Design and Implementation of Continuous Flow FFT Processors for OFDM in Wireless GigaHertz Standards

A

Thesis submitted

for the award of the degree of

Doctor of Philosophy

By

Sumit Agarwal

Department of Electronics and Electrical Engineering

Indian Institute of Technology Guwahati

Guwahati, Assam - 781039, INDIA

November, 2023

Design and Implementation of Continuous Flow FFT Processors for OFDM in Wireless GigaHertz Standards

SUMIT AGARWAL

C E R T I F I C A T E

This is to certify that the thesis entitled **"Design and Implementation of Continuous Flow FFT Processors for OFDM in Wireless GigaHertz Standards"** submitted by **Sumit Agarwal** bearing Roll No. 09610213, a research scholar in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, for the award of the Degree of **Doctor of Philosophy** is a record of an original research work carried out by him under our supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and in our opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted elsewhere for award of any degree or diploma.

Dated:
Guwahati

Prof. Rafi Ahamed Shaik
Professor
Dept of Electronics and Electrical Engg.
Indian Institute of Technology
Guwahati, Assam, India

Salutation to the noble Guru, who is Brahma, Vishnu and Maheswara, the direct Parabrahma, the Supreme Reality.

Acknowledgement

I would like to articulate my deep gratitude to my supervisor, **Prof. Rafi Ahamed Shaik**, who has always been my motivation for carrying out the project. I would like to thank him for his valuable advice, support and the confidence he placed in me. I am also grateful to my ex-supervisor, **Prof. Anup Kumar Gogoi** for his never ending support.

An assemblage of this nature could never have been attempted without reference to and inspiration from the works of others whose details are mentioned in the reference section. I acknowledge my indebtedness to all of them.

I would also like to express my gratitude to all the faculty members, the technical and non-technical staff of the department for guiding me in one way or the other. I would like to thank specially, Dr Gaurav Trivedi, with whom, I had fruitful discussions. Special thanks are due to the team at Semi-conductor Laboratory, Chandigarh, a unit of Department of Space, Government of India. The thesis could not have been completed without the teams support especially, Shri H.S. Jatana, Uday Khambete, Debjyoti Mallik, Gaurav Sarkar who extended their great help in the fabrication of the chip. Thanks to all my friends for the thoughtful and motivating discussions we had. Special thanks goes to Prof. Apurba Kalita of Assam Engineering College for discussions on matters related to digital circuits. I am especially grateful to my parents for their love and support. Last but not the least, special thanks to

my wife Meenakshi Agarwalla, MTech (VLSI) who had been a constant source of technical discussion and to my daughters, Manvi and Dhvija Agarwal, who have waited patiently all along for the completion of the thesis extending full support.

I extend my sincere thanks to all of them who have patiently extended all sorts of help for accomplishing this undertaking.

Date: 08.11.2023 **Sumit Agarwal**

Indian Institute of Technology Guwahati

Abstract

The throughput requirement of latest OFDM based IEEE 802.11ay WLAN standard is between 20 to 40 Gbps. Also, the FFT processor needed for OFDM must work in continuous mode for real time communication. The number of FFT points can be variable. In this thesis, we propose a Continuous flow architecture for a 512 point FFT to meet this requirement at about 28 Gbps. The number of points are kept fixed here to illustrate the essential features of the design. They can be varied, if desired, with minimal architectural modifications. Architectures to meet the throughput requirement (10 Gbps) of earlier WLAN standard IEEE 802.11ad have been reported in the literature. The proposed architecture achieves more than double this throughput at 28 Gbps with similar chip area and clock as the best existing 10 Gbps designs. This is made possible through a specialized design for OFDM unlike the earlier FFT chips which were designed for general purpose FFT. The proposed architecture uses two radix-16 and one radix-2 stages to meet the high throughput requirement. Standard continuous flow (CF) FFT designs use two memories. The proposed design exploits the smaller wordlength of 4 bit (for 64 QAM) of OFDM to introduce an additional smaller input memory and a simpler processing element (PE) for the input stage. Combined with the existing two memories, there are now three memories for the three stage FFT. Thus this design allows memories to assume dedicated roles for each stage. Compared to the existing practice of switching of memories, dedicated memories need a novel addressing scheme to maintain CF as data is replaced in same memory rather than switching the memories.

Another feature of the architecture is that each memory feeds a single stage PE and is

not shared with other stages. Thus, a simplified conflict free memory bank access circuit is developed which saves area. Conflict-free addressing (CFA) techniques are necessary for all FFT hardware. For low radices, well-proven XOR-based addressing architectures are available in the literature. In the previous CFA techniques, the complexity increases with increasing radices or memory sets. In the proposed technique, the higher number of memory sets and radix is an advantage. A novel scheme is suggested to reduce the complexity using a progressive shifting technique. The mathematical basis of the scheme is derived here and illustrated with an example of 512-point radix-16 FFT. The proposed and existing CFA architectures are designed using verilog and synthesized in the SCL library. The result of synthesis shows that the proposed scheme achieves a 33% area reduction compared to the XOR-based scheme. The area reduction factor further improves with a higher number of FFT points.

A comprehensive theory for CF is formulated and two CF schemes are proposed both of which save significant switching area and power. CF of data for mixed-radices is a challenge because the output index bit-pattern is asymmetric to that of the input. Reported CF techniques are all based on data reordering to make the address symmetric. Mostly, they are specific to an architecture and lack any mathematical framework. Additionally, the presented solutions are applicable only for particular stages of the architecture. In this thesis, group theory is used to develop a universal data reordering scheme for CF. It is proved to be valid for all mixtures of radices and is applicable to all the stages viz. input, intermediate and output. This thesis further proposes that an alternative to shifting of data is to shift the addresses instead. A mathematical framework is developed which uses group theory to prove that a circular periodicity in addresses exists for FFT algorithms. This periodicity allows new addresses to be generated without any reordering of data. Depending on the FFT architecture, either technique may result in minimum hardware. A mixed radix-16 FFT processor with 512 points is designed using both the techniques and the ASIC synthesis results on SCL technology show that the overall designs are about 70% more area efficient compared to the existing designs. The 512-point chip accomplishes FFT symbol processing in 32 clock cycles compared with 74

clock cycles of earlier processors. The design synthesized in 90 nm technology meets the 4.8 GSps (28 Gbps) processing rate at a clock speed of 300 MHz. To test its functionality, the chip is fabricated in SCL 180 nm technology and packaged in 120 pin package.

Contents

List of Figures

List of Tables

1

Introduction

Contents

1.1 High Speed Wireless Standards

When Wi-Fi was initially introduced, it was meant for Laptops which could be taken anywhere, and wired connections for LAN were difficult to move. The unlicensed band of 2.4 GHz and 5 GHz were used with speeds up to 54 Mbps. That was way back before 2005. The standard which was most famous for this purpose was the 802.11g. In 2009, 802.11n was introduced in the 5 GHz band to meet speed demands of 100 Mbps or more using MIMO. Since 2012, 60 GHz wireless research has become the focus of the industry. Newer standards were introduced in this band, such as 802.11ad, which was modified for further increase in throughput under the IEEE specifications 802.11ay. The advancement of electronics in newer fields is the driving force behind the high throughput requirement for Wireless connectivity. New usage models which are fueling the growth of the latest WLAN standards are listed below:

- Augmented Reality and Virtual reality
- Multiple connected gaming devices including online gaming
- HDTV video streaming to various devices in a household such as laptops, projectors, TVs, etc.
- Internet browsing/ download/ upload and smartphones are expected to use the home or office WiFi network to do heavy downloads and uploads of high-resolution photos and videos. This traffic has increased tremendously due to popular social media sites such as Whatsapp, Facebook, Twitter, etc., allowing live capturing and sharing of events.
- Wireless display and streaming directly from camcorders to display devices
- Outdoor usage and rural and remote area connectivity such as across water bodies
- Mesh networks for disaster management, traffic monitoring, and public safety

An endeavor has been made in all the IEEE standards to keep things as similar as possible till the MAC layer. Only the physical layer is different with similarities still being there. A

salient comparison of the features of the latest standards is listed in Table 1.1. The IEEE 802.11ay standard has been mandatory for the Enhanced Directional Multi-Gigabit stations (EDMG).

Table 1.1: Features of high-throughput (Gigasamples per second) IEEE WPAN / WLAN standards

Standard	IEEE 802.15.3c		IEEE 802.11ad		IEEE 802.11ay [3]	
PHY mode	Single Carrier	OFDM	Single Carrier (SC)	OFDM	Single Carrier	OFDM
Sampling Rate	1.76 GHz	2.64 GHz	1.76 GHz	2.64 GHz	1.76 GHz	2.64 GHz
FFT Block size		512		512		512
Working Environment	Residential		Single room		Single room	
Modulation formats	BPSK, QPSK, 8-PSK, 16-QAM	QPSK, 16-QAM, 64-QAM	BPSK, QPSK, 16-QAM	QPSK, 16-QAM, 64-QAM	BPSK, QPSK, 16-QAM	QPSK, 16-QAM, 64-QAM
Max. data rate (After LDPC)	5.28 Gb/s	5.78 Gb/s	4.62 Gb/s	6.76 Gb/s	4.62 Gb/s	20-100 Gb/s

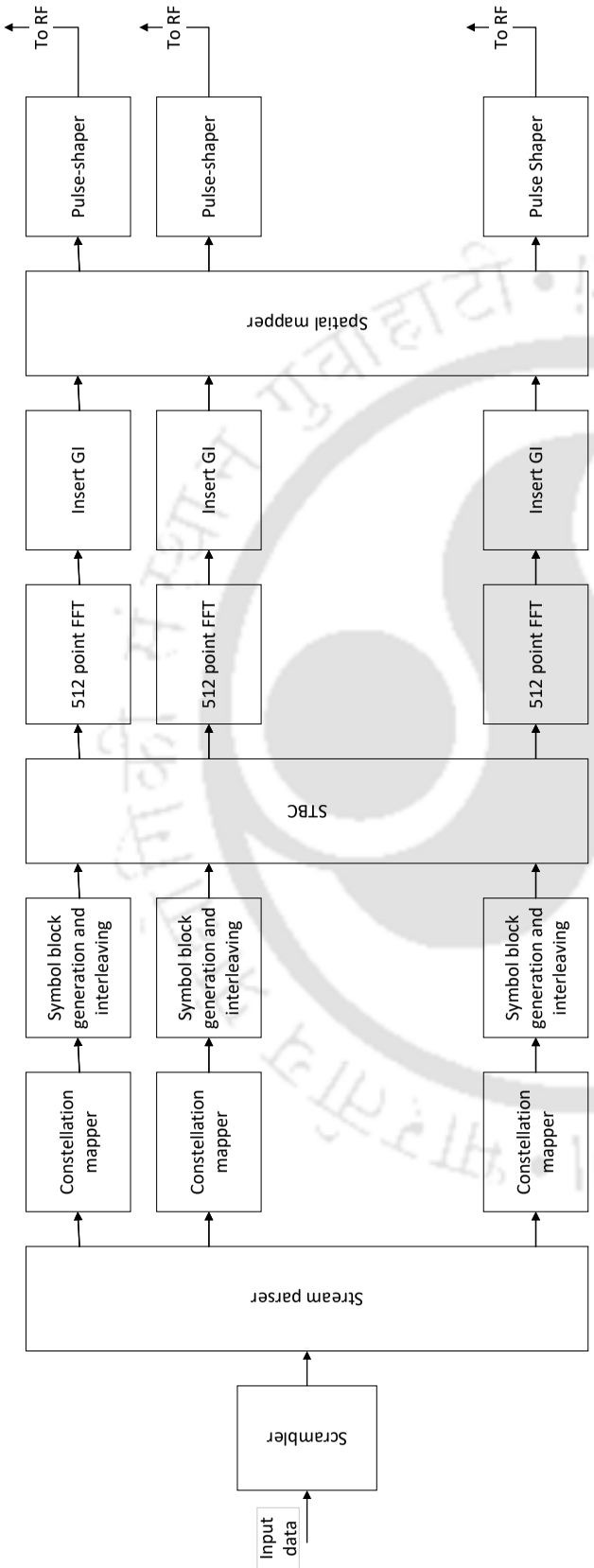


Figure 1.1: A modern OFDM system with multiple FFT blocks

The various blocks in the block diagram of OFDM system as shown in Fig. 1.1 are briefly described below:

- **Scrambler**: Reduces the probability of long sequences of 0's and 1's.
- **Low-Density Parity Codes (LDPC)**: Error-correcting codes are inserted based on parity. Extra padding is carried out to convert the length to an integer number of code-words or SC symbol blocks. This will help to separate the symbol blocks in the next block.
- **Stream Parser**: The encoded bits are separated into blocks as defined by the size in modulation format.
- **Constellation mapper**: The encoded bits are converted into symbols as per the modulation format.
- **Symbol blocker and interleaver**: Symbols from the previous blocks are grouped into SC symbol blocks. Within an SC symbol block, interleaving is done between various symbol blocks to improve burst error resilience.
- **Fast Fourier Transform (FFT)**: FFT is performed on blocks of symbols.
- **Space-time block codes (STBC)**: The various symbols are combined with STBC encoding for MIMO transmission.
- **Guard interval (GI)**: GI symbols are inserted after a few blocks of symbols as defined by the standard.
- **Spatial Mapper**: Space-time streams are mapped to transmit chains.

1.1.1 OFDM: A divide-and-conquer approach against multi-path fading

The advantages of Multi-carrier modulation over Single carrier modultion are highlighted in Table 1.2. OFDM was proposed way back by Chang [4] and [5] as a form of transmission which

could combat the effects of multipath fading. Chang suggested the division of the spectrum into orthogonally related frequency bins followed by modulation and then added before transmission over a band-limited channel. If the frequency bins are realistically orthogonal to each other, the data can be recovered at the receiver using a bank of correlators. In the frequency domain, it can be viewed as multiplying each sinusoid separated by symbol rate, by the symbol's value, and then adding them up.

Table 1.2: Hardware complexity of Single-Carrier vs Multi-Carrier Modulation

Single-Carrier	**Multi-Carrier**
Easy transmitter design	FFT block makes design of transmitter moderately complex
No option to remove multipath fading	Several key hardware blocks present to tackle multi-path fading
Poor resilience to errors	Much better resilience to errors

The OFDM transmission model is shown in Fig. 1.2.

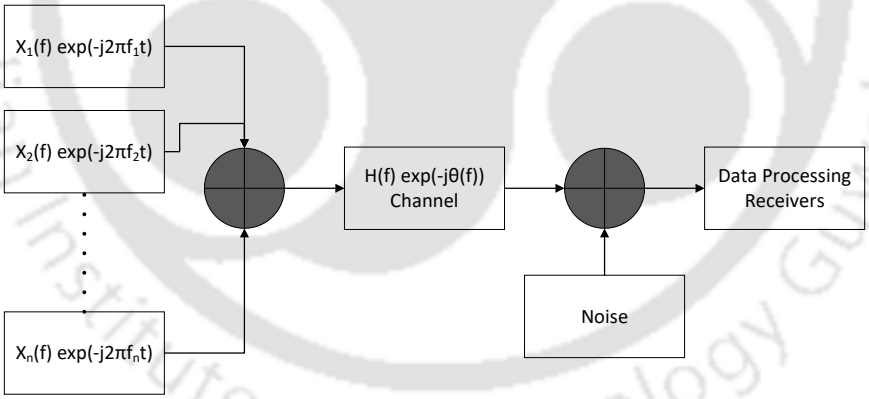


Figure 1.2: OFDM Transmission Model

The fundamental principle used in OFDM is that increasing symbol duration decreases the effect of multipath fading. One OFDM symbol duration is N times the bit-duration, where N is the number of symbols multiplexed together to create the OFDM symbol. Summing N such bits results in an OFDM signal.

The major problem with this scheme was that the data would require an inverse Fourier Transform to be done at the transmitter end. This was difficult to achieve in practice due

to the fact that getting perfect synchronization for a large number of channels or frequency bins was a difficult task in the analog domain. Then came to rescue an idea by Salz [6], which proposed the use of the Discrete Fourier transform (DFT) to digitally realize the addition of the orthogonal subcarriers modulated by data. It also suggested that the DFT could be realized by the use of FFT. This idea was further reinforced by Weinstein [7] who gave mathematical support and presented the design of equalizers for a receiver that used FFT as the OFDM modulation vehicle.

1.1.2 OFDM through FFT: Mathematical formulation

OFDM can be realized through FFT as shown by the series of equations below. The basis function for OFDM is a series of time-limited orthogonal exponential signals. The equation below suggests one such series

$$\Psi_k(t) = e^{(j2\pi f_k t)/N}, k = 0...N-1 \tag{1.1}$$

OFDM is a combination of such Ψ_k multiplied by the modulating symbols such as 64-QAM etc.

$$x(t) = \sum_{l=0}^{\infty} \sum_{k=0}^{N-1} X(k)\Psi_k(t - lT_s) \tag{1.2}$$

where $X(k)$ are the modulating symbols drawn from a constellation of 64-QAM etc. Expanding the basis functions and then sampling the equation at $t = nTs$, we get

$$x(n) = \sum_{l=0}^{\infty} \sum_{k=0}^{N-1} [X(k)e^{((j2\pi(n-l)k)/N)}], \tag{1.3}$$

where l indicates OFDM symbol number

or

$$x_l(n) = \sum_{k=0}^{N-1} [X(k)e^{((j2\pi nk)/N)}],\ l = 0, 1, ...\infty \tag{1.4}$$

This is the equation of IDFT, which can be realized using FFT.

1.2 Fast Fourier Transform: Algorithms and Architectures

The Fast Fourier Transform (FFT) is a faster way of doing a Discrete Fourier Transform (DFT). The FFT algorithm uses several properties of the DFT to reduce complex multiplications and additions, inevitable if direct DFT computation is performed. In order to compute an N-point DFT for each output point, N points have to be multiplied and added. The idea is to divide the N input points by an integer called radix r of the FFT algorithm, and hence to use only N/r points in the computation of a single DFT point. These N/r points can be iteratively processed in a similar fashion until only r points are needed to compute a DFT point. Each such iteration leads to formation of a stage in the FFT algorithm. A higher radix FFT algorithm requires more complex processing elements. Its advantage lies in the reduction in the number of stages that need memory access. Higher radix architecture reduce the number of memory accesses. For e.g. in a radix-16 FFT for 512 points, 512×3 read accesses and same number of write accesses are needed for complete processing. Comparatively, in radix-2, 512×9 read and write operations are performed on memory. Memory accesses being power consuming, a higher radix algorithm is generally considered more efficient than its lower radix counterpart. Further power and area savings are achieved by reducing the number of twiddle factor multiplication.

Hence, there are many FFT algorithms based on radix.

- Single Radix
- Mixed Radix
- Split Radix

They are characterized by the difference in the memory access patterns, twiddle factor multiplication, input and output reordering capability, etc. For single or mixed radix, the radices are shown in Fig. 1.3.

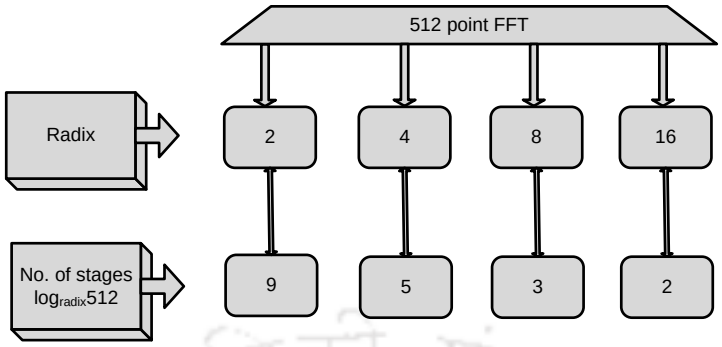


Figure 1.3: Number of main stages for different radices of FFT algorithm

There are two types of FFT architectures common in the literature. Most FFT designs are variants of either of these two designs.

- Pipelined FFT
- Memory-based FFT

Both these designs require different amounts of memory and processing elements. Pipelined architectures are FIFO based, which limits the random access capability as compared to memories. Hence, for high radix algorithms, memory-based architectures provide better flexibility in storage and access of data, which is a fundamental requirement in such algorithms. A typical memory-based FFT design is shown in Fig.1.4.

The increase in throughput has led to an increase in sampling rates as well as the FFT block size. The block size has increased from 64 in 802.11g to 512 in many of these standards. The throughput rates are 2.64 Gsps or even more up to around 20 Gsps. Realistic clock frequencies are in the range of 200 to 300 MHz. For an assumed clock rate of 264 MHz,

$$\begin{aligned} No.\ of\ cycles\ for\ complete\ processing\ of\ 'data &= 512 \div \frac{sampling\ rate}{clock\ rate} \\ &= 512 \times \frac{2640\ Msps}{264\ MHz} \\ &= 51.2 \end{aligned} \tag{1.5}$$

This indicates that 512 samples must be processed in about more than 50 clock cycles. Hence use of a high radix algorithm would be preferred. If we use radix 16, the number of simul-

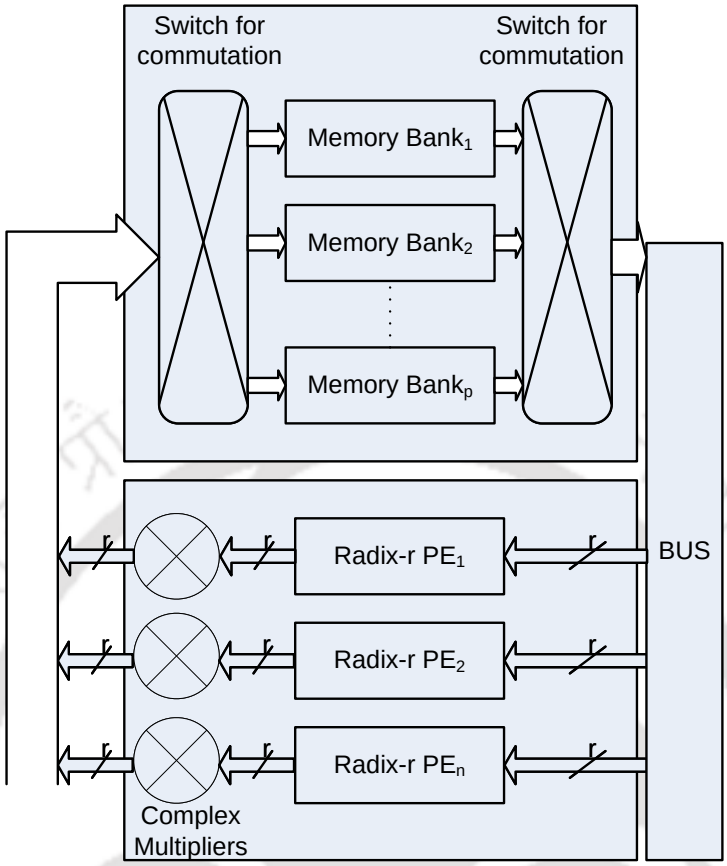


Figure 1.4: General Structure of memory based FFT architecture

taneous data processed is 16, requiring a minimum of 32(=512/16) clock cycles for complete processing. In this case, we assume a 16 parallel processing capability with a fully pipelined architecture. This work proposes an architecture that completes well in about 32 clock cycles. The architecture can be operated at 300 MHz to achieve a data rate of 4.8 Gsps.

1.3 Background and Related Work

High throughput FFT processors have been in demand for several OFDM applications ranging from Digital Video Broadcasting -Terrestrial (DVB-T), Ultra-Wide Band (UWB) to WLANs and Optical OFDM. The earlier designs were targeted for UWB communication. The 128-point processor design reported in [8] achieves a throughput of 1 Gsps using mixed-radix

Multi-path Delay Feedback (MDF) architecture. [9] have also implemented a 128-point radix 2^4 processor with an improved speed of 1.8 Gsps. In a later WPAN standard 802.15.3c, the number of points has been increased to 512 and throughput to about 2 Gsps. [10] proposes a processor design that achieves this speed with radix 2^4 MDF architecture. It uses radix 2^4 split operations, which breaks down the radix-16 process into multiple radix-2 operations. These radix-2 operations are realized through the number of radix-2 PEs denoted by the term parallel datapath. The design is a three-stage one with 8-datapath and three PEs, one at each stage, as is common in pipelined architectures. [11] presents a processor design of the same throughput, which is suitable for MIMO applications. The FFT core still uses two PEs but the complexity of PEs comes down to 4 datapaths. One drawback of increasing throughput by parallelizing the pipelined structure is that memory is segmented into smaller chunks. One memory, segmented into two, occupies almost 1.5 times the original area. The parallel datapath approach leads to further division of memory, which significantly increases the memory area. For a throughput of 2.4 Gsps, [2] proposes a 512-point, radix-16 FFT based on a single memory and PE. This design enjoys a lower total area compared to the previous pipelined designs. The method in [12] uses 2^5 pipelined architecture with some algorithmic modifications leading to a lower number of multipliers and hence, a reduced area for the core. Compared with [2], the design in [12] achieves savings in the core area but no claims are made on areas of memories as such. The work in [13] proposes a radix 2^3 pipelined processor with an 8-parallel datapath. Due to lower radix, the number of stages increases. The area required is slightly on the higher side, as they have increased the pipelining to achieve a higher clock frequency of 500 MHz for improving the throughput to 4 Gsps. The throughput is doubled without much increase in area with the help of increased clock frequency but this will result in additional power consumption. Also, a lower technology node is required to achieve this clock frequency, and architectural modifications alone may not suffice.

Some architectures improve performance by using the discontinuity in the input data flow to reuse resources during idle cycles. [14] presents a very high throughput design with a processing speed about 12 times higher than the best existing data speeds. They use an eight-datapath

design where a single datapath can handle 2.4 Gsps. The clock frequencies have been increased by around 1.5 times from 300 MHz to 430 MHz using additional pipelining and retiming. The design attempts to save area by sharing multipliers. It borrows clock cycles from the idle clock cycles during data loading to use the shared multipliers. But in CF processors, there cannot be any idle clock cycles for input loading, and saving area using idle clock cycles is not possible. Hence, even though this design possesses high throughput, it is not designed to handle the continuous flow and is therefore not suitable for high throughput WLAN applications. [15] proposes a programmable Application Specific Instruction Processor (ASIP), which can process FFT of varying lengths from 16 points to 4096 points. This design uses two memories switching between each other to allow CF. It even uses a radix-16 PE with modifications to allow it to be also used as a radix-2,3,5, etc. Because of this programmability, the size of both the PE and supporting structures increases. Another disadvantage of the design is that it would require three full-sized memories to achieve CF with normal order output. This design is an example of a trade-off between the area of design and the flexibility of design. In [16], the processor is said to have CF capability, but the necessary addressing or switching schemes are not discussed anywhere. Moreover, the design uses radix-4, which is unsuitable for high throughput designs. The output stage requires additional full-sized memory to achieve normal order. In [17], [18], and [19], high performance processors are designed. However, neither of them meets the 1 Gsps throughput. The work reported in [20] is a 512-point processor with 3.08 Gsps performance, but the data flow is not continuous and falls below the required throughput for Wireless Gigahertz standards.

1.3.1 Conflict free memory access

Data has to be stored in the proper memory banks so that the required data is available when necessary. This is known as conflict-free addressing (CFA) and has been a classic topic of research in FFT. CFA refers to single clock access to all the data required at the r inputs of a radix-r PE. The first step is to split memory into at least r banks. The second requirement is the storage of data in such a bank so that data required simultaneously are stored in different

banks for the respective stage. There are two situations:

(i) In place storage where the memory feeds all the stages of the FFT.

(ii) The memory bank feeds only one or two stages of the FFT algorithm.

Most of the algorithms published in literature deal with the first case for radix 2 [21] [22] [23] [24] [25]. For higher radices,the realizations based on XOR or modulo additions [24–27] are useful. Permutation based solutions are proposed by [28, 29]. While [23] uses a simultaneous solution of constraints based on in-place addressing concerns for radix-2, Tsai [24] derives it for higher radices in a more elaborate way and the architecture is amenable to modifications. Richardson [26] adds up to the theory by presenting schedules for pipeline cases. The problem with pipeline cases is latency which makes the read and write time to a memory different, requiring care that the data is only written after it has been read out. Xia [25] takes up a more comprehensive view and presents results for parallel PEs. The structure for this type of design uses a bank and address generator, a cross switch for moving the bank and generated address to the inputs of a multiplexer unit, and is shown in Fig. 1.5. Hsiao [27] uses modulo-radix addition instead of XOR logic as his formulations deal with higher radices. In [30], the CFA scheme uses four times the bandwidth for a standard memory bank while other banks remain idle, leading to an inefficient design. The design in [16] uses a memory based architecture with CFA. The CFA used by them looks similar to the one used in our proposed design but differs in fundamental issues. This design assumes that data is stored serially first in bank 1. Only after that bank 2 is loaded and so on. But this would require large number of cycles just in loading input data and may not be a good strategy for real-time communication. Apart from this, the mathematical logic presented is inadequate to extend it to similar cases of interest.

For the permutation type, Reisis [28] presents an elaborate permutation scheme based on a look-up table for multiple memory cases. Such cases arise in high radix or high throughput designs. The advantage of this technique is that the hardware is much simplified as there are no bank generators, only address generators. The flow of data is well defined and can be taken care of by permutation switches. [29] proposes a similar technique using an additional memory

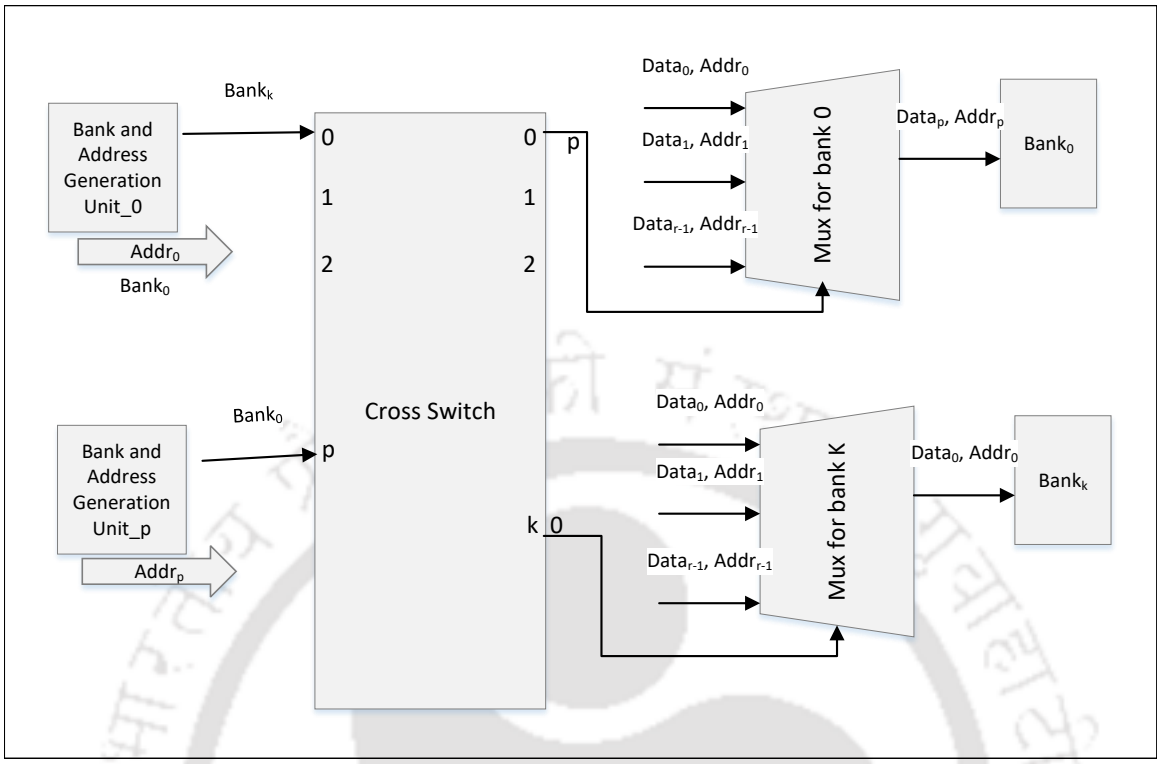


Figure 1.5: Switching unit for conflict-free addressing using modulo addition or XOR technique

to design a natural order reordering circuit at the FFT output. The application areas of these techniques differ. The XOR techniques are generally applicable for all cases but lead to more complicated switching circuits. Further they assume that a single memory is used to feed all the stages. This switch has to be replicated for all the memories. Hence, in a multiple memory situation, switches also would consume a large area. Since, the XOR or modulo addition technique leads to unpredictable bank allocation, continuous flow techniques may become more difficult to apply. On the other hand, permutation-based designs are useful for cases where one memory feeds only one or two stages, and they result in much simpler circuits as shown in Fig. 1.6.

