Low-Power Architectures for Signal Processing and Classification Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Manohar Ayinala

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Keshab Parhi

August, 2012

Acknowledgements

First of all, I wish to thank my advisor, Professor Keshab K. Parhi, for his continuing encouragement, guidance and financial support throughout my Ph.D. study at University of Minnesota. I would also like to thank Professor Gerald Sobelman, Professor Tay Netoff, and Professor Sachin Sapatnekar for their support as members of my Ph.D. committee. Their comments and suggestions helped me improve my thesis.

I would like to thank Prof. Chris Kim and his students for the technology libraries and tools support used in this research. I would like to thank the Graduate School for their financial support with Doctoral Dissertation Fellowship. I would also like to thank Minnesota Supercomputing Institute for their support with computing resources. Part of this work on FFT architectures is carried out at Leanics Corporation, Minneapolis.

I would like to give special thanks to my parents and sister for their love, encouragement and constant support to continue my studies because without them this work would not have been possible.

My thanks also go to current and former members of our research group. Particularly, Dr. Yun Sang Park, Dr. Renfei Liu for our numerous discussions on various research topics. Also, I am grateful to Dr. Aaron Cohen, Chuan Zhang, Sohini Roychowdury, Bo Yuan for their support and encouragement during my Ph.D.

Additionally, I would like to thank my friends Prasanth Ganta, Raviteja Pavuluri and erie group in Minneapolis especially Krishna, Vivek, Narayanarao, Ravali, Srikar, Prathyusha for their special assistance and encouragement to continue my studies.

Abstract

Digital signal processing and classification algorithms play a crucial role in modern day biomedical monitoring systems. Fortunately, emerging sensors and stimulators as well as specialized networking technologies have enabled biomedical devices to advance to new frontiers. Deep-brain stimulators, for instance, offer unprecedented modalities for delivering therapy to patients affected by neurological conditions, ranging from Parkinson's disease to epilepsy; out-patient monitoring networks raise the possibility of comprehensive yet cost-scalable healthcare delivery over large populations with increasingly diverse disease states. The central need, as these systems advance towards intelligent, closed-loop operation, is the ability to detect specific physiological states of interest from signals that are available through sensors. A key challenge in closed-loop biomedical systems is the ability to detect complex physiological states from the patient data within a constrained power budget. Signal processing and data-driven machine learning techniques are major enablers for modeling and detection of such states. However, the computational power scales with the complexity of models required.

This thesis considers the VLSI implementation of basic signal processing techniques such as fast Fourier transform (FFT), power spectral density (PSD) computation. Reconfigurable architectures for classification algorithms including support vector machines (SVM) and Adaboost are also presented. The proposed architectures improve performance and reduce area/power consumption.

First, we present a novel methodology to design parallel pipelined FFT architectures using folding transformation and register minimization techniques. Novel parallel-pipelined architectures for the computation of complex valued fast Fourier transform are derived. The proposed architectures overcome prior bottlenecks and achieve full hardware utilization. The operating frequency of the proposed architecture can be decreased which in turn reduces the power consumption. This significantly reduces power at same speed or increases speed at same power consumption level. The power consumption can be reduced up to 37% in 2-parallel architectures. Further, we propose a novel approach to develop pipelined fast Fourier transform (FFT) architectures for real-valued signals. Novel 2-parallel and 4-parallel architectures are presented for radix-2^3

and radix-2^4 algorithms. The proposed radix-2^3 and radix-2^4 architectures lead to low hardware complexity compared to a prior RFFT architecture.

We propose an efficient architecture for memory-based in-place FFT/IFFT computation. A conflict-free memory addressing scheme is proposed to ensure the continuous operation of the FFT processor. The proposed architecture requires fewer computation cycles along with the low hardware cost compared to prior work. We then present a low-complexity algorithm and architecture to compute power spectral density (PSD) using the Welch method. The complexity reduction comes at the cost of slight performance loss in accuracy due to the approximation used for the implementation of the fractional delay filter. The performance loss is 6-8% using fractional delay filter with 2-3 multipliers. A novel architecture is presented based on the proposed algorithm which consumes 33% less energy compared to the original method.

We propose a low-energy reconfigurable architecture for support vector machines (SVMs) based on *approximate computing* by exploiting the inherent error resilience in the computation. We present two design optimizations, fixed-width multiply-add and non-uniform look-up table (LUT) for exponent function to minimize power consumption and hardware complexity while retaining the classification performance. The proposed design consumes 31% less energy on average compared to a conventional design. Finally, we present a novel low-complexity patient-specific algorithm for seizure prediction using spectral power features. The proposed algorithm achieves a sensitivity of 94.375% for a total of 71 seizure events with a low false alarm rate of 0.13 per hour and 6.5% of time spent in false alarms using an average of 5 features for the Freiburg database. The low computational complexity of the proposed algorithm makes it suitable for an implantable device.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Design of intelligent medical monitoring devices that provides real-time feedback to the patient is an increasingly important application in healthcare industry. A patient can wear the device during normal daily activity, allowing medical staff to obtain a much clearer view of the patient's condition than is available from short periods of monitoring in the hospital or doctor's office. In recent years, a number of promising clinical prototypes of implantable and wearable monitoring devices have started to emerge. If we imagine an embedded wearable terminal (or implantable in certain cases) that can monitor the data collected by these sensors through appropriate signal processing algorithms and then provide warning signs of abnormalities, then several fatalities can be avoided. Fig. 1.1 shows two examples of implantable devices.

These kinds of monitoring systems require merging of sensing and stimulation systems together to arrive at a bi-directional interface or a closed loop system. Some applications of bi-directional interfaces that prove beneficial include closed-loop epilepsy control based on seizure detection, dynamic titration of a Parkinson's deep brain stimulation (DBS) device based thalamo-cortical oscillation state, and sensory feedback for motor prosthesis [1]. The challenge with these systems is to balance all of the system constraints to make the design practical, safe and effective. Fig. 1.2 shows an example of a closed-loop system for epilepsy.

The main requirement in the closed-loop systems is the ability to detect/predict

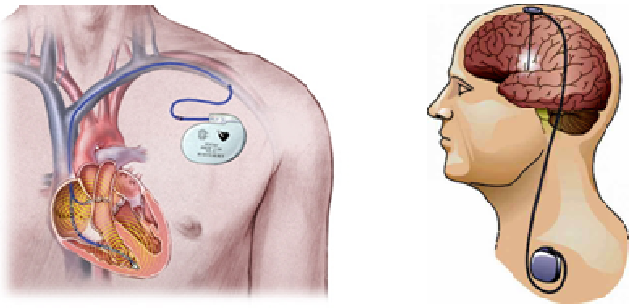


Figure 1.1: Examples of implantable/wearable devices. Pacemaker (left) and Deep brain stimulator (right).

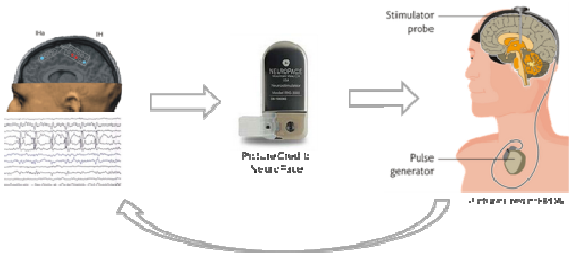


Figure 1.2: Closed-loop: monitor, predict/detect, and suppress seizures.

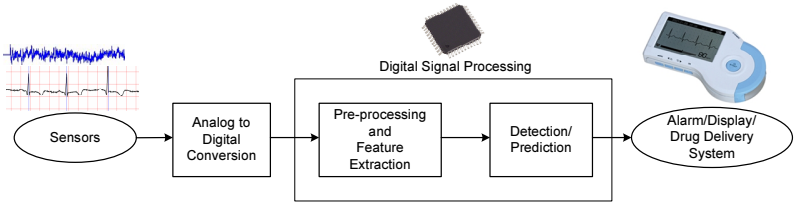


Figure 1.3: Block diagram of an implantable/wearable system.

the abnormalities from signals that are available through the sensors. This poses two essential challenges: (1) the signal correlations to clinically relevant states are generally too complex to model, and (2) these correlations often varies from patient to patient [2]. Machine learning based techniques can lead to powerful approaches to overcome these challenges. This has been prompted by the recent availability of sensors data in the healthcare domain as well as the development of machine learning techniques for modeling specific correlations in the data and efficiently applying these models [3]. However, the computations involved must be achieved at very low-power levels (e.g., 1-10mW for wearable devices and 10-100μW for implantable devices).

1.2 Summary of Contributions

In typical, patient-monitoring applications, bio-signals such as bio-potentials are acquired by sensors attached to a patient's body, and sent to a nearby intermediate terminal for processing. Most of the processing will be done on remote terminals [4], [5]. With the advancement of signal processing and VLSI technology, the processing can be done at the sensor node in the wearable/implantable systems. Low-power consumption and high sensitivity and specificity are the main requirements in a wearable/implantable monitoring system. Fig. 1.3 shows a generic implantable/wearable system with real-time processing. The tradeoffs involved in low-power architectures for biomedical monitoring applications have not yet been fully explored. To this end, this thesis will address the design of low-power architectures for signal processing and classification algorithms for biomedical applications.

Our main contributions can be classified under four categories: FFT computation, PSD computation, SVM computation and an application of these in design of a seizure

prediction system. The first three categories represent the feature computation and classification steps of a general biomedical monitoring system.

1.2.1 FFT Computation

Complex FFT

Fast Fourier transform (FFT) is widely used for biomedical signal analysis to compute the discrete Fourier transform (DFT). Numerous architectures for FFT have been proposed in the literature [18]-[22]. A formal method of developing these architectures from the algorithms is not well established. Further, most of these hardware architectures are not fully utilized and require high hardware complexity. High-throughput and low-power designs are required to meet the speed and power requirements while keeping the hardware overhead to a minimum. Therefore, in this thesis we have proposed a new approach to design these architectures from the FFT flow graphs using folding transformation [11]. Several novel architectures are also developed using the proposed methodology which have not been presented in the literature before.

The approach based on use of decimation-in-time algorithms reduce the number of delay elements by 33% compared to the decimation-in-frequency based designs. The number of delay elements required for an N-point FFT architecture is $N-4$ which is comparable to that of delay feedback schemes. The number of complex adders required is only 50% of those in the delay feedback designs. The proposed approach can be extended to any radix-2^n based FFT algorithms. The proposed architectures are feed-forward designs and can be pipelined by more stages to increase the throughput. Further, a novel four parallel 128-point FFT architecture is derived using the proposed approach.

Real FFT

Further, 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 [43], [44], a specialized hardware implementation is best suited to meet the real-time constraints. This type of implementation also saves power in implantable or portable devices which is a key constraint.

Even though specific algorithms for the computation of the RFFT [45] - [50] have

been proposed in the past, these algorithms lack regular geometries to design pipelined architectures. These approaches are based on removing the redundancies of the complex FFT when the input is real and can be efficiently used in in-place architectures [51] or digital signal processors (DSP). We proposed a novel general methodology for designing pipelined FFT architectures for real-valued signals. The proposed methodology is based on modifying the flow graph of the FFT algorithm such that it has both real and complex datapaths. The imaginary parts of the computations replace the redundant operations in the modified flowgraph. New butterfly structures are designed to handle the hybrid datapaths. The proposed hybrid datapath leads to a general approach which can be extended to all radix-2^n based FFT algorithms. Further, architectures with arbitrary level of parallelism can be derived using the folding methodology. Novel 2-parallel and 4-parallel architectures are presented for radix-2^3 and radix-2^4 algorithms. The proposed architectures maximize the utilization of hardware components with no redundant computations. The proposed architectures based on radix-2^3 and radix-2^4 algorithms require less multiplication complexity compared to prior RFFT architectures.

In-Place FFT

We propose an efficient architecture for memory-based in-place FFT/IFFT computation for real-valued signals. The proposed computation is based on a modified radix-2 algorithm, which removes the redundant operations from the flow graph. A new processing element is proposed using two radix-2 butterflies which can process four inputs in parallel. A conflict-free memory addressing scheme is proposed to ensure the continuous operation of the FFT processor. Further, the addressing scheme is extended to support parallel processing elements. The proposed real-FFT processor requires fewer computation cycles along with the low hardware cost compared to prior work.

1.2.2 PSD Computation

We also propose a low-complexity algorithm and architecture to compute power spectral density (PSD) using the Welch method. The Welch algorithm provides a good estimate of the spectral power at the cost of high computational complexity. We propose a new modified approach to reduce the computational complexity of the Welch PSD computation for a 50% overlap. In the proposed approach, an $N/2$-point FFT

is computed, where N is the length of the window and is merged with the FFT of the previous $N/2$-point to generate an N-point FFT of the overlapped segment. This requires replacing the windowing operation as a convolution in the frequency domain. Fortunately, the frequency domain filtering requires a symmetric 3-tap or 5-tap filter FIR filter for raised cosine windows. The proposed method needs to compute $(L+1)$ $N/2$-point FFTs instead of L N-point FFTs, where L is the number of overlapping segments.

In the proposed FFT merging approach, the even samples are computed exactly, while the odd samples require a shift by a half sample delay and are estimated using a fractional-delay filter. The complexity reduction comes at the cost of slight performance loss due to the approximation used for the implementation of the fractional delay filter. The performance loss is 6-8% using fractional delay filter with 2-3 multipliers. A novel architecture is presented based on the proposed algorithm. The proposed architecture is estimated to consume 33% less energy compared to the original method. Further a low-complexity architecture is presented to compute a special case of the short-time Fourier transform based on the proposed PSD computation algorithm.

1.2.3 SVM Computation

A wide variety of classification algorithms exist in the literature including artificial neural networks (ANN), linear discriminant analysis (LDA), Bayes' classifier etc. Support Vector Machines (SVM) is one such popular machine learning classifier that can be efficiently trained offline to derive the support vectors. SVMs provide good generalization performance for a wide range of regression and classification tasks based on the structural risk minimization induction principle [70], [71]. In recent years, SVMs have been effectively used as a classification tool in a wide range of problems including pattern recognition, image analysis, communications, and biomedical signal analysis [8] - [10]. While SVM training can be considered as an offline task, the classification is mostly performed in real-time on newly obtained data. Face detection, speech recognition, biomedical signal analysis require online classification and have real-time constraints. However, the SVM classification is a computationally expensive task, and is linearly dependent on the classification data load, the population of the support vectors and the problem's dimensionality.

SVM architectures employing different kernels have been proposed in the recent literature either for FPGA or ASIC. Not much research has been published on optimizing the implementation of radial basis function (RBF) kernel either for area or energy minimization. SVM computations are inherently error resilient as the decision function depends on the sign but not on the magnitude of the final value. Therefore, we propose to reduce the precision of the computations for energy minimization which has not been exploited earlier in the case of SVM design. We propose a general-purpose architecture for SVM computation that takes advantage of inherent error resiliency of the SVM algorithms. We consider the common SVM kernels used in biomedical signal analysis and multimedia recognition applications and propose design techniques and optimizations to minimize power consumption.

