.3 Split Radix method 10

4. AVAILABLE ALGORITHMS 11

4.1 Available Algorithms. 11

4.1.1 Row Column decomposition method 11

4.1.2 Row Transpose Column method 12

4.1.3 Sparse Matrices method 13

5. WORKING ALGORITHM 15
5.1 Algorithm 15
5.1.1. Radix 4 Engine 15
5.1.2. 16 Input Radix 4 Engine 17
5.1.3. Block Diagram 17

6. MATLAB IMPLEMENTATION 22

7. CONCLUSION AND FUTURE WORK 23

8. BIBLIOGRAPHY 24

LIST OF EQUATIONS

(A) 1 Dimensional DFT. 4

(B) 1 Dimensional IDFT. 4

(C) 2 Dimensional DFT. 5

(D) 2 Dimensional IDFT. 5

(E) Symmetry Property. 7

(F) Periodicity Property 7

(G) Matrix Multiplication method. 8

(H) 2 D Matrix Multiplication method 8

(J), (K), (L) & (M) Radix 4 Butterfly Equations. 15

LIST OF FIGURES

Fig 1: Row Transpose Column method Block Diagram..13

Fig 2: Radix 4 Engine 16

Fig 3: 4 Input Radix 4 Engine…. 18

Fig 4: 16 Input Radix 4 Engine…. .19

Fig 5: Block Diagram of the implementation 21

INTRODUCTION

1. Now a days, providing fool proof security to various installations like airports, railway stations, banks, defence establishments and etc. is of utmost critical importance. Various security systems are available in market for ready use. These security systems keep monitoring the area of interest and allow access to people whose details are provided in their database. Data provided in these database, can be used to determine whether to provide or prevent access to area of importance. Face recognition is one such process, where we compare the facial features of a person and accordingly process his access request.

2. For efficient and reliable working of a face recognition system, we need high resolution images of personnel. Super resolution is a process of constructing a high resolution image from a single or multiple low resolution images. In a face recognition system, the images captured by surveillance cameras are generally of low resolution. When these low resolution images are fed to a face recognition system, the performance of the system is poor. Hence, before forwarding the image to a face recognition system, the resolution of the image needs to be enhanced. Several reconstruction techniques are already in use in the image processing field. These techniques improve the performance and reliability of a face recognition system. In this regard, converting image data (which is in spatial domain) into frequency domain becomes an unavoidable requirement in order to ensure easy and fast data processing.

3. This frequency domain representation is achieved by using various techniques. Fourier analysis is one such technique. Fourier analysis in itself has been a topic of very intense research in the past half a century. Basic principle of Fourier analysis is representing any signal as a complex sum of Sine and Cosine terms. As image size increases, representing it in Fourier (frequency) domain becomes more

and more difficult. Therefore, parameters of main concern here are size of data and complexity in computation involved. Towards this, a brief overview of several prevalent algorithms/techniques related to Fourier representation is presented in this report.

4. In Chapter 2, a brief overview of Fourier representation is presented followed by various popular and frequently used methods to calculate Discrete Fourier transform, in Chapter 3 and 4 respectively. In Chapter 5, some of the available computation algorithms are discussed. In Chapter 6, the status of implementation of the algorithm in MATLAB is presented followed by concluding remarks and the future work in Chapter 7.

OVERVIEW OF FOURIER REPRESENTATION

2.1 FOURIER SERIES

5. Jean Baptist Joseph Fourier, a French mathematician, had invented during early 19th century that any function that repeats periodically, can be expressed as a sum of sine and/or cosine relations of various frequencies, each multiplied by a coefficient. No matter how complicated the function is; it can be represented by such a sum, if it is periodic & meets some mathematical condition. Such a sum is now called Fourier series. On the other hand, functions which are finite but not periodic in nature can also be represented as an integral of sine and cosine functions multiplied by a weighing function. We call this representation as Fourier Transform.

6. Both Fourier series and Fourier transform representations have an important characteristic which enables them to reconstruct the input function with no loss of information. This allows us to work in the "Fourier domain" and then return to the original domain of the function without losing any information. As a result of intense research in Fourier analysis, enormous improvements in digital computation and discovery of various state of the art technologies and formulation of various algorithms/techniques to obtain Fourier representations, signal processing industry is attaining greater importance, every passing day. In this report, only those functions which have finite duration are discussed as the subject of discussion is an Image, which is a 2 dimensional function in spatial domain.