1.3.2 Continuous flow processors

CF for two memory architecture is easily done in single radix FFTs as output and input are related by bit or digit reversal. Hence the address map alternates between only two states. Chen [29] and Wang [31] both use this property of bit and digit reversed algorithms to design

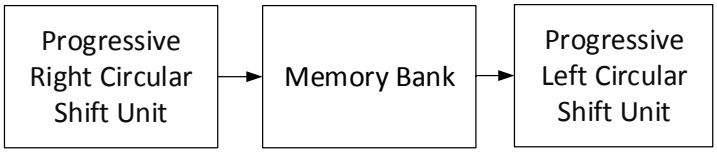


Figure 1.6: Simpler switching circuits for permutation-based conflict-free addressing

architectures to realize CF. Mixed radix FFT lacks the property of input to output symmetry. Hence, alternate addressing is not directly possible here. Baek [32] and Jo [1] suggested a type of bit-rearrangement for mixed radix so that the output map becomes symmetric and alternate addressing becomes feasible. But the scheme discussed applies to that particular architecture only. The design proposed by Yang et al. [33] supports continuous flow but is applicable only in the case of multiple streams, such as in MIMO. Luo et al. [34] discusses a merged memory bank structure for reducing conflicts in addressing rather than allowing parallel access for continuously feeding several simultaneous inputs. Xia [25] restricts itself to a single clock and increases parallel processing to finish computation in time. There are several high-speed applications where the data rate is in Gigahertz, and the processor clock is a divided clock. In Xing [35], the CF scheme is not for mixed radix. Hsiao [27] discusses schemes for continuous flow for prime factor FFTs. If multiple PEs are used, each PE would need to process different radices, which is not efficient. In work by Mao [36], a real-valued FFT is designed with CF. This work is an extension of [37]. The technique used for CF with two memories for two radix-2 PEs in a single stage is a modification of the output dataflow graph. The success of the design lies in being able to modify the input flowgraph so as to reduce a stage of memory access. The output map is then reordered to be the same as the input map by requiring an extra stage of memory access, but it is limited to radix 2. In [38] and [39], the design has been further modified, but the essential CF part is left untouched. The work by [30] assumes a constant geometry approach. CFA is achieved by using a recently popular method where multiple data are stored in a single address location. Since a number of data are read simultaneously, data selection has to be done using multiplexers and routed to proper PE inputs using alignment

circuits. Thus data is read from memory A, but it is not immediately replaceable by real-time input data because the read-out data is not fully utilized and needs to be reread. CF with two memories is hence difficult in this case. Long [40] discusses a circuit for both CFA and CF. But the logic behind CF is not provided, making it difficult to verify its claims. Gyan [41] discusses an efficient circuit for continuous flow output reordering for pipelined circuits. This design takes care of bit-reversed or digit-reversed cases. In their design, mixed-radix CF is not possible. Apart from these, they have suggested breaking down the memory into $\frac{N}{P^2}$ banks, where N and P represent the number of FFT points and parallelism, respectively. For a modest size of $N = 512$ points and parallelism P=16, this represents a bank of size two, which is inefficient. In Wang, [42] a landmark technique is elaborated, which is applicable for any size of parallelism using only four single port memories. Since the number of memories in the bank is limited and single port hence the savings in the memory area is truly remarkable. The work is focused on establishing a conflict-free addressing approach. The CF technique used here is to make the storage in the output stage resemble the input stage. This technique is shown to be one of the subsets of the more generalized techniques developed in our work.

There have been two distinct approaches for CFA in the literature. The first is based on finding the bits in the data index, which would help in identifying a bank for it [21], [23], [22]. However, all these schemes explored the radix-2 architecture. To achieve higher throughputs, in [24], a high radix scheme is proposed using parallel butterflies. On similar lines, [26] derived schedules for CFA. The scheme is elaborated for the case where multiple butterflies operated in parallel in the same stage. It also took pipelining delays into account to take care of folded or pipelined butterflies and hence, different read and write times for a set of data. In [25], the work of [24] is extended, and CFA schemes for the cases of both prime factor FFT and common factor FFT are presented. Various constraint sets are derived and based on them, banks and addresses are generated using XOR or modulo-r addition.

In all these cases, the bank number is generated using XOR or modulo addition logic for a particular data index. It isn't easy to decipher any definite pattern. This randomness necessitates movement of not only data but also "row address" from some input port of a switch

to some other output port, resulting in a complex switching mechanism. The other addressing scheme is designed by Reisis [28], where the concepts of forward permutation and reverse permutation are proposed. However, rules or guidelines for a hardware-efficient permutation are not discussed anywhere. Additionally, it assumed an independent memory for each stage. It dealt with the particular case of one memory per stage, and no solution is provided for the case when a memory fed more stages. Hence, this permutation technique has limited applications. Hsiao [27] focused on modulo addition and suggested a form of permutation but did not go into any study on its applications or theory. Chen [29] used a similar technique for the output stage for the simple case of power-of-two radix implementation, but they did not deal with other stages. For real FFT realization, Ma in [37] suggested a procedure to fill up the entire memory bank sequentially rather than filling up multiple banks simultaneously. Memory bandwidth usage is inefficient, as other banks are unused during the input or output phases. This inefficiency is later taken care of in [36] at the cost of extra exchange stages to attain a bit-order as required for CF. The CFA scheme from the classic work of Johnson [23] is used. Hence, these works did not truly add anything new to CFA, although they came up with a more efficient architecture for real-valued FFTs. In [39], an XOR-based scheme is derived based on the postulates and the basic principles as suggested by Johnson with some modification. The design in [38] proposed a CFA for real FFT. The scheme is explained for a 32-point FFT. The derivation of the CFA scheme is not provided. The implementation used counters to store data depending on the count, but in effect, the switching complexities remained similar. In [43], mixed radix algorithms are discussed. The design considered only a single memory without banks. Address generation is discussed to obtain the operands. However, the architecture did not use parallelism and is a low-throughput structure. Long [40] reduced the complexity of the CFA circuits developed by [44], which in turn is an improvement of [23]. The advantage of this design is the lower complexity of implementation compared to [23]. However, the design is limited to radix-2 and radix-4; additionally, the complexity increased when the number of memories or PE is increased. The work in [15] is on a flexible number of FFT points. They used a single PE with 16-way parallelism. The CFA scheme is challenging, as the PE had

to cater to a large variety of points. The number of memories used for CF is three. The presented scheme is directly based on [27] and used a sophisticated switching mechanism for data and addresses. In [16], a storage pattern that initially looked similar to progressive shifting is suggested. However, it is inefficient because it used two multibank memories to realize CFA, whereas two memories could be used to realize both CFA and CF simultaneously [1]. In [42], Wang followed a new trend of using single port memories. The CFA challenge increased tremendously in this design, which is solved using modulo addition techniques. Although the design saved a significant area in memory by replacing the dual port with a single port, the actual advantage is offset due to the use of more complicated CFA circuits.

1.3.3 General FFT processors

The work in [45] provides a review of the 50 years of FFT algorithms and applications. [46] outlines a survey on FFT/IFFT Processors for Next Generation Telecommunication Systems. In [47], a new type of cordic multiplier is proposed to reduce memory footprint. In [48], a 24Gbps design is proposed for 802.11ad. Reference [49] suggests a cost-efficient and high-precision VLSI design for fixed-point reconfigurable FFT processors. [50] presents a less complex and memory-efficient algorithm using pipelining for a large-FFT processor based on the FPGA. [51] proposes scheduling techniques that lead to high speed and reduced area requirements for Mixed-radix MDC FFT processors. [52] outlines a low-overhead fault-tolerant parallel-pipelined FFT design by using a combination of modified, reduced precision redundancy (RPR) and error correction codes (ECCs). [53] discusses a reconfigurable multiplier-less and ROM-less rotator design for Mixed-Radix FFT Processor. This has resulted in low area and power consumption. [54] devises algorithms based on variable word length in each stage of FFT computation. The proposed algorithms provide a trade-off between hardware utilization and performance. [55] proposes a fast and hardware efficient FFT design using the CORDIC algorithm for OFDM WPAN applications. [56] designs a cost-effective Bit-Slice Butterfly Processing Unit, which processes data continuously for 64-point rapid single-flux-quantum (RSFQ) FFT Processors.

Mahdavi in [57] designs a low latency FFT architecture for MIMO-OFDM utilizing guard

bands to reduce computation. In [58], Zadnik develops a programmable processor which works below 1 Gbps but is great at low power operations. In [59], Wang designs a 1024 point FFT with hybrid floating point and fixed point architecture. The research by [60] focuses on an analysis of the bit truncation process in the FFT algorithm for OFDM systems. [61] designs an 8-parallel 128-2048/1536-point FFT processor for the 4G LTE system. It contains a radix-3 unit to increase the reconfigurability. Regarding memory access techniques, Kitsakis in [62] introduces an efficient technique for parallel data addressing in FFT architectures performing in-place computations. Chen in [63] designs a variable-length FFT and evaluates its performance at 1 GHz. Kim [64] designs a low latency SDF architecture with low memory in the first stage using an input reordering approach. In [65], scalability issues in FFT computation are brought out in detail. The paper by [66] proposes a design for 802.11ax, which has a variable symbol duration. In [67], Han proposes a design for approximated small-point FFT architecture. Such designs are not very useful in communication but do represent an increased interest in approximate FFTs. The design by [68] is an FPGA hardware of 512 points where the focus is on the reduction of complexity by using modified CSD multiplications. Tang et al. [69] designs an FFT hardware with low area for multistandard MIMO-OFDM applications using the guard interval in an intelligent way. In [70], Lin et al discusses the architectural optimizations of a low-complexity and low-latency FFT processor for MIMO-OFDM Communication Systems. The application of FFT to radar signal processing incited interest in many researchers. In [71], the design and development of an FPGA-based FFT co-processor for synthetic aperture radar (SAR) is discussed. [72] proposes an FPGA-Based Four-Channel 128k-point FFT Processor with high accuracy, reduced hardware resources, and high performance meant for Spaceborne SAR. [73] designs a low Complex zero-padded FFT Processor using single-path delay feedback pipelining for Radar Applications. Chan, in his paper, [74] implements a 4-parallel 64k-point FFT processor with low storage requirements for the THz imaging radar system. [75] discusses a low complex architecture of FFT Processor supporting variable-length operation for Multi-mode Radar Signal Processing. [76] implements an FPGA Implementation of FFT Processor that supports windowing, accumulation, and magnitude/phase calculations

along with FFT in hardware for FMCW Radar Signal Processing. This results in high performance and low resource requirements. The work in [77] discusses a pipelined FFT processor for zero-padded signals using the fact that several twiddle factors are multiplied by zero and can be avoided to reduce area. [78] derives all optimum multi-path delay commutator (MDC) FFT Hardware Architectures in terms of Delays and Multiplexers through analysis of orders at the FFT stages. [79] surveys various parallel and serial architectures on pipelined FFT and presents a comparative analysis. [80] outlines the design of a pipelined FFT processor using polyphase decomposition. [81] proposes a low-power twiddle factor addressing architecture for split-radix FFT processor. [82] presents a design and implementation of systolic array FFT Processor based on shared memory. [83] demonstrates a 47.8 GHz high speed, low power FFT Processor using single flux quantum technology. [47] proposes a high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms. [84] discusses a scalable split-radix 4/8 FFT algorithm for high throughput applications.

1.4 Problem formulation for the proposed FFT processor design

It is proposed that when memories have fixed roles, memory design may be optimized both in terms of word length and speed of operation for that particular stage. It is this idea that forms the basis for the proposed design. The following design guidelines have been framed to meet this objective:

(i) A memory should feed those stages where word length requirements are similar.

(ii) Higher radix operations reduce the number of stages, hence reducing the complexity of memory access circuits.

(iii) Memory should not switch roles. A dedicated role will help to save area in switching circuits, which can be significant for larger radices.

Hence, to keep them fully utilized, input memory has to be fed with input data only; the other memories have to be handled in a similar way. Hence, a novel replacement-based CF

scheme would be required in place of the switching one. Therefore, in this work, we first briefly propose the principles of CF and then utilize them to derive our circuit architecture. Also, a "Circular Shift" technique of CF is proposed, which can be used in those places where the main data flow is not to be disturbed. The Circular Shift technique uses slightly more area but makes changes only to the addressing logic while leaving the data flow undisturbed. We use this technique here to demonstrate its flexibility compared to re-ordering one.

OFDM works in the continuous flow mode. Hence, the input to the FFT block has to be buffered while the other memory set is being used for storing intermediate computation results. Then, the output of the FFT processor is not ordered in the natural order. Hence, another set of memory would be required to buffer up all the output data and then send the output in a natural order. This necessitates three sets of memories in the FFT processor. The incorporation of these memories in the design of the core FFT block gives way to further optimize the design.

1.4.1 Proposed Continuous flow technique

It may be possible to achieve continuous flow using only two memories under the following circumstances. When the memory A is filled up with input data, it is further used for computation data. During this time, the input data is fed to memory B. When memory A is ready with output data, it starts sending out output data in a natural order. These vacant memory spaces are filled up with input data coming in serially, while memory B starts getting used in the computation. This input data is now at a different memory location, originally occupied by the output data. Hence, an addressing scheme is needed to find the required input data in this new memory map. Several papers have addressed this issue for the special case of those algorithms which permit bit or digit reversed addressing at the input with the other being addressed in natural order or vice-versa. Although the scheme looks attractive at first sight and it is indeed good for general FFT applications, which require continuous flow and natural order output, it suffers from some subtle disadvantages.

- It may be advantageous to use different word lengths on the various memories. This optimization, which may significantly reduce the size and increase SNR, will not be possible

if only two sets of memory are used. Always, the higher word length memory of the three, viz. input, output, and computation, will have to be used.

- Multiplexers are required to switch the inputs to the memories and various units. Although this does not seem significant for a design with a single datapath, for multi-data path designs, it may lead to the use of a significantly large number of multiplexers for switching data, read address, write address, etc.

- Both the memories will need to function at two clock speeds, one related to computation and the other related to sampling. If the two clock frequencies are significantly different by orders of magnitude, as is generally the case in FFT processors for WPAN, two different clock networks will be needed at the clock pins of the banks of memories. This problem may be mitigated by the use of SIPOs at the inputs of the multibank memories and PISOs at the output.

For the case of OFDM, some observations can be made:

- The inputs to the FFT come from a fixed constellation set

- The inputs are uniformly distributed

The first characteristic can be used to reduce the word length of the input memory. The second observation allows us to pre-compute scaling factors at various stages and incorporates them in the design phase itself, avoiding the use of exponent detection units. Several levels of scaling can hence be included, improving overall SNR. Such separation of memories into three banks raises new issues in addressing. These problems form the core work of this research.

1.4.2 Proposed Conflict free technique

To overcome the above-mentioned limitations of the CF technique, in this work, we propose a progressive shifting permutation, which is applicable for cases where one or two stages are fed by one memory set. The proposed technique avoids the complicated XOR-based switching mechanism presented earlier. Moreover, as the shifting mechanism in progressive shifting is

regular, shifting of addresses is eliminated, thus making the switch simpler. Hence, the proposed scheme is more efficient than XOR-based schemes.

1.5 Organization of the thesis

Chapter 2 "Conflict free Addressing" presents the design of the proposed progressive shift technique for CFA. It compares the existing XOR scheme with the proposed technique and discusses the results.

The design of a new CF scheme, along with its mathematical formulation, is incorporated in Chapter 3 "Continuous Flow Addressing" of the thesis.

Chapter 4 "Continuous Streaming 4.8 Gsps Radix-16 512-Point FFT Processor for OFDM" presents the design and implementation of the chip utilizing the CFA and CF techniques developed in the previous chapters.

Chapter 5 "Physical Design of ASIC" discusses the techniques and tools used in the physical realization of the chip.

Chapter 6 "Conclusion and Future work" presents the conclusion and also summarizes the outcomes of the research work, and outlines possible directions for future work.

2

Conflict-Free Addressing (CFA):Progressive Shifting

Contents

2.1 Introduction

High-radix Fast Fourier Transform (FFT) processors are in great demand today for building high-speed systems such as Orthogonal Frequency Division Modulation (OFDM) based communication standards, including Wireless Personal Area Network (WPAN) (IEEE 802.15c) and Wireless High Definition (HD) (IEEE 802.11ad, IEEE 802.11ay), vehicle and military radars, and medical imaging such as Magnetic Resonance Imaging (MRI). These processors are based on different radices of the FFT algorithm, which are radix 2, 4, 8, 16, etc. Higher radices (greater than 4) are needed for higher throughput, especially for FFT sizes of 512 points and above. For example, realizing architectures for 512 point FFT with higher radices (8 or 16) would only require two or three stages compared to nine stages when using radix 2. The existing XOR or modulo-addition schemes for accessing multiple banks of memory in a conflict-free manner work well for lower radices but for higher radices, the complexity and area consumed increase significantly. Continuous-flow (CF) high-performance FFTs need at least two memories. Additional memories are required to feed more processing elements (PEs) to boost performance. Therefore, in high-performance FFTs, the number of memories is usually two or three or even more sometimes. A Conflict-Free Access (CFA) [22] scheme has to be designed for each memory. Present CFA techniques would lead to a sizeable increase in area. In this thesis, the higher number of memories is used to an advantage. The progressive shifting (PS) technique proposed in this design can reduce the area of CFA circuits by 33%. This technique relies on the fact that when higher radices are used, the number of stages is also limited to approximately 3 to 4. When the number of stages and memories are in the same range, each memory will be tied up with only one or two stages. In such a case, the existing techniques of XOR or modulo addition logic can be replaced with our proposed “progressive shift” (PS) technique, which is more area efficient. But the drawback of the PS technique is that it is not easy to apply in architectures where one memory is used to supply data for several stages of the system instead of one or two. Nevertheless, in all high performance designs, with high radix and multiple memories, its application can make the system more efficient. In order to demonstrate the advantages of the

Table 2.1: A comparison of various techniques of conflict free access in FFT algorithms

Design →	[22], [21], [23]	Tsai [24]	Huang [2]	Tian [16]	Reisis [28]	Jinti [38]	Ours
Bank logic	XOR	Modulo Addition	XOR	Shifting	Permutation	XOR	Shifting
Radix	2	2^k	16	2^k	2^k	2	2^k
Stages per memory ($N = 512$, $k = 4$)	9	3	3	3	1	9	1
Nos. of memory	1	1	2	1	Same as stages	1	3 or more
Continuous flow	No	No	Yes	No	No	No	Yes

proposed technique, it is compared with existing techniques, and the significant differences are outlined in Table 2.1.

The motivation and main contributions of this thesis are highlighted below:

- The progressive shift technique is proposed for FFT architectures with multiple memories and fewer stages serviced by a PE. Such types of FFT architectures usually find application in high throughput FFT designs. The main idea is to reduce the complexity of the CFA circuits, which tend to occupy larger areas as the number of memories increases to improve performance.
- A theoretical framework for the technique is also formulated.
- Progressive shifting and modulo addition/XOR switching are embedded in a single mathematical framework.
- Simulation studies are carried out to show the superiority of the proposed technique in terms of the usage of hardware resources and plotted in the form of a graph.

The proposed architecture finds applications in OFDM based MIMO communication systems that demand high throughput and low area. The proposed multi-bank memory architecture

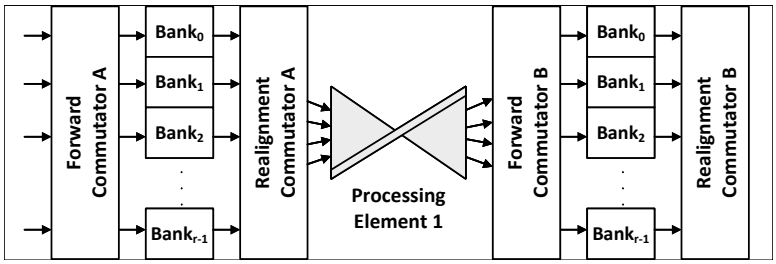


Figure 2.1: A generalized structure for CFA showing switches before and after the memory

can be extended to implement 5G communication systems and Internet of Things (IoT) architectures in the future.

2.2 Conflict Free Addressing

An FFT processor generally uses some form of the Cooley-Tukey algorithm and a butterfly structure for processing data using a PE. The PE needs r inputs and generates r outputs in every clock cycle, where r is the radix of the FFT algorithm. As per the usual requirement of FFT algorithms, data have to be reordered before giving at the input of PE. This reordering is commonly done by first storing the data in a memory and then retrieving it using an address different from the one used during storage. Since one memory can only give one output, to fulfill the need for r simultaneous outputs, r banks are required in a memory. Bank size is taken as N/r so that the total storage in the memory is N, where N is the number of FFT points. The main challenge is the distribution of data to the memory banks such that data required simultaneously by the butterfly are stored in different banks. This problem is called bank generation for CFA and uses switching to reorder the data. The switching circuit is called a forward commutator, which consists of a circuit (BAGU) to generate the necessary bank number and address, and a multiplexer array, which directs the incoming data and address to that bank. Additionally, after the memory, a realignment commutator is needed that realigns the data with the inputs of the PE. The commutators use a large number of multiplexers to do this and hence consume a significant area. Each memory will require its own switching circuitry, as shown in Fig. 2.1. Thus, in FFTs that use multiple memories for real-time continuous flow

Table 2.2: Mathematical symbols used in the chapter

Symbol	Represents
N	Number of FFT points
r	Radix of the FFT algorithm
B	Number of banks in a memory
D	Data-width
S	Total number of stages
q	$log_2 N$

of data, the area for conflict-free switching increases significantly. Hence, developing a CFA scheme with a lower area is a challenging task that needs to be addressed.

In order to overcome the limitations of XOR and modulo addition based CFA, in this thesis, we propose a progressive shifting permutation. This scheme is applicable for cases where one or two stages are fed by one memory set. The proposed technique avoids the complicated switching mechanisms. Moreover, as the shifting mechanism in progressive shifting is regular, shifting of addresses is eliminated, thus making the switch simpler. Hence, the proposed scheme is more efficient.

2.3 Proposed Bank Generation Scheme for CFA using Progressive Shifts

Hsiao [27] outlined a procedure for bank generation based on modulo addition of the characteristic variable of each stage. The mathematics proposed by Hsiao is for prime factor FFTs and cannot be applied here directly to derive the bank generation equations of PS technique. The assumptions of the derivation have to be constrained and hence a few more Lemmas 1 and 2 have been proposed to assist the derivation. This results in a powerful expression which can be used to derive several bank generation techniques as shown subsequently. For ease of reading, a table of important symbols is provided in Table 2.2.

For an N point FFT, let the factors of N be as

$$N = N_1 N_2 N_3 ... N_S \tag{2.1}$$

where each factor defines a stage and S is the total number of stages. We define the characteristic variable n_i of each stage $i = 1, 2, \ldots S$:

$$n_1 = 0...N_1 - 1,\ n_2 = 0...N_2 - 1\ etc. \tag{2.2}$$

The input equation of an FFT algorithm is expressed as

$$n = Xn_1 + Yn_2 + Zn_3 + \ldots \tag{2.3}$$

where the constants X, Y, Z, etc. are defined as

$$X = N_2N_3 \ldots N_S$$

$$Y = N_3N_4 \ldots N_S$$

$$Z = N_4N_5 \ldots N_S$$

The last constant would be 1. This is the method of the common factor algorithm. For the prime factor algorithm, it is difficult to apply the progressive shifting method, as the flow of data is complicated owing to modulo operations ([85], [27]).

Lemma 1. *The input indices to a butterfly have a one-digit difference among them. It is this digit that is used to decide the banks. Here, the digits are* $n_1, n_2...n_S$.

Proof. A stage of FFT is defined by the $n_i's$. Hence, for a particular stage, all other $n_i's$ will remain constant except for the stage in question. Thus, the fundamental definition of stage ensures that for each stage, the different digit is the one that controls the stage. Furthermore, it seems logical to use this digit to differentiate banks. □

Lemma 2. *If a memory is feeding several stages, all the corresponding stage variables must be taken into account while deciding the bank. If any variable is left out, the corresponding stage cannot be fed by the memory in a conflict-free manner.*

Proof. Let only a single-stage variable be used to find the bank. Then, $Bank = n_1$ ensures that for the first stage, the data are in conflict-free banks. For any other stage, there will be at least one situation where all other $n_i's$ are the same, and only the stage variable is different. However, since this variable is not taken into account, the bank for those inputs is guaranteed to be the same, leading to conflict. Hence, all variables have to be considered while determining the banks for a stage, as in (2.4).

$$Bank = f(n_1, n_2, \ldots n_s) \tag{2.4}$$

□

Theorem 1 (Progressive Shifting)**.** *If only one of the $n_i's$ is changed in the bank generation function, it will lead to the CFA scheme of progressive shifting. This shall happen when the function is addition, and it is conducted modulo B, where B is the number of banks.*

It should be noted that usually $B \geq r$, where r is the radix of the FFT algorithm. In the case of $B > r$, instead of continuous progressive shifts, shifts may occur every other cycle or even after more cycles.

$$Bank = (n_1 + n_2 + \ldots n_s) \, mod \, B \tag{2.5}$$

When only one- or two-stage variables are changed, the flow in which data have to be stored follows a simple pattern leading to simplifications in switching commutators. In (2.4), the digits n_1, n_2, etc., are added together to find the banks. For a stage, only one digit is supposed to be different; this ensures that the required inputs go to different banks. In higher-radix algorithms, the banks are large in number, while the number of stages comes down drastically. Hence, the interaction among the stages does not severely affect the flow of data. Additionally, in cases where each stage has its own memory [28] for demanding throughputs or in input or output memories [29], there is virtually no interaction among the stages. Under such circumstances, the switching complexity may be reduced considerably using progressive shifts, as derived below. The derivation of the technique is based on (2.5). If a memory has to feed only one stage, only one stage variable will increase by 1. Thus, the bank for data changes only by 1 unit. This means that the data are stored in the banks in a progressively shifting manner. The data required simultaneously for a PE can be shown in a row of a table as in Table 2.3. Thus, the progressive shift represents increasing the shift amount by one for every consecutive or alternate row. The modulo operation ensures that the data are shifted circularly in a progressive fashion so that the required data are directly stored in different columns, as shown in Table 2.3. The corresponding hardware is proposed below.

Hardware for Progressive Shifting

In this scheme, the banks to which the data are to be shifted are known as priori. Hence, they can be stored in a series of registers, and the permutations can be easily derived by

Table 2.3: Progressive Shifts ($0, r, 2r \ldots$ are the required data indices for the first butterfly operation)

$Bank_0$	$Bank_1$	$Bank_2$			$Bank_{r-1}$
0	1	2	3	...	$r-1$
$2r-1$	r	$r+1$	$r+2$	...	$2r-2$
...					
...	...	...	...	...	$N-1$

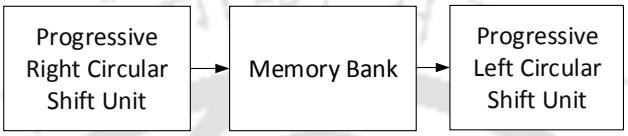


Figure 2.2: Progressive Forward and Reverse Circular Shift Units

appropriately shifting the stored bank values. Then, the registers can be hardwired to the bank selection logic, as shown in Fig. 2.3. This design is more straightforward than the other technique where the bank values have to be made available at any of the multiplexers. The two main components of the proposed architecture are the progressive circular right shift unit (PCRSU) and progressive circular left shift unit (PCLSU), as shown in Fig. 2.2. PCRSU is placed before the memory bank. Its purpose is to increase the input data shift right by 1 whenever a shift is required. A shift may be required after every clock cycle, alternate cycles, or even after a few more cycles. The maximum shift amount is r, assuming a radix r butterfly. To achieve circular progressive switching, multiplexer banks and a shift register array (SRA) of data-width $D = log_2 r$ and length r are used. The multiplexer bank consists of r multiplexers, each with r data inputs of width d (d is the width of the data) and a select input of size D. The r data values from the previous stage are connected to each of the r inputs of all the data multiplexers, as shown in Fig. 2.3. The select logic is designed using SRA. It is preloaded with values $0, 1, 2, \ldots, r-1$. The outputs of the r registers of SRA are connected to the r select inputs of the data multiplexers. On every clock cycle or as required, the data in SRA are shifted by one register. At the start, the selected inputs of data multiplexers are $0, 1, 2, \ldots, r-1$. This leads to data output without any shifts. Another clock edge now leads to the shifting of

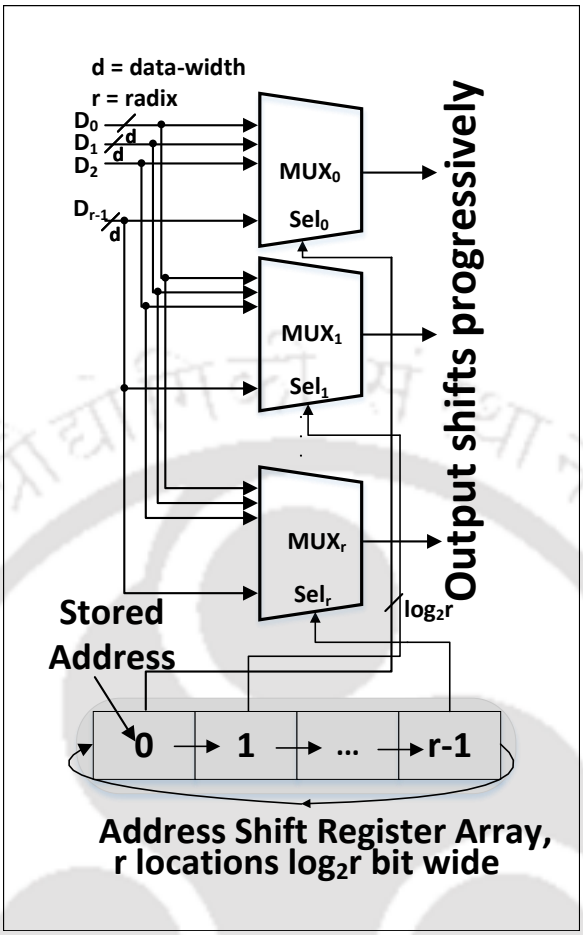


Figure 2.3: Right Progressive Shift Unit

addresses in the register array to the right in a circular fashion. Hence the selected inputs now become $r-1, 0, 1, 2, \ldots, r-2$. Now, the data output of the PCRSU is as if shifted right by one unit. The multiple clock cycle logic for activating the shifting in the SRA can be realized using a counter of an appropriate size. This is demonstrated by considering a 3-bit address as an example. The shift in the output of the multiplexers with respect to the change in the addresses of the SRA is shown in Table 2.4. An example of radix-4 continuous right progressive shifting for 32-point FFT of [1] is shown in Tables 2.5 and 2.6. PCLSU is placed after the memory bank to align the right-shifted data in line with the inputs of the PE. The design of this unit is similar to that of the PCRSU. Only the direction of the shift is changed to be left circular.

Table 2.4: Output of the multiplexers with respect to the change in data of the shift register unit

Shift register unit					Output of multiplexer				
a_{r-1}	a_{r-2}	a_{r-3}	...	a_0	1^{st}	2^{nd}	3^{rd}	...	r^{th}
7	6	5		0	d_0	d_1	d_2	...	d_7
6	5	4		7	d_7	d_0	d_1		d_6
5	4	3		6	d_6	d_7	d_0		d_5
...					...				
0	7	6		1	d_1	d_2	d_3		d_0

Table 2.5: Original digit-reversed output sequence for 32-point FFT when stored serially [1]

$Bank_0$	$Bank_1$	$Bank_2$	$Bank_3$
0	8	16	24
4	12	20	28
1	9	17	25
5	13	21	29
2	10	18	26
6	14	22	30
3	11	19	27
7	15	23	3

Table 2.6: Shifted digit-reversed output data indices for the output bank for 32-point FFT

$Bank_0$	$Bank_1$	$Bank_2$	$Bank_3$
0	8	16	24
28	4	12	20
17	25	1	9
13	21	29	5
26	2	10	18
22	30	6	14
11	19	27	3
7	15	23	3

2.4 Bank Generation using XOR Logic

In this section it is shown that the bank generation equation of the previous section are more general in nature and can even be used to derive the existing XOR based scheme of [23]. Here, the XOR based bank equation of [23] is derived with the help of equations formulated for progressive shifting showing the superiority of the mathematical approach of the previous section for bank generation.

For radix 2 decomposition, the digits n_i's of (2.5) can be replaced by bits b_i's.

$$Bank = b_1 + b_2 + b_3 \ldots + b_S \tag{2.6}$$

Single-bit addition is equivalent to an XOR operation. The result of XOR defines the parity of the data index. It is thus shown that the two methods are related by a common mathematical framework. The differentiating factor is the constraints imposed on the number of b_i's that are allowed to change simultaneously.

