# DESIGN AND IMPLEMENTATION OF HARDWARE-EFFICIENT ARCHITECTURES FOR FFT ALGORITHMS




*JINTI HAZARIKA*

# DESIGN AND IMPLEMENTATION OF HARDWARE-EFFICIENT ARCHITECTURES FOR FFT ALGORITHMS

*A*

*Thesis submitted*

*for the award of the degree of*

**DOCTOR OF PHILOSOPHY**

By

**JINTI HAZARIKA**


DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, ASSAM, INDIA

DECEMBER 2023

# Certificate

This is to certify that the thesis entitled "**DESIGN AND IMPLEMENTATION OF HARDWARE - EFFICIENT ARCHITECTURES FOR FFT ALGORITHMS**", submitted by **Jinti Hazarika** (166102017), 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 her under my supervision and guidance. The thesis has fulfilled all requirements as per the regulations of the institute and in my opinion has reached the standard needed for submission. The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dated:

Guwahati.

Prof. Shaik Rafi Ahamed
Professor
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati - 781 039, Assam, India.

Prof. Harshal B. Nemade
Professor
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Guwahati - 781 039, Assam, India.

To

My supervisors

**Prof. Shaik Rafi Ahamed and Prof. Harshal B. Nemade**

for their guidance and inspiration

&

My dear parents and my parents-in-law

&

My husband

**Janardhan Dehingia**

for their love and support

# Acknowledgements

I would like to express my sincere gratitude to several individuals who have supported me throughout my research journey. Firstly, I am deeply indebted to God Almighty for blessing me with the opportunity and strength to carry out my research. My parents, Prasanta Hazarika and Arunima Hazarika, have been a constant source of support and inspiration. Their sacrifices and unwavering support have helped me to excel in various stages of my life. I am deeply grateful to them and words cannot express my appreciation for their love and guidance. A special thanks to my husband Janardhan Dehingia for his kind support during the hardships in my research life. His love and encouragement have been a great source of strength for me. I would like to thank him for his kind support and for being there for me through thick and thin.

I feel great privilege in expressing my deepest and most sincere gratitude to my supervisors, Prof. Shaik Rafi Ahamed and Prof. Harshal B. Nemade, for their excellent guidance throughout my study which sparked various ideas during the course of my research. Their kindness, dedication, hard work and attention to detail have been a great inspiration to me. My heartfelt thanks to both my supervisors for the unlimited support and patience shown to me. I would particularly like to thank them for all their help in patiently and carefully reviewing all my manuscripts. I am also very thankful to my doctoral committee members Prof. Manas Kamal Bhuyan, Dr. Rishikesh Dilip Kulkarni, and Dr. Mahima Arrawatia for sparing their precious time to evaluate the progress of my work.

I would like to thank the Head of the Department and the other faculty members for their kind help in carrying out this work. I am also grateful to all the members of the research and technical staff of the department without whose help I could not have completed this thesis. I would also like to extend my gratitude to the Department of Electronics and Electrical Engineering and its staff members.

Thanks go out to all my friends at the VLSI Laboratory-I, II. They have always been around to provide useful suggestions, companionship and created a peaceful research environment. They all contributed directly or indirectly to this thesis through academic help and proofreading. The support of many fellow students has helped me get through the pressures during the course of my research life. My work at IIT Guwahati definitely would not be possible without their love and care which helped me to enjoy my life at IIT Guwahati. I have received some good suggestions and help from my senior Dr. Mohd. Tasleem Khan who later turned out to be the co-author for a few publications. Special thanks

to Dr. Satyajit Bora, Himakshi Mishra, Himangshu Jyoti Gogoi, Thockchom Birjit Singha, Paromita Bhattacharjee, Dr. Arunima Dutta, Atanu Purkayastha, Shalini Kumari, Rituparna Choudhury, Nadeem Atif, Riyaj Uddin Khan, Raktim Acharjee, Aikendrajit Ningthoujam, and Rahimul Islam Majumdar.

At the end I would like to express my thanks to all the friends, relatives and many others who have helped me in several ways for the completion of my research work. Finally, I believe this research experience will greatly benefit my career in the future.

*Jinti Hazarika*

# Abstract

Fast Fourier transform (FFT) plays a vital role in a wide range of applications in the fields of wireless communications, audio, image and signal processing. Due to its ubiquitous applications, it is essential to increase its hardware efficiency while designing low-area and low-power FFT processors. Over the years, researchers have made continuous efforts to improve the hardware efficiency of FFT processors. Usually, it is related to the type of FFT algorithm employed for a given application that decides the number of multipliers, adders, and registers. These computational resources are directly related to the power consumption and area of FFT processors. They pose as limiting factors for potential FFT applications due to throughput and area/power trade-off. This trade-off must be considered in the design of FFT processors using VLSI methodologies. For low-power applications, the baseband FFT processors in battery operated portable devices are expected to operate for longer periods of time. Depending on throughput requirement and computation complexity, FFT architectures are mainly categorized into serial, parallel, or a combination of both. Serial FFTs require the least computational resources, thereby reducing chip area and power consumption at the expense of throughput. The throughput performance can be improved by invoking pipelining at the cost of pipelined registers, increasing the chip area and power consumption. In contrast, parallel FFTs inherently possess high throughput at the expense of computational resources. Therefore, the chip area and power consumption are always higher than serial FFT. Throughput in parallel FFTs can be further increased using pipelining. A combination of serial and parallel with or without pipelining can be used to trade between the computational resources and throughput.

This thesis considers three types of FFT architectures which are basically focused on the hardware-efficient realization of FFT algorithms as per the level of parallelism: serial, parallel and fully-parallel. Each of these employ a different suggested technique to bring down the computational resources without sacrificing the performance with respect to state-of-the-art designs. Lowering the complexity

also reduces the overall area and power consumption as the number of hardware elements would be reduced. However, focusing primarily on reducing the complexity of a design can degrade the system's performance. Complexity reduction without compromising on the throughput of the FFT processor is a challenging area that needs to be addressed. First, serial real-valued FFT (RFFT) architectures are presented with the objective of reducing the hardware complexity. Although serial FFT systems have less hardware complexity than parallel FFT architectures, the existence of multipliers might still restrict the overall performance. With this goal, a new data flow graph (DFG) is proposed that reduces the number of multiplication stages undergoing multiplications with different twiddle factor (TF). Further, we propose a shift-accumulation based multiplier designed by exploiting partial-product symmetries at the expense of an offset. When this multiplier is extended for higher wordlengths, resource sharing has been observed in the offset computation and the shift-add logic. Another challenge in FFTs is the reordering of the output. This is addressed by a new data mapping scheme that eliminates the post-processing stage required for normal-order output using a merged unit for butterfly computation and data-reordering. Next, two low-complexity memory-based FFT architectures for real-valued signals are presented. The first design processes four inputs in parallel and contains a half-complex multiplier whereas the second design processes two inputs in parallel and contains a quarter-complex multiplier. Two new memory accessing schemes corresponding to each design are proposed. Analysis of computational complexities shows that the first architecture provides lower complexity in terms of registers and multiplexers while second architecture provides lower complexity in terms of the multiplier over prior works. The most important feature is that for the same number of clock cycles, the proposed scheme requires less computation time to compute an FFT due to shorter clock period. Lastly, a fully parallel-pipelined split-radix (SR) based low-cost FFT architecture is presented. The pipelined implementation of fully parallel SRFFT requires considerably higher number of registers compared to other FFT algorithms. Therefore, it leads to more power consumption and high latency despite having the least number of multipliers among all FFTs. This is addressed with a modified DFG for SRFFT which results in the absence of non-trivial twiddle factors in the odd stages. By doing so, a significant reduction in the number of pipelined registers with lower latency is achieved. Further, an efficient shift-add multiplier is designed with architectural analysis and substructure sharing. A comparison of the proposed design with existing fully-parallel FFTs shows that the proposed method outperforms other reported works.

Finally, it is to be noted that the hardware-efficient FFT architectures presented in this thesis are core elements in many signal processing applications, and can lead to a number of improved architectures that are suggested for future research work.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

| | |
|---|---|
| ADP | Area-Delay Product |
| ADPP | Area-Delay-Power Product |
| ASIC | Application Specific Integrated Circuit |
| BF | Butterfly |
| BFDR | Butterfly-cum-Data-Reordering |
| BRAM | Block Random Access Memory |
| BS | Barrel-Shifter |
| CC | Clock Cycles |
| CFFT | Complex-valued FFT |
| Clk | Clock |
| CM | Constant Multiplier |
| CMOS | Complementary Metal Oxide Semiconductor |
| CP | Clock Period |
| CPD | Critical Path Delay |
| CSD | Canonic Sign Digit |
| CW | Computational Workload |
| DAT | Data Arrival Time |
| DFG | Data Flow Graph |
| DIF | Decimation-in-Frequency |
| DIT | Decimation-in-Time |
| DMS | Data Mapping Scheme |
| DR | Data-Reordering |
| DSP | Digital Signal Processing |

| | |
|---|---|
| DSM | Digit Serial Multiplier |
| EPP | Even-Partial Products |
| EPS-PP | Pseudosymmetric EPP |
| FF | Flip-Flop |
| FFT | Fast Fourier Transform |
| FP | Fixed Point |
| FPGA | Field Programmable Gate Array |
| HCM | Half-Complex Multiplier |
| HDL | Hardware Description Language |
| IoT | Internet-of-Things |
| LSB | Least Significant Bit |
| LUT | Look-up Table |
| MB | Memory Bank |
| MBE | Modified Booth Encoding |
| MCP | Minimum Clock Period |
| MOT | Multi Offset Term |
| MSE | Mean Square Error |
| OFDM | Orthogonal Frequency Division Multiplexing |
| OPP | Odd-Partial Products |
| OS-PP | Odd-Symmetric Partial Products |
| PE | Processing Element |
| PnR | Place&Route |
| POT | Power-of-Two |
| PPG | Partial Product Generator |
| QCM | Quarter Complex Multiplier |
| RFFT | Real-valued FFT |
| RSC | Real-valued Serial Commutator |
| RTL | Register-Transfer Level |
| SA | Shift-Accumulate |
| SL | Sign Logic |

| | |
|---|---|
| SLUT | Sliced LUT |
| SRFFT | Split Radix FFT |
| TC | Two's Complement |
| TF | Twiddle Factor |
| TP | Throughput |
| TSMC | Taiwan Semiconductor Manufacturing Company Limited |
| VLSI | Very Large Scale Integration |

# List of Symbols

| | |
|---|---|
| $a$ | Multiplier |
| $\alpha$ | Switching activity |
| $b$ | Multiplicand |
| $b_k$ | Input-digit |
| $B$ | Number of bits to represent $b_k$ |
| $c$ | Product |
| $C$ | Capacitance |
| $C_L$ | Switching capacitance |
| $C_{n-2}$ | $(n-2)$-bit counter |
| $\epsilon_p$ | BF input quantization noise at $p$th stage |
| $\hat{\epsilon}_{p+1}$ | Round-off error |
| $\epsilon_{p+1}$ | Quantization noise at $(p+1)$th stage |
| $f$ | Operating frequency |
| $f_f$ | Faster clock |
| $f_s$ | Sample clock |
| $\gamma_{BF}^{R,P}$ | Number of registers in half-butterfly unit |
| $\gamma_{BFDR}^{R,P}$ | Number of registers in BFDR unit |
| $\gamma_{DR,I}^{R,P}$ | Number of registers in input data-reordering |
| $\gamma_{DR,IM}^{R,P}$ | Number of registers in intermediate data-reordering |
| $\gamma_{DR,O}^{R,P}$ | Number of registers in output data-reordering |
| $\gamma_{QM}^{R,P}$ | Number of registers in QM unit |
| $\gamma_{T}^{R,P}$ | Total number of registers in proposed design |
| $\gamma_{T}^{R,E}$ | Total number of registers in existing design |

| | |
|---|---|
| $\Delta\gamma_T^{R,P}$ | Percentage reduction in number of registers |
| $I_L$ | Leakage current |
| $Im$ | Imaginary component of complex number |
| $I_{SC}$ | Short-circuit current |
| $j$ | Imaginary number ($\sqrt{-1}$) |
| $k_0$ | Number of digits |
| $L$ | Buffer length |
| $n$ | Number of stages |
| $N$ | Number of points of FFT |
| $p$ | Number of pipelined registers |
| $P$ | Dynamic power consumption |
| $P_{\text{total}}$ | Total power consumption |
| $r$ | Order of recursion |
| $Re$ | Real component of complex number |
| $\sigma^2$ | Input quantization noise of serial RFFT |
| $t$ | Time instant |
| $V_{DD}$ | Supply voltage |
| $W$ | Number of bits of $a$ and $b$ |
| $W_N^{nk}$ | Twiddle factor |
| $w_s$ | Sign bit |
| $x[n]$ | Samples in time domain |
| $X[k]$ | Samples in frequency domain |
| $X_f^p$ | Fixed-point representation of the fractional part of input at $p$th stage |
| $X_i^p$ | Fixed-point representation of the integer part of input at $p$th stage |
| $\xi_{p+1}$ | Mean square error of $\epsilon_{p+1}$ |
| $\hat{\xi}_{p+1}$ | Mean square error of $\hat{\epsilon}_{p+1}$ |
| $Y_f^p$ | Fixed-point representation of the fractional part of output at $p$th stage |
| $Y_i^p$ | Fixed-point representation of the integer part of output at $p$th stage |
| $\zeta_{BF}^P$ | Computational workload on half-butterfly units of proposed design |
| $\zeta_{BF}^E$ | Computational workload on half-butterfly units of existing design |

| | |
|---|---|
| $\Delta$ | Level of input LSB of BF |
| $\Delta\zeta_{BF}$ | Reduction in computational workload on half-butterfly units |

# 1

# Introduction



Contents

## 1.1 Introduction

The technology landscape is constantly evolving, driven by advancements in various areas such as CMOS scaling, non-CMOS technologies, and in-memory computing technologies. These advancements have a significant impact on system-level performance and efficiency, particularly in domains like wireless communication and bioinformatic signal processing. CMOS (Complementary Metal-Oxide-Semiconductor) scaling involves reducing the size of transistors and components on integrated circuits. As transistors become smaller, higher integration onto a chip can be achieved, leading to increased processing power, higher energy efficiency, and higher throughput. However, challenges like power consumption and heat dissipation become more pronounced as we approach the physical limits of CMOS scaling. Non-CMOS technologies involve alternatives to traditional CMOS-based approaches. These may include emerging technologies like quantum computing, neuromorphic computing, and memristors. These technologies offer the potential for significantly higher computational power and novel computing paradigms, although they are still in the early stages of development and face challenges in terms of scalability, reliability, and integration. In-memory computing refers to processing and storing data within the memory itself, as opposed to transferring data back and forth between memory and the processor. This approach can considerably reduce data movement, leading to faster and more efficient computation. In-memory computing technologies can be particularly beneficial for tasks that involve large-scale data processing, such as FFT (Fast Fourier Transform), where significant time can be saved by minimizing data movement.

In Very Large Scale Integration (VLSI) design, the relationships among complexity, hardware efficiency, and run-time are integral to the overall performance of integrated circuits. Higher complexity often implies a greater demand for hardware resources. On the other hand, efficient hardware design aims to minimize resource usage while optimizing functionality, resulting in reduced power consumption and minimal area. Improving the runtime of FFT using advancements in technology can have a cascading effect on system-level performance, especially in domains like wireless communication. For wireless communication, FFT is a critical component used in modulation and demodulation, channel estimation, and equalization. Improving FFT runtime can have a significant impact on the system-level runtime for modules in wireless communication or bioinformatic signal processing [4]. Specifically, for wireless communication, where FFT plays a crucial role in modulation, demodulation, channel estimation, and equalization, a faster FFT runtime directly translates to quicker processing of signals.

A notable improvement in FFT runtime could lead to reduction in the overall runtime of wireless communication modules. This reduction contributes to faster data processing and lower latency, enhancing the efficiency and responsiveness of the communication system. The same principle applies to bioinformatic signal processing modules, where optimizing FFT runtime can result in accelerated data analysis and processing, leading to an overall improvement in system-level runtime. Altogether, these three factors are interconnected, and a higher complexity may impact hardware efficiency and runtime. Efficient designs are expected to achieve a balance between functionality and resource utilization, contributing to improved overall system performance and faster run-times, especially critical in real-time applications. In VLSI, achieving an optimal relationship between complexity, hardware efficiency, and runtime is fundamental for designing effective integrated circuits.

Fourier transform plays a very important role in the analysis of signal spectrum by converting a signal from time-domain to frequency-domain, and vice-versa. With the help of Fourier transform, a signal can be segregated into different frequency components which are difficult to ascertain in the time-domain. In digital signal processing (DSP), a discrete version of the Fourier transform is used, namely discrete Fourier transform (DFT). However, its real-time implementation is difficult due to high-computational cost. Cooley and Tukey paved the way for electronic devices by introducing fast Fourier transform (FFT) which computes the DFT efficiently [5]. FFT and inverse FFT (IFFT) are fundamental building blocks used in majority of DSP systems. It is one of the most highly researched algorithm in DSP, with applications ranging from biomedical, communication and instrumentation systems to various consumer electronic products such as TV, mobile phones, etc.

Although its algorithm is quite easily understood, the variants of the architectures and specifications are significant and are time consuming for hardware engineers. Over the years, researchers have made continuous efforts for improving the performance of FFT processors in terms of throughput, area and power consumption. By adjusting the operation according to the system's needs, the efficiency of resource usage can be improved. The evolution in very large scale integration (VLSI) technology has provided several million times higher integration complexity along with better performance and functionality per die in the last decades. However, with the high speed of operation being in demand, the power consumption of these devices increases significantly. The increase in power consumption has led to the current circumstances, where prospective FFT applications, previously constrained by performance, are now typically constrained by available power supply. Hence, it is essential to explore

methodologies for reducing the power consumption. In particular, low-area and low-power FFT architectures are in demand due to the recent remarkable growth in the number of portable applications which rely on limited power supply and are expected to occupy as less area as possible.

Based on the type of input, FFT architectures have been categorized as complex-valued FFT (CFFT) and real-valued FFT (RFFT). When the input samples are real, the spectrum is symmetric and approximately half of the operations are redundant. In applications such as speech, audio, image, radar, and biomedical signal processing [6, 7], a specialized hardware implementation is best suited to meet real-time constraints. In implantable or portable devices, this approach also saves power which is a critical constraint. Even though specific algorithms for the computation of the RFFT [8] have been proposed in the past, these algorithms lack regular geometries for designing pipelined architectures. These approaches are based on removing the redundancies of the CFFT when the input is real. The implementation of these algorithms can be further investigated for hardware-efficient implementation in DSP processors.

A brief introduction to the FFT algorithm and its hardware architectures are discussed in the following subsections.

## 1.2 The FFT Algorithm

Consider an $N$-point sequence $x[n] = [x(0), x(1), x(2), ....., x(N-1)]$. Its DFT can be expressed as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^k \quad \text{where } W_N^k = e^{\frac{-j2\pi nk}{N}}; k \in [0, N-1] \tag{1.1}$$

The straightforward implementation of (1.1) requires $N(N-1)$ complex additions and $N^2$ complex multiplications. In 1965, Cooley-Tukey introduced the FFT algorithm to compute the DFT in an efficient manner [5]. FFT exploits the symmetry and periodicity properties of the twiddle factor (TF), $W_N^k$. These two properties are stated below.

$$\text{Symmetry property:} W_N^{k=N/2} = -W_N^k \tag{1.2}$$

$$\text{Periodicity property:} W_N^{k=N} = W_N^k \tag{1.3}$$

When $N$ is an even integer, the input sequence $x[n]$ can be decomposed into two sub-sequences $x[2n]$ and $x[2n+1]$ of size $(N/2)$ each, comprising of the even-indexed and odd-indexed elements respectively for $0 \leq n \leq (N/2) - 1$. This algorithm is known as Decimation-in-Time (DIT) FFT.

When $N/2$ is again an even integer, each of the pair of sub-sequences $x[2n]$ and $x[2n+1]$ can be further decomposed into two sub-sequences corresponding to their even-and odd-indexed elements. Using the time-decimated sub-sequences $x[2n]$ and $x[2n+1]$, Equation (1.1) can be expressed as

$$X[k] = \sum_{n=0}^{N/2-1} x[2n] W_N^{(2n)k} + \sum_{n=0}^{N/2-1} x[2n+1] W_N^{(2n+1)k} \tag{1.4}$$

$$X[k+N/2] = \sum_{n=0}^{N/2-1} x[2n] W_N^{(2n)(k+N/2)} + \sum_{n=0}^{N/2-1} x[2n+1] W_N^{(2n+1)(k+N/2)} \tag{1.5}$$

where $0 \leq k \leq N/2-1$.

Let $x_0[k] = \sum_{n=0}^{N/2-1} x[2n] W_N^{(2n)k}$ and $x_1[k] = \sum_{n=0}^{N/2-1} x[2n+1] W_N^{(2n)k}$, therefore (1.4) and (1.5) after simplification becomes

$$X[k] = x_0[k] + W_N^k x_1[k] \tag{1.6}$$

$$X[k+N/2] = x_0[k] - W_N^k x_1[k] \tag{1.7}$$

Clearly, this requires $N\log_2 N$ complex additions and $(N/2)\log_2 N$ complex multiplications. Hence, significant algorithmic improvements have been achieved. This leads to a butterfly unit which is basically a 2-point DFT and the core processing element for the computation of FFT, as shown in Fig. 1.1.

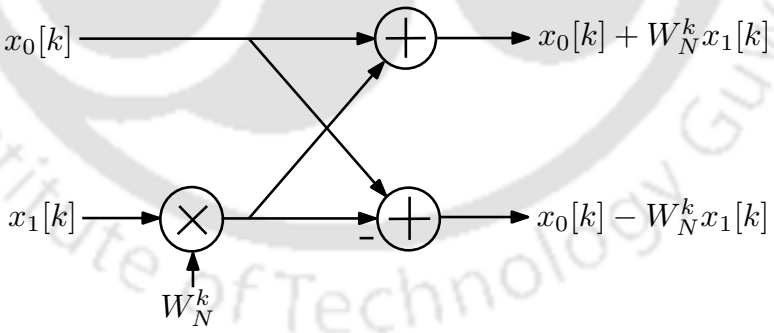


**Fig. 1.1.** Radix-2 butterfly unit

Another way of decomposition, when $N$ is an even integer, is popularly known as the Decimation-in-Frequency (DIF). Equation (1.1) can be split into first $N/2$ data points $[n = 0, ....., N/2-1]$ and last $N/2$ data points $[n = N/2, ...., N-1]$ and re-arranged as follows:

$$X[k] = \sum_{n=0}^{N/2-1} x[n] W_N^{(n)k} + \sum_{n=N/2}^{N-1} x[n] W_N^{(n)k} \tag{1.8}$$

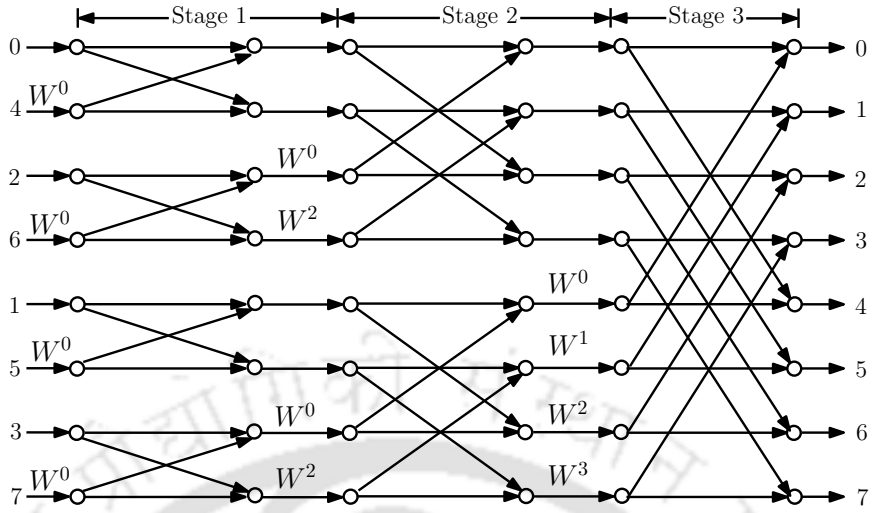


**Fig. 1.2.** DFG for 8-point DIT FFT

$$= \sum_{n=0}^{N/2-1} x[n]W_N^{(n)k} + W_N^{Nk/2} \sum_{n=0}^{N/2-1} x[n+N/2]W_N^{(n)k} \tag{1.9}$$

where $k, n = 0, 1, ..., N-1$.

These decompositions are depicted in the form of a data flow graph (DFG). The DFG of DIT and DIF FFT differ only in the locations of the TFs as opposed to the notion that in DIT FFT the data

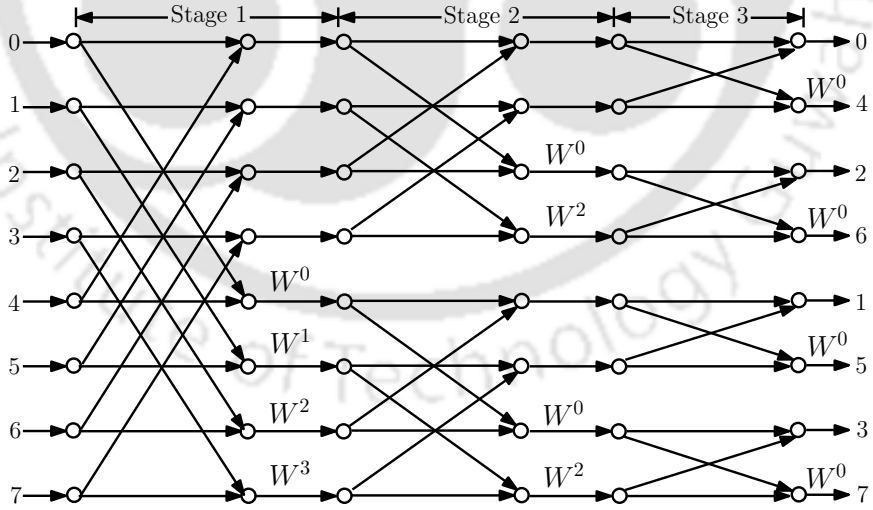


**Fig. 1.3.** DFG for 8-point DIF FFT

must arrive in bit-reversal and in DIF FFT data must arrive in natural order as shown in Fig. 1.2 and Fig. 1.3. This can be understood clearly by redrawing the DIT FFT in Fig. 1.2 so that the order of the input and output are same as those of DIF FFT. By comparing Fig. 1.3 and Fig. 1.4, it can be observed that the DIT and DIF FFT differ only on the location of TFs.

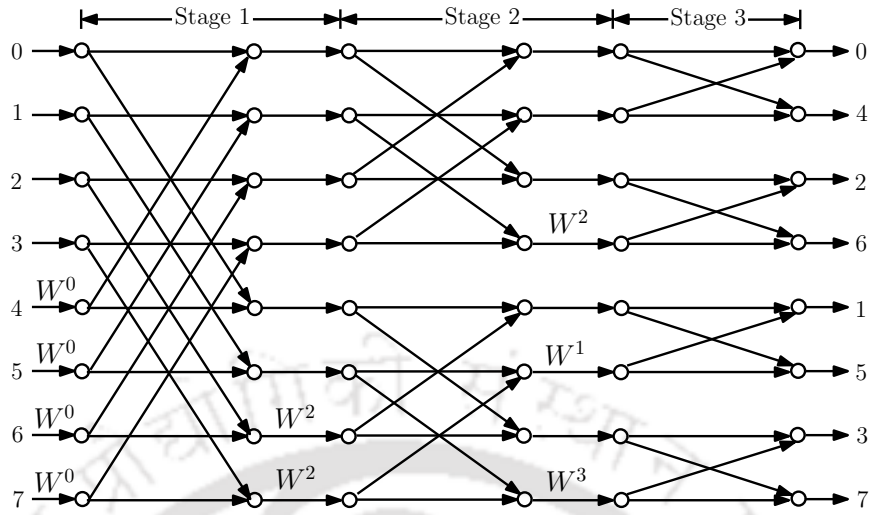


**Fig. 1.4.** DFG for 8-point DIT FFT with the input and output order of DIF FFT

When $N$ is a power-of-2 integer, such decomposition is used recursively to compute an $N$-point FFT in $\log_2 N$ stages. The conventional Cooley-Tukey radix-2 FFT algorithm requires $(N/2)\log_2 N$ complex butterfly operations for a $N$-point FFT computation. Along with the butterfly unit, one needs memory to store the complex twiddle factors and complex intermediate data, complicated addressing logic and control circuitry. Combining all these circuit modules it is expected that the power consumption of the entire processor will be quite high.

In order to speed up the FFT computation, more advanced solutions have been proposed using an increase of the radix-size (radix-4, radix-8) [9, 10]. These approaches result in an increase of arithmetic complexity within the butterfly itself. The radix-4 FFT algorithm is most popular and has the potential to satisfy the current need. However, a single radix-4 butterfly requires three complex multiplications and eight complex additions. Thus, in order to carry out one radix-4 butterfly operation per clock cycle, one needs to complete 12 real multiplications and 16 real additions in each cycle. Since multipliers are typically very power-hungry elements in a VLSI design, this type of arrangement results in significant power consumption. Hence, most of the architectures presented in this thesis are focused on radix-2.

Another optimization to balance between computation efficiency and complexity is the split-radix FFT [8, 11–13]. The split-radix FFT algorithm reduces the number of arithmetic operations and memory requirements compared to radix-2 FFT. The split-radix FFT is built upon the principles of Cooley-Tukey FFT and the radix-2 algorithm. It factorizes the FFT length into factors of 2 and

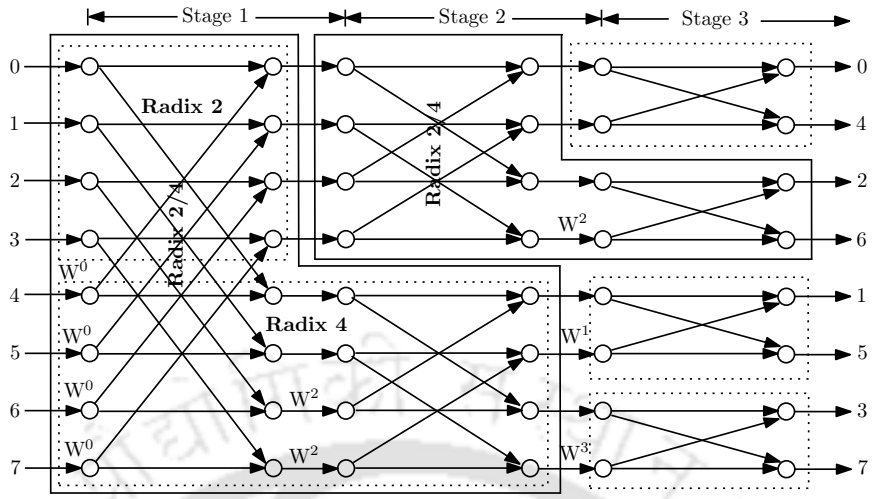


**Fig. 1.5.** DFG for 8-point Split-radix FFT

4 as shown in Fig. 1.5, leading to a more efficient computational structure. The split-radix FFT reduces the number of multiplications and additions required to compute the FFT, resulting in a faster execution time. Its efficiency and reduced computational complexity make it a valuable tool for real-time and high-performance applications. It is specifically advantageous in hardware where a multiplier/accumulator is the fundamental processor, as in the case of several VLSI implementations. For better understanding, the computational complexity in terms of number of non-trivial multiplications (multiplications other than 1, -1, $j$,$-j$) in different algorithms are listed in Table 1.1.

Table 1.1: Number of non-trivial multiplications in different FFT algorithms

| $N$ | Radix-2 | Radix-4 | Split-radix |
|---|---|---|---|
| 8 | 2 | 2 | 2 |
| 16 | 10 | 8 | 8 |
| 32 | 34 | 28 | 26 |
| 64 | 98 | 76 | 72 |
| 128 | 258 | 204 | 186 |
| 256 | 642 | 492 | 456 |
| 512 | 1538 | 1196 | 1082 |
| 1024 | 3586 | 2732 | 2504 |

## 1.3 FFT Architectures

The FFT architectures primarily consist of butterflies, multipliers and circuits for data-reordering. The butterflies and multipliers are the computational units in the flow graph while the data-reordering

circuits do not perform any computation. The data-reordering circuits are often made up of memories or buffers, and multiplexers. Data-reordering is one of the important factors in FFT computation. They rearrange the data, as per the flow graph, so that correct samples are available for the next stage computation. This requires conflict-free addressing strategies which decides the time at which a particular butterfly and multiplication is to be computed. This is performed by interchanging and bypassing samples. This results in a large number of hardware architectures of FFT. Depending on the application requirements such as area, power, throughput and latency, one has to choose which architecture is more suitable. Throughput is defined as the number of data samples processed per second whereas latency is the time taken by a sample from the input to the output. To meet the real-time requirements of modern applications, high-throughput and low-latency are in demand. These can be achieved by direct implementation of the flow graph. However, it would result in large area and high power consumption. An alternative solution to this is to reuse the hardware resources to perform the butterflies and multiplication operations.