2.2 1 D FOURIER TRANSFORM

7. Fourier transform, commonly called as Discrete Fourier Transform (hence forth called DFT) of a discrete function x (n), n=0, 1, 2 ...N-1, is given by equation (A). Similarly, to reconstruct the original function, we use Inverse Discrete Fourier Transform (hence forth called IDFT), given by equation (B). (A) and (B) together, are called Discrete Fourier Transform pair. A Discrete Transform Pair has an important property, that, unlike the continuous case, we need not be concerned about the existence of the DFT or IDFT. The Discrete Fourier transform and IDFT always exist.

$$Y(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}$$

k = 0, 1, 2...N-1. ---(A)

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} Y(k) e^{j2\pi nk/N}$$

k = 0, 1, 2...N-1 ---(B)

8. In order to compute the forward and reverse transforms, we start by substituting k=0 in the exponential term and then take the sum for all values of n. This is followed by k=1, 2, N-1. As is evident, it takes approximately N^2 multiplications and N (N-1) additions to represent the function in the other domain. For simplicity and better understanding, we confine our discussion in this report to finite quantities. These comments are directly applicable to both 1 and 2 dimensional functions.

2.3 2 D FOURIER TRANSFORM

9. A data is said to be 2 dimensional if it is arranged in a matrix, i.e. Row-Column format. An image is one good example of such data. Extension of the 1 dimensional Discrete Fourier transform and its

inverse to 2 dimensions is straightforward. The DFT and IDFT of a function (image) f(x,y) of size M*N is given by the equations (C) and (D).

$$Y(k_1, k_2) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(k_1 x/M + k_2 y/N)} \quad \text{---(C)}$$

k1= 0,1,2,...M-1; k2 = 0,1,2,...N-1.

$$f(x, y) = \frac{1}{MN} \sum_{k1=0}^{M-1} \sum_{k2=0}^{N-1} Y(k_1, k_2) e^{j2\pi(k_1 x/M + k_2 y/N)} \quad \text{---(D)}$$

x= 0, 1, 2...M-1; y = 0, 1, 2...N-1.

10. In these equations, the variables k1, k2 are frequency variables while x and y are spatial variables. If f (x, y) is real, then its Fourier transform is conjugate symmetric. It is difficult to make any direct associations between specific components of an image and its transform. However, some general statements can be made about the relationship between the frequency components of the Fourier transform and spatial characteristics of an image. For instance, since frequency is directly related to rate of change, we can intuitively associate, very comfortably frequencies in the transform with patterns of intensity variations in the image. Usually, slowest varying frequency component corresponds to the average gray level of an image. As we move away from the origin of the transform the low frequencies correspond to the slowly varying components of the image. The higher frequencies correspond to faster gray level changes in the image. These are the edges of objects and other components of an image which have abrupt changes in gray level.

11. An important property of 2D DFT is that, it is separable. It means, we can take first 1D DFT row-wise and then take another 1D DFT

column-wise within appropriate boundary conditions. For standardization, we will be considering square matrices, in other words, images with same number of rows and columns, both being integer powers of 2. Calculating DFT of a test data involves complex multiplications and additions. As the data size increases, the number of such complex multiplications and additions involved in calculating DFT of the test data increases very rapidly, running into several orders of magnitude. In order to overcome such a computation intensive situation, reduce the complexity and make such calculations faster, various techniques and algorithms are developed. We will discuss some of these techniques in the following Chapters.

DISCRETE FOURIER TRANSFORM

3.1 METHODS TO COMPUTE DFT

12. Calculating DFT of an N point sequence requires N^2 complex multiplications and additions. For smaller values of N, handling so many computations does not project any major difficulties. But as the data size (N) starts increasing, we face a very highly computation intensive task. Before we discuss how to overcome this computationally intensive difficulty, let us briefly discuss various methods of calculating DFT.

3.1.1 BRUTE FORCE METHOD

13. In this method, we use standard equations, (A) and (B) to calculate 1D DFT whereas (C) and (D) to calculate 2D DFT. By varying values of n, k successively for the first case and x, y, k1 & k2 in the second case, we compute DFT. As told before, this requires N^2 complex multiplications and N(N-1) complex additions for all values of a 1D DFT, not to mention the number of indexing and addressing operations we need to do to fetch the data and to store the results. This is inefficient as it does not exploit the Symmetry and Periodicity properties of DFT given below.

Symmetry Property: **---(E)**

$$e^{-j2\pi(k+N/2)/N} = -e^{-j2\pi(k/N)}$$

Periodicity Property: **---(F)**

$$e^{-j2\pi(k+N)/N} = e^{-j2\pi(k/N)}$$

3.1.2 Matrix Multiplication Method

14. In this method, we represent the input sequence as a column vector of N rows (for 1D data) or as an N*N matrix (2D data). Their respective DFTs are obtained by multiplying these matrices with an N order square matrix called Twiddle Factor matrix, as shown below.

$$\mathbf{Y'} = \mathbf{F\,X\,'} \quad \text{---- (G)}$$

Where X is input matrix, Y is DFT matrix,

$$F = \begin{bmatrix} 1 & 1 & 1 & . & . & 1 \\ 1 & \omega & \omega^{2} & \omega^{3} & & \omega^{(N-1)} \\ 1 & \omega^{2} & \omega^{4} & \omega^{6} & & \omega^{2(N-1)} \\ 1 & \omega^{3} & \omega^{6} & \omega^{9} & . & \omega^{3(N-1)} \\ . & . & . & . & . & . \\ 1 & \omega^{(N-1)} & \omega^{2(N-1)} & \omega^{3(N-1)} & & \omega^{(N-1)(N-1)} \end{bmatrix}$$

$$\omega = e^{-j2\pi/N},$$

For 2D data, we follow a procedure as mentioned below,

$$Y = F_M * X * F_N \quad \text{---- (H)}$$

where FM and FN are Twiddle factor matrices for M rows and N columns, X is input data while Y is the DFT matrix.