2.5 CFA for 512-point radix-16 FFT

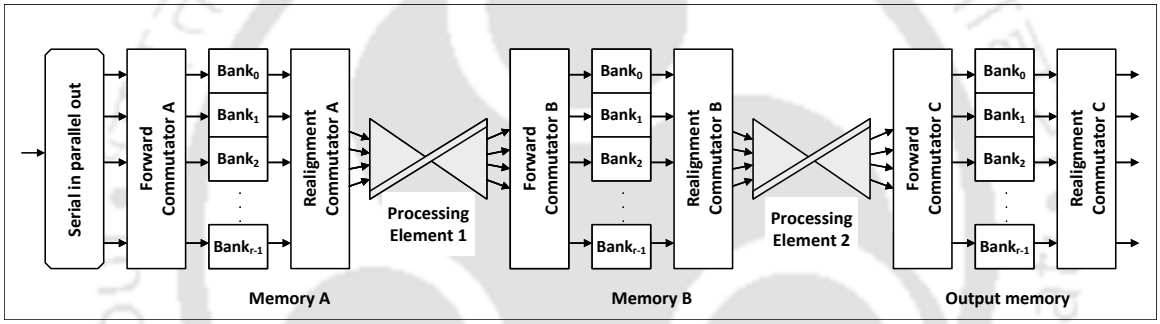


Figure 2.4: Architecture for 512-point FFT showing memories and CFA logic

A 512 point FFT chip is designed in Huang [2] using a radix-16 algorithm. The architecture used two radix-16 stages and a radix-2 stage. In our design, an additional PE is introduced in the pipeline for stage 2 processing to double the throughput. The new architecture is shown in Fig. 2.4. The design is thus modified so that the input memory feeds only PE_1, and there is a second memory to feed PE_2. An output memory is finally used to obtain the outputs in the natural order. This circuit is aptly suited for the progressive shifting technique of CFA. The hardware for this design consists of several subunits, as described in 2.3. These are detailed in Table 2.7. For XOR-based addressing in higher and mixed radix FFTs, the results of [24] can be used. The equation derived by Huang is reproduced in (2.7) for reference:

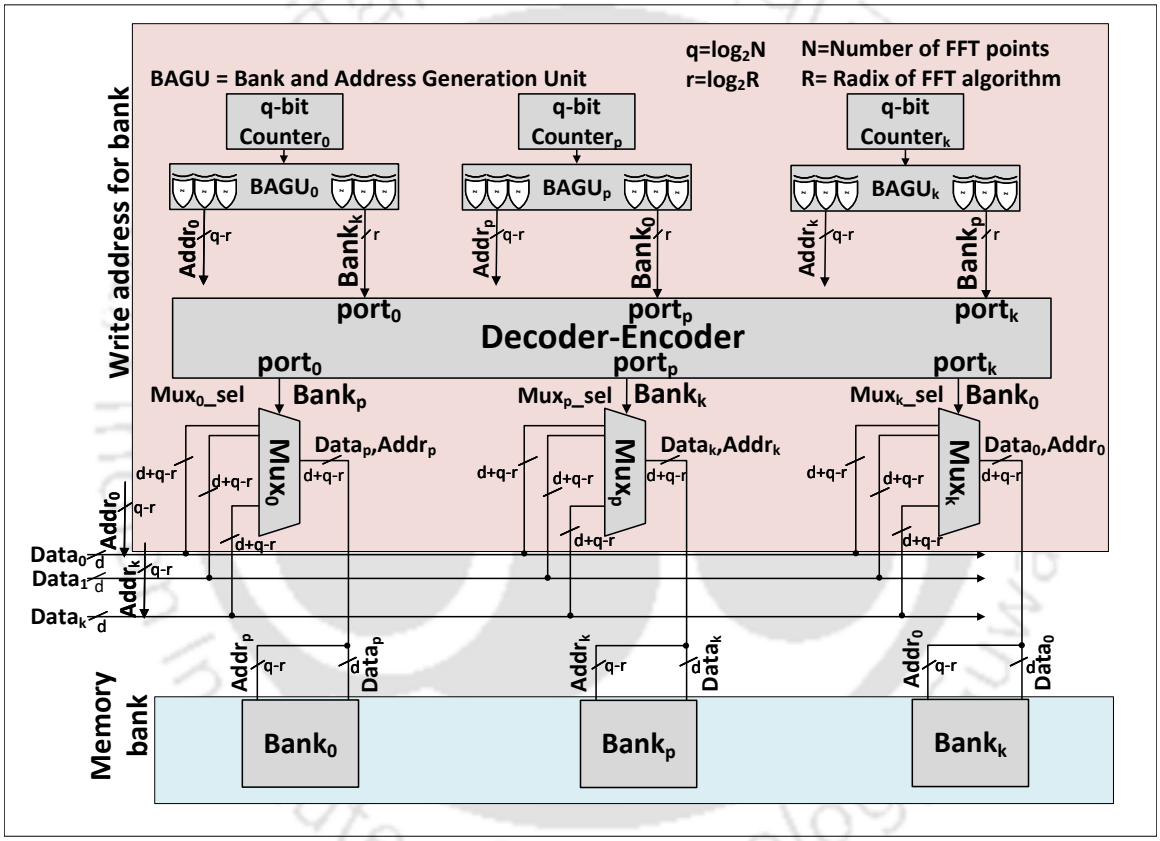


Figure 2.5: Forward commutator in CFA using XOR-based logic

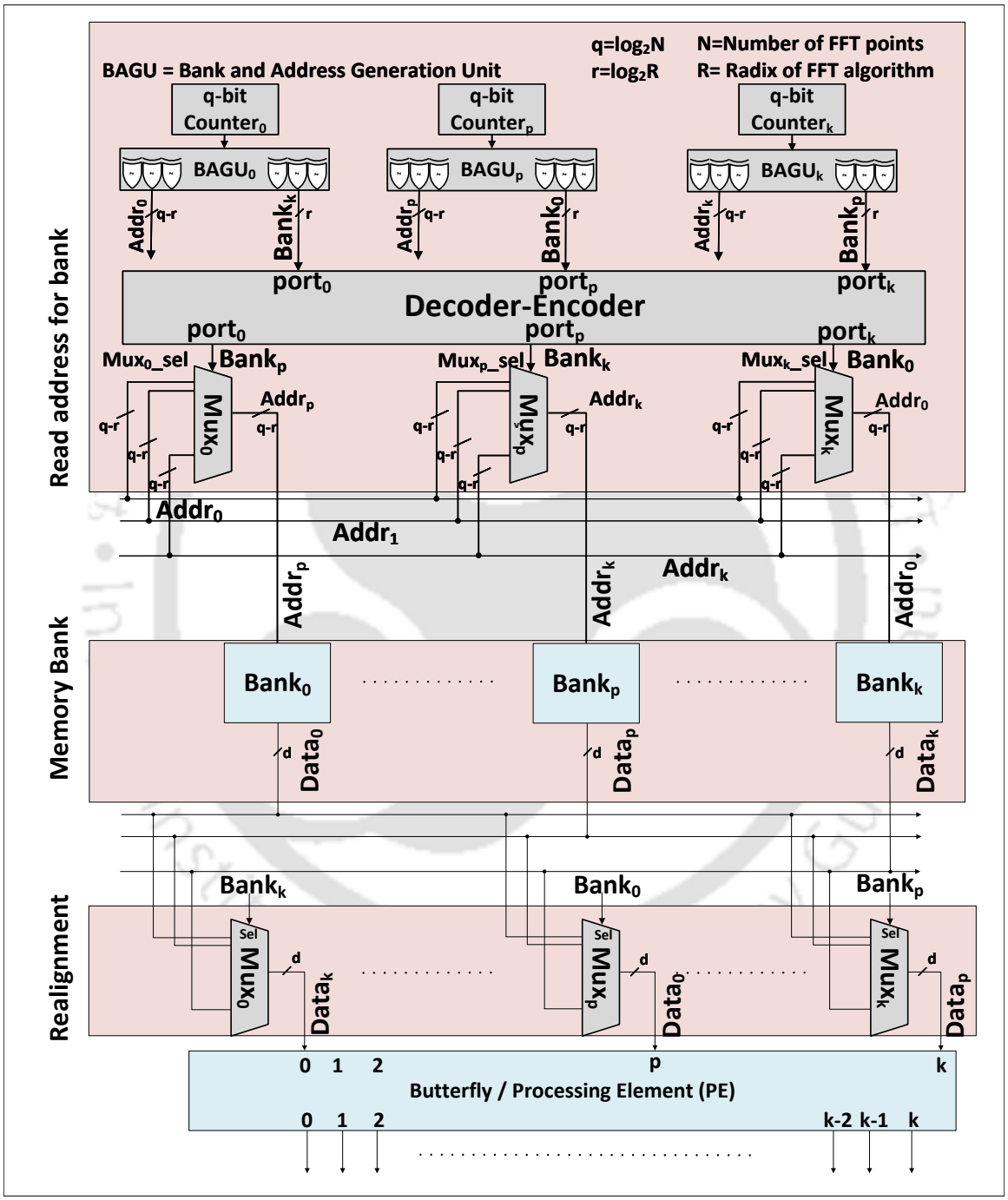


Figure 2.6: Reverse commutator for CFA using XOR-based logic

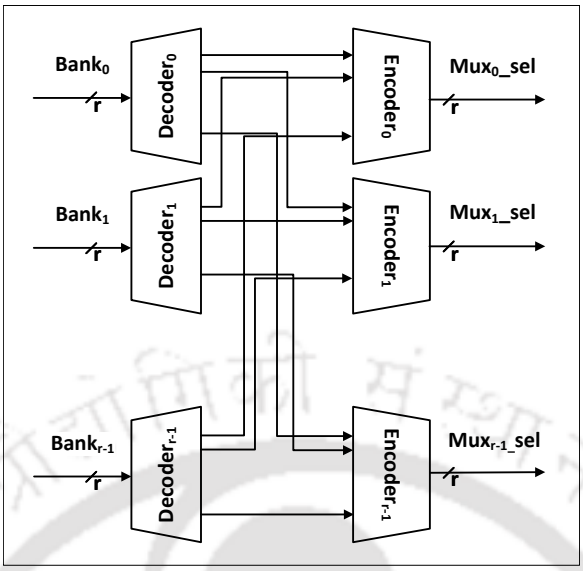


Figure 2.7: Decoder encoder used as a switch in XOR-based CFA

$$Bank = (8(b_5 \oplus b_1) + 4b_0 + 4b_8 + ((b_8b_7)_2 + 2b_6 + (b_4b_3)_2 + (b_2b_1)_2 + (2b_5 - 1)(b_5 \oplus b_1))mod\,4)mod\,16 \quad (2.7)$$

The bank generation logic defined in (2.7) shows that it is not possible to identify the bank simply by inspection. In Huang, the circuit is implemented using XOR logic, but details are not presented. We redesign and implement the circuit using Verilog and analyze it for the purpose of comparison with the PS technique. The detailed design of XOR based forward commutator is shown in Fig. 2.5. The first step is to generate the index of the data using q-bit counters with offset adders, where $q = log_2N$. This data index is then fed to a bank and address generator unit (BAGU). A BAGU implements (2.7) using XOR gates. In Fig. 2.5, $Bank_k$ is calculated by $BAGU_0$. This means that $Data_0$ is to be stored at $Bank_k$. Hence, the multiplexer at k^{th} port must have the selection input as $Bank_0$. For this transfer, a unique switching unit called decoder-encoder switch is required, which transfers data from the input port to the output port in such a way that if the input is $Bank_k$ at $port_0$, then the output at $port_k$ is $Bank_0$. Additionally, the address generated by $BAGU_0$ is for $Bank_k$, and it has to be transported to

that bank. Thus, this design requires switching of both data and address, hence requiring extra multiplexers for address switching.

To read data from the banks, a reverse commutator as shown in Fig. 2.6 is used. The reading order is determined by the FFT algorithm. BAGUs determine the banks and addresses at which the requisitioned data is available. The decoder-encoder switch transfers this information to the select inputs of an address-select multiplexer, and the address is directed to the proper bank. The read data have to be realigned with the inputs of the PE. A second array of multiplexer bank is used to achieve this, where the select input is the bank generated by the associated BAGU. Thus, CFA is achieved with the help of both the forward and reverse commutator, the former placed before the memory to write data and the latter placed after the memory to read data. The decoder-encoder switch is made with an array of decoders and encoders, as shown in Fig. 2.7.

2.6 Synthesis results and comparison

The design implementations of both the XOR-based technique and the proposed PS-based technique are carried out using Verilog. The correctness of the designs is verified through simulations carried out using Synopsys VCS. For synthesis, an SCL Chandigarh foundry node of 180 nm [86] is used, and synthesis is carried out using Synopsys DC.

Table 2.7: Calculation of the area for radix (r=16),(N= 512) points using the SCL 180 nm library for (d=12) bit wordlength

Unit	Sub-unit	XOR			This work			
		Complexity	**Specification**	**Area (μm^2)**	**Complexity**	**Specification**	**Area (μm^2)**	Savings (%)
Address generator	**Counters**	$Rlog_2N$	9 bit counters: 32 nos.:16 each for read and write	28800	$Rlog_2\frac{N}{R}$	5 bit counters: 32 nos.: 16 each for read and write	10100	65
	BAGU	$Rlog_2N$	32 nos: 16 each for read and write	20800	N/A	N/A	N/A	N/A
Multiplexer Unit	**Mux Array**	$R^2(d + log_2\frac{N}{R})$	2 nos: 1 each for forward and realignment commutator	364400	R^2d	2 nos.: 1 each for forward and realignment commutator	268000	26
	Select logic	R^2log_2R	Decoder Encoder: 2 nos: 1 each for forward and realignment commutator	48000	$R^2log_2\frac{N}{R}$	Shift Registers: 2 nos.: 1 each for forward and reverse commutators	23000	52
Buffers		$R(d + log_2\frac{N}{R})$	Inserted at multiplexer outputs by synthesis tool	24800	Rd	Inserted at multiplexer outputs by synthesis tool	23800	4
Total Area				**486800**			**324900**	**33**

Table 2.8: Comparison of hardware complexity using the SCL 180 nm library for 12 bit wordlength for $N = 512$ points

Metric	XOR	This work (PS)	Savings (%)
Area $(mm)^2$	0.49	0.33	**33**
Power (mW)	22	17	**23**
Execution Time (ns)	7.4	4.5	**39**

Additionally, various parts of the addressing logic are synthesized separately to evaluate their relative complexity, and the corresponding results are presented in Table 2.7. The table shows that the proposed scheme uses 65% fewer counters, 26% fewer multiplexers, 52% less selection logic, and achieves an overall 33% area efficiency, compared with the XOR-based scheme. The area, power, and execution time are obtained from the synthesis result and are shown in Table 2.8. In terms of power, the proposed scheme is 23% better, and the timing performance is 39% faster. Complexity analysis is performed to study the dependence of the area of logic elements on radix R, the number of FFT points N, and wordlength D, and the results are tabulated in columns 3 and 6 of Table 2.7. Simulations are carried out for different numbers of FFT points with variable radix sizes of 8, 16, and 32 to determine the scalability of the proposed technique. The results of the simulations are tabulated in Table 2.9 and plotted in the graph shown in Fig. 2.8. (The subscript r in XOR_r and PS_r represents the radix of the FFT). From the table, it is clear that as the number of points increases, the savings in hardware by using the proposed PS over conventional XOR-based design improves. For both schemes, as radix increases, complexity grows in an exponential manner, but the PS scheme remains more or less 30-40% efficient for all radices.

To estimate overall savings on silicon area, it must be observed that one CFA circuit is used per memory. Hence, for an FFT with continuous flow using two memories, there are two CFA circuits. In order to compare the performance, the 512-point CF FFT chip is designed using both PS and XOR based techniques. The results show that an area of 3 mm^2 is needed for PS and 3.35 mm^2 for XOR based design. It is also observed that the CFA logic portion of the proposed architecture occupies 0.65 mm^2 area while that of the XOR based design occupies 1

mm^2 area. Hence, it can be concluded that the area savings of the CFA logic and the complete chip of the proposed architecture is 35% and 10%, respectively, compared with XOR based architecture.

Table 2.9: Comparison of hardware complexity for various FFT sizes (N) and radices (r) for d-bit wordlength (d=12)

Radix	r=8				r=16				r=32			
FFT points, N→	256	512	1024	2048	256	512	1024	2048	256	512	1024	2048
Scheme ↓	Area (mm^2)											
This work (PS)	0.090	0.092	0.094	0.096	0.318	0.325	0.332	0.338	1.187	1.209	1.232	1.254
XOR-based	0.135	0.143	0.152	0.161	0.458	0.487	0.515	0.544	1.658	1.758	1.857	1.957
Savings (%)	**33.3**	**35.9**	**38.2**	**40.3**	**30.6**	**33.3**	**35.6**	**37.8**	**28.4**	**31.2**	**33.7**	**35.9**

From the above observations, it is inferred that due to the following features, the proposed scheme outperforms the XOR-based bank generation scheme:

- Bank and address generation logic is simpler.
- Since the address for a bank is generated by its linked address module, multiplexers for address shifting are not required.
- The decoder-encoder switch is replaced with an SRA, which is less complex.

2.7 Conclusion

CFA circuits have to be repeated as many times as the number of memories in the FFT architecture. Hence, they do contribute to the overall area and complexity of the design. The progressive shifting presented here is an efficient technique; in a typical case, it saves approximately 33% of the area in commutation circuits. It is best suited for FFTs with a high radix and a lower number of stages or for architectures with multiple memories. Additionally, designs with a memory dedicated to a particular stage of PE can take advantage of this technique. On the other hand, this scheme is not easy to use for designs in which one memory feeds more than

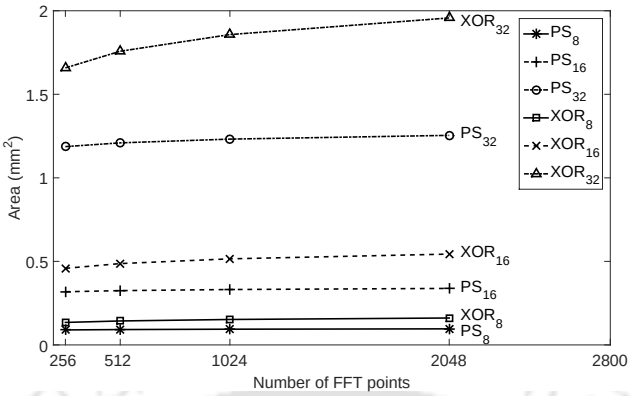


Figure 2.8: Comparison of area for XOR-based design vs. progressive shifting for different numbers of FFT points

two stages, as in low-radix architectures. More investigation is necessary to adapt it for such cases. Another advantage of this design is that it does not change much from one architecture to another, facilitating its reusability once designed.

Comparatively, the other addressing techniques are mathematically rigorous and require the design of individual addressing logic for different FFT architectures and need to start from scratch each time the architecture changes. In conclusion, for high-throughput FFTs, the progressive shifter is expected to result in substantial benefits in terms of area and design effort.

3

Continuous Flow Techniques: Reordering and Circular Shift

Contents

3.1 Introduction

Real-time applications such as communication, radar signal processing, artificial intelligence-based scene recognition for vehicles, etc., require the data flow in a continuous manner. FFT plays an important part in the signal processing of such applications. Traditionally, a single memory is used to compute FFT using the concept of in-place computations. Same memory is used for storing the initial input data, the intermediate computation results, and finally, the output data. However, more than one memory is required when the FFT data is to be processed continuously. In Fig.3.1, a second memory is shown whose purpose is to store the incoming data, while the first memory is used for intermediate results. In many designs, such as the one reported in [87], a third memory is also used to reorder the output to the natural order. These memories help to achieve both Continuous Flow (CF) of data and improve the overall performance of the processor. New addressing schemes are developed in this work to handle CF in the multi-memory FFT architectures.

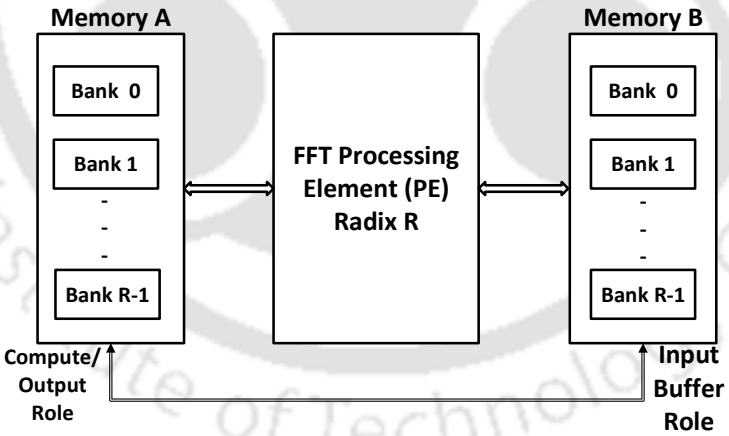


Figure 3.1: Two memories switching roles as needed for continuous flow of data in FFT

The existing techniques in the literature survey as presented in Chapter 1 can be seen to have the following shortcomings:

(i) The existing scheme is applicable only to a particular architecture. Only a single solution is discussed out of many possible solutions.

(ii) In most cases, reordering of data at output stages is addressed but not at the input or intermediate stages.

(iii) In the existing data reordering techniques, an increase in wordlength leads to an increased size of reordering buffers and multiplexers, which makes it inefficient for large wordlengths. No alternative to data addressing is available in the literature.

(iv) The absence of a mathematical theory in the previously reported schemes makes it difficult to extend them to complex cases.

This thesis proposes two general CF techniques for FFT processors. The key contributions are as follows:

(i) **Alternate Addressing (AA)** scheme is proposed for FFT architectures through reordering of data. It is based on the mathematical axioms of group theory and gives several possible solutions for both single or mixed radix FFTs. The scheme is applicable to all stages of FFT, viz. input, output, or any intermediate stage, even with multiple memories. It is shown that the existing techniques form a subset of this scheme.

(ii) The other proposed **Circular Shift Addressing (CSA)** scheme is applicable to a wide range of single and mixed radix FFTs using address rearrangement instead of data rearrangement. The mathematical derivations are again based on group theory.

(iii) A **512 point FFT processor** is designed for OFDM transmission with CF, which is 70% area efficient compared to existing designs.

The proposed AA scheme reorders the data in a generalized way so as to be applicable to many FFT architectures. It uses buffers to store data for reordering. When data word size is limited, the buffer area is also small, and this scheme performs well. In designs where data reordering is not allowed, neither the existing schemes nor this scheme can be applied. Examples are intermediate stages of FFT and legacy designs because rearrangement buffers introduce latency leading to a disruption in the entire timing and state machine of the design,

requiring extensive redesign. The other proposed scheme, CSA, is applicable in such cases. CSA uses an addressing table, and new addresses are generated from this table without the need for data rearrangement buffers. It has the additional property that the area requirement of the CF circuits does not increase with increased wordlength. On the other hand, when the number of FFT points is very high, address storage in CSA consumes a larger space compared to the rearrangement buffers in AA. Hence both schemes may be applied to a given architecture, and the one which gives better results may be opted for.

3.2 Continuous flow FFT architectures for real-time applications

In real-time FFT, data flow is continuous, and hence block processing is not suitable. For discontinuous flow, a single set of memory is used to process the entire FFT. During its processing, the memory is under use by the FFT processor; hence if the input is to be buffered during this time, extra memory is required. For single radix designs, CF is easily achieved by using two memories alternately, one in input/output (I/O) mode and the other in computation mode, and switching the roles after each block of data [29]. Hence, a minimum of two sets of memory is required for real-time operations as depicted in Fig. 3.2. For mixed radix FFTs, rearrangement of input data [1] is required before shifting data into memory. Also, depending on the architecture, a third memory may be required to send out the outputs in normal order [29]. Switching memory roles and high throughput designs add new issues, which are discussed in this section.

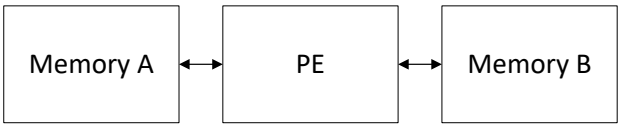


Figure 3.2: Continuous flow FFT needs at least two memory

At one point in time, one memory Mem A, functions as output memory, while the other one, say, Mem B functions as computation memory. So, in using two sets of memory, we are

faced with the problem of storing the incoming data at this point in time. The only possible solution is to store the incoming data at locations emptied by the outgoing stream. This solution, although seems attractive, creates a new problem to address because the output order is different from the input order. Hence, the input data sits at addresses different from the ones used initially and would need a different addressing scheme. Since data is stored in place, this problem resurfaces when it is time for sending out the output stream in natural order, i.e., the output addressing also needs to be changed. Hence, a minimum of two addresses for both input and output are required. In this section, it is shown that the output memory cannot be avoided for real-time systems. Since the FFT algorithm needs a **block** of data to start processing and does its processing in several stages, one set of memory (Set A) becomes busy in computation after acquiring input. During this time, another set of memory (Set B) is used to prevent the loss of real-time incoming data. In a continuous flow operation, Set B would be used for computation once Set A operations are over. The time allowed for Set A operations is hence equal to the time in which Set B gets filled up.

Time for filling up the B memory (assuming B contains locations for N data),

$$T_B = \frac{N}{D}$$

where T_B is the time required to generate N data points at the input, assuming a data clock of D symbols per second.

Hence, time allowed for Set A operations is

$$T_A = T_B$$

If output data is not in the required order, the data is stored in output memory and sent out in the required order by using a different addressing scheme from the one used during storage or computation. On the other hand, when the output data generated in the last stage is in the required order, it can be sent out directly without storage. It is shown here that this cannot be done in a real-time system. In both the scenarios, the fundamental principle of real-time

systems, as stated in (3.1) must be fulfilled:

$$Output\ data\ rate = Input\ data\ rate \tag{3.1}$$

Since a typical FFT processor would require several stages of memory access, the number of clock cycles required would be much greater than N. To finish the processing in time T_B would then require a clock frequency much higher than D. Let us call it S, the system clock.

Then,

$$T_B = \frac{N_C}{S},$$

where N_C is the number of cycles required to generate all N outputs.

For a 4096 point FFT at a data rate of 16 MegaSamples per second (Msps) using a radix-2 algorithm and a single radix-2 butterfly,

$$\begin{aligned} N_C &= Number\ of\ stages \times Number\ of\ butterflies\ per\ stage \\ &= log_2 4096 \times 4096/2 \\ &= 12 \times 2048 \\ &= 24576 \end{aligned}$$

$$\begin{aligned} T_B &= \frac{4096}{16M} \\ &= 256\,\mu s \end{aligned}$$

Therefore,

$$S = \frac{N_C}{T_B} = \frac{24576}{256\mu s} = 96\ MHz$$

This example shows that the clock frequencies required for input data and for processing are different. If we use the system clock for generating the output data, the output data will be generated at a much faster pace (S Msps) than required (D Msps).

A possible solution could be to run the last stage at the D/r clock (the divide by r is to account

for the fact that each clock produces r samples). But this would mean DN seconds for the last stage, leaving no time for other stages since the total time allowed for processing is $T_B = DN$. Hence, the only solution left is to store the output data in memory and send it out at the clock rate of D at the end of the computation. The output is sent out from memory Set A after the end of the computation period, T_A. This process would need time equal to ND, which is again equal to T_A. During this next T_2, memory Set B is used at clock frequency S, as the computation memory. Since, Set B is to be used at clock S, and also it is continuously storing results of computation, it is not possible to use this memory for storing incoming data during T_2. Memory Set A, which is the output memory, also stores the input. At the end of I/O, the memory changes its role and now functions as computation memory repeating this for every block of data. The two memories have to work at different clocks. The I/O memory works at the data clock, while the computation memory works at the system clock, as shown in Fig. 3.3. Hence, the clock also is switched after every block of data.

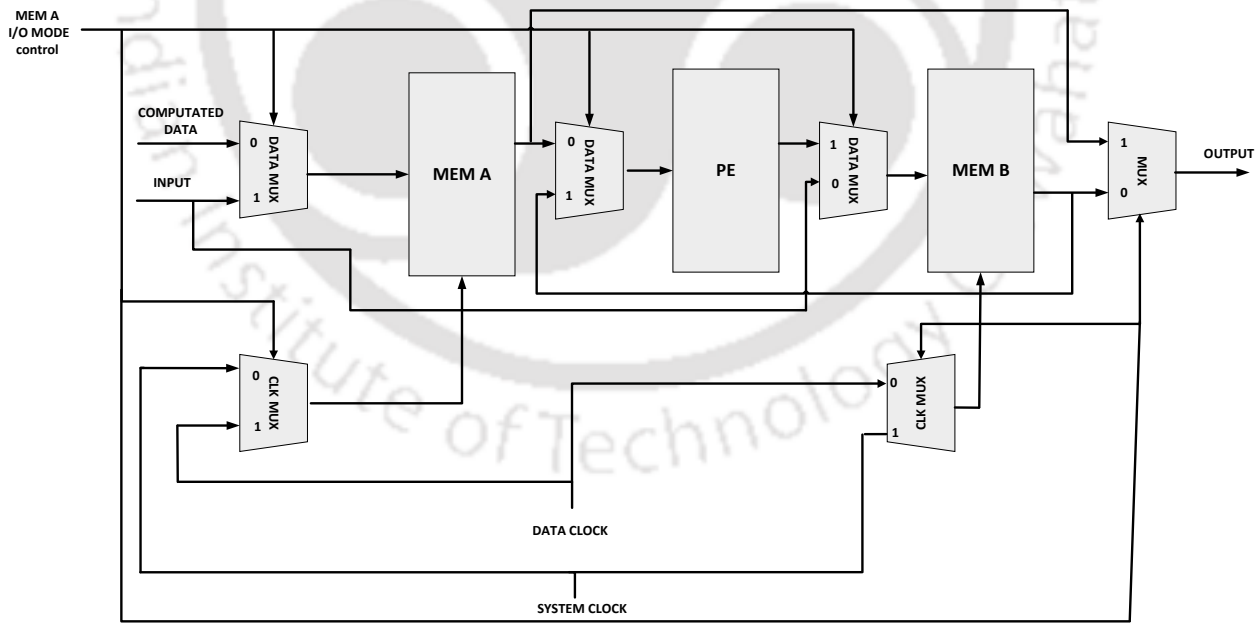


Figure 3.3: Data and system clock interchange for two-memory continuous-flow architecture

When the output is sent out, the input data can be stored in the vacated locations. Since the output addressing order is different, the new order in which the input data is stored is hence

different in this time block. This effectively means that in the third time block T_3, memory Set A would again be used for computation but with different addressing logic than the one used in time block T_1. Such an addressing scheme is easy in the case of single radix and usually fulfilled automatically by the use of bit or digit reversed addressing [29]. In the case of mixed radix, both Sunwoo [1], and Tsai [24] suggest the solution as reordering of the outputs. This technique is not generalized to multiple memory cases. Also, it modifies the flow of data and may be difficult to use in all cases. More research is required in this area to evolve better techniques for multiple memory systems.

3.2.1 Multiple memory designs

In most designs, only two memories may suffice, as suggested by Sunwoo [1] and Tsai [24]. The bottleneck in using multiple memories are as follows:

(i) Higher area: In practice all designers would like to avoid multiple memories to save area.

(ii) More complexity: In general, designs with multiple memories tend to be more complex until special measures are taken to reduce it.

If parallelism or higher radices are being used to finish the computation in time, then additional PEs may be required, which would mean either more data ports in existing memory or an additional set of memory.

Multiple sets of memory (greater than two) in an FFT processor have the following advantages:

(i) Allows separation of input, computation, and output memory. It is a matter of fact that in many applications, input data, computational data, and output data have different requirements of wordlength. The use of three sets of memory will allow to choose different wordlengths, thus optimizing the storage.

(ii) Clock switching is avoided. Optimum memory designs are possible since different storage can be designed using different techniques.

(iii) Extra memory allows the use of additional PE for almost doubling the throughput. This extra PE would need an extra multiplier unit for its twiddle factors along with extra ROM. Sometimes, the different stages of the algorithm have different complexities in their PEs, usually differing in twiddle factor multiplications. Hence, the additional PE may use constant multiplications, which do not even require ROM. In such designs, throughput is doubled without doubling the area.

3.3 Proposed Scheme: Alternate Addressing

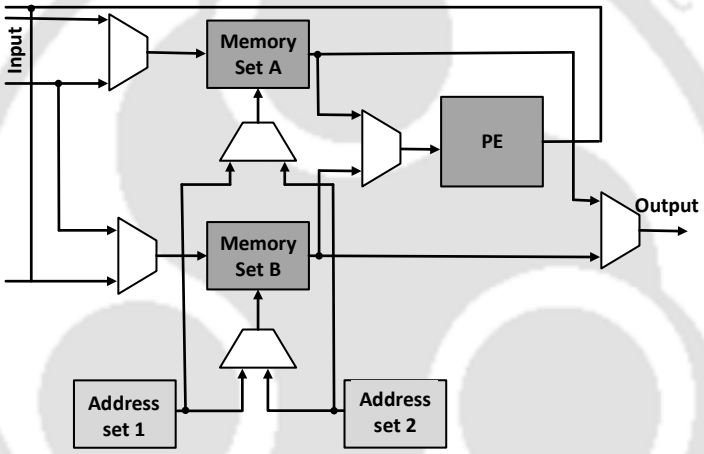


Figure 3.4: Switching the roles of memory after every symbol

FFT processing of N points can begin after a block of N data values has been stored in N locations of a multi-bank memory Mem_A using address set $Addr_1$ as shown in Fig. 3.4. Mem_A becomes busy for several more cycles exchanging data with PEs for processing various stages. After processing all the stages, Mem_A changes its role to output memory, and the address set is then changed to $Addr_2$ to read outputs. At every clock cycle, after reading the output, the read memory locations are no more useful and are filled up with input data. But the input data is now stored at $Addr_2$. Thus, for the second block of N data, the locations of the input data have changed. A new set of addresses would be required to retrieve the data required for processing. It is desirable that $Addr_1$ should be able to do the job so that only two sets of addresses would be required alternating between each other. This is possible by properly reordering the data

before feeding it to memory. The scheme achieves CF with only two alternating addresses and is hence referred to here as Alternate Addressing (AA) and is shown in Fig. 3.5. When Mem_A performs I/O, Mem_B is switched in for storing intermediate PE results and exchanges role with Mem_A after every block of N data.

Data reordering for AA is an additional penalty which is inevitable in all existing mixed radix CF schemes as they are mostly based on some form of AA. This will require around 4 buffers of the order of the size of the radix (R) and R numbers of a 4 to 1 multiplexer.