Taking all these into consideration, there exists several types of FFT architectures. They are broadly categorized as: memory-based and pipelined architectures. Depending on the application requirement, one has to choose which architecture is more suitable. Memory-based FFT architectures are preferred whenever speed is not a concern but area-constraint is present. On the other hand, pipelined architectures are preferred when high speed is required. A classification of the different types of FFT architectures is shown in Fig. 1.6. These are explained in the following sections.

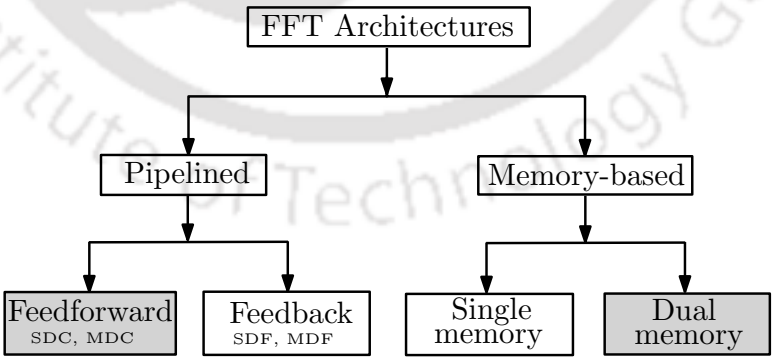


**Fig. 1.6.** A broad classification of FFT architectures. [1]

### 1.3.1 Memory-based architectures

Memory-based FFT architectures are a type of FFT design that utilizes memories to store intermediate results and twiddle factors in the computation of an FFT. They are comprised of one or more memory units and a processing element (PE) that includes butterfly circuits and multipliers. The FFT is calculated through a series of iterations in which data is retrieved from memory, processed in the PE, and stored back into memory. This cycle is repeated until all necessary computations have been carried out. The PE is responsible for calculating butterfly operations and TF multiplications corresponding to each stage of the FFT flow graph. These types of architectures provide low complexity, but may require more memory elements compared to other FFT designs.

There exists a number of memory-based FFT architectures. The most basic configuration, namely, the single memory-based FFT involves a memory, a butterfly unit, and a TF multiplier. These elements are responsible for computing all the butterflies and multiplications of the FFT. This configuration requires minimal hardware components, but it is associated with a high latency and low throughput. An illustration of this architecture can be seen in Fig. 1.7, which includes a memory and a PE. This configuration is not capable of processing a continuous flow of data due to the requirement of computing the entire FFT before clearing the memory and accepting new samples. To simulate a continuous flow, another memory block can be added to store incoming data while the current FFT is being calculated as depicted by Fig. 1.8. This configuration is referred as the dual memory-based FFT. During each clock cycle, a sample is taken from the first memory and the result of the FFT computation is saved in the second memory. Once an FFT stage is computed, the two memory units switch roles, with the second memory becoming the input and the first memory becoming the output. This cycle repeats until the FFT computation is complete. This design enables simultaneous reading and writing of samples into memory within a single clock cycle, and the output of the FFT can be retrieved from either of the memory units, based on the number of FFT stages. Additionally, the throughput of memory-based architectures is influenced by the latency and the number of points in the FFT. An increase in the latency of the PE leads to an increase in computation time for an iteration, and thus, a decrease in throughput. Similarly, larger FFTs result in a greater number of iterations and a lower throughput. These dependencies make memory-based architectures less appropriate for real-time processing.

Overall, memory-based FFT architectures are suitable for applications that require low-complexity

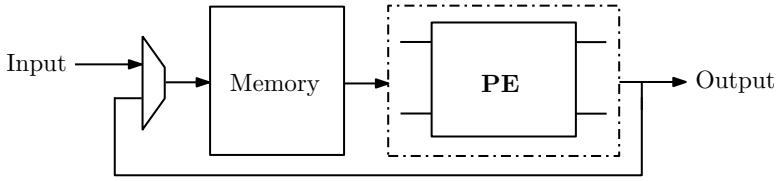


**Fig. 1.7.** Single memory-based FFT. PE: Processing Element

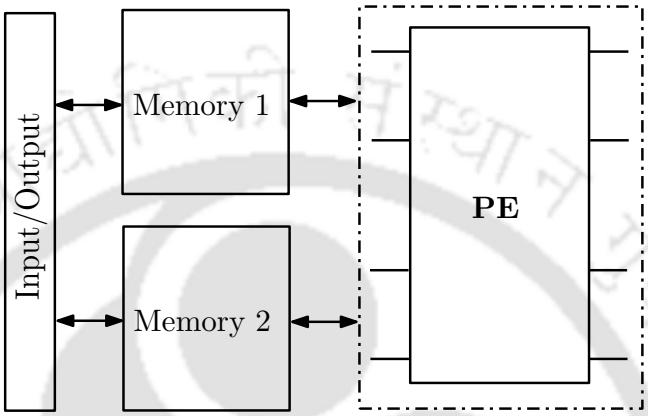


**Fig. 1.8.** Dual memory-based FFT

FFT computation with moderate to high throughput requirements. They provide a trade-off between complexity and performance and can be a good choice for low-area, low-power implementations. One of the key features of memory-based FFT architectures is the use of memory accessing schemes to optimize the performance of the FFT computation. These memory accessing schemes are carefully designed to maximize the utilization of the memory while minimizing the complexity of the design.

### 1.3.2 Pipelined architectures

Pipelined FFT architecture aims to reduce the computation time of FFT by breaking down the FFT calculation into smaller stages that can be processed simultaneously. The idea behind this architecture is to pipeline the computation by dividing the calculation into smaller stages and processing each stage in parallel, allowing multiple stages to be in progress at the same time. A general pipelined FFT architecture for an $N$-point FFT and $\log_r N$ number of pipeline stages is shown in Fig. 1.9, where $r$ is the radix. This architecture enables faster computation of FFT compared to non-pipelined architectures. Unlike memory-based architectures, the throughput of pipelined architectures is independent of both the latency and the number of FFT points. This means that the computational delay can be reduced by adding registers, which raises the clock frequency and enhances the overall throughput [14].

As a result, the pipelined architectures are capable of delivering fast processing speeds, making them a suitable choice for real-time computing.

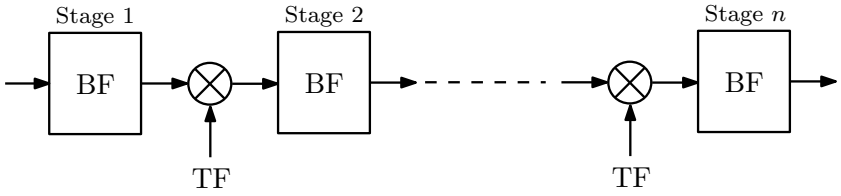


**Fig. 1.9.** Pipelined FFT

Pipelined architectures, also known as streaming architectures, contain either feedback or feedforward datapaths [15]. In feedback architectures, every stage has feedback loops. The output of the butterfly is fed back to the memory of the same stage, where they are held until the subsequent stage requires them. In general, they have a throughput of one sample per clock cycle and thus require less hardware. On the other hand, feedforward architectures advance the sample to the subsequent stage after being processed by the butterflies and the multipliers. The majority of them enable the computation of several samples in parallel. This results in a throughput higher than that of feedback architectures at the cost of more hardware. Further, on the basis of the number of inputs processed at a time, they are again categorized into serial and parallel architectures. This results in four types of architectures: single-path delay feedback (SDF) architectures [1, 16–18], multi-path delay feedback (MDF) architectures [19–21], single-path delay commutator (SDC) [22] and multi-path delay commutator (MDC) architectures [9,10,23–25]. The MDF architecture, which is composed of multiple parallel SDF architectures, and the MDC architecture offer high throughput but at the cost of increased hardware expenses. On the other hand, serial pipelined FFT architectures (SDF, SDC) are less complex and consume less power compared to parallel pipelined FFTs, but they have lower throughput.

There has been a continuous effort for several decades to reduce the hardware cost of pipelined FFTs, while simultaneously improving their performance. Some approaches which have been investigated to reduce the computational complexity of FFT algorithms [12, 26–39].

## 1.4 Literature on Hardware-efficient FFT Architectures

In the past, several authors have suggested efficient architectures for the computation of complex-valued FFT (CFFT) [24, 32, 33, 40]. Over the years, the focus has shifted towards efficient implementation of real-valued FFT (RFFT) architectures [8, 15, 26, 41, 42], as most of the physical signals

are real-valued. Moreover, CFFT architectures cannot achieve the same efficiency while processing a RFFT due to Hermitian symmetry in real-valued signal spectrum which eliminates redundant computations. Thus, only half of the outputs need to be computed. In the past, there has been continuous efforts to improve the performance of memory-based FFT architectures [15,41,42]. An in-place RFFT processor with a conflict-free memory accessing scheme to achieve low hardware usage and high resource utilization was proposed in [15]. Another architecture was proposed in [41], to further reduce the hardware usage and the number of computation cycles while maximizing the resource utilization. It is found that the number of cycles required for FFT computation can also be reduced by increasing the number of processing elements (PEs). Another scheme has been reported which has the advantage of supporting continuous-flow operation and normal-order output while minimizing the resource usage. Further, the number of multipliers required for implementation depends on the PEs used, rather than the number of stages. Therefore, hardware cost is relatively lower as compared to pipelined architectures for large number of points. Serial FFT architectures are inherently streamlined in nature, and suitable for low-area, low-power and low throughput requirements. It is primarily due to the lower usage of computational resources compared to its parallel counterparts. Earlier, several serial pipelined RFFT architectures have been presented in [2, 17, 43, 44]. In [17], interleaved datapaths were used to conserve hardware resources, but the butterfly units remain underutilized. Later, a DFG was proposed in [2] to achieve full hardware utilization resulting in reduction in the number of resources. In [43], a DFG for a modified serial commutator was proposed to achieve lower latency with lower hardware resources. Recently, a design was proposed in [44] to achieve minimum hardware by introducing a TF multiplier with fewer registers. On the other hand, parallel FFTs are suitable for high-throughput applications with low area constraints. However, it results in large area and power consumption due to high hardware cost. Hence, fully-parallel FFTs where each addition and multiplication in the flow graph requires an adder and a multiplier have not been explored explicitly. In the past, fully-parallel FFTs have been employed in applications such as high definition (HD) streaming, chromatic dispersion filtering, beamspace processing and radar [3, 45–47]. So far, only a few fully-parallel architectures, viz. radix-$2^k$, radix-4 and split-radix (SR) have been reported in the literature either to realize high-throughput or low-energy [3, 45–47].

All of these designs mainly focused on maximizing the hardware utilization and minimizing resource usage by introducing a new DFG and/or reducing the number of registers in the processing elements

mainly in the multiplier unit. The multipliers are the main source of power consumption and chip-area for the FFT implementation. In FFT, the multipliers are either trivial (i.e., multiplying with $1, 1, -j, j$) or constant (i.e., multiplying with fixed or variable twiddle coefficients). The reduction of multiplier complexity has remained a topic of intense research over the last three decades because the computational requirement becomes high as the size of the FFT increases. As a consequence, the real-time operation of the FFT is difficult, especially because of the multiplier complexity. In earlier studies, different types of multipliers have been used to implement the twiddle factor multipliers in FFT. Traditionally, a complex multiplier can be used consisting of four real multipliers, although it can be also carried out by three real multipliers [48]. Apart from these, conventional shift-and-add, Booth's, modified Booth's, Wallace, Dadda and CORDIC-based multipliers have been used to implement the complex multiplier [18,49–53]. In exploring various multiplier design options concerning area requirements and critical path delays, the conventional shift-and-add multiplier is recognized for addressing low area needs but is accompanied by very high critical path delays. Similarly, both serial and serial/parallel multipliers aim for reduced area requirements but may result in notably high critical path delays. Wallace and Dadda design-based parallel multipliers, though commonly employed for achieving low output latencies through parallel addition of partial products, but are associated with high area requirements. The Booth and Wallace/Dadda multiplier schemes are mentioned in the context of their characteristics, emphasizing the inherent trade-offs between area requirements and output latencies [49, 50]. Kumm et al. [51] and Walters [52] have implemented area-efficient radix-4 multipliers using the modified Booth's algorithm. Notably, their implementations sidestep partial products compressor trees but exhibit large critical path delays. CORDIC-based multipliers [18] have been used to implement the constant multiplier in serial FFTs. However, due to its iterative nature, it can lead to increased latency in comparison to non-iterative algorithms. The number of iterations required depends on the precision needed for the computation. Higher precision requirement will result in higher latency and more hardware resources compared to alternative methods, especially in terms of the required shifters and adders. This can impact the overall efficiency and area utilization of the hardware implementation.

## 1.5 Motivation

In today's digital era, many applications demand real-time signal processing with specific constraints in terms of area, clock frequency, power consumption, throughput and latency. In order to achieve the best performance, hardware devices such as FPGAs (Field Programmable Gate Arrays) or ASICs (Application Specific Integrated Circuits) are used to implement the signal processing algorithms. However, hardware implementation of FFT algorithms is totally different from its mathematical counterpart. From mathematical point of view, some operations might seem trivial but it might not be the case while implementing in hardware. An FFT algorithm can be implemented in a number of ways in hardware. One way is the direct implementation of the algorithm where every operator in the algorithm is replaced by its equivalent hardware e.g. addition by an adder, multiplication by a multiplier and so on. However, it results in high complexity in case of higher order FFT implementations, leading to high area and high power consumption. Another approach is to reuse hardware resources which results in less area but at the cost of increase in latency. Other factors such as number of bits required to maintain the accuracy of the results, choosing/designing an efficient multiplier to perform the multiplications must also be taken into consideration in hardware. Though none of these information is reflected through the algorithm, all are important aspects of hardware implementation which must be taken into account depending on the constraints such as area, power, throughput, latency, etc. Over the years, researchers have made continuous efforts for improving the performance of FFT processors in terms of throughput, area and power consumption. However, the potential for further improvements still exists. Possible outcomes include more advanced FFT architectures with higher throughput and less area, and optimization of multipliers. Another outcome may be low-latency FFTs which are in demand in real-time applications. Further, trade-off between power consumption and throughput of FFT architectures has not been explored extensively. These may be achieved by designing hardware-efficient architectures that consume low-area and low-power without compromising much in terms of speed of operation.

## 1.6 Problem Formulation

A lot of research has gone into reducing the complexity of designs at both algorithmic and architectural levels. However, hardware-efficient implementations assist not only in achieving complexity reduction but also result in lower leakage power due to less number of hardware elements. However,

focusing primarily on reducing the design complexity might lower the system's throughput without attaining the desired performance. The right optimization either reduces the complexity with respect to the system's performance or increases the throughput for a given number of computations. This thesis focuses on similar problems in principle with an aim of reducing the complexity of FFT architectures through hardware-efficient implementations without compromising much on throughput.

As mentioned earlier, hardware-efficiency is a major concern for the implementation of FFTs. As the FFT size increases, both chip area and power requirements go up, and can make real-time operation difficult. The implementation complexity is mainly dominated by the presence of bulky multipliers. Possible solutions might include multiplierless approaches such as shift-add multipliers and exploiting the redundancies in the partial products. In this thesis, three types of FFT architectures based on different FFT algorithms are proposed. Majority of the work focuses on realizing RFFT architectures. The tasks include choosing the appropriate FFT algorithm along with efficient implementation of multiplier for realizing low-complexity, energy-efficient and low-latency FFTs leading to hardware-efficient architectures. Evaluation of the performance of the proposed designs versus that of the state-of-the-art designs are carried out. The intended capabilities and advantages of the proposed designs are validated through ASIC and FPGA results.

## 1.7 Thesis Organization

This thesis aims to increase the hardware efficiency of FFT architectures and their variants. The performance of all the proposed architectures are evaluated and compared with state-of-the-art designs. The thesis is organized as follows.

Chapter 1 is the introductory chapter which provides a brief about the importance of FFT in DSP systems, the FFT algorithm and architectures. Further, a literature survey on existing architectures is presented. It discusses the motivation behind the work and major research in the complexity reduction of FFT over the past decades. The problem formulation is also presented in Chapter 1. The chapter ends with the organization of the thesis.

Chapter 2 focuses on the implementation of serial pipelined FFT to process real-valued signals. A new data mapping scheme is presented to obtain a normal order input-output without the requirement of a post-processing stage. It facilitates reduction in computational workload on the hardware resources and it is confirmed through mathematical derivations. Further, an effective approach to minimize the

hardware complexity of the multiplier based on a novel symmetrical power-of-two (POT) scheme for efficient implementation of serial RFFT is presented.

Chapter 3 presents two memory-based FFT architectures for real-valued signals. An alternative to the pipelined FFT is the memory-based FFT. Pipelined FFT offers high-throughput while memory-based FFTs offer low-area. Parallel processing or high-radix butterfly units can be employed to increase the throughput of memory-based architectures. Both the proposed designs employ processing elements (PEs). However, Type-I PE processes four inputs while Type-II PE processes two inputs in parallel. The PE in Type-I design contains a half-complex multiplier, and that of Type-II design contains a quarter-complex multiplier. A new conflict-free memory addressing scheme is presented along with the architectural details of both the designs.

Chapter 4 presents a fully-parallel split radix (SR) FFT. Fully-parallel pipelined FFT offers the highest throughput but requires high area and power. In fully-parallel FFTs, each addition and multiplication in the flow graph requires an adder and a multiplier, respectively. The multipliers are considered as the main source of area and power consumption in FFT realization. To keep the multipliers to a minimum, the SRFFT is chosen for implementing the proposed design. However, when pipelining is employed to increase the throughput, the number of pipelined registers increases significantly in an SRFFT thereby increasing the power consumption and clock latency. It prevents the SRFFT from taking advantage of its inherent low computational complexity. In this work, an efficient FFT processor based on the SR algorithm is presented to reduce the hardware complexity of fully-parallel FFTs.

Finally, Chapter 5 concludes with a summary of the thesis and future research directions.

भारतीय प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Technology Guwahati

# 2

# Serial-Pipelined FFT Architectures for Real-Valued Signals

Contents

## 2.1 Introduction

FFT is an essential digital signal processing tool with numerous applications in the areas of biomedical, communications, and instrumentation [6, 7, 54–58]. During the last 50 years [59], there has been substantial development in the field of FFT hardware architectures. However, they have not been explored explicitly for low-throughput applications such as biomedical signals, i.e., electroencephalograph (EEG) or electrocardiogram (ECG). In general, these biomedical signals are sampled between 256 Hz to 1 kHz, and when CFFT operates over real-valued input samples, it becomes inefficient for processing. As discussed in Section 1.4 of Chapter 1, for real-valued signals, only half of the outputs need to be computed [8, 60]. The elimination of the redundant computations results in an irregular data flow graph (DFG). Every stage must have an equal number of inputs and outputs for pipelined processing. Hence, the complex values generated after a TF multiplication are separated into real and imaginary components to achieve a regular DFG for RFFT.

Based on the number of samples processed per clock cycle, FFTs can be categorized into two groups: serial and parallel. Serial FFT architectures are inherently streamlined in nature and suitable for low-area/low-power and low throughput requirements such as biomedical applications. This is primarily due to the lower usage of computational resources compared to its parallel counterparts. In the recent past, several serial pipelined RFFT architectures have been presented [2, 17, 43, 44]. In [17], interleaved datapaths are used to conserve hardware resources, but the BFs remain underutilized. In [61], a half-butterfly operation is introduced to further enhance the utilization of butterfly units over existing serial-pipelined FFTs. However, when it processes real-valued data, the butterfly units become under-utilized. This has been addressed in [2] by employing half-butterfly units for the half-butterfly operations. In other words, the half-butterfly unit has a single real adder per stage of FFT computation and takes two clock cycles to compute one butterfly operation. It solves the aforementioned problem, albeit extra registers are used to hold the data for multiplication with twiddle factor. Moreover, FFT computation suffers from an inherent problem of scrambled order of the output samples. As a consequence, the sorting of the samples in normal order necessitates a post-processing circuit, which results in an increase in both area and latency. In [43], a DFG for a modified serial commutator is proposed to achieve lower latency with lower hardware resources. Recently, a design is proposed in [44] to achieve minimum hardware by introducing a TF multiplier with fewer registers. All of these designs mainly focus on maximizing the hardware utilization and minimizing resource

usage by introducing a new DFG and/or reducing the number of registers in the processing elements mainly in the multiplier unit. The multipliers are the main source of power consumption and chip-area for the FFT implementation. In FFT, these multipliers are either trivial (multiplication with $1, 1, -j, j$) or constant (multiplication with fixed or variable twiddle coefficients). The reduction of multiplier complexity has remained a topic of intense research over last three decades. It is because the computational requirement becomes high as the size of the FFT increases. As a consequence, the real-time operation of the FFT is difficult, especially because of the multiplier complexity. In the past, shift-and-add [53] or CORDIC-based multipliers [18] have been used to implement the constant multiplier in serial FFTs. Due to their performance issues, they are not very suitable for serial FFT [62]. For certain accuracy requirements in RFFTs, it is desired to choose appropriate bits for the input data and the TF. It heavily depends on the desired accuracy and hardware needed. A few works in literature have studied the effect of fixed-point arithmetic on the FFT performance [63–66] and spectral analysis [67, 68]. To address these issues, this work proposes a new scheme which does not require a post-processing circuit and facilitates reduction in the computational workload on the hardware resources. To reduce the hardware complexity, a quarter complex multiplier is investigated, which performs the quarter operation of a complex multiplier in one clock cycle. In addition, a merged unit for butterfly computation and data-reordering for the last stage is proposed.

In general, a multiplier has partial products (PP) generation and accumulation stages with optimizations revolving around them to achieve better area-time complexities [69,70]. The accumulation in multiplier can be performed using either adder-tree reduction approach [71–76] or shift-accumulation (SA) [77, 78]. The former could result in a significant delay as it depends on the adder tree depth, while the latter depends on the PP generation and is suitable for low-throughput applications. In the past, some authors have reduced the complexity of multipliers for various signal-processing algorithms by utilizing different approaches. For instance, a modified Booth encoding has been used to simplify the PP generation. However, the odd-PP (OPP) requires more number of adders without a significant reduction in clock cycles. A method to obtain the OPP in terms of power-of-two (POT) differences has been suggested in [72]. Although the number of OPP partially reduces by processing multiple bits, these designs do not address the complexity of generating OPP. Thus, OPP is difficult to generate on hardware while storing them in physical LUT leads to high-cost [79]. The fundamental limitation of the SA-based multiplier is that the complexity of PP generation increases with reduction in the

number of clock cycles. Consequently, the area and power requirements also increase proportionally, which may not be suitable for biomedical applications. No study has so far shown a reduction in the complexity of generating OPP. It motivates us to design a completely new SA-based multiplier for efficient implementation of serial RFFT.

This chapter presents two approaches for implementation of serial RFFT. Section 2.2 presents the details of the first serial RFFT architecture. It is followed by Section 2.3, which presents the second RFFT architecture with a novel methodology to design the TF multiplier.

## 2.2 Proposed Methodology 1

The computation of FFT requires data-reordering (DR) at every stage to map the intermediate data in order to maintain a regular data flow. In case of memory-based FFT architectures, specific memory bank assignment schemes are employed, while in the case of pipelined FFT architectures, dedicated circuits are needed for DR. In principle, the hardware complexity of DR units cannot be reduced, since it maps the data from previous stage to next stage based on the shuffling requirement at that particular stage. In order to reduce the hardware complexity in serial-pipelined FFT, one possible solution is to reduce the complexity of the multiplier or to enhance the resource utilization of the half-butterfly units. This is feasible with the proposed data flow graph (DFG), quarter complex multiplier (QCM) and butterfly cum data-reordering (BFDR) units.

### 2.2.1 Data Flow Graph (DFG)

The proposed DFG for a 16-point FFT computation is shown in Fig. 2.1. Clearly, it follows a regular geometry necessary to implement a pipelined FFT architecture. Unlike conventional DFG [26], it indicates how the data is processed in a single-path FFT processor at different time instances across all the stages. Note that the order of data arrival is from top to bottom, where time indices of all the inputs/outputs are denoted by the letter '$t$'. According to the proposed DFG, each processing unit in the architecture would require a certain number of clock cycles indicated by their respective latency ($Lat$) at the bottom of the DFG in Fig. 2.1. The latency of the DR units depends on the shuffling requirements at that particular stage, whereas multipliers and butterfly units have fixed latency. Further explanation with architectural details of half-butterfly and DR units is described in the next section.

The proposed DFG for 16-point FFT would require cascaded DR units with latency 7 and 2 in

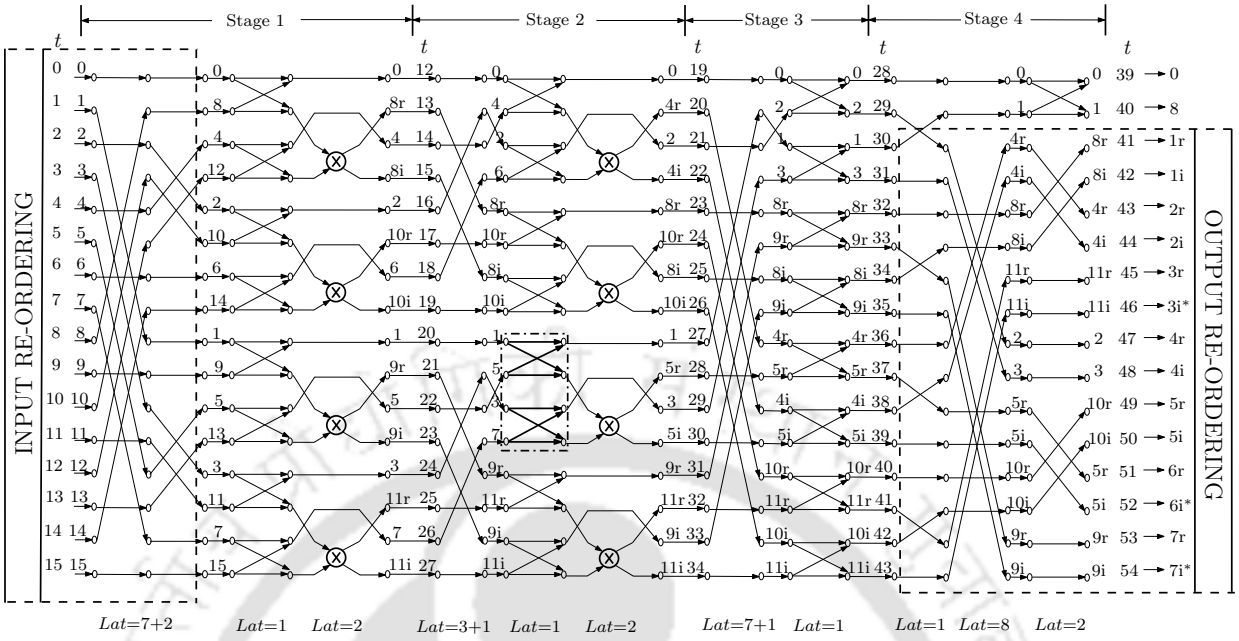


**Fig. 2.1.** Proposed data flow graph (DFG) for a 16-point FFT, where '$t$' denotes time indices of the samples across the stages.

stage 1 to reorder the input, while the output DR begins in stage 4 before the computation of the last butterfly. Notably, the butterfly operations and the multiplications appear together in all the stages, except last two stages. Thus, the proposed serial architecture requires a total of $\log_2 N - 2$ multipliers. It is important to note that one output of each butterfly in two consecutive butterflies (shown within a dashed box in stage 2) enters into the same multiplier. In other words, one corresponds to the real part while the other corresponds to the imaginary part of a complex butterfly indicated by the notations '$r$' and '$i$' in Fig. 2.1 respectively. The computation of these two butterflies is performed in successive clock cycles as one output from each butterfly has to be processed through the same multiplier. In stage 3, only DR and butterfly operations are required as the multiplications are shifted to the previous stage 2, which is more efficient in terms of hardware. In other words, the multiplier in stage 2 is used to perform the original stage 2 multiplications alongwith the stage 3 multiplications which are now shifted to stage 2 in Fig. 2.1. This eliminates the requirement of multiplier at each stage. It is interesting to note that most of the arithmetic computations are completed within stage 3 except one butterfly computation in stage 4. It allows us to utilize stage 4 for output reordering along with the butterfly computation by introducing a merged unit for performing both butterfly computation and DR using the same unit. The proposed design utilizes half-butterfly units for computations across all

the rows of each stage, except in the last stage where the half-butterfly unit is utilized for only first two clock cycles. It allows better utilization of half-butterfly units in the proposed architecture over the existing real-valued serial commutator (RSC)-FFT architecture [2]. Moreover, the computational workload on individual half-butterfly units is also reduced. This can be understood by estimating the half-butterfly computations involved in the proposed and existing RSC-FFT designs stage-by-stage. In case of the proposed DFG, the computational workload on half-butterfly units ($\zeta_{BF}^{A,P}$) based on stage-by-stage computation is relatively easier to calculate, where superscript '$A, P$' indicates the additions in half-butterfly units for the proposed design and subscript '$BF$' indicates the half-butterfly operations. In general, the value $\zeta_{BF}^{A,P}$ for $N$-point FFT computation using the DFG (shown in Fig. 2.1) can be computed as

$$\begin{aligned}\zeta_{BF}^{A,P} &= N + N + N + ..... + 2 = N\sum_{i=1}^{n-1} 1 + 2 \\ &= N(\log_2 N - 1) + 2 \end{aligned} \tag{2.1}$$

where $n = \log_2 N$ is the number of stages. Similarly, the computational workload on half-butterfly units ($\zeta_{BF}^{A,E}$) for existing RSC-FFT architecture [2] using their DFG scheme could be expressed as

$$\begin{aligned}\zeta_{BF}^{A,E} &= N + \frac{N}{2} + \left(\frac{N}{2} + \frac{N}{4}\right) + ..... + (N-2) \\ &= 2N - 2 + N\sum_{j=1}^{n-2}\sum_{i=1}^{j}\left(\frac{1}{2^i}\right) \\ &= N(\log_2 N + 1) - 6 \end{aligned} \tag{2.2}$$

The reduction in computational workload on the half-butterfly units ($\Delta\zeta_{BF}$) for the proposed design with respect to the existing design [2] for $N$-point FFT can be calculated as

$$\Delta\zeta_{BF} = \zeta_{BF}^{A,E} - \zeta_{BF}^{A,P} = 2N - 8 \tag{2.3}$$

For clarity, an explicit plot depicting the reduction in computational workload with the number of FFT points is shown in Fig. 2.2. Further, it also illustrates the computational workload on half-butterfly units. It is clear that the reduction in computational workload on half-butterfly units would be more advantageous when the number of points in FFT computation increases. Thus, the proposed DFG scheme has direct impact on the power consumption.

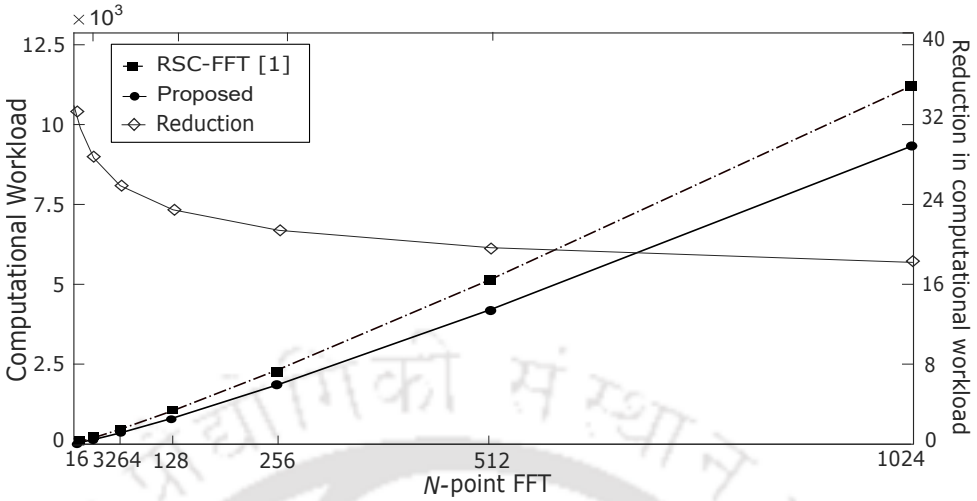


**Fig. 2.2.** Comparison of computational workload on half-butterfly units of proposed and existing RSC-FFT designs [2].

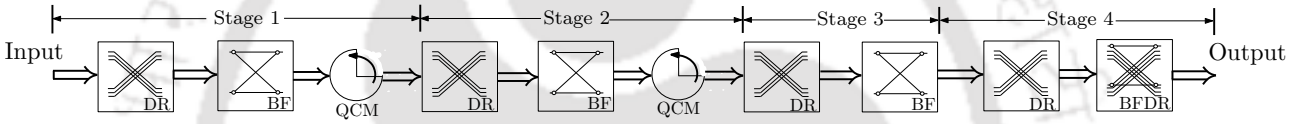


**Fig. 2.3.** Block schematic of serial-pipelined architecture for 16-point FFT. DR: Data-reordering, BF: Half-butterfly, QCM: Quarter complex multiplier, and BFDR: Butterfly cum data-reordering (BFDR).