We present two design optimizations, fixed-width multiply-add and non-uniform look-up table (LUT) for an exponent function to minimize power consumption and hardware complexity while retaining the classification performance. A novel non-uniform quantization scheme is proposed for implementing the exponent function which reduces the size of the look-up table by 50%. The proposed non-uniform look-up table reduces the power consumption by 35% using 10-bit quantization. The proposed architecture is programmable and can evaluate three different kernels (linear, polynomial, radial basis function (RBF)). The proposed design consumes 31% less energy on average compared to a conventional design. We demonstrate that SVM computation using RBF kernel can be performed in 109.2nJ for 36 features and 5000 support vectors with 0.5V Vdd using 65nm technology.

1.2.4 Seizure Prediction

We propose a novel low-complexity patient-specific algorithm for seizure prediction based on spectral power features. Adaboost algorithm is used in two stages of the algorithm: feature selection and classification. The algorithm extracts spectral power features in 9 different sub-bands from the electroencephalogram (EEG) recordings. We have proposed a new feature ranking method to rank the features. The key (top ranked) features are used to make a prediction on the seizure event. Further, to reduce the complexity of classification stage, a non-linear classifier is built based on the Adaboost algorithm using decision stumps (linear classifier) as the base classifier. A non-linear

decision function is built using a combination of linear decision functions (in general linear decision functions are less computationally complex). The proposed algorithm uses several linear classifiers and are combined using the Adaboost algorithm. The computational complexity of the classifier depends on the number of iterations (T, also the number of linear classifiers used) needed to achieve the required performance. The proposed method only requires comparison and addition operations, and does not require multiplications at all.

The proposed algorithm achieves a sensitivity of 94.375% for a total of 71 seizure events with a low false alarm rate of 0.13 per hour and 6.5% of time spent in false alarms using an average of 5 features for the Freiburg database. The low computational complexity of the proposed algorithm makes it suitable for an implantable device.

1.3 Outline of the Thesis

The thesis is outlined as follows. Fast Fourier transform (FFT) is introduced in Chapter 2. Afterwards, we introduce novel methodology to design efficient parallel-pipelined FFT architectures using folding technique. Then novel pipelined architectures for FFT computation for complex-valued signals are described.

Chapter 3 introduces the design of FFT architectures for real-valued signals. A novel methodology is presented to modify the data flow graph. Then novel pipelined FFT architectures for real-valued signals are described.

Chapter 4 introduces the idea of memory-based FFT architectures. Afterwards, a novel conflict-free addressing scheme is described. Then the design of novel FFT processors for both complex and real-valued signals are discussed.

Chapter 5 introduces the computation of power spectral density (PSD). A novel low-complexity PSD computation is presented based on the Welch method. Afterwards, a low-complexity architecture for PSD computation is described.

Chapter 6 introduces the theory of support vector machines (SVM). A low-energy reconfigurable architecture for SVM computation is discussed.

In Chapter 7, background on seizure prediction is discussed. A low-complexity algorithm for seizure prediction using spectral power features is described. Adaboost algorithm is used to reduce the complexity in the classification step.

Finally, Chapter 8 concludes with a summary of total contributions of this thesis and future research directions.

Chapter 2

FFT Architectures for complex inputs

In this chapter, we present numerous FFT architectures for complex signals. Section 2.2 provides the mathematical background behind the fast Fourier transform for various radices. Section 2.3 discusses the prior work on FFT architectures for complex signals. In Section 2.4, the proposed methodology to design FFT architectures using folding transformation is described. Section 2.5 and Section 2.6 presents the various architectures developed based on decimation-in-frequency and decimation-in-time algorithms, respectively. The circuits to reorder the output samples are presented in Section 2.7. Finally, Section 2.8 compares the proposed architectures with prior designs.

2.1 Introduction

Fast Fourier Transform (FFT) is widely used in the field of digital signal processing (DSP) such as filtering, spectral analysis etc., to compute the discrete Fourier transform (DFT). FFT plays a critical role in modern digital communications such as digital video broadcasting and orthogonal frequency division multiplexing (OFDM) systems. The FFT core is one of the modules having high computational complexity in the physical layer of these communication systems. Therefore, we will present different FFT algorithms and explore efficient techniques for implementing the FFT computation.

2.2 FFT Algorithms

In this section, radix-2, radix-2^2 and radix-2^3 algorithms are reviewed and comparison between these algorithms is made in terms of multiplications required. The N-point Discrete Fourier Transform (DFT) of a sequence $x[n]$ is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \tag{2.1}$$

where $W_N^{nk} = e^{-j(2\pi/N)nk}$.

The FFT includes a collection of algorithms that reduce the number of operations of the DFT. The Cooley-Tukey algorithm [12] is the most used among them. It is based on decomposing the DFT in $n = log_r N$ stages, where r is the radix. This achieves a reduction on the number of operations from order $O(N^2)$ in the DFT to order $O(NlogN)$ in the FFT. The decomposition can be carried out in different ways. The most common ones are the Decimation in Time (DIT) and the Decimation in Frequency (DIF).

2.2.1 Radix-2

For radix-2, the DIF decomposition separates the output sequence $X[k]$ into even and odd samples. The following equations are obtained:

$$\begin{aligned} X[2r] &= \sum_{n=0}^{N/2-1} (x[n] + x[n+N/2]) e^{-j\frac{2\pi}{N/2}rn}, r = 0, 1..., N/2-1 \\ X[2r+1] &= \sum_{n=0}^{N/2-1} (x[n] - x[n+N/2]) e^{-j2\pi/Nn} e^{-j\frac{2\pi}{N/2}rn}, r = 0, 1..., N/2-1 \end{aligned} \tag{2.2}$$

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

2.2.2 Radix-2^2

The radix-2^2 FFT algorithm is proposed in [13]. We can derive the algorithm by using the following new indices,

$$\begin{aligned} n &= < \frac{N}{2} n_1 + \frac{N}{4} n_2 + n_3 >_N \\ k &= < k_1 + 2k_2 + 4k_3 >_N \end{aligned} \tag{2.3}$$

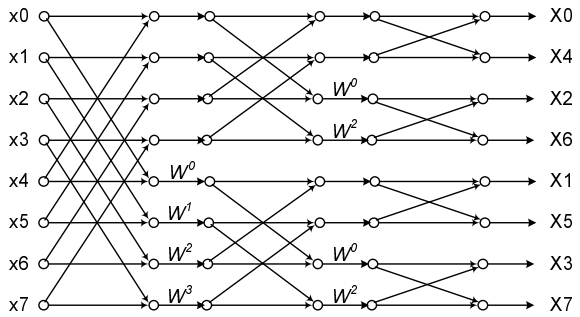


Figure 2.1: Flow graph of a radix-2 8-point DIF FFT.

Substituting (2.3) in (5.3), we get

$$\begin{aligned} & X(k_1 + 2k_2 + 4k_3) \\ & = \sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} x(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3) W_N^{nk} \end{aligned} \tag{2.4}$$

By decomposing the twiddle factor, we get

$$W_N^{nk} = (-j)^{n_2(k_1+2k_2)} W_N^{n_3(k_1+2k_2)} W_N^{4n_3k_3} \tag{2.5}$$

Substituting (2.5) in (2.4) and expanding the summation with indices n_1, n_2, and we get,

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1} [H_{\frac{N}{4}}(n_3, k_1, k_2) W_N^{n_3(k_1+2k_2)}] W_{\frac{N}{4}}^{n_3k_3} \tag{2.6}$$

where $H(k_1, k_2, n_3)$ is expressed as,

$$\begin{aligned} & H_{\frac{N}{4}}(n_3, k_1, k_2) = B_{\frac{N}{2}}(n_3, k_1) + (-j)^{(k_1+2k_2)} B_{\frac{N}{2}}(n_3 + \frac{N}{4}, k_1) \\ & B_{\frac{N}{2}}(n_3, k_1) = x(n_3) + (-1)^{k_1} x(n_3 + \frac{N}{2}) \end{aligned} \tag{2.7}$$

Equation (2.7) represents the first two stages of butterflies with only trivial multiplications. After these two stages, full multipliers are required to compute the product of decomposed twiddle factors. The complete radix-2^2 algorithm can be derived by applying (2.6) recursively. Fig. 2.2 shows the flow graph of an $N = 16$ point FFT an decomposed according to decimation in frequency (DIF). The numbers at the inputs

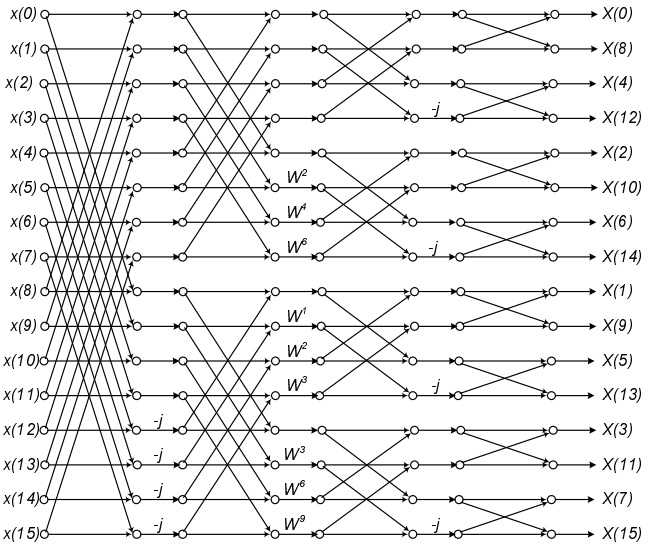


Figure 2.2: Radix-2 Flow graph of a 16-point radix-2^2 DIF FFT.

and output of the graph represent the index of input and output samples, respectively. The advantage of the algorithm is that it has the same multiplicative complexity as radix-4 algorithms, but still retains the radix-2 butterfly structures. We can observe that, only every other stage of the flow graph has non-trivial multiplications. The $-j$ notion represents the trivial multiplication, which involves only real-imaginary swapping and sign inversion.

2.2.3 Radix-2^3

The radix-2^3 FFT algorithm is proposed in [14]. Similar to radix-2^2, we can derive the algorithm by using the following new indices,

$$\begin{aligned} n &= < \frac{N}{2}n_1 + \frac{N}{4}n_2 + \frac{N}{8}n_3 + n_4 >_N \\ k &= < k_1 + 2k_2 + 4k_3 + 8k_4 >_N \end{aligned} \tag{2.8}$$

Substituting (2.8) in (5.3), we get

$$\begin{aligned} & X(k_1 + 2k_2 + 4k_3 + 8k_4) \\ & = \sum_{n_4=0}^{\frac{N}{8}-1} \sum_{n_3=0}^{1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} x(\frac{N}{2}n_1 + \frac{N}{4}n_2 + \frac{N}{8}n_3 + n_4) W_N^{nk} \end{aligned} \tag{2.9}$$

The twiddle can be decomposed into the following form

$$\begin{aligned} W_N^{nk} &= (-j)^{n_2(k_1+2k_2)} W_N^{\frac{N}{8}n_3(k_1+2k_2+4k_3)} \\ &\quad .W_N^{n_4(k_1+2k_2+4k_3)} W_N^{8n_4k_4} \end{aligned} \tag{2.10}$$

Substitute (2.10) into (2.9) and expand the summation with regard to index n_1, n_2 and n_3. After simplification we have a set of 8 DFTs of length $N/8$,

$$\begin{aligned} &X(k_1 + 2k_2 + 4k_3 + 8k_4) \\ &= \sum_{n_4=0}^{\frac{N}{8}-1} [T_{\frac{N}{8}}(n_4, k_1, k_2, k_3) W_N^{n_4(k_1+2k_2+4k_3)}] W_{\frac{N}{8}}^{n_4k_4} \end{aligned} \tag{2.11}$$

where a third butterfly structure has the expression of

$$\begin{aligned} &T_{\frac{N}{8}}(n_4, k_1, k_2, k_3) \\ &= H_{\frac{N}{4}}(n_4, k_1, k_2) + W_N^{\frac{N}{8}(k_1+2k_2+4k_3)} H_{\frac{N}{4}(n_4+\frac{N}{8}, k_1, k_2)} \end{aligned} \tag{2.12}$$

As in radix-2^2 algorithm, the first two columns of butterflies contain only trivial multiplications. The third butterfly contains a special twiddle factor

$$W_N^{\frac{N}{8}(k_1+2k_2+4k_)} = (\frac{1}{\sqrt{2}}(1-j))_1^k (-j)^{k_2+2k_3} \tag{2.13}$$

It can be easily seen that applying this twiddle factor requires only two real multiplications. Full complex multiplications are used to apply the decomposed twiddle factor $W_N^{n_4(k_1+2k_2+4k_3)}$ after the third column. An $N = 64$ example is shown in Fig. 2.3.

2.3 Prior Work

Much research has been carried out on designing pipelined architectures for computation of FFT of complex valued signals. Various algorithms have been developed to reduce the computational complexity, of which Cooley-Tukey radix-2 FFT [12] is very popular.

Algorithms including radix-4 [15], split-radix [16], radix-2^2 [13] have been developed based on the basic radix-2 FFT approach. The architectures based on these algorithms are some of the standard FFT architectures in the literature [17]-[22]. Radix-2

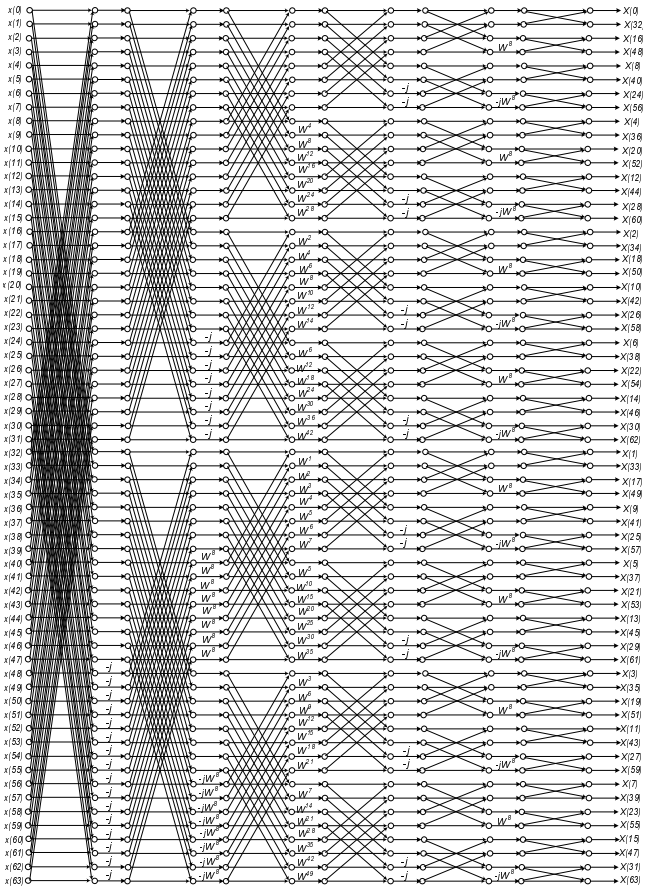


Figure 2.3: Flow graph of a 64-point radix-2^3 DIF Complex FFT

Multi-path delay commutator (R2MDC) [17] is one of the most classical approaches for pipelined implementation of radix-2 FFT is shown in Fig. Efficient usage of the storage buffer in R2MDC leads to Radix-2 Single-path delay feedback (R2SDF) architecture with reduced memory [18]. Fig shows a radix-2 feedback pipelined architecture for $N = 16$ points. R4MDC [19] and R4SDF [20], [21] are proposed as radix-4 versions of R2MDC and R4SDF respectively. Radix-4 single-path delay commutator (R4SDC) [22] is proposed using a modified radix-4 algorithm to reduce the complexity of R4MDC architecture.