3.2 FAST FOURIER TRANSFORMS

15. In order to overcome the challenge of intense computation and speed, various new approaches have been invented. By adopting suitable methodology, we can make the entire process of calculating the DFT easier and simpler. Divide-and-Conquer approach is one such

technique. The main essence of this approach is to divide the input signal into smaller sequences and successively calculate DFTs for those smaller sequences. In this regard, many techniques are prevalent out of which some important ones will be discussed in the next few paragraphs.

3.2.1 RADIX 2 METHOD

16. In this method, we consider N to be an integer power of 2 and divide the entire input sequence into even and odd numbered sub sequences. Now the N point DFT can be expressed in terms of DFTs of these subsequences which are periodic with period N/2. In the next step we divide these 2 subsequences further into 2 each subsequences (total of 4) and take their DFT. This approach is repeated till we reduce the N point input sequence into N 1 point sequences. At the end, the number complex multiplications required, will reduce from N^2 to $N \log_2 N$. This method has an advantage of being very simple and hence easy to implement. However, when N takes higher values, implementation uses considerable amount of resources for computation, which in turn becomes slower.

3.2.2 RADIX 4 METHOD

17. When the number of data points N in the DFT is a power of 4, we can, of course, always use a radix-2 algorithm for the computation. However, for this case, it is more efficient computationally to employ a radix-4 FFT algorithm. In this method, we divide the input sequence into 4 N/4 length subsequences, after which we follow a modified procedure which loosely resembles Radix 2 method. With this method we are able to reduce complex multiplications by 25% but the additions increases by 50%. Further details about this method will be discussed later in Chapter 4.

3.2.3 SPLIT RADIX METHOD

18. An inspection of the radix-2 method indicates that the even-numbered points of the DFT can be computed independently of the odd-numbered points. This suggests the possibility of using different computational methods for independent parts of the algorithm with the objective of reducing the number of computations. The split-radix FFT (SRFFT) algorithms exploit this idea by using both Radix 2 and Radix 4 decomposition in the same FFT algorithm.

19. Thus the N-point DFT is decomposed into one N/2-point DFT without additional twiddle factors and two N/4-point DFTs with twiddle factors. The N-point DFT is obtained by successive use of these decompositions up to the last stage. Thus we obtain SRFFT algorithm.

20. There are a few more methods which use higher radices like Radix 8, powers of 2, 4, 8, 16 and etc. But implementing these methods becomes more and more complex as too many parameters are involved in their implementation as well as possessing non homogeneous structure. With this background, we restrict our discussion of various methods/algorithms of calculating DFT to this point, while focusing our attention to a method which we intend to use in our implementation.

AVAILABLE DFT ALGORITHMS

4.1 AVAILABLE ALGORITHMS

21. Most of the methods that we discussed briefly, in the previous chapter, have been implemented commercially, with considerable success. Huge amount of research and analysis efforts have been put in for implementing these methods into hardware. During the last couple of decades, these efforts have resulted in publishing of several literature works. While implementing a 2D FFT, we need to take various parameters like data size, speed of operation and resources available among others, into consideration. A delicate compromise will have to be made between size, memory resources and speed of operation. We, in the next section, carry out a qualitative study of some of the methods which are presented in these literatures.