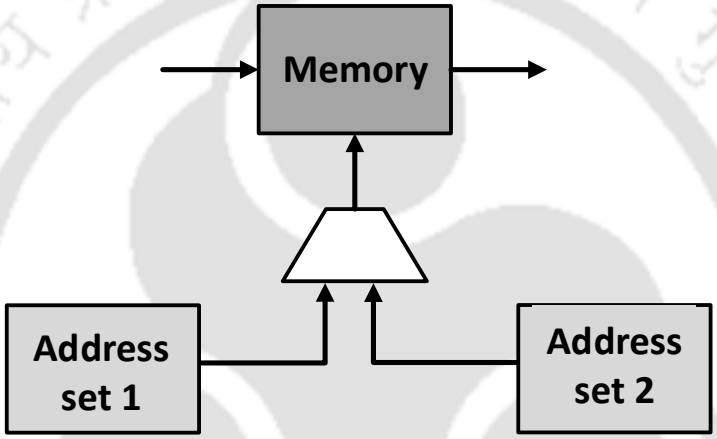


Figure 3.5: Switching the address generators after every block of N points for N-point FFT: Alternate Addressing

Let y_0 be a natural number. Then it can be represented as a weighted sum of bits.

$$y_0 = p_{n-1}2^{n-1} + p_{n-2}2^{n-2} + \ldots p_0$$

where

$$p_i \in \{1, 0\},\ i = 0 \ldots n-1$$

y_0 can be compactly represented using sequences of bits

$$A_0 = \{p_{n-1}, p_{n-2}, \ldots, p_0\}$$

Again let y_1 be another n bit natural number formed by permuting the bits of A_0. Its sequence

can be written as

$$A_1 = \{q_{n-1}, q_{n-2}, \ldots, q_0\}$$

where $q_i \in \{1, 0\}, \forall i = 0 \ldots n-1$.

Let $\mathfrak{R}$ represent a function representing the replacement of bits of y_0 with the bits of y_1.

$$\mathfrak{R} : A_0 \longrightarrow A_1 \, such \, that \, p_i \Longleftarrow q_i$$

where $p_i \in A_0, q_i \in A_1, \forall\, i = 0 \ldots n-1$.

The symbol $\Longleftarrow$ represents "Left-hand side is replaced by Right-hand side."

Let the vacated locations of memory, identified by address A_k, are filled with an incoming sequence which is read out using a different address A_{k+1}. An iteration, k, of the processor refers to reading out the contents of the memory while simultaneously writing to the vacated locations using address A_k. Several iterations of the processor may be viewed as continuously replacing A_k with A_{k+1} and can be shown as

$$A_0 \xrightarrow[iter\ 1]{\mathfrak{R}} A_1 \xrightarrow[iter\ 2]{\mathfrak{R}} A_2 \xrightarrow[iter\ 3]{\mathfrak{R}} A_3 \ldots$$

or more compactly as

$$A_{k+1} = \mathfrak{R}(A_k) \tag{3.2}$$

Lemma 3. *The sequences A_k form a symmetric group under the operation of permutation.*

Proof. A_k forms a group under the operation of permutation because it satisfies the four properties of the group, which are Closure, Associative, Identity, and Inverse, which is a standard proof in group theory. □

Lemma 4. *Whenever (3.2) is applied iteratively, it can be shown that the maximum value of k for A_k to be the same as A_0 is $k = n$, where n is the number of bits.*

Proof. Lemma 3 states that $\mathfrak{R}$ forms a permutation group. From group theory, we know that this series of replacements can be represented in a circular notation. For example, for the 5 bit sequences shown in (3.3), the bit replacements carried out multiple times are shown in Fig. 3.6.

$$\begin{aligned} A_0 &= \{a_4, a_3, a_2, a_1, a_0\} \\ A_1 &= \{a_0, a_2, a_1, a_4, a_3\} \end{aligned} \tag{3.3}$$

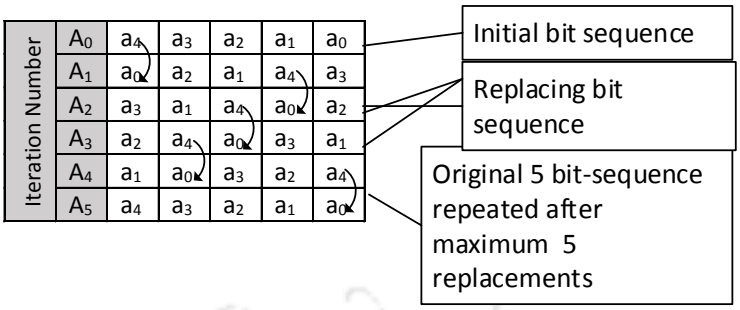


Figure 3.6: Bit replacements for several iterations

$$a_4 \Longleftarrow a_0$$
$$a_0 \Longleftarrow a_3$$
$$a_3 \Longleftarrow a_2$$
$$a_2 \Longleftarrow a_1$$
$$a_1 \Longleftarrow a_4$$
$$\Re = (a_4, a_0, a_3, a_2, a_1)$$

It is obvious and also known from the properties of groups that the maximum number of shifts necessary for a sequence to repeat is equal to the length of the sequence. Since the maximum length of this sequence is n, the maximum number of iterations, k_{max}, for the sequence to repeat itself is n. □

3.3.1 Condition for alternate addressing

In a memory, input data is written using a sequence of addresses B_0, and output is read out using another address sequence as required by the algorithm. For AA, this reading sequence of the algorithm may need to be modified to B_1 so that *Theorem 2*, as stated below, is satisfied.

Theorem 2. ***Alternate addressing is possible if and only if*** B_0 ***and*** B_1 ***are related by bit-wise symmetry*** *as defined below. Let* a_k *be the bit at bit position* k *of the binary representation of* B_0*. The bit sequence of* B_1 *is formed by interchanging the bits of* B_0 *among themselves. Then* B_1 *is said to be bit symmetric to* B_0 *under the following condition:* ***If*** a_k ***of*** B_0 ***is moved to bit position 'm' of*** B_1 ***then*** a_m ***of*** B_1 ***is moved to bit position 'k' of*** B_0**.**

This is shown in Table 3.1.

Proof. In group theory, the composition is defined as the various sequences of which a permutation is composed. Order is defined as the length of the sequence. Let the permutation group

Table 3.1: Symmetric Bit map

Input order B_0	a_n	a_{n-1}	a_k	...	a_1	a_0
Output order B_2	a_n	a_0	a_1	...	a_k	a_{n-1}

for alternate addressing be $\mathfrak{R}_{\mathfrak{AA}}$. AA requires that the sequence should repeat in alternate iterations, that is, in two iterations. Again from group theory, the order of a group is found by the LCM of the order of composition of the group. If k_{LCM} is the order of the group $\mathfrak{R}_{\mathfrak{AA}}$, then applying Lemma 4, we get $k_{LCM} = 2$. Since k_{LCM} has to be 2, the order of the individual compositions has to be factors of 2. Thus $\mathfrak{R}_{\mathfrak{AA}}$ can be composed as the product of permutations of order 2 or 1.

$$\mathfrak{R}_{\mathfrak{AA}} = (a_{n-1}, a_p)(a_m, a_t)(a_0, a_q)(a_r)\, etc.$$

Permutations of order two are known as transpositions, while order one is known as identity. Mathematically, transposition can be represented as

$$If\ a_k \Longleftarrow a_m\ then\ a_m \Longleftarrow a_k$$

Identity is represented as

$$a_p \Longleftarrow a_p$$

They represent the exact condition that defines bit- symmetry. This proves the theorem. □

3.3.2 Mathematical Definition

Let the function for bit replacement be $\mathfrak{R}_{\mathfrak{P}}$

$$\mathfrak{R}_{\mathfrak{P}} : B_0 \longrightarrow B_1$$

The definition of alternate addressing in terms of bit replacement function is as shown in (3.4)

$$\begin{aligned} If\ B_1 &= \mathfrak{R}_{\mathfrak{P}}(B_0) \\ then\ B_0 &= \mathfrak{R}_{\mathfrak{P}}(B_1) \end{aligned} \tag{3.4}$$

3.3.3 Time symmetry

The bit reordering choices are wide in number. In order to aid in the selection of the bit order, a corollary to the theorem is derived, which is helpful as a visual aid in the selection of ordering.

Corollary 1. *If in a stream loading into memory or being read out of memory, m^{th} input data appears at n^{th} output time slot, then to satisfy alternate addressing, n^{th} input data should go to*

the m^{th} output time slot. In other words, for AA, the input data positions are exchanged in the output stream.

Proof. In a stream, the sequence is numbered naturally. If the number of address bits required to represent the sequence is n, then the initial sequence can be written by using its bit-level representation as

$$B_0 = \{a_{n-1}, a_{n-2}, \ldots, a_k, \ldots, a_2, a_1, a_0\} \tag{3.5}$$

The output sequence is a permutation of (3.5) and can be written as

$$B_1 = \{a_r, a_s, \ldots, a_m, \ldots, a_{n-1}\}$$

where the subscripts represent arbitrary numbers less than n. Let $A_p \in B_0$ be an arbitrary data position and $A_q \in B_1$ be its replacing position. Then A_q is found from A_p by using the mapping $\mathfrak{R}_{\mathfrak{B}}$

$$\mathfrak{R}_{\mathfrak{B}}(A_p) = A_q \tag{3.6}$$

To find the replacement position of $A_q \in B_0$, we need to apply the mapping $\mathfrak{R}_{\mathfrak{B}}$ on it.

By applying definition of alternate addressing as in (3.4) on (3.6), we have

$$\mathfrak{R}_{\mathfrak{B}}(A_q) = A_p$$

Thus, A_q is replaced by A_p. Since the set produced by B_0 represents a naturally ordered sequence, A_p and A_q are both representations of time slots in both input and output data streams. Hence data on both the time slots stand exchanged, which completes the proof. □

3.3.4 Concurrent I/O

Roles of memory in FFT algorithms are of two types:

- **Independent**: Memory is used independently only as input or only as output. In output mode, the order of the data coming into the memory is defined by the FFT algorithm and referred to as algorithmic order. The data is read out in natural order. The vacated output locations are refilled with the **output** stream. For input mode, the process is similar.

- **Switched / Concurrent I/O**: Memory functions once in computation mode and then switches to input/output (I/O) mode after the end of computations, as shown in Fig. 3.4 [1]. When output data are being read, the vacated locations are refilled with **input** data, unlike in the other case.

In independent memories, the previously derived theorems apply directly. But for the second

case, an auxiliary condition is required to be fulfilled. The condition for CF in concurrent I/O is stated in Theorem 3.

Theorem 3. ***The last stage output order*** B_1 ***must be bit-symmetric to the first stage input order*** A_1**.** *Reordering at both input and output may be necessary to fulfill this condition.*

Proof. Let the input order written to memory be represented by the natural order bit-sequence

$$A = \{a_{n-1}, a_{n-2}, \ldots, a_0\}$$

The first stage reading order for processing by the first stage of FFT can be represented as A_1, which is a permutation of A.

$$A_1 = \{a_q, a_r, \ldots, a_z\}$$

The mapping $\mathfrak{R}_{\mathfrak{BJ}}$ defines the relation between A and A_1.

$$A_1 = \mathfrak{R}_{\mathfrak{BJ}}(A)$$

During intermediate computations, A_1 functions both as read and write orders. After all the data have been processed, the final stage output is in the sequence B_0. But it must be reordered to sequence B_1 before storage in memory so that CF is achieved. The problem now is to find B_1. A mapping $\mathfrak{R}_{\mathfrak{BO}}$ defines the order B_1. Hence,

$$B_1 = \mathfrak{R}_{\mathfrak{BO}}(B_0)$$

In concurrent I/O, when output is being read out in natural order B, the vacated locations are filled with **input** data A. After this stage, data are read out to feed the first stage in order A_1. In effect, write-in order B_1 is replaced by read-out order A_1 as in (3.7).

$$\mathfrak{R}_{\mathfrak{BOJ}}(B_1) = A_1 \tag{3.7}$$

The condition for alternate addressing imposed on the memory now requires that the read-out order A_1 be bit symmetric to write-in order B_1. Conversely, B_1 should be bit symmetric to A_1, which is the required proof. □

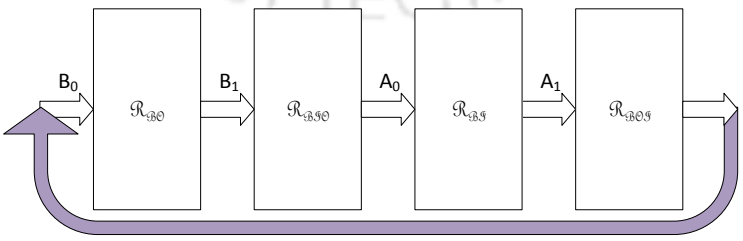


Figure 3.7: Reordering requirements in Concurrent I/O

Corollary 2 (Sufficient condition for CF in concurrent I/O)**.** *For concurrent I/O, if the output order* B_1 *is* ***same*** *as the first stage input order* A_1*, then CF can be achieved.*

Proof. The proof for this simply follows from Theorem 3. The theorem states that B_1 should be bit-symmetric to A_1. This is easily satisfied when

$$B_1 = A_1$$

i.e., the output order is the same as the first stage input order. □

Corollary 3. *Bit and Digit Reversed addressing techniques can be treated as special cases of AA because their output orders are bit-symmetric to input order.*

3.3.5 Design of 32-point AA technique for FFT

AA can be applied at several places in an FFT hardware to obtain CF. Some examples are given in this section to illustrate its use.

3.3.5.1 A 32-point mixed radix FFT: CFMR [1]

A 32-point FFT using radix-4 consists of two radix-4 and one radix-2 stages, making it a mixed radix processor. The original output sequence is shown in Table 3.2. To verify whether the condition for AA is met, corollary 1 is applied to it. Data number 16 is at time slot 1; hence according to corollary 1, data number 1 should come at time slot 16. It is observed here that one actually appears at time slot eight, thus violating the condition for AA. The input sequence has to be modified by bit reordering in such a way that both Theorem 2 is satisfied and the physical hardware is realizable. The reordering is shown in row 3 of Table 3.3. Since this is an output section, the order of Butterfly PE (BF) processing should remain unchanged as far as possible to minimize storage requirements. Within a butterfly, the outputs are allowed to be rearranged as this requires little hardware. The bits which control the BF processing order are b_1, b_4, b_3. But only less than 50% bits can be fixed to allow flexibility for reordering. Since there are 5 bits here, only 2 bits can be fixed. So b_4 and b_3 in row 3 are kept the same as in the original input order of row 2. The remaining bits in row 3, b_0, b_1, and b_2 are put in such locations as to satisfy bit-symmetry with row 1, which is the write-in order. The sequence is expanded in Table 3.2 and seen to satisfy corollary 1 also.

Table 3.2: Original and reordered output sequences for CFMR FFT [1]

	Time slot	0	1	2	3	4	5	6	7	8	...
Original	Data no.	0	16	4	20	8	24	12	28	1	...
Reordered	Data no.	0	8	16	24	4	12	20	28	1	...

Table 3.3: Re-ordering the output order of CFMR to be bit-symmetric

Order	Bit-number				
Natural read-out	b_4	b_3	b_2	b_1	b_0
Original write-in	b_0	b_2	b_1	b_4	b_3
Reordered write-in	b_1	b_0	b_2	b_4	b_3

3.3.5.2 Development of Mixed Radix-16 FFT of 512-points [2]

3.3.5.2.1 Input: A 512-point radix 16 FFT has two radix-16 stages and one radix-2 stage. Hence it is a mixed radix FFT. The input equation for the FFT is

$$n = 32n_1 + 2n_2 + n_3 \tag{3.8}$$

The addressing order used in [2] for first stage input, A_1, is shown in Table 3.4.

Table 3.4: Input Order Original: Unsuitable for CF

	Input number of radix-16 BF				
	I	II	III	...	XVI
BF_0	0	32	64	...	480
BF_1	1	33	65	...	481
BF_2	2	34	66	...	482
...	...	...	...	...	...
BF_{16}	16	48	80	...	496
...	...	...	...	...	...

By applying corollary 1, it is observed that this order does not meet the AA condition and hence is unsuitable for CF with two memories. The bit reordering for AA can be derived as shown in Table 3.6. Referring to Table 3.6, row 1 is the write-in order to the memory, while row 2 is the order in which data would be processed by the first stage of BF. In row 3, the read-out order of the memory is to be derived from row 2 such that it is suitable for CF. The aim is to change row 2 to row 3 such that the new order satisfies the AA condition with row 1 through little hardware modification. As in the previous example, the bits to be fixed are

Table 3.5: Input order reordered using AA for CF

Input number of radix-16 BF				
I	II	III	...	XVI
0	32	64	...	480
16	48	80	...	496
1	33	65	...	481
17	49	81	...	497
...	...	...	...	...
...	...	...	...	...

Table 3.6: Re-ordering the input of 512-point mixed radix FFT to be bit-symmetric

Order	Bit-number								
Natural write-in	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Original read-out	b_3	b_2	b_1	b_0	b_7	b_6	b_5	b_4	b_8
Reordered read-out	b_3	b_2	b_1	b_0	b_4	b_8	b_7	b_6	b_5

derived from a physical understanding of the problem. The order of inputs to a particular BF is not commutative and hence cannot be modified. Here, lower order bits, $b_3b_2b_1b_0$, form the 16 inputs to BF. Their positions in row 3 would be the same as in row 2. The other bits of row 3 are reshuffled to meet bit-symmetry with row 1. This would modify the processing order of BFs, and it has to be taken care of by modifying the later stages of the FFT algorithm slightly.

3.3.5.2.2 Output: The output order is as seen in Table 4.21. In two memory architectures, for seamless switching between input and output mode with CF, corollary 2 states the output order must be same as the input order in Table 3.5. In order to derive this bit rearrangement, it is important to note that this is the output section, and the butterfly processing order should not be modified much. Since only less than 50% bits can be fixed, we fix the bits $b_8b_7b_6b_5$ of row 3 to be the same as in row 2 as seen in Table 3.8. The other bits are placed to satisfy bit-symmetry with row 1. Since b_4 also controls the processing order and changes its position, the output processing order is somewhat modified and will need reordering buffers, as explained in the next section.

Table 3.7: Original output order for mixed radix-16 FFT of 512-points [2]

	O_0	O_1	O_2	O_3	O_4	O_{5-11}	O_{12}	O_{13}	O_{14}	O_{15}
BF_0	0	256	16	272	32	...	96	352	112	368
BF_1	128	384	144	400	160	...	224	480	240	496
BF_2	1	257	17	273	33	...	97	353	113	369
BF_3	129	385	145	401	161	...	225	481	241	497
BF_4	2	258	18	274	34	...	98	354	114	370
BF_5	130	386	146	402	162	...	226	482	242	498
BF_{6-31}	...	...	...	...	...	...	...	...	...	...

Table 3.8: Re-ordering the output to be bit-symmetric

Order	**Bit-number**								
Natural read-out	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
Original write-in	b_0	b_4	b_3	b_2	b_1	b_8	b_7	b_6	b_5
Reordered write-in	b_3	b_2	b_1	b_0	b_4	b_8	b_7	b_6	b_5

3.3.6 Reordering Hardware for AA

The reordering hardware for different architectures follows a general structure. It consists of a set of buffers and multiplexers (muxes) placed between a stage's output and the memory in which this output is to be stored. This is shown as a block in Fig. 3.9 and detailed in Fig. 3.8 for a particular case of 512 point radix-16 FFT architecture. Two buffers, which are called here as a buffer set, are required to store two consecutive cycles of output from PE, amongst which the reordering is to be done. A second buffer set is required to store the incoming data while reordering takes place in the first buffer set. The role of the two buffer sets is interchanged after data is read from one set. This requires the use of the buffer selection multiplexers, as shown in Fig. 3.8. The hardware implementation of the AA technique is compared with the other technique, Circular Shift Addressing (CSA) in section 3.7.

3.4 Circular-Shift Algorithm for Continuous Flow Addressing

In its generalized form, CF addressing has to ensure that a continuous input stream is stored in memory and read out as a continuous output stream with the order of the data in the two streams being different. The input data is placed on locations vacated by the output stream.

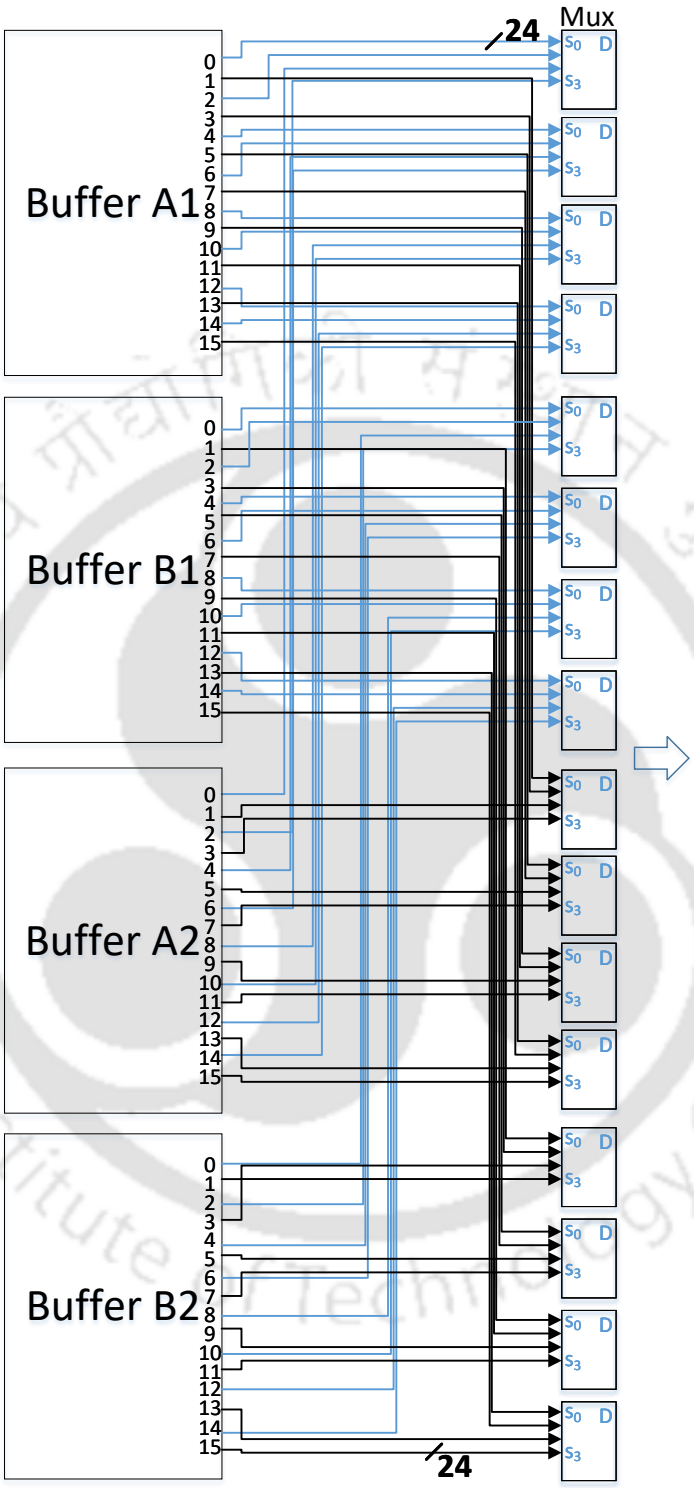


Figure 3.8: Buffer insertion for Reordering between stages for output stage of radix-16 FFT of 512 points

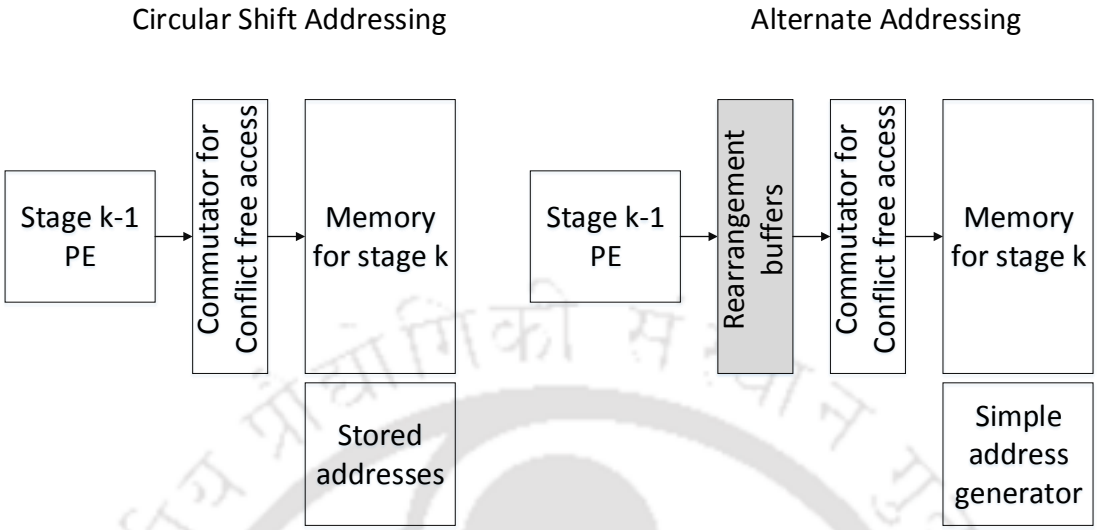


Figure 3.9: CSA contrasted with AA showing disturbance in the flow of data in latter case

Hence, the address of input data gets changed with every block of new data. This effectively means that in the next time block, if the output stream is to effectively maintain its predefined order, the address for the output data must change. A sufficient condition for this solution to exist is embodied in Theorem 4, and the technique is referred to as CSA. From Group theory, AA can be shown to be a special case of CSA. In CSA, the shift occurs in longer groups, while in AA, the group size of two is maximum. In comparison with the reordering scheme, the circular shift scheme does not disturb the flow of data. Both the diagrams are contrasted in Fig. 3.9.

Mathematically,

Let A be an input stream that is stored in memory

$$A = \{a_0, a_1, ..., a_{n-1}\},\ where\ A \subset N$$

where N is the set of natural numbers. Let B_0 be the output stream generated by applying an address set AD_0

$$B_0 = \{b_0, b_1, ..., b_{n-1}\},$$

$$Here\ B_0 \subseteq A$$

Then,

$$AD_0 : A \rightarrow B_0, \; such\, that\, a_i \Leftarrow b_i, \forall a_i \in A_0 and\, b_i \in B_0$$

Lemma 5. *The Lemma states that a_i can always be replaced by b_i in each iteration, given proper mapping or addresses. Mathematically,*

$$AD_{k+1} : B_k \rightarrow B_{k+1}, \; such\, that\, a_i \Leftarrow b_i, \forall a_i \in B_k, b_i \in B_{k+1}$$

where k is the iteration number.

Proof. In order to prove the Lemma, it is required to examine the existence of such addresses AD_k. In the zeroth iteration, $k = 0$, it is obvious that a_0 would be replaced by b_0, a_1 by b_1 etc. So, the new memory map would still be $\{a_0, a_1, \ldots\}$ albeit in different locations. If the locations are known (they actually are known, since locations of $a's$ are known in the first iteration), then $\{a_0, a_1, \ldots\}$ can be read out again by asserting the required addresses. These $\{a_0, a_1, \ldots\}$ would again be replaced by $\{b_0, b_1, \ldots\}$ coming in serial order. Hence, it is ensured that for every $\{a_0, a_1, \ldots, a_{n-1}\}$, and $\{b_0, b_1, \ldots, b_{n-1}\}$, a_i's are always replaced by b_i's, given proper addresses. □

Lemma 6. *The replacement function AD_0 can be represented as a circular series. This property of the replacements leads to periodicity.*

Proof. Here B_0 is formed by permuting the elements of A. B_0 belongs to the permutation group of A.

$$AD_0 : A \rightarrow B_0, \; such\, that\, f(a_i) = a_j, \forall a_i \in A, a_j \in B \tag{3.9}$$

Input order	a_0	a_1	a_2	a_3	a_4	a_5	...
Output order	a_k	a_l	a_m	a_n	a_o	a_p	...

By using the concept of permutation groups, AD_0 may be treated as formed by a number of cyclic permutations.

$$AD_0 = (a_0, a_k, \ldots, a_2, a_m)(a_1, a_l, \ldots, a_5, a_p)\, etc. \tag{3.10}$$

The parentheses (..) represent cyclic notation of permutation groups from group theory. The symmetric group AD_0 is formed by shifting the members in a cyclic group. The shifting operation on the group results in a similar arrangement after a finite number of operations. Also, we are interested in rearrangement within a cyclic set. The shifting operation produces a periodicity which proves the Lemma.

□

Theorem 4. *If in a sequence stored in memory, each output element is replaced every time by a particular input element, it is possible to realize CF using an addressing scheme without reordering the sequence.*

This condition is fulfilled almost in all cases of practical interest, where the orders of input and output stream are predefined. In this case, by Lemma 5, replacements will be circular in

nature. Hence by Lemma 6, periodicity is established. Thus an addressing scheme is guaranteed to exist and can be derived with the help of the same Lemma.

Proof. This is true for iteration, $k = 0$ by definition of AD_0. For the next iteration to move from state B_0 to state B_1, a different mapping is required, which can be called as

$$AD_1 : B_0 \rightarrow B_1, such\, that\, a_i \Leftarrow b_i \tag{3.11}$$

Here $B_1 \subseteq B_0$. This process would continue indefinitely. Now, if we consider AD_1, AD_2, AD_3, etc. as the successive mappings, the problem boils down to find the mapping AG which satisfies:

$$AG : AD_i \rightarrow AD_{i+1}\ for\ all\ values\ of\ i \tag{3.12}$$

From axioms of Group Theory, the existence of such a mapping is mathematically guaranteed since they are bijective mappings within a group. □

To illustrate, the output stream entering the output memory is taken from Table 4.21 and shown in row 2 of Table 3.9. Now data is read out in natural order as $\{0, 1, 2, 3, \ldots\}$. Then, by applying Theorem 4, the read-out data is filled by serially incoming data; thus Table 3.9 is obtained.

Table 3.9: Replacement of data locations in output memory by serially incoming data locations

Data location	0	1	2	3	...
Replacement	0	256	16	272	...

Lemma 7. *If the input and output series follow some regularity, it is possible to derive a formula for such replacements.*

A replacement formula gives the data location in the next iteration. It is not absolutely necessary, and computer 'find' routines may be used to derive the replacements and generate the replacement tables. The existence of the formula depends on the regularity of the series and hence cannot be guaranteed in general. Fortunately, in FFTs, the butterfly gives outputs based on some formulae. Hence, regularity can definitely be expected. The problem of finding out the formula is dependent on the case at hand. When natural numbers form one of the series, it is an easier problem to deal with, and the formula which is used to generate the algorithmic series can be used as a starting point. An example is taken up later in section 3.9.1 to illustrate the procedure.

Lemma 8. *Even if shifts are applied to the incoming data, if the same shifts are applied to each data every time, it is equivalent to a new series replacing the existing series. Hence, all theorems proposed above are applicable. Only the replacement formula would need changes to account for the shifts.*

Proof. Let the shifts be embodied in a sequence Sh

$$Sh = \{s_0, s_1, ...s_n\}$$

Then the mapping f can be modified

$$f' : B_{s0} \rightarrow B_{s1},\ such\ that\ b'_i = a'_i$$

Here,

$$B = \{b_0, b_1, \ldots, b_n\}$$

is a sequence containing only integer values from 0 to n. The various suffixes represent different permutations of the sequence. Now the following additions are defined:

$$B_{s0} = B_0 + Sh \bmod n$$
$$B_{s1} = B_1 + Sh \bmod n$$

Then the various $B's$ are closed under modulo addition. Hence B_s also forms a permutation group. Therefore, the series of the mapping f'_k applied repeatedly on B_s forms a permutation group. Hence all lemmas stated in the context would still apply. But the values b'_i would change according to the new sequence. □

3.5 Algorithm of CSA

A multi-bank data memory is considered. In CSA, the address generator is the core design element. Unlike in AA, where it is a circuit, it is made up of LUT called the address memory and offset correction circuit. The address memory stores a table of circular series which is derived as follows:

(i) If in a bank of data memory, data index A is always replaced by B, B by C, C by D, etc. then by Theorem 4 at some point, let us say D, it will be seen that D is replaced by A. Thus A, B, C, and D form a replacement group. Find and replace computer routines may also be used to locate the replacements. The find routines look for the outgoing data and replace it with the incoming data. Equivalently, the groups can be found out using the replacement formula for finding replacement addresses. This technique is illustrated by the use of an example in Appendix A.

(ii) For each memory of a bank, there will be many such groups containing a varying number of elements. All these groups can be concatenated together in a column of a table, where the various columns would represent the various bank in the memories. This table is called a replacement table.

(iii) The values in the replacement table are replaced with positions in bank memory to form a circular series table. This circular series table is stored in the address memory LUT. The address memory can be divided into r address generation units (AGU) consisting of a LUT with size as in (3.13). Each column of the circular series table is stored in one such memory. One AGU addresses one memory of the bank.

$$LUT\ size = n \times b \tag{3.13}$$

where n = number of butterflies in the stage, and $b = log_2 n$.

(iv) **Decoder Order**: A new table called Decoder Order is formed from Circular Series. The columns of this table are found by looking for the location from the Circular Series to generate the proper data sequence. This process is detailed in section 3.6.