Many parallel architectures for FFT have been proposed in the literature. A formal method of developing these architectures from the algorithms has not been proposed till now. In the recent literature, four parallel designs have been proposed to increase

the throughput of the FFT processors for OFDM systems [25] - [30] for a fixed 128-point FFT. These architectures are based on radix-2^3 and radix-2^4 algorithms and mixed radix (e.g., radix-2 and radix-8) algorithms. A 128-point four parallel multi-path delay feedback (MDF) architecture has been proposed in [25] based on radix-2 and radix-8 algorithms (mixed radix) combining the features of single delay feedback (SDF) and multi-path delay commutator (MDC). Another 4-parallel 128-point radix-2^4 FFT architecture based on MDF has been proposed in [26]. Further, an 8-parallel 2048-point FFT architecture has been proposed in [31] based on radix-2^3 and radix-2^4 algorithms. These architectures are based on different approaches which have their own advantages and disadvantages. The required number of delay elements in MDF architectures is less due to the delay feedback (SDF) design while the number of datapaths (consisting of butterflies) is equal to the level of parallelism.

Further, most of these hardware architectures are not fully utilized which leads to high hardware complexity. In the era of high speed digital communications, high throughput and low power designs are required to meet the speed and power requirements while keeping the hardware overhead to minimum. In this thesis, we present a new approach to design these architectures from the FFT flow graphs. Folding transformation [33] and register minimization techniques [34], [35] are used to derive several known FFT architectures. Novel architectures are developed using the proposed methodology which have not been presented in the literature.

In the folding transformation, all butterflies in the same column can be mapped to one butterfly unit. If the FFT size is N, then this corresponds to a folding factor of $N/2$. This leads to a 2-parallel architecture. In another design, we can choose a folding factor of $N/4$ to design a 4-parallel architectures, where 4 samples are processed in the same clock cycle. Different folding sets lead to a family of FFT architectures. Alternatively, known FFT architectures can also be described by the proposed methodology by selecting the appropriate folding set. Folding sets are designed intuitively to reduce latency and to reduce the number of storage elements. It may be noted that prior FFT architectures were derived in an *adhoc* way, and their derivations were not explained in a systematic way. This is the first attempt to generalize design of FFT architectures for arbitrary level of parallelism in a systematic manner via the folding transformation. In

this paper, design of prior architectures is first explained by constructing specific folding sets. Then, several new architectures are derived for various radices, and various levels of parallelism, and for either the decimation-in-time (DIT) or the decimation-in-frequency (DIF) flow graphs. All new architectures achieve full hardware utilization. It may be noted that all prior parallel FFT architectures did not achieve full hardware utilization.

2.4 FFT Architectures via Folding

In this section, we present a method to derive several known FFT architectures in general. The process is described using an 8-point radix-2 DIF FFT as an example. It can be extended to other radices in a similar fashion. Fig. 2.4 shows the flow graph of a radix-2 8-point DIF FFT. The graph is divided into 3 stages and each of them consists of a set of butterflies and multipliers. The twiddle factor in between the stages indicates a multiplication by W_N^k, where W_N denotes the Nth root of unity, with its exponent evaluated modulo N. This algorithm can be represented as a data flow graph (DFG) as shown in Fig. 2.5. The nodes in the DFG represent tasks or computations. In this case, all the nodes represent the butterfly computations of the radix-2 FFT algorithm. In particular, assume nodes A and B have the multiplier operation on the bottom edge of the butterfly.

The folding transformation is used on the DFG in Fig. 2.5 to derive a pipelined architecture. To transform the DFG, we require a folding set, which is an ordered set of operations executed by the same functional unit. Each folding set contains K entries some of which may be null operations. K is called the folding factor, the number of operations folded into a single function unit. The operation in the j-th position within the folding set (where j goes from 0 to $K-1$) is executed by the functional unit during the time partition j. The term j is the folding order, the time instance to which the node is scheduled to be executed in hardware.

For example, consider the folding set $A = \{\phi, \phi, \phi, \phi, A0, A1, A2, A3\}$ for $K = 8$. The operation $A0$ belongs to the folding set A with the folding order 4. The functional unit A executes the operations $A0, A1, A2, A3$ at the respective time instances and will be idle during the null operations. We use the systematic folding techniques to derive

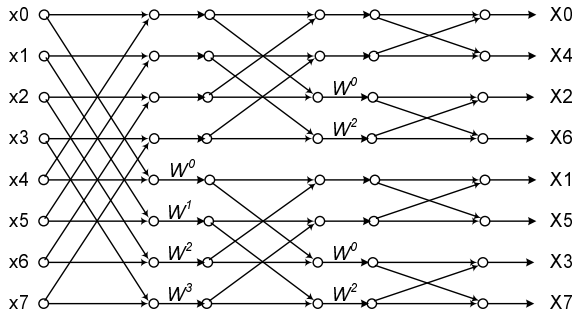


Figure 2.4: Flow graph of a radix-2 8-point DIF FFT.

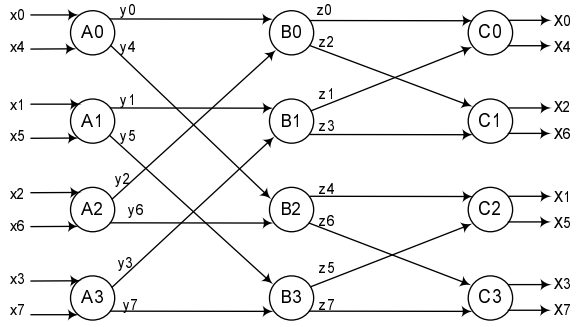


Figure 2.5: Data Flow graph (DFG) of a radix-2 8-point DIF FFT.

the 8-point FFT architecture. Consider an edge e connecting the nodes U and V with $w(e)$ delays. Let the executions of the l-th iteration of the nodes U and V be scheduled at the time units $Kl + u$ and $Kl + v$, respectively, where u and v are the folding orders of the nodes U and V. The folding equation for the edge e is

$$D_F(U \rightarrow V) = Kw(e) - P_U + v - u \tag{2.14}$$

where P_U is the number of pipeline stages in the hardware unit which executes the node U [33].

2.4.1 Feed-forward Architecture

Consider folding of the DFG in Fig. 2.5 with the folding sets

$$\begin{aligned} A &= \{\phi, \phi, \phi, \phi, A0, A1, A2, A3\}, \\ B &= \{B2, B3, \phi, \phi, \phi, \phi, B0, B1\}, \\ C &= \{C1, C2, C3, \phi, \phi, \phi, \phi, C0\}. \end{aligned}$$

Assume that the butterfly operations do not have any pipeline stages, i.e., $P_A = 0$, $P_B = 0$, $P_C = 0$. The folded architecture can be derived by writing the folding equation in (2.14) for all the edges in the DFG. These equations are

$$\begin{aligned} &D_F(A0 \rightarrow B0) = 2 && D_F(B0 \rightarrow C0) = 1 \\ &D_F(A0 \rightarrow B2) = -4 && D_F(B0 \rightarrow C1) = -6 \\ &D_F(A1 \rightarrow B1) = 2 && D_F(B1 \rightarrow C0) = 0 \\ &D_F(A1 \rightarrow B1) = -4 && D_F(B1 \rightarrow C1) = -7 \\ &D_F(A2 \rightarrow B0) = 0 && D_F(B2 \rightarrow C2) = 1 \\ &D_F(A2 \rightarrow B2) = -6 && D_F(B2 \rightarrow C3) = 2 \\ &D_F(A3 \rightarrow B1) = 0 && D_F(B3 \rightarrow C2) = 0 \\ &D_F(A3 \rightarrow B3) = -6 && D_F(B3 \rightarrow C3) = 1 \end{aligned} \tag{2.15}$$

For example, $D_F(A0 \rightarrow B0) = 2$ means that there is an edge from the butterfly node A to node B in the folded DFG with 2 delays. For the folded system to be realizable, $D_F(U \rightarrow V) \geq 0$ must hold for all the edges in the DFG. Retiming and/or pipelining

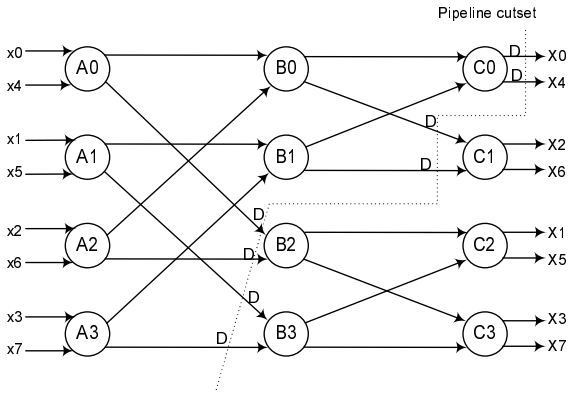


Figure 2.6: Pipelined Data Flow graph (DFG) of a 8-point DIF FFT as a preprocessing step for folding

can be used to either satisfy this property or determine that the folding sets are not feasible. We can observe the negative delays on some edges in (2.15). The DFG can be pipelined as shown in Fig. 2.6 to ensure that folded hardware has a non-negative number of delays. The updated folding equations for all the edges are

$$\begin{aligned}
D_F(A0 \rightarrow B0) &= 2 \qquad D_F(B0 \rightarrow C0) = 1 \\
D_F(A0 \rightarrow B2) &= 4 \qquad D_F(B0 \rightarrow C1) = 2 \\
D_F(A1 \rightarrow B1) &= 2 \qquad D_F(B1 \rightarrow C0) = 0 \\
D_F(A1 \rightarrow B1) &= 4 \qquad D_F(B1 \rightarrow C1) = 1 \\
D_F(A2 \rightarrow B0) &= 0 \qquad D_F(B2 \rightarrow C2) = 1 \\
D_F(A2 \rightarrow B2) &= 2 \qquad D_F(B2 \rightarrow C3) = 2 \\
D_F(A3 \rightarrow B1) &= 0 \qquad D_F(B3 \rightarrow C2) = 0 \\
D_F(A3 \rightarrow B3) &= 2 \qquad D_F(B3 \rightarrow C3) = 1
\end{aligned} \tag{2.16}$$

From (2.16), we can observe that 24 registers are required to implement the folded architecture. Lifetime analysis technique [35] is used to design the folded architecture that use the minimum possible registers. For example, in the current 8-point FFT design, consider the variables $y0, y1, ...y7$ i.e., the outputs at the nodes $A0, A1, A2, A3$ respectively. It takes 16 registers to synthesize these edges in the folded architecture. The linear lifetime chart for these variables is shown in Fig. 2.7. From the lifetime chart,

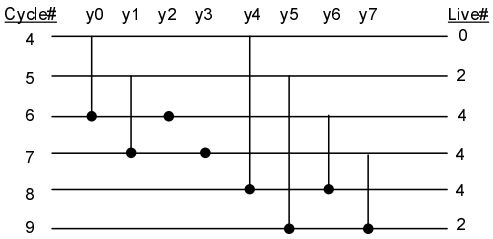


Figure 2.7: Linear lifetime chart for the variables $y0, y1, ...y7$.

it can be seen that the folded architecture requires 4 registers as opposed to 16 registers in a straightforward implementation. The next step is to perform forward-backward register allocation. The allocation table is shown in Fig. 2.8. From the allocation table in Fig. 2.8 and the folding equations in (2.16), the delay circuit in Fig. 2.9 can be synthesized. Fig. 2.10 shows how node A and node B are connected in the folded architecture.

	I/P	R1	R2	R3	R4
4	y0,y4				
5	y1,y5	y4	y0		
6	(y2),y6	y5	y4	(y0)	y1
7	(y3),y7	y6	y5	y4	(y1)
8		y7	(y6)	y5	(y4)
9			(y7)		(y5)

Figure 2.8: Register allocation table for the data represented in Fig. 2.7.

The control complexity of the derived circuit is high. Four different signals are needed to control the multiplexers. A better register allocation is found such that the number of multiplexers are reduced in the final architecture. The more optimized register allocation for the same lifetime analysis chart is shown in Fig. 2.11. Similarly, we can apply lifetime analysis and register allocation techniques for the variables $x0, ..., x7$ and

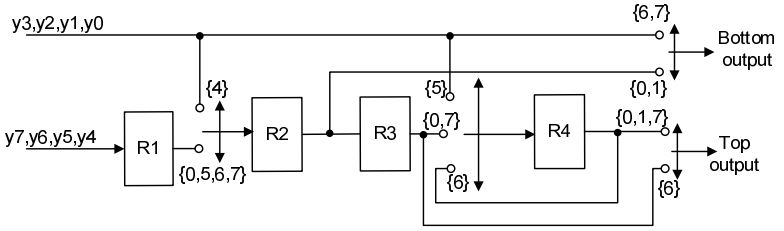


Figure 2.9: Delay circuit for the register allocation table in Fig. 2.8.

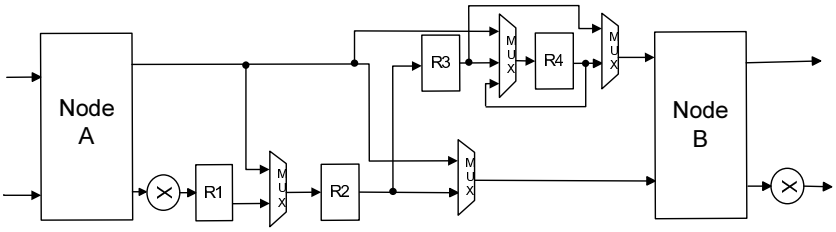


Figure 2.10: Folded circuit between Node A and Node B.

$z0, ..., z7$, inputs to the DFG and the outputs from nodes $B0, B1, B2, B3$ respectively as shown in Fig. 2.5. From the allocation table in Fig. 2.11 and the folding equations in (2.16), the final architecture in Fig. 2.12 can be synthesized.

We can observe that the derived architecture is the same as R2MDC architecture [17]. Similarly, the R2SDF architecture can be derived by minimizing the registers on all the variables at once.

2.4.2 Feedback Architecture

We derive the feedback architecture using the same 8-point radix-2 DIT FFT example in Fig. 2.4. Consider the following folding sets

$$A = \{\phi, \phi, \phi, \phi, A0, A1, A2, A3\},$$
$$B = \{\phi, \phi, B2, B3, \phi, \phi, B0, B1\},$$
$$C = \{\phi, C1, \phi, C2, \phi, C3, \phi, C0\}.$$

Assume that the butterfly operations do not have any pipeline stages, i.e., $P_A = 0$, $P_B = 0$, $P_C = 0$. The folded architecture can be derived by writing the folding equation

	I/P	R1	R2	R3	R4
4	y0,y4				
5	y1,y5	y4		y0	
6	y2,y6	y5	y4	y1	y0
7	y3,y7	y6	y5	y4	y1
8		y7	y6	y5	y4
9			y7		y5

Figure 2.11: Register allocation table for the data represented in Fig. 2.7.