4.1.1 ROW COLUMN DECOMPOSITION [10]

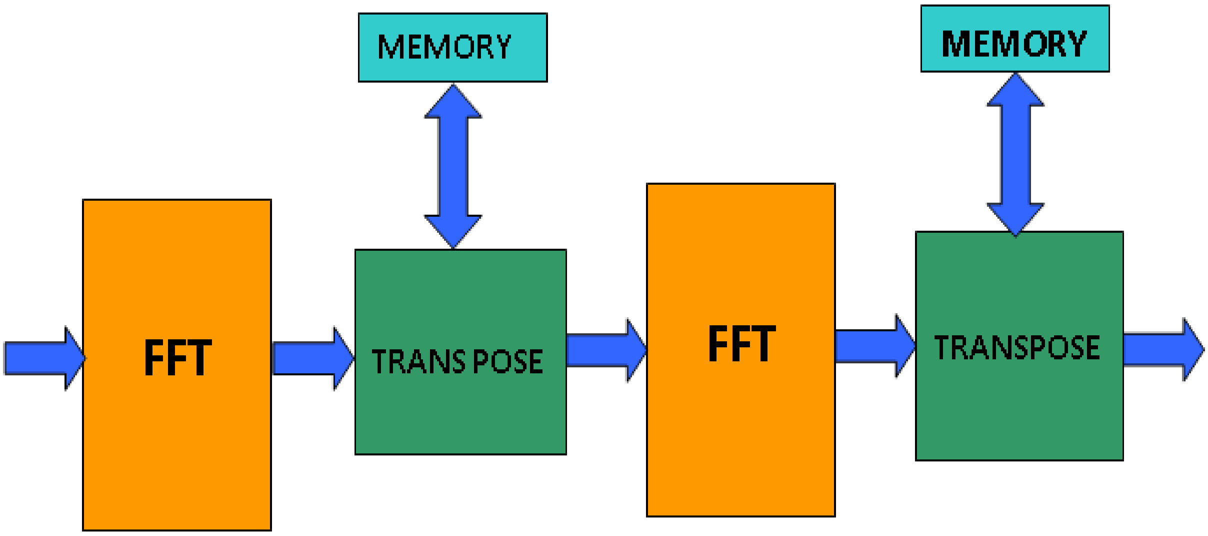
22. As we know that image data is stored in 2 dimensional matrices consisting of rows and columns, a simple method of processing this data is by using Row Column operations, as done in general matrices operations. In Row Column decomposition method, a 2D DFT on N*N data is computed by first performing N row-wise 1D DFTs and then N column-wise 1D DFTs. If these 1D DFTs are implemented using FFT, the computational complexity is $N^2 \log_2 N$. In implementations like this, input 2D data is initially stored in the external memory and DFT operations are carried out. When we make use of SRAM for such data storage, for smaller data sizes, implementation performs very well, as row and column access times are same. For large data sizes, using SRAM becomes impractical due to limited size and high cost. Using SDRAM with larger capacity, as external memory for larger data sizes proves effective due to high throughput and high capacity.

23. We know that SDRAM is arranged in 3D structure consisting of rows and columns, which are again arranged in banks, it has non uniform access time; accessing consecutive elements in a row has lower latency than accessing data across different rows, which has high latency. As a result, row-wise DFTs are much faster than column-wise DFTs. Even as we use DDR2 or DDR3 standards for SDRAM, existing memory interfaces fail to utilize bandwidth of SDRAM device for column DFTs. This method gives poor performance even when we custom design the memory interface. Hence, data transfer between off-chip and on-chip memories is the most critical bottleneck.

4.1.2. ROW TRANSPOSE COLUMN DECOMPOSITION [11]

24. This method resembles the previous method to a large extent. However, the difference between the two methods lies in the point that additional transpose operation is added here which makes this method to be better than the previous one. The problem of data transfer between memories is addressed by undertaking additional transpose operations. After taking row-wise 1D DFT, we save the result of first step as a transposed matrix.

25. As next step, we take another row-wise 1D DFT on this transposed matrix, this time, actually taking DFT on the column data. Once DFT calculation is over, data is saved in the same order. To get the final result, we need to do another transpose operation, which puts the result into actual order in which the input data was initially accessed. Even though the performance is improved, requirement of transferring data to and from memory for repeated transpose operations makes it cumbersome and inefficient for large data sizes. Figure below shows the block diagram representation of this method and practically how this method is achieved.



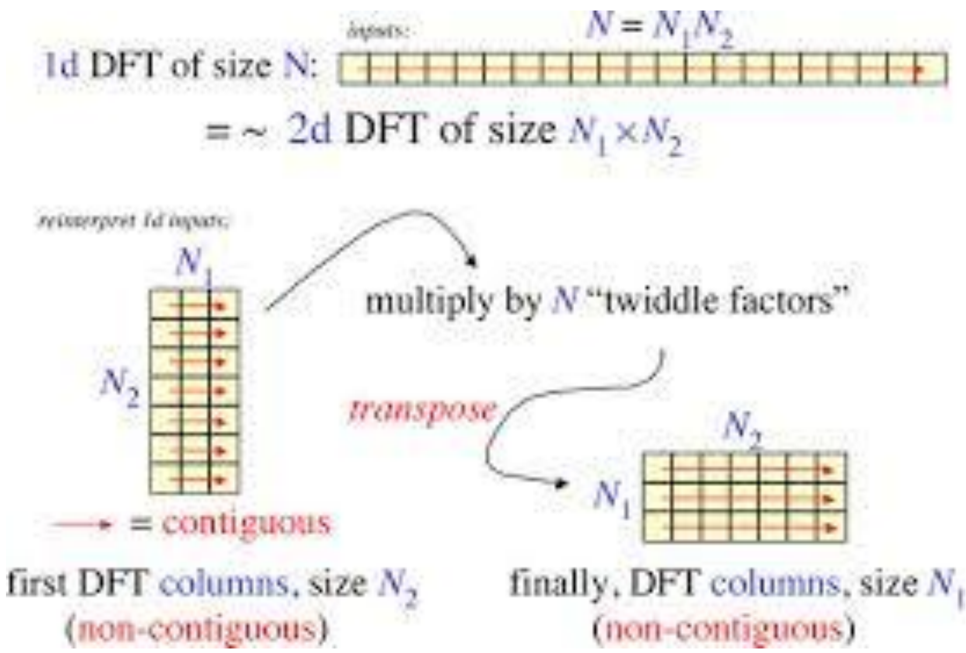


Fig 1: Row-Transpose-Column method with Block Diagram.