(v) **Special Decoder**: The addresses from the AGUs would be read out as per the corresponding column of the Decoding Order table. Generally, data from memory cells are read out in order of the received address because a standard memory decoder activates the memory rows in the same order as the input address. In the case of this AGU, the decoder has to be specially designed so that the input natural order address actually activates rows in the decoding order for the particular column. This can be easily done in HDL by replacing the decoder outputs for each value of the decoder counter with the decoder order value.

(vi) **Offset subtraction**: The addresses are used after subtracting an offset value. The offset value to be subtracted is selected based on Algorithm 1. This process takes care of the circular shifting of the address values in the circular series table.

(vii) **Circular Shift**: A circular shift table is shown in Fig. 3.10. The differently grouped cells in a column correspond to circular buffers. After each iteration of the processor, the columns are shifted up by one row. This shifting does not proceed linearly; instead, the shifting takes place circularly, the boundaries being the group boundaries. This shifting is initiated after every iteration of the processor, where iteration means the complete processing of one block of N-point data. Thus, the change in address positions after each iteration is automatically taken care of by circularly rotating the locations as stored in address memory. The offset values take care of the rotation. After each block of data is processed, the offset values are decremented by one as in Algorithm 1 to affect the rotation.

Algorithm 1: Algorithm for offset calculation and address update

---

Let the number of series in a bank be K_b. Then there are k_b offset counters. Let B_{kb} be the points at which the series ends.

Active k:

$A = Addr_{seqgen}$;

$O_b = O_{kb}$, where

$k_s = 0, if\ 0 < A \leq B_{0b}$

$k_s = 1, if\ B_{0b} < A \leq B_{1b}$

...

$k_s = K_b, if\ B_{K_b} < A \leq 31$

Offset update:

for all $k = 0 \ldots K_b - 1$

(i) L_b= Length of k_b

(ii) Initialization: Offset value $O_{kb} = L_b$
After each iteration of the processor

(iii) $O_{kb} = O_{kb} - 1$
If $O_{kb} == 0$
then go to step 1
end

Final Offset value: $O_b = L_b - O_{kb}$

$Addr_b = A - O_b$;

3.6 Architecture of Circular-Shift technique for output addressing of 512-points mixed radix 16-16-2 FFT

In order to demonstrate the effectiveness of the CSA, a 512-point radix 16 FFT processor is designed with the algorithm taken from Huang [2], and the addressing logic is modified in accordance with this addressing algorithm. The focus here is on output addressing, assuming there is a dedicated output memory. As soon as the output memory is filled up, the output data needs to be sent out in the proper order of 0, 1, 2, ..., 15. This creates space for new data whose indices are as per the rows of Table 4.21. To ensure conflict-free addressing, we introduce an alternate progressive circular right shift to the input vector as shown in Table of Fig. 3.10.

From Fig. 3.10, replacement series are formed for each column. Replacement for column 1 is shown in Table 3.14. Since the number of address bits is 5, the size of the required address storage memory would be small. Hence register-based address memories would be used. This would allow the use of the special type of memory which could avoid the need for a replacement circuit. In Fig. 3.10, Table 1 may be formed by the use of a replacement formula or by the use of a computer search and replace function. It is to be noted that column 1 is replaced by data from column 1 itself in the replacement process. A chain of replacements is now made, resulting in several series (row-wise) for bank 1. Shades are used to identify the different series.

When the above data locations are replaced by their positions in column 1, a circular series table is obtained, as seen in Fig. 3.10.

In the example at hand, for the output to be in natural order, 16 columns of memory would produce 16 outputs which are then progressively left-shifted. The column 1 data sequence to be generated is shown in Fig. 3.10, and it is a monotonically increasing sequence. In order to generate this data sequence, addresses have to be applied to the memory for column 1. These addresses are read from column 1 of the Circular series table, which is in the form of Address LUT and shown in row 2. The serial order for reading these addresses is called Decoder Order and is shown in row 1. The Decoder Order for other columns is found similarly.

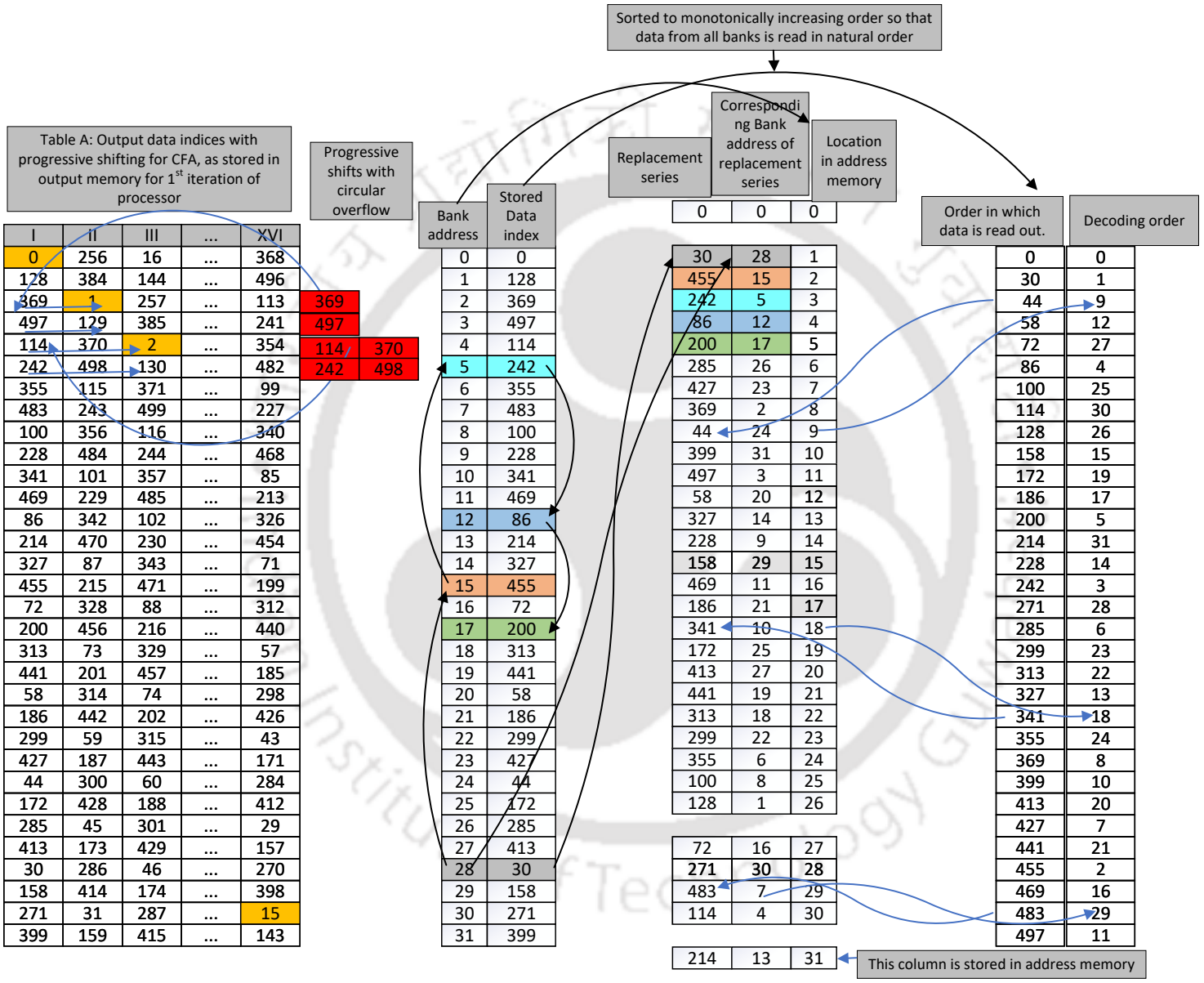


Figure 3.10: Tables for bank 1 showing CSA related data

Table 3.10: Comparison of results of the proposed designs with state-of-the-art

Parameter	Cho [12]	Wang [31]	Huang [2]	Proposed (AA)	Proposed (CSA)
Architecture	8 way MDF	MDF	Memory	Memory	Memory
Radix	2^5	$2^4 - 2^2 - 2^3$	16	16	16
Throughput (GSps)	2.6	1.76	2.4	4.8	4.8
Technology node	90	130	90	180	180
Clock	350	220	324	300	300
Area at 180 nm including an additional memory for CF	4.9	3.15	3.54	4.3	4.1
Word length	14	14	12	12	12
Area efficiency (GSps/mm^2)	0.53	0.55	0.68	1.12	1.17

3.7 Synthesis Result and Comparison

Simulation of both CSA and AA-based address generators is carried out by creating the Verilog model of the architecture for a 512-point radix 16 FFT. The simulation is carried out in Synopsys VCS. After the initial design is done, the Verilog code is converted to netlist using Synopsys DC. The area is found from the DC synthesis results. For comparison with other designs, area efficiency is chosen as a metric. The comparison of various designs is shown in Table 3.10. For a fair comparison, other designs are extrapolated to 180 nm, which is our target technology. The formula used for extrapolation is (3.14).

$$Normalized\,area = \frac{Nand\,Gate\,Area_{tgt}}{Nand\,Gate\,Area_{src}} \times Area_{src} \tag{3.14}$$

where tgt refers to target technology and src refers to source technology. An additional area of $0.8mm^2$ is added for dual port memory of 512 points, which is required because all processors

in the table claim CF capability, but this additional area is not included in the final area considerations. For Wang [31], it is further assumed that the control circuit area for the second memory has been included in the FFT core area.

AA technique is used when changing the order of outputs is acceptable and when wordlength is not very high. The circular shift technique is to be preferably used if reordering is not allowed, as in the intermediate stages of FFT algorithms. It gives a lower hardware count when wordlength is high. This technique does not require any computed data storage and switching, reducing the latency. Also, it does not disturb the data flow and may be used in legacy systems to convert them into continuous flow by just changing the address logic. In the case of those FFT stages, where reordering is not allowed or in legacy designs, the proposed technique of CSA is the only known way to achieve CF.

3.8 Conclusion

The conditions for continuous flow have been derived for FFT processors with the number of points as the power of 2. These conditions help to use mixed radix algorithms in continuous flow FFTs with better understanding. Earlier work had been for single cases and is not extended to cover all cases. Two methods are outlined for this; the first one, called Alternate Addressing which changes the order of outputs; the other technique of Circular-Shift Addressing is more general in nature and can be used even where changing the output order is difficult or not at all allowed. Converting legacy designs with proven control flow to CF would be an ideal use of the Circular-Shift Addressing method. Both have their preferred areas of application, and one may lead to simpler hardware than the other, depending on the architecture of FFT. The theoretical framework provided in this work is expected to help FFT designers with a large number of choices in designing FFT with CF. Also, the circular shift technique discussed here is a new entrant to the field of CF addressing and merits further research.

3.9 Appendix

3.9.1 Derivation of replacement formula

The output data generated by the FFT processor of Huang [2] is governed by the output equation (3.15)

$$Output\ order = 256k_3 + 16k_2 + k_1 \tag{3.15}$$

where k_3 changes first from 0 to 1; followed by k_2 from 0 to 15; and concluded by change in k_1 from 0 to 15. The replacement formula is derived using this information as in (3.16)
Replacing data location,

$$m = 256\ (n \bmod 2) + 16\,(\lfloor n \rfloor \bmod 16) + \lfloor \frac{n}{32} \rfloor \tag{3.16}$$

The detailed explanation is as follows:

- n mod 2: k_3 changes first, hence k_3 is odd or even alternately.
- ($\lfloor n \rfloor$ mod 16): k_2 changes for next 32 elements, being same for each adjacent odd or even element, taken care of by floor function.
- $\lfloor \frac{n}{32} \rfloor$: k_1 changes after every 32 elements.

This formula (3.16) gives the replacement values for each of the natural order outputs. The replacement series of output can now be easily derived, as shown in Table 3.11. But the progressive shifts have not yet been taken into account. These shifts modify the actual natural number into a nearby number which, when fed as input to the formula, gives the actual replacement. The shifts are decided by (3.17).

$$S = \lfloor \frac{n}{16} \rfloor + \lfloor \frac{n+16}{32} \rfloor \tag{3.17}$$

Thus, S will remain constant for every 16 elements. In every block of 16 elements, 16 is subtracted from the last S shifts to obtain the final shifts. Hence, in every block of 16 values, the last shift values are negative. The shifts are shown in Table 3.12.

Table 3.11: Replacement formula based location of replacement addresses for output of 512-point FFT

Data into vacated memory	0	1	2	3	...	511
Data moving out as per (3.2)	0	256	16	272	...	511

Table 3.12: Shifts for adding to natural numbers row-wise to obtain shifted input for replacement formula for output of 512-point FFT

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	2	2	2	2	2	2	2	2	2	-14	-14
3	3	3	3	3	3	3	3	3	3	3	3	3	-13	-13	-13
5	5	5	5	5	5	5	5	5	5	5	-11	-11	-11	-11	-11
6	6	6	6	6	6	6	6	6	6	-10	-10	-10	-10	-10	-10
8	8	8	8	8	8	8	8	-8	-8	-8	-8	-8	-8	-8	-8
9	9	9	9	9	9	9	-7	-7	-7	-7	-7	-7	-7	-7	-7
11	11	11	11	11	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5

Table 3.13: Replacement table for bank 1 (read row-wise)

0	30	455	242	86	200	285	427	369	44	399
497	58	327	228	158	469	186	341	172	413	441
313	299	355	100	128	72	271	483	114	214	

Table 3.14: Replacements taking shifts into account for column 1 of Table in Fig. 3.10

Column 1	0	128	369	...	399
Replacement	0	100	427	...	44

4

Continuous Streaming 4.8 Gsps Radix-16 512-Point FFT Processor for OFDM

Contents

4.1 Introduction

The ever increasing demand for better applications of Digital Signal Processing (DSP) in areas of communication, medical image processing, and emerging areas of computer vision have led to the increasing use of high throughput Fast Fourier Transform (FFT) IP-cores. For Wireless Local Area Networks (WLAN) communication, the need for higher data rates has led to the evolution of newer standards using the 60 GHz band. The IEEE 802.11ad standard released in 2013 uses a speed of 7 Gbps; while the recently evolved IEEE 802.11ay, an extended version of 802.11ad, proposes speeds higher than 20 Gbps. One of the suggested modulation techniques is Orthogonal Frequency Division Multiplexing (OFDM) transmission with 512 subcarriers. FFT is a computationally intensive operation in OFDM. In order to meet the throughput requirement of the standard, it is necessary to develop a high throughput and continuous flow (CF) architecture for FFT. The FFTs for real-time applications such as communication must handle real-time flow or CF, while for offline processing, the CF requirement may not be there. Several designs are available in the literature for general-purpose FFT with throughputs up to 2.4 Gsps (Giga samples per second), which translates to 14.4 Gbps for 64-QAM in OFDM. However, none of them are specifically designed for CF. High throughput FFTs require higher radices which require a larger number of memory banks and hence, a more elaborate Conflict Free Access (CFA) scheme. To increase the throughput of a design, the number of processing elements (PE) and associated memory have to be increased, leading to greater difficulty in meeting CFA and CF at the same time.

An OFDM-specific FFT architecture possessing simple CFA and CF capabilities, along with normal order input-output (I/O), is proposed. This special purpose design is much more efficient than a general FFT for the 802.11ay WLAN transmitter. This FFT processor takes only 8-bit complex words as input for the 64-QAM modulation schemes as needed in 802.11ad / ay. The earlier designs could process higher wordlengths which is not required for an OFDM transmitter. A separate input memory is introduced along with a PE to double the throughput, thus keeping the number of memories to three. Since the algorithm is based on radix-16, the

number of stages is also three. Thus, each memory can serve a single stage, and their roles can be fixed as shown in the lower part of Fig. 4.1. Due to the fixed role, 8-bit memory is sufficient at the input. This is in contrast with the conventional "switched" memory configuration shown in the first part of Fig. 4.1, which is of 24-bit wordlength to store computational data also. Dedicated role memories are a unique feature of this design. The main contributions of this work are as follows:

- A FFT chip with a designed throughput of 28 Gbps is implemented and fabricated to meet the requirements of above 20 Gbps throughput for Wireless Gigahertz. This FFT chip is the fastest and the most efficient for Wireless LANs. The proposed architecture achieves double throughput with only 20% increase in overall area compared to the best existing 10 Gbps designs at similar clock speed. This achievement is due to the elimination of memory switching multiplexers, reduced complexity of PEs, and a lower complexity CFA scheme.

- Compared to existing switched memory systems, we have developed an efficient addressing scheme based on a novel "Replacement CF Addressing" technique. In section 4.8, we propose the principles of CF and then utilize them to derive our circuit architecture.

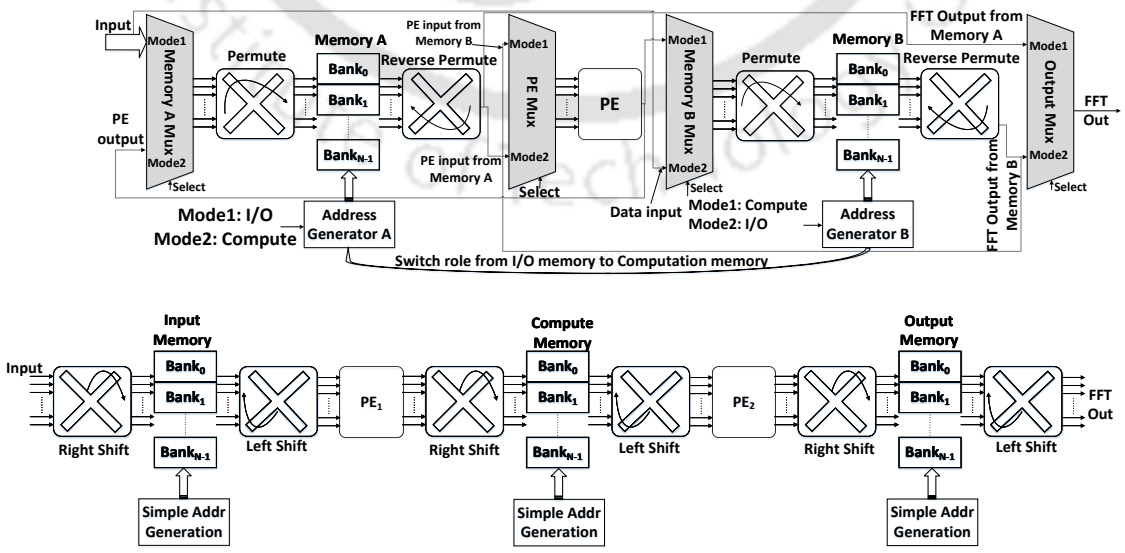


Figure 4.1: Data-flow diagram for Switched memory and Dedicated memory design

- In this design, a memory feeds only one or two stages compared to the earlier designs meant for feeding a large number of stages. Hence, a novel CFA scheme termed as "Progressive Shift" technique is proposed here, resulting in 30% simpler commutation logic than existing schemes.

4.2 FFT Algorithm

The first selection is the radix of the FFT algorithm. A very low radix increases the number of stages, while a large radix makes the realization of the PE more complicated. For a data rate of 4.8 Gsps, 512 samples are available in 512 / 4.8=106 ns. Assuming a clock frequency of about 300 MHz i.e. clock period of 3 ns,

$$Number\ of\ cycles\ required = \frac{106}{3} = 35 \tag{4.1}$$

If one PE is used to realize one Butterfly (BF), the radix for meeting this target can be derived using

$$BF\ Operations\ (BF_{op}) \leq \begin{array}{l} BF\ per\ stage \\ \times No.\ of\ stages \end{array}$$

$$BF_{op} \leq \frac{N}{R} log_R N$$

where N and R represent the number of FFT points and radix of FFT algorithm respectively.

Assuming that a single BF is processed per clock cycle, BF_{op} is the same as the number of clock cycles for complete processing. It is tabulated for different radices in Table 4.1. From the table, it is inferred that radix-32 FFT will meet the requirement. However, radix-32 is practically difficult to implement; one reason being a larger number of twiddle factor multipliers (32 for the first stage and 8 for the next stage). Hence radix-16 with 2 to 3 PEs is selected for our design.

The radix-16 algorithm used by [2] meets the criteria of being small enough to keep the PE simple yet large enough to reduce the number of stages to just three. It is hence used here. The

Table 4.1: Clock cycles for different radices for N=512

Radix	2	4	8	16	32
Clock cycles	2304	576	192	72	29

equations describing the input and output are reproduced here for reference, where n_1, n_2, n_3 are defined in (4.12), k_1, k_2, k_3 are output indices of stage 1, 2, 3 respectively and W stands for twiddle factor.

$$
\begin{aligned}
&X(k_1 + 16k_2 + 256k_3) = \\
&= \sum_{n_2'=0}^{31} \left(\underbrace{\sum_{n_1=0}^{15} x(32n_1 + n_2') W_{16}^{(n_1 k_1)}}_{\text{16-point DFT of stage 1}} \underbrace{W_{512}^{(n_2' k_1)}}_{\text{tf stage 1}} \right) \\
&\qquad \times W_{32}^{(n_2' k_2')} \\
&= \sum_{n_3=0}^{1} \left(\underbrace{\sum_{n_2=0}^{15} BO_{16}^{(2n_2+n_3)}(k_1) W_{16}^{(n_2 k_2)}}_{\text{16-point DFT of stage 2}} \underbrace{W_{32}^{(n_3 k_2)}}_{\text{tf stage 2}} \right) \\
&\qquad \times W_{2}^{(n_3 k_3)}
\end{aligned}
\tag{4.2}
$$

The architecture used by Huang is presented in Fig. 4.2. It takes about 64 cycles and uses two memory sets for continuous flow (not shown explicitly in the figure). To bring it down to 32 cycles, one more PE has to be inserted. There are two ways in which this can be done. The first method is to insert it in parallel with the existing one. But this approach has several shortcomings. The input memory bank has to be now divided into 32 blocks instead of 16 blocks to feed 32 inputs. The total area consumed will increase by 55% as found from memory compilers and shown in Table 4.2. The second stage processing will be performed jointly by these two units again. Hence, both the first and second stages are performed on same hardware. So the size of input memory will have to be sufficient to take care of *processing* word length. The twiddle factor multipliers of the first stage use 16 full complex multipliers. The number

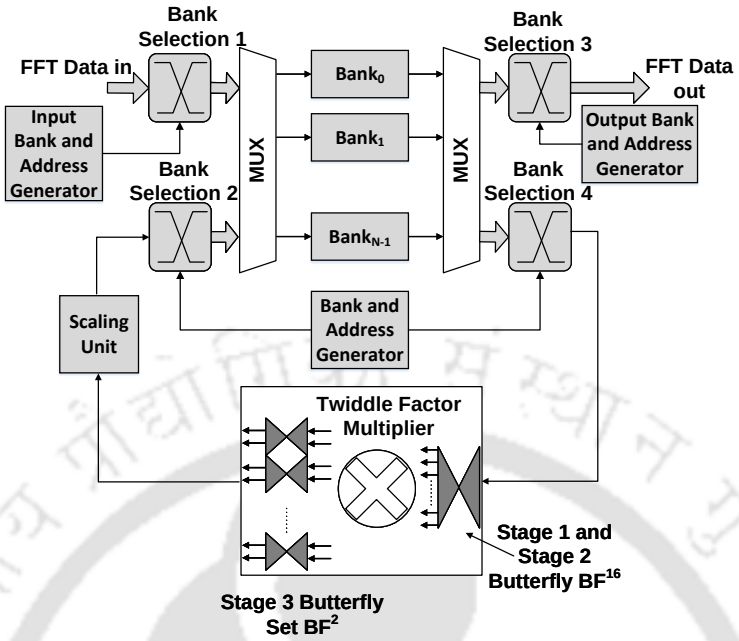


Figure 4.2: Architecture details of 512-point radix-16 FFT [2]

Table 4.2: Overall area increases when memory is halved (32 nm process)

Memory Cut	Area (μm^2)	No. of banks	Total area	Area increase (%)
32×24 bit	7361	16	117776	
16×24 bit	5724	32	183168	55

of multipliers will double to take care of data coming out of 32 outputs. Moreover, increased complexity in addressing, control and commutation will be required.

On the other hand, the approach adopted in this work is to use the second PE to process the *second stage*. Then several architectural benefits are derived as discussed below:

(i) One additional set of 16 bank memory is required to feed the second PE. This memory is of full size to accommodate the full wordlength of processing. The inclusion of this 3^{rd} memory separates the input, computation, and output blocks.

(ii) The input memory, its associated commutators, and the first stage PE can be downsized to just the necessary wordlength of the input data. In case of OFDM, the input wordlength is

much smaller than the processing wordlength for FFT. Hence, the area for input memory (8 bits) is almost equal to $1/3^{rd}$ of computation memory (24 bits).

(iii) Second stage multipliers are constant for each path and, hence can be realized in simple form using Canonic Signed Digit (CSD) multipliers.

(iv) Moreover, savings are generated because of the removal of switching logic which is used to convert an input memory to computation memory. Also, the independence of the stages allows use of a simpler low area CFA technique.

(v) The CF addressing here is now different from the first approach. This design uses a novel replacement based approach rather than the memory switching technique used in earlier designs.

4.3 Proposed FFT Architecture

The proposed design uses three memories, two radix-16 PEs, and eight radix-2 PEs along with commutators and address generation logic as shown in Fig. 4.3. The presented architecture is a memory based design. In order to justify the selection of memory based architecture over pipelined one, the area calculations are shown in Table 4.3. The value of the area for the various sizes is found using Synopsys Memory Compiler and Synopsys DC for 32 nm Synopsys generic library. From the area calculations shown in the Table 4.3, it is seen that the memory area required in memory-based design is 28% lower. Also, memory based designs provide a lower complexity in control when CF is implemented. The use of PEs in all stages of this design makes this design pipelined. Notwithstanding, we refer to it as memory based as the essential control and memory structure which is the one used in memory based FFT circuits.

4.3.1 Memory Architecture

An N-point FFT processor essentially consists of at least one memory with N storage locations. The memory is subdivided into smaller banks. For memory-based architecture, the memory is divided into r banks each of size $\frac{N}{r}$ as shown in Fig. 4.4. For the pipelined case,

Table 4.3: Comparison of memory area for a 512-point 8-datapath pipelined and a radix-16 memory based FFT processor (32 nm node)

Pipelined				Memory based		
Size	Area (μm^2)	Area × 8 (μm^2)	Type	Size	Area (μm^2)	$Area \times 16 (\mu m^2)$
32×24	7361	58888	SR[1]	SR 32×24	7361	117776
16×24	5724	45792	Reg[2]			
8×24	2960	23680	Reg			
4×24	1558	12464	Reg			
2×24	855	6840	Reg			
1×24	500	4000	Reg			
Total area		151664				117776
% Savings=$\frac{151664-117776}{117776}$= 28%						

the bank size is explained below. There are various options when choosing memory for any architecture. The important considerations are discussed below:

- **Memory-based design and pipelined design**:

  Pipelined designs have been quite popular [88] and even now [89]. They are thought to consume lesser memory and provide continuous flow. It is shown that for multiple data path cases, i.e., parallel processing and higher radices, it is not necessarily true. Memory-based designs of [2] are shown to consume a lesser area. The general structure of a memory-based design consists of at least one bank of memory along with necessary address generation logic, PE, switches called commutators between memory and PE, twiddle factor storage and a multiplication unit. Pipelined designs use the general structure of a PE in each stage, while memory-based designs are distinguished by a single PE for all stages. Other designs are a mix of the two extreme types. The usual criteria for the selection of one type over another is the number of PEs required to meet the throughput requirements. If a larger number of them are needed, it may be convenient to use the pipeline architecture. But, if a smaller number of PEs are sufficient to meet the throughput requirements, memory-based architecture would be a logical choice.

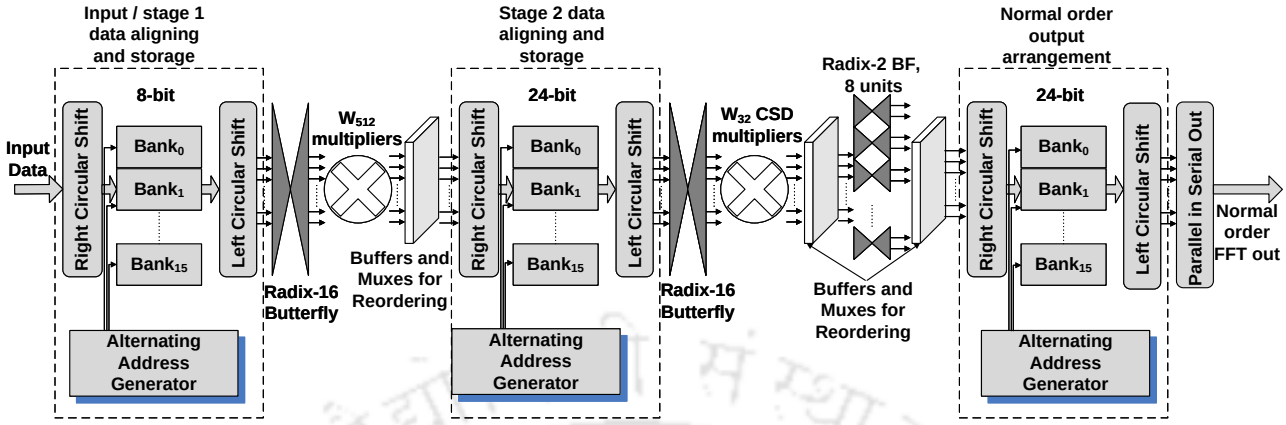


Figure 4.3: Detailed architecture showing the separated stages

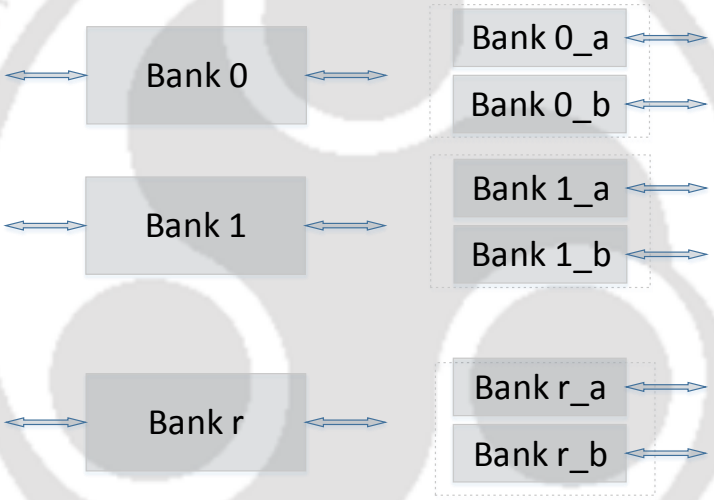


Figure 4.4: Dual port compared with single port memory bank

Both memory-based and pipelined architectures [89] use an equivalent number of storage units. But few papers have attempted to analyze the actual storage area. In pipelined architectures, the size of the memories (FIFOs) becomes progressively small from one stage to the next. It is to be noted that for SRAMs, one memory of size N is smaller than two memories of size $N/2$ [90]. Pipelined architectures divide the memory for an N point FFT as $N/2$ in the first stage, $N/4$ in the second stage, and so on. They require slightly lesser memory than N, but the savings in the number of locations is offset by the increase in the area because of the memory being built from smaller units. For the smaller size of

memories, SRAM memories are of larger size, and register memories have a lower area.

Table 4.4: Area comparison of 64-point pipelined and memory-based FFT processor

32nm Synopsys library		
Memory size	Area (μ^2)	Type (Selected on the basis of lower area)
32×24 bit	7361	SRAM
16×24 bit	5724	Register
8×24 bit	2960	Register
4×24 bit	1558	Register
2×24 bit	855	Register
1×8 bit	500	Register
Total area	18958	

Comparison with two memories of size 32×24 bit used in memory based designs

% increase in pipelined memory area $= \frac{18958-(7361\times2)}{(7361\times2)} = 29\%$

In Table 4.4, the area required for the memory of a 64-point pipelined processor is computed. We use the lower of the two areas of the Analog memory block and its register-based counterpart for the calculation of the total area. This is compared with the area for two memory banks of size 32×24 bits. It is seen from the last row of Table 4.4 that pipelined processors may need **30% more area** for memory. Hence, in pipelined designs area requirement is higher because of a larger number of PEs and the use of a mix of register memories and SRAMs. On the other hand, for memory-based designs, for increasing performance, radix of FFT needs to be increased. In order to control the increased complexity, split radix is used. For improving throughputs in pipelined designs, the number of radix is also increased, but the increased radix is usually split and implemented in multiple stages. The further performance increase is managed by parallelizing the datapath. But the most distinguishing feature is that while pipelined designs run at a data clock rate, the memory-based system can run at a clock rate different from the data rate. For smaller data rates, pipelined designs lead to istage of area since the use of a higher clock frequency could have saved area by reducing the number of PEs. Another advantage of memory-based designs is that output can be reordered without extra mem-

ory, while for pipelined designs, another set of memory is needed if the output is to be reordered.