### 2.2.2 Proposed Architecture

As discussed in previous section, the proposed DFG scheme reduces the computational workload on half-butterfly units while utilizing the same number of data-reordering units and multiplier units. In this section, the details of the proposed serial-pipelined architecture for 16-point FFT will be covered. The proposed architecture consists of four stages which include four DR units, three half-butterfly units, two QCM units and a BFDR unit, as shown in Fig. 2.3. As stated in the previous subsection, both half-butterfly and QCM units are employed in stage 1 and stage 2, whereas a half-butterfly unit and a BFDR unit are used in stage 3 and stage 4 respectively. The DR units are used in all the stages to shuffle the intermediate data. The architectural details of the DR, half-butterfly, QCM and BFDR units are explained below.

#### 2.2.2.1 Data-Reordering (DR) and Half-Butterfly units

The DR and half-butterfly units employed in the proposed design are identical to [2]. The data permutations and butterfly operations shown in the proposed DFG are performed by the DR and

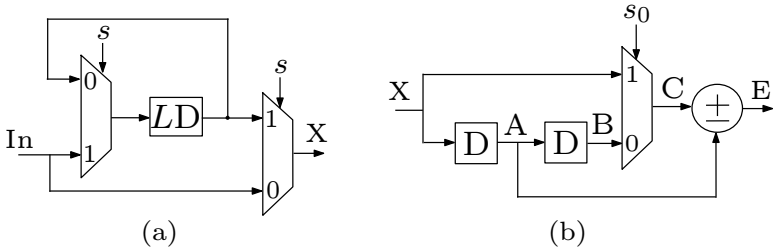


**Fig. 2.4.** (a) Data-reordering (DR) unit and, (b) Half-butterfly unit [2].

half-butterfly units respectively. The basic DR unit shown in Fig. 2.4(a) consists of a buffer of length '$L$' and two multiplexers to select the stored data or the input. If the control of multiplexer '$s$' is set to '1', then the sample is passed through the buffer, whereas if it is set to '0', then the input sample at that instant is interchanged with the sample '$L$' clock cycles apart. Unlike half-butterfly unit, the buffer length of DR unit is different for different stages, for example, in stage 1, two DR units are cascaded with $L = 7$ and $L = 2$, in stage 2, two DR units are cascaded with $L = 3$ and $L = 1$ whereas in stage 3, two DR units are cascaded with $L = 7$ and $L = 1$. For a better understanding, the FFT input data which is received in sequential order is considered. In Fig. 2.1, the input data is received in the order 0, 1, 2,...., 13, 14, 15. To ensure proper data alignment for the subsequent butterfly, the data needs to be arranged such that the samples that are to be processed in the same stage are grouped together. Although in Fig. 2.1, all the butterfly computations in stage 1 [(0, 8), (1, 9), (2, 10), (3, 11), (4, 12), (5, 13), (6, 14), (7, 15)] require samples that are 8 clock cycles apart, the butterfly computations in the subsequent stages require the even input samples [(0, 8), (4, 12), (2, 10), (6, 14)] to be computed prior to the odd input samples [(1, 9), (5, 13), (3, 11), (7, 15)] in stage 1 in this specific order. So, these cannot be performed with a single DR unit which can either bypass a sample or interchange two samples '$L$' clock cycles apart.

In stage 1 butterfly computation, '1 ($u_3u_2u_1u_0 = 0001$)' needs to be interchanged with '8 ($u_3u_2u_1u_0 = 1000$)' so that '0' and '8' are received together for the first butterfly computation. Further, '2 ($u_3u_2u_1u_0 = 0010$)' needs to be interchanged with '4 ($u_3u_2u_1u_0 = 0100$)' and '3 ($u_3u_2u_1u_0 = 0011$)' needs to be interchanged with '12 ($u_3u_2u_1u_0 = 1100$)', so that '4' and '12' are received together for the second butterfly computation. In all of these cases, it is observed that elementary bit-exchanges $u_3 \leftrightarrow u_0$ and $u_2 \leftrightarrow u_1$ are required. The number of registers required to perform this DR is calculated

as

$$D_0 = 2^3 - 2^0 = 7 \tag{2.4}$$
$$D_1 = 2^2 - 2^1 = 2$$

Hence, we need two cascaded DR units with latency 7 + 2 connected one after the other as shown in Fig. 2.5 to achieve the aforementioned order.

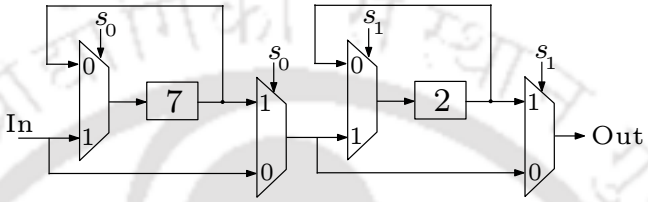


**Fig. 2.5.** Cascaded DR

The half-butterfly unit shown in Fig. 2.4(b) comprises of two registers, a multiplexer and a real adder. A full-butterfly unit typically requires one adder and one subtractor. A half-butterfly operation, on the other hand, involves only one adder or a subtractor. In summary, the term "half-butterfly unit" refers to a computational element that is a simplified version of the butterfly operation. It is called "half" because it employs half of the hardware required in a full-butterfly unit and handles half the number of operations compared to that of a full-butterfly operation in one clock cycle. It computes the addition of butterfly in one clock cycle and the subtraction of butterfly in the next clock cycle using a '$s_0$' signal. To understand the operation of both the units, an explicit timing diagram for the cascaded DR units and the half-butterfly unit in stage 1 is shown in Table 2.1.

#### 2.2.2.2 Quarter Complex Multiplier (QCM)

It is well known that the computation of FFT requires several complex multipliers. In general, each complex multiplier is considered equivalent to four real multipliers and two real adders. For instance, a complex multiplication in FFT can be expressed as $\mathcal{R}e + j\mathcal{I}m$, where $\mathcal{R}e$ and $\mathcal{I}m$ are real and imaginary components respectively, and $j = \sqrt{-1}$. These are usually in the form of $\mathcal{R}e = cx_i + dx_j$ and $\mathcal{I}m = cx_j - dx_i$, where $c$ (or $\cos(\alpha)$) and $d$ (or $\sin(\alpha)$) are real and imaginary components of twiddle factor respectively with $\alpha$ being the angle of rotation.

It is interesting to note that the single real multiplier can compute $cx_i, dx_j, cx_j$ and $-dx_i$ in four successive clock cycles which would be advantageous for efficient usage of hardware resources. In other

Table 2.1: Timing Diagram of DR Unit and Half-Butterfly Unit in Stage 1

| DR | | | | | | Half-BF | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Clk** | **In** | $s^1$ | $\mathbf{X}^1$ | $s^2$ | $\mathbf{X}^2$ | **Clk** | **X** | **A** | **B** | $s_0$ | **E** |
| 0 | $x_0$ | 1 | - | - | - | 9 | $x_0$ | - | - | - | - |
| 1 | $x_1$ | 1 | - | - | - | 10 | $x_8$ | $x_0$ | - | 0 | $x_0^1$ |
| 2 | $x_2$ | 1 | - | - | - | 11 | $x_4$ | $x_8$ | $x_0$ | 1 | $x_8^1$ |
| 3 | $x_3$ | 1 | - | - | - | 12 | $x_{12}$ | $x_4$ | $x_8$ | 0 | $x_4^1$ |
| 4 | $x_4$ | 1 | - | - | - | 13 | $x_2$ | $x_{12}$ | $x_4$ | 1 | $x_{12}^1$ |
| 5 | $x_5$ | 1 | - | - | - | 14 | $x_{10}$ | $x_2$ | $x_{12}$ | 0 | $x_2^1$ |
| 6 | $x_6$ | 1 | - | - | - | 15 | $x_6$ | $x_{10}$ | $x_2$ | 1 | $x_{10}^1$ |
| 7 | $x_7$ | 1 | $x_0$ | 1 | - | 16 | $x_{14}$ | $x_6$ | $x_{10}$ | 0 | $x_6^1$ |
| 8 | $x_8$ | 0 | $x_8$ | 1 | - | 17 | $x_1$ | $x_{14}$ | $x_6$ | 1 | $x_{14}^1$ |
| 9 | $x_9$ | 1 | $x_2$ | 1 | $x_0$ | 18 | $x_9$ | $x_1$ | $x_{14}$ | 0 | $x_1^1$ |
| 10 | $x_{10}$ | 0 | $x_{10}$ | 1 | $x_8$ | 19 | $x_5$ | $x_9$ | $x_1$ | 1 | $x_9^1$ |
| 11 | $x_{11}$ | 1 | $x_4$ | 0 | $x_4$ | 20 | $x_{13}$ | $x_5$ | $x_9$ | 0 | $x_5^1$ |
| 12 | $x_{12}$ | 0 | $x_{12}$ | 0 | $x_{12}$ | 21 | $x_3$ | $x_{13}$ | $x_5$ | 1 | $x_{13}^1$ |

LEGEND: $s^1$: control signal of 1st DR stage, $s^2$: control signal of 2nd DR stage, $\mathrm{X}^1$: output of 1st DR stage, $\mathrm{X}^2$: output of 2nd DR stage, $x_m^n$: output of $n^{\text{th}}$ stage with sample index $m$.

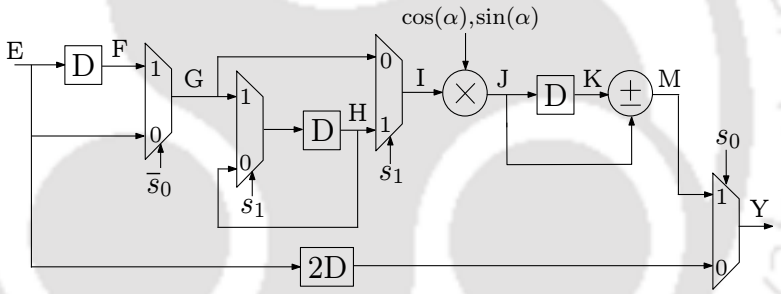


**Fig. 2.6.** Proposed Quarter Complex Multiplier (QCM).

words, a single real multiplier can perform a quarter operation of a complex multiplier in single clock cycle and has complexity one-fourth that of a complex multiplier. The proposed QCM includes five registers, four multiplexers, one real multiplier and an adder, as shown in Fig. 2.6. It is important to note that the proposed QCM requires only one register after the real multiplier, unlike [2]. Moreover, there is no need to bypass the half-butterfly unit because of the proposed DFG. Thereby, it reduces the switching activity while transferring the data through or over the QCM unit. The control signal $\overline{s}_0$ to the first multiplexer decides which data passes on to the multiplier, as indicated in Fig. 2.6. The topology of two multiplexers and a register driven by the control signal $s_1$ is identical to DR unit. To understand the operation of QCM unit, it is convenient to consider both half-butterfly unit and QCM unit together, as shown in Fig. 2.4(b) and Fig. 2.6 respectively. Both of them operate sequentially

Table 2.2: Timing Diagram of Proposed QCM (Fig. 2.6) in Stage 1

| **Clk** | $s_0$ | **E** | **F** | **G** | **H** | $s_1$ | **I** | **J** | **M** | **Y** |
|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 0 | $x_0^1$ | - | - | - | - | - | - | - | - |
| 11 | 1 | $x_8^1$ | - | $x_8^1$ | - | 1 | - | - | - | - |
| 12 | 0 | $x_4^1$ | $x_8^1$ | $x_8^1$ | $x_8^1$ | 1 | $x_8^1$ | $cx_8^1$ | - | $x_0^1$ |
| 13 | 1 | $x_{12}^1$ | - | $x_{12}^1$ | $x_8^1$ | 0 | $x_{12}^1$ | $dx_{12}^1$ | $cx_8^1 + dx_{12}^1$ | $cx_8^1 + dx_{12}^1$ |
| 14 | 0 | $x_2^1$ | $x_{12}^1$ | $x_{12}^1$ | $x_8^1$ | 1 | $x_8^1$ | $dx_8^1$ | - | $x_4^1$ |
| 15 | 1 | $x_{10}^1$ | - | $x_{10}^1$ | $x_{12}^1$ | 1 | $x_{12}^1$ | $cx_{12}^1$ | $cx_{12}^1 - dx_8^1$ | $cx_{12}^1 - dx_8^1$ |
| 16 | 0 | $x_6^1$ | $x_{10}^1$ | $x_{10}^1$ | $x_{10}^1$ | 1 | $x_{10}^1$ | $cx_{10}^1$ | - | $x_2^1$ |
| 17 | 1 | $x_{14}^1$ | - | $x_{14}^1$ | $x_{10}^1$ | 0 | $x_{14}^1$ | $dx_{14}^1$ | $cx_{10}^1 - dx_{14}^1$ | $cx_{10}^1 - dx_{14}^1$ |
| 18 | 0 | $x_1^1$ | $x_{14}^1$ | $x_{14}^1$ | $x_{10}^1$ | 1 | $x_{10}^1$ | $dx_{10}^1$ | - | $x_6^1$ |
| 19 | 1 | $x_9^1$ | - | $x_9^1$ | $x_{14}^1$ | 1 | $x_{14}^1$ | $cx_{14}^1$ | $cx_{14}^1 + dx_{10}^1$ | $cx_{14}^1 + dx_{10}^1$ |
| 20 | 0 | $x_5^1$ | $x_9^1$ | $x_9^1$ | $x_9^1$ | 1 | $x_9^1$ | $cx_9^1$ | - | $x_1^1$ |
| 21 | 1 | $x_{13}^1$ | - | $x_{13}^1$ | $x_9^1$ | 0 | $x_{13}^1$ | $dx_{13}^1$ | $cx_9^1 - dx_{13}^1$ | $cx_9^1 - dx_{13}^1$ |

LEGEND: $s^1$: control signal of 1$^{\text{st}}$ DR stage, $s^2$: control signal of 2$^{\text{nd}}$ DR stage, X$^1$: output of 1$^{\text{st}}$ DR stage, X$^2$: output of 2$^{\text{nd}}$ DR stage, $c$: cos($\alpha$), $d$: sin($\alpha$), $x_m^n$: output of $n^{\text{th}}$ stage with sample index $m$.

on sample-by-sample basis in two modes:

**Mode 0:** They perform the addition using the half-butterfly unit in one clock cycle and the result is passed via a bypass line to the multiplexer with output Y.

**Mode 1:** They perform the subtraction using the half-butterfly unit in the next clock cycle and the result is passed on for further processing. Subsequently, it undergoes real multiplication in the QCM unit which takes two clock cycles to compute $cx_i + dx_j$ (or $cx_j - dx_i$) using the four quarter complex terms $cx_i, cx_j$, $+dx_j$ and $-dx_i$, where $i, j$ denote the sample indices with $i \neq j$.

The complete operation of proposed QCM is explained in Table 2.2 using a separate timing diagram for each node of QCM unit mentioned in Fig. 2.6.

#### 2.2.2.3 Butterfly-cum-Data-Reordering unit (BFDR)

The DFG computes most of the butterflies and QCMs in the first $\log_2 N - 2$ stages. Since the design is serial, the half-butterfly units in $\log_2 N - 1$ stage are always operational in every clock cycle. In contrast, only one butterfly computation is required in the last stage. The proposed design does not include half-butterfly unit in the last stage, rather it uses a composite half-butterfly and DR unit. As soon as the computations in stage 3 are completed, further processing is performed through the composite half-butterfly and DR unit.

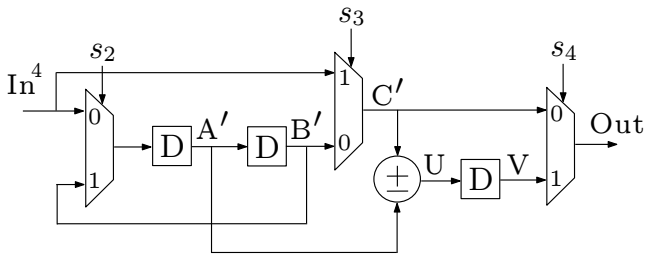


**Fig. 2.7.** Proposed butterfly cum data-reordering (BFDR) unit.

Table 2.3: Timing Diagram of Proposed BFDR unit (Fig. 2.7) in Stage 4

| **Clk** | **In**$^4$ | $s_2$ | **A′** | **B′** | $s_3$ | **C′** | **U** | **V** | $s_4$ | **Out** |
|---|---|---|---|---|---|---|---|---|---|---|
| 37 | $x_0^3$ | 0 | - | - | 0 | - | - | - | - | - |
| 38 | $x_1^3$ | 0 | $x_0^3$ | - | 1 | $x_1^3$ | $x_0^4$ | - | 1 | - |
| 39 | $x_{4r}^3$ | 0 | $x_1^3$ | $x_0^3$ | 0 | $x_0^3$ | $x_1^4$ | $x_0^4$ | 1 | $x_0^4$ |
| 40 | $x_{4i}^3$ | 0 | $x_{4r}^3$ | $x_1^3$ | 0 | $x_1^3$ | - | $x_1^4$ | 1 | $x_1^4$ |
| 41 | $x_{8r}^3$ | 1 | $x_{4i}^3$ | $x_{4r}^3$ | 1 | $x_{8r}^3$ | - | - | 0 | $x_{8r}^4$ |
| 42 | $x_{8i}^3$ | 1 | $x_{4r}^3$ | $x_{4i}^3$ | 1 | $x_{8i}^3$ | - | - | 0 | $x_{8i}^4$ |
| 43 | $x_{11r}^3$ | 0 | $x_{4i}^3$ | $x_{4r}^3$ | 0 | $x_{4r}^3$ | - | - | 0 | $x_{4r}^4$ |
| 44 | $x_{11i}^3$ | 0 | $x_{11r}^3$ | $x_{4i}^3$ | 0 | $x_{4i}^3$ | - | - | 0 | $x_{4i}^4$ |
| 45 | $x_2^3$ | 0 | $x_{11i}^3$ | $x_{11r}^3$ | 0 | $x_{11r}^3$ | - | - | 0 | $x_{11r}^4$ |
| 46 | $x_3^3$ | 0 | $x_2^3$ | $x_{11i}^3$ | 0 | $x_{11i}^3$ | - | - | 0 | $x_{11i}^4$ |
| 47 | $x_{5r}^3$ | 0 | $x_3^3$ | $x_2^3$ | 0 | $x_2^3$ | - | - | 0 | $x_2^4$ |

As mentioned, the last stage requires only a single butterfly computation while rest of the samples are already computed in the previous stage. Hence, to avoid underutilization of the half-butterfly unit in the last stage and using additional DR units to obtain a normal order output after the butterfly computation in the last stage, a new merged unit for butterfly and DR is proposed. It is used to perform any one of the following three operations in one clock cycle: 1) a half-butterfly computation, 2) an interchange of samples separated by two clock cycles or 3) pass a sample through the buffer.

The proposed BFDR unit is shown in Fig. 2.7. It includes one real adder, three multiplexers and three registers. To understand the operation of the proposed BFDR, a separate timing diagram is shown in Table 2.3. Consider the proposed DFG scheme for a 16-point FFT, as shown in Fig. 2.1. In stage 4, after two stages of DR, the samples with indices 0 and 1 are fed into the BFDR unit while the rest of the samples undergo a final stage of DR to obtain a normal order FFT output. It is interesting to note that in the proposed architecture, the output DR operation is performed by the architecture itself without any dedicated DR circuit. The architecture of BFDR has the advantage of

fewer registers over using individual butterfly and post-processing DR units. The following algorithm explains the operation of proposed serial pipelined FFT.

**Algorithm 2.1** Explains the proposed serial-pipelined FFT

**Input:** $In^1$: Input to DR in stage 1; $X, A, B, C, E$: Internal nodes of butterfly unit; $F, G, H, I, J, K, M, Y$: Internal nodes of QCM (Fig. 2.6); $In^4, A', B', C', U'$: Internal nodes of BFDR (Fig. 2.7); $s, s_0, s_1, s_2, s_3, s_4$: Control signals

```
for i = 0 to log2N−1 do
    DR if s = 0 then
        X = In^1;
    else
        X = D_L{In^1};
    end
    BF if s_0 = 1 then
        E = C + D_1{X};
    else
        E = D_2{X} − D_1{X};
    end
    QCM if s_0 = 0 then
        G = D_1{E}, Y = D_2{E};
    else
        G = E, Y = M, M = J ± D_1 J; J = Icos(α) or Isin(α);
    end
    if s_1 = 1 then
        I = D_1{G};
    else
        I = G;
    end
end
for i = log2N − 2 do
    DR repeat from 5 to 7
    BF repeat from 10 to 12
end
for i = log2N − 1 do
    DR repeat from 5 to 7
     BFDR if s_2 = s_3 = 0 then
        C' = D_2{In^4};
    else if s_3 = 1 then
        C' = In^4;
    end
    if s_4 = 0 then
        Out = C';
    else
        Out = D_1{U}, U = C' ± A', A' = D_1{In^4} or D_1{D_1{A'}}
    end
end
```

### 2.2.3 Performance Comparison

In this section, the computational complexities of different designs are discussed. Next, the performance of proposed design along with the existing designs is evaluated through application specific integrated circuit (ASIC) and field programmable gate array (FPGA) platforms in terms of area, power, minimum-clock-period (MCP), area-delay product (ADP), energy, slice LUTs (SLUT) and flip-flops (FF). For comparison, the existing designs referred from the respective references are as follows: RSC in [2], R2SDF and R2$^2$SDF in [1], R2$^3$SDF in [16], R2SDF H (Hybrid) and R2SDF FR (Fully Real) in [17], and SC in [61].

### 2.2.4 Computational Complexities

#### 2.2.4.1 Hardware Complexity

In this work, the focus was to reduce the overall hardware complexity in serial-pipelined FFT. One possible solution is to reduce the complexity of the multiplier or to enhance the resource utilization of the half-butterfly units. By employing the BFDR, the proposed design achieves minor reduction of two registers when focusing only on hardware reduction due to the BFDR. However, it maximizes the resource utilization in the last stage compared to RSC-FFT design. Since this is a serial architecture, in traditional case, the butterfly unit in the last stage would remain mostly idle after the only butterfly in that stage is computed i.e. after two clock cycles and we need to have a bypass path for rest of the output to move forward to the post-processing stage in order to obtain a normal-order output. The hardware complexities of the proposed and existing serial-pipelined FFT designs [1, 2, 16, 17, 61] are listed in Table 2.4. It can be observed that the complexity of registers and computational workload on butterflies of the proposed design are less as compared to the best existing RSC-FFT design with the same number of multipliers involved. The main difference between RSC-FFT and the proposed design lies in the architecture of the multipliers directly impacting the hardware complexities.

To determine the total number of registers in the proposed design, one has to find the registers in input DR ($\gamma_{DR,I}^{R,P}$), intermediate DR ($\gamma_{DR,IM}^{R,P}$), output DR ($\gamma_{DR,O}^{R,P}$), half-butterfly unit ($\gamma_{BF}^{R,P}$), QCM unit ($\gamma_{QCM}^{R,P}$) and BFDR unit ($\gamma_{BFDR}^{R,P}$). For $N$-point FFT, the number of registers in the proposed design for input DR is $\gamma_{DR,I}^{R,P} = (\sqrt{N}-1)^2$ when $n$ is even, or $\gamma_{DR,I}^{R,P} = (\sqrt{2N}-1)(\sqrt{N/2}-1)$ when $n$ is odd [80]. The number of registers in the intermediate DR is $\gamma_{DR,IM}^{R,P} = \log_2 N - 2 + 3 + \sum_{s=3}^{n-1}(2^s - 1)$. The number of registers in the output DR (excluding the reordering registers in BFDR) is $\gamma_{DR,O}^{R,P} = N - 7$. The number of registers in half-butterfly units is $\gamma_{BF}^{R} = 2\log_2 N - 2$. The number of registers in

QCM unit is $\gamma_{QCM}^{R} = 5\log_2 N - 10$. The number of registers in BFDR unit is $\gamma_{BFDR}^{R} = 3$. Therefore, the total number of registers involved in the proposed design can be estimated as $\gamma_{T}^{R,P} = \gamma_{DR,I}^{R,P} + \gamma_{DR,IM}^{R,P} + \gamma_{DR,O}^{R,P} + \gamma_{BF}^{R,P} + \gamma_{QCM}^{R,P} + \gamma_{BFDR}^{R,P}$. Clearly, the total number of registers $\gamma_{T}^{R,P}$ depends on the value of $n$, whether the number of FFT points is equal to even or odd powers of 2. If $n$ is even then total number of registers for the proposed design can be estimated as $\gamma_{T}^{R,P} = (\sqrt{N} - 1)^2 + 7\log_2 N + 2N - 20$. In similar manner, one can compute the number of registers when number of FFT points is equal to odd power of 2. It can be noted that the total number of registers for the proposed design is $2\log_2 N - 4$ less than the existing RSC-FFT. Furthermore, the number of registers in the proposed design is $N/4 - 7\log_2 N + 13$ and $N/2 - 7\log_2 N + 10$ less compared to R2SDF H and R2SDF FR respectively. The percentage reduction in the number of registers ($\Delta\gamma_{T}^{R,P}$) of the proposed design with respect to RSC-FFT when $n$ is even is calculated as

$$\Delta\gamma_{T}^{R,P} = \frac{\gamma_{T}^{R,E} - \gamma_{T}^{R,P}}{\gamma_{T}^{R,E}} = \frac{2\log_2 N - 4}{(\sqrt{N} - 1)^2 + 9\log_2 N + 2N - 24} \tag{2.5}$$

Similarly, the percentage reduction in the number of registers can be estimated when $n$ is odd. Further, the complexity of multiplexers for the proposed architecture is relatively less as compared to RSC-FFT. For the proposed design, the total number of multiplexers is $12\log_2 N - 21$, while it is $14\log_2 N - 24$ in RSC-FFT. Therefore, the difference in multiplexers between the designs is $2\log_2 N - 3$. Interestingly, the reductions in number of registers and multiplexers further increases for higher FFT points.

In terms of number of real multipliers, the proposed serial-pipelined FFT has the same as that of RSC-FFT but lower than that of R2SDF H and R2SDF FR. However, the number of registers in the multiplier is lower compared to RSC-FFT. In contrast with the proposed multiplier, there are two multiplexers and two registers after the real multiplier in the multiplier unit in the RSC-FFT resulting in more area and power consumption. The number of adders is the same for RSC-FFT and the proposed FFT. However, R2SDF H and R2SDF FR has more hardware as they use full butterfly unit (two adders). The number of registers and multiplexers involved in data reordering units varies depending on the stage. Hence, it would not be appropriate to compare in terms of the number of data reordering units. The proposed design employs a butterfly-cum-data-reordering unit (BFDR) which performs the output data-reordering alongwith the last stage butterfly computation and does not require an external circuit. Hence, reduces the total hardware complexity.

In summary, the proposed serial-pipelined FFT has an advantage of less hardware resources with

respect to existing RSC-FFT for obtaining a normal order FFT. On the other hand, R2SDF H, R2SDF FR and RSC-FFT designs process real-valued data, however, they have higher implementation cost over the proposed architecture.

#### 2.2.4.2 Time Complexity

The proposed design is serial in nature as the existing designs reported in [1, 2, 16, 17, 61]. It implies that the throughput is simply one sample per clock cycle, while the latency is $N$ clock cycles respectively. The duration of clock is basically decided by the maximum register-to-register delay in the serial path of FFT computation. For the proposed design, the maximum register-to-register delay for 16-point FFT is $2T_{MA} + 16T_{MUX} + 6T_A$ while it is $2T_{MA} + 19T_{MUX} + 6T_A$ in [2], where $T_{MA}$, $T_{MUX}$ and $T_A$ are delays of multiply-add unit, 2-to-1 multiplexer and 16-bit adder. The maximum register-to-register delay in these architectures is dominated by the computational delay of QCM, data-reordering units and associated multiplexers. In the proposed design, this value is slightly less than the existing RSC-FFT design as fewer number of multiplexers are encountered in register-to-register delay of the proposed design.

Table 2.4: Comparison of Computational Complexities of Different Serial Pipelined FFTs

| **Design** | **Real Multipliers** | **Real Adders** | **Registers** | **CW** | **TP** |
|---|---|---|---|---|---|
| Complex-valued | | | | | |
| R2SDF [1] | $4\log_2 N - 8$ | $6\log_2 N - 4$ | $2N$ | - | 1 |
| R$2^2$SDF [1] | $2\log_2 N - 4$ | $5\log_2 N - 2$ | $2N$ | - | 1 |
| R$2^3$SDF [16] | $4/3\log_2 N - 2$ | $14/3\log_2 N - 2$ | $2N$ | - | 1 |
| SC [61] | $2\log_2 N - 4$ | $3\log_2 N - 2$ | $2N$ | - | 1 |
| Real-valued | | | | | |
| R2SDF H [17] | $4\log_2 N - 12$ | $6\log_2 N - 10$ | $5N/4 - 2$ | - | 1 |
| R2SDF FR [17] | $4\log_2 N - 12$ | $4\log_2 N - 6$ | $3N/2 - 5$ | - | 1 |
| RSC [2] | $\log_2 N - 2$ | $2\log_2 N - 2$ | $9\log_2 N + 2N - 24$[a] | $N(\log_2 N + 1) - 6$ | 1 |
| Proposed[b] | $\log_2 N - 2$ | $2\log_2 N - 2$ | $7\log_2 N + 2N - 20$ | $N(\log_2 N - 1) + 2$ | 1 |

[a]: For fair comparison, the $N - 5$ registers in output reordering units are included in addition to the reported $N + 9\log_2 N - 19$ registers. [b]: Complexity of post-multiplication register in QCM is half of that in RSC. $N$: Number of FFT points, CW: Computational workload on half-BF units, SDF: Single-path delay feedback, SDC: Single-path delay commutator, TP : Throughput. Note the latency of all the designs is $N$ clock cycles.

### 2.2.5 Performance Comparison

#### 2.2.5.1 Application Specific Integrated Circuit (ASIC)

The proposed design along with existing designs are coded in Verilog for 512 and 1024 FFT points with 8-bits. ASIC synthesis is performed using Taiwan semiconductor manufacturing company limited (TSMC) 90 nm complementary metal oxide semiconductor (CMOS) Library [81] by Cadence RTL Compiler to estimate area, power and throughput performances. The corresponding results obtained from the tool are listed in Table 2.5. Further, two figures of merit, area-delay product (ADP) and energy are defined to determine the energy efficiency of different designs for fair comparison.

The proposed design for 8-bit, 1024-point FFT occupies 33.48% and 10.26% less area over the existing R2SDF FR and RSC-FFT respectively. The main reason is that the number of registers per stage is reduced with only one register after the real multiplication in QCM unit over the existing RSC-FFT. Moreover, last-stage employs a BFDR circuit for final output reordering, unlike a post processing stage in RSC-FFT. The results of power consumption of the proposed design are more significant than area results, due to the fact that the reduction in number of registers has direct impact on the power consumption. Also, part of this reduction in power consumption is due to the reduced complexity of multiplexers as discussed in section 4.3.1.

The reduction in power consumption can be explained via the following governing equation in digital circuit as

$$P_{\text{total}} = C_L V_{DD} f^2 + I_{SC} V_{DD} + I_L V_{DD} \tag{2.6}$$

where $P_{\text{total}}$, $C_L$, $V_{DD}$, $f$, $I_{SC}$ and $I_L$ are, respectively, total power, switching capacitance, supply voltage, operating frequency, short circuit current and leakage current. The first term in (2.6) is of great importance as it entails switching (or dynamic) power. The proposed architecture has less number of computational nodes over the existing RSC-FFT design. As a result, the value of switching capacitance $C_L$ decreases [82]. Moreover, there is another term associated with switching capacitance popularly known as switching activity which is lower for the proposed design. Since, the number of transitions in the architecture is reduced due to less number of multiplexers, the dynamic power consumption of the proposed design is significantly reduced [83]. Further, the proposed design employs BFDR unit in the last stage, therefore post-processing circuit for DR is reduced in the computation of a normal I/O FFT. Thus, the power consumption of the proposed design is lower as compared to the existing RSC-FFT [2].