4.1.3. Sparse Matrices Multiplication Method [7]

26. Another way of computing 2D DFT is to use Sparse matrices multiplication method. The data is partitioned into a mesh of sub blocks in both rows and columns, each sub block having a predetermined number of rows and columns, perform butterfly operations between sub blocks which involve data exchange with individual elements and

twiddle factor multiplication. First, we do this operation on rows followed by columns. Then, we compute local 2D DFT on each sub block. In the last step, by rearranging this matrix according to a specific permutation, we obtain final 2D DFT.

27. In the next chapter, we propose and explain an algorithm for implementation and discuss the block diagram of the algorithm. We also explain the functioning of Controller which is being designed.

WORKING ALGORITHM

5.1 ALGORITHM

28. In this project work, we are using Row Transpose Column method using simple Radix 4 algorithm to accomplish the task of calculating 2D FFT of the images being considered. Before we go further into details of the proposed work, it is imperative for us to understand Radix 4 algorithm, as such.

5.1.1 RADIX 4 ENGINE

29. As mentioned earlier in chapter 3, Radix 4 method involves a very easy technique of using twiddle factors 1,-1, j and –j. To understand this method easily, let us consider a simple Radix-4 butterfly unit. Such a butterfly unit takes 4 inputs and using 3 complex adders/ subtractors, 4 outputs are generated sequentially. The block diagram representation is depicted in Fig 2. The basic operations in a Radix-4 butterfly are given by 4 equations, as given below.

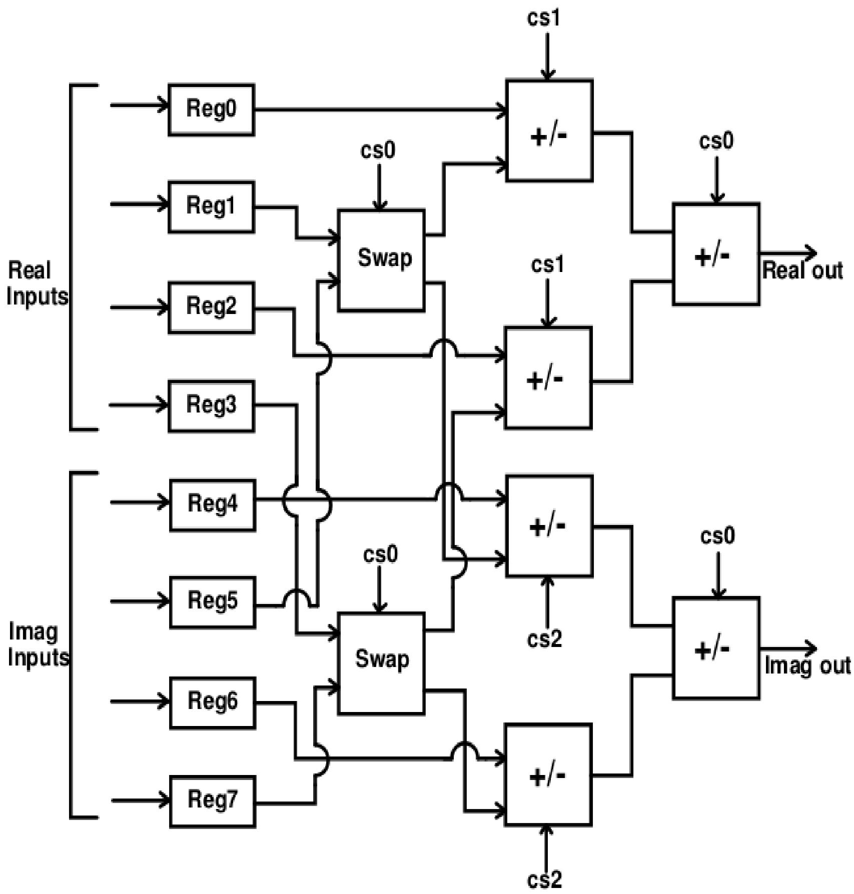
$$Y(0) = X(0) + X(1) + X(2) + X(3) \quad \text{-------------} \ \textbf{(J)}$$

$$Y(1) = X(0) - j\,X(1) - [X(2) - j\,X(3)] \quad \text{-----------} \ \textbf{(K)}$$

$$Y(2) = X(0) - X(1) + X(2) - X(3) \quad \text{-----------------} \ \textbf{(L)}$$

$$Y(3) = X(0) + j\,X(1) - [X(2) + j\,X(3)] \quad \text{-----------} \ \textbf{(M)}$$

Fig 2: Radix 4 Engine.



Control signals in Radix 4 Engine.