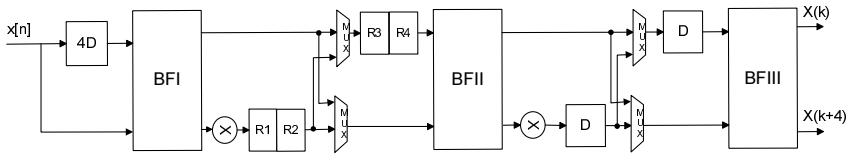


Figure 2.12: Folded architecture for the DFG in Fig. 2.6. This corresponds to the well known radix-2 feed-forward (R2MDC) architecture.

in (2.14) for all the edges in the DFG. The folding equations are

$$
\begin{aligned}
&D_F(A0 \rightarrow B0) = 2 && D_F(B0 \rightarrow C0) = 1 \\
&D_F(A0 \rightarrow B2) = -2 && D_F(B0 \rightarrow C1) = -5 \\
&D_F(A1 \rightarrow B1) = 2 && D_F(B1 \rightarrow C0) = 0 \\
&D_F(A1 \rightarrow B1) = -2 && D_F(B1 \rightarrow C1) = -6 \\
&D_F(A2 \rightarrow B0) = 0 && D_F(B2 \rightarrow C2) = 1 \\
&D_F(A2 \rightarrow B2) = -4 && D_F(B2 \rightarrow C3) = 3 \\
&D_F(A3 \rightarrow B1) = 0 && D_F(B3 \rightarrow C2) = 0 \\
&D_F(A3 \rightarrow B3) = -4 && D_F(B3 \rightarrow C3) = 2
\end{aligned} \tag{2.17}
$$

It can be seen from the folding equations in (2.17) that some edges contain negative delays. Retiming is used to make sure that the folded hardware has non-negative number

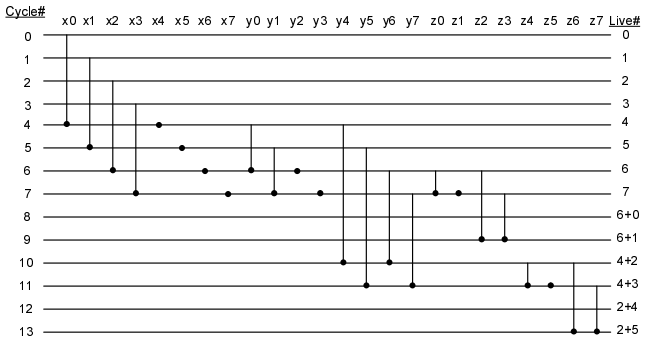


Figure 2.13: Linear lifetime chart for variables for a 8-point FFT architecture.

of delays. The pipelined DFG is the same as the one in the feed-forward example and is shown in Fig. 2.6. The updated folding equations are shown in (2.18).

$$
\begin{aligned}
&D_F(A0 \rightarrow B0) = 2 \qquad D_F(B0 \rightarrow C0) = 1 \\
&D_F(A0 \rightarrow B2) = 6 \qquad D_F(B0 \rightarrow C1) = 3 \\
&D_F(A1 \rightarrow B1) = 2 \qquad D_F(B1 \rightarrow C0) = 0 \\
&D_F(A1 \rightarrow B1) = 6 \qquad D_F(B1 \rightarrow C1) = 2 \\
&D_F(A2 \rightarrow B0) = 0 \qquad D_F(B2 \rightarrow C2) = 1 \\
&D_F(A2 \rightarrow B2) = 4 \qquad D_F(B2 \rightarrow C3) = 3 \\
&D_F(A3 \rightarrow B1) = 0 \qquad D_F(B3 \rightarrow C2) = 0 \\
&D_F(A3 \rightarrow B3) = 4 \qquad D_F(B3 \rightarrow C3) = 2
\end{aligned} \tag{2.18}
$$

The number of registers required to implement the folding equations in (2.18) is 40. The linear life time chart is shown in Fig. 2.13 and the register allocation table is shown in Fig. 2.14. We can implement the same equations using 7 registers by using these register minimization techniques. The folded architecture in Fig. 2.15 is synthesized using the folding equations in (2.18) and register allocation table.

The hardware utilization is only 50% in the derived architecture. This can also be observed from the folding sets where half of the time null operations are being executed, i.e., hardware is idle. A family of high throughput FFT architectures have been obtained

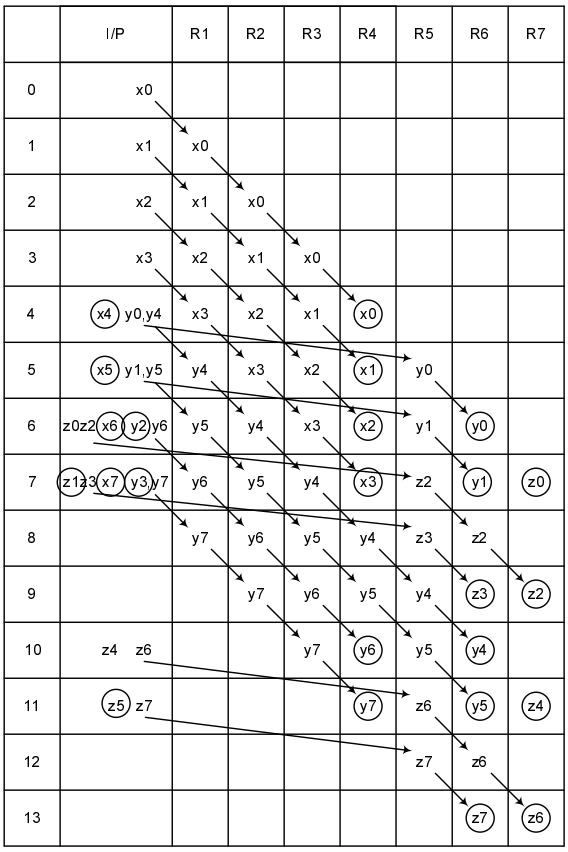


Figure 2.14: Register allocation table for the data represented in Fig. 2.13.

by using the new approach using both decimation-in-frequency (DIF) and decimation-in-time (DIT) algorithms. The parallelization can be arbitrarily chosen. Furthermore, architectures based on radix-2^2 and radix-2^3 algorithms have been developed. The circuits to reorder the output samples are also presented.

2.5 Architectures using DIF flow graph

2.5.1 Radix-2 FFT Architectures

In this section, we present parallel architectures for complex valued signals based on radix-2 algorithm. These architectures are derived using the approach presented in the previous section. The same approach can be extended to radix-2^2, radix-2^3 and other

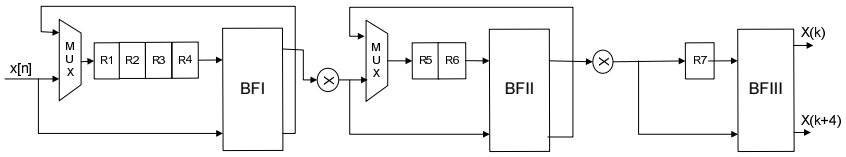


Figure 2.15: Folded architecture for the DFG in Fig. 2.6. This corresponds to the well known radix-2 feedback (R2SDF) architecture.

radices as well. Due to space constraints, only folding sets are presented for different architectures. The folding equations and register allocation tables can be obtained easily.

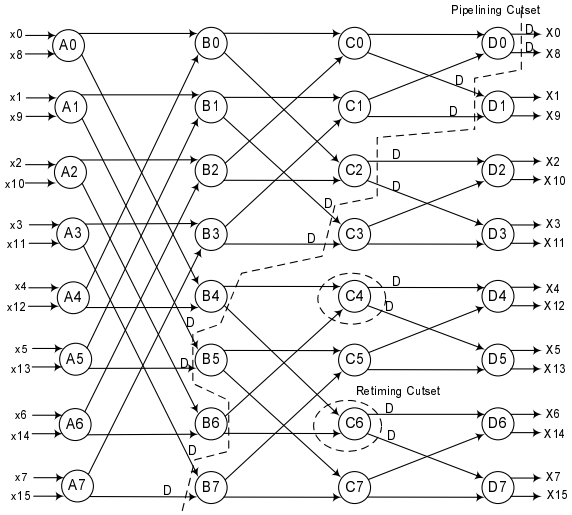


Figure 2.16: Data Flow graph (DFG) of a Radix-2 16-point DIF FFT with retiming for folding.

2-parallel Radix-2 FFT Architecture

The utilization of hardware components in the feedforward architecture is only 50%. This can also be observed from the folding sets of the DFG where half of the time null operations are being executed. We can derive new architectures by changing the folding sets which can lead to efficient architectures in terms of hardware utilization and power consumption. We present here one such example of a 2-parallel architecture which leads to 100% hardware utilization and consumes less power.

Fig. 2.16 shows the DFG of radix-2 DIF FFT for $N = 16$. All the nodes in this figure represent radix-2 butterfly operations. Assume the nodes A, B and C contain the multiplier operation at the bottom output of the butterfly. Consider the folding sets

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\}, \\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\}, \\ C &= \{C3, C5, C7, C0, C2, C4, C6, C1\}, \\ D &= \{D2, D4, D6, D1, D3, D5, D7, D0\} \end{aligned} \tag{2.19}$$

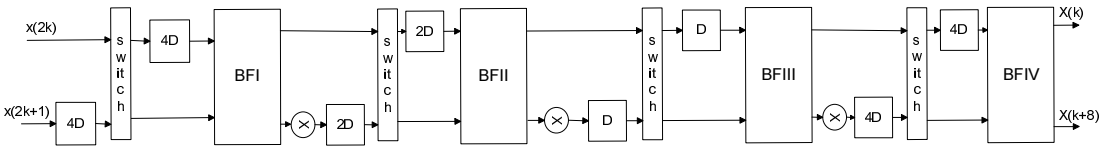


Figure 2.17: Proposed 2-parallel (Architecture 1) for the computation of a radix-2 16-point DIF FFT.

We can derive the folded architecture by writing the folding equation in (2.14) for all the edges. Pipelining and retiming are required to get non-negative delays in the folded architecture. The DFG in Fig. 2.16 also shows the retimed delays on some of the edges of the graph. The final folded architecture is shown in Fig. 2.17. The register minimization techniques and forward-backward register allocation are also applied in deriving this architecture as described in Section II. Note the similarity of the datapath to R2MDC. This architecture processes two input samples at the same time instead of one sample in R2MDC. The implementation uses regular radix-2 butterflies. Due to the spatial regularity of the radix-2 algorithm, the synchronization control of the design is very simple. A log_2N-bit counter serves two purposes: synchronization controller i.e., the control input to the switches, and address counter for twiddle factor selection in each stage.

We can observe that the hardware utilization is 100% in this architecture. In a general case of N-point FFT, with N power of 2, the architecture requires $log_2(N)$ complex butterflies, $log_2(N) - 1$ complex multipliers and $3N/2 - 2$ delay elements or buffers.

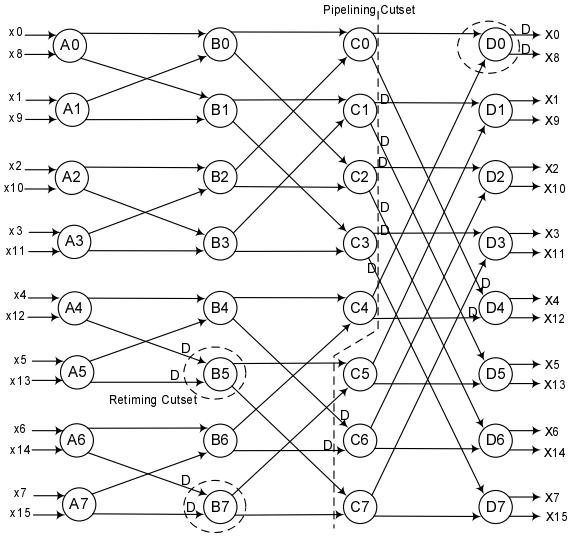


Figure 2.18: DFG of a radix-2 16-point DIT FFT with retiming for folding.

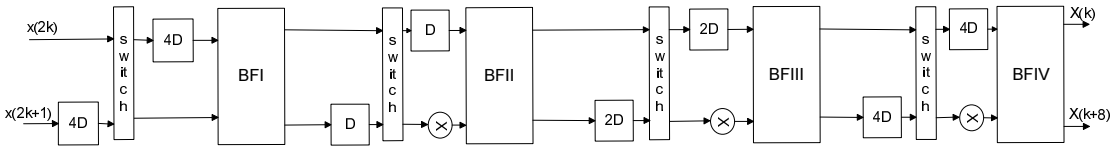


Figure 2.19: Proposed 2-parallel (Architecture 1) for the computation of a radix-2 16-point DIT Complex FFT.

In a similar manner, we can derive the 2-parallel architecture for Radix-2 DIT FFT using the following folding sets. Assume that multiplier is at the bottom input of the nodes B, C, D.

$$A = \{A0, A2, A1, A3, A4, A6, A5, A7\},$$
$$B = \{B5, B7, B0, B2, B1, B3, B4, B6\},$$
$$C = \{C6, C5, C7, C0, C2, C1, C3, C4\},$$
$$D = \{D2, D1, D3, D4, D6, D5, D7, D0\}$$

The pipelined/retimed version of the DFG is shown in Fig. 2.18 and the 2-parallel architecture is in Fig. 2.19. The only difference in the two architectures (Fig. 2.17 and Fig. 2.19) is the position of the multiplier in between the butterflies. The rest of the design remains same.

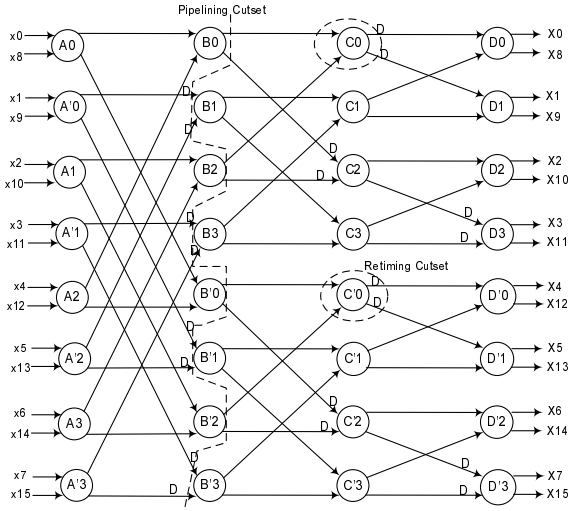


Figure 2.20: Data Flow graph (DFG) of a Radix-2 16-point DIF FFT with retiming for folding for 4-parallel architecture.

4-parallel Radix-2 FFT Architecture

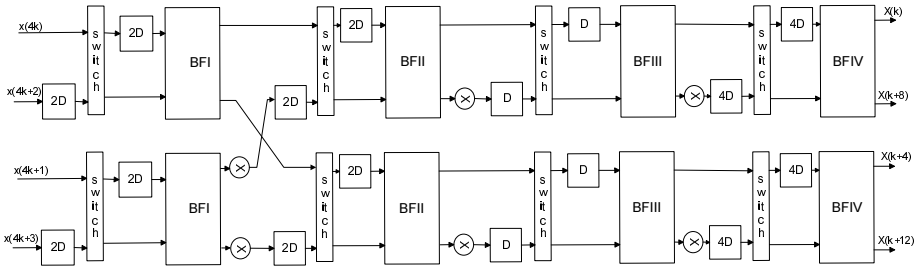


Figure 2.21: Proposed 4-parallel (Architecture 2) for the computation of 16-point radix-2 DIF FFT.