- **SRAM or Register bank**: Many a time, designers are faced with the dilemma of choosing between Register memories and SRAM. It is generally thought that if the memory requirement is large, then SRAM is the better choice. We see, in the paragraphs to follow, that this is not always the case. It is true only for smaller radix FFT algorithms. For larger radices, it would depend on the number of points for which the FFT is computed. In the earlier days of FFT, SRAMs have been dominating the scene because for sizes above 16 locations; their area is smaller than that of their register memories. But in high throughput designs, the large memory has to be broken into much smaller banks. Register banks would then provide an area advantage. They have other benefits also, such as 5 to 6 times faster access times, ease in designing for any size, easily available as part of the standard cell library, and the flexibility to have any number of ports or even no ports. They have the disadvantage that larger memory would consume more area. Nevertheless, other aspects of physical design, such as DFT, which is easy to register banks, may further tilt the choice in its favor. SRAMs have a comparatively smaller area for large size. But they come in specific sizes and have to be purchased or designed separately. For custom design, different design teams are required. During the physical design phase, SRAMs require additional effort and cost for testing. Hence, the final choice would be influenced by all the above-mentioned factors. The primary differences between register and SRAM-based memories are shown in Table 4.5. Also, in Table 4.4, we see the size of register-based memory is smaller than memory macros after a particular size.

  Hence, it is imperative that all aspects should be studied carefully before selecting any particular type of memory.

- **Multiport Memory**: The memory may be a single port or dual port. Dual port memory is generally preferred because whenever memory is read, the data enters the PE. The outputs of the PE from the previous cycle will hence get lost in this cycle if they are

Table 4.5: Differences between register and SRAM-based memories

	Register Memory	Analog Memory
Speed	5 to 6 times faster	Slower access times
Size	Can be easily custom designed	Comes in specific sizes. Different design teams are required for custom design
Area	Usually larger for larger sizes	Larger area for smaller sizes and comparatively smaller area for large size
No. of ports	Any number of ports can be custom made	Usually single port memory is most widely available
Cost	Part of standard cell library	Has to be purchased or designed separately
Testing (DFT)	Easy	Requires additional effort
Others	Input or output ports may or may not use decoders	Both ports are decoded by default for dual port memory

not stored back in memory. Thus every cycle normally involves both read and write operations on the memory.

In the case where single port memory is used, the number of individual memories in the bank would need to be twice the radix, r. This will allow one set of memory to be read while the other is being used to write. The area of dual port memory is 30% less compared with two single port memory of half size. Single port memory may complicate matters related to address generation. Also, additional multiplexers and control structures would be needed to switch between the two sets of memories in the same memory bank, thus ultimately increasing the area. Further, the size of the individual memories may become so small as to compete with register-based memories. Register-based memories may be easily converted to dual ports while providing more flexibility. This suggests the possibility of trade-offs in the selection of memory structure among area, speed, and flexibility.

4.3.2 Dedicated Multi-bank Memory Architecture

Input to the 64 QAM OFDM FFT engine comes from the set 0, 1, 2, 3, ..., 7 and -1,-2, -3, ..., -7, which can be represented by only 4 bits. Considering that there are both real and

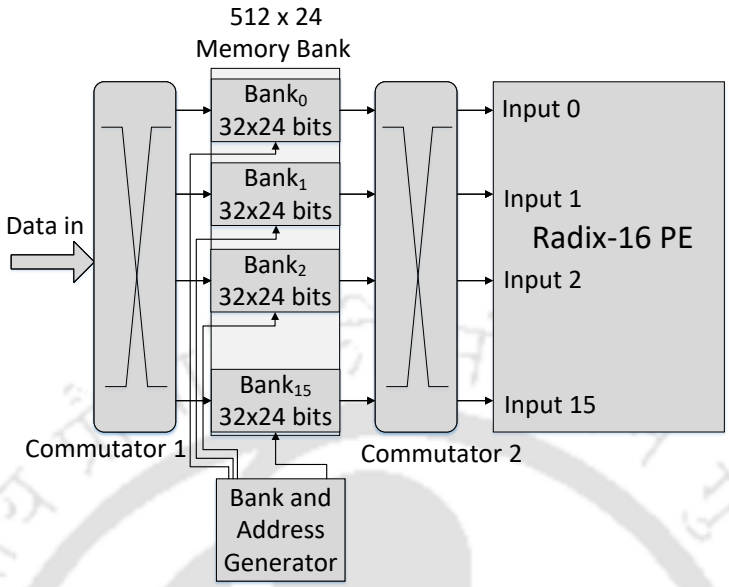


Figure 4.5: Memory bank in a memory based radix-16 FFT architecture

imaginary parts, the input memory needs 8-bit storage. In the three-memory architecture, the input memory deals with only input data; hence it is 8-bit wide. Intermediate memory contains 24-bit computation data, while the last stage memory also contains 24-bit computation data, which is finally sent out as output. Data would be read in and out from memory simultaneously, which necessitates a dual port memory.

For a high radix PE, also known as a butterfly, multiple inputs have to be provided at the same time. Since the order of inputs to the PE differs from the natural order in which data arrives, these inputs must be stored in a memory of N locations for an N point FFT, before the processing can begin. For a radix-r PE, r inputs are required which would come from r memories. Hence, all the memories are divided into r banks with $\frac{N}{r}$ locations each as shown in Fig. 4.5 for the input memory with $N = 512$ and $r = 16$.

4.3.3 Pre-Fixed Scaling

Block floating point scaling (BFP) is most widely used to scale down data a priori to prevent overflow from additions. BFP are large units, hence fixed scaling helps to reduce area consumption. However, fixed scaling might lead to loss of Signal to Noise Ratio (SNR), which

may necessitate longer wordlengths, thus offsetting the advantages. To overcome this problem, several simulations are carried out with typical OFDM inputs using the MATLAB fixed point model of the architecture. Necessary scaling factors are decided for all stages of the design. Since the fixed scaling units cost minimal, scaling could be done at all stages. In this way, the overall SNR is maintained. Fixed point simulations with 12-bit wordlength show that the achieved SNR is around 37 dB which is acceptable for our purpose.

4.3.4 Combinational Logic Blocks

There are five major combinational blocks. They are the two radix-16 PEs (PE^{16}), the radix-2 PE unit with eight PE^2, and the two multiplier blocks. They are described below:

4.3.4.1 PE^{16} for Stage 1

In [2], PE^{16} is designed using radix-4 PEs in a sequential architecture . However, the proposed implementation uses a fully pipelined and dedicated architecture instead. The advantages of the proposed architecture are:

(i) In stage-1, the initial 4 adders can be 8-bit ones instead of 24-bit. This reduces the complexity involved in addition.

(ii) Since the multiplication factor for a particular path is fixed, the radix-4 multipliers can be designed using CSD, thus saving area.

Pipelined registers are needed at several locations such as, after the first stage radix-4, after the twiddle factor multipliers, and finally at the output of PE^{16}. This structure helps to achieve a low critical path, and thereby a high throughput.

4.3.5 Multipliers for W_{16}, W_{512} and W_{32}

The architecture uses a fixed W_{16} multiplication at each output of the first stage of radix-4 butterflies of PE^{16}. Since constant multiplications are used, they can be replaced with single CSD multipliers. CSD uses a minimum number of additions [91] to replace a multiplication.

It is based on representing a binary number by its equivalent canonic representation. In this design, a fully dedicated architecture is used unlike in [2], where several CSD multipliers are required due to multiplexing.

The W_{512} complex multipliers after stage-1 do not have constant multiplying factors, hence they are designed as full multipliers. A complex multiplier is normally realized using 4 real multipliers. However, by using certain input transformations, it can be realized using three multipliers to save the area. The system uses 12-bit twiddle factors (TF) for both real and imaginary parts. Since all the 16 complex multipliers are addressed simultaneously, their TF can be stored at a single address. There are 32 such sets of multiplications. Hence, a single Look-up Table (LUT) based ROM with a word size of $24 \times 16 = 384$ and 32 locations is used as TF ROM. The synthesis tool simplifies the storage to use a minimum area. The 16 W_{32} multipliers of stage-2 are designed using CSD due to the constant nature of the multipliers. This saves more than 50% area compared to full-fledged complex multipliers. Additionally, it also saves the area of ROM which would otherwise be needed for storing the twiddle factors W_{32}.

4.4 Design Space Exploration

The design space consists of radix, number of memories, number of banks in memories, and CFA and CF techniques. Each one is detailed in the subsections below.

4.4.1 Radix Selection

For a small throughput, radix-2 based designs [88] are popular even today. In a simple radix-2 architecture with a single memory, the computation of N- point FFT needs $N/2 \times log_2 N$ clock cycles and $2N$ additional clock cycles for input and output. The processing is done in blocks and hence discontinuous. This can be called a block mode operation. Hence for block mode operation at a clock speed of 300 MHz, a 1024-point FFT attains a throughput of about 60 Msps. If higher throughputs are required, one option is to use higher radices. Radix-4 is an optimum choice for direct hardware implementation. Radices above that, such as 8 and 16, may warrant

undue complexity when implemented directly. The other option is pipelined architecture [92] in which latches are placed between the stages. The commutation complexity is high between the stages and increases with the radix. Nevertheless, pipelined designs have been popular from the early days of FFT design [88, 89]. Their popularity is due to the simultaneous processing of multiple radix-2 elements. With split radix [89], higher radices can be easily used without the complexity associated with their realization. This reduces the number of stages drastically, and hence split radix are popular in high-performance memory-based designs, making them quite competitive [2]. But for high radix PE, the memory is broken down into smaller units. The overall area occupied by the memory increases due to this cleavage [90]. When the size of the memory is small enough, even register-based memory becomes competitive. Register-based memories have other advantages also. Hence, in the realm of high throughput designs, new possibilities arise in the selection of memory which may even affect the choice of the entire architecture itself.

Any design starts with some **user specifications** to be met. In this case, the specification would be data rate, D; number of points, N; and output SNR, where the output signal is the HDL output, and noise is the difference between this signal and the output from floating point FFT of Matlab or a similar standard result.

The designer's specifications arise from process limitations, available libraries, and macros. The process limitations limit the maximum clock frequency, S, which the system can use. The task at hand is to find the maximum number of allowed butterfly operations, given S. This would then be used to select R, the radix of the FFT algorithm.

Since D is data rate, D data arrive in 1 sec.

Therefore, N data arrive in $T = N/D$ sec. To maintain continuous flow, the processor should finish the process in T sec. Assuming one butterfly operation per clock cycle of the system clock,

if B is the number of butterfly operations, then the time required for complete processing is

$$T_c = B \times S$$

Since $T_c \leq T$, the number of cycles, $B \leq \frac{NS}{P}$.

For an example, D=16 Msps, S=100 MHz, N=4096, for use in tracking in a GPS receiver.

Here,

$$B \leq 4096 \times 100/16 = 25600$$

Now, the required number of butterflies for a given radix, R is

$$B_R = (N/R) \log_R N$$

The least radix which meets $B_R < B$ criteria is selected for our implementation. From Table 4.6, even radix-2 meets our criteria.

Table 4.6: Required number of BF for different radices ($N = 4096$)

Radix	B_R for N=4096
2	24576
4	6144
8	2048
16	768

For the design in this work, the data rate of D=2.4 Gsps, for WPAN, target technology clock speed, S=300 MHz. This clock frequency represents the maximum delay in the critical path which passes through the multiplier and is expected to be less than 3 ns for SCL 180 nm library. The number of FFT points, N=512. Therefore

$$B \leq 512 \times 300/2400 = 64$$

Table 4.7: Required number of BF for different radices ($N = 512$)

Radix	B_R for N=512
2	2304
4	576
8	192
16	72
32	29

Now, it is observed from Table 4.7 that something around radix-16 would serve the purpose.

Moreover, since 512 is not in the form of powers of 16, a mixed radix would be required, and we would have a stage of radix-2 and two stages of radix-16. We can call it a 16-16-2 FFT.

4.4.1.1 Split Radix vs Single Radix

Different types of architecture provide different benefits. But it is commonly accepted that split-radix algorithms which realize a larger radix by using butterflies of smaller radices are an optimum choice. A radix-R butterfly would require $(R-1)$ additions, R number of times, totaling to

$$Add_R = R(R-1)$$

For the same case, a split radix-r implementation would require

$$Add_r = (log_r R)\,(R/r)$$

Assuming $r = 2$, i.e. radix-2 as the split radix, Table 4.8 is formed.

Table 4.8: Number of adders required for split radix

R	Add_R	Add_r(r=2)	Add_r(r=4)
2	2	2	-
4	12	8	12
8	56	24	-
16	240	64	96

Additional advantage would be the possibility of the introduction of scaling units in the split radix stages, which would not have been possible in direct implementation. An increase in the number of stages is a disadvantage. Nevertheless, the movement of data here is local in nature in the split radix stages. Hence, a cache-based approach serves to reduce both the memory accesses and the cache further acts as pipelining registers, improving the throughput to a level as provided by a direct radix implementation or even better.

4.5 Design Methodology

A memory-based architecture has been adopted for the proposed design. The design procedure commences with the selection of the radix algorithm as outlined in section 4.4.1. The

steps followed for deriving the detailed data flow of the algorithm are discussed in this section.

4.5.1 Design Equations

Considering the DFT equation:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N} \tag{4.3}$$

where n and k represent the input and output indices respectively, n and k need to be split into appropriate equations to satisfy the radix requirements. The n and k split equations are derived as follows:

After the main radix is chosen, the number of stages along with their associated radix is calculated. Let i denote the stage number, r_i the associated radix, and S the number of stages. The radices satisfy the equation

$$N = r_1 r_2 \ldots r_S$$

The variable n_i, $i = 1, 2, \ldots, S$ are set to values $0, \ldots r_i - 1$. Now, the input equation for n is formed so that the final value of $n = N - 1$ can be constructed by summing up the final values of n_i with appropriate multiplying factors.

To derive the output index k, the indices k_i are set to the corresponding n_i values. For k, the multiplication factors are decided so that the i^{th} terms of n and k, multiplied together, become equal to N/r_i. Now, simple algebraic manipulations can be done to derive the design equations. Once it is decided that 512 points must be realized by 16-16-2 stage breakups, the n and k are split as outlined in the steps above.

Step 1 $n_1 = 0, \ldots, 15$, $n_2 = 0, \ldots, 15$, $n_3 = 0, 1$; $k_1 = 0, \ldots 15$, $k_2 = 0, \ldots 15$, $k_3 = 0, 1$

Step 2

$$N - 1 = (32 \times 15) + (2 \times 15) + 1$$

Hence

$$n = 32n_1 + 2n_2 + n_3$$

For finding the multiplier for n_1, we note that to form 512, we need 32×16. Now since maximum value of $n_1 = 15$, the RHS falls short 1 times 32. So this 32 is contributed by other terms, i.e., remaining n_i's.

It is interesting to note that the first multiplier is formed by combining the rest n_i's.

Step 3 The proper factor for k_1 is $(N/r_1)/32$=1; for $k_2 = (N/r_2)/2 = 16$; for $k_3 = (N/r_3)/1 = 256$. Thus, $k = k_1 + 16k_2 + 256k_3$. It would be illuminating to verify that the maximum value of $k = N - 1$ can be obtained by putting the maximum values of the various k'$_i$s.

Now, the FFT design equation can be obtained by substituting these values of n and k in the DFT equation and performing algebraic manipulations. Only two n's shall be present in each step. So in first stage all n's except n'_1 are combined as n'_2. For our example, the following design equation is obtained

$$X(k_1 + 16k'_2) = \sum_{n'_2=0}^{31} \sum_{n_1=0}^{15} x(32n_1 + n'_2) W_{512}^{(16k'_2+k_1)(32n_1+n'_2)} \tag{4.4}$$

where $n'_2 = 2n_2 + n_3$

$k'_2 = k_2 + 16k_3$

4.5.2 Butterfly diagram

The design equations form the basis of the FFT architecture. From these equations, the data flow diagram as shown in Fig. 4.6 has to be derived. The data flow diagram can be easily organized in stages where $i = 1, 2, \ldots S$ is the stage number with n_i as input to the stage and k_i as the output of the stage. There are a number of PEs in each stage. A PE refers to a butterfly (BF) with input and output derived as described below:

(i) The inputs of a butterfly in the first stage are found by changing n_1 and keeping n'_2 as constant. Change of n'_2 creates a new butterfly. Hence, the first stage butterflies can be numbered on the basis of n'_2. The outputs of these butterflies are numbered on the basis of k_1.

Expanding n'_2 in Equation 4.4, we get

$$X(k_1+16k_2') = \sum_{n_2'=0}^{31}\left(\underbrace{\sum_{n_1=0}^{15} x(32n_1+n_2')W_{16}^{(n_1k_1)}}_{\text{16-point DFT of stage 1}}\ \underbrace{W_{512}^{(n_2'k_1)}}_{\text{twiddle factor of stage 1}}\right)W_{32}^{(n_2'k_2')} \quad (4.5)$$

$$= \sum_{n_2'=0}^{31}\left(BO_{16}^{(1,n_2')}(k_1)\right)W_{32}^{(n_2'k_2')} \quad (4.6)$$

Where $BO_{16}^{(1,n_2')}(k_1)$ represents the k_1-th output of the first stage's n_2'-th radix-16 butterfly operation.

(ii) In the second stage n_1 has been replaced by k_1 as variable. Breaking n_2'& k_2'now.

$$X(k_1+16k_2') = \sum_{n_3=0}^{1}\left(\underbrace{\sum_{n_2=0}^{15} BO_{16}^{(1,2n_2+n_3)}(k_1)W_{16}^{(n_2k_2)}}_{\text{16-point DFT of stage 2}}\ \underbrace{W_{32}^{(n_3k_2)}}_{\text{twiddle factor of stage 2}}\right)W_2^{(n_3k_3)} \quad (4.7)$$

$$= \underbrace{\sum_{n_3=0}^{1}\left(BO_{16}^{(2,2k_1+n_3)}(k_2)\right)W_2^{(n_3k_3)}}_{\text{2-point DFT of the 3rd stage}} \quad (4.8)$$

$$= BO_2^{(3,16k_1+k_2)}(k_3) \quad (4.9)$$

According to the equation, for a particular value of k_1, n_2 and n_3 will change. A change of n_2 will lead to the formation of a butterfly. Change of n_3 will lead to the formation of a different set of butterflies for each value of k_1.

(iii) For the third stage, n_1 and n_2 are now gone and replaced by k_1 and k_2; change of n3 will form a butterfly; change of k_2 will lead to different butterflies for each value of k_1.

(iv) In the final stage, all of n's are now replaced by k's. The outputs are represented by k_3 for each value of k_2 and k_1.

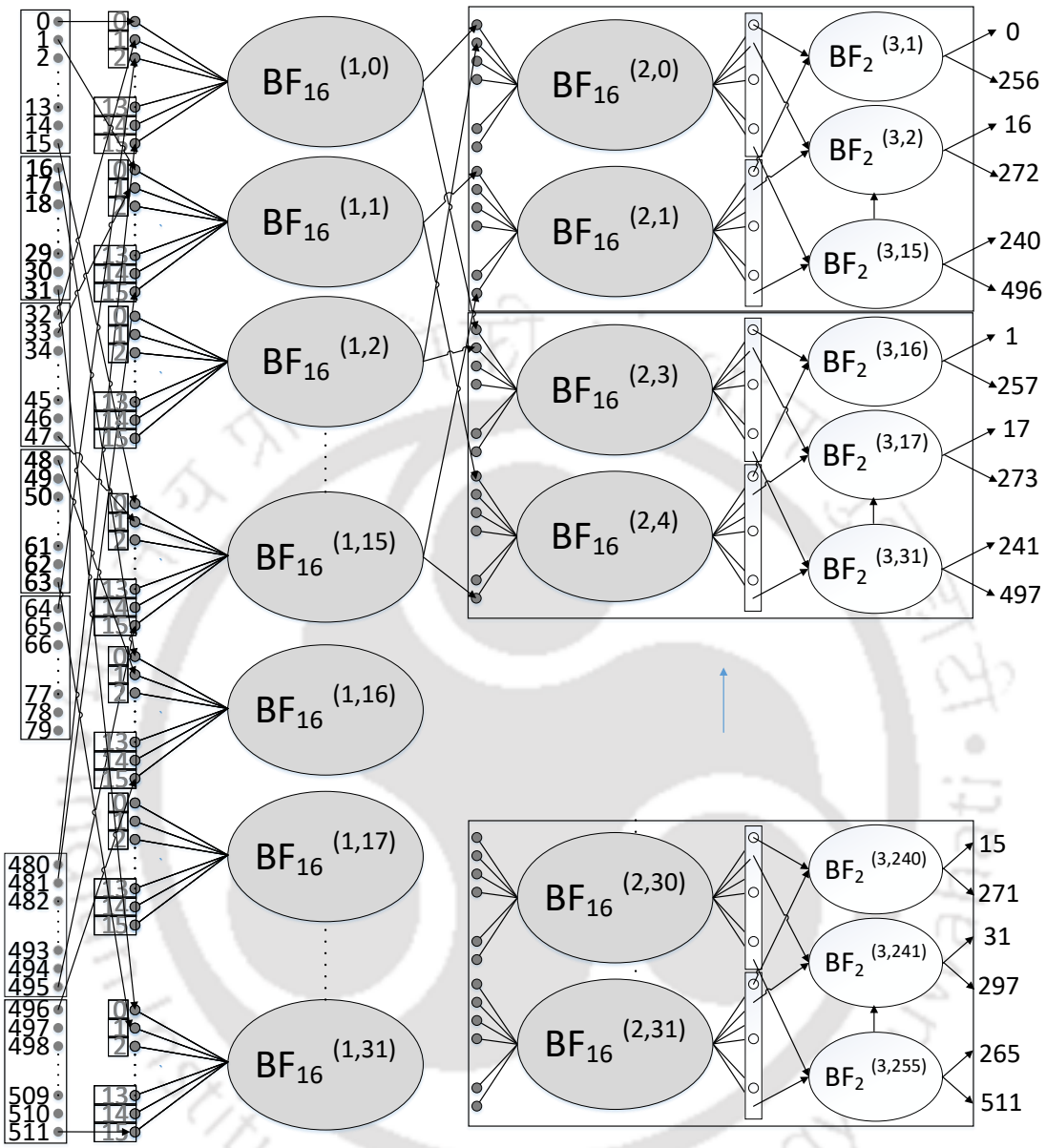


Figure 4.6: DIF radix-16 BF stage

4.5.3 Order of Outputs

The output index is derived from the output equation of radix-16 butterfly i.e.,

$$k = k_1 + 16k_2 + 256k_3 \tag{4.10}$$

To find the values of the various k_i's, three things have to be considered as described below:

Table 4.9: Number of bits for output index k

k_i	Range	No. of bits required	Bits
k_1	0-15	4	b_8 b_7 b_6 b_5
k_2	0-15	4	b_4 b_3 b_2 b_1
k_3	0-1	1	b_0

Table 4.10: Placement of bits to derive output index k

k_i	Multiplier	Shifting units	Bit positions								
			8^{th}	7^{th}	6^{th}	5^{th}	4^{th}	3^{th}	2^{th}	1^{th}	0^{th}
k_1	0 (2^0)	0						b_8	b_7	b_6	b_5
k_2	16 (2^4)	4		b_4	b_3	b_2	b_1				
k_3	256 (2^8)	8	b_0								
k			b_0	b_4	b_3	b_2	b_1	b_8	b_7	b_6	b_5

(i) Order of bits: The order of bits starts from k_3 (LSB) to k_1 (MSB).

(ii) Number of bits: The number of bits for each k depends on range of the k's as seen in Table 4.9.

(iii) Placement: Initially, the LSB of k_i is placed at the 0^{th} position in the binary expansion of k, and afterwards it is left-shifted by n bits, where 2^n is the multiplier of the k_i as shown in Table 4.10.

4.6 RTL Design Approach

The RTL design process uses a progressive flow starting from Matlab reference model and ending in the actual physical design. The advantage of this progressive flow is that each step has a predecessor which serves as a reference for debugging. The design methodology of such a complex DSP system starts with the creation of a reference model in a language such as Matlab. Any other language with fixed-point manipulation capability would be equally suggested. For example, Octave has fixed-point toolbox although with a different format and lesser functionality. We use the Matlab fft function as the golden reference for comparing all

our designs.

The Matlab code is much easier to write because it's not required to deal separately with real and imaginary parts. Even complex multiplications are carried out without any explicit knowledge on the part of the programmer. So at this stage, the designer can focus on generating the correct addresses of the inputs for each stage. A butterfly function would then generate the outputs which would appear in some random order. Hence, the designer should write necessary code to convert the output stream to natural order.

In most cases, a single set of for loops would suffice to perform the butterfly operations. In some cases, it would be better to perform different stages in different loops in a manner so as to reflect the different stages of processing as decided in the hardware architecture. Such type of loop change may be needed, when different stages have different addressing requirements, PEs, multipliers, scaling factor, etc. Different loops for different stages should be used only if necessary, as it would unnecessarily introduce more effort.

4.6.1 Matlab Floating Point to Fixed Point

The various loop variables and architectures having been finalized in the previous step, the code is now re-written to explicitly show the real and imaginary parts of the variables and twiddle factors separately. Thus complex multiplications would have to be represented explicitly by using real and imaginary variables. The names of the various variables should be made similar to the ones planned for RTL code; this similarity will be useful during debugging. The Matlab code with explicit real & imaginary parts is now converted to fixed point. It's a simple process, wherein the wordlength and fraction length are decided and the input data is converted to fixed point. Explicit fixed point conversions and configuration options would be needed to make the code behave as in RTL. Once this is done, the code is now verified with the results of the fft function. In SNR calculation, the output of fixed point data is the signal and the difference between these fixed point values and floating point values is the noise. The ratio of the RMS of signal to noise is the SNR. The fixed point model serves good use in changing the values of wordlength and seeing the effect on SNR for several set of input conditions.

This code now serves as the reference. Matlab being an interpreter based language, its debugging features are very useful during comparison with RTL simulation. All RTL designs are verified with this and discrepancies between RTL and Matlab fixed point simulations must be resolved by either reconfiguring the Matlab code or the RTL code.

While writing the HDL code, the following important points are taken care of:

- **Hierarchical Structure**: The verilog code is written in a hierarchical manner with various top modules and sub-modules.
- **Coefficient ROM generation**: The ROM data for HDL may be generated directly from Matlab using the file I/O operations and string functions of Matlab. These coefficients are stored in a look-up table.
- **Complex multiplier using three multipliers**: Let us consider the multiplication of x=a+ib, y=c+id. Direct multiplication uses 4 multipliers.

$$Z = (ac - bd) + i(bc + ad)$$

However complex multiplication can be carried out using only three real multiplications, while increasing one adder.

$$\begin{aligned} Real[(a+ib)(c+id)] &= ac - bd \\ Imag[(a+ib)(c+id)] &= (a+b)(c+d) - ac - bd \end{aligned}$$

- **Constant CSD multiplier using Matlab and fixed point**: Routines for converting a binary number to its Canonic Signed Digit (CSD) counterpart are easily available or can be written. Once the hardware for CSD multiplication is written, it has to be compared with the Matlab fixed point results for the same.
- **HDL Simulation and comparison with Matlab results**: Verilog $fmonitor statements or something similar can be used to store results of simulation in a text file in

hexadecimal format. Hence, a physical eyeview of the results will give only an approximate comparison with Matlab Hexadecimal results. It may not always be possible to get one-to-one bit equivalence in Matlab and HDL, because of the use of certain hardware constructs such as three multiplier based complex multiplication in HDL, whereas the same multiplication in Matlab is done using the multiplier operation which may behave slightly differently. Since, hexadecimal eye-based comparison would not be such a good idea, a decimal-based comparison is always a better way. For this, the Xilinx hexadecimal text data should be imported into Matlab and read as the hex data to a fixed point variable using some variant of the following code.

```
a=fi([],1,dig,frac);
for i=1:length(imported_data)
a.hex=imported_data_in_hex(i,:);
data_vec(i)=double(a);
end
```

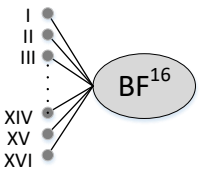
The results can then be plotted on the same figure as the Matlab results and, thus comparison is much simpler.

- **HDL coding**: There are some designs which are generic in nature and can be adapted to various sizes of FFT just by changing some variables. One such example is a simple radix-2 FFT. While coding such designs in HDL, parameterized variables should be used. The design should initially be done at lower number of points, where it is easy to debug. After this, the parameter values should be changed to reflect the actual design, and the code should be re-verified with the Matlab model.

- **Use of simple memory models**: We would like to suggest here that, the first round of simulation may be performed using simple memory structures. Once results have been obtained, these memories may be replaced with the actual models of memory. This

Table 4.11: Required data indices for first stage to be read from input memory

BF	Input number				
	I	II	III	...	XVI
BF_0	0	32	64	...	480
BF_1	1	33	65	...	481
BF_2	2	34	66	...	482
...	...	...	...	...	...
BF_{15}	31	63	95	...	511



approach helps to concentrate on design functionality, as actual memory models contain a lot of timing data and signals, and may require too much initial effort, which can actually be put towards the end. In the initial phase, the simpler architecture would allow one to compare the results of the HDL simulation with Matlab fixed-point simulation and fix the design errors. Afterwards, this project would become a golden reference, against which the complete synthesis with actual library models would be done. Any errors introduced due to memory or other macro complexity, in the final project, could be compared with this model and, hence corrected.

4.7 Proposed Conflict-free Addressing: Circular Progressive Shifting

Conflict-free addressing requires that data which is required simultaneously at the PE inputs, should be available at the various outputs of the memory bank. The memory bank must hence be filled up with incoming data in such a way that data required simultaneously is stored in different banks of the memory. In our case, the incoming data comes in a natural order. The inputs to the first PE are required in the order as shown in Table 4.11. A simple straight fill-up of memory would lead to the storage as shown in Table 4.12.

Since all the input data required at the same instant are stored in same column (each column represents a bank), the simplest solution would be to shift them all to different columns. It can be achieved simply by progressively shifting each row of data by one additional column as we progress down the rows. This shifting has to be done in a circular manner. Such shifting can be

Table 4.12: Data indices stored serially in input memory causing conflict in simultaneous access

		Memory Bank number (Ref Fig. 4.5)				
		I	II	III	...	XVI
Address	0	0	1	2	...	15
	1	16	17	18	...	31
	2	32	33	34	...	47
	3	48	49	50	...	63
	4	64	65	66	...	79
	...	...	...	...	...	...

Table 4.13: Data indices stored after progressive shifting: Allows conflict free access

		Memory Bank number (Ref Fig. 4.5)						
		I	II	III	IV	V	...	XVI
Address	0	0	1	2	3	4	...	15
	1	16	17	18	19	20	...	31
	2	47	32	33	34	35	...	46
	3	63	48	49	50	51	...	62
	4	78	79	64	65	66	...	77
	...	...	...	...	...	...	...	,,,

denoted by the term "Progressive Circular Shift"; the term progressive denotes the increase in shift amount as we progress. Since this memory feeds only one stage, simple logic is sufficient to decide the banks. The resulting configuration of data is shown in Table 4.13.

The theoretical support for the addressing scheme can be derived by using a formulation similar to the one used in [27], where the bank is found by summing the characteristic stage variables as shown below:

For an N point FFT, let N be factorised as

$$N = N_1 N_2 N_3 ... N_s \tag{4.11}$$

In our case $N_1 = 16$, $N_2 = 16$, $N_3 = 2$. We define the characteristic variable n_s of each stage s, as

$$n_1 = 0...N_1 - 1, n_2 = 0...N_2 - 1, n_s = 0...N_s - 1 \tag{4.12}$$

The incoming data index n is written in terms of the characteristic stage variables as

$$n = 32n_1 + 2n_2 + n_3 \tag{4.13}$$

Then bank can be determined for a memory which feeds all stages from

$$Bank = (n_1 + n_2 + ...n_s)\, mod\, B \tag{4.14}$$

If it is ensured that a memory bank feeds only a single stage, then for feeding a particular butterfly, only one stage variable will change; others will remain constant. This stage variable will increase by 1 to get a new input to the butterfly, changing the bank value by 1. The bank increment by 1 leads to "Progressive Shifting". Since the address required at each memory would be different, 16 address generators are required to generate the addresses in the memory. The design of the address generators is a subject of section 4.8.

4.8 Continuous Flow Addressing: Replacement Addressing

This section proposes a new method of Continuous Flow using replacement addressing rather than in-place addressing. When data is read out for processing, new data is written at the vacated address rather than the processed data. The vital thing to note here is that the order in which data is read differs from the order in which data is written. In this design, data flows into the memory in a natural order. On the other hand, it is read out in an order determined by the equations of the algorithm as in (4.2), which is referred to hereafter as algorithmic order. When this addressing order is analyzed in terms of bits, it is seen that the bit-order is not symmetric.

For each bank of memory, one set of addresses is required to write data, while for reading out data, a different set of addresses is required. The locations vacated by reading out data are filled in by naturally incoming data. The data of the second symbol are now written at different addresses compared to the first one. A different address set is now required to read the data from this new order in the algorithmic sequence. It is highly desired that the new

address be same as the input read address of first symbol to avoid generating a large number of address sets. To achieve this, the input data has to be reordered to fulfill certain conditions of symmetry. We propose a condition for CF in such a case. This condition ensures that two sets of addresses are sufficient, which are to be used alternately during iterations of the processor. For the first iteration of the processor, the first address set is used. After replacement, the second address set is used, and in the next iteration, the first address set is reused and so on. The reordering has to take place such that the following condition is satisfied.