Table 2.5: Performance Comparison of Different Serial Pipelined FFT Architectures

| **Design** | $N$ | **ASIC** | | | | | **FPGA** | |
|---|---|---|---|---|---|---|---|---|
| | | **Area** ($\mu m^2$) | **Power** (mW) | **MCP** (ns) | **ADP** ($\mu m^2 \times$ ns) | **Energy** (mW $\times$ ns) | **SLUT** ($\times$ 1000) | **FF** ($\times$ 1000) |
| R2SDF [1] | 512 | 1602212 | 17.54 | 4.81 | 7706639.72 | 84.37 | 116.82 | 29.62 |
| | 1024 | 1957642 | 19.22 | 5.05 | 9886092.10 | 97.06 | 148.12 | 37.83 |
| $R2^2SDF$ [1] | 512 | 910187 | 10.38 | 4.10 | 3731766.70 | 42.56 | 65.28 | 16.95 |
| | 1024 | 1131105 | 13.12 | 4.47 | 5056039.35 | 58.65 | 83.21 | 21.04 |
| $R2^3SDF$ [16] | 512 | 798978 | 8.86 | 3.98 | 3179932.44 | 35.26 | 58.19 | 14.87 |
| | 1024 | 905263 | 10.11 | 4.24 | 3838315.12 | 42.87 | 66.04 | 16.02 |
| R2SDF H [17] | 512 | 742321 | 8.32 | 3.81 | 2828243.01 | 31.69 | 54.67 | 13.92 |
| | 1024 | 863516 | 9.16 | 4.12 | 3557685.92 | 37.74 | 63.67 | 15.85 |
| R2SDF FR [17] | 512 | 675603 | 7.41 | 3.66 | 2472706.98 | 27.12 | 49.55 | 12.34 |
| | 1024 | 757267 | 8.38 | 3.97 | 3006349.99 | 33.27 | 55.12 | 14.25 |
| RSC [2] | 512 | 513139 | 5.96 | 3.21 | 1647176.19 | 19.13 | 37.02 | 9.59 |
| | 1024 | 561243 | 6.72 | 3.47 | 1947513.21 | 23.32 | 40.03 | 10.21 |
| Proposed | 512 | 473561 | 4.87 | 3.02 | 1430154.22 | 14.71 | 34.62 | 8.74 |
| | 1024 | 503673 | 5.32 | 3.21 | 1616790.33 | 17.08 | 36.91 | 9.52 |

LEGEND: MCP: minimum-clock-period, ADP: area-delay product, SLUT: sliced look-up tables, FF: flip-flops.

The minimum clock period (MCP), which is a measure of critical path, is less for the proposed architecture compared to RSC-FFT, in accordance with their time-complexities. For instance, the difference in critical path of the proposed and existing RSC designs for 16-point FFT is $3T_{MUX}$ which increases with the number of FFT points. As expected, the synthesized value of proposed design is less as compared to RSC-FFT for both 512 and 1024 FFT points. For example, the proposed design for 8-bit, 1024-points approximately consumes 36.51% less power with 46.22% less ADP and 48.66% less energy as compared to R2SDF FR, while it consumes 20.83% less power with 16.98% less ADP and 26.76% less energy as compared to RSC-FFT. A comparison of these serial-pipelined architectures in terms of ADP and Energy is shown in Fig. 2.8.

#### 2.2.5.2 Field Programmable Gate Array (FPGA)

The performances of the proposed and existing designs are also evaluated by implementing on Xilinx ZYNQ-XC7Z020-1CLG484C FPGA device. It has block random access memory (block RAM), 6-input distributed RAMs and flip-flops (FF). Distributed RAMs are arranged in groups of four to make a slice (referred as SLUT). According to hardware description language (HDL) primitive [84], the hardware units of different designs are mapped onto the SLUT and FF logic elements. Adders and multipliers of different designs are also mapped to several slices with each containing up to four 6-input distributed RAMs. All the designs are implemented on FPGA device for both 512 and 1024 FFT-points with 8-bits. The corresponding SLUT and FF results are listed in Table 2.5 by setting the system clock at 50 MHz. Clearly, the proposed design occupies less number of SLUT and FF over all the other existing designs. For example, the proposed design for 8-bit, 512-points has 6.48% less SLUT and 8.86% less FF over the best existing design [2]. While the proposed design for 8-bit, 1024-points has almost 7.79% less SLUT and 11.93% less FF over best existing design [2].

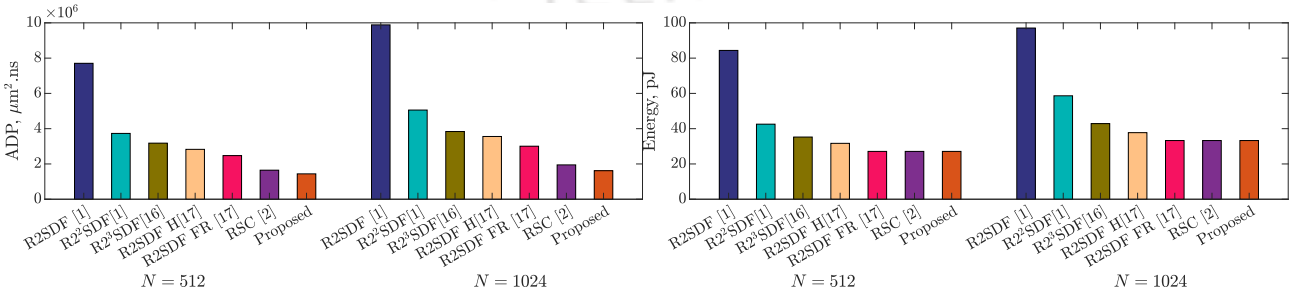


**Fig. 2.8.** Performance Comparison (a) Area-Delay-Product (ADP) (b) Energy of Different Serial Pipelined FFT Architectures.

## 2.3 Proposed Methodology 2

One of the main concerns related to FFT architectures is the hardware complexity of the multipliers. The size of the multiplier limits the performance of the FFT architectures. Another concern is the switching activity of the TF which is proportional to the number of TF multiplications. To reduce the number of TF multiplications, higher radices such as radix-4, 8, or even higher are sometimes employed. The radix-$2^2$ algorithm is often preferred over these algorithms owing to its simpler radix-2 implementation, low switching activity, and the same TF multiplicative complexity as radix-4. Radices higher than 4 are not preferred as the reduction in the number of multiplication is offset by the increase in the number of additions. Moreover, the TF multiplications in a radix-$2^2$ FFT appear after every two radix-2 stages compared to radix-2 FFT. However, the direct mapping of radix-$2^2$ RFFT algorithm to serial architecture results in underutilization of butterfly (BF) units in all the stages except in stage 1. Moreover, the resultant serial RFFT architecture is similar to a CFFT architecture, and fails to take advantage of the properties of the RFFT in which almost 50% of the output samples are redundant.

To maximize the utilization of the BF unit, the radix-2 RFFT DFG presented in [41] could be used. However, it would require a multiplier with variable coefficients at every multiplier stage of the serial FFT architecture. In other words, although the RFFT decreases the number of computations compared to CFFT, the aforementioned DFGs either result in an underutilized architecture or require more complex multiplier units in the serial FFT compared to that of CFFT.

To address the aforementioned issues, a radix-2 based DFG is proposed based on the TF transformation strategy in Fig. 2.9. It lessens the number of multiplier stages with variable coefficients and reduces both the switching activity and resource usage. The proposed DFG, as shown in Fig. 2.10, depicts that if the same TF is present on the upper and lower input paths of a BF, it is equivalent to performing the TF multiplication after the BF computation and vice-versa. For instance, $W^n$ is common to both the upper and lower paths of the BF. It can be shifted and the TF multiplication

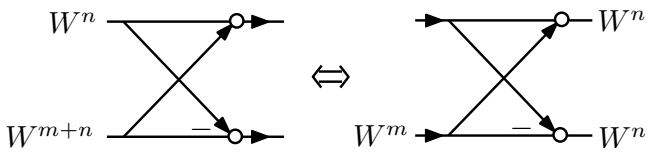


**Fig. 2.9.** Twiddle factor (TF) transformation strategy.

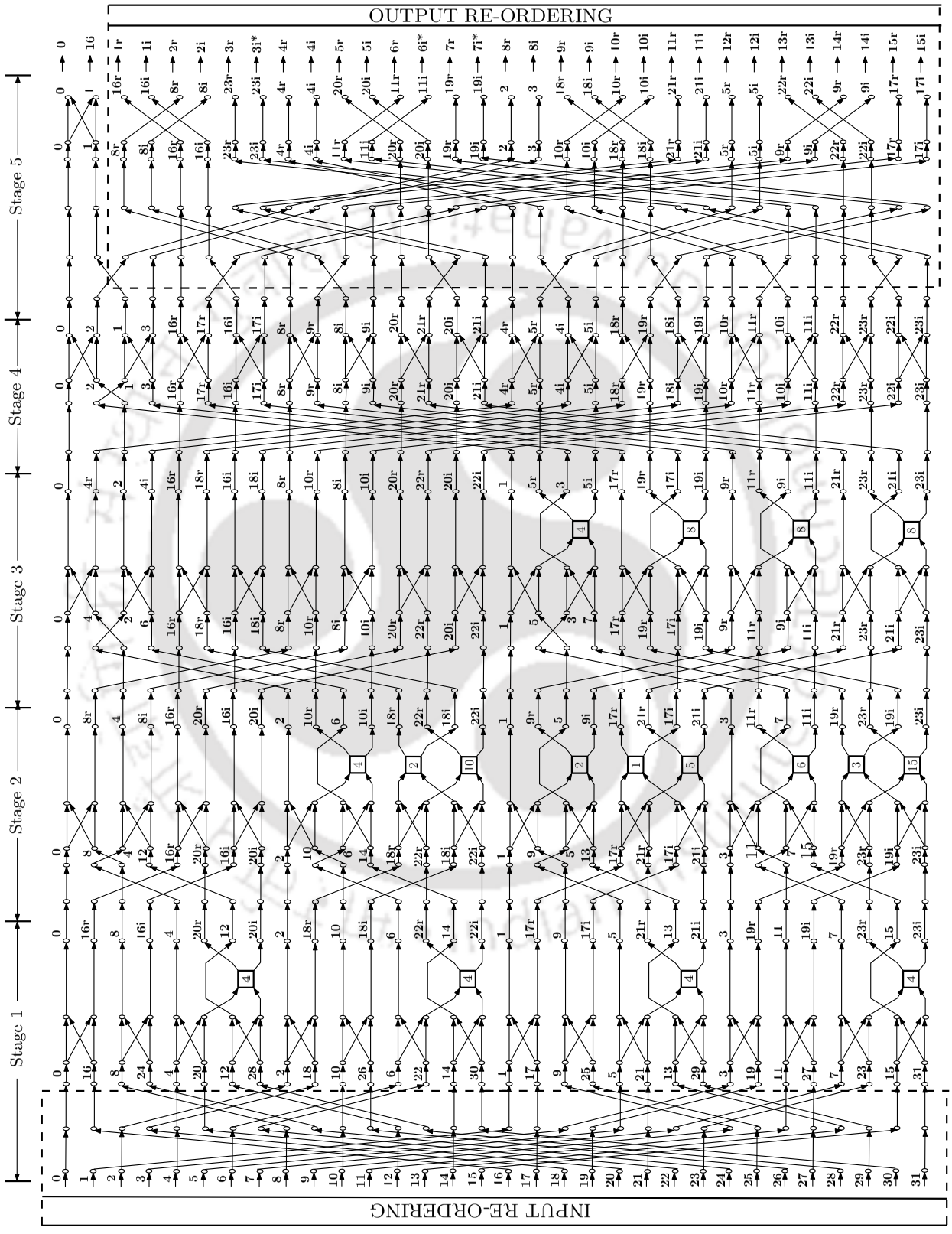


**Fig. 2.10.** Proposed DFG for 32-point RFFT.

can be performed after the BF computation. The $W^m$ that remains is either a trivial multiplication if $m = 0, N/4, N/2, 3N/4$ or a constant non-trivial multiplication for an entire stage. The trivial multiplications (multiplications by -1, 1, $j$, $-j$) can be performed by sign alteration and/or by exchanging real and imaginary parts while the constant non-trivial multiplication for a stage can be implemented by a constant digit multiplier.

In the proposed DFG for 32-point RFFT, the TF $W^{1-3}$ are shifted from stage 1 towards the right of the stage 2 BF leaving the constant TF $W^4$ at the lower half of the BF input. Thus, stage 1 consists of only the non-trivial constant TF $W^4$. Similarly, stage 3 requires only one $W^4$. It can be seen that only stage 2 has a different TF and the rest of the stages have a constant TF. Hence, only one multiplier stage with variable coefficients is required in a 32-point serial RFFT.

### 2.3.1 Data Flow Graph (DFG)

The proposed DFG for serial computation of a 32-point RFFT with normal order input-output is shown in Fig. 2.10. It facilitates in understanding the flow of the input and the intermediate data through the FFT processor. The proposed DFG requires cascaded data-reordering (DR) circuits with latency 15 and 6 in stage 1 to reorder the input, while the output DR begins in stage 5 before the computation of the last BF. In stage 1 butterfly computation, '1 ($u_4u_3u_2u_1u_0 = 00001$)' needs to be interchanged with '16 ($u_4u_3u_2u_1u_0 = 10000$)' so that '0' and '16' are received together for the first butterfly computation. Further, '2 ($u_3u_2u_1u_0 = 0010$)' needs to be interchanged with '8 ($u_3u_2u_1u_0 = 1000$)' and '3 ($u_4u_3u_2u_1u_0 = 00011$)' needs to be interchanged with '24 ($u_4u_3u_2u_1u_0 = 11000$)', so that '8' and '24' are received together for the second butterfly computation. In all of these cases, it is observed that elementary bit-exchanges $u_4 \leftrightarrow u_0$ and $u_3 \leftrightarrow u_1$ are required. The number of registers required to perform this DR is calculated as

$$D_0 = 2^4 - 2^0 = 15 \tag{2.7}$$
$$D_1 = 2^3 - 2^1 = 6$$

Hence, we need two cascaded DR units with latency 15 + 6 connected one after the other to achieve the aforementioned order.

The BF operations and the multiplications appear together in all the stages, except the last two stages. Thus, the proposed serial architecture requires a total of $\log_2 N - 2$ multipliers. It is to be noted that one output of each BF in two consecutive BFs is processed by the same multiplier. One

multiplier output corresponds to the real part while the other output corresponds to the imaginary part of a complex TF multiplication. The computation of these two BFs is performed in successive clock cycles as one output from each BF has to be processed through the same multiplier. In stage 4, only DR and BF operations are performed. Note that most of the arithmetic computations are completed within stage 4. However, there is one more BF computation left to be processed. This allows us to utilize stage 5 for output reordering along with BF computation using the same unit.

### 2.3.2 Proposed RFFT Architecture

The architecture for 32-point serial RFFT is shown in Fig. 2.11. It consists of eleven DR units, four BF units, and three multiplier units and a combined BF-cum-DR unit. There are three stages of TF multiplication. In general, the number of TF multiplication stages for the computation of $N$-point serial RFFT is $\log_2 N - 2$. The DR units and the BF units employed are identical to [2]. The detailed explanation of each of these units is described below.

The DR unit is required to map the data from the previous stage to the next stage based on the shuffling requirement shown in Fig. 2.10. It is to be noted that the DR circuit does not perform any computation. Fig. 2.10 shows how the data is permuted in each stage. The DR units consist of two or three 2-to-1 multiplexers separated by '$L$' number of buffers as shown in Fig. 2.11. Depending upon the control signal of the two multiplexers in the DR unit, the sample is either passed through the buffer or interchanged with the sample '$L$' clock cycles apart. This is employed in stages 1-4. On the other hand, the DR unit with three multiplexers in stage 5 can interchange two samples that are apart at a distance equal to the length of the first buffer or the total of the first and second buffer. The length of the buffer is stage-dependent. The BF computation in Fig. 2.10 is computed by the BF unit in Fig. 2.12(a). The BF unit consists of two registers, one multiplexer, and an adder/subtractor. It carries out the addition and then subtraction in the subsequent clock cycle.

In the last stage, a combined unit for one BF computation and DR is employed to perform any one of the operations: a BF addition/subtraction, an interchange of samples two clock cycles apart, or passing a sample through the buffer. As shown in Fig. 2.12(b), it consists of three registers, three multiplexers, and an adder/subtractor.

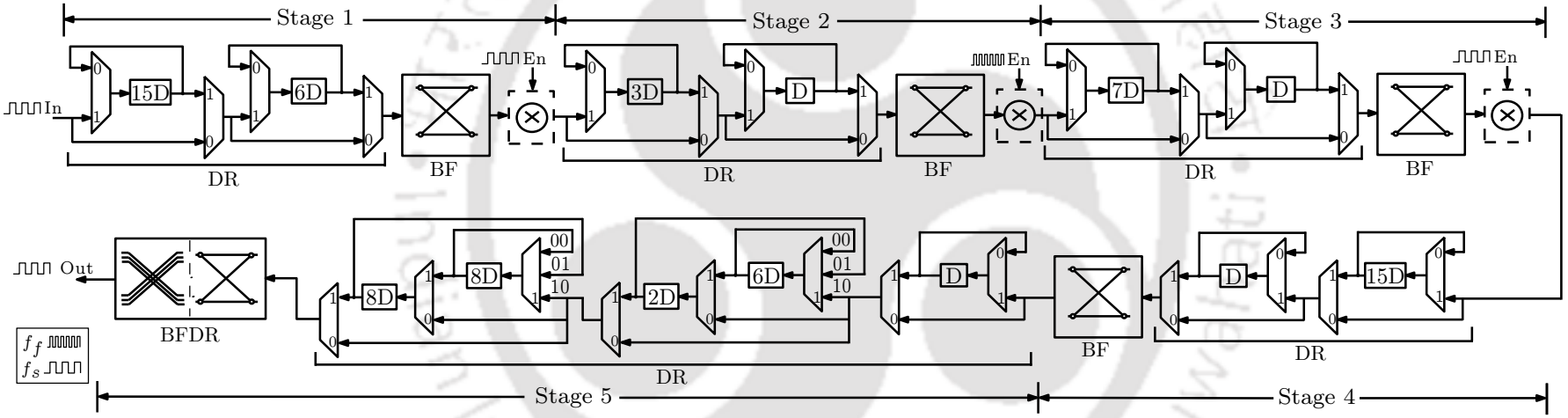


**Fig. 2.11.** Proposed 32-point serial RFFT architecture. En: Clock enable, BF: Butterfly unit, DR: Data reordering unit

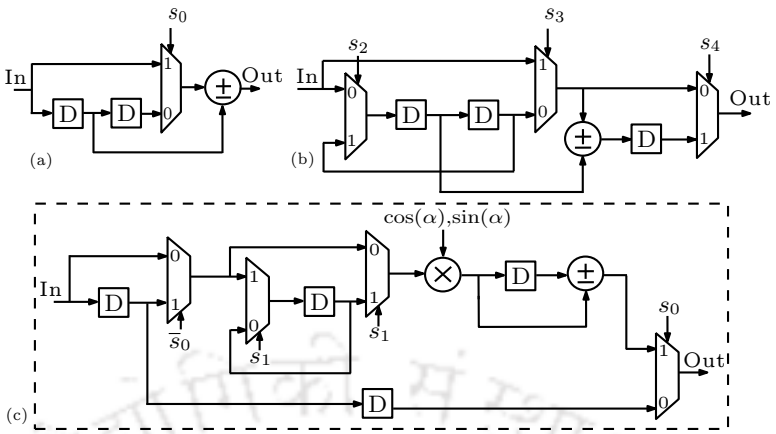


**Fig. 2.12.** (a) BF unit. (b) Last stage BF-cum-data-reordering unit. (c) Multiplier unit

The details of the proposed multiplier to perform the multiplications with different TF are explained in the next section. From the DFG in Fig. 2.10, it is clear that the BF units are used in every clock cycle. However, the multipliers need to operate only when there is a TF multiplication to be performed. Hence, a clock enable signal is assigned to turn off the components conditionally. These multiplier units run on a faster clock to reduce the dynamic power consumption [83, 85].

### 2.3.3 Rationale of Faster Clock and Gating in Multiplier Unit

Consider a digital circuit running on a sample clock $f_s$ having total capacitance $C$ and operating at $V_{DD}$ power supply. One of its sections, let's say, is running on a faster clock $f_f$ having capacitance $C_f$ and operating at $\gamma V_{DD}$ power supply, where $\gamma > 0$ is a scale factor. The scale factor $\gamma$ is applicable when employing two clock frequencies to establish the relationship between them and minimize power consumption. For analysis, it is assumed that the digital circuit and its section have the same switching activity $\alpha$. Therefore, the dynamic power consumption of the digital circuit can be expressed as

$$P = \alpha C_f \gamma^2 V_{DD}^2 f_f + \alpha C V_{DD}^2 f_s \tag{2.8}$$

This can be compared with the same digital circuit running only at sample clock $f_s$. One can find an upper bound on $\gamma$ to lower the power consumption as $\gamma < \sqrt{f_s/f_f}$. Since the proposed multiplier is based on shift accumulation, it can be run at a faster clock for each sample clock to perform a complete TF multiplication. Depending on the number of bits processed in the proposed multiplier, $f_s$ and $f_f$ can be appropriately chosen. Since the proposed multiplier is based on shift accumulation, $f_f$ can be decided to complete the multiplication within the duration of the clock $f_s$. For instance,

in $W$-bit multiplication with $B$-bit processed at a time, $f_s$ and $f_f$ can be related as $f_f = \lceil W/B \rceil \ f_s$ i.e., $W = 8$ and $B = 2$ would lead to $f_f = 4f_s$, where $\lceil \cdot \rceil$ is the least-integer function. In general, the upper bound on $\gamma$ in such a case can be simplified as $\gamma < \sqrt{\lceil B/W \rceil}$. The power consumption in digital circuits having two clocks can be reduced, if $\gamma$ is considered in the range of $0 < \gamma < \sqrt{\lceil B/W \rceil}$. Thus, it is always desirable to design a system with a faster clock in some section of the digital circuit.

In the proposed architecture, we employ clock gating [86] corresponding to $f_s$ to reduce power consumption in the multiplier unit. This would decrease $\alpha$ by selectively removing the clock from some sections of the circuit when not in use. It can be noted from (2.8) that all the parameters except $\alpha$ depend on the technology and/or performance. The clock gating is generally applied when outputs do not change. It is clear from Fig. 2.10 that in every stage inputs are either passed or multiplied with a TF. Therefore, clock gating could be applied to the multiplier when bypassing the inputs to the output of the next stage using $f_s$, and thereby it allows to reduce the power consumption.

### 2.3.4 Fixed-Point Design Consideration

We present an efficient fixed-point (FP) quantization strategy by analyzing different wordlengths across the BF structure of a serial RFFT. We assume the unfolded BF unit to show the wordlengths at different intermediate nodes and to analyze the structure. Moreover, we consider the output quantization as an additive noise on both of its branches, as shown in Fig. 2.13. Let $(X_i^p, X_f^p)$ and $(Y_i^p, Y_f^p)$ be the FP representations of the input and output at the $p$th-stage respectively, where indices $i$ and $f$ denote the integer and fractional parts. The complex-valued quantization noise present at the BF input is denoted by $\epsilon_p$, while the corresponding error due to round-off operation is denoted by

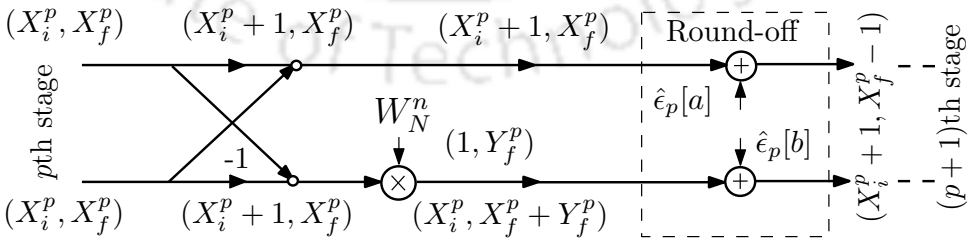


**Fig. 2.13.** Quantization noise model of radix-2 BF structure.

$\hat{\epsilon}_{p+1}$. Therefore, the resultant quantization noise ($\epsilon_{p+1}$) at $(p+1)$th stage BF output is given as:

$$\epsilon_{p+1}[u] = \epsilon_p[u] + \epsilon_p[v] + \hat{\epsilon}_{p+1}[u] \tag{2.9a}$$

$$\epsilon_{p+1}[v] = (\epsilon_p[u] - \epsilon_p[v])W_N^n + \hat{\epsilon}_{p+1}[u] \tag{2.9b}$$

where the initial values $\epsilon_0[u]$ and $\epsilon_0[v]$ are set as zero and fed to the input of proposed RFFT. Here, the variables $u$ and $v$ represents the two inputs to a butterfly. The relation between $u$, $v$ and the quantization noise model in Fig. 2.13 is that the variable $(X_i^p, X_f^p)$ depicts the fixed-point representation of the butterfly inputs $u$ and $v$. For the analysis to be tractable, we follow the independent assumptions [63] as:

- At every FFT stage, the BF input quantization noise ($\epsilon_p$) is independent of each other.
- At every FFT stage, the added quantization noise by the BF is independent of each other.
- The round-off error ($\hat{\epsilon}_{p+1}$) is independent of input quantization noise ($\epsilon_p$).
- The mean value of $\hat{\epsilon}_{p+1}$ is assumed to be zero. Otherwise, the biases can propagate FFT architecture and lead to the FFT output bin dependent on bias which is not desirable.

Based on the above assumptions, the mean square error (MSE) of the resultant quantization noise, $\epsilon_{p+1}$ can be given as

$$\xi_{p+1}[u] = \xi_p[u] + \xi_p[v] + \hat{\xi}_{p+1}[u] \tag{2.10a}$$

$$\xi_{p+1}[v] = \xi_p[u] + \xi_p[v] + \hat{\xi}_{p+1}[v] \tag{2.10b}$$

where $\xi_{p+1}$ is the MSE of $\epsilon_{p+1}$ and $\hat{\xi}_{p+1}$ is the MSE of $\hat{\epsilon}_{p+1}$. In more accurate form, (2.10a) and (2.10b) can be expressed as a summation:

$$\xi_{p+1}[u] = \sum_{i=0}^{p-2} 2^{p-i}(\hat{\xi}_i[u] + \hat{\xi}_i[v]) + \hat{\xi}_{p+1}[u] \tag{2.11a}$$

$$\xi_{p+1}[v] = \sum_{i=0}^{p-2} 2^{p-i}(\hat{\xi}_i[u] + \hat{\xi}_i[v]) + \hat{\xi}_{p+1}[v] \tag{2.11b}$$

The fractional bits ($X_f^p$) used at the BF output decide the distribution of ($\hat{\epsilon}_{p+1}$), which impacts the system performance, while the integer bits ($X_i^p$) at the BF output decide the saturation. We need to grow the integer bit at most by 1 from BF input to BF output (i.e., $X_i^p + 1$) to prevent BF output from saturation. The proposed quantization strategy does not allow the growing of ($\log_2 N$)-bits at the

FFT output by dropping 1 least-significant bit (LSB) per stage at every BF computation. This would minimize the impact of quantization error for large point FFT and make the same precision for the inputs and outputs in every FFT stage. Fig. 2.13 shows the BF structure for this output quantization strategy. It can be noted that only 1 LSB of the upper adder leg is rounded-off for output $\epsilon_{p+1}[u]$, and $(Y_f^p+1)$ LSBs of the lower adder leg are rounded-off for output $\epsilon_{p+1}[v]$. To make the quantization zero mean, we employ the unbiased rounding scheme to avoid the quantization noise bias problem [87]. The MSE of $\hat{\xi}_{p+1}[u]$ in such a case is $\Delta^2$, where $\Delta$ is the level of input LSB of BF. In contrast, the MSE of $\hat{\xi}_{p+1}[v]$ is $4\Delta^2/6$ as $(Y_f^p+1)$ LSBs are dropped for the $(p+1)$th stage output.

$$\hat{\xi}_{p+1}[u] = 4^{p-1} \times 6\sigma^2, \quad 1 \leq p \leq \log_2 N \tag{2.12a}$$

$$\hat{\xi}_{p+1}[v] = 4^{p}\sigma^2, \quad 1 \leq m \leq \log_2 N \tag{2.12b}$$

where $\sigma^2$ is the input quantization noise of serial RFFT. It can be noted from Fig. 2.10 that the last two stages either require trivial TF multiplication or no TF multiplication. This implies that we have to quantize only 1 LSB at the BF output. The MSE of $\hat{\xi}_{p+1}[u]$ and $\hat{\xi}_{p+1}[v]$ are given by

$$\hat{\xi}_{p+1}[u] = 4^{p-1} \times 6\sigma^2, \quad 1 \leq p \leq \log_2 N \tag{2.13a}$$

$$\hat{\xi}_{p+1}[v] = \begin{cases} 4^{p}\sigma^2, & 1 \leq p \leq \log_2 N - 2 \\ 4^{p-1} \times 6\sigma^2, & \log_2 N - 1 \leq p \leq \log_2 N \end{cases} \tag{2.13b}$$

The total MSE due to the quantization noise at the FFT output can be estimated by using (2.13a) and (2.13b) as $\hat{\xi}[u] = \hat{\xi}[v] \approx \frac{23}{8} N^2\sigma^2$. Observably, the $\hat{\xi}[u]$ and $\hat{\xi}[v]$ are equal.

### 2.3.5 Proposed Multiplier

Consider the product $c = ab$ with $a$ and $b$ are considered to be $W$-bits operands. In two's complement form, $b$ in terms of adjacent $B$-bits can be expressed as

$$b = -b_{k_0-1} + \sum_{k=0}^{k_0-2} b_k 2^{-k} \tag{2.14}$$

where $k_0 = \lceil W/B \rceil$ denotes the number of digits, $b_k \in [0, 1, ..., 2^B - 1]$ with $B$ as the number of bits to represent $b_k$ $\forall$ $0 \leq k \leq k_0 - 1$, and $\lceil \cdot \rceil$ is the least-integer function. Each input-digit $b_k$ can be expressed in $B$-adjacent bits $w_{B(k+1)-l}$ as

$$b_k = \sum_{l=1}^{B} w_{B(k+1)-l} 2^{-(B-l)} \tag{2.15}$$

where $1 \leq l \leq B$. The product $c$ can be then expressed as

$$c = -c_{k_0-1} + \sum_{k=0}^{k_0-2} c_k 2^{-k} \tag{2.16}$$

where $c_k = ab_k$. By expressing (2.15) in terms of LSB, we get

$$b_k = 2\sum_{l=1}^{B-1} w_{B(k+1)-l} 2^{-(B-l)} + w_{Bk} \tag{2.17}$$

Clearly, $c_k$ can be written in term of $w_{Bk}$ as $c_{k,w_{Bk}} = a(2\sum_{l=1}^{B-1} w_{B(k+1)-l}2^{-(B-l)} + w_{Bk})$. It can be noted that $w_{Bk}$ either assumes '0' or '1' logic value. Therefore, $c_{k,w_{Bk}}$ would result in two separate sets of partial products i.e., $c_{k,0} \in \{0, 2a, ..., (2^B - 2)a\}$ and $c_{k,1} \in \{a, 3a, ..., (2^B - 1)a\}$ denote the even and odd partial product sets respectively. By combining $c_{k,1}$ and $c_{k,0}$ in (2.16), we obtain

$$c_{k,0} = c_{k,1} - a \tag{2.18}$$

Clearly, $c_{k,0}$ can be obtained from $c_{k,1}$ by subtracting $a$. For $W = 3$, $c_{k,0}$ and $c_{k,1}$ would assume the value from sets $c_{k,0} \in \{0, +2a, +4a, +6a\}$ and $c_{k,1} \in \{+a, +3a, +5a, +7a\}$. POT representation simplifies the logic for realization of $c_{k,1}$, for instance, the POT form of $c_{k,1}$ can be given as $c_{k,1} \in \{+(2^1-1)a, +(2^2-1)a, +(2^2+1)a, +(2^3-1)a\}$. It is interesting to note that the above representation allows the use of operand $a$ and its single POT version to produce all the elements of $c_{k,1}$. In contrast, when $W \geq 4$, the above representation cannot be used since multiple POT terms would be needed e.g., $+11a = (2^3 + 2 + 1)a$ and $+13a = (2^3 + 2^2 + 1)a$. However, an offset of $2^{B-1}a$, if subtracted/added from/to $c_{k,1}$ would lead to symmetries in the elements. For instance, an offset $+8a$ for $B = 4$ if subtracted/added from/to $c_{k,1}$ would result in symmetric $c^{(1)}_{k,1}$. The symmetric $c^{(1)}_{k,1}$ would assume the values from a set $\{-7a, -5a, -3a, -a, +a, +3a, +5a, +7a\}$. Only the first half needs to be generated as the other half is mirror-symmetric, which can be obtained by sign-inversion logic. Based on the above explanation, $c_{k,1}$ in terms of $c^{(1)}_{k,1}$ using (2.17) can be written as

$$\begin{aligned} c_{k,1} &= [2\sum_{l=1}^{B-1} w_{B(k+1)-l} 2^{-(B-l)} \pm 2^{B-1}]a \\ &= w_{B-1} \oplus c^{(1)}_k + 2^{B-1}a \end{aligned} \tag{2.19}$$

where '$\oplus$' is the bit-wise XOR operator; $c^{(1)}_k = [2\sum_{l=2}^{B-1} w^{(1)}_{B(k+1)-l} 2^{-(B-l)} + 1 - 2^{B-1}]a$ and $w^{(1)}_{B(k+1)-l} = w_{B-1} \oplus w_{B(k+1)-l}$.