A 4-parallel architecture can be derived using the following folding sets.

$$A = \{A0, A1, A2, A3\} \qquad A' = \{A'0, A'1, A'2, A'3\}$$
$$B = \{B1, B3, B0, B2\} \qquad B' = \{B'1, B'3, B'0, B'2\}$$
$$C = \{C2, C1, C3, C0\} \qquad C' = \{C'2, C'1, C'3, C'0\}$$
$$D = \{D3, D0, D2, D1\} \qquad D' = \{D'3, D'0, D'2, D'1\}$$

The DFG shown in Fig. 2.20 is retimed to get the non-negative folded delays. The

final architecture in Fig. 2.21 can be obtained following the proposed approach. For a N-point FFT, the architecture takes $4(log_4 N - 1)$ complex multipliers and $2N - 4$ delay elements. We can observe that hardware complexity is almost double that of the serial architecture and processes 4-samples in parallel. The power consumption can be reduced by 50% (see Section V) by lowering the operational frequency of the circuit.

2.5.2 Radix-2^2 and Radix-2^3 FFT Architectures

The hardware complexity in the parallel architectures can be further reduced by using radix-2^n FFT algorithms. In this section, we consider the cases of radix-2^2 and radix-2^3 to demonstrate how the proposed approach can be used to radix-2^n algorithms. Similarly, we can develop architectures for radix-2^4 and other higher radices using the same approach.

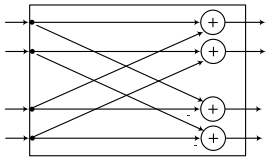
2-parallel radix-2^2 FFT Architecture

The DFG of radix-2^2 DIF FFT for $N = 16$ will be similar to the DFG of radix-2 DIF FFT as shown in Fig. 2.16. All the nodes in this figure represent radix-2 butterfly operations including some special functionality. Nodes A and C represent regular butterfly operations. Nodes B and D are designed to include the $-j$ multiplication factor. Fig. 3.7 shows the butterfly logic needed to implement the radix-2^2 FFT. The factor $-j$ is handled in the second butterfly stage using the logic shown in Fig. 3.7b to switch the real and imaginary parts to the input of the multiplier.

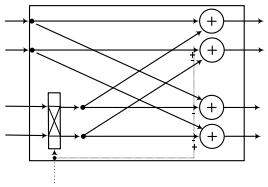
Consider the folding sets

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\}, \\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\}, \\ C &= \{C3, C5, C7, C0, C2, C4, C6, C1\}, \\ D &= \{D2, D4, D6, D1, D3, D5, D7, D0\} \end{aligned} \tag{2.20}$$

Using the folding sets above, the final architecture shown in Fig. 2.23 is obtained. We can observe that the number of complex multipliers required for radix-2^2 architecture is less compared to radix-2 architecture in Fig. 2.17. In general, for a N-point FFT, radix-2^2 architecture requires $2(log_4 N - 1)$ multipliers.



(a) BFI



(b) BFII

Figure 2.22: Butterfly structures for the proposed FFT architecture

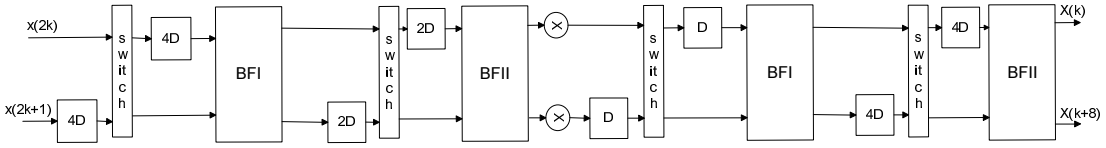


Figure 2.23: Proposed 2-parallel (Architecture 3) for the computation of a radix-2^2 16-point DIF FFT.

Similar to 4-parallel radix-2 architecture, we can derive 4-parallel radix-2^2 architecture using the similar folding sets. The 4-parallel radix-2^2 architecture is shown in Fig. 2.24. In general, for a N-point FFT, 4-parallel radix-2^2 architecture requires $3(log_4 N-1)$ complex multipliers compared $4(log_4 N-1)$ multipliers in radix-2 architecture. That is, the multiplier complexity is reduced by 25% compared to radix-2 architectures.

2-parallel radix-2^3 FFT Architecture

We consider the example of a 64-point radix-2^3 FFT algorithm [14]. The advantage of radix-2^3 over radix-2 algorithm is its multiplicative complexity reduction. A 2-parallel architecture is derived using folding sets in (2.20). Here the DFG contains 32 nodes instead of 8 in 16-point FFT.

The folded architecture is shown in Fig. 2.25. The design contains only two full multipliers and two constant multipliers. The constant multiplier can be implemented

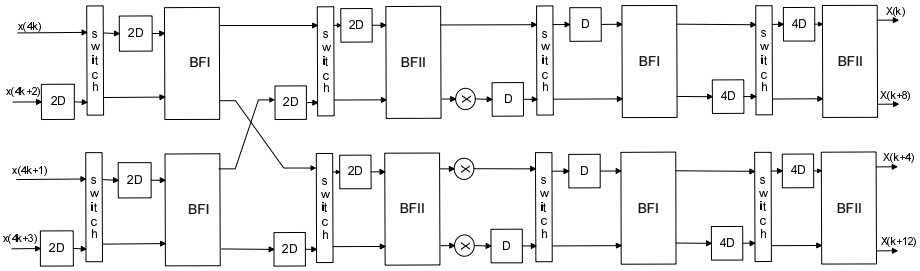


Figure 2.24: Proposed 4-parallel (Architecture 4) for the computation of a radix-2^2 16-point DIF FFT.

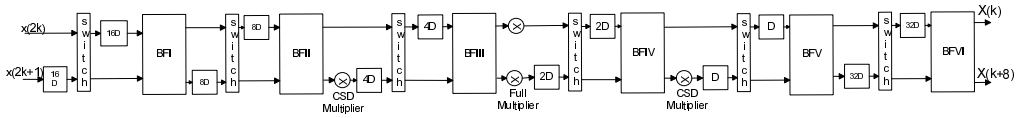


Figure 2.25: Proposed 2-parallel (Architecture 5) for the computation of 64-point radix-2^3 DIF FFT.

using Canonic Signed Digit (CSD) format with much less hardware compared to a full multiplier. For a N-point FFT, where N is a power of 2^3, the proposed architecture takes $2(log_8 N - 1)$ multipliers and $3N/2 - 2$ delays. The multiplier complexity can be halved by computing the two operations using one multiplier. This can be seen in the modified architecture shown in Fig. 2.26. The only disadvantage of this design is that two different clocks are needed. Multiplier has to be run at double the frequency compared to the rest of the design. The architecture requires only $log_8 N - 1$ multipliers.

A 4-parallel radix-2^3 architecture can be derived similar to 4-parallel radix-2 FFT architecture. A large number of architectures can be derived using the approach presented in Section II. Using the folding sets of same pattern, 2-parallel and 4-parallel architectures can be derived for radix-2^2 and radix-2^4 algorithms. We show that changing the folding sets can lead to different parallel architectures. Further DIT and DIF algorithms will lead to similar architectures except the position of the multiplier operation.

2.6 Architecture using DIT flow graph

The proposed architectures using decimation-in-time (DIT) flow graph reduce the number of delays in the pipelined architecture compared to DIF based designs.

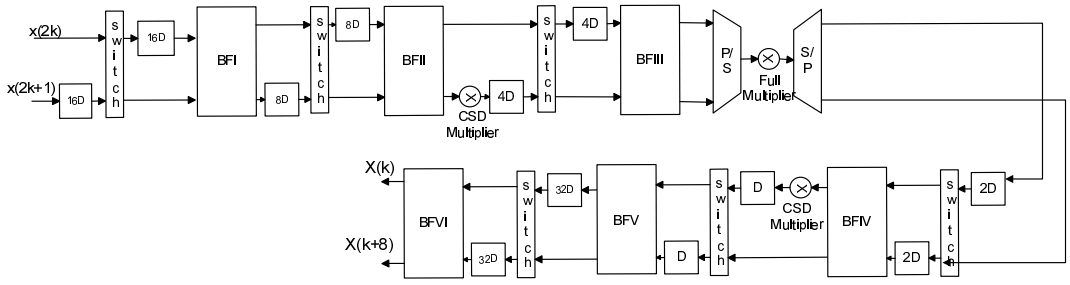


Figure 2.26: Proposed 2-parallel (Architecture 6) for the computation of 64-point radix-2^3 DIF FFT.

2.6.1 4-parallel design

The proposed method can be explained using two different design approaches: *adhoc* and folding approach. Further, the same method can be extended to design an eight-parallel architecture. An example of 16-point radix-2 FFT algorithm is used to describe the proposed method.

Ad-hoc design

Fig. 2.16 shows the flow graph of 16-point radix-2 DIT algorithm, where the nodes from $A0, ..., A3$ represent the top 4 butterflies (processing even samples) in the first stage of the FFT and $A'0, ..., A'3$ represent the bottom 4 butterflies (processing odd samples). Similarly, $B0, ...B4, B'0, ..., B'4$, $C0, ..., C3$, $C'0, ..., C'3$, and $D0, ..., D3$, $D'0, ..., D'3$ represent nodes in second, third and fourth stages, respectively. It can be observed that the even and odd samples can be processed independently until the final stage which is illustrated in Fig. 2.16. The outputs of the two N/2 FFTs can be combined in the final butterfly stage. We can develop a two parallel architecture for each of the N/2-point FFTs which can process two consecutive even $(4k, 4k + 2)$ and odd $(4k + 1, 4k + 3)$ samples, respectively. Further, the initial reordering delay elements are not required, if the input buffer is available to reorder the data before FFT processor. As the outputs of the two N/2-point FFTs arrive at the same time, the final stage does not require reshuffling circuit (delays and switches). This will reduce the required number of delays to implement the pipelined architecture.

The four parallel pipelined architecture for 16-point FFT is shown in Fig. 2.27.

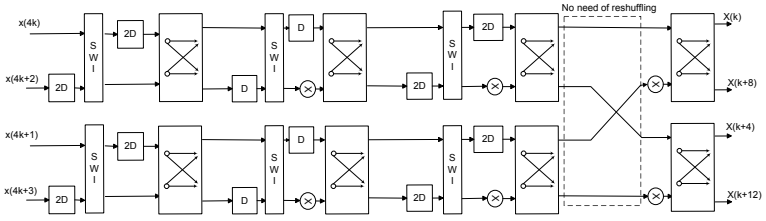


Figure 2.27: Proposed 4-parallel architecture for 16-point radix-2 DIT FFT.

We can observe that there is no reshuffling circuit before the last stage of butterflies. The implementation uses regular radix-2 butterflies. The rest of the datapath contains switches and delay elements. The function of the switch and delay elements is to reorder the incoming samples to provide the corresponding samples at the input of each butterfly stage according to the data flow during every clock cycle. The control signal controls the multiplexers which connects the input and output of the switch in two different ways (either straight or cross paths). The control signals for switches in different stages of the architecture can be generated by using simple counter logic.

Folding approach

The proposed approach can also be described using folding methodology [33]. The four parallel architecture can be derived using the following folding sets.

$$A = \{A0, A1, A2, A3\} \qquad A' = \{A'0, A'1, A'2, A'3\}$$
$$B = \{B3, B0, B1, B2\} \qquad B' = \{B'3, B'0, B'1, B'2\}$$
$$C = \{C1, C2, C3, C0\} \qquad C' = \{C'1, C'2, C'3, C'0\}$$
$$D = \{D1, D2, D3, D0\} \qquad D' = \{D'1, D'2, D'3, D'1\}$$

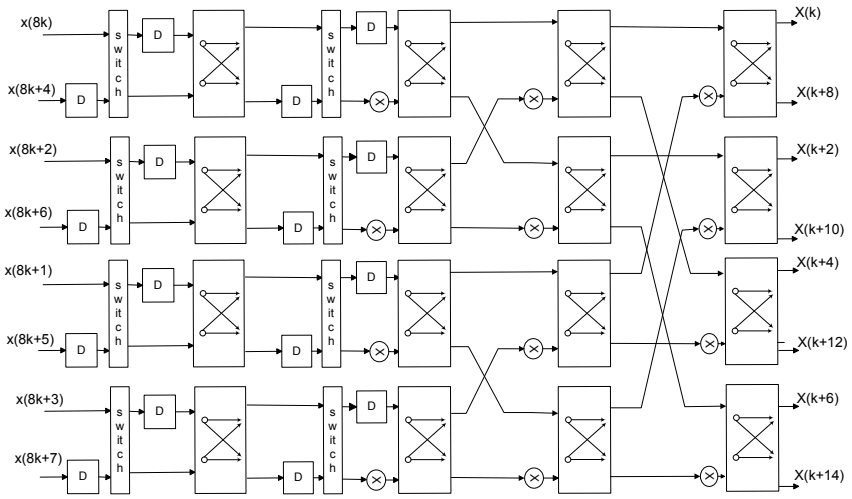


Figure 2.28: Proposed 8-parallel architecture for 16-point radix-2 FFT.

We can derive the folded architecture by writing the folding equation [33] for the edges in the flow graph. The register minimization techniques and the forward and backward register allocation scheme [35] are applied to derive the architecture. The final architecture can be derived to be same shown as the design in Fig. 2.27.

In general, for an N-point FFT, the proposed four parallel architecture requires $N-4$ delay elements. An additional $N/2$ delay elements are required when the input buffer is not in place to reorder the input samples. The proposed architecture reduces the number of delay elements by 33% compared to the design in [11], which requires $3N/2-4$ delay elements. Further, the proposed architecture has only two parallel data paths (each processing two samples) compared to four data paths in [25], [26]. The proposed architecture requires $2log_2N$ butterfly units compared to $4log_2N$ in the prior designs. That is, the proposed design reduces the number of complex adders by 50% compared to the designs in [25] and [26].

2.6.2 8-parallel design

In a similar fashion, an 8-parallel architecture can be derived based on the DIT flow graph. Now the input samples can be divided into four sections, $8k, 8k+4$, $8k+2, 8k+6$, $8k+1, 8k+5$, and $8k+3, 8k+7$. These sample groups can be processed independently until the last two stages. The 8-parallel architecture can be derived

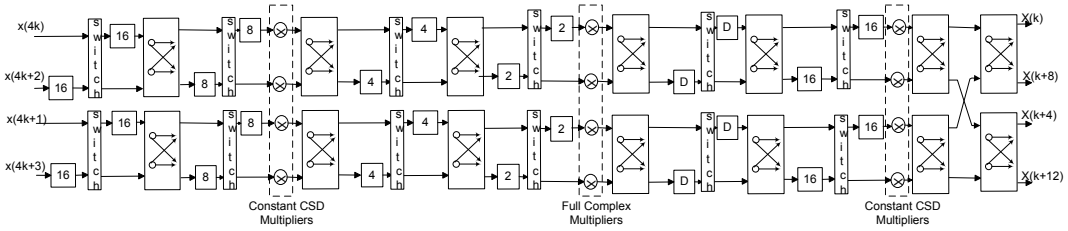


Figure 2.29: Block diagram of the proposed 4-parallel 128-point FFT architecture based on radix-2^4 algorithm

using folding approach. Fig. 2.28 shows the 8-parallel architecture for a 16-point radix-2 FFT. In general, for an N-point FFT, the proposed 8-parallel architecture requires $N - 8$ delay elements excluding the input buffers and $4log_2N$ butterfly units. The 8-parallel architecture in [31] consists of 8 datapaths which leads to $8log_2N$ butterfly units.