Mode	Output	cs0	cs1	cs2
0 0	X(0)	0	0	0
0 1	X(1)	1	0	1
1 0	X(2)	0	1	1
1 1	X(3)	1	1	0

30. The 4 complex inputs are loaded in parallel to the input registers. The output to be generated for each set of inputs is controlled by mode select signal. The individual control signals cs0, cs1 and cs2 are derived from mode select signal. The adder/subtractor control signal selects adder or subtractor. Depending on the signal inputs cs0, cs1and cs2, output is generated as per the control signal table. Multiplication with j is done using a swap unit with a control signal to select whether to swap or not. Whenever swap is selected, depending on the Mode input, real and imaginary parts are swapped, and on similar lines, adders or subtractors are selected as per the 4 equations mentioned earlier. Another advantage of Radix 4 engine is that it can be operated in Radix 2 mode also, by feeding 2 signals as inputs in any 2 input terminals and taking outputs by suitably selecting 2 output terminals. This feature proves to be very important for data sizes which are not integer powers of 4 e.g. N=32 and 128. This feature of radix 4 engine can be exploited to make a homogeneous design in which by taking a radix 4 engine as the basic building block and used repetitively across the design to accomplish calculation of higher point FFT. Using Radix 4 engine in Radix 2 mode in the final stage and in normal mode in all previous stages eases the process and makes the algorithm flexible too. A Radix 4 engine is the effective building block for this algorithm, around which all other elements are built.

5.1.2. 16 INPUT RADIX 4 ENGINE

31. Taking this knowledge a level higher, we intend to use 4 such Radix 4 engines in parallel (shown in Fig 3, henceforth to be referred to as 4R4 engine) thereby taking 16 inputs simultaneously and taking 4 outputs depending on the mode inputs. This becomes another building block, along with Radix 4 engine.

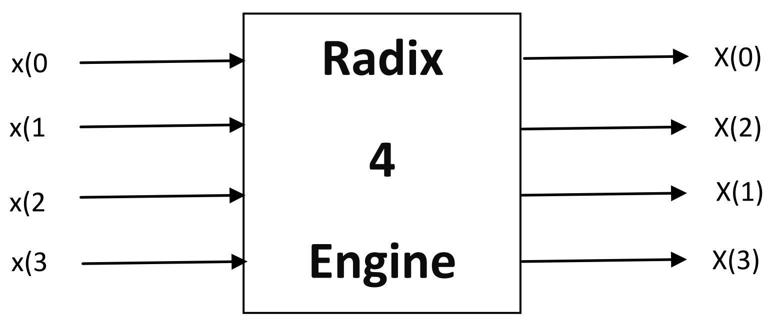
5.1.3. BLOCK DIAGRAM

32. At the beginning of operation, input data is transferred into the on chip memory from external memory, and depending on the size, input

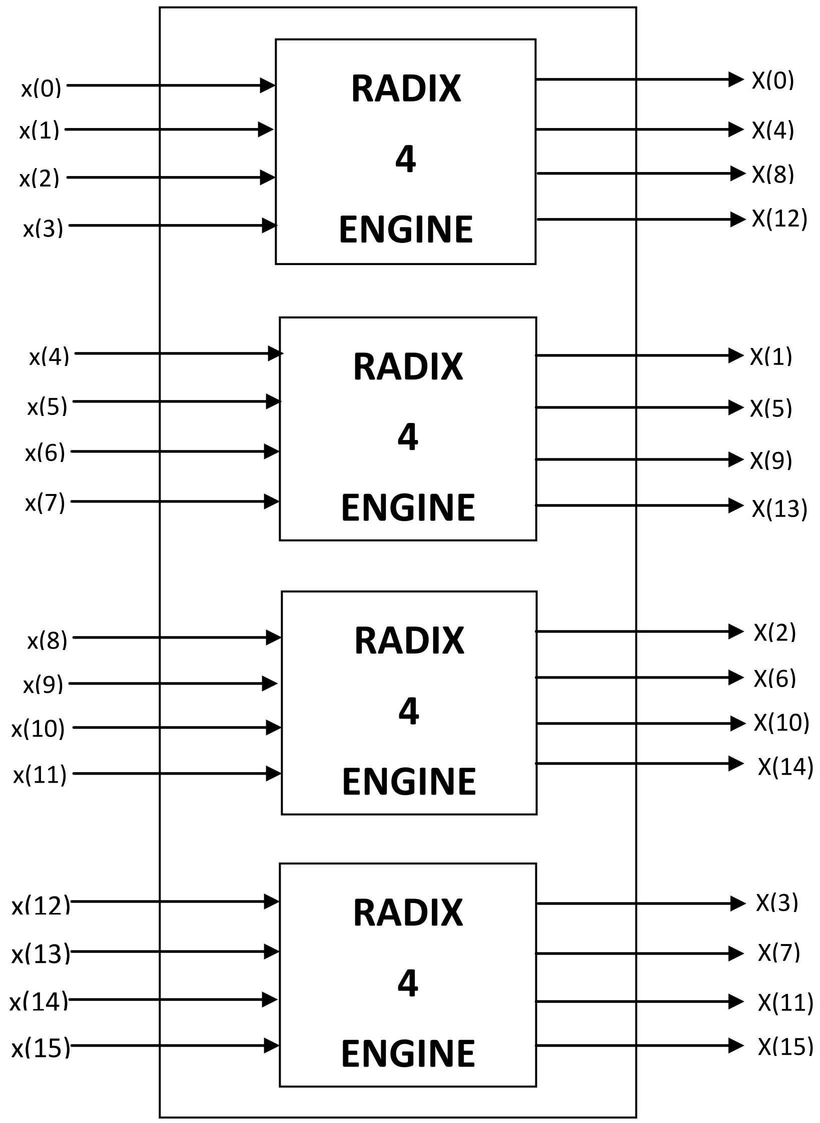
is taken into the engine, as per the required combination. Once we process this data using 4R4 engine, result will be saved in a buffer where we will have to undertake multiplication with suitable Twiddle factors. Depending on the size, we will repeat this operation to cover all the data inputs. After completing the first stage processing and multiplication with twiddle factors, we will undertake next stage activities, which involve repetition of the above said steps but with changed data input order and different twiddle factors.