Unlike $c_{k,1}$, the even partial-products $c_{k,0}$ possess pseudo-symmetries in their elements. For in-

stance, an offset $+8a$ is subtracted from $c_{k,0}$ using (2.18) would result pseudo-symmetric $c_{k,0}^{(1)} \in \{-7a-a=-8a, -5a-a=-6a, -3a-a=-4a, -a-a=-2a, +a-a=0, +3a-a=+2a, +5a-a=+4a, +7a-a=+6a\}$. The pseudo-symmetries in $c_{k,0}$ arise due to different magnitudes with '$-$' and '$+$' signs in the first half and second half respectively. In general, (2.19) can be expressed as

$$c_k = w_{B-1} \oplus c_k^{(1)} + w_{Bk}a + 2^{B-1}a = c_k^{(0)} + c_{ot}^{(0)} \tag{2.20}$$

where $c_k^{(0)} = w_{B-1} \oplus c_k^{(1)} + w_{Bk}a$ and $c_{ot}^{(0)} = 2^{B-1}a$ is the offset-term. Substituting (2.20) in (2.16), we have

$$c = -c_{k_0-1}^{(0)} + \sum_{k=0}^{k_0-2} c_k^{(0)} 2^{-k} + c_{ot}^{(0)} \tag{2.21}$$

where $c_{ot}^{(0)} = -2^{B-1}a + 2^{B-1}a \sum_{k=0}^{k_0-2} 2^{-k} = -2^{B-1}a/2^{k_0-1}$ denotes the offset term.

Following is the POT analysis of odd partial-products $c_{k,1}^{(1)}$ and even partial-products $c_{k,0}^{(1)}$ for $W = B = 4$ with one order symmetries. As listed in Table 2.6, the $c_{k,1}^{(1)}$ (Column I) are obtained by converting $c_{k,1}$ using (2.19). For instance, $c_{k,1}^{(1)}$ (Column II) would assume the values from the set $\{-7a, -5a, -3a, -a, +a, +3a, +5a, +7a\}$ with each element being expressed in POT form as $\{\pm(2^3-1)a, \pm(2^2+1)a, \pm(2^2-1)a, \pm(2^1-1)a\}$. Likewise, $c_{k,0}^{(1)}$ (Column III) can be similarly expressed as $c_{k,1}^{(1)}$. However, they are different from the elements of $c_{k,1}^{(1)}$ as either $2a$ or $0$ is to be subtracted from POT, as per (2.18). As mentioned previously, the number of elements in $c_{k,1}^{(1)}$ and $c_{k,0}^{(1)}$ would increase for large $B$. Further, these cannot be restricted to the same set with $B = 4$, and therefore cannot be expressed in POT form. This can be overcome by exploiting inherent recursive symmetries in $c_{k,1}^{(1)}$ for $B > 4$ at the expense of a more complex offset term. For instance, $c_{k,1}$ with first order symmetry for $B = 5$ correspond to $c_{k,1}^{(1)} \in \{-15a, -13a, -11a, ..., +11a, +13a, +15a\}$. Clearly, $\pm 13a$ and $\pm 11a$ cannot be expressed in the above representation. However, it is observed that the upper-half of $c_{k,1}^{(1)}$ for $B = 5$ corresponds to $c_{k,1}^{(1)}$ for $B = 4$. Therefore, it allows the exploitation of the recursive symmetries in $c_{k,1}$ which makes their elements to be in the same set as with $B = 4$. In general, if symmetries are exploited with an order of recursion $r$, then (2.21) can be expressed as

$$c = -c_{k_0-1}^{(r-1)} + \sum_{k=0}^{k_0-2} c_k^{(r-1)} 2^{-k} + c_{ot}^{(r-1)} \tag{2.22}$$

where $c_k^{(r-1)} = [2\sum_{l=2}^{B-(r-1)} w_{B(k+1)-l}^{(r)} 2^{B-(r-1)-l} + 1 - \sum_{s=2}^{r} w_{B-s}^{(s-1)} 2^{B-s}a - 2^{B-1}]a$; $w_{B(k+1)-l}^{(r)} = w_{B-1}^{(r-1)} \oplus w_{B(k+1)-l}^{(r-1)}$; $c_{ot}^{(r-1)} = c_{ot}^{(0)}(\sum_{s=2}^{r} 2^{B-s} w_{B-s}a + 2^{B-1})$. A large number of adjacent $B$-bits can reduce $k_0$ for a given wordlength $W$ since $k_0 = \lceil W/B \rceil$. By successively exploiting the symmetries in

Table 2.6: POT Analysis of $c_{k,1}^{(1)}$ and $c_{k,0}^{(1)}$ with First Order Symmetry

| $c_{k,1}^{(1)}$ | $c_{k,1}^{(1)}$ in POT | $c_{k,0}^{(1)}$ | $c_{k,0}^{(1)}$ in POT |
|---|---|---|---|
| $-7a$ | $-(2^3-1)a$ | $-8a$ | $-(2^3-1)a-a=-2^3a$ |
| $-5a$ | $-(2^2+1)a$ | $-6a$ | $-(2^2+1)a-a=-2^2a-2^1a$ |
| $-3a$ | $-(2^2-1)a$ | $-4a$ | $-(2^2-1)a-a=-2^2a$ |
| $-a$ | $-(2^1-1)a$ | $-2a$ | $-(2^1-1)a-a=-2^1a$ |
| $+a$ | $+(2^1-1)a$ | $0$ | $+(2^1-1)a-a=+2^1a-2^1a$ |
| $+3a$ | $+(2^2-1)a$ | $+2a$ | $+(2^2-1)a-a=+2^2a-2^1a$ |
| $+5a$ | $+(2^2+1)a$ | $+4a$ | $+(2^2+1)a-a=+2^2a$ |
| $+7a$ | $+(2^3-1)a$ | $+6a$ | $+(2^3-1)a-a=+2^3a-2^1a$ |

Assuming $W=B=4$. No SA unit case.

$c_{k,1}$ would allow to select a higher $B$. Interestingly, the offset term needs to be modified, as per (2.22). For instance, the elements of the offset term $c_{ot}^{(2)}$ for $W=18$, $B=6$ and $r=3$ in terms of $c_{ot}^{(0)}$ becomes $c_{ot}^{(2)} \in \{+c_{ot}^{(0)}, +3c_{ot}^{(0)}, +5c_{ot}^{(0)}, +7c_{ot}^{(0)}\}$. Clearly, the elements of $c_{k,1}^{2}$ and $c_{ot}^{(2)}$, except $-a$ is replaced with $c_{ot}^{(0)}$. It is interesting to note that as long as $B-r \leq 3$ and $B \leq 6$ hold the elements can be restricted to the set $B=4$, and thus can be represented in single POT form. Any $W$-bit proposed multiplier can be realized by successive shift-and-accumulation of partial products over $k_0$ clock cycles.

The top-level schematic of the proposed multiplier with a partial-product generator (PPG), sign-logic (SL), and shift-accumulate (SA) unit for $B=4$ and $r=1$ is shown in Fig. 2.14. The PPG generates the partial products, SL performs signed multiplication with $w_s$ as a sign-bit, and the SA unit operates on the PPG output for $k_0$ clock cycles to produce $c$. The control signal $S_0$ acts as a select line for a 2-to-1 multiplexer to load $c_{ot}^{(0)}$ into the SA unit during the initial clock cycle. It is worth noting that $c_{ot}^{(0)}$ is the scaled version of $c_{ot}^{(0)}$ by $1/2^{k_0-1}$. Therefore, $c_{ot}^{(0)}$ can be loaded into the SA unit by providing the shift on $-a$. The PPG details for $W=B=4$ and $r=1$ are shown in Fig. 2.15. It consists of XOR gates, a 4-to-1 multiplexer, two 2-to-1 multiplexers, one conditional adder, and an AND gate with one inverted input. As listed in Table 2.6, the elements of $c_{k,1}^{(1)}$ i.e., $\pm 5a$ and $\pm 3a$ have the same POT term $-2^2a$, except that $-a$ is added with $-2^2a$ to obtain $-5a$, while it is subtracted from $-2^2a$ to obtain $-3a$. Thus, all the elements of $c_{k,1}^{(1)}$ can be generated using a conditional adder and a 4-to-1 multiplexer. The output of XOR gates $d_1$ and $d_2$ are the select lines of a 4-to-1 multiplexer. Another XOR gate is used after the conditional adder for sign-inversion logic to exploit mirror symmetry. The conditional adder performs either addition or subtraction ($A/\overline{S}$) to compute the $c_{k,1}^{(1)}$ as $-(2^3-1)a$, $-(2^2+1)a$, $-(2^2-1)a$ and $-(2^1-1)a$. From the Boolean simplification, the signal

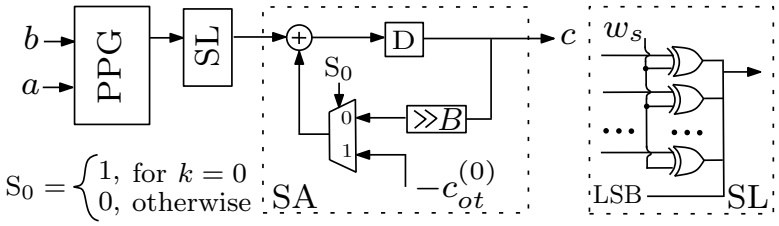


**Fig. 2.14.** Top-level schematic of the proposed multiplier for $r = 1$.

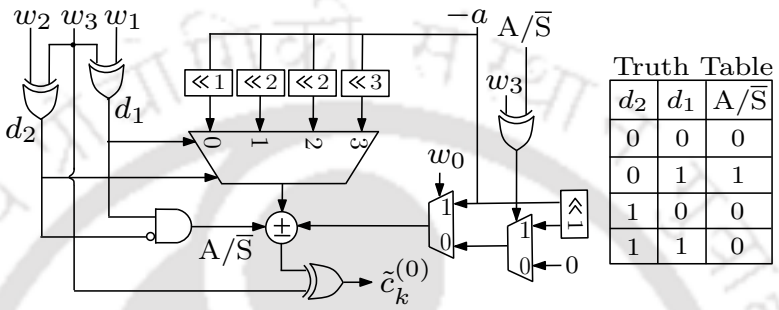


**Fig. 2.15.** PPG details for $B = 4$ and $r = 1$.

A/S can be generated using an AND gate with inverted input as shown in Fig. 2.15. Interestingly, the same logic could be used to obtain $c_{k,0}^{(1)}$ except in place of $-a$, either $-2a$ or 0 needs to be selected, as listed in Table 2.6. It is achieved with two 2-to-1 multiplexers in series. The first 2-to-1 multiplexer selects either $-2a$ or 0 to compute $c_{k,0}^{(1)}$ with select line as XOR of A/S and $w_3$. The XOR gate is used to obtain the lower-half combinations of $c_{k,0}^{(1)}$ as $-2a$ is present in the lower-half combinations except for $+4a$ element, while it is absent in the upper half combinations except for $-6a$ element. While second 2-to-1 multiplexer selects either $-a$ for $c_{k,1}^{(1)}$ or the output of first 2-to-1 multiplexer with $w_0$ as its select line. Note that the left shifts at the inputs of $2^{B-r-1}$-to-1 multiplexer causes its size to increase rapidly with $B$. Alternatively, these left shifts can also be realized with optimization on barrel-shifter (BS). The left shifts $2^1, 2^2, 2^2$ and $2^3$ at the input of multiplexer are divisible by $2^1$. These can be realized with pre-shifting of $a$ by 1, as shown in Fig. 2.15. By doing so, one additional level needed to implement the above shifts using BS is reduced. Thus, the remaining left shifts $2^0, 2^1, 2^1$, and $2^2$ can be realized with optimized BS using two-level multiplexers. Note that the select lines $d_1d_2$ are translated to $d_1'd_2'$ as it has a different order of shifts as compared to the conventional BS. From the Boolean simplification, the new select lines $d_1'd_2'$ can be obtained from $d_1d_2$ as shown in Fig. 2.16. Clearly, if $d_1'd_2' = 01$, then the output of optimized BS is the left-shifted version of input by 2.

So far, $a$ and $b$ have been considered unsigned binary numbers, they can be signed binary numbers

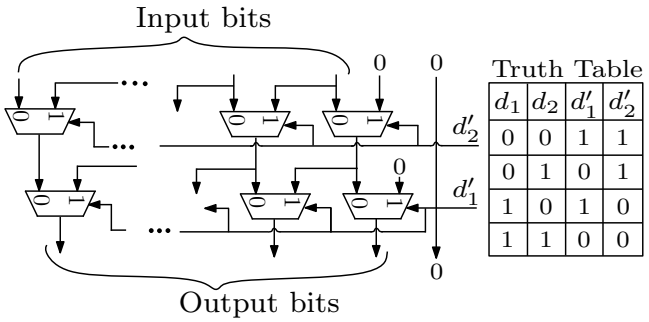


**Fig. 2.16.** Optimized barrel-shifter for $B = 4$.

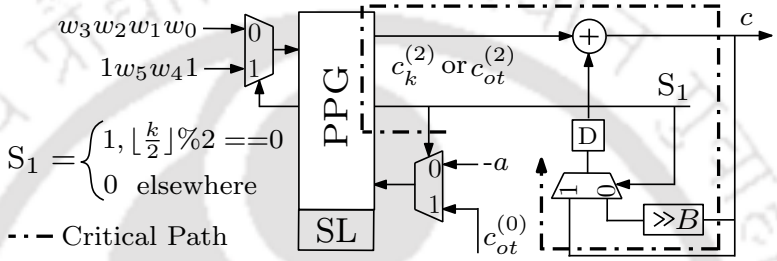


**Fig. 2.17.** Modified SA unit for $B = 6$ and $r = 3$.

as well. To find the signed product $c$, an additional bit ($w_s$) of $b$ is used to indicate the sign. From Table 2.6, it is observed that the elements of $c_{k,1}^{(1)}$ are odd. To obtain the signed product $c$, it is required to know whether $a$ is odd or even. If the operand $a$ is odd, it implies the elements of $c_{k,1}^{(1)}$ will also be odd and two's complement of an odd binary number can be obtained by taking one's complement (as shown in Fig. 2.14) except the LSB based on $w_s$. When $a$ is even, a similar argument holds good since it can be expressed as $a = a_{\text{odd}} \times$ POT, where $a_{\text{odd}}$ is the odd component of $a$.

For $B = 6$ and $r = 3$, it is first required to compute $c_{ot}^{(2)}$ and then load it into SA unit. Observably, the elements of $c_{ot}^{(r-1)}$ are similar to positive half of $c_{k,1}^{(2)}$ except their inputs. Hence, there is a possibility to share the same PPG for their computation using time-multiplexing. This is possible to achieve by relocating the adders and registers of the SA unit. The relocation can be done by first knowing the properties of the SA unit:

(i) Output is produced after one clock cycle with initial content of register as '0'.

(ii) Adder is located before the register.

(iii) $c_{ot}^{(0)}$ is loaded into SA unit during initial clock cycle.

Based on (i-iii), the SA unit is modified as shown in Fig. 2.17. According to (i) and (ii), we can

relocate the register before the adder since the output is produced after one clock cycle. The PPG first computes $c_k^{(2)}$ with address lines as the $w_3w_2w_1w_0$ and input as $-a$ in the first-half of a clock cycle using select line $S_1 = 0$. The computed $c_k^{(2)}$ would be then loaded to the register of the SA unit in the second-half of a clock cycle. At the same time, the PPG computes $c_{ot}^{(2)}$ via 2-to-1 multiplexers due to its similar elements as $c_k^{(2)}$. The select lines to the optimized BS and XOR gates are set as $1w_5w_41$, where MSB and LSB of address lines are kept as 1 since $c_{ot}^{(2)}$ possess all positive odd-multiples in terms of $c_{ot}^{(0)}$ using $S_1 = 1$. In the second-half of clock cycle, both $c_k^{(2)}$ and $c_{ot}^{(2)}$ are added followed by the right-shift by $B$. It is then available to be added with $c_k^{(2)}$ for the next first-half of a clock cycle, and so on until all the digits $b_k$ are consumed during the SA operation. It is clear that as long as $B \leq 6$, the proposed multiplier can be realized with two adders, one register, and some combinational logic. The design can be extended for $B > 6$ by employing the wordlength decomposition on $B$ such that $B = 6B_0$, where $B_0 \leq 1$ when $B \leq 6$ and $B_0 > 1$ when $B > 6$. It is interesting to note that the multiplier design with $B > 6$ results in additional clock cycles $B/6$ in shift accumulation. Algorithm 2.2 explains the operation of the proposed SA-based multiplier with POT representation.

**Algorithm 2.2** SA-based with POT Multiplier Algorithm

**Input:** Operands $a, b$; wordlength $W$; $\{w_{B(k+1)-l}\}_{l=1}^{B}$ adjacent $B$-bits of $b$, symmetry order $r$; $k_0 = \lceil W/B \rceil$; $B = 6B_0$; $B_0 = 1$ ($\Rightarrow B = 6$) and $B_0 > 1$ ($\Rightarrow B > 6$); $k_0 = \lceil W/(6B_0) \rceil$; $c_{ot}^{(0)} = -a >> |B - k_0|$; $p_1, p_2 \mapsto [1, \log_2 B]$

**Output:** Product $c$

Initialization:**for** $k \leftarrow 0$ **to** $k_0 - 1$ **do**
  **assign** $p_1, p_2 \leftarrow w_{Bk-B+1} \ldots w_{Bk-1} w_{Bk}$
  **assign** $c_{ot}^{(r-1)} \leftarrow (2^{p_2} \pm 1) c_{ot}^{(0)}$
  **assign** $c_{k,1}^{(r-1)} \leftarrow \mp(2^{p_1} \pm 1)a$ **if** $k == 0$ **then**
    $c \leftarrow c_{ot}^{(r-1)}$
  **else if** $k == k_0 - 1$ **then**
    $c \leftarrow -c$
  **else**
    $c \leftarrow c + c_k^{(r-1)} >> B$
  **end**
**end**

### 2.3.6 Performance Comparison

For sake of clarity in discussion, the existing designs [2, 17, 43, 44] are referred to as $\text{FFT}_{0-3}$.

#### 2.3.6.1 Computational Complexities

The hardware complexities of different architectures in terms of adders, multipliers, registers, and multiplexers are listed in Table 2.7. It is clear that $FFT_0$ has more number of multipliers/adders, but possesses less number of registers. In serial $FFT_1$, the number of multipliers/adders is reduced at the expense of an increase in the number of registers over $FFT_0$. In $FFT_{2-3}$, efforts have been made to trade off between the number of registers and adders; while having the same multipliers as that of $FFT_1$. Unlike $FFT_{0-3}$, the proposed design employs an SA-based multiplier in place of multipliers by processing $B$-bits at a time. The simultaneous use of the POT scheme and shared PPG unit simplifies the implementation complexity of the proposed multiplier. For instance, it requires two adders ($B \leq 6$) and a register with a few multiplexers and logic gates per TF multiplication stage. On the other hand, to reduce the adders, for $B > 6$, the multiplication can be performed with more clock cycles. This is especially advantageous for low-power FFT since the PPG complexity in traditional SA-based multiplier grows linearly with higher number of SA clock cycles corresponding to higher power consumption and vice-versa. For clarity, the complexity of the proposed SA-based multiplier per stage TF multiplication is expressed in terms of the number of adders ($A$), registers ($R$), and multiplexers ($M$). In addition, it requires a few XOR, AND, and NOT gates whose complexities are listed in the footnote of Table 2.7. The proposed design supports normal input-output operation which none of these architectures have reported. The output reordering is accommodated in the last stage of FFT processing which utilizes the reordering registers to perform the last BF computation, thereby reducing the overall complexity of registers required to implement the proposed serial FFT as compared to $FFT_{0-3}$.

The time complexities of the proposed and $FFT_{0-3}$ designs are listed in terms of critical path delay (CPD) and latency in Table 2.7. The throughput of each serial RFFT architecture is one sample per clock cycle. The clock duration of each design can be decided by the CPD. In general, the CPD is predominantly influenced by the multiplier. All the other designs employ conventional multipliers whereas the proposed design employs SA-based multiplier. It requires $k_0$ faster clock cycles to perform the SA-based multiplication, hence its effective CPD becomes $k_0$ times as compared to $FFT_{0-3}$ designs. Unlike the other designs, the estimated CPD of the faster clock in the proposed design has a delay of two adders, a few multiplexers, and a XOR gate but without any conventional multiplier. The latency of the proposed architecture is higher compared to $FFT_2$ and $FFT_3$. This is due to the use of shift-

Table 2.7: Comparison of Computational Complexities of Different Serial $N$-Point RFFT Architectures

| **Design** | **Adders** | **Multipliers** | **Registers**$^{\dagger}$ | **Multiplexers** | **CPD** | **Latency**$^{\dagger}$ |
|---|---|---|---|---|---|---|
| $FFT_0$ [17] | $4\log_2 N - 6$ | $4\log_2 N - 12$ | $5N/2 - 10$ | $12\log_2 N - 13$ | $T_M + T_A + T_{DMX} + 2T_{MX}$ | $2N + 5\log_2 N - 19$ |
| $FFT_1$ [2] | $2\log_2 N - 2$ | $\log_2 N - 2$ | $2N + 9\log_2 N - 24$ | $10\log_2 N - 14$ | $T_M + T_A + 9T_{MX}$ | $2N + 3\log_2 N - 13$ |
| $FFT_2$ [43] | $2\log_2 N - 2$ | $\log_2 N - 2$ | $2N + 5\log_2 N - 14$ | $7\log_2 N - 12$ | $T_M + 2T_A + 4T_{MX}$ | $2N + 2\log_2 N - 10$ |
| $FFT_3$ [44] | $2\log_2 N - 3 + Y/W$ | $\log_2 N - 2$ | $2N + 5\log_2 N - 12$ | $8\log_2 N - 9$ | $T_M + T_A + 4T_{MX}$ | $2N + 2\log_2 N - 9$ |
| Proposed | $2\log_2 N - 2 + A$ | 0 | $2N + 6\log_2 N - 18 + R$ | $9\log_2 N - 2 + MX$ | $2k_0(2T_A + 4T_{MX} + T_X)^{\dagger\dagger}$ | $2N + 3\log_2 N - 14$ |

CPD: Critical Path Delay; $^{\dagger}$ includes $(N-5)$ additional register/clock cycles for output reordering in each design; $^{\dagger\dagger}$ adders, multiplexers and XOR gate are of B-bit; $A = 2N_M$; $R = N_M$; $MX = (4 + ((B+2)/2W))N_M$; $N_M = \log_2 N - 2$; $k_0 = \lceil W/B \rceil$; $Y = \min(W/K)$, $K = 1, ...., (N-2)/2$. In addition, the proposed design requires $(((r-1)W + (W-1) - r(r-1)/2)/W)N_M$ XOR, $N_M$ NOT and $N_M$ AND gates.

accumulate (SA) multiplier.

In addition, the multiplier in all the other designs process $W$-bit data whereas the SA-based multiplier in the proposed design processes $\lceil W/B \rceil$-bit data. Despite a slightly higher CPD compared to $\text{FFT}_3$, the proposed design outperforms $\text{FFT}_{0-2}$ in terms of the area-time trade-off. This is considered acceptable, as the primary focus of this work is on achieving low complexity and hardware efficiency. Noticeably, the CPD of the proposed design also involves a factor of 2 which is due to the shared PPG unit corresponding to $-a$ and $c_{ot}^{(0)}$. In the proposed design, due to the BFDR unit in the last stage, the latency for output reordering decreases to $N-7$ instead of the usual $N-5$. As listed in Table 2.7 for fair comparison, the output reordering is considered in all the architectures when estimating the latency. A comparison of all these serial RFFT architectures is shown in Fig. 2.19.

#### 2.3.6.2 FPGA and ASIC Synthesis

The proposed design was implemented on a Xilinx FPGA VC709 XC7VX690TFFG1761-2 using the FPGA methodology shown in Fig. 2.18. The Verilog code was imported into the Xilinx Vivado tool, translating them into a netlist in Xilinx netlist format (.xnf). The simulator then performed functional verification through random binary test vectors. The mapping was done to translate the netlist on the available resources of the FPGA at 450 MHz clock frequency. Note that the proposed design was synthesized by setting "Vivado Synthesis Defaults" and implemented by setting "Performance_Retiming". After this step, the place&route (PnR) process was performed at specific locations of the FPGA device with the available routing resources. The placement of LUTs and flip-flops were done by the tool with no constraints, and they were not constrained to the proposed design footprint. Subsequently, the delay information corresponding to interconnections can be used to obtain a more accurate netlist through timing analysis. A bitstream file was generated for each design, which was then transferred to the Xilinx VC709 XC7VX690TFFG1761-2. The resource utilization of the different designs for $N = 1024$, $W = 16$ (with $B = 6$, $r = 3 \Rightarrow k_0 = 3$) in terms of look-up tables (LUT), flip flops (FF), DSP, block RAM (BRAM) and latency are listed in Table 2.8. Compared to $\text{FFT}_{1-3}$, the number of adders, registers and multiplexers required in the proposed architecture is higher. However, the increase in the number of these modules is small compared to the amount of DSPs that are saved which leads to reduced area and power consumption in the proposed architecture. The designs in $\text{FFT}_{0-3}$ utilize large number of DSPs to realize the multipliers, whereas the proposed design has the advantage of not using any DSP as it employs an SA-based multiplier instead of a real

multiplier.

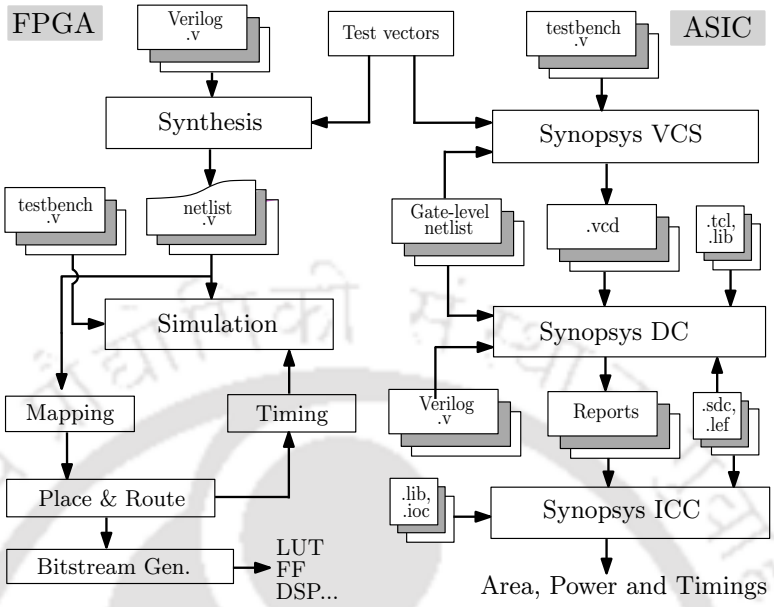


**Fig. 2.18.** ASIC and FPGA synthesis methodology for the proposed design.

ASIC synthesis methodology to extract area, power, and timings is shown in Fig. 2.18. The Synopsys Design Compiler (DC) was used to obtain the gate-level netlist and the standard delay format (.sdf) files of the proposed design, with input as 1) Verilog (.v) file, 2) TSMC 45 nm library (.lib), 3) Synopsys design constraint (.sdc) file; and 4) Technology library exchange format (.lef) and macro files. The slow-slow process corner with voltage = 1.0 V and temperature = 125°C was used. The proposed design was validated and its fed to the Synopsys VCS. Subsequently, a value change dump (.vcd) file was generated to estimate the switching activity. Based on this, the tool then performed the synthesis using the Synopsys DC to generate the area, power, and timing reports. These reports were then used in Synopsys IC Compiler to perform the floorplan and PnR. From the implementation, it was found that, during the PnR step, routing congestion could not be avoided when minimizing the area with low driving-strength standard cells. This was overcome by tweaking the Verilog codes, by using incremental spacing between the logic blocks. During the floorplan, constraints such as die size and I/O pin locations (using input-output constraint (.ioc) format) were also taken into consideration. After a few attempts, the proposed design was made routable. The corresponding results for $N = 1024$ are listed in Table 2.8. Due to the digit-serial nature of the proposed multiplier, it offers a substantial amount of savings in area and power as compared to the existing designs. For instance, the proposed architecture occupies 27.27% less area and consumes 36.25% less power as compared to $\text{FFT}_1$. A

Table 2.8: Performance Comparison of Different 1024-point RFFT Architectures

| PF | Parameter | $\mathbf{FFT}_0$ [17] | $\mathbf{FFT}_1$ [2] | $\mathbf{FFT}_3$ [44] | Proposed |
|---|---|---|---|---|---|
| FPGA | **Device** | V7 | V7 | V7 | V7 |
| | **Frequency** | 450 MHz | 450 MHz | 450 MHz | 450 MHz |
| | **LUT** | 3239 | 2390 | 2043 | 1892 |
| | **FF** | 4860 | 3471 | 3382 | 2962 |
| | **DSP** | 28 | 8 | 8 | 0 |
| | **BRAM** | – | – | 2 | 2 |
| | **Latency** (in c.c.) | 2079 | 2065 | 2059 | 2064 |
| ASIC | **Tech.** (nm) | TSMC 45 | TSMC 45 | – | TSMC 45 |
| | **Frequency** | 100 MHz | 100 MHz | 100 MHz | 100 MHz |
| | **Voltage** (V) | 1.0 | 1.0 | – | 1.0 |
| | **Area** ($mm^2$) | 0.284 (*0.33) | 0.22 | – | 0.16 |
| | **Power** (mW) | 14.80 (*4.5) | 3.20 | – | 2.04 |

**PF**: Platform; $*$: As reported in [2]; **LUT**: Look-Up Table, **FF**: Flip-Flops, **BRAM**: Block RAM

comparison of these serial RFFT architectures in terms of area and power is shown in Fig. 2.19.

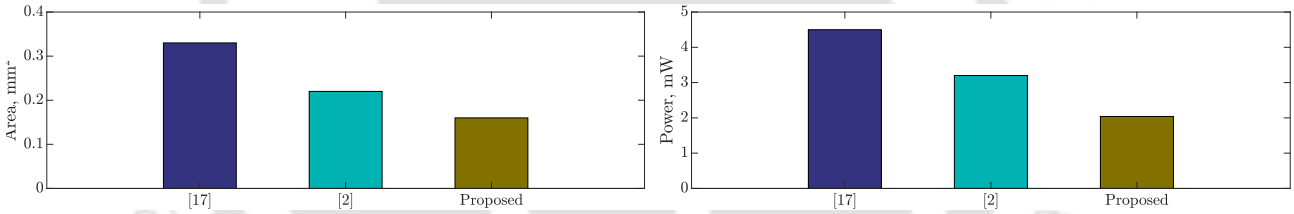


**Fig. 2.19.** Performance Comparison (a) Area (b) Power of 1024-point Serial RFFT Architectures

## 2.4 Conclusion

In this Chapter, initially an energy-efficient serial pipelined architecture of FFT to process real-valued signals has been presented. The proposed DFG has eliminated the use of a post-processing stage for FFT computation in normal I/O order and reduced the computational workload on the half-butterfly units which results in significant savings in power consumption for large number of FFT points. The proposed design involves a novel QCM with relatively lower hardware complexity. Furthermore, the proposed QCM and the composite half-butterfly and DR units have resulted lower register complexity in the design over the existing design [2]. Moreover, in the last stage, a merged unit for butterfly computation and DR has been proposed to perform either a half-butterfly operation or interchanging of data, and thereby reducing the hardware usage. The proposed serial pipelined FFT architecture can be adopted for low-power applications and is particularly useful for realizing large

number of points. Subsequently, another serial RFFT processor with a new DFG has been presented that reduces the number of multiplication stages undergoing multiplications with different TF in addition to maximum utilization of BF units. It results in reduced resources per multiplier in a serial FFT. Further, a shift-accumulation based multiplier designed by exploiting partial-product symmetries at the expense of an offset has been proposed for the RFFT processor. When this multiplier is extended for higher wordlengths, resource sharing is observed in the offset computation and the shift-add logic. A quantization strategy is presented to reduce the round-off error. FPGA and ASIC results have been presented to validate the efficiency of the designs, and compared with existing FFT architectures.