2.6.3 Proposed 128-pt FFT architecture

The number of complex multipliers required depends on the underlying FFT algorithm. In higher radix algorithms (radix-2^3, radix-2^4), only the twiddle factors will change at each input/output stage of the butterflies. For example, in radix-2^2 algorithm twiddle factor multiplication is required in every alternate stage while it is required in every stage in radix-2 algorithm. To demonstrate that the proposed approach can be extended to any radix-2^n algorithms, we present a 4-parallel 128-point FFT architecture based on the radix-2^4 algorithm.

The 128-point FFT flow graph is based on radix-2^4 algorithm which is decimated in time. The higher radix algorithms lead to low multiplication complexity. For example, radix-2^4 algorithm leads to only one complex multiplication every 4 stages and radix-2^3 algorithm leads to one complex multiplication every 3 stages. But constant multipliers are required to implement trivial twiddle factors at the intermediate stages. The proposed 128-point FFT architecture is based on the radix-2^4 algorithm to reduce the number of constant multipliers. The radix-2^4 algorithms are described in detail in [23].

To achieve the high throughput requirement with low hardware cost, both the proposed pipelining method and radix-2^n algorithms are exploited in this design. The

proposed 4-parallel 128-point FFT architecture is shown in Fig. 2.29. It consists of two parallel data paths processing two input samples. Each data path consists of seven butterfly units, four constant and two full complex multipliers, delay elements and multiplexers. The function of delay elements and switches is to store and reorder the input data until the other available data is received for the butterfly operation. The four output data values generated after the first stage are multiplied by constant twiddle factors ($W_8^1 = e^{-j2\pi/8}, W_8^3 = e^{-j2\pi 3/8}$). These twiddle factors can be implemented efficiently using canonic signed digit (CSD) approach. The outputs after the third stage are multiplied by the nontrivial twiddle factor. Another constant multiplier stage is required before the sixth butterfly stage. The CSD complex constant multiplier processes the multiplication of twiddle factors $W^8, W^{16}, W^{24}, W^{48}$. These twiddle factors correspond to $cos(\pi/8), sin(\pi/8)$, and $cos(\pi/4)$.

2.7 Reordering of the Output Samples

Reordering of the output samples is an inherent problem in FFT computation. The outputs are obtained in the bit-reversal order [17] in the serial architectures. In general the problem is solved using a memory of size N. Samples are stored in the memory in natural order using a counter for the addresses and then they are read in bit-reversal order by reversing the bits of the counter. In embedded DSP systems, special memory addressing schemes are developed to solve this problem. But in case of real-time systems, this will lead to an increase in latency and area.

The order of the output samples in the proposed architectures is not in the bit-reversed order. The output order changes for different architectures because of different folding sets/scheduling schemes. We need a general scheme for reordering these samples. One such approach is presented in this section.

The approach is described using a 16-point radix-2 DIF FFT example and the corresponding architecture is shown in Fig. 2.17. The order of output samples is shown in Fig. 2.30. The first column (index) shows the order of arrival of the output samples. The second column (output order) indicates the indices of the output frequencies. The goal is to obtain the frequencies in the desired order provided the order in the last column. We can observe that it is a type of de-interleaving from the output order and

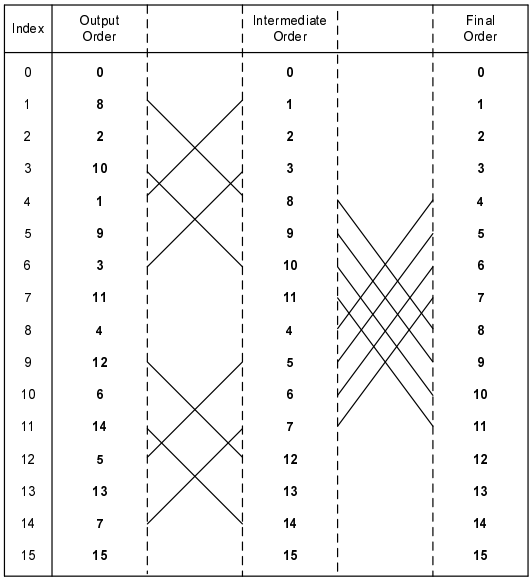


Figure 2.30: Solution to the reordering of the output samples for the architecture in Fig. 2.17.

the final order. Given the order of samples, the sorting can be performed in two stages. It can be seen that the first and the second half of the frequencies are interleaved. The intermediate order can be obtained by de-interleaving these samples as shown in the table. Next, the final order can be obtained by changing the order of the samples. It can be generalized for higher number of points, the reordering can be done by shuffling the samples in the respective positions according to the final order required.

A shuffling circuit is required to do the de-interleaving of the output data. Fig. 2.31 shows a general circuit which can shuffle the data separated by R positions. If the multiplexer is set to "1" the output will be in the same order as the input, whereas setting it to "0" the input sample in that position is shuffled with the sample separated by R positions. The circuit can be obtained using lifetime analysis and forward-backward register allocation techniques. There is an inherent latency of R in this circuit.

The life time analysis chart for the 1st stage shuffling in Fig. 2.30 is shown in Fig. 2.32 and the register allocation table is in Fig. 2.33. Similar analysis can be done for the 2nd stage too. Combining the two stages of reordering in Fig. 2.30, the circuit in Fig. 2.34 performs the shuffling of the outputs to obtain them in the natural order. It uses

seven complex registers for a 16-point FFT. In general case, a N-point FFT requires a memory of $5N/8 - 3$ complex data to obtain the outputs in natural order.

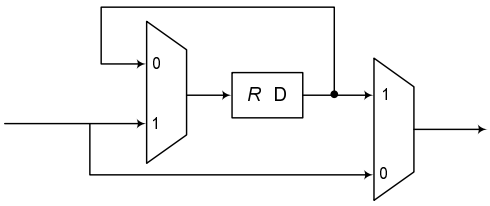


Figure 2.31: Basic circuit for the shuffling the data.

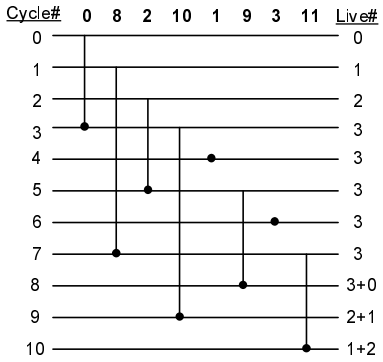


Figure 2.32: Linear lifetime chart for the 1st stage shuffling of the data.

2.8 Comparison and Analysis

A comparison is made between the previous pipelined architectures and the proposed ones for the case of computing an N-point complex FFT in Table 2.1. The comparison is made in terms of required number of complex multipliers, adders, delay elements and twiddle factors and throughput.

The proposed architectures are all feed-forward which can process 2 samples in parallel, thereby achieving a higher performance than previous designs which are serial in nature. When compared to some previous architectures, the proposed design doubles the throughput and halves the latency while maintaining the same hardware complexity. The proposed architectures maintain hardware regularity compared to previous designs.

The proposed DIT based 4-parallel and 8-parallel architectures require $3N/2 - 4$ and

	I/P	R1	R2	R3
0	0			
1	8	0		
2	2	8	0	
3	10	2	8	0
4	1	10	2	8
5	9	8	10	2
6	3	9	8	10
7	11	10	9	8
8		11	10	9
9			11	10
10				11

Figure 2.33: Register allocation table for the 1st stage shuffling of the data.

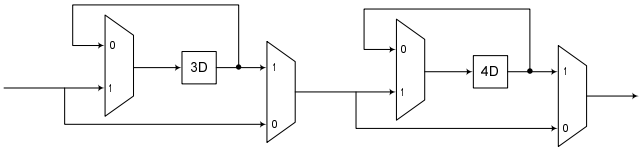


Figure 2.34: Structure for reordering the output data of 16-point DIF FFT.

$3N/2 - 8$ delay elements, respectively for an N-point FFT. The 4-parallel architecture based on DIF algorithm requires $2N - 4$ delay elements [11]. This number does not depend on the radix of the algorithm, i.e., it will remain same for all radix-2^n algorithms. In general, delay feedback based architectures require $N - 4$ delay elements compared to $3N/2 - 4$ in the proposed architecture. Even though there is a difference of $N/2$ delay elements, these are required at the input stage to reorder the samples according to the flow graph. These delays are not required if we can reorder the samples before feeding them to the FFT processor (in general applications, memory will be used to store the input samples and we can read the samples in the required order instead

Table 2.1: Comparison of pipelined hardware architectures for the computation of N-point FFT

Architecture	# Multipliers	# Adders	# Delays	Throughput
R2MDC	$2(log_4N-1)$	$4log_4N$	$3N/2-2$	1
R2SDF	$2(log_4N-1)$	$4log_4N$	$N-1$	1
R4SDC	(log_4N-1)	$3log_4N$	$2N-2$	1
R2^2SDF	(log_4N-1)	$4log_4N$	$N-1$	1
R2^3SDF*	(log_8N-1)	$4log_4N$	$N-1$	1
Proposed Architectures				
Arch 1 (radix-2)	$2(log_4N-1)$	$4log_4N$	$3N/2-2$	2
Arch 2 (radix-2)	$4(log_4N-1)$	$8log_4N$	$2N-4$	4
Arch 3 (radix-2^2)	$2(log_4N-1)$	$4log_4N$	$3N/2-2$	2
Arch 4 (radix-2^2)	$3(log_4N-1)$	$8log_4N$	$2N-4$	4
Arch 5 (radix-2^3)*	$2(log_8N-1)$	$4log_4N$	$3N/2-2$	2
Arch 6 (radix-2^3)*	log_8N-1	$4log_4N$	$3N/2-2$	2

* These architectures need 2 constant multipliers as described in Radix-$2^3 algorithm$

of natural order). Further, the proposed architecture requires $4log_2N$ complex adders compared to the $8log_2N-8$ complex adders in delay feedback designs. The proposed architectures, however, can be pipelined at any level since these do not contain feedback loops. In contrast, the delay feedback architectures cannot be pipelined at any arbitrary level. The hardware cost of the proposed 128-point FFT architecture is summarized as follows:

- number of complex multipliers: 4+0.41, where the complexity of the constant multipliers is only 41% of the complex multiplier [26]
- number of complex adders: 28
- number of complex registers: 124, additional 64 registers are needed if an input buffer is needed.

Table 2.2 compares the hardware requirement, FFT algorithm, and throughput rate of the proposed and prior architectures for computing 128-point FFT. The hardware complexity is measured in terms of required number of complex multipliers (C.M.), adders (C.A.), and delay elements (REG) and throughput (TP). We can observe that the number of adders required in the proposed design is only 50% of those in the MDF architectures [25], [26]. The number of complex multipliers is almost the same compared

Table 2.2: Comparison of architectures for the computation of 128-point FFT

	# C.M.	# REG	# C.A.	TP
Proposed	4+0.41	124	28	4
[11]	4+0.41	190	28	4
[25]	2+4*0.62	124	48	4
[26]	4+0.41	252	52	4
[27]	7	220	48	4
[28]	2+4*0.62	148	42	4

to the previous designs as this depends on the underlying algorithm. The number of delay elements required in the proposed design is only 65% of those DIF based designs in [11]. When compared to the design in [25] including the input buffers, the proposed design requires extra 64 delay elements while the former design [25] requires another 20 complex adders. The cost of these two additional requirements will be comparable. The throughput of proposed architecture can be increased by adding more pipelining stages to increase the frequency of operation which is not possible in the design of [25].

2.8.1 Power Consumption

We compare power consumption of the serial feedback architecture with the proposed parallel feedforward architectures of same radix. The dynamic power consumption of a CMOS circuit can be estimated using the following equation,

$$P_{ser} = C_{ser} V^2 f_{ser}, \tag{2.21}$$

where C_{ser} denotes the total capacitance of the serial circuit, V is the supply voltage and f_{ser} is the clock frequency of the circuit. Let P_{ser} denotes the power consumption of the serial architecture.

In an L-parallel system, to maintain the same sample rate, the clock frequency must be decreased to f_{ser}/L. The power consumption in the L-parallel system can be calculated as

$$P_{par} = C_{par} V^2 \frac{f_{ser}}{L}, \tag{2.22}$$

where C_{par} is the total capacitance of the L-parallel system.

For example, consider the proposed architecture in Fig. 2.17 and R2SDF architecture [18]. The hardware overhead of the proposed architecture is 50% increase in the

number of delays. Assume the delays account for half of the circuit complexity in serial architecture. Then $C_{par} = 1.25C_{ser}$ which leads to

$$\begin{aligned} P_{par} &= 1.25C_{ser}V^2\frac{f_{ser}}{2} \\ &= 0.625P_{ser} \end{aligned} \tag{2.23}$$

Therefore, the power consumption in a 2-parallel architecture has been reduced by 37% compared to the serial architecture.

Similarly, for the proposed 4-parallel architecture in Fig. 2.21, the hardware complexity doubles compared to R2SDF architecture. This leads to a 50% reduction in power compared to serial architecture.

2.9 Conclusion

We presented a novel approach to derive the FFT architectures for a given algorithm. The proposed approach can be used to derive partly parallel architectures of any arbitrary parallelism level. Using this approach parallel architectures have been proposed for the computation of complex FFT based on radix-2^n algorithms. The DIT approach reduces the number of delay elements by 33% compared to DIF based designs. Further, a novel four parallel 128-point FFT architecture has been developed using proposed method. The hardware costs of delay elements and complex adders are reduced by using proposed scheduling approach. The number of complex multipliers is reduced using higher radix FFT algorithm. The throughput can be further increased by adding more pipeline stages which is possible due to the feed-forward nature of the design. The power consumption can be reduced by 37% and 50% in proposed 2-parallel and 4-parallel architectures, respectively.

Chapter 3

FFT Architectures for real-valued signals

In this chapter we present two approaches to design parallel-pipelined FFT architectures for real-valued signals (RFFT). Section 3.2 provides a brief review of prior work on FFT computation for real-valued signals. Section 3.3 presents a simple method to design RFFT architectures based on complex FFT architectures. This is followed by Section 3.4 which presents a novel methodology to design RFFT architectures using hybrid datapaths. Section 3.5 presents the proposed architectures based on radix-2^3 and radix-2^4 algorithms. In Section 3.6, the hardware complexity of the proposed architectures are compared with the prior art along with some simulation results.

3.1 Introduction

Fast Fourier Transform (FFT) is one of the widely used algorithms in digital signal processing and has been of interest for many years. Many FFT algorithms have been proposed following the essential idea of decimation either in the frequency or in time domain [12] - [17]. Numerous architectures have been developed based on these algorithms for different radices [16] - [37]. Most of these architectures assume the input signal to be complex.