33. Controller takes the size of input data as input and calculates whether Radix 4 engine needs to be operated in Radix 2 mode, the number of iterations required for the data processing, how many repetitions each iteration consists of, as well as the mode inputs. It is intended to calculate all the twiddle factors required, in priory and store them in the form of Look up Tables for ready reference.

<u>Fig 3: 4 Input Radix 4 Engine.</u>



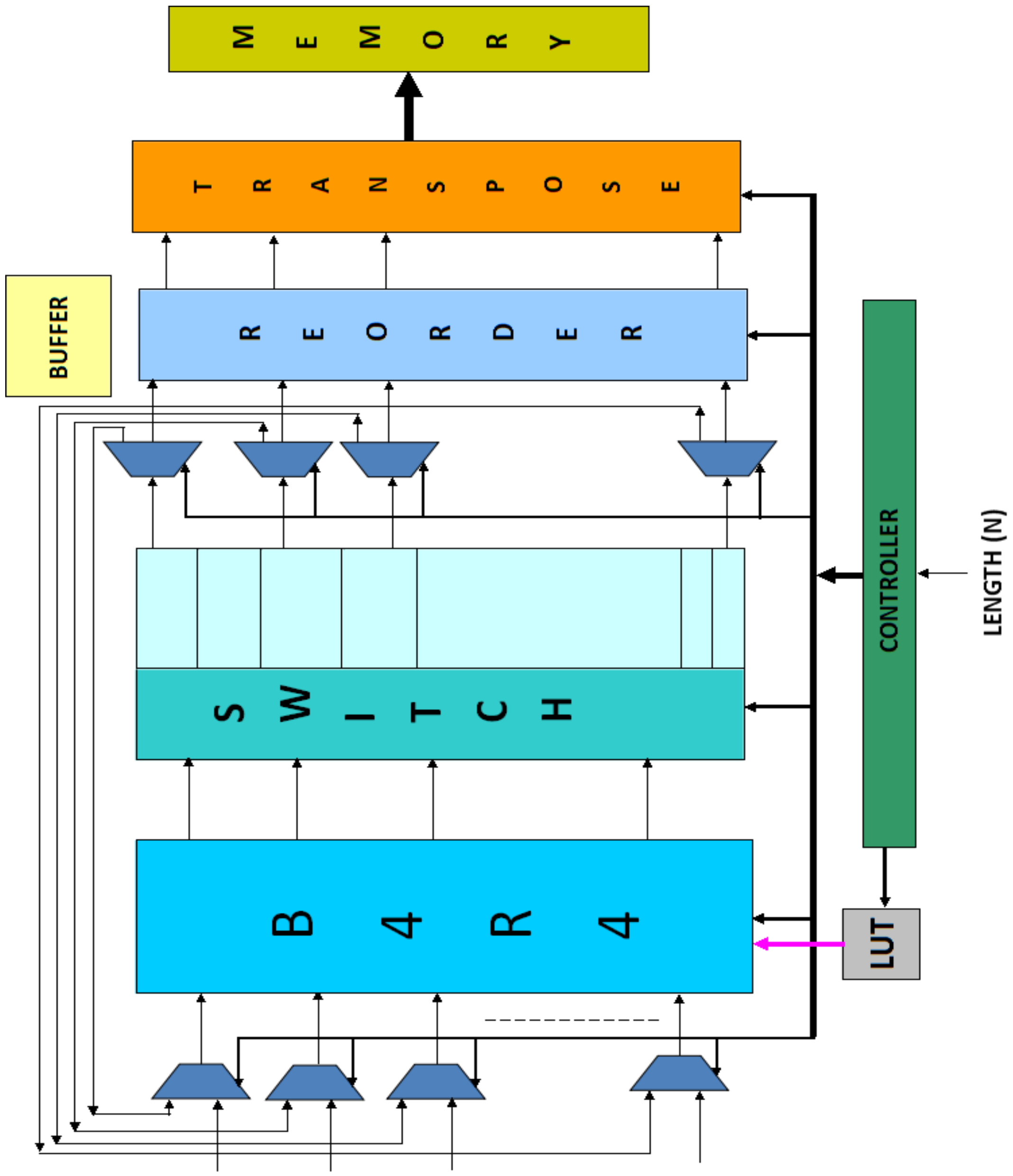
<u>Fig 4: 16 Input Radix 4 Engine.</u>



34. Input data is fed to 4R4 engine through a multiplexer stage. Second set of input to this multiplexer stage is given from the output stage as a feedback which is given from different stages, decided by the controller. If the number of iterations required for complete computation of row FFT of all the data points in the dataset is not over, then the feedback comes from the lower stages. On the other hand, if the iterations are over in computing Row FFT, then this output is reordered and fed to Transpose stage, where the computed data is transposed before being fed back to compute the Column FFT. After execution of required number of iterations, data is again reordered and transposed. This transposed data will deliver the result in exactly the same order as the input data.

35. Various twiddle factors for different data sizes are computed beforehand and stored as lookup tables. Depending on the requirement, particular set of twiddle factors are accessed from these tables and used during FFT computation. This eliminates the additional task of twiddle factor computation at run time. For ease of computation of twiddle factors, we are considering data sizes which are integer powers of 2.

Fig 5: Block Diagram.



MATLAB IMPLEMENTATION

36. The algorithm proposed, is implemented in MATLAB in stages. In the first stage, a simple Radix 4 engine was designed. After designing, 4 such engines are initiated in parallel, which take 16 inputs (real or complex) and produce 4 outputs simultaneously depending on mode signal input.

37. In the second stage, subroutines concerned with reordering function were written. Then, function to calculate twiddle factors was written. Later functions for buffering data after multiplication with twiddle factors and the main function which integrates all the subroutines were written. Finally, the code for integrating all these functions was written to get the final output.