# 3

# Memory-Based FFT Architectures for Real-Valued Signals

Contents

## 3.1 Introduction

In the past, many research works have been carried out on pipelined FFT architectures for high-throughput applications [20,24,40,88]. However, the complexity of these architectures depends on the number of butterfly and multiplier units used. An alternative to the pipelined FFT is the memory-based FFT. As discussed in Chapter 1, pipelined FFT offers high-throughput while memory-based FFTs offer low-area. Parallel processing or high-radix butterfly units can be employed to increase the throughput of memory-based architectures. Several authors have suggested efficient memory-based architectures for computation of CFFT [32, 33, 36, 89–94]. Over the years, the focus has shifted towards efficient implementation of RFFT architectures, as most of the physical signals are real-valued. Moreover, CFFT architectures cannot achieve the same efficiency while processing an RFFT because the Hermitian symmetry in real-valued signal spectrum eliminates the redundant computations. In the past, there has been continuous efforts to improve the performance of memory-based RFFT architectures [15, 41, 42]. An in-place RFFT processor with low-hardware usage and high-hardware resource utilization has been proposed in [15]. Another architecture has been proposed in [41], to further reduce the hardware usage and the number of computation cycles while maximizing the resource utilization. It is found that the number of cycles required for FFT computation can also be reduced by increasing the number of processing elements (PEs). Further, number of multipliers required for implementation depends on the PEs used, rather than the number of stages. Therefore, hardware cost is relatively lower as compared to pipelined architectures for higher-points. This chapter presents two low-complexity RFFT architectures with new memory accessing schemes for conflict-free operation. In this work, different PEs are investigated to reduce the implementation complexity of proposed memory based RFFT processors.

## 3.2 Review of Existing Works

For real-valued input signals, it is not necessary to compute all the FFT coefficients, since $X(k) = X^*(N-k)$, where $X^*(\cdot)$ indicates conjugate of $X(\cdot)$. Fig. 3.1 shows data flow graph (DFG) of a 32-point RFFT which computes the outputs in five stages. A few important observations from the DFG are as follows: The last two stages do not require any twiddle factor (TF) multiplication with only one butterfly operation in stage 5. Also, $W^0$ is a real TF of value equal to one, therefore, its multiplication can be ignored. Conceptually, memory-based architectures are based on folding

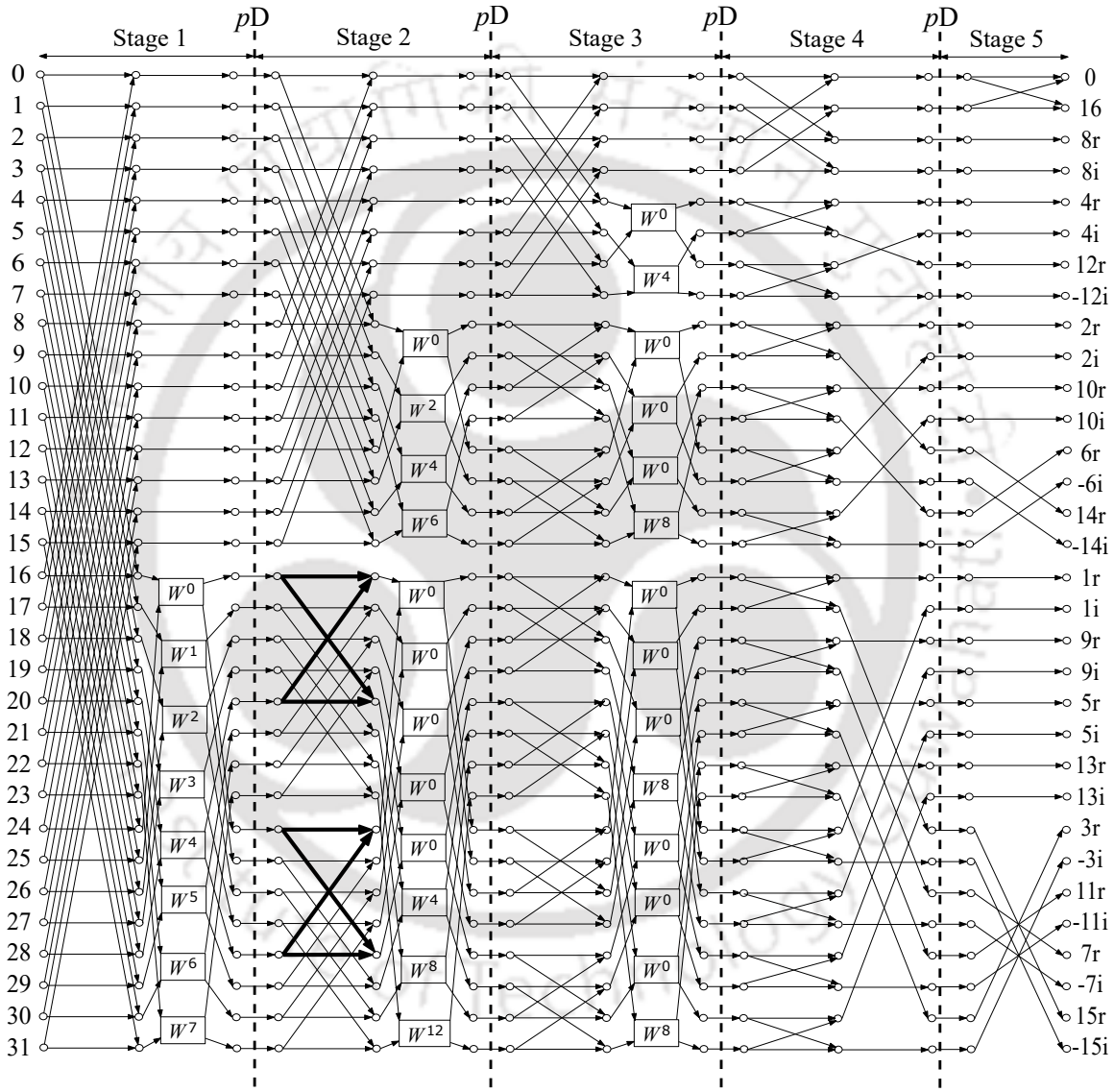


**Fig. 3.1.** Data-flow-graph (DFG) of a 32-point RFFT.

technique [14] for stage-by-stage FFT computation. It comprises of one or more PEs and memory units for data-buffering, intermediate computations and output reordering. In general, memory-based architectures require $3N$ memories for the computation of $N$-point FFT. Later, it has been established by some researchers that input and output memories can be merged which results in $2N$ memories [32, 33]. This is because the intermediate results of a particular stage have to be written to a memory location from which the next input is to be read and is referred as concurrent input/output (I/O) operation. It is found that the multiplier is one of the main area and power consuming unit in a memory based RFFT processor. In the past, several schemes [15, 32, 41, 42] have been suggested to reduce the hardware complexity of the multiplier in the PE or the overall complexity of the processor. In case of memory-based RFFT architectures, PE contains fixed number of multipliers and adders. In [15], two butterfly units are present in PE which employs four real adders, four multiplexers, a switch and a complex multiplier. A new stage-partition scheme with a less complex PE was introduced in [41]. However, it suffers from the problems of non continuous-flow and scrambled output data. For an architecture to support continuous-flow, it must perform both computation and data I/O simultaneously. This issue has been solved in [42] by a continuous-flow, normal-order output RFFT processor based on [41]. Although continuous-flow is gained, it leads to increase in the cost of multiplexers and registers. From the above discussion, it is clear that achieving low-hardware complexity and continuous-flow simultaneously is difficult in memory-based RFFT architectures. This motivates us to present two low-complexity RFFT architectures (Type-I, II) that support continuous-flow and normal-order output. In Type-I, the primary focus is on reducing the complexities of multiplexers and registers in the PE with proposed memory accessing scheme. In Type-II, the idea of Type-I design is extended to reduce the number of multipliers alongwith a different memory accessing scheme.

## 3.3 Proposed Scheme

The proposed RFFT architectures for Type-I, II are based on the DFG presented in [42]. Fig. 3.1 shows the DFG to compute the butterflies operation and multiplications in the first $(\log_2 N - 2)$ stages sequentially. As stated earlier, last-stage of such DFG requires a single butterfly operation which can be performed in one clock cycle. While the number of clock cycles required for computation from stage 1 to stage $(n-1)$ with four-input PE would be $(N/4)(\log_2 N - 1)$, where $n = \log_2 N$. Hence, one can estimate the total number of clock cycles for $N$-point memory based RFFT computation as

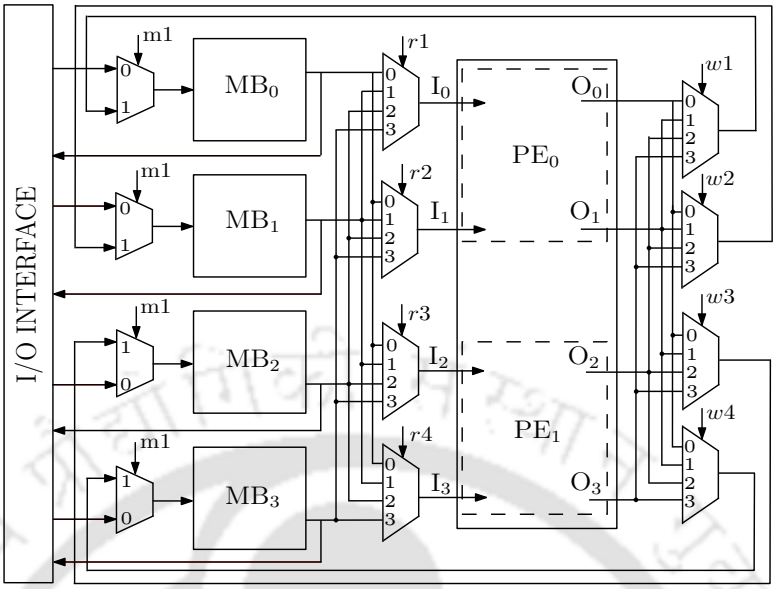


**Fig. 3.2.** Top-Level RFFT architecture with two-input PEs, $PE_{0,1}$ (Type-I).

$(N/4)(\log_2 N - 1) + 1$. In the proposed designs, $p$ registers are inserted for pipelining. The value of $p$ decides the number of clock cycles required by the PE to produce the correct output in Type-I, II designs. It is independent of the FFT size $N$. Noticeably, the pipelined registers would increase the number of clock cycles, but reduce the number of multipliers to be employed in a particular PE. For instance, an $N$-point FFT with one pipelined register in the PE would require additional number of clock cycles over the design reported in [41]. It leads to two real multipliers in each PE instead of one complex multiplier. In such case, the effective number of clock cycles would be $(N/4)(\log_2 N - 0.5) + 1$. In general, if $p$ be the number of pipelined registers to be employed in a four-input PE, then the total number of clock cycles would be $(N/4)(\log_2 N - (1/(p+1))) + 1$. Now, if instead of four inputs, two inputs are used, then total number of clock cycles for $N$-point RFFT computation would become $(N/2)(\log_2 N - (1/(p+1))) + 1$.

### 3.3.1 Architectural Details of Type-I Design

Fig. 3.2 shows the top-level architecture of the proposed FFT processor. It consists of two PEs, namely, $PE_0$ and $PE_1$, input-output (I/O) interface and sets of multiplexers. Further, four memory banks $MB_0$, $MB_1$, $MB_2$, $MB_3$ are used with depth $N/4$ and words size $W$-bits to store the input data, output data and the intermediate data samples. Thus, the total size of memory required for the implementation of Type-I design is $4 \times N/4 \times W$. For the proposed design, $W$ is taken as 16. Note

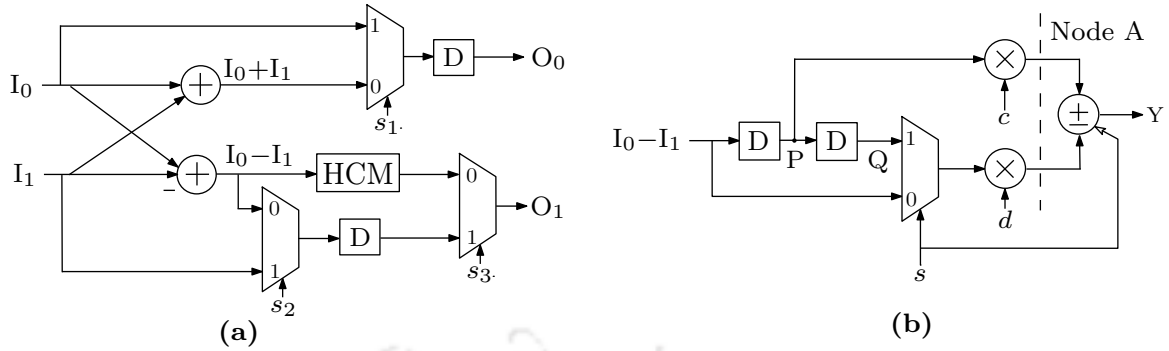


**Fig. 3.3.** Type-I design (a) Schematic of $PE_{0,1}$. (b) Schematic of half-complex multiplier (HCM). $c$ : $\cos(\alpha)$ and $d$ : $\sin(\alpha)$ which are the real and imaginary components of the twiddle-factor.

Table 3.1: Timing Diagram of HCM

| Clk | $I_0$-$I_1$ | P | Q | $s$ | Node A | Y |
|---|---|---|---|---|---|---|
| 0 | $x_{16}$ | – | – | – | – | – |
| 1 | $x_{24}$ | $x_{16}$ | – | 0 | $cx_{16}, dx_{24}$ | $cx_{16} - dx_{24}$ |
| 2 | $x_{20}$ | $x_{24}$ | $x_{16}$ | 1 | $cx_{24}, dx_{16}$ | $cx_{24} + dx_{16}$ |
| 3 | $x_{28}$ | $x_{20}$ | $x_{24}$ | 0 | $cx_{20}, dx_{28}$ | $cx_{20} - dx_{28}$ |
| 4 | $x_{18}$ | $x_{28}$ | $x_{20}$ | 1 | $cx_{28}, dx_{20}$ | $cx_{28} + dx_{20}$ |
| 5 | $x_{26}$ | $x_{18}$ | $x_{28}$ | 0 | $cx_{18}, dx_{26}$ | $cx_{18} - dx_{26}$ |
| 6 | $x_{22}$ | $x_{26}$ | $x_{18}$ | 1 | $cx_{26}, dx_{18}$ | $cx_{26} + dx_{18}$ |
| 7 | $x_{30}$ | $x_{22}$ | $x_{26}$ | 0 | $cx_{22}, dx_{30}$ | $cx_{22} - dx_{30}$ |

that the memory employed has dual ports to perform the read and write operations simultaneously. Initially, all the input samples are stored in respective $MB_i$ $(i = 0, 1, 2, 3)$ via four multiplexers labelled 'm1', as shown in Fig. 3.2. As mentioned, the computation of every stage requires the intermediate results back to the memory banks. When computation begins, the input samples already stored in the memory banks are read out via another set of multiplexers '$r1, r2, r3$ and $r4$'. It can be noted that the numbers in the multiplexers are in accordance with the memory bank indices '$i$'. After the samples are processed by the PEs, the four multiplexers '$w1, w2, w3$ and $w4$' decide the write sequence for the intermediate results. In the last two stages, reordering and write operation for intermediate computations can be performed concurrently before the next $N$-point data samples are read into the PEs. The notations of input ports and output ports are $I_i$ and $O_i$ respectively.

In this design, low-complexity PEs and memory addressing scheme are proposed to achieve continuous-flow architecture. Each of the PEs ($PE_0$ and $PE_1$) consists of a single butterfly, two registers (D), one multiplexer and a half-complex multiplier (HCM) to process two inputs in parallel, as shown in Fig. 3.3(a). The control signals to the multiplexers shown in Fig. 3.3(a) are set to $s_2 = 0$ in the current

clock cycle and $s_3 = 1$ in the next clock cycle to by-pass the HCM corresponding to $(n-1)$th stage and one butterfly computation in the $n$th stage, while $s_1 = 1$, $s_2 = 1$ in the current clock cycle and $s_3 = 1$ in the next clock cycle corresponding to $n$th stage. The schematic of proposed HCM is shown in Fig. 3.3(b). It consists of two real multipliers, one real adder, one multiplexer and two delay elements. The computation of two butterflies and one TF multiplication by one PE are performed in two consecutive clock cycles. Thus, the PEs process eight samples in two clock cycles. In other words, there are two PEs in the Type-I design, each PE takes 2 samples in one clock cycle as shown in Fig. 3.3(a). In other words, in one clock cycle, 4 samples are processed by the two PEs. Hence, in two clock cycles, 8 samples would be processed. To understand clearly, an explicit timing diagram for the operation of proposed HCM is shown in Table 3.1. The proposed HCM structure leads to less complexity as it involves one multiplexer. Further, registers in HCM are pre-placed to store the samples instead of post-placed as in [42] to store the results. By doing so, the register complexity in the proposed HCM almost reduces by $2W$ and leads to less power consumption over the design for several computation clock cycles.

#### 3.3.1.1 Memory Addressing Scheme

In every computation cycle, it is required to read and write the data value in the memory at appropriate address locations. For this, a conflict-free memory addressing scheme for Type-I RFFT architecture is investigated. To generate the addresses for each of the four memory banks $MB_0$, $MB_1$, $MB_2$ and $MB_3$, one main counter is required which is divided into sub-counters of depth $N/4$.

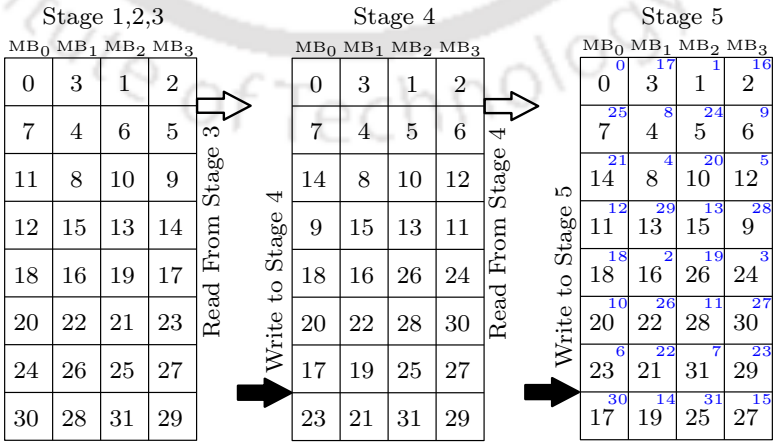


**Fig. 3.4.** Memory addressing scheme for 32-point RFFT, where $A^b$ indicates b is time index of final output sample A.

In memory-based FFT implementations, counters are commonly used to address the memory banks and access the appropriate data during the computation. The design of these counters and their next state depends on the specific memory structure and addressing scheme employed in the FFT implementation. First, the number of memory banks, the size of each bank and how the data is distributed across the banks needs to be determined from the data flow graph (DFG). Next, an addressing scheme is defined to access the memory banks effectively. This scheme specifies how the counters will increment or update to access the required memory locations. The addressing scheme is typically based on the specific FFT algorithm being implemented, such as radix-2. The counter design involves determining the number of bits needed to represent the memory bank addresses and designing the next state for these counters. The next state for the counters determine how the counter values change from one clock cycle to the next. The design of the counters and the next state requires correct addressing of the memory banks and efficient data access during the FFT computation.

In the proposed RFFT, all the input samples from 0 to $N-1$ are placed in the memory banks, as shown in Fig. 3.4. It is to be noted that as all four memories are accessed simultaneously, it must be assured that the four samples processed in the PE in every clock cycle come from different memories. For example, in case of a 32-point RFFT, the data with indices $(0, 16, 1, 17)$ are read from their respective memory banks and fed into the PE. Note that all these data belong to different memory banks. The butterflies in the DFG are computed in top-to-bottom order with a number of pre-defined rules. The related butterflies (shown in bold lines in Fig. 3.1) which enter the same multiplier in the first $(n-2)$ stages are to be computed through the same PE in consecutive cycles. As mentioned earlier, the data in the last two stages require data-reordering to obtain a normal-order output. Hence, in the $(n-1)$th stage, the butterflies with the two data samples which interchange their address locations are processed simultaneously. In the $n$th stage, the data samples that are to be interchanged are read and written in consecutive clock cycles. This leads to continuous-flow operation. For an $N(=2^n)$-point RFFT, a $(n-2)$-bit counter, $C_{n-2} = c_{n-3}c_{n-4}....c_0$ is used to generate addresses from 0 to $(N/4-1)$ for the four memory banks. The output of counter is physically mapped to address location within each bank across all the stages according to specific patterns. To understand the address patterns and memory addressing scheme clearly, an example of a 32-point RFFT is considered as shown in Fig. 3.4 and Fig. 3.5. A 3-bit counter generates the address locations from 0 to 7 for the four memory banks in each stage. The address of every sample index within a bank changes with each stage according

to Fig. 3.5. Each column in Fig. 3.4 corresponds to one memory bank and the numbers correspond to the indices of the input/output samples at each stage. The read and write patterns of $MB_{0,1,2,3}$ for stage 1, 2 and 3 are same *i.e.* the first $(n-2)$ stages follow in-place computation, while the last two stages need data reordering to obtain a normal order output. Moreover, data is always read from each of the memory banks in top-to-bottom order across all the stages. The last stage requires $N/8$ clock cycles as operations on only $N/4$ samples are carried out. The write and address pattern in the last stage is used to support normal-order output. When the operations in the last stage are over, the final result is tapped out through the output data path. The numbers depicted by blue colour in Fig. 3.4 indicate the time indices of the samples at the output.

### 3.3.2 Architectural Details of Type-II Design

The top-level architecture of the proposed Type-II RFFT is shown in Fig. 3.6. It consists of four memory banks, two single-input PEs ($PE_0$ and $PE_1$), an I/O interface and sets of multiplexers. The memory banks, $MB_0$, $MB_1$, $MB_2$, $MB_3$ work in groups of two at every stage. The two groups switch between two operational modes: computational and concurrent I/O. In other words, while one set of memory banks is involved in computations, the other set is handling data input and output

| Stage | Count | $c_2c_1c_0$ | $\bar{c}_2c_1c_0$ | $c_2\bar{c}_1c_0$ | $c_2c_1\bar{c}_0$ | $c_2\bar{c}_1\bar{c}_0$ | $\bar{c}_2\bar{c}_1\bar{c}_0$ |
|---|---|---|---|---|---|---|---|
| 1 | 0,3,4,7 | $MB_{0,2}$ | $MB_{1,3}$ | | | | |
| | 1,2,5,6 | | | | | $MB_{1,3}$ | $MB_{0,2}$ |
| 2 | 0,2 | $MB_{0,3}$ | | $MB_{1,2}$ | | | |
| | 1,3 | $MB_{1,2}$ | | $MB_{0,3}$ | | | |
| | 4 | $MB_{1,3}$ | | | $MB_{0,2}$ | | |
| | 5 | | | $MB_{1,3}$ | | $MB_{0,2}$ | |
| | 6 | | | $MB_{0,2}$ | | $MB_{1,3}$ | |
| | 7 | $MB_{0,2}$ | | | $MB_{1,3}$ | | |
| 3 | 0,1 | $MB_{0,2}$ | | | $MB_{1,3}$ | | |
| | 2,3,4,7 | $MB_{0,1,2,3}$ | | | | | |
| | 5,6 | | | | | $MB_{0,1,2,3}$ | |
| 4 | 0,1 | $MB_{0,1,2,3}$ | | | | | |
| | 2,3 | $MB_{1,3}$ | | | $MB_{0,2}$ | | |
| | 4 | $MB_{1,3}$ | | $MB_{0,2}$ | | | |
| | 5 | | | | $MB_{0,2}$ | $MB_{1,3}$ | |
| | 6 | | | | $MB_{1,3}$ | $MB_{0,2}$ | |
| | 7 | $MB_{0,2}$ | | $MB_{1,3}$ | | | |
| 5 | 0,3,4,7 | $MB_{0,2}$ | $MB_{1,3}$ | | | | |
| | 1,2,5,6 | | | | | $MB_{1,3}$ | $MB_{0,2}$ |

**Fig. 3.5.** Address pattern permutations of $MB_0$, $MB_1$, $MB_2$, $MB_3$ for 32-point RFFT.

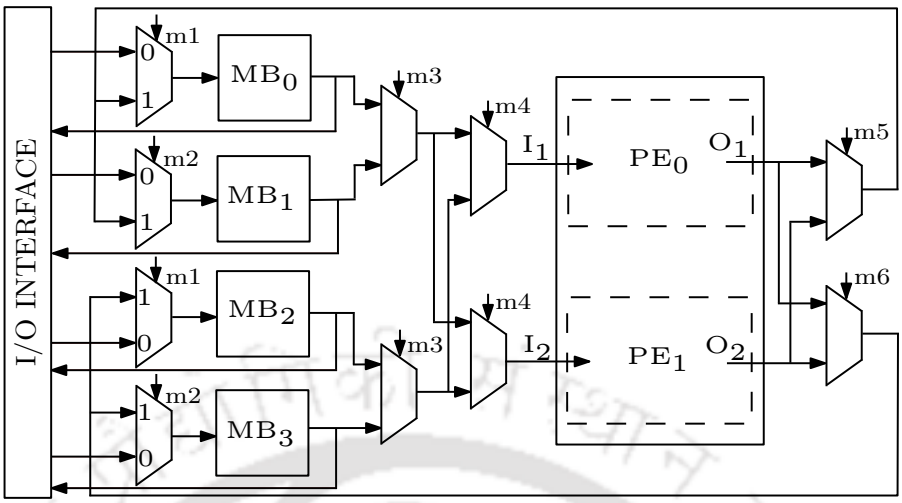


**Fig. 3.6.** Top-Level RFFT architecture with two single-input PEs, $PE_{0,1}$ (Type-II).

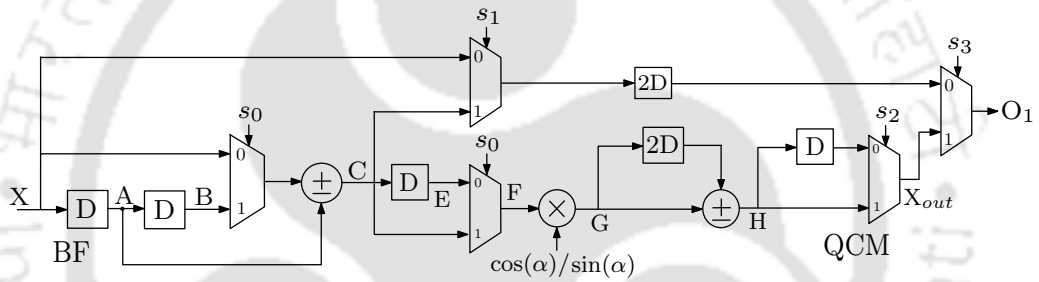


**Fig. 3.7.** Type-II design: Schematic of $PE_{0,1}$ with quarter-complex multiplier (QCM).

concurrently, that is, two memory banks are always working in the computational mode while the remaining two are working in the concurrent I/O mode. The multiplexers, m1 and m2 facilitate the memory groups in switching between the I/O interface and the intermediate results. The multiplexers, m3 and m4 select the two memory banks whose data are to be read simultaneously into the PEs. Finally, m5 and m6 select the banks to which the processed output is written. The controls of these multiplexers alternate for consecutive symbols. The architectural details of the single-input $PE_{0,1}$ is shown in Fig. 3.7. Unlike Type-I design, it has a half-butterfly unit and a new quarter-complex multiplier (QCM) unit to process the data from a memory bank. To clearly understand the operation of QCM unit, it is convenient to include the half-BF unit with QCM, as shown in Fig. 3.7. Both half-BF and QCM units operate sequentially during the first $(n-2)$ stages on sample-by-sample basis in the following two modes:

- The addition using the half-BF unit is performed in one clock cycle and the result is passed via

Table 3.2: Timing Diagram of $PE_{0,1}$ including QCM Operation in Stage 1

| Clk | X | A | B | $s_0$ | C | $s_1$ | E | F | G | H | $s_2$ | $s_3$ | $O_1$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $x_0$ | – | – | – | – | – | – | – | – | – | – | – | – |
| 1 | $x_{16}$ | $x_0$ | – | 0 | $x_0^1$ | 1 | – | – | – | – | – | – | – |
| 2 | $x_8$ | $x_{16}$ | $x_0$ | 1 | $x_{16}^1$ | – | – | $x_{16}^1$ | $cx_{16}^1$ | – | – | – | – |
| 3 | $x_{24}$ | $x_8$ | $x_{16}$ | 0 | $x_8^1$ | 1 | $x_{16}^1$ | $x_{16}^1$ | $dx_{16}^1$ | – | – | 0 | $x_0^1$ |
| 4 | $x_4$ | $x_{24}$ | $x_8$ | 1 | $x_{24}^1$ | – | – | $x_{24}^1$ | $dx_{24}^1$ | $cx_{16}^1 + dx_{24}^1$ | 1 | 1 | $cx_{16}^1 + dx_{24}^1$ |
| 5 | $x_{20}$ | $x_4$ | $x_{24}$ | 0 | $x_4^1$ | 1 | $x_{24}^1$ | $x_{24}^1$ | $cx_{24}^1$ | $cx_{24}^1 - dx_{16}^1$ | – | 0 | $x_8^1$ |
| 6 | $x_{12}$ | $x_{20}$ | $x_4$ | 1 | $x_{20}^1$ | – | – | $x_{20}^1$ | $cx_{20}^1$ | – | 0 | 1 | $cx_{24}^1 - dx_{16}^1$ |
| 7 | $x_{28}$ | $x_{12}$ | $x_{20}$ | 0 | $x_{12}^1$ | 1 | $x_{20}^1$ | $x_{20}^1$ | $dx_{20}^1$ | – | – | 0 | $x_4^1$ |
| 8 | $x_2$ | $x_{28}$ | $x_{12}$ | 1 | $x_{28}^1$ | – | – | $x_{28}^1$ | $dx_{28}^1$ | $cx_{20}^1 - dx_{28}^1$ | 1 | 1 | $cx_{20}^1 - dx_{28}^1$ |
| 9 | $x_{18}$ | $x_2$ | $x_{28}$ | 0 | $x_2^1$ | 1 | $x_{28}^1$ | $x_{28}^1$ | $cx_{28}^1$ | $cx_{28}^1 + dx_{20}^1$ | – | 0 | $x_{12}^1$ |
| 10 | $x_{10}$ | $x_{18}$ | $x_2$ | 1 | $x_{18}^1$ | – | – | $x_{18}^1$ | $cx_{18}^1$ | – | 0 | 1 | $cx_{28}^1 + dx_{20}^1$ |
| 11 | $x_{26}$ | $x_{10}$ | $x_{18}$ | 0 | $x_{10}^1$ | 1 | $x_{18}^1$ | $x_{18}^1$ | $dx_{18}^1$ | – | – | 0 | $x_2^1$ |
| 12 | $x_6$ | $x_{26}$ | $x_{10}$ | 1 | $x_{26}^1$ | – | – | $x_{26}^1$ | $dx_{26}^1$ | $cx_{18}^1 - dx_{26}^1$ | 1 | 1 | $cx_{18}^1 - dx_{26}^1$ |

$x_m^n$ is output of $n$th stage with sample index $m$

the by-pass line with two-delay units to the output multiplexer.

- The subtraction using the half-BF unit is performed in the next clock cycle and the result is processed by the QCM. Subsequently, it undergoes real multiplication in the QCM unit which takes two clock cycles to compute $cx_j + dx_k$ (or $cx_k - dx_j$) using the four quarter complex terms $cx_j, cx_k$, $+dx_k$ and $-dx_j$, where $j, k$ denote the sample indices with $j \neq k$, and $c$ (or $\cos(\alpha)$) and $d$ (or $\sin(\alpha)$) are real and imaginary components of twiddle-factor respectively.

The proposed QCM unit comprises of one real adder, one real multiplier, two multiplexers and four registers. Table 3.2 shows the complete operation of proposed QCM using a separate timing diagram with half-BF unit in stage 1. Note that the letters in the table correspond to various intermediate nodes of half-BF and the QCM units.