There has been an increasing interest in the computation of FFT for real valued signals (RFFT), since virtually most of the physical signals are real. The real valued

signals which are of prime importance in real-time signal processing exhibit conjugate symmetry giving rise to redundancies. This property could be exploited to reduce both arithmetic and memory complexities.

The RFFT plays an important role in different fields such as communication systems, biomedical applications, sensors and radar signal processing. In multi-carrier modulation schemes FFT and IFFT are core functions discrete multi-tone (DMT) based transmission systems, which usually consume significant silicon area [38], [39]. A dedicated RFFT processor architecture will reduce the hardware complexity in DMT based technologies like very high bit-rate digital subscriber line (VDSL) [40] and asymmetric digital subscriber line (ADSL) [41].

In the area of sensor signal processing, the FFT is used in frequency domain beamforming, source tracking, harmonic line association and classification [42], [43]. On the other hand RFFT is one of the key algorithms in analyzing biomedical signals such as electrocardiography (ECG), and electroencephalography (EEG) [44]. Frequency domain based features like power spectral density (PSD) can identify and/or predict the abnormalities in these signals. A low complexity implementation of RFFT can reduce the power consumption in implantable or portable devices.

Even though specific algorithms for the computation of the RFFT [45] - [50] have been proposed in the past, these algorithms lack regular geometries to design pipelined architectures. These approaches are based on removing the redundancies of the complex FFT (CFFT) when the input is real and can be efficiently used in in-place architectures [51] or digital signal processors (DSP). A novel approach to design efficient pipelined architectures for RFFT was presented for the first time in [52].This architecture is based on modifying the radix-2 flow graph to achieve real datapaths and processes 4 samples in parallel. This approach is specific to radix-2 algorithm and was limited to a 4-parallel architecture. A general approach which can be extended to other radix-2^n algorithms is needed to take advantage of less number of multiplication operations.

The main contribution of this theis is a novel general methodology for designing pipelined FFT architectures for real valued signals. Unlike in prior design [52], this approach accommodates arbitrary level of parallelism and arbitrary power-of-2 radix designs. We introduce a combination of real and complex datapaths (referred to as

hybrid datapath) to achieve a generalized approach which can be extended to all radix-2^n algorithms. Further, folding approach in [11] can be used to design architectures for any arbitrary level of parallelism. The use of hybrid datapath allows us to use the same folding methods of [11]. The resulting architectures are very similar to the architectures for complex signals except for datapaths. This style of datapath deviates from that in [52] where all the datapaths are real. Novel 2-parallel and 4-parallel architectures are derived for radix-2^3, and radix-2^4 algorithms. The proposed architectures based on radix-2^3 and radix-2^4 algorithms require less multiplication complexity compared to prior RFFT architectures.

3.2 Prior RFFT Approaches

3.2.1 Algorithms for Computing RFFT

The CFFT architecture can be efficiently used to compute RFFT using two different algorithms. One such algorithm is the doubling algorithm [45], where an existing CFFT is used to calculate two RFFTs simultaneously. Another one is the packing algorithm [45], which forms a complex sequence of length $N/2$ taking the even and odd indexed samples of a real input sequence of length N, and computes the $N/2$-point CFFT of the complex sequence. In these algorithms, additional operations are necessary to obtain the final results.

Specific algorithms for the computation of RFFT can be used to reduce more number of operations. The first proposed algorithms were based on decimation in time (DIT) decomposition [46]. The basic idea of the algorithm is to compute only one half of the intermediate outputs, while the rest can be obtained by conjugating them. This will save almost half of the computations. This idea is extended to higher radices as well as split-radix [48], [49]. In [50], an algorithm for RFFT is proposed for decimation in frequency (DIF) decomposition making use of linear phase sequences.

3.2.2 Architectures for computing RFFT

The RFFT can be computed using the CFFT architecture (by setting the imaginary part to zero) which is the trivial solution. Further, doubling and packing algorithms can

be used to compute the RFFT in an efficient manner using CFFT architecture. Few architectures have been proposed for RFFT in the literature. An in-place architecture is proposed using the packing algorithm in [53]. In-place computation and memory schemes are developed to implement a variable size radix-4 RFFT processor [40].

An approach to design of VLSI architectures for RFFT is presented in [52]. In [52], the radix-2 flow graph is modified after removing the redundant operations to obtain a regular geometry. A 4-parallel pipelined architecture is derived based on the modified flow graph. The flow graph is modified such that all the data paths are real. Although it achieves a low complexity architecture, the design requires an additional memory of N real samples for the reorder buffer when the input samples arrive in natural order. The 4-parallel architecture is shown in Fig. 3.1. Further, the approach is a 4-parallel design specific to radix-2.

The higher radix algorithms lead to low multiplication complexity. For example, radix-2^4 algorithm leads to only one complex multiplication every 4 stages and radix-2^3 algorithm leads to one complex multiplication every 3 stages. But constant multipliers are required to implement trivial twiddle factors at the intermediate stages. To take advantage of higher radix algorithms, we propose a new approach to modify the FFT flow graph of any radix to obtain a regular geometry. The modified flow graph consists of both real and complex data paths. Further, 2-parallel and 4-parallel architectures are derived based on the modified flow graph using folding methodology [11]. Due to space constraints, the folding sets and corresponding derivations are not presented.

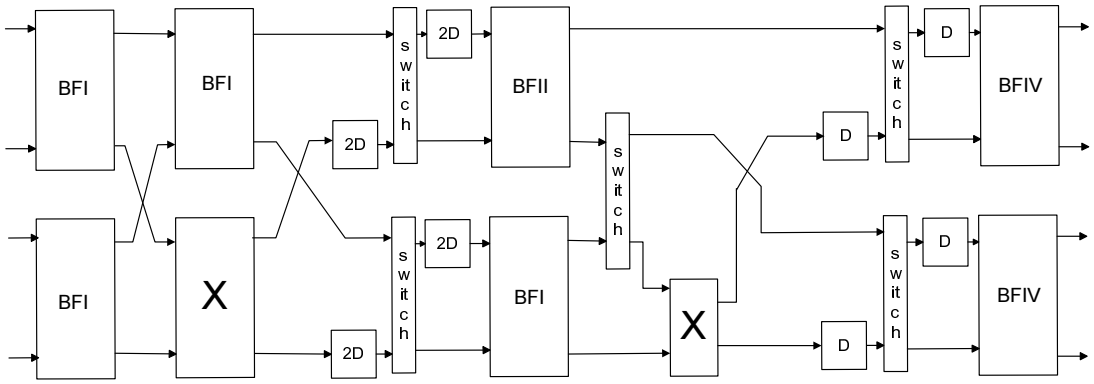


Figure 3.1: 4-parallel architecture for the computation of 16-point RFFT based on radix-2 algorithm [52].

3.3 Proposed Method 1

For RFFT, the input sequence is assumed to be real, i.e., $\forall n, x[n] \in \Re$. It is easy to show that, if $x[n]$ is real, then the output $X[k]$ is symmetric, i.e.,

$$X[N-k] = X^*[k]$$

Using this property, $\frac{N}{2} - 1$ outputs can be removed which are redundant. Most of the approaches in literature obtain the frequencies with indices $k = [0, N/2]$ or $k = [0, N/4] \bigcup [N/2, 3N/4]$. A new approach to find the redundant samples, which simplifies the hardware implementation has been proposed in [52]. We use this approach [52], to find the redundant samples in the flow graph. The shaded regions in Fig. 3.2 can be removed and only $N/2 + 1$ outputs of the FFT are needed.

The other simplification is that $Im(x[n]) = 0$, since inputs are real. According to this, every piece of data is real until it is multiplied by a complex number. Thus, the additions performed on this data are real. Thus we can save few adders required in implementing the FFT in hardware. We propose novel pipelined architectures based on these modifications which can process two samples in parallel. Two of these architectures are proposed in [54], which are derived using the new approach. Further this idea is extended to radix-2^3 algorithm.

3.3.1 2-parallel Radix-2 Architecture

The DFG of the radix-2 DIF FFT is shown in Fig. 2.16. The pipelined architecture for RFFT can be derived with the following folding sets similar to the CFFT architecture.

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\}, \\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\}, \\ C &= \{C3, C5, \phi, C0, C2, C4, \phi, C1\}, \\ D &= \{D2, D4, \phi, D1, \phi, D5, \phi, D0\} \end{aligned} \tag{3.1}$$

The architecture will be similar to the radix-2 DIF CFFT architecture shown in Fig. 2.17 except that the first two stages of the butterfly will contain a real datapath. The hardware complexity of this architecture is similar to the CFFT architecture. Although the proposed architecture and the architecture in [52] have same hardware complexity,

the proposed architecture is more regular. By extending the same approach to higher radices, multiplier complexity can be reduced.

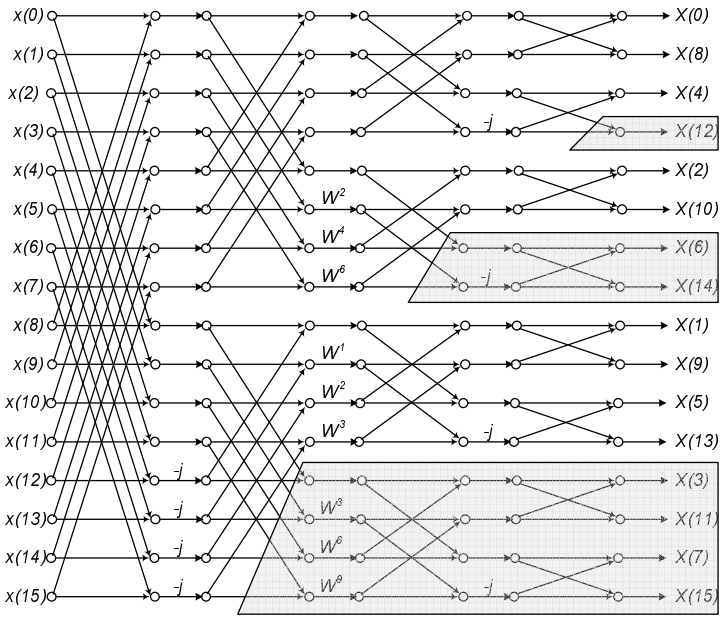


Figure 3.2: Flow graph of a radix-2^2 16-point DIF FFT. The boxed regions are redundant operations for RFFT.

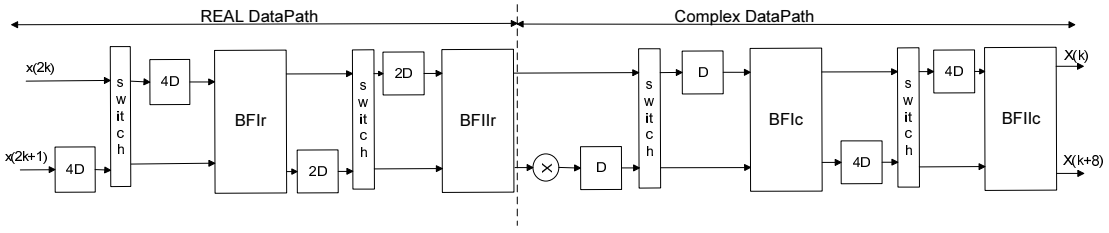


Figure 3.3: Proposed 2-parallel for the computation of 16-point radix-2^2 DIF RFFT.

3.3.2 2-parallel Radix-2^2 Architecture

The direct mapping of the radix-2^2 DIF FFT algorithm to a hardware architecture is simple. We can simply use the Radix-2^2 Single-path Delay Feedback (R2^2SDF) approach presented in [13], with just modifying the first complex butterfly stage into real. But, this cannot exploit the properties of the RFFT, where almost half of the output samples

are redundant. In R2^{2}SDF architecture, we can also observe that the utilization of multipliers will be 50% after the redundant operations are removed. Therefore, we propose a novel architecture for computing real FFT based on the flow graph shown in Fig. 3.2. The proposed architecture can process two input samples in parallel as opposed to serial architectures proposed in the literature. Two different architectures are derived using two different scheduling approaches, i.e., changing the folding order of the butterfly nodes.

Scheduling Method 1

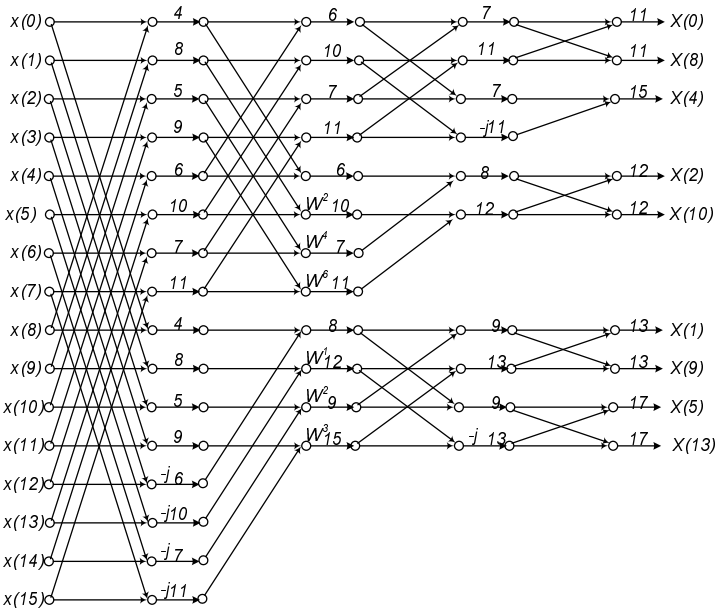


Figure 3.4: Simplified flow graph of a 16-point radix-2^2 DIF RFFT along with the proposed scheduling 1

Fig. 3.3 shows the proposed parallel-pipelined architecture for a 16-point DIF RFFT obtained from the flow graph in Fig. 3.2. The architecture can be derived using the

following folding sets.

$$A = \{A0, A2, A4, A6, A1, A3, A5, A7\},$$
$$B = \{B5, B7, B0, B2, B4, B6, B1, B3\},$$
$$C = \{C3, C5, \phi, C0, C2, C4, \phi, C1\},$$
$$D = \{D2, D4, \phi, D1, \phi, D5, \phi, D0\}$$

The nodes from $A0, ...A7$ represent the 8 butterflies in the first stage of the FFT and $B0, ...B7$ represent the butterflies in the second stage and so on. Assume the butterflies $B0, ..., B7$ have only one multiplier at the bottom output instead of on both outputs. This assumption is valid only for the RFFT case due to the redundant operations. This radix-2^2 feedforward pipelined architecture maximizes the utilization of multipliers and processes 2 input samples per clock cycle. As shown in the flow graph, all the edges in the first two stages carry real samples and later stages carry complex samples. Therefore, the radix-2 butterflies in the first two stages process two real inputs and they consist of only a real adder and a real subtractor. The butterflies and the multipliers in the remaining stages operate on complex inputs. In a general case of N-point RFFT, with N power of 2, the architecture requires $log_2(N) - 1$ real butterflies, $log_4(N) - 1$ complex multipliers and $9N/8 - 2$ delay elements or buffers.