38. For coding the algorithm in MATLAB, a matrix of order 64 is considered. To test the consistency of the algorithm, a MATLAB code was written for a 4*4 matrix. Later another Matlab code was written for a data size of 64. Program was tested for both real and complex inputs. Results were checked with those obtained from built in MATLAB functions. Results are matching till 3 decimal points.

CONCLUSION AND FUTURE WORK

39. Literature survey of various DFT calculation and implementation techniques was carried out and then a suitable algorithm to implement 2D DFT for Super Resolution application is proposed. Row Transpose Column technique is used for the said implementation. For calculating the DFT, Radix 4 algorithm is used. The algorithm was implemented in MATLAB obtaining desired results.

40. The future work in this project would involve the following:-

(i) Hardware implementation of the algorithm on an FPGA platform.

(ii) Analysis of performance testing and associated performance enhancement of the hardware implementation of the algorithm.

BIBLIOGRAPHY

[1] **“Digital Signal Proccessing”**, 3rd edition, John G. Proakis, Dimitris G. Manolakis.

[2] **“Understanding FFT Application”**, 2nd edition, Anders E Zonst, Citrus Press, 2004.

[3] **“Digital Image Processing”**, 2nd edition, Rafael C. Gonzalez, Richard C. Woods.

[4] **“The FFT Via Matrix Factorizations”**, A Key to Designing High Performance Implementations, Charles Van Loan, Department of Computer Science, Cornell University.

[5] **“On Computing the 2-D FFT”**, Dragutin Sevic, IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 47, NO. 5, MAY 1999: 1429-1431.

[6] **“FAST FOURIER TRANSFORMS: A TUTORIAL REVIEW AND A STATE OF THE ART”**, **P**. Duhamel & M**.** Vetterli, Signal Processing 19 (1990) Elsevier: 259-299.

[7] **“FPGA Architecture for 2D Discrete Fourier Transform Based on 2D Decomposition for Large-sized Data”**, Chi-Li Yu, Jung-Sub Kim, Lanping Deng, Srinidhi Kestur, Vijaykrishnan Narayanan & B Chaitali, J Sign Process Syst (2011) 64:109–122(Springer).

[8] **“A Hardware Acceleration Platform for Digital Holographic Imaging”**, Thomas Lenart, Mats Gustafsson & Viktor Owall, J Sign Process Syst (2008) 52: 297–311(Springer).

[9] **“FAST FOURIER TRANSFORM ALGORITHMS WITH APPLICATIONS”**, A Dissertation presented to the Graduate School of Clemson University by Todd Mateer, August 2008.

[10] **“Generating FPGA accelerated DFT libraries”**. IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM) D’Alberto, P., et al. (2007), 173–184.

[11] **“A Hardware Acceleration Platform for Digital Holographic Imaging”**, T. Lenart, M. Gustafsson, & V. Owall (2008). Journal of Signal Processing System, 52(3), 297–311.