#### 3.3.2.1 Memory Addressing Scheme

The memory addressing scheme and address patterns for Type-II design is the same as that of Type-I design. The only difference lies in the way of accessing the memory banks. For instance, in Type-I, the PEs access all the four banks simultaneously while in the case of Type-II, the PEs access only two of the banks at a time. Note that in Type-II design, these two memory banks form one group. In stage 1 and stage 3, ($MB_0$, $MB_2$) forms one group while ($MB_1$, $MB_3$) forms the second group. In stage 2, for the upper half, the two groups are ($MB_0$, $MB_3$) and ($MB_1$, $MB_2$) while for the

lower half, the two groups are ($MB_0$, $MB_2$) and ($MB_1$, $MB_3$). In stage 4, the groups are ($MB_0$, $MB_2$) and ($MB_1$, $MB_3$) for C = 0; ($MB_0$, $MB_3$) and ($MB_1$, $MB_2$) for C = 1, and ($MB_0$, $MB_1$)and ($MB_2$, $MB_3$) for remaining cases. In stage 5, groups are ($MB_0$, $MB_2$), ($MB_1$, $MB_3$). The butterflies in the DFG are computed in top-bottom order with the pre-defined rules of Type-I, as discussed earlier. The read and write patterns are in accordance with the DFG. The write cycle is three copy delay of the read cycle for the first $(n-2)$ stages as the PE has a latency of three. In stage $(n-1)$ and stage $n$, the write cycle changes the memory banks of the processed samples that require data reordering according to the DFG shown in Fig. 3.1. Algorithm 3.1 explains the operation of Type-I, II designs.

**Algorithm 3.1** Compute $N$-point RFFT with $i$-input PE

1: **Notations:**
WP: write port, RP: read port, Addr: $MB_i$ address
2: **Initialize:**
$C_i = 0 \ \forall \ l \in [0, N/i-1]$; $m, k \in [0, N-1]$
3: **for** $X(k) = \sum_{m=0}^{N-1} x(m) W_N^{mk}$ **do**
  **if** $E_{\overline{WP}/RP} == 0$ **then**
    $I_i[\mathrm{MB}_i^{C'_i}[O_i]] \leftarrow O_i$ // $\mathrm{WP}[\mathrm{MB}_i^{\mathrm{Addr}}[\mathrm{RP}]]$
  **else**
    $I_i \leftarrow \mathrm{I}_i[\mathrm{MB}_i^{C'_i}[O_i]]$
  **end**
  **for** $n = 0$ **to** $\log_2 N - 1$ **do**
    **for** $l = 0$ **to** $N/i - 1$ **do**
      $C = C + 1$
      **load** $C_i = C$
      **map** $C'_i = C_i$ // (refer Fig. 3.5)
    **end**
  **end**
**end**

## 3.4 Performance Comparison

In this section, the hardware and time complexities of the proposed and existing designs are first compared. Next, the performance of Type-I, II architectures along with the existing designs is evaluated and compared through ASIC synthesis and FPGA implementation.

### 3.4.1 Hardware and Time Complexities

The estimated hardware and time complexities of the proposed and existing schemes in terms of mutlipliers, adders, multiplexers, memory and throughput complexities are listed in Table 3.3. The architecture in [32] supports continuous-flow operation, but involves redundant operations when applied to RFFT computations. In the sequel, RFFT architectures reduce the memory by a factor of two over CFFT architectures. The existing architectures [15, 41] support neither concurrent I/O operation nor continuous-flow operation. Though the design presented in [42] supports continuous-flow operation, it requires high complexity of registers and more number of multiplexers. To address this issue, two new designs for RFFT computations are presented in this chapter. To reduce the hardware computations, the multipliers are moved towards the output side, while the inputs to the HCM are directly stored in the registers in the Type-I design. The complexity of multiplier is further decreased in Type-II design by employing a QCM. When these computational units are used over several clock cycles, it leads to increase in power consumption. Since multiplication always results in increase of wordlength to $2W$ for the operands having wordlength of W-bits, the PE of Type-I design allows reduction in power consumption. The proposed Type-I requires $2W$ less registers per clock cycle compared to [42], therefore total utilization of registers for $2W \times [(N/4)(\log_2 N - 0.5) + 1]$. The savings have direct impact on dynamic power consumption and it would be higher for both memory wordlengths $W$. Further, Type-I design uses less number of multiplexers both in PE and memory access, while using the same number of multipliers and adders as that of [15, 41] and [42]. It is important to note that the number of multiplexers in Type-II design is least among all the existing schemes while Type-I uses less number of registers over [42].

The FFT architectures presented in [15] and [41], each employ a memory consisting of $2N$ words. However, neither of these designs facilitate concurrent I/O operations. Moreover, for continuous-flow operation, both require an additional $N$-word memory as an output buffer. Additionally, the output order in these designs is scrambled, demanding supplementary processing to attain a normal-order output. It is well known that when the size of the RFFT increases, memory usage takes precedence over the hardware complexity. Notably, when compared with the designs presented in [15] and [41], which require an additional $N$-word memory for continuous-flow operation, our proposed designs excel in facilitating continuous-flow operation and delivering a normal-order output.

Table 3.3: Comparison of Computational Complexities of Different Memory-Based FFT

| Design | Multipliers | Adders | Registers | Memory | MUX/ DEMUX | CC | CP |
|---|---|---|---|---|---|---|---|
| [32] | 12 | 22 | 0 | $4N$ | 38 | $(N/4)\log_4 N$ | $T_{MA}+2T_A+8T_{MX}+2T_M$ |
| [15] | 4 | 6 | 0 | $2N$ | 38 | $(N/4)\log_2 N$ | $T_{MA}+T_A+8T_{MX}+2T_M$ |
| [41] | 4 | 6 | 0 | $2N$ | 28 | $(N/4)(\log_2 N-1)+1$ | $T_{MA}+T_A+5T_{MX}+2T_M$ |
| [42] | 4 | 6 | 10 | $2N$ | 50 | $(N/4)(\log_2 N-(1/2))+1$ | $T_{MA}+T_A+11T_{MX}+2T_M$ |
| Proposed Type-I | 4 | 6 | 8 | $2N$ | 36 | $(N/4)(\log_2 N-(1/2))+1$ | $T_{MA}+T_A+6T_{MX}+2T_M$ |
| Proposed Type-II | 2 | 4 | 16 | $2N$ | 20 | $(N/2)(\log_2 N-(1/3))+2$ | $T_{MA}+T_A+7T_{MX}+2T_M$ |

MUX/DEMUX: MUltiplexers/Demultiplexers, CC: clock cycles, CP: clock period, Throughput = 1/(CC×CP), $T_{MA}$, $T_A$, $T_{MX}$ and $T_M$ are delays of multiply-add unit, 16-bit adder, 2-to-1 multiplexer and memory-access, respectively.

The time-complexities of different designs are listed in terms of throughput. It is determined by the clock-period (CP) (or critical path) and number of clock cycles (CC) involved in the design, as listed in Table 3.3. As shown in Table 3.3, the critical path of proposed Type-I design is reduced as compared to [42], while it is the same as that of [15]. On the other hand, number of clock cycles required for the proposed design is less than that of [15] and it leads to higher throughput. While the scheme in [41] consumes second least number of clock cycles and clock period, however it uses non-continuous flow. Furthermore, the number of clock cycles required in the proposed scheme is the equal to that of the architecture in [42]. However, the computation time required to compute the $N$-point RFFT is less than [42] owing to shorter clock period. In above definition, [41] is considered as the reference due to aforesaid reasons.

### 3.4.2 Implementation Results

#### 3.4.2.1 Application Specific Integrated Circuit (ASIC)

The proposed and existing designs are coded in Verilog for 16 and 32-points FFT. Application specific integrated circuit (ASIC) synthesis is performed by Cadence 14.1 RTL compiler using TSMC 90 nm CMOS technology. The corresponding synthesis results of Type-I, II designs in terms of area, power, minimum clock period (MCP), area delay product (ADP) and energy are listed in Table 3.4. It can be noted that the proposed Type-I design occupies slightly less area, while it consumes significantly less power compared to existing designs. Further, there is also marginal reduction in MCP for the Type-I design since less combinational logic delay is involved for the RFFT computation over the design [42]. For example, a 32-point Type-I RFFT occupies nearly 5.06% less area, 15.1% less power, 6.58% less SLUT and 5.25% less FF while Type-II offers 47.76% less area, 43.64% power, 48.22% less SLUT and 43.48% less FF over the RFFT in [42]. In case of proposed Type-II design, the amount of area and power reduction are substantial since multiplier complexities are significantly reduced as compared to Type-I design. It comes at the cost of increased MCP over the Type-I design. For better comparison, ADP and energy values of different designs are estimated for their computational cycles, and corresponding results are listed in Table 3.4 and represented in Fig. 3.8. As stated earlier, while analyzing the results, it is important to note that [32] is a CFFT, whereas [15] and [41] does not support concurrent I/O and continuous flow operation.

Table 3.4: Performance Comparison of Different Memory-Based FFT Architectures

| **Design** | $N$ | **Area** ($\mu$m$^2$) | **Power** (mW) | **MCP** (ns) | **ADP** ($\mu$m$^2\times$ ns) | **Energy** (mW.ns) | **SLUT** | **FF** |
|---|---|---|---|---|---|---|---|---|
| [32] | 16 | 88124 | 10.92 | 5.13 | 452076.12 | 56.01 | 3291 | 3487 |
| | 32 | 169231 | 22.03 | 6.18 | 1045847.58 | 136.14 | 6253 | 6501 |
| [15] | 16 | 31921 | 4.62 | 2.14 | 68310.94 | 9.89 | 1198 | 1215 |
| | 32 | 43676 | 7.13 | 2.17 | 94776.92 | 15.47 | 2435 | 2502 |
| [41] | 16 | 32253 | 4.71 | 2.25 | 72569.25 | 10.59 | 1312 | 1347 |
| | 32 | 45238 | 7.42 | 2.33 | 105404.54 | 17.29 | 2652 | 2763 |
| [42] | 16 | 35497 | 4.97 | 2.42 | 85902.74 | 12.03 | 1485 | 1507 |
| | 32 | 49187 | 7.81 | 2.51 | 123459.37 | 19.60 | 2853 | 2932 |
| Proposed Type-I | 16 | 34078 | 4.22 | 2.31 | 78720.18 | 9.75 | 1394 | 1432 |
| | 32 | 46698 | 6.42 | 2.38 | 111141.24 | 15.28 | 2665 | 2778 |
| Proposed Type-II | 16 | 18578 | 2.88 | 4.42 | 82114.76 | 12.13 | 783 | 881 |
| | 32 | 24903 | 4.40 | 4.49 | 111814.47 | 19.15 | 1477 | 1657 |

MCP: minimum clock period, ADP: area delay product, SLUT: sliced look-up table, FF: flip-flop

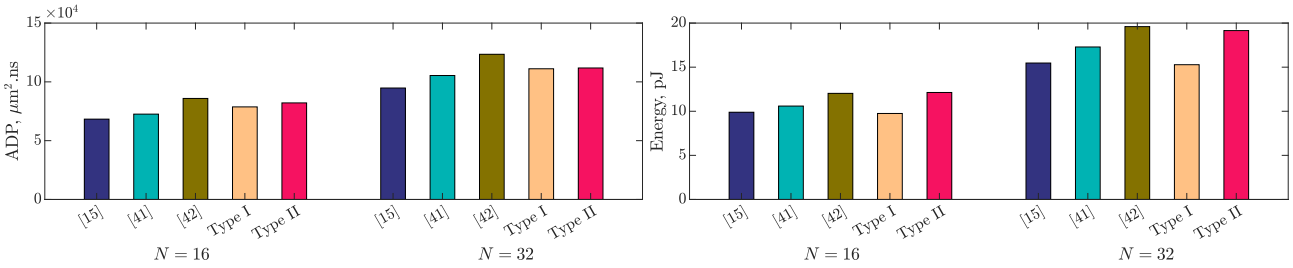


**Fig. 3.8.** Performance Comparison (a) Area-Delay-Product (ADP) (b) Energy of the Different Memory-Based FFT Architectures with Proposed Type-I and Type-II

#### 3.4.2.2 Field Programmable Gate Array (FPGA)

The proposed and existing designs are also implemented on a Xilinx ZYNQ FPGA device (XC7Z020-1CLG84C) for 16-point and 32-point RFFT. The logic utilization is obtained in terms of slice LUTs (SLUT) and flip-flops (FF) by setting the system clock at 50 MHz, as listed in Table 3.4. From the implementation results, it is clear that the proposed Type-I design for 16-point RFFT has almost 5.45% less SLUT and 4.97% less FF over best existing design [42]. On the other hand, the proposed Type-II design offers 45.66% less SLUT and 41.86% less over the design in [42].

## 3.5 Conclusion

In this chapter, two new memory-based architectures for RFFT computation based on novel conflict-free memory access strategies has been presented to achieve continuous-flow and a normal-order output. In Type-I design, a new approach has been developed to minimize the multiplexers and registers complexity. Furthermore, for the same number of clock cycles, the proposed scheme requires less computation time compared to [42] to compute the $N$-point RFFT due to shorter clock period. It helps achieve a reduction in power consumption and increase the throughput. The proposed RFFT has been implemented efficiently on ASIC and FPGA platforms. Further reductions can be achieved for higher-point FFT computations. The proposed PE achieves lower hardware usage and complexity as compared to the recently published schemes. Synthesis results validate the feasibility of the proposed designs. The most important feature of proposed scheme is that it is continuous-flow architecture with low-hardware complexity and is applicable to real-time processing.

Both Type-I and Type-II designs can be adapted to various applications depending on specific processing requirements and constraints of the system. Type-I is preferable for applications with higher

throughput and low area constraints such as wireless communications (4G LTE) for processing multiple parallel data (MIMO), medical imaging to accelerate image reconstruction, allowing faster scan times, audio and speech processing when dealing with multiple audio channels. Type-II is preferable in applications such as audio filtering to process stereo audio signals with two channels, spectral analysis, feature extraction, etc. where area constraints are high but does not require very high throughput.

# 4

# Fully-Parallel Split-Radix FFT



Contents

## 4.1 Introduction

With the advent of technology, the demand for efficient wireless communication systems raises new design challenges. In the recent times, low-latency, high-throughput FFT processors have gained a lot of interest in real-time applications. Parallel FFTs are often employed in high-throughput applications, however they result in large area and power consumption due to high hardware cost [24, 95]. Hence, fully-parallel FFTs where each addition and multiplication in the flow graph requires an adder and a multiplier have not been explored explicitly [96]. Fully-parallel FFTs offer the highest throughput. Hence, there is a growing need to design and develop low-area and low-power fully-parallel FFT architecture.

In the past, fully-parallel FFTs have been employed in applications such as high definition (HD) streaming, chromatic dispersion filtering, beamspace processing and radar [3, 45–47]. FFT is a computationally intensive algorithm consisting of many multiplication and addition operations. Therefore, in fully-parallel FFTs, the number of multipliers becomes significant thereby increasing the area and power consumption. So far, only a few fully-parallel architectures, viz. radix-$2^k$, radix-4 and split-radix (SR) have been reported in the literature either to realize high-throughput or low-energy [3, 45–47]. The SRFFT has the least computational complexity among its peers, especially in number of non-trivial multiplications [3]. It is quite evident in FFT size higher than 16. For instance, a 32-point SRFFT has 26 non-trivial twiddle factors (TFs), whereas a radix-$2^2$ FFT has 28 non-trivial TFs. Interestingly, the difference in the number of non-trivial multiplications increases significantly for higher point FFTs [3]. These are considered to be the main source of area and power consumption in an FFT realization. To keep the multipliers to a minimum, the SRFFT algorithm is chosen for the proposed design. However, when pipelining is employed to increase the throughput, the number of pipelined registers increases significantly in an SRFFT thereby increasing the power consumption along with clock latency [3]. This is due to the non-trivial multiplications present at each stage (except first and last one/two stages) as all the paths in a fully-parallel pipelined FFT have to be balanced with equal number of pipelined registers to make the architecture regular. It prevents the SRFFT from taking advantage of its inherent low computational complexity.

The designs [3, 45] were implemented on FPGA. The SRFFT design in [46] focuses on achieving scalability and high-throughput. Nonetheless, it is a non-pipelined architecture disregarding multiplier complexity for area and power reduction. In the recent past, a radix-4 design for fully-parallel pipelined

FFT with ASIC implementation has been presented [47]. The improvements in area and energy are due to short wordlengths of TF coefficients and input/output data. The main limitation of the existing methods is that further analysis to obtain low-complexity while achieving a higher throughput through architectural enhancements has not been done.

It is worth noting that only $N/8$ TFs need to be implemented for an $N$-point FFT [97]. Thus, for $N = 32$, only $W_{32}^1, W_{32}^2, W_{32}^3$ and $W_{32}^4$ need to be implemented owing to the symmetry property, $W_N^{\phi+N/2} = -W_N^{\phi}$ ($\phi$ is the angle) among the TFs. Further, using trigonometric identities, only the constant coefficients, $\cos(\pi/16)$, $\sin(\pi/16)$, $\cos(\pi/8)$, $\sin(\pi/8)$, $\cos(3\pi/16)$, $\sin(3\pi/16)$ and $\cos(\pi/4)$ are required to implement $W_{32}^1$, $W_{32}^2$, $W_{32}^3$ and $W_{32}^4$. Thus, all the multipliers corresponding to a 32-point FFT can be implemented by these seven constant coefficients. Usually, TF multipliers are realized with traditional constant multiplication approaches such as canonic signed digit (CSD) [47] representation, when one of the operand is known a priori. To the best of our knowledge, the effect of separating the shifters and adders in a shift-and-add network on area has not been reported. It is clear from the above discussion that there is a growing need to develop an efficient fully-parallel pipelined FFT. In this work, a low-area, low-power and low-latency architecture of fully-parallel pipelined SRFFT is presented. The key contributions of this work are as follows:

(i) Data flow graph for SRFFT that reduces the multiplication stages and brings down the number of pipelined registers significantly thereby reducing the power consumption along with latency.

(ii) Architectural analysis of constant multipliers (CM) to achieve low-area and low-power.

(iii) A substructure sharing scheme for the TF multiplier to compute the real and imaginary outputs with the same CM unit rather than dedicated real and imaginary circuits. The multipliers are run at double clock frequency compared to the rest of the architecture to facilitate the sharing and allow continuous flow with significantly reduced area.

## 4.2 Proposed Design

To address the issue of large number of pipelined registers in fully-parallel pipelined SRFFT, a TF shifting scheme is proposed as shown in Fig. 4.1. It depends on the relation between the TFs on the upper and lower paths of the butterfly. Accordingly, the butterflies are divided into two categories depicted by Fig. 4.1(a) and Fig. 4.1(b). Both the schemes show that if the same constant coefficients

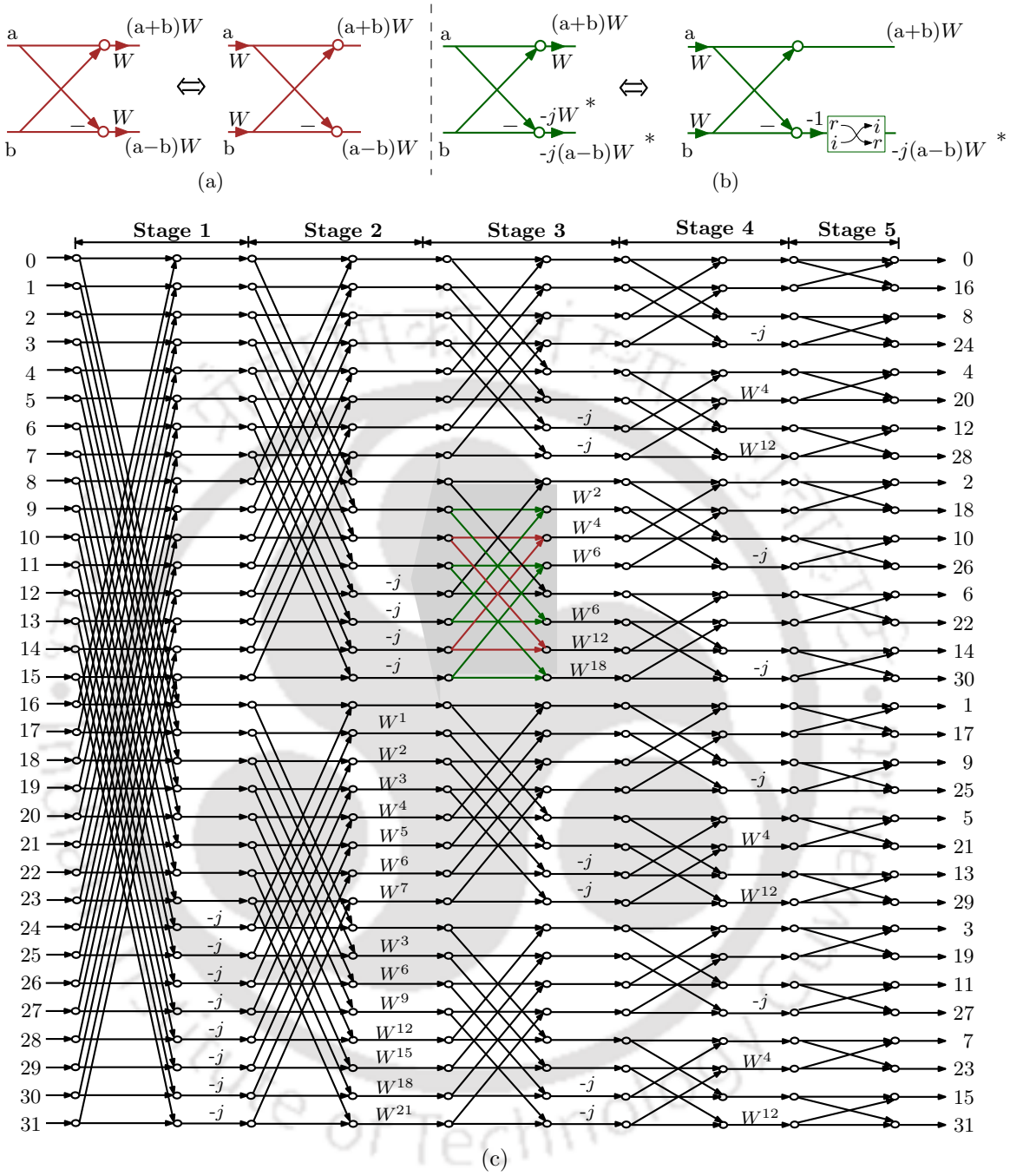


**Fig. 4.1.** Proposed twiddle factor (TF) shifting schemes. (a) Scheme 1. (b) Scheme 2. (c) TF shifting in a portion of 32-point SRFFT flow graph.

appear on the upper and lower paths after a butterfly operation, it is equivalent to performing the TF multiplication prior to the butterfly operation. Contrary to TFs in Fig. 4.1(a), some TFs cannot be directly expressed in terms of one another, although they have the same constant coefficients in the real and the imaginary parts, viz. $W^2$, $W^6$ and $W^{18}$. In such cases, the lower path requires sign alteration

and/or swapping of the real and imaginary parts after the butterfly operation to obtain the desired output, as shown in Fig. 4.1(b). It results in absence of non-trivial TF multiplications in odd stages of the data flow graph. The implementation of the proposed TF shifting in a portion of Stages 2 and 3 in a 32-point SRFFT is shown in Fig. 4.1(c). Hence, the latency and the number of registers needed to pipeline and balance them in each path in the odd stages is reduced as compared to SRFFT [3]. For instance, conventional SRFFT has a latency of $3L+6$ for a 32-point and $6L+3$ for a 256-point fully-parallel FFT, whereas after shifting the TFs, the latencies are $2L+6$ and $3L+9$, respectively, where '$L$' is the multiplier latency. For clarity, a comparison of latency of the state-of-the-art SRFFT [3], and the proposed SRFFT is listed in Table 4.1.

Table 4.1: Latency Comparison of SRFFT [3] and Proposed SRFFT

| $N$ | 16 | 32 | 64 | 128 | 256 |
|---|---|---|---|---|---|
| SRFFT [3] | 12 | 19 | 27 | 35 | 43 |
| Proposed | 9 | 14 | 15 | 20 | 21 |

Latency (in clock cycles)

### 4.2.1 Architectural Analysis of Constant Multipliers

As stated earlier, TF multiplications are composed of constant-coefficient multiplications by $\cos(\phi)$ and $\sin(\phi)$. These are represented in a 16-bit canonic signed digit (CSD) format for reasonable accuracy. Subsequently, they are realized using the shift-and-add approach. For instance, the CSD representation of $\cos(\pi/8)$ and $\sin(\pi/8)$ in $W_{32}^2$ are $1000\bar{1}0\bar{1}001000010$ and $010\bar{1}000100000\bar{1}00$ respectively. To reduce the implementation cost, the common sub-expression such as $100000\bar{1}$ is shared between the coefficients, as shown in Fig. 4.2. Clearly, they are designed with shift-and-add approach based on the frequency of occurrence of 1 or $\bar{1}$, for instance, $(x)\times\cos(\pi/8) = [x + (x - x << 5) << 5 + (x << 6 - x) << 8] << 1$ and $(x)\times\sin(\pi/8) = [(x << 2 - x) << 10 + (x << 6 - x)] << 2$. Similarly, it holds for other constant coefficients. Such an implementation is referred as straightforward (SF) implementation of the proposed CM. It is clear that SF structure has shifters and adders merged together. In contrast, the modified (MF) approach of shift-and-add structure separates out the left shifts to the input side. Hence all the inputs undergo the required shifts before addition. The adders then solely perform their operation on the shifted versions of the operands, while the remaining operands are sign-extended to have the same size. It is well known that if one of the operands has left shifts more than the length of the other operand, then during the addition, the other operand can

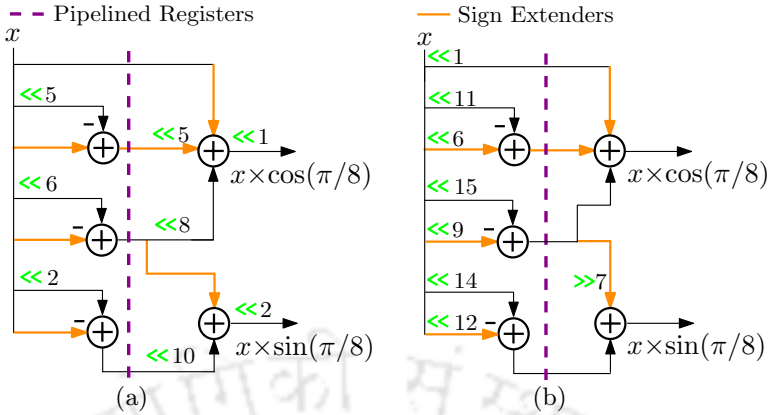


**Fig. 4.2.** Shift-add CM for $W_{32}^{2}$ with/without pipelining (a) SF (b) MF.

be placed directly at the least significant bits position. Nonetheless, if the adders in tree topology are realized with the '+' Verilog operator during the synthesis, then the tool would completely flatten the structure to the basic logic gates with high effort. In case of SF, the merged shift-and-add structure does not allow the tool to optimize the structure to the extent of MF. Both the structures are extended to their pipelined variants in order to reduce the critical path delay due to adders. By breaking down the computation into smaller stages, pipelining allows for better resource utilization. Each stage can be optimized independently, and the resources in each stage can be utilized more efficiently. Pipelined designs are generally easier to close in timing because of the reduced critical path delay and improved distribution of workload across stages.

The design analysis of both the proposed structures (with/without pipelining) is carried out through Synopsys Design Compiler Tool. The synthesis results are obtained at 100 MHz and 990 MHz (maximum) clock frequencies using TSMC 65 nm CMOS Library. The corresponding results in terms of area, power, data arrival time (DAT) and area-delay-power product (ADPP) are listed in Table 4.2. Clearly, the pipelined MF (MF*) performs superior in terms of area, power and ADPP for both frequencies compared to pipelined SF (SF*). For instance, the MF* offers 15.09% less area and 7.29% less power compared to the SF* at 990 MHz. Moreover, the ADPP of MF* is 20.83% less than that of SF*. As explained earlier, pipelining alongwith relocating the shifts on the input side leads to a more compact and optimized design, reducing delays and improving data arrival time (DAT). This helps in achieving better timing closure during the ASIC design process. Hence, the area, power and ADPP of pipelined SF (SF*) are higher compared to that of pipelined MF (MF*). As this work focuses on area and power-efficient implementation of pipelined SRFFT, the MF* variant of CM is

used to design the TF multiplier.

Table 4.2: Performance Comparison of Constant Multipliers

| **Design** | **Clock frequency** (MHz) | **Area** ($\mu m^2$) | **Power** ($\mu W$) | **DAT** (ns) | **ADPP** ($\mu m^2 \cdot ns \cdot W$) |
|---|---|---|---|---|---|
| SF | 100 | 807.84 | 26.88 | 2.48 | 0.05 |
| | 990 | 1291.32 | 470.54 | 1.01 | 0.61 |
| MF | 100 | 748.80 | 25.55 | 2.28 | 0.04 |
| | 990 | 1083.24 | 418.73 | 1.01 | 0.46 |
| SF* | 100 | 1010.16 | 55 | 2.52 | 0.14 |
| | 990 | 1562.76 | 778 | 0.99 | 1.20 |
| MF* | 100 | 974.16 | 54 | 2.36 | 0.12 |
| | 990 | 1326.96 | 721.31 | 0.99 | 0.95 |

*: Pipelined versions of SF and MF, DAT: Data arrival time, ADPP: Area-delay-power product, ADPP = Area × DAT × Power

### 4.2.2 Half-Complex TF Multiplier

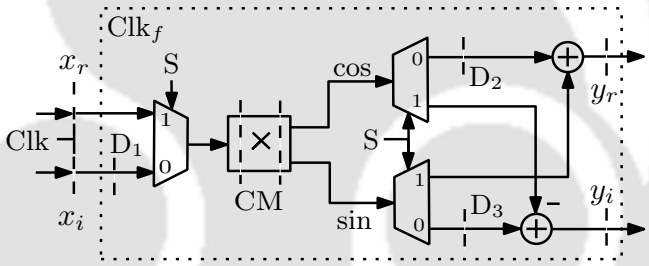


**Fig. 4.3.** TF multiplier with proposed substructure sharing scheme.

Usually, TF multiplier computes the real and imaginary components of the input with twice the hardware resources because the CM unit has to be replicated twice to process the real and imaginary inputs individually. To reduce the hardware complexity and maximize the utilization of resources, a substructure-sharing scheme is proposed to compute the real and imaginary components with a single hardware unit of CM, as shown in Fig. 4.3. It requires one multiplexer and two de-multiplexers, five registers ($D_1$, $D_2$, $D_3$, $y_r$ and $y_i$), an adder and a subtractor, where signal 'S' selects either real input or imaginary input from the input multiplexer and the corresponding de-multiplexers. To facilitate the sharing, the TF multiplier (area within the box) is run at twice the input clock frequency to allow continuous flow of data. This can be understood from the data sequence of the TF multiplier listed in Table 4.3. It shows that the composite structure requires four fast clock cycles (Clk$_f$ 0-3) to compute the real ($y_r$) and imaginary ($y_i$) components. An additional clock cycle (Clk 0) is required to store

Table 4.3: Data Sequence of TF Multiplier

| Clk | $\text{Clk}_f$ | S | $x_r$ | $x_i$ | $D_1$ | $D_2$ | $D_3$ | $y_r$ | $y_i$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | – | 0 | $r_1$ | $i_1$ | – | – | – | – | – |
| – | 0 | 1 | – | – | $i_1$ | – | – | – | – |
| 1 | 1 | 0 | $r_2$ | $i_2$ | – | – | – | – | – |
| – | 2 | 1 | – | – | $i_2$ | c·$r_1$ | s·$r_1$ | – | – |
| 2 | 3 | 0 | $r_3$ | $i_3$ | – | – | – | c·$r_1$+ s·$i_1$ | c·$i_1$-s·$r_1$ |
| – | 4 | 1 | – | – | $i_3$ | c·$r_2$ | s·$r_2$ | – | – |
| 3 | 5 | 0 | $r_4$ | $i_4$ | – | – | – | c·$r_2$+ s·$i_2$ | c·$i_2$-s·$r_2$ |

Clk: clock, $\text{Clk}_f$: fast clock, $\text{Clk}_f = 2\times\text{Clk}$, c: cos, s: sin

the BF output prior to the TF multiplication. Hence, a total latency of three clock cycles (Clk 0-2) is required in stage 2 and stage 4, as shown in Fig. 4.4.