4.8.1 Proposed condition for CF

Let the incoming bit-order be denoted by a_i. In this case, for generating 512 data indices,

$$log_2 512 = 9$$

bits are required, so that the incoming and outgoing bit orders are both 9 bits. Hence,

$$a_i, i = 0...8, \text{ and}$$

the outgoing bit-order is

$$b_k, k = 0...8,$$

Then:

If a_m is replaced by b_n, then a_n is to be replaced by b_m as shown in Fig. 4.7.

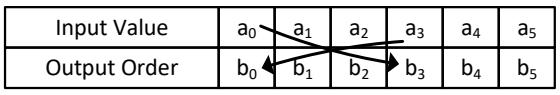


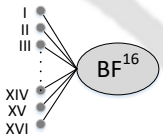
Figure 4.7: Bit-order modification for Continuous Flow

Bit-value	256	128	64	32	16	8	4	2	1
Natural input order	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
Required order	b_3	b_2	b_1	b_0	b_7	b_6	b_5	b_4	b_8
Reordered for CF	b_3	b_2	b_1	b_0	b_4	b_8	b_7	b_6	b_5

Figure 4.8: Modified bit order for feeding the first stage radix-16 BF

Table 4.14: Modified order for feeding the inputs of the radix-16 BF^{16} at stage 1

BF	Input number				
	I	II	III	...	XVI
BF_0	0	32	64	...	...
BF_1	16	48	80	...	...
BF_2	1	33	65	...	...
BF_3	17	49	81	...	...
BF_4	2	34	66	...	...
...	...	...	...	...	...



4.8.2 Input Stage

The first set of address logic is a straight binary counter whose output is used as write addresses for all 16 units. Then the read addresses required are derived by expanding the binary address logic of the row titled "Reordered for CF" of Table in Fig. 4.8. This results in Table 4.14, which is the order in which processing would be done. Since the actual storage is as in Table 4.13, read addresses are derived from both the tables and collected for all the memory banks as shown in Table 4.15. The write addresses are delayed versions of the read addresses.

These addresses could easily be generated by storing the even and odd sequences separately and shifting them when required, as illustrated in Fig. 4.9. After reading out data from Table 4.13, the vacated locations are filled with serially incoming data after progressive shifting. This leads to the new memory map as shown in Table 4.16. The input data can now be read out by using a straight binary counter. Thus, two alternating sets of addresses that are simple to generate are all that are needed for 1^{st} stage.

Table 4.15: Read Addresses

Bank	Clock cycles				
I	0	1	30	31	...
II	2	3	0	1	...
III	4	5	2	3	...
IV	6	7	4	5	...
...	...	...	...	...	...

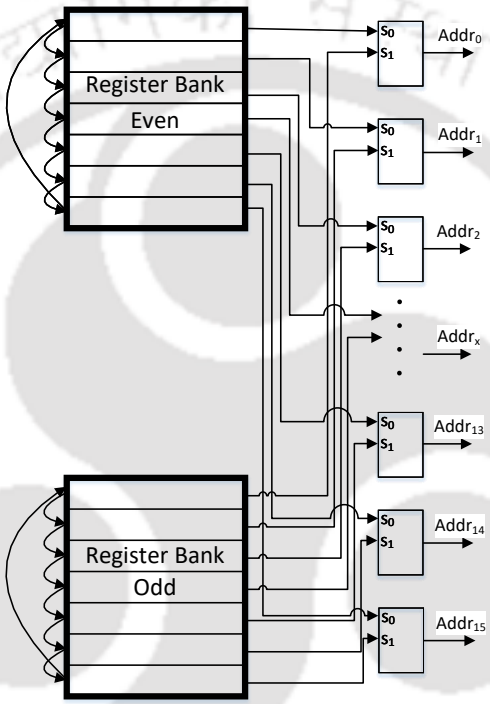


Figure 4.9: Address generation for first stage of the 512-point radix-16 FFT

4.8.3 Second Stage

From the flow diagram of Fig. 4.6, it is seen that the radix-2 butterfly of the last stage takes inputs from two adjacent butterflies of stage 2. Hence, the radix-2 processing may be carried out easily by storing results of only two adjacent BF's if it is ensured that the second stage BF processing is done in the order seen in Table 4.18 (read along the row). In the first stage, the order followed is different from normal to meet CF condition. It is shown in Table 4.17 as read along the rows, for first stage.

If the output of stage-1 is stored in normal order, then the outputs of BF_0 follow the order

Table 4.16: Memory units after progressive shifting

		Memory Bank number (Ref Fig. 4.5)				
		I	II	III	...	XVI
Address	0	0	32	64	...	480
	1	16	48	80	...	496
	2	481	1	33	...	449
	3	497	17	49	...	465
	4	450	482	2	...	418
	...	...	...	...	...	...

Table 4.17: Butterfly processing order followed for first stage (read row-wise)

0	16	1	17	2	18	3	19
4	20	5	21	6	22	7	23
8	24	9	25	10	26	11	27
12	28	13	29	14	30	15	31

0,1,..., 15, while outputs of BF_{16} are numbered 16,17,..., 31, and so on as shown in Table 4.19 (without shifting for conflict free addressing). From Tables 4.18 and 4.19, the required input order is derived and is shown in Table 4.20. To achieve this order, the required addressing logic is as shown in row 3 of Table in Fig. 4.10. However, this order does not fulfill the symmetry requirements of CF. Hence, it is necessary to modify the order. As it is not allowed to fix more than 50% of the bits, we can fix only 4 out of 9 bits. If we decide to fix the lower 4 bits b_0 to b_3, we have to reorder the remaining bits as per the replacement rule of CF addressing. It can be noted that a_8 is replaced by b_2; hence a_2 needs to be replaced by b_8; Similarly, a_7 is replaced by b_1, initiating a replacement of a_1 by b_7 and so on. The order is as shown in row 4 (Modified Order 1) of Table in Fig. 4.10.

Bit-value	256	128	64	32	16	8	4	2	1
Natural input	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
Required order	b_2	b_1	b_0	b_4	b_3	b_8	b_7	b_6	b_5
Modified order 1	b_2	b_1	b_0	b_5	b_3	b_4	b_8	b_7	b_6
Modified order 2	b_3	b_2	b_1	b_0	b_4	b_8	b_7	b_6	b_5

Figure 4.10: Second stage bit orders

But the butterfly *processing order* is modified in this case, which will create difficulty in processing the next stage. Therefore in this case, we observe that it is better if the MSB's

Table 4.18: BF processing order required in second stage (read row-wise)

0	2	4	6	8	10	12	14
16	18	20	22	24	26	28	30
1	3	5	7	9	11	13	15
17	19	21	23	25	27	29	31

Table 4.19: First stage outputs renumbered naturally before storage into second stage memory

BF_0	0	1	2	3	4	...	15
BF_{16}	16	17	18	19	20	...	31
BF_1	32	33	34	35	36	...	47
BF_{17}	48	49	50	51	52	...	63
BF_2	64	65	66	67	68	...	79
BF_{18}	80	81	82	83	84	...	95
BF_3	96	97	98	99	100	...	111
BF_{19}	112	113	114	115	116	...	127
BF_4	128	129	130	131	132	...	143
...	...	...	...	...	...	...	...

$b_8 - b_4$ which control the butterfly order remain constant, while the bits $b_3 - b_0$ which control the internals may move among themselves. Modified order 2 is thus generated. Incidentally, this processing order is the same as that of stage 1 and it is possible to use stage 1 addressing circuits for the second stage. After 32 cycles of first stage, when 2^{nd} stage starts, addresses also restart. Hence, the addresses generated by the first stage can directly be used by the second stage.

4.8.4 Output Stage

The output addressing unit takes its input from the "Radix 2 unit", and its purpose is to send out output data in normal order. The requirement is to read-out 16 normally ordered outputs from the 16 output memory banks in a single clock, and then use a 24-bit 16-input parallel-in-serial-out to send the FFT data out. The locations vacated by these 16 data have to be continuously filled in with algorithmically ordered data coming from the Radix-2 unit to maintain CF. The output data from the last stage PE (Radix-2 unit) comes out in the order as shown row-wise in Table 4.21 (shown for the first 6 Radix-16 outputs). To meet the CF

Table 4.20: Required input order for stage 2 after renumbering the output of stage 1 in natural order

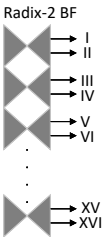
BF	BF input number															
	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII	XIV	XV	XVI
BF_0	0	64	128	192	256	320	384	448	16	80	144	208	272	336	400	464
BF_1	32	96	160	224	288	352	416	480	48	112	176	240	304	368	432	496
BF_2	1	65	129	193	257	321	385	449	17	81	145	209	273	337	401	465
BF_3	33	97	161	225	289	353	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

addressing requirement for this stage using only two alternating set of addresses, this output has to be reordered as shown in Table in Fig. 4.11. The reordering rule states that the output order should be bit symmetric to achieve CF. Since only less than 50% bits can be fixed, the criteria for fixing the bits has to be decided. Here, it is important that the bit reordering of addresses should not modify the butterfly processing order. Hence, the bits $b_8b_7b_6b_5$ which control the butterfly processing order are fixed as seen in Table in Fig. 4.11. The output from the butterflies may come in any order and can be taken care of by internal reordering. Hence, the remaining bits are reordered to satisfy the bit-symmetry rule. Here, b_4 also changes its position. Since it controls processing order, hence the output processing order is somewhat modified and will need reordering the buffers.

Reordering will require storing data blocks (two rows of data), and then performing inter-block switching. Storage of the 16 outputs of the radix-2 unit for two cycles is done in two buffers. While these two buffers are being processed, another set of two buffers is needed for storage. Multiplexers switch between the two sets of buffers both at the input and output of the buffers. Further, two sets of addresses are required for the two modes of the output address generator, as shown in Fig. 4.12. One address set can be generated by a simple counter. For the second address, it is noted that the output order is same as modified input order of the FFT. A circuit similar to the first stage addressing circuit can be used for the second mode of output address generator.

Table 4.21: Output order for mixed radix 16-16-2 for 512-points

Clock Cycle	Output number					
	I	II	III	IV	...	XVI
1	0	256	16	272	...	368
2	128	384	144	400	...	496
3	1	257	17	273	...	369
4	129	385	145	401	...	497
5	2	258	18	274	...	370
6	130	386	146	402	...	498
...	...	...	...	...	...	...



Bit-value	256	128	64	32	16	8	4	2	1
Natural output	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
Algorithmic order	b_0	b_4	b_3	b_2	b_1	b_8	b_7	b_6	b_5
Modified order	b_3	b_2	b_1	b_0	b_4	b_8	b_7	b_6	b_5

Figure 4.11: Re-ordering the data order to be stored in output memory to be bit-symmetric

4.9 Chip Design

There are four steps in chip design:

(i) Simulation and verification

(ii) Synthesis

(iii) Chip implementation

(iv) Testing

These steps are carried out using a variety of tools and are explained in detail below:

4.9.1 Simulation, verification and Synthesis

The simulation of the whole algorithm is performed in MATLAB fixed-point environment. The HDL is written in Verilog and simulated using Synopsys VCS tool. The test data taken is 64-QAM symbols with a uniform distribution. The processor may not work satisfactorily with other types of data as fixed scaling factors are introduced at several points assuming the OFDM data statistics. The verilog simulation results are brought back to MATLAB, and reconstruction is carried out using in-built MATLAB FFT function. The reconstructed data is

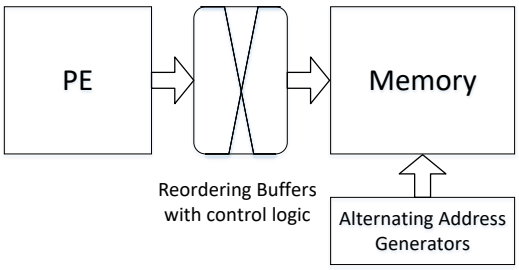


Figure 4.12: Reordering of PE outputs

plotted together with the original data and is shown in Fig. 4.13. Overall results are found to be satisfactory except for some deviations towards the edges. This is because radix-16 has been realized here by combining radix-4 PEs. The first output of a radix-4 PE uses only addition compared to the use of a combination of addition and subtraction at other outputs. As a result, round-off errors cancel each other at all outputs except the first. MATLAB simulations verify the magnitude of the errors to be within the acceptable limit for 64 QAM with an SQNR of around 37dB. Also, from matlab simulations, it is observed that by randomizing the addition process at the first output, an SQNR of 55 dB can be realized. This randomization was not carried out on chip as the solution for the low SQNR was not clear till chip fabrication. Synthesis from Verilog RTL to gate level netlist is done using Synopsys DC. The area of the various units is found separately and shown in Table 4.22. The overall area is found to be 4.3 mm^2, including all memories.

4.9.2 Comparison of synthesis results

The design is synthesized using Synopsys Design Compiler using SCL 180 nm library. The total area, according to DC optimized results, is 4.3 mm^2. The area is compared with other designs after adding a second memory for CF. Also, additional control and addressing circuitry are required, which are synthesized separately and their area found out as shown in the Table 4.23 for [2]. Similarly, it is assumed that the area increases linearly with the number of

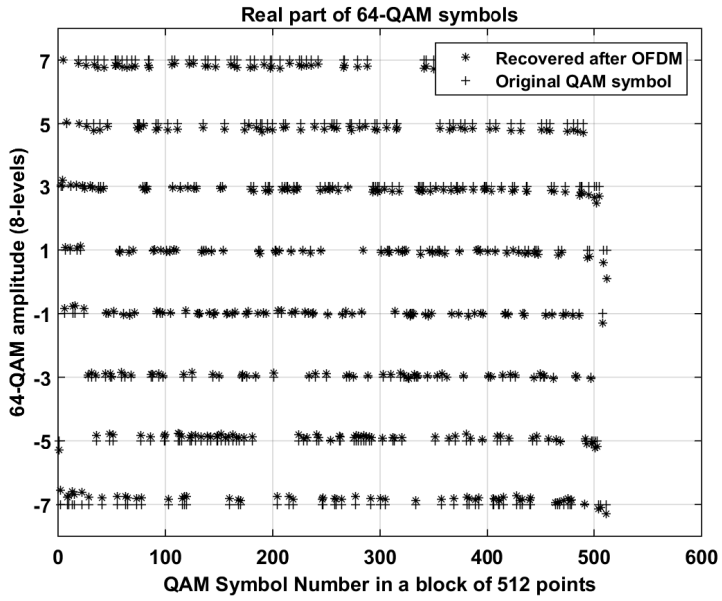


Figure 4.13: Recovered Signal after OFDM from RTL model

points, technology nodes and datawidth. Area is normalized using (4.15)

$$Normalized\,area_{ref} = \frac{Nand\,Gate\,Area_{180nm}}{Nand\,Gate\,Area_{ref}} \times \frac{FFTpoints_{512}}{FFTpoints_{ref}} \times \frac{Wordlength_{12}}{Wordlength_{ref}} \times Area_{ref} \quad (4.15)$$

$$FOM_{ref} = \frac{Area_{Our\,work}}{Normalized\,area_{ref}} \times \frac{Throughput_{ref}}{Throughput_{Our\,work}} \times 100\% \quad (4.16)$$

Comparison is made with the designs as shown in Table 4.24. The final comparison parameter is the Figure of Merit (FOM) as defined in 4.16 which can be thought of as a type of area efficiency per Gsps. It is observed that the area consumed by our design per Gsps is 50% lower than others, as indicated by the best FOM. This efficiency can be attributed to the lower complexity of switching circuits which include the commutators and multiplexers. Also, the decreased wordlengths of stage-1 and lower complexities of the PEs have compensated for the increased number of units adding only a little to the overall area.

Details of Chip implementation and testing steps are given in the next chapter.

Table 4.22: Synthesized Area for the present design

Design unit	Area μm^2
Input Right Circular Shift	48091
Input Memory Bank	265848
Input Left Circular Shift	48091
Input Memory Bank Address Generator	25825
Stage 1 Radix-16 Butterfly	228739
W_{512} Mulitiplier Set	713261
Stage 1 Right Circular Shift	109868
Stage 1 Memory Bank	813859
Stage 1 Left Circular Shift	128290
Stage 1 Address Generator	16214
Stage 2 Radix-16 Butterfly (buffered)	413902
Stage 2 W_{32} CSD Multipliers	138911
Stage 3 Radix-2 BF (32 units)	132399
Stage 3 Buffers and Muxes for reordering	124504
Output Right Circular Shift	109868
Output Memory Bank	813858
Output Left Circular Shift	128290
Output Address Generator	25825
Total design area	**4285642**

4.10 Conclusion

A high performance FFT architecture with double the existing throughput is presented with only 20% increase in area and without any increase in clock frequency. The FFT chip is a part of the Wireless Gigahertz system and would be used in a test system in the future. The FFT processor uses the characteristics of OFDM input signal to reduce its area footprint. Other designs use switching of memories to achieve continuous flow but this design integrates the memories as part of the architecture. The overall architecture is designed to allocate fixed roles to memories. The switching complexities then reduce significantly. The increase in the area due to added memories and PE for increasing the performance is further compensated by designing a novel conflict free addressing scheme. Also, a new understanding of Continuous Flow is developed to achieve a real-time processor. Specializing the FFT processor for applications can

Table 4.23: Area calculation for Huang's design [2]

Unit	Area at 90 nm (mm^2)
Memory	0.40
Processing units	0.52
Additional Memory	0.40
Additional area for addressing and control circuit	0.20
Total design area	1.52

be seen as a promising way to reduce pressure on area and performance budgets.

Table 4.24: Comparison of high throughput FFT processors

Parameter	[12]	[31]	[2]	[17]	[18]	[19]	[15]	[20]	Our work
Architecture	8-way MDF	MDF	Memory based	2-way MDF	Serial Commutator	In-Memory	Memory based	8-way MDF	Memory based
Radix	2^5	$2^4 - 2^2 - 2^3$	16-2	2^2	2^k	2	2-4-8-16	$2^4 - 2^3$	16-2
FFT points	512	512	512	1024	128	1024	4096	512	512
Word length	14	14	12	32	12	12	14	12	12
Technology node (nm)	90	130	90	65	90	65	65	90	180
Clock (MHz)	350	220	324	600	250	-	250	385	300
Throughput (Gsps)	2.6	1.76	2.4	1	1	1	1	3.08	4.8
Area at 180 nm	4.9	-	3.54	3.47	2.96	2.08	2.03	4.62	4.3
Figure of Merit (FOM)	46.43	-	59.32	25.22	29.56	42.07	43.1	58.33	100

5

Physical Design of 512-point FFT ASIC with Continuous Streaming

Contents

5.1 Introduction

The FFT chip designed in chapter 5 has been fabricated in SCL foundry using 180 nm and packaged in a 120-pin QFN. This chapter deals with the details of the implementation of the ASIC. The physical design of the proposed FFT processor is challenging because of the involvement of almost half a million gates, several SRAM 's and input/output (I/O) cells. Around ten to fifteen software EDA tools are required to convert RTL to GDS. This chapter collects the design methodologies and place and route (PnR) strategies required to implement the chip. Also, the physical choice of the memory elements and the architecture is closely related to the implementation strategies at the chip level. These have to be usually taken care of at the beginning of the design to arrive at a physically implementable architecture. A separate section has been dedicated in this chapter to illustrate the choices and their relationship to the physical implementation. There are challenges unique to this design to realize a physically working chip. More than 200 iterations of the design are accomplished to meet the timing and other performance parameters.

FFT processor design is a matured field with a large number of architectures published. Some of these designs have been realized on silicon. But very few articles dwell on the hurdles in the design in moving from RTL to the chip level. The RTL is written based on the FFT algorithm and has a general structure consisting of a processing element (PE), some storage, and I/O interfaces. Usually, there are multiple options at the physical level, which affect the timing, area, and power of silicon. These options are brought out in relevant sections. In the next few sections, the tool flow used for the design is discussed. The chapter concludes with a discussion on chip finishing, which is the final step before the release of GDS to the foundry.

5.1.1 Technology Node and Process Design Kit (PDK)

The first choice in physical design is the technology node and the process design kit (PDK) libraries. The selection of the technology node for the realization of the chip is on the basis of several parameters, the primary being cost and support from the foundry to the Institute. The foundry Semiconductor Laboratory with a 180 nm process node is selected for the present chip

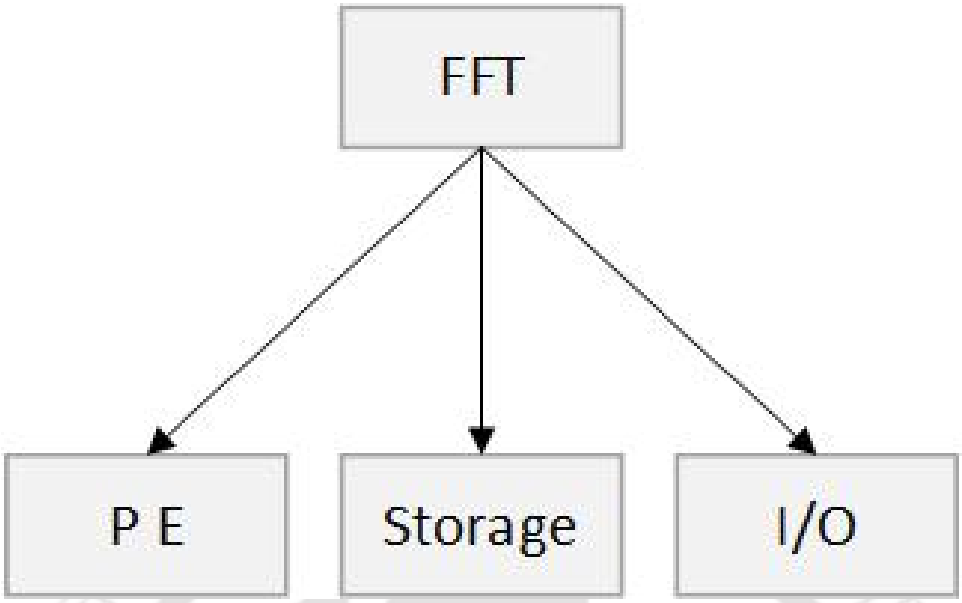


Figure 5.1: Components of FFT

as it is deemed sufficient for testing the algorithmic and architectural aspects of the design. The achievable clock speed is a function of the storage delays and the combinational delays, especially clock to Q delay and delay of NAND or inverting gate in the critical path. Every technology has a minimum value for these parameters, and hence the maximum clock speed is highly dependent on the choice of the technology node. For SCL technology, a 300 MHz clock frequency seemed achievable at the intellectual property (IP) level. At the chip level, it is expected to come down to 100 MHz due to limitations of I/O pads.

5.2 Storage

The type and amount of storage are dependent on the architecture and access type requirements. High speed, small memory, and multi-port access dictate the use of register memory, while for larger sizes, SRAM is preferred. In order to get the proper sizes of SRAM memory, a memory compiler may be available from the foundry. Otherwise, memory cuts of some particular sizes may be made available by the fab. The physical design and RTL have to consider these sizes to come out with a practically possible design.

5.2.1 Memory Compiler for SRAM

SRAMs are usually designed with two options: Single port and dual port. Dual port memory is very popular in FFT processors because it is very convenient to read from it and write to it simultaneously. But if dual port memory is not available from the foundry, it is best to avoid it right at the design stage itself. The designer would be better off using single port memory and switches instead. When both are available, then in order to be able to choose between them, the area is a major consideration. For the calculation of the area of various sizes of SRAMs, it is convenient if a memory compiler for the technology node is available. A memory compiler is a tool for designing memories and their related libraries at the designers' computer for a large number of useful memory sizes. For the SCL foundry, the memory compiler is not available, so a generic memory compiler (32 nm) available from Synopsis is used. A detailed study using a 32 nm memory compiler is carried out to compare SRAM area with register memory. It is assumed that relative scaling would lead to similar results at 180 nm. From the study, it is concluded that even with dual port memories, SRAMs are 33% area efficient compared to register memories. Hence, the storage elements for the present design are chosen as SRAMs. Three memories are needed, one each for input, computation, and output. Memory had to be chosen from the available designed sizes. This knowledge is required right at the beginning of RTL design. Otherwise, if some size of memory is assumed and it is not available from the foundry, the radix and hence, architecture may need a complete change. Also, it is good to have memory models at the RTL design phase itself to reduce unnecessary RTL changes later on. Every change in RTL is usually followed by extensive verification and increases design time and effort. In the present case, 16 memory cuts of 32×8 bits wide are used to realize the input memory of 512×8. For the computation and output memories, 48 such memory cuts are used to realize 512×24 bit memory. So the total memory cuts used in the design are 112.

5.2.2 HDL models of memory

As memories are inherently analog blocks, to do a digital simulation, their simulation models are used. HDL models of memory are used later in the simulation to confirm that timing

parameters are met at the simulation level. Also, these models are needed to do the final simulation after the layout. So, it is imperative that all the various models are available from the foundry to make sure that the design process does not stall at a later stage.

5.3 Input / Output interface (I/O) and clocking

Basically, a high-speed FFT chip is a mixed signal design. The I/O runs at multi-gigahertz speed, while the actual FFT core runs at a few hundred megahertz. The interface between the real-world bit-stream and computation engine is typically a Serializer-Deserializer (SERDES). Multi-bit deserializers are used at the input to convert the incoming serial stream into a parallel stream, which is then fed to the digital core. Similarly, at the output of the FFT core, deserializers would be required to convert the data to a serial stream. SERDES are inherently analog blocks and must be available from the foundry as an IP. In their absence, digital Serial-in and Parallel-out (SIPO) and Parallel-in-Serial-out (PISO) would be used, which are inherently slow, being combinations of a large number of digital gates. In the present design, they are used instead of SERDES for proof of concept.

It is preferred to have an On-chip Clock available as an IP from the foundry. A clock fed from outside the chip will have limitations due to I/O pads until, of course, sufficiently high-speed pads are available. Even if the technology allows high-speed clocking, the data needs to be fed in and out at the same high speeds. If the speeds are envisaged in the Giga-Hertz range, then LVDS pads would be required as I/O pads. Such pads will allow data in and out of the chip at a multi-Giga-Hertz rate. Otherwise, the speeds would be limited to a few hundred megahertz only.

5.4 Physical Design

The technology library used for fabrication is SCL 180 nm [86] with 1P6M layers. A number of tools are used for generation of GDS from the RTL as shown in Fig. 5.2.

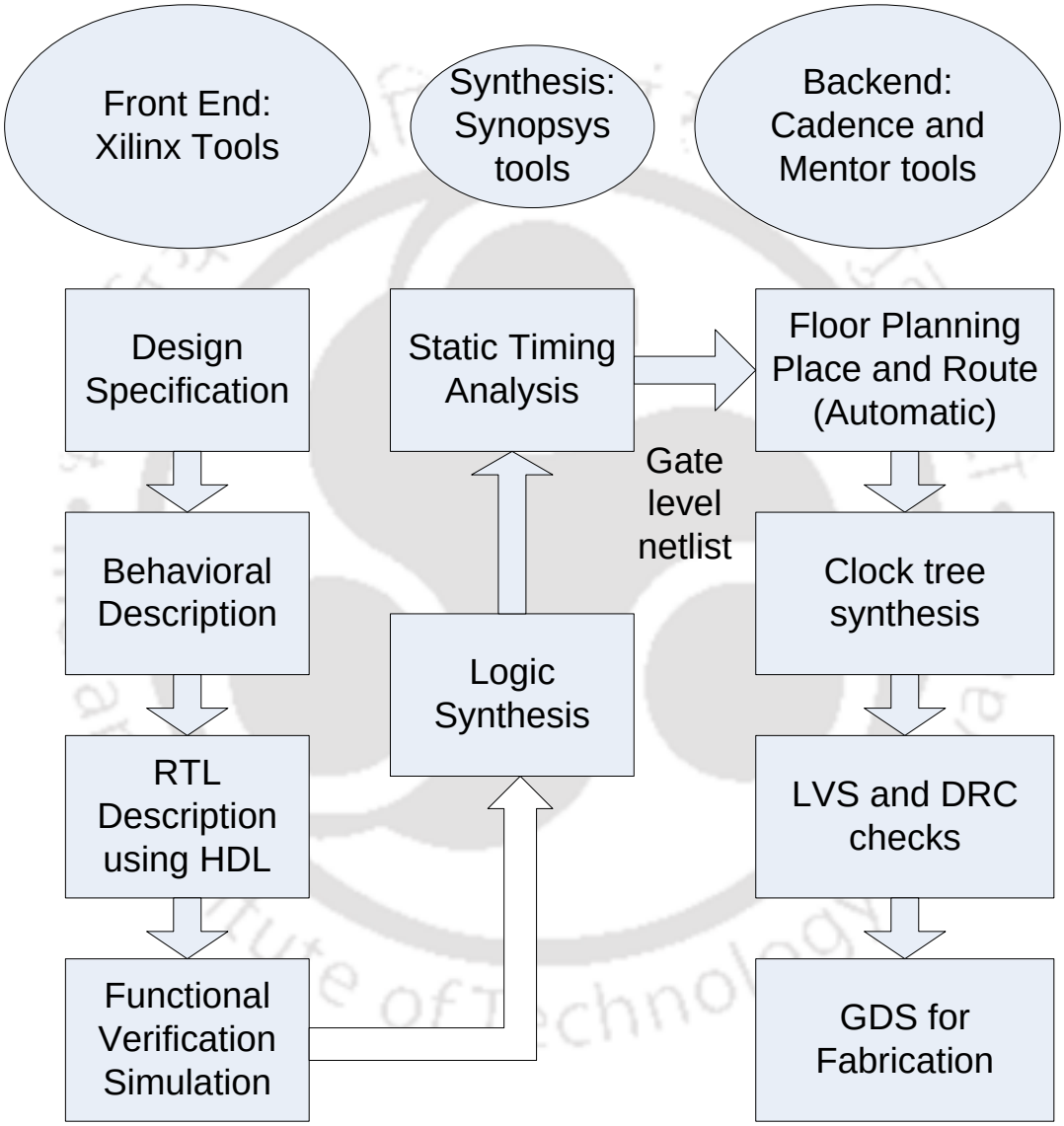


Figure 5.2: Tool flow of physical design

5.4.1 Steps in Physical Design

The physical design consists of 3 stages:

(i) Synthesis and timing closure

(ii) Placement, Floorplan, Clock tree synthesis and Routing (PnR)

(iii) Verification, Parasitic Extraction, Static timing, and post-layout simulation

5.4.2 Synthesis and Timing closure

This is the first step in converting the RTL to ASIC. The most common tool for this purpose is Synopsys Design Compiler or its equivalent. The output of this step is a netlist which is fed as input to the PnR step. The specific points in relation to this step worth considering are listed below:

- **Synthesized area and actual area**: The area given by the synthesis tools, such as the Design Compiler from Synopsys, is a sum of the areas of the various combinational and sequential elements and any other IP used in the design. The interconnect area is not estimated at this point. The actual core area is dependent on two factors:

  (i) Whether the design is core limited or Pad limited: In a limited core design, the area is controlled by the area given by synthesis tools. Usually, only 30 to 50% area is occupied by the cells, and a balanced area is required for routing. Whereas in a pad-limited design, the core area is already fixed and large enough for both standard cells and routing to fit in. The area given by synthesis tools is rendered meaningless in this case.

  (ii) Number of layers available for routing: The more the layers available, the lesser would be the overall physical area required.

- **Registered outputs for individual modules**: Sometimes, it may be necessary at the physical design level to carry out a bottom-up approach where smaller modules have to

be optimized separately. It is good in such cases if the individual modules have their output registered. This would make it easier to identify input-output constraints at the module level. In case the outputs are not registered, it would be difficult to predict the delays, and more effort would be necessary in such cases.

- **Timing Closure with min-max analysis**: In order to make sure that the chip performs in all environments, a min-max analysis may be carried out for the extreme operating conditions. Timing closure then involves meeting the setup time. The hold correction involves inserting delay cells. The same delay cells would have a lower delay in hold analysis and higher delay in setup analysis. Until the setup slack is high enough, hold corrections would easily lead to setup violations. This finally leads to lower clock frequencies.

- **I/O delays**: The initial input and output delays may be kept as 50% of the clock period. This rule can be used as a guideline when deciding on initial constraints for synthesis tools. The delays make sure that the input and output signals have sufficient time to reach their destination. This helps to design the input and output interfacing circuits. Also, max and min values have to be specified, which are in the same ratio as the ratio of delays of the standard cells in the min and max libraries or as per the ratio in a conservative approach.

- **Comparison of area using Nand / Inverter gate size**: Many authors use the following formula for comparing the results of one technology with another [33].

$$Area(newtech) = (Area(oldtech))/(newtech/oldtech)^2 \tag{5.1}$$

It is suggested that the ratio of areas of the basic gates should be taken as a base for an accurate result. Some examples are given in Table 5.1& 5.2.

Also, by comparison of an FFT design consisting of both memory and standard cells, we actually find their area as shown in Table 5.3.