### 4.2.3 Fully-Parallel Pipelined SRFFT Architecture

Based on the new data flow graph, the corresponding architecture for a 32-point fully-parallel SRFFT is shown in Fig. 4.4, where the letter '$L$D' specifies the latency of each stage with $L$ representing the number of pipelined registers in a path. It can be noted that after an initial latency of 10 clock cycles, 32 samples are obtained at every clock, *i.e.*, the throughput of the proposed design is $N$ samples per clock cycle. This is due to aggressive pipelining used in the architecture. In addition, the architecture consists of butterfly units, TF multipliers and bit-reversal mechanism (in stage 5). The butterfly unit consists of an adder and a subtractor, and bit-reversal can be readily performed by hardwiring.

## 4.3 Performance Comparison

In order to validate the proposed design, fully-parallel pipelined FFTs are coded in Verilog for $N$ = 32, 64, 128 and 256, and ASIC synthesis is carried out to estimate the area and power by Synopsys Design Compiler using UMC 65nm process library. All the simulations are performed using the typical corner case. The corresponding results are listed in Table 4.4 and compared with the state-of-the-art fully-parallel FFT designs reported in [46] and [47]. For fair comparison, area- and energy-efficiency are considered as figures of merit wherein the area and power are normalized with respect to the clock frequency.

The area listed in Table 4.4 seems to be comparable, however, it is to be noted that these values

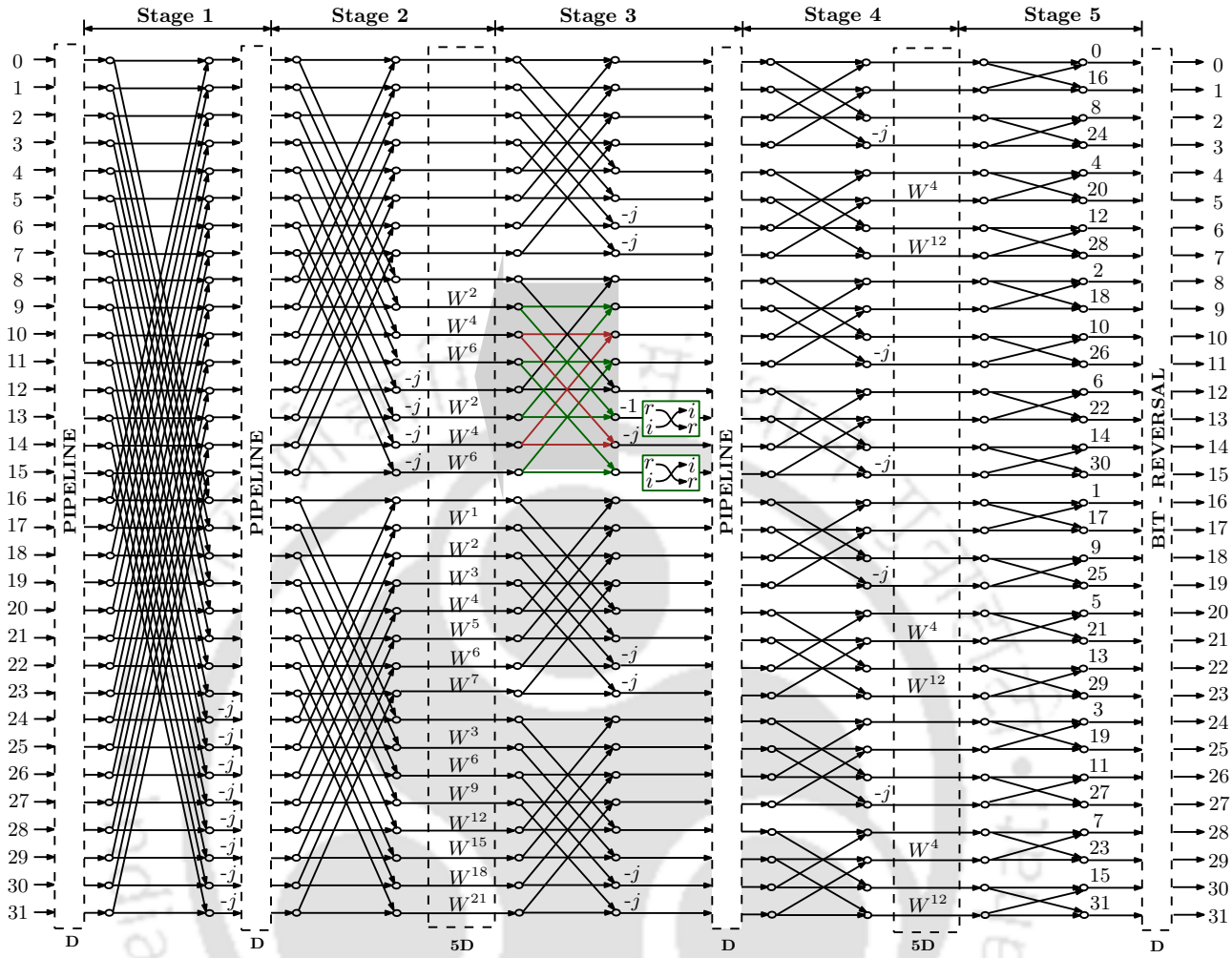


**Fig. 4.4.** Hardware architecture of proposed fully-parallel pipelined SRFFT

are obtained at the maximum clock frequency. The clock frequency indirectly influences the area of an ASIC through its impact on the critical path delay and the design optimizations made to achieve higher frequencies. The critical path delay in the proposed design consists of only one adder which helps to achieve higher clock frequency compared to [47]. For a fair comparison, we normalize the area and power by dividing them by the maximum clock frequency. This normalization allows the comparison to focus on the efficiency of the designs rather than their absolute values. From Table 4.4, it is found that the proposed design offers the least area and energy consumption compared to the existing fully-parallel designs [46] and [47]. Clearly, the proposed design outperforms the design in [46] which has an area of 3.05 $mm^2$/G transform/s on 45 nm, whereas the proposed design has an area of 0.32 $mm^2$/G transform/s on 65 nm. In case of energy-efficiency, the split-radix in [46] consumes 2.02 nJ/transform, whereas the proposed design consumes 0.12 nJ/transform. Compared to the radix-4

Table 4.4: Performance Comparison with State-of-the-Art Fully-Parallel FFT Architectures

| | [46] | [47] | | | | Proposed | | | |
|---|---|---|---|---|---|---|---|---|---|
| Technology | 45 nm | 65 nm | | | | 65 nm | | | |
| $N$ | 64 | 32 | 64 | 128 | 256 | 32 | 64 | 128 | 256 |
| $B$ | 15 | 9/10 | 9/11 | 9/11 | 9/11 | 16/16 | 16/16 | 16/16 | 16/16 |
| Voltage (V) | 1.1 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 | 1.2 |
| Clock freq. (MHz) | 200 | 750 | 500 | 500 | 500 | 990 | 980 | 877 | 877 |
| Area ($mm^2$) | 0.61 | 0.22 | 0.32 | 0.91 | 2.03 | 0.13 | 0.31 | 0.88 | 1.51 |
| Power (mW) | 404.39 | 140 | 210 | 570 | 1350 | 53.80 | 120.38 | 327.61 | 597.94 |
| Latency(cc) | 3 | 18 | 18 | 20 | 20 | 14 | 15 | 20 | 21 |
| Area-efficiency ($mm^2$/G transform/s) | 3.05 | 0.29 | 0.64 | 1.82 | 4.06 | 0.13 | 0.32 | 1.00 | 1.72 |
| Energy-efficiency (nJ/transform) | 2.02 | 0.18 | 0.41 | 1.14 | 2.7 | 0.05 | 0.12 | 0.37 | 0.68 |

B: Number of input/output bits in real/imaginary data, cc: clock cycles, $mm^2$/G transform/s: square millimeters per giga transform per second, Area-efficiency = $\frac{\text{Area}}{\text{Clock frequency}}$, Energy-efficiency = $\frac{\text{Power}}{\text{Clock frequency}}$

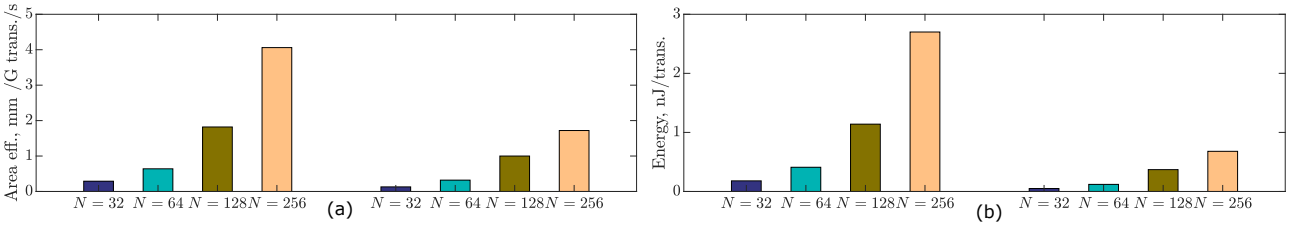


**Fig. 4.5.** Performance Comparison (a) Area-efficiency (b) Energy-efficiency with State-of-the-Art Fully-Parallel FFT Architectures

design in [47], the proposed design provides 45% to 57% area savings and 67% to 74% energy savings. This is mainly due to the use of substructure sharing scheme instead of dedicated real and imaginary CMs besides the less number of multipliers in SRFFT compared to radix-4. Also, because of the reduction in the multiplication stages for higher point FFTs, the number of pipelined registers required in each path decreases, and hence the power decreases considerably. In addition, the optimized CMs also contribute to the reduction in area and power consumption as discussed earlier. The design in [47] has a shorter wordlength of 9-bit input and 10-bit output compared to the proposed design which has fixed-width wordlength of 16-bit real and imaginary data. Moreover, the TF coefficients in [47] are being implemented using only two non-zero bits for a wordlength of 5-bits. In spite of about 60% larger wordlength compared to [47], the proposed design offers the least area and power consumption among the designs listed in Table 4.4. A comparison of area-efficiency and energy-efficiency is shown in Fig. 4.5.

## 4.4 Conclusion

In this work, a new approach to design a fully-parallel pipelined SRFFT has been presented. The proposed design has reduced the number of multiplication stages, and hence the number of required pipelined registers has reduced significantly as compared to a conventional fully-parallel pipelined SRFFT. The proposed design is found to be advantageous since it possesses low number of real multipliers alongwith their efficient implementation through architectural analysis and substructure sharing scheme. Synthesis results show that the proposed FFT offers the least area and power for implementing a fully-parallel pipelined FFT till date.

# 5

# Conclusions and Future Work

Contents

## 5.1 Conclusion

This thesis mainly investigates recent developments in the design of hardware architectures for FFT computation. The thesis presents design and implementation of different hardware-efficient architectures for low-complexity FFT realization to meet the real-time processing requirements of present-day applications. The main contributions of the thesis are summarized below.

Firstly, serial pipelined RFFT architectures suitable for low-throughput applications are proposed, with an emphasis on low-complexity implementation of multiplier. Methodology 1 eliminates the use of a post-processing stage to obtain a normal order output using a merged unit for butterfly computation and data re-ordering in the last stage. The merged unit and a quarter complex multiplier reduces the hardware complexity of the FFT. In Methodology 2, the flow graph is modified exclusively for serial implementation of the RFFT in order to take advantage of the fact that the RFFT has fewer operations than the CFFT. None of the existing serial RFFT architectures has taken this fact into consideration. A digit-serial multiplier is also presented. It results in reduced resources per multiplier. When this multiplier is extended for higher wordlengths, resource sharing has been observed.

Secondly, two memory-based RFFT architectures are proposed for applications where the area-constraints are high. Two new memory-accessing schemes corresponding to each design are proposed. Hardware reduction is achieved in terms of number of registers, multiplexers and multiplier complexity. It is found that although the number of clock cycles remains the same as the latest existing design, the computation time is reduced due to shorter clock period. This leads to a decrease in power consumption and improvement in the throughput.

Thirdly, a new approach to design a fully-parallel pipelined SRFFT is presented. The proposed design has reduced the number of multiplication stages, and hence the number of pipelined registers required has reduced significantly as compared to a conventional fully-parallel pipelined SRFFT. The proposed design is found to be advantageous since it possesses low number of real multipliers alongwith their efficient implementation through architectural analysis and substructure sharing scheme. ASIC results show that the proposed FFT offers the least area and power for implementing a fully-parallel pipelined FFT till date.

## 5.2 Suggestions for Future Research

In this thesis, a number of low-complexity FFT architectures are designed and implemented while maintaining the desired throughput and maximizing the utilization of existing resources. However, there exists considerable scope to extend this work in various directions for future investigation. This research work can be extended in the following directions:

(i) The thesis is primarily concentrated on design and implementation of low-complexity architectures for power-of-two (POT) FFT sizes using radix-2 and split-radix algorithms. However, in the future, non-POT FFTs might be preferred over POT FFTs owing to their optimized and significantly more developed realizations. Not much research has been carried out in this area. Because of this, most of the applications have to restrict themselves to POT FFTs even when it is not required.

(ii) As discussed in Chapter 4, with the advances in technology, there is a high demand for increasing throughput. With increased throughput, the power consumption of the devices would increase significantly, and definitely impact the battery life of wireless devices. There is only limited research work focusing entirely on reducing the power consumption. This could be a potential topic for research.

(iii) Majority of the power consumption in FFTs is due to the presence of the complex multipliers. New ways of implementing the complex multiplier can reduce the power consumption significantly. This has not been extensively studied and may gain importance in the upcoming times.

# Bibliography


[1] Shousheng He and Mats Torkelson. Design and implementation of a 1024-point pipeline FFT processor. In *Custom Integrated Circuits Conference, 1998. Proceedings of the IEEE 1998*, pages 131–134. IEEE, 1998.

[2] Mario Garrido, Nanda K Unnikrishnan, and Keshab K Parhi. A serial commutator fast Fourier transform architecture for real-valued signals. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(11):1693–1697, 2017.

[3] Mario Garrido, Konrad Möller, and Martin Kumm. World's fastest FFT architectures: Breaking the barrier of 100 GS/s. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(4):1507–1516, 2018.

[4] Saulo Queiroz, Joao Vilela, and Edmundo Monteiro. Is FFT fast enough for beyond 5G communications? A throughput-complexity analysis for OFDM signals. *IEEE Access*, PP:1–1, 01 2022.

[5] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[6] Samaneh Kouchaki, Saeid Sanei, Emma L Arbon, and Derk-Jan Dijk. Tensor based singular spectrum analysis for automatic scoring of sleep EEG. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23(1):1–9, 2014.

[7] Sai Sanjeet, Bibhu Datta Sahoo, and Keshab K Parhi. Low-energy real FFT architectures and their applications to seizure prediction from EEG. *Analog Integrated Circuits and Signal Processing*, pages 1–12, 2022.

[8] H V Sorensen, D Jones, Michael Heideman, and C Burrus. Real-valued fast Fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 35(6):849–863, 1987.

[9] Earl E Swartzlander, Wendell KW Young, and Saul J Joseph. A radix-4 delay commutator for fast Fourier transform processor implementation. *IEEE Journal of Solid-state Circuits*, 19(5):702–709, 1984.

[10] Earl E Swartzlander, Vijay K Jain, and Hiroomi Hikawa. A radix-8 wafer scale FFT processor. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 4:165–176, 1992.

[11] M.A. Richards. On hardware implementation of the split-radix fft. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(10):1575–1581, 1988.

[12] Pierre Duhamel. Implementation of "split-radix" FFT algorithms for complex, real, and real-symmetric data. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(2):285–295, 1986.

[13] Steven G. Johnson and Matteo Frigo. A modified split-radix FFT with fewer arithmetic operations. *IEEE Transactions on Signal Processing*, 55(1):111–119, 2007.

[14] Keshab K Parhi. *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 2007.

[15] Manohar Ayinala, Yingjie Lao, and Keshab K Parhi. An in-place FFT architecture for real-valued signals. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(10):652–656, 2013.

[16] Manohar Ayinala and Keshab K Parhi. FFT architectures for real-valued signals based on radix-$2^3$ and radix-$2^4$ algorithms. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 60(9):2422–2430, 2013.

[17] Aravinth Chinnapalanichamy and Keshab K Parhi. Serial and interleaved architectures for computing real FFT. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1066–1070. IEEE, 2015.

[18] Mario Garrido, Rikard Andersson, Fahad Qureshi, and Oscar Gustafsson. Multiplierless unity-gain SDF FFTs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(9):3003–3007, 2016.

[19] Erling H Wold and Alvin M. Despain. Pipeline and parallel-pipeline FFT processors for VLSI implementations. *IEEE Transactions on Computers*, 33(05):414–426, 1984.

[20] Song-Nien Tang, Jui-Wei Tsai, and Tsin-Yuan Chang. A 2.4-GS/s FFT processor for OFDM-based WPAN applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(6):451–455, 2010.

[21] Seung-Won Yang and Jong-Yeol Lee. Constant twiddle factor multiplier sharing in multipath delay feedback parallel pipelined FFT processors. *Electronics Letters*, 50(15):1050–1052, 2014.

[22] Guan Bi and EV Jones. A pipelined FFT processor for word-sequential data. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):1982–1985, 1989.

[23] Chao Cheng and Keshab K Parhi. High-throughput VLSI architecture for FFT computation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54(10):863–867, 2007.

[24] Manohar Ayinala, Michael Brown, and Keshab K Parhi. Pipelined parallel FFT architectures via folding transformation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(6):1068–1081, 2011.

[25] Mario Garrido, Jesús Grajal, MA Sanchez, and Oscar Gustafsson. Pipelined radix-$2^k$ feedforward FFT architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1):23–32, 2011.

[26] Mario Garrido, Keshab K Parhi, and Jesús Grajal. A pipelined FFT architecture for real-valued signals. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(12):2634–2643, 2009.

[27] Alan V Oppenheim. *Discrete-time Signal Processing*. Pearson Education India, 1999.

[28] Pramod Kumar Meher, Basant Kumar Mohanty, Sujit Kumar Patel, Soumya Ganguly, and Thambipillai Srikanthan. Efficient VLSI architecture for decimation-in-time fast Fourier transform of real-valued data. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(12):2836–2845, 2015.

[29] Shousheng He and Mats Torkelson. A new approach to pipeline FFT processor. In *Parallel Processing Symposium, 1996., Proceedings of IPPS'96, The 10th International*, pages 766–770. IEEE, 1996.

[30] Chia-Hsiang Yang, Tsung-Han Yu, and Dejan Markovic. Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE example. *IEEE Journal of Solid-state Circuits*, 47(3):757–768, 2012.

[31] LG Johnson. Conflict free memory addressing for dedicated FFT hardware. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 39(5):312–316, 1992.

[32] Byung G Jo and Myung Hoon Sunwoo. New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(5):911–919, 2005.

[33] Pei-Yun Tsai and Chung-Yi 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*, 19(12):2290–2302, 2010.

[34] Dionysios Reisis and Nikolaos Vlassopoulos. Conflict-free parallel memory accessing techniques for FFT architectures. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(11):3438–3447, 2008.

[35] Shuenn-Yuh Lee, Chia-Chyang Chen, Chyh-Chyang Lee, and Chih-Jen Cheng. A low-power VLSI architecture for a shared-memory FFT processor with a mixed-radix algorithm and a simple memory control scheme. In *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4–pp. IEEE, 2006.

[36] Chen-Fong Hsiao, Yuan Chen, and Chen-Yi Lee. A generalized mixed-radix algorithm for memory-based FFT processors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(1):26–30, 2010.

[37] Jienan Chen, Jianhao Hu, Shuyang Lee, and Gerald E Sobelman. Hardware efficient mixed radix-25/16/9 FFT for LTE systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(2):221–229, 2015.

[38] Xin Xiao, Erdal Oruklu, and Jafar Saniie. An efficient FFT engine with reduced addressing logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 55(11):1149–1153, 2008.

[39] Hsin-Fu Luo, Yi-Jun Liu, and Ming-Der Shieh. Efficient memory-addressing algorithms for FFT processor design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2162–2172, 2015.

[40] Jinqi Liu, Qianjian Xing, Xiaobo Yin, Xiubin Mao, and Feng Yu. Pipelined architecture for a radix-2 fast Walsh–Hadamard–Fourier transform algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(11):1083–1087, 2015.

[41] Zhen-Guo Ma, Xiao-Bo Yin, and Feng 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*, 62(9):876–880, 2015.

[42] Xiu-Bin Mao, Zhen-Guo Ma, Feng Yu, and Qian-Jian Xing. A continuous-flow memory-based architecture for real-valued FFT. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(11):1352–1356, 2017.

[43] Sungjin Park and Dongsuk Jeon. A modified serial commutator architecture for real-valued fast Fourier transform. In *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2020.

[44] Hongji Fang, Bo Zhang, Feng Yu, Bei Zhao, and Zhenguo Ma. A pipelined algorithm and area-efficient architecture for serial real-valued FFT. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(11):4533–4537, 2022.

[45] Gokhan Polat, Sitki Ozturk, and Mehmet Yakut. Design and implementation of 256-point radix-4 100 Gbit/s FFT algorithm into FPGA for high-speed applications. *ETRI Journal*, 37(4):667–676, 2015.

[46] Adnan Rauf, Muhammad Adeel Pasha, and Shahid Masud. Towards design and automation of a scalable split-radix FFT processor for high throughput applications. *Microprocessors and Microsystems*, 65:148–157, 2019.

[47] Seyed Hadi Mirfarshbafan, Sueda Taner, and Christoph Studer. SMUL-FFT: A streaming multiplierless fast Fourier transform. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1715–1719, 2021.

[48] Axel Wenzler and Ernst Luder. New structures for complex multipliers and their noise analysis. In *1995 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 2, pages 1432–1435. IEEE, 1995.

[49] Hong-An Huang, Yen-Chin Liao, and Hsie-Chia Chang. A self-compensation fixed-width booth multiplier and its 128-point fft applications. In *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4 pp.–3541, 2006.

[50] Young Eun Kim, J.O. Yoon, K.J. Cho, J.G. Chung, S.I. Cho, and S.S. Choi. Efficient design of modified booth multipliers for predetermined coefficients. In *2006 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4 pp.–, 2006.

[51] Martin Kumm, Shahid Abbas, and Peter Zipf. An efficient softcore multiplier architecture for xilinx FPGAs. In *2015 IEEE 22nd Symposium on Computer Arithmetic*, pages 18–25. IEEE, 2015.

[52] E George Walters. Array multipliers for high throughput in xilinx fpgas with 6-input luts. *Computers*, 5(4):20, 2016.

[53] Florent de Dinechin, Silviu-Ioan Filip, Martin Kumm, and Luc Forget. Table-based versus shift-and-add constant multipliers for FPGAs. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pages 151–158. IEEE, 2019.

[54] Giovanni Betta, Consolatina Liguori, and Antonio Pietrosanto. A multi-application FFT analyzer based on a DSP architecture. *IEEE Transactions on Instrumentation and Measurement*, 50(3):825–832, 2001.

[55] Alessandra Flammini, Daniele Marioli, Emiliano Sisinni, and Andrea Taroni. A multichannel DSP-based instrument for displacement measurement using differential variable reluctance transducer. *IEEE Transactions on Instrumentation and Measurement*, 54(1):178–183, 2005.

[56] Ediz Cetin, Richard CS Morling, and Izzet Kale. An extensible complex fast Fourier transform processor chip for real-time spectrum analysis and measurement. *IEEE Transactions on Instrumentation and Measurement*, 47(1):95–99, 1998.

[57] Damiano Crescini, Alessandra Flammini, Daniele Marioli, and Andrea Taroni. Application of an FFT-based algorithm to signal processing of LVDT position sensors. *IEEE Transactions on Instrumentation and Measurement*, 47(5):1119–1123, 1998.

[58] Manohar Ayinala and Keshab K Parhi. Parallel-pipelined radix-$2^2$ FFT architecture for real valued signals. In *2010 Conference Record of the Forty Fourth Asilomar Conference on Signals, Systems and Computers*, pages 1274–1278. IEEE, 2010.

[59] Mario Garrido. A survey on pipelined FFT hardware architectures. *Journal of Signal Processing Systems*, pages 1–20, 2021.

[60] G Bergland. A radix-eight fast Fourier transform subroutine for real-valued series. *IEEE Transactions on Audio and Electroacoustics*, 17(2):138–144, 1969.

[61] Mario Garrido, Shen-Jui Huang, Sau-Gee Chen, and Oscar Gustafsson. The serial commutator FFT. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(10):974–978, 2016.

[62] Xin Xiao, Erdal Oruklu, and Jafar Saniie. Reduced memory architecture for CORDIC-based FFT. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 2690–2693. IEEE, 2010.

[63] Wei-Hsin Chang and Truong Q Nguyen. On the fixed-point accuracy analysis of FFT algorithms. *IEEE Transactions on Signal Processing*, 56(10):4673–4682, 2008.

[64] Pankaj Gupta. Accurate performance analysis of a fixed point FFT. In *2016 Twenty Second National Conference on Communication (NCC)*, pages 1–6. IEEE, 2016.

[65] Nanda K Unnikrishnan, Mario Garrido, and Keshab K Parhi. Effect of finite word-length on SQNR, area and power for real-valued serial FFT. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[66] Abdelmohsen Ali and Walaa Hamouda. Generalized FFT-based one-bit quantization system for wideband spectrum sensing. *IEEE Transactions on Communications*, 68(1):82–92, 2019.

[67] E Bidet, D Castelain, C Joanblanq, and P Senn. A fast single-chip implementation of 8192 complex point FFT. *IEEE journal of Solid-state Circuits*, 30(3):300–305, 1995.

[68] Xia Feng, Jia Ye, Lianshan Yan, Jianwei Luo, Peng Li, Wei Pan, Xihua Zou, and Bin Luo. Improving spectral efficiency of digital radio-over-fiber transmission using two-dimensional discrete cosine transform with vector quantization. *Optics Express*, 29(16):25868–25875, 2021.

[69] Jeng-Shyang Pan, Chiou-Yng Lee, Anissa Sghaier, Medien Zeghid, and Jiafeng Xie. Novel systolization of subquadratic space complexity multipliers based on toeplitz matrix–vector product approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(7):1614–1622, 2019.

[70] Chiou-Yng Lee and Jiafeng Xie. Digit-serial versatile multiplier based on a novel block recombination of the modified overlap-free Karatsuba algorithm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(1):203–214, 2018.

[71] Shen-Fu Hsiao, Jian-Ming Chen, Yu-Hong Chen, Hung-Ching Li, and Yi Hsu. Comparison of digit-serial and bit-level designs for acceleration of convolutional neural network computation. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2021.

[72] Henry Samueli. An improved search algorithm for the design of multiplierless FIR filters with powers-of-two coefficients. *IEEE Transactions on Circuits and Systems*, 36(7):1044–1047, 1989.

[73] Shiann-Rong Kuang, Jiun-Ping Wang, and Cang-Yuan Guo. Modified Booth multipliers with a regular partial product array. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 56(5):404–408, 2009.

[74] Yajuan He and Chip-Hong Chang. A new redundant binary Booth encoding for fast $2^n$-bit multiplier design. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(6):1192–1201, 2008.

[75] Duncan JM Moss, David Boland, and Philip HW Leong. A two-speed, radix-4, serial-parallel multiplier. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(4):769–777, 2019.

[76] Yen-Jen Chang, Yu-Cheng Cheng, Shao-Chi Liao, and Chun-Huo Hsiao. A low power radix-4 Booth multiplier with pre-encoded mechanism. *IEEE Access*, 8:114842–114853, 2020.

[77] Essam Elsayed and Hatem El-Boghdadi. Area-efficient digit serial-serial two's complement multiplier. *Journal of Circuits, Systems, and Computers*, 23(07):1450099, 2014.

[78] Essam Elsayed and Hatem M El-Boghdadi. A novel power-efficient multi-operand digit-multiplier using reconfiguration and clock gating. *The Journal of Supercomputing*, 71(7):2539–2564, 2015.

[79] Pramod Kumar Meher. LUT optimization for memory-based computation. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 57(4):285–289, 2010.

[80] Mario Garrido, Jesús Grajal, and Oscar Gustafsson. Optimum circuits for bit reversal. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 58(10):657–661, 2011.

[81] TSMC 90 nm general-purpose CMOS standard cell libraries−tcbn90ghp. [Online] Available: `www.tsmc.com/`.

[82] Anantha P Chandrakasan and Robert W Brodersen. Minimizing power consumption in digital CMOS circuits. *Proceedings of the IEEE*, 83(4):498–523, 1995.

[83] Mohd Tasleem Khan and Rafi Ahamed Shaik. Optimal complexity architectures for pipelined distributed arithmetic-based LMS adaptive filter. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(2):630–642, 2018.

[84] Valery Sklyarov, Iouliia Skliarova, Alexander Barkalov, and Larysa Titarenko. *Synthesis and Optimization of FPGA-based Systems*, volume 294. Springer Science & Business Media, 2014.

[85] Krishna Praveen Yalamarthy, Saurabh Dhall, Mohd Tasleem Khan, and Rafi Ahamed Shaik. Low-complexity distributed-arithmetic-based pipelined architecture for an LSTM network. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(2):329–338, 2019.

[86] Jagrit Kathuria, M Ayoubkhan, and Arti Noor. A review of clock gating techniques. *MIT International Journal of Electronics and Communication Engineering*, 1(2):106–114, 2011.

[87] Israel Koren. *Computer Arithmetic Algorithms*. AK Peters/CRC Press, 2018.

[88] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee. A 1-GS/s FFT/IFFT processor for UWB applications. *IEEE Journal of Solid-state Circuits*, 40(8):1726–1735, 2005.

[89] Yinghui Tian, Yong Hei, Zhizhe Liu, Qi Shen, Zhixiong Di, and Tao 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*, 66(1):106–110, 2019.

[90] Kai-Feng Xia, Bin Wu, Tao Xiong, and Tian-Chun Ye. A memory-based FFT processor design with generalized efficient conflict-free address schemes. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(6):1919–1929, 2017.

[91] Mario Garrido, Miguel Ángel Sánchez, María Luisa López-Vallejo, and Jesús Grajal. A 4096-point radix-4 memory-based FFT using DSP slices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1):375–379, 2016.

[92] Shaohan Liu and Dake 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*, 27(3):511–523, 2018.

[93] Hsin-Fu Luo, Yi-Jun Liu, and Ming-Der Shieh. Efficient memory-addressing algorithms for FFT processor design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10):2162–2172, 2014.

[94] Qianjian Xing, Zhenguo Ma, and Yingke Xu. A novel conflict-free parallel memory access scheme for FFT processors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017.

[95] Minhyeok Shin and Hanho Lee. A high-speed four-parallel radix-$2^4$ FFT/IFFT processor for UWB applications. In *2008 IEEE International Symposium on Circuits and Systems*, pages 960–963. IEEE, 2008.

[96] Mario Garrido. Evolution of the performance of pipelined FFT architectures through the years. In *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2020.

[97] T Sansaloni, A Pérez-Pascual, V Torres, and Javier Valls. Scheme for reducing the storage requirements of FFT twiddle factors on FPGAs. *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 47(2):183–187, 2007.

# List of Publications

### International Peer Reviewed Journals

1. **J. Hazarika**, M. T. Khan, S. R. Ahamed and H. B. Nemade, "An efficient implementation approach to FFT processor for spectral analysis", *IEEE Transactions on Instrumentation and Measurement*, vol. 72, 2023.

2. **J. Hazarika**, S. R. Ahamed and H. B. Nemade, "Low-complexity, energy-efficient fully-parallel split-radix FFT architecture", *IET Electronics Letters*, vol. 58, no. 18, 2022.

3. **J. Hazarika**, M. T. Khan, S. R. Ahamed and H. B. Nemade, "Energy efficient VLSI architecture of real-valued serial pipelined FFT", *IET Computers and Digital Techniques*, vol. 13, no. 6, 2019.

### Conference Publications

1. **J. Hazarika**, M. T. Khan, S. R. Ahamed and H. B. Nemade, "An area and power efficient serial commutator FFT with recursive LUT multiplier", *International Conference on Modelling, Simulation and Intelligent Computing (MoSICom) 2020*, **Dubai**, UAE.

2. **J. Hazarika**, M. T. Khan, S. R. Ahamed and H. B. Nemade, "High performance multiplierless serial pipelined VLSI architecture for real-valued FFT", *25th National Conference on Communication (NCC), IISc Bengaluru, Feb. 2019.* [**Nominated for Best Paper Award**]

3. **J. Hazarika**, M. T. Khan and S. R. Ahamed, "Low-complexity continuous-flow memory-based FFT architectures for real-valued signals", *2019 32nd International Conference on VLSI Design and 18th International Conference on Embedded Systems (VLSID), Delhi, Jan. 2019.* [**Best Student Paper Award**].