Table 5.1: Area comparison of technologies at Gate level

New tech, old tech	(new tech/old tech)^2	Ratio of area of gates
180nm SCL foundry, 32nm Synopsys generic library	36	12.54/2.28=5.5

Table 5.2: Area comparison of memory sized 32 x 24 bits

180nm SCL foundry (μm^2)	32 nm Synopsys generic library (μm^2)	Ratio of area
50000	7361	7

Table 5.3: Comparison of chip area for two technologies

180nm SCL foundry (mm^2)	32 nm Synopsys generic library (mm^2)	Ratio of area
4.2	0.7	6

5.4.3 Design for testing (DFT)

DFT is done to test for manufacturing faults in a design. Standard tools are available for this. After an HDL design has been finalized, DFT logic is introduced into the design. If DFT is to be used later during physical design, it would be good to plan it at the RTL stages. The major changes would be making sure that the global reset reaches all the flip-flops in the test mode. Also, if there is more than one clock, then in the test mode, only the test clock should reach all the modules. This further entails including extra pads in the design for Test-Pattern IN, OUT, and Test-Mode-Enable. Provision must be made in the pad layout for these pins. Also, design modifications may be needed at the HDL level. Such modifications may introduce errors in the design. Hence, after DFT, the design should be rechecked for tallying with the original fixed point reference model or a golden HDL model. The few points of importance that we would like to point out for this design are as follows:

If there are two clocks in the design, only one is used for DFT. This should be made active in the TEST mode in the HDL code itself, if possible. If MBIST is available, memory level

testing can be done; else, memory would need to be enclosed in a wrapper and bypassed in TEST mode. This wrapper may be created by the tool or handwritten by the designer. The second process may introduce changes to the HDL code, and hence, the project would need to be re-verified.

5.4.4 Floorplan, Clock tree synthesis, and Routing

This is the most difficult step in the flow. It usually requires technology and tool-specific variables, which are available only to experts in backend flow. Usually, PnR scripts are developed with their help; nevertheless, several things are commonly done to avoid pitfalls later. Some experiences are enumerated below:

- **Floorplan and Power pads**: If the FFT processor is to be packaged as a single unit, it will have I/O pads. The I/O pads will draw large amounts of currents on switching. Also, the core would need the power to do these operations. Hence, a proper number of power pads need to be placed on the chip.
  We outline here the general guidelines for placement and routing of the FFT processor, assuming it has macros of memory while the clock is fed from outside.

  (i) Since there will be two clocks, it is always better to derive one from the other by clock division.

  (ii) The second clock being a generated clock, it must be declared so at the synthesis level, as the placement system may otherwise be unaware that the signal coming out of the clock divider is actually a clock.

  (iii) The placement system treats all cells with or above a particular size as a macro. Hence, all memory elements are invariably macro.

  Routing wires over macros are usually not allowed. Hence, in general, macros should be placed near the edges of the core area to avoid obstructions in routing, as shown in Figure 5.1. If the memory has been constructed by using several memories, a number of such macro cells will exist. They should be put in the form of an array and placed

near the edges. There should be adequate spacing between the macro cells in an array to allow power and ground lines as well as signal routes. After the placement of macro cells, an initial congestion analysis can be done to ascertain that the macro placement does not lead to routing congestion. If routing congestion exists, macro placement may be changed, the gap between the cells of the array may be increased, and if nothing works out, the area of the core has to be increased.

- **Power planning**: Once congestion has been brought under control, it is time for power planning. Rings of VDD and VSS should be created around the edges, and power straps should be drawn vertically. Horizontal straps should be avoided, as they may lead to conflicts with the finer horizontal VDD and VSS tracks which ultimately interface with the standard cells. The macros should be surrounded by power rings. These will serve two functions, viz.

  (i) Allow bypass of power lines coming from the core rectangular rings.

  (ii) Form sources of supply to the power-hungry macros.

  Then, the VDD and VSS connections of the pads and macros should be completed. Only then should the power tracks for standard cells be created. This flow will avoid power tracks in areas over macro cells due to vias created for connections to macro cells. This is acceptable since standard cells do not have rigid placement attributes and may be moved slightly here and there. On the otherhand, vias created for standard cell tracks may prevent the creation of vias at the power rings to connect to macro cells, which is not tolerable. While routing the finer tracks of power, it should be ensured that there are routing and placement blockages around the macro cells. Placement of standard cells very near to memory cells may lead to congestion. Since standard cells are not present in the macro-region, the power tracks should be avoided in this region to allow the space to be used for routing lines. After power network design, standard cells are properly placed using placement commands, and timing checks are done to ensure sufficient slack.

- **Island formation after Clock tree synthesis**: Clock tree synthesis is carried out for both clocks. After the clock route, it is imperative to make sure that there are no island formations where standard cells make a cluster. This may happen if the timing constraints are slightly tight. It is always prudent to break down this island formation by doing some physical optimization (e.g., Pysnopt in Synopsys ICC); otherwise, the cluster formations will lead to routing DRCs. Now, the chip is ready for chip routing.

- **Routing**: While routing, the signal integrity option should be switched on as well as antenna rules, and I/O gate width should be defined. These are foundry specific and obtained from the foundry.

5.4.5 Verification, Parasitic extraction, Static timing, and Post-Layout simulation

These steps are carried out at several stages in the design process. A brief description is attempted here.

- **Verification and Post-Layout Simulation**: The routed design is to be verified for its correctness with the help of tools such as Synopsys Formality or a similar tool. For this, the Verilog netlist is extracted and then compared in Formality with the reference Verilog design. Further, the design is exported to GDS format and opened in an analog layout tool such as Cadence Virtuoso. After some minor modifications at this stage, the design is now verified for DRC errors using a DRC tool. If DRC errors are found, then necessary corrections need to be done in the Layout. Then, the LVS tool is run. Necessary cleanups are done at the layout level. Now the design is ready for parasitic extraction (PEX). The results of the PEX tool are fed to the Static Timing analysis engine such as Primetime. Necessary iterations are done between PT, PEX, and Layout to achieve timing closure.

- **Guidelines for DRC**: In the macro-region, there are no standard cells or their power tracks. Hence standard cell fillers are not necessary for this area. Their presence may lead to unwanted DRCs with the memory cells with respect to N and P wells. Because

of the presence of memories and pads as macros, several vias may not be visible to the place and route tools. Hence, it may place unnecessary vias there. On the other hand, the layout tool can see these vias, and hence during DRC, it will flag errors on the double vias. They need to be manually deleted in the layout tool.

- **Post layout Simulation and GDS extraction**: The clean timing design is now extracted in the netlist file from the Layout tool, ICC. The netlist file is simulated in a simulation engine such as Synopsys VCS to verify its functionality. If this is fine, the design is ready for its final GDS extraction after some housekeeping work in the layout tool. The housekeeping work may involve giving a number to the Chip for identification, putting a seal ring to isolate the design from other designs, etc. The extracted GDS is now sent to the foundry for fabrication.

5.4.6 Chip finishing

This is the last step. Standard cell filler is inserted on the whole chip except in placement blockages and around macros. For this step, the designer should be aware of the filler cells to be used for the purpose from the standard cell library documentation. After all the DRC steps, at the fag end of the design, metal fillers are inserted to improve metal coverage as per the foundry rules for the process, which are provided in the form of a rule set for dummy insertion. If sufficient coverage is not seen during verification, the corresponding metal coverage has to be improved manually by increasing the width of lines wherever space is available.

5.4.7 Chip implementation results

Synthesis is carried out using Synopsys DC tool to generate the gate level netlist. The next step is floor-planning which needed the synthesized netlist, the foundry technology files (.tf) and the design constraints file (.sdc). At this stage, memory placement, power pad placement, I/O pad placement, power distribution rings, and straps design are done in Synopsys ICC. After this initial placement, clock tree synthesis, and routing is done. The next step is the extraction of parasitics using Mentor Calibre tools. The parasitic extracted file (.pex) is fed

to Synopsys Prime Time (PT), a static timing analysis tool, and the timing results are thus revised. The back annotated timing results from Synopsys PT are fed back to ICC, and routing is done again. The process is done several times until the timing is finally met. After this, the layout (.gds) is extracted and fed to Mentor calibre for Design Rule Checks (DRC) and Antenna analysis. The necessary DRC, Antenna corrections, and chip finishing are done in Cadence Virtuoso. The final layout (.gds) is extracted from Cadence Virtuoso and submitted to the foundry for fabrication. After every major stage, Synopsys Formality is used for formal verification. During the final designing phases, Design for Testing (DFT) is included using the Synopsys DC tool. The overall die-size is 6 mm^2 compared to the area of 4.3 mm^2 as suggested by Synopsys DC. This is due to a larger area for routing. Another reason for the increased chip size here is the non-availability of properly sized memory cuts. As a result, several small-sized memories are used, which led to an increase in fabricated area. Due to limitations of packaging and I/O interfaces, the processor is fabricated for 100 MHz for testing. The layout is shown in Fig. 5.3. The design is tested by ATPG testers for faults and packaged in a 120 pin Quad-fin package. Further testing required board design with a test-data generating FPGA and also experience of RF engineering due to the high speed data movements. The testing has been delayed due to lack of all the resources.

5.5 Conclusion

The chapter brings out considerations during the FFT ASIC design process. It is involved work right from RTL to final Packaging using a large number of software tools. If it is desired to make a design into ASIC, proper knowledge of the physical design flow is required of the RTL designer also. Physical design is a specialized work and needs help from backend designers well-versed in the tools, but RTL designers have to lead the design process and upgrade themselves, working hand in hand with PnR experts to realize the working chip on silicon.

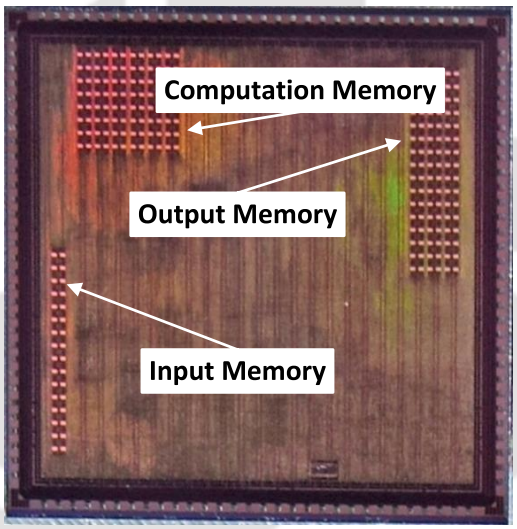


Figure 5.3: Micrograph of FFT Processor

6

Conclusion

Contents

6.1 Summary of the Present Work

The objective of the work presented in this thesis is to develop a high throughput FFT processor (28 Gbps) for use in OFDM transmission in GHz standards. The crux of the work is on the principle that to achieve high performance and high throughput, the ASICs have to be more specialized. The existing OFDM FFTs are designed as general purpose FFT processors and for that matter could work on data from any domain. In 5G and other wireless chips, area and power savings is a must alongwith performance. Most chip designers are willing to give more emphasis on to design time if there are significant savings on these aspects. Hence, going further, it is envisaged that high volume applications in 5G and wireless will demand such specialized FFT processors. The proposed FFT processor uses simplifications which are applicable to OFDM transmission to achieve its performance. In **Chapter 1**, a detailed literature survey of the state-of-the-art is presented. Also,a complete design methodology is analyzed to arrive at a architecture for high performance designs. Such designs are usually based on high radix. Several options for choice of radix, memory architecture, pipelining, multipliers etc. are studied and contrasted with each other. Detailed methodologies for derivation of flow diagram from design equations are presented to serve as a complete flow for a high-radix design. Further, the process of deriving a high level model in Matlab and then using it for debugging and verification during RTL design phase is also discussed.

Chapter 2 addresses a fundamental design problem in all FFT processors. It relates to the way data is stored in memory banks to allow single cycle access for all data which needs to be processed at once. This problem, popularly known in literature as Conflict free addressing has been classically solved using XOR or modulo addition based logic. Later a new class of simplified CFA technique based on permutation is proposed but its applications are not studied comprehensively. In this chapter, the permutation technique is derived using simpler mathematical tools to arrive at a new CFA technique which is referred to as Progressive Shifting. It is shown that for the proposed design progressive shift technique results in 30% area savings compared to existing ones.

In **Chapter 3**, another well-known problem in FFT processors is solved using a novel approach. The problem is storage of present data when previous data has been stored in memory for processing and is known as continuous flow technique. The present methods use one or two extra memories to store the incoming and outgoing data. The disadvantage is the under utilization of these memories as these memories could be used to feed data to processing elements. They are better used to increase throughput by integrating them as part of the processor. Detailed mathematics using group theory is developed to understand the interaction among various memories and address generators. Two specific techniques are developed namely, Data Shifting and Address Shifting, both of which are suitable under different circumstances.

Chapter 4 describes the main design. The chosen algorithm is based on radix-16 taken from [2]. The use of simplifications specific to OFDM, the CFA technique described in chapter 2 and the continuous flow technique of chapter 3 result in a very efficient FFT design. Increase in performance is achieved without much increase in area. This design integrates the memories as part of the architecture, unlike other designs which use switching of memories to achieve continuous flow. The added memory helps to feed extra PEs. The only increased cost is that of PEs now, which is offset by optimizing the other PEs. Additionally, the conflict free addressing circuit is very simple now reducing the switching complexities greatly. All design innovations done together result in a chip which is 50% more area efficient than existing ones.

In **Chapter 5**, the physical design aspects of the ASIC are discussed. The FFT processor is sufficiently large causing challenges in physical design. The fabrication is done at SCL Chandigarh, India in 180 nm process. The relationship of physical design with RTL is brought out here in sufficient detail. Also, the tool flow and guidelines followed during backend PnR are described here.

6.2 Suggestions for future research

In this thesis, we have shown the effectiveness of specializing FFT processors for the domain. It is felt that this is just the beginning of a new class of FFT architectures. Further research should be taken up in the following areas to improve on this design:

(i) New requirements are emerging in various emerging and high volume domains. In such areas, power savings in silicon area is a priority. Example of these areas are automotive radar, space applications, Internet of Things and so on. Specialized FFT processors can be designed in these areas.

(ii) The use of group theory in CF techniques is a new idea. It merits further study to extend to more complex memory hierarchies.

(iii) The design is for wireless applications with fixed size. Converting to flexible size processing will require further innovation. This is especially relevant in 5G where the size of FFT is variable.

(iv) Integrating the FFT with its pre and post processing DSP units may result in overall savings and is an area worth investigating.

List of Publications

International peer reviewed journals

- S. Agarwal, S. R. Ahamed, A. K. Gogoi and G. Trivedi, "A Low-Complexity Shifting-Based Conflict-Free Memory-Addressing Architecture for Higher-Radix FFT," in IEEE Access, vol. 9, pp. 140349-140357, 2021, doi: 10.1109/ACCESS.2021.3119598.
- S. Agarwal, S. R. Ahamed, A. K. Gogoi and G. Trivedi, "A 28 Gbps Radix-16, 512 point FFT Processor based Continuous Streaming OFDM for WiGig", Springer Nature, Circuits, Systems, and Signal Processing, vol. 41, pp. 2871–2897 (2022).

Conference

- S. Agarwal, A. Gogoi, D. Mallik, H. S. Jatana, "Physical Design Considerations on the architecture and layout of a FFT processor", INAC-03, SCL, Chandigarh, Oct, 2017

Submitted

- S. Agarwal, S. R. Ahamed, A. K. Gogoi and G. Trivedi, "Continuous Flow FFT: Data and Address Shifting Schemes based on Group Theory", IEEE Trans. on Circuits and Systems-I, Regular papers

Bibliography

[1] B. Jo and M. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 5, pp. 911–919, May 2005.

[2] S.-J. Huang and S.-G. Chen, "A high-throughput radix-16 FFT processor with parallel and normal Input/Output ordering for IEEE 802.15.3c systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 8, pp. 1752–1765, Aug 2012.

[3] C. R. C. M. da Silva, A. Lomayev, C. Chen, and C. Cordeiro, "Analysis and simulation of the IEEE 802.11ay single-carrier phy," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[4] R. W. Chang, "Synthesis of band-limited orthogonal signals for multichannel data transmission," *Bell System Technical Journal*, vol. 45, no. 10, pp. 1775–1796, 1966. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1966.tb02435.x

[5] B. Saltzberg, "Performance of an efficient parallel data transmission system," *IEEE Transactions on Communication Technology*, vol. 15, no. 6, pp. 805–811, 1967.

[6] J. Salz and S. B. Weinstein, "Fourier transform communication system." New York, NY, USA: Association for Computing Machinery, 1969. [Online]. Available: https://doi.org/10.1145/800165.805240

[7] S. Weinstein and P. Ebert, "Data transmission by frequency-division multiplexing using the discrete fourier transform," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 628–634, 1971.

[8] L. Yu Wei, L. Hsuan Yu, and L. Chen Yi, "A 1-GS/s FFT/IFFT processor for UWB applications," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 8, pp. 1726–1735, August 2005.

[9] H. Lee and M. Shin, "A high-speed low-complexity two-parallel radix-2^4 FFT/IFFT processor for UWB applications," in *IEEE Asian Solid-State Circuits Conference*, Nov 2007, pp. 284–287.

[10] S. Tang, J. Tsai, and T. Chang, "A 2.4-GS/s FFT processor for OFDM-based WPAN applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 6, pp. 451–455, 2010.

[11] Y. Chen, Y. Lin, Y. Tsao, and C. Lee, "A 2.4-Gsample/s DVFS FFT processor for MIMO OFDM communication systems," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 5, pp. 1260–1273, 2008.

[12] T. Cho, H. Lee, J. Park, and C. Park, "A high-speed low-complexity modified radix-2^5 FFT processor for gigabit WPAN applications," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2011, pp. 1259–1262.

[13] H. L. Davila, C. Liu, W. Liu, S. Huang, S. Jou, and S. Chen, "A 802.15.3c/802.11ad compliant 24 Gb/s FFT processor for 60 GHz communication systems," in 28^{th} *IEEE International System-on-Chip Conference (SOCC)*, Sept 2015, pp. 44–48.

[14] E. J. Kim and M. H. Sunwoo, "High speed eight-parallel mixed-radix FFT processor for OFDM systems," in *IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 1684–1687.

[15] S. Liu and D. Liu, "A high-flexible low-latency memory-based FFT processor for 4G, WLAN, and future 5G," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 511–523, 2018.

[16] Y. Tian, Y. Hei, Z. Liu, Q. Shen, Z. Di, and T. Chen, "A modified signal flow graph and corresponding conflict-free strategy for memory-based FFT processor design," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 1, pp. 106–110, 2019.

[17] N. Le Ba and T. T.-H. Kim, "An area efficient 1024-point low power radix-2^2 FFT processor with feed-forward multiple delay commutators," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 10, pp. 3291–3299, 2018.

[18] S.-C. Hsu, S.-J. Huang, S.-G. Chen, S.-C. Lin, and M. Garrido, "A 128-point multi-path SC FFT architecture," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2020, pp. 1–5.

[19] H. E. Yantir, W. Guo, A. M. Eltawil, F. J. Kurdahi, and K. N. Salama, "An ultra-area-efficient 1024-point in-memory FFT processor," *Micromachines*, vol. 10, no. 8, p. 509, 2019.

[20] T. T. B. Nguyen and H. Lee, "High-throughput low-complexity mixed-radix FFT processor using a dual-path shared complex constant multiplier," *JSTS: Journal of Semiconductor Technology and Science*, vol. 17, no. 1, pp. 101–109, 2017.

[21] D. Cohen, "Simplified control of FFT hardware," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 24, no. 6, pp. 577–579, Dec 1976.

[22] Y. Ma and L. Wanhammar, "A hardware efficient control of memory addressing for high-performance FFT processors," *IEEE Transactions on Signal Processing*, vol. 48, no. 3, pp. 917–921, 2000.

[23] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 39, no. 5, pp. 312–316, May 1992.

[24] P.-Y. Tsai and C.-Y. Lin, "A generalized conflict-free memory addressing scheme for continuous-flow parallel-processing FFT processors with rescheduling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2290–2302, Dec 2011.

[25] K. Xia, B. Wu, T. Xiong, and T. Ye, "A memory-based FFT processor design with generalized efficient conflict-free address schemes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1919–1929, June 2017.

[26] S. Richardson, D. Markovic, A. Danowitz, J. Brunhaver, and M. Horowitz, "Building conflict-free FFT schedules," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 4, pp. 1146–1155, April 2015.

[27] C. Hsiao, Y. Chen, and C. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 1, pp. 26–30, 2010.

[28] D. Reisis and N. Vlassopoulos, "Address generation techniques for conflict free parallel memory accessing in FFT architectures," in 13^{th} *IEEE International Conference on Electronics, Circuits and Systems, ICECS*, 2006, pp. 1188–1191.

[29] S.-G. Chen, S.-J. Huang, M. Garrido, and S.-J. Jou, "Continuous-flow parallel bit-reversal circuit for MDF and MDC FFT architectures," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 10, pp. 2869–2877, Oct 2014.

[30] Y. Tian, Y. Hei, Z. Liu, and Z. D. et al, "A memory-based FFT processor using modified signal flow graph with novel conflict-free address schemes," *J-STAGE, IEICE Electronics Express*, vol. 14, no. 15, p. 20178005, july 2017.

[31] C. Wang, Y. Yan, and X. Fu, "A high throughput low complexity radix - 2^4 - 2^2 - 2^3 FFT/IFFT processor with parallel and normal input/output order for IEEE 802.11ad systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 11, pp. 2728–2732, 2015.

[32] J. H. Baek, B. S. Son, B. G. Jo, M. H. Sunwoo, and S. K. Oh, "A continuous flow mixed-radix FFT architecture with an in-place algorithm," in *Proceedings of the International Symposium on Circuits and Systems, ISCAS '03.*, vol. 2, 2003, pp. 133–136.

[33] K. Yang, S. Tsai, and G. Chuang, "MDC FFT/IFFT processor with variable length for MIMO-OFDM systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 4, pp. 720–731, April 2013.

[34] H. Luo, Y. Liu, and M. Shieh, "Efficient memory addressing algorithms for FFT processor design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2162–2172, Oct 2015.

[35] Q. Xing, Z. Ma, and Y. Xu, "A novel conflict-free parallel memory access scheme for FFT processors," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 11, pp. 1347–1351, 2017.

[36] X. Mao, Z. Ma, F. Yu, and Q. Xing, "A continuous-flow memory-based architecture for real-valued FFT," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 11, pp. 1352–1356, 2017.

[37] Z. Ma, X. Yin, and F. Yu, "A novel memory-based FFT architecture for real-valued signals based on a radix-2 decimation-in-frequency algorithm," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 9, pp. 876–880, 2015.

[38] J. Hazarika, M. T. Khan, and S. R. Ahamed, "Low-complexity continuous-flow memory-based FFT architectures for real-valued signals," in 32^{nd} *International Conference on VLSI Design (VLSID)*, 2019, pp. 46–51.

[39] M. Yuan, Z. Ma, F. Yu, and Q. Xing, "A novel address scheme for continuous-flow parallel memory-based real-valued FFT processor," *Electronics*, vol. 8, p. 1042, 09 2019.

[40] S. Long, M. Hong, and M. T. Shiue, "A low-complexity generalized memory addressing scheme for continuous-flow fast fourier transform," in 3^{rd} *International Conference on Computer and Communication Systems (ICCCS)*, April 2018, pp. 492–496.

[41] Gyanendra, B. Raman, and B. K. Kaushik, "Novel bit-reordering circuit for continuous-flow parallel FFT architectures," *IEEE Transactions on Circuits and Systems II: Express Briefs*, April 2020.

[42] J. Wang, S. Li, and X. Li, "Scheduling of data access for the radix-2^k FFT processor using single-port memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 7, pp. 1676–1689, 2020.

[43] C. Ma, H. Chen, Y. Liu, and Y. Wang, "An efficient design for general mixed radix FFT processors," *IEICE Electronics Express*, vol. 13, no. 6, pp. 1–7, March 2016.

[44] X. Xiao, E. Oruklu, and J. Saniie, "An efficient FFT engine with reduced addressing logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 11, pp. 1149–1153, 2008.

[45] G. G. Kumar, S. K. Sahoo, and P. K. Meher, "50 years of FFT algorithms and applications," *Circuits, Systems, and Signal Processing*, vol. 38, no. 12, pp. 5665–5698, 2019.

[46] E. Konguvel and M. Kannan, "A survey on FFT/IFFT processors for next generation telecommunication systems," *Journal of Circuits, Systems and Computers*, vol. 27, no. 03, p. 1830001, 2018. [Online]. Available: https://doi.org/10.1142/S0218126618300015

[47] N. H. Nguyen, S. A. Khan, C.-H. Kim, and J.-M. Kim, "A high-performance, resource-efficient, reconfigurable parallel-pipelined FFT processor for FPGA platforms," *Microprocessors and Microsystems*, vol. 60, pp. 96–106, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933117302788

[48] L. Davila, Henry, Y. Liu, Chun, C. Liu, Wei, J. Huang, Shen, J. Jou, Shyh, and G. Chen, Sau, "A 802.15.3c /802.11 ad compliant 24 Gb/s FFT processor for 60 GHz communication systems," *2015* 28^{th} *IEEE International System-on-Chip Conference (SOCC)*, 2015.

[49] H. Xiao, X. Yin, N. Wu, X. Chen, J. Li, and X. Chen, "VLSI design of low-cost and high-precision fixed-point reconfigurable FFT processors," *IET Computers & Digital Techniques*, vol. 12, no. 3, pp. 105–110, 2018.

[50] Y. Ma and H. Liang, "Implementation of a pipeline large-FFT processor based on the FPGA," in *International Conference in Communications, Signal Processing, and Systems*. Springer, 2017, pp. 638–644.

[51] J. K. Jang and M. H. Sunwoo, "Efficient scheduling schemes for low-area mixed-radix MDC FFT processor," *Journal of The Institute of Electronics and Information Engineers*, vol. 54, no. 7, pp. 29–35, 2017.

[52] Y. Xie, C. Yang, C.-A. Mao, H. Chen, and Y.-Z. Xie, "A novel low-overhead fault tolerant parallel-pipelined FFT design," in *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE, 2017, pp. 1–4.

[53] W.-L. Tsai, S.-G. Chen, and S.-J. Huang, "A low-complexity mixed-radix FFT rotator architecture," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2018, pp. 183–186.

[54] W. Liu, Q. Liao, F. Qiao, W. Xia, C. Wang, and F. Lombardi, "Approximate designs for fast fourier transform (FFT) with application to speech recognition," *Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 12, pp. 4727–4739, 2019.

[55] Y. D. Borole and C. Dethe, "Mixed radix cordic FFT algorithm for OFDM wpan applications," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. IEEE, 2017, pp. 2975–2979.

[56] G.-M. Tang, P.-Y. Qu, X.-Y. Zheng, J.-H. Yang, X.-C. Ye, D.-R. Fan, and N.-H. Sun, "Bit-slice butterfly processing units for 64-point RSFQ FFT processors," *IEEE Transactions on Applied Superconductivity*, vol. 30, no. 1, pp. 1–6, Jan 2020.

[57] M. Mahdavi, O. Edfors, V. Öwall, and L. Liu, "A low latency FFT/iFFT architecture for massive MIMO systems utilizing OFDM guard bands," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 7, pp. 2763–2774, 2019.

[58] J. Žádník and J. Takala, "Low-power programmable processor for fast fourier transform based on transport triggered architecture," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 1423–1427.

[59] M. Wang and Z. Li, "An area-and energy-efficient hybrid architecture for floating-point FFT computations," *Microprocessors and Microsystems*, vol. 65, pp. 14–22, 2019.

[60] S. Ryoo, "Analysis of bit truncation process in FFT algorithm for OFDM systems," *International Information Institute (Tokyo). Information*, vol. 20, no. 7A, pp. 4991–4998, 2017.

[61] W.-L. Tsai, S.-G. Chen, and S.-J. Huang, "Reconfigurable radix-$2^k \times 3$ feedforward FFT architectures," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[62] V. Kitsakis, K. Nakos, D. Reisis, and N. Vlassopoulos, "Parallel memory accessing for FFT architectures," *Journal of Signal Processing Systems*, vol. 90, no. 11, pp. 1593–1607, 2018.

[63] X. Chen, Y. Lei, Z. Lu, and S. Chen, "A variable-size FFT hardware accelerator based on matrix transposition," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1953–1966, 2018.

[64] Y.-J. Kim, I.-G. Jang, K.-J. Cho, and J.-G. Chung, "Low-latency and memory-efficient SDF IFFT processor design for 3 GPP LTE," *IEICE Electronics Express*, pp. 14–20 170 395, 2017.

[65] A. Ayala, S. Tomov, M. Stoyanov, and J. Dongarra, "Scalability issues in FFT computation," in *International Conference on Parallel Computing Technologies*. Springer, 2021, pp. 279–287.

[66] P. T. Dinh, L. Lanante, M. D. Nguyen, M. Kurosaki, and H. Ochi, "An area-efficient multimode FFT circuit for IEEE 802.11 ax WLAN devices," in *2017 19th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2017, pp. 735–739.

[67] X. Han, J. Chen, B. Qin, and S. Rahardja, "A novel area-power efficient design for approximated small-point FFT architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4816–4827, 2020.

[68] J. Yu and K.-J. Cho, "A low-area and low-power 512-point pipelined FFT design using radix-$2^4 - 2^3$ for OFDM applications," *The Journal of Korea Institute of Information, Electronics, and Communication Technology*, vol. 11, no. 5, pp. 475–480, 2018.

[69] S.-N. Tang and Y.-H. Chen, "Area-efficient FFT kernel with improved use of GI for multistandard MIMO-OFDM applications," *Applied Sciences*, vol. 9, no. 14, p. 2877, 2019.

[70] H.-J. Lin and C.-A. Shen, "The architectural optimizations of a low-complexity and low-latency FFT processor for MIMO-OFDM communication systems," *Journal of Signal Processing Systems*, vol. 93, no. 1, pp. 67–78, 2021.

[71] Y. Lee, V. Koo, and Y. Chan, "Design and development of FPGA-based FFT co-processor for synthetic aperture radar (sar)," in *2017 Progress in Electromagnetics Research Symposium-Fall (PIERS-FALL)*. IEEE, 2017, pp. 1760–1766.

[72] Y. Li, H. Chen, and Y. Xie, "An FPGA-based four-channel 128k-point FFT processor suitable for spaceborne sar," *Electronics*, vol. 10, no. 7, p. 816, 2021.

[73] Y. Jung, J. Cho, and Y. Jung, "Low complexity pipelined FFT processor for radar applications," in *Advances in Computer Science and Ubiquitous Computing*. Springer, 2021, pp. 447–453.

[74] C.-K. Chan, H.-K. Lin, and C.-W. Liu, "High-throughput 64k-point FFT processor for THz imaging radar system," in *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. IEEE, 2019, pp. 1–4.

[75] Y. Park, Y. Jung, and Y. Jung, "Design of low-complexity FFT processor for multi-mode radar signal processing," *The Journal of Advanced Navigation Technology*, vol. 24, no. 2, pp. 85–91, 2020.

[76] H. J, J. Y, L. S, and J. Y, "FPGA implementation of an efficient FFT processor for FMCW radar signal processing," *MDPI, Sensors (Basel)*, vol. 19, p. 6443, Sept 2021.

[77] Y. Jung, J. Cho, S. Lee, and Y. Jung, "Area-efficient pipelined FFT processor for zero-padded signals," *Electronics*, vol. 8, no. 12, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/12/1397

[78] M. Garrido and P. Paz, "Optimum MDC FFT hardware architectures in terms of delays and multiplexers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 3, pp. 1003–1007, 2020.

[79] M. Garrido, "A survey on pipelined FFT hardware architectures https://doi.org/10.1007/s11265-021-01655-1," *J Sign Process Syst.*, 2021.

[80] N. Nadar, M. Mehta, K. Bhat, J. Mendes, and R. Chaudhari, "Hardware implementation of pipelined FFT using polyphase decomposition," in *2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*. IEEE, 2021, pp. 339–343.

[81] M. Liu, P. Zhao, T. Wu, K. K. Parhi, X. Zeng, and Y. Chen, "A low-power twiddle factor addressing architecture for split-radix FFT processor," *Microelectronics Journal*, vol. 117, p. 105276, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026269221002627

[82] D. Jeong, H. Son, Y. Jung, Y. Jung *et al.*, "Design and implementation of systolic array FFT processor based on shared memory," *Journal of IKEEE*, vol. 24, no. 3, pp. 797–802, 2020.

[83] F. Ke, O. Chen, Y. Wang, and N. Yoshikawa, "Demonstration of a 47.8 GHz high-speed FFT processor using single-flux-quantum technology," *IEEE Transactions on Applied Superconductivity*, vol. 31, no. 5, pp. 1–5, Aug 2021.

[84] A. Rauf, M. A. Pasha, and S. Masud, "Towards design and automation of a scalable split-radix FFT processor for high throughput applications," *Microprocessors and Microsystems*, vol. 65, pp. 148–157, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933117304751

[85] C. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 25, no. 3, pp. 239–242, June 1977.

[86] Jatana and Desai, "SCL 180nm CMOS foundry: High reliability ASIC design for aerospace applications," in 19^{th} *International Symposium on VLSI Design and Test*, 2015, pp. 1–2.

[87] "Jaguar II variable-point (8-1024) FFT/IFFT specification," 1998.

[88] L. R. Rabiner and B. Golds, *Theory and Application of Digital Signal Processing.* Prentice-Hall, 1975.

[89] S. He and M. Torkelson, "A New Approach to Pipeline FFT Processor," *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, pp. 766–770., Apr 1996.

[90] S. Mehmood, D. Dasalukunte, and V. Owall, "Hardware architecture of IOTA pulse shaping filters for multicarrier systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 3, pp. 733–742, 2013.

[91] K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation.* Wiley Interscience, 2007.

[92] J. K. Y. Jung, H. Yoon, "New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 1, pp. 14–20, February 2003.