The scheduling for the proposed architecture in Fig. 3.3 is shown in Fig. 3.4. The numbers on the edges indicate the clock cycle numbers in which those intermediate samples are computed. The first two stages compute only real butterflies. According to the input order of the data, the first butterfly computes the pairs of samples in the following order:(0,8), (2,10), (4,12), (6,14), (1,9), (3,11), (5,13), (7,15). This circuit first processes the even samples and then the odd samples.

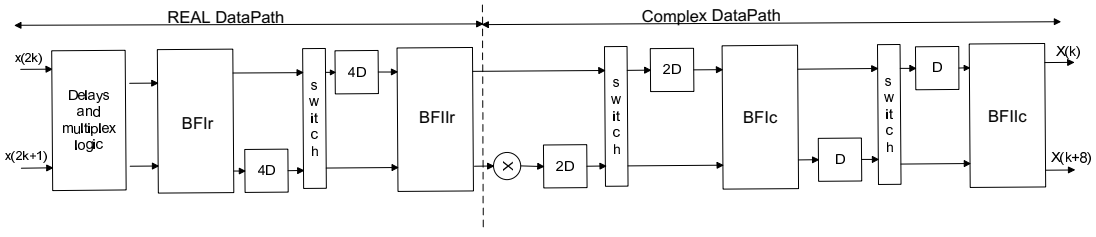
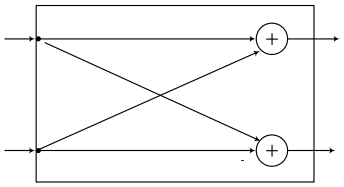
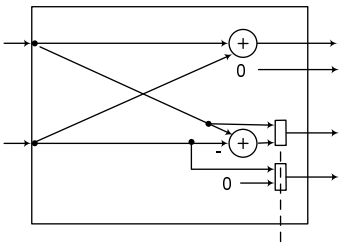


Figure 3.5: Proposed 2-parallel for the computation of 16-point radix-2^2 DIF RFFT.



(a) BF2Ir



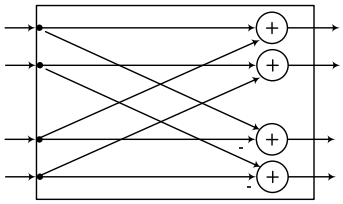
(b) BF2IIr

Figure 3.6: Butterfly structure for the proposed FFT architecture in the real datapath

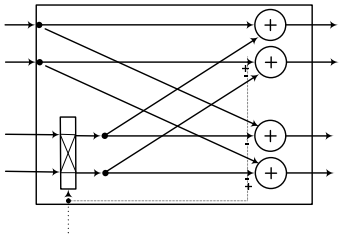
We can observe the similarity of the data-path to Radix-2 Multipath delay commutator (R2MDC) and the reduced number of multipliers. The implementation uses four types of butterflies as shown in the architecture. BF2Ir and BF2Ic are regular butterflies which handle real and complex data, respectively as shown in Fig. 3.6 and 3.7. Although there is a multiplicative factor "$-j$" after the first stage, the first two stages consists of only real-valued datapath. We need to just combine the real and imaginary parts and send it as an input to the multiplier in the next stage. For this, we do not need a full complex butterfly stage. The factor $-j$ is handled in the second butterfly stage using a bypass logic which forwards the two samples as real and imaginary parts to the input of the multiplier. The adder and subtractor in the butterfly remains inactive during that time. Fig. 3.6b shows BF2IIr, a regular butterfly which handles real data and also contains logic to implement the twiddle factor "$-j$" multiplication after the first stage. Fig. 3.7b shows the complex butterfly structure for BF2IIc and contains logic to implement the twiddle factor $(-j)$ multiplication.

Scheduling Method 2

Another way of scheduling is proposed which modifies the architecture slightly and also reduces the required number of delay elements. In this scheduling, the input samples



(a) BF2Ic



(b) BF2IIc

Figure 3.7: Butterfly structures for the proposed architecture in the complex datapath

are processed sequentially, instead of processing the even and odd samples separately. This can be derived using the following folding sets.

$$A = \{A0, A1, A2, A3, A4, A5, A6, A7\},$$
$$B = \{B4, B5, B6, B7, B0, B1, B2, B3\},$$
$$C = \{C2, C3, C4, C5, \phi, \phi, C0, C1\},$$
$$D = \{D1, D2, \phi, D4, D5, \phi, \phi, D0\}$$

The nodes from $A0, ...A7$ represent the 8 butterflies in the first stage of the FFT and $B0, ...B7$ represent the butterflies in the second stage. Assume the butterflies $B0, ..., B7$ have only one multiplier at the bottom output instead of both outputs.

Fig. 3.5 shows the modified architecture and the corresponding scheduling is shown in Fig. 3.8. In a general case of N-point RFFT, with N power of 2, the architecture requires $log_2(N) - 1$ real butterflies, $log_4(N) - 1$ complex multipliers and $N - 2$ delay elements or buffers. We can observe that the savings in number of delay elements is due to the last stage. In the last stage, we need to store complex samples, i.e., double delay elements are required to store both real and imaginary parts. By decreasing the number of delay elements in this stage, we are able to reduce the total number of delay elements

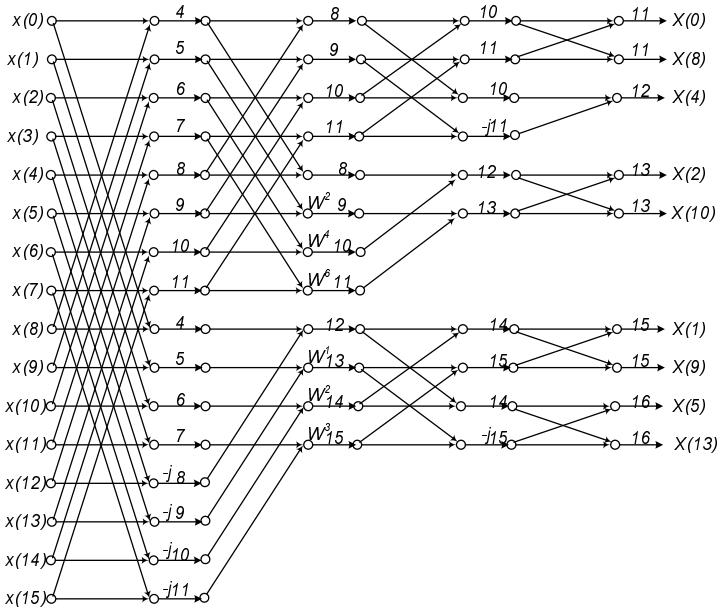


Figure 3.8: Simplified flow graph of a 16-point radix-2^2DIF RFFT along with the proposed scheduling 2

required. The control logic becomes slightly complex for this scheduling as shown in the first stage. Register minimization techniques [55] are used to find the optimal number of registers required for this scheduling. $N/2$ registers are needed in the first stage to achieve the required scheduling.

3.3.3 Radix-2^3

The approach of radix-2^2 RFFT architecture can be extended to radix-2^3 algorithm which has low multiplier complexity. The radix-2^3 FFT algorithm is shown in Fig. 3.9. The shaded regions show the redundant operations for RFFT. We can observe that only one multiplier is required at the end of the third butterfly column due to redundancy. Two multipliers are required for a complex FFT as shown in Fig. 2.25.

The parallel architecture can be derived using the same kind of folding sets described in Scheduling Method I. The folded architecture is shown in Fig. 3.10. For an N-point RFFT with N power of 2^3, the architecture requires $log_8(N) - 1$ complex multipliers and $N - 2$ delay elements or buffers. It also requires constant multipliers to perform

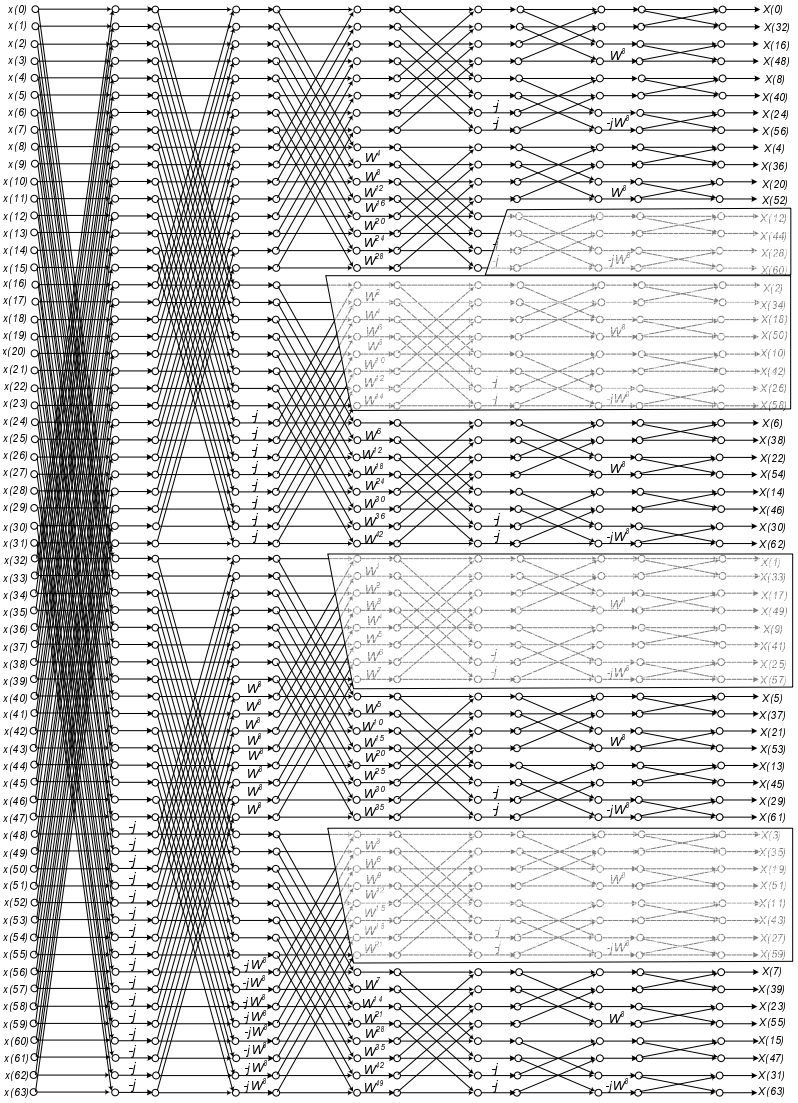


Figure 3.9: Flow graph of a 64-point radix-2^3 DIF RFFT

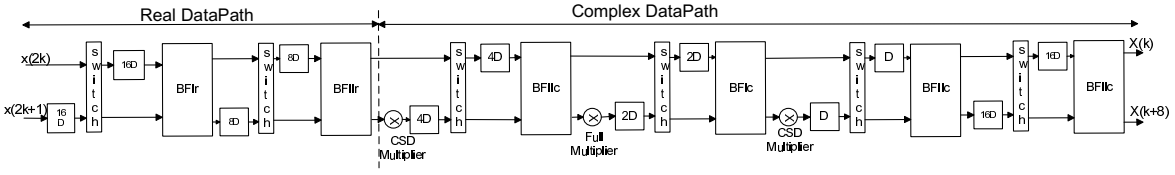


Figure 3.10: Proposed 2-parallel for the computation of 64-point radix-2^3 DIF RFFT.

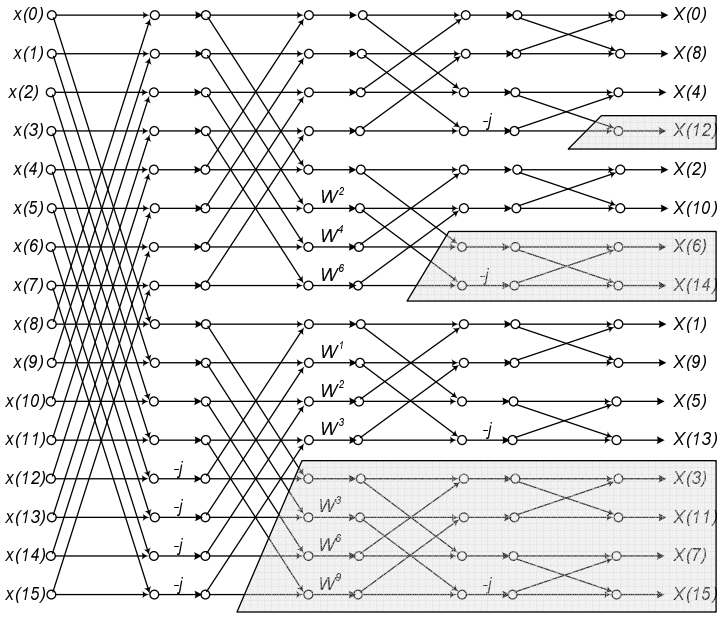


Figure 3.11: Flow graph of a 16-point radix-2^2 DIF FFT.

the trivial multiplication operations. The advantage of this architecture is its reduced multiplier complexity.

Similar to 4-parallel radix-2 CFFT, 4-parallel radix-2^2 and radix-2^3 architectures for RFFT can be derived using the folding sets of the same pattern.

3.4 Proposed Methodology 2

We propose a novel methodology to develop pipelined architectures based on removing the redundant operations. The proposed methodology consists of three steps which are explained below.

3.4.1 Modifying the flow graph

In the first step, the FFT flow graph is modified by removing the redundant samples. Most of the approaches in literature compute the frequencies with indices $k = [0, N/2]$ or $k = [0, N/4] \bigcup [N/2, 3N/4]$. We use the approach proposed in [52] to find the redundant samples in the flow graph. Fig. 3.11 shows the flow graph of 16-point FFT decimated in

frequency. The boxed regions show the redundant samples which can be removed from the flow graph. These samples and the corresponding computations can be removed from the flow graph. The flow graph after removing the redundant samples will be irregular. It could be used to implement efficient in-place architectures, but not to efficient pipelined architectures.

We need to modify the flow graph to obtain a regular geometry to simplify the hardware implementation. Fig. 3.12 shows the modified flow graph which is obtained by scheduling the imaginary operations in place of redundant operations. All the edges in Fig. 3.12 are real, i.e., the data have been separated into real and imaginary parts. The continuous edges compute the real parts and the broken edges represent the computations of the imaginary parts. The output samples of the flow graph include a letter (r or i) to indicate if the value corresponds to the real part or the imaginary part of the output.

Further, in higher point FFTs, a complex sample (real and imaginary components computed separately) may need to be multiplied by a complex twiddle factor, i.e., a full complex multiplier is required. In that situation, the proposed modifications will not lead to a regular flow graph. This problem can be solved by reordering the data before the multiplier to have corresponding samples at the input of the multiplier at the same time. At the architecture level, this problem can be solved by introducing an extra stage of reordering circuit before the multiplier. We modify the multiplier stage similar to that of the butterfly stage by adding a reordering circuit. This will increase the latency by a few cycles and the number of delay elements depending on the stage at which multipliers are required. This situation will be described with an example in Section IV.

3.4.2 Hybrid datapaths

It can be seen that the modified flow graph involves both real and complex operations. Some of these paths which involve real and complex numbers are pointed out in Fig. 3.12. The multiplication operation at each stage determines whether the datapath is real or complex. For example, in the radix-2^2 algorithm, multiplication operation is required at every alternate stage and the datapath will be complex at every other stage.

Further, we can observe that the butterfly operations sometimes consist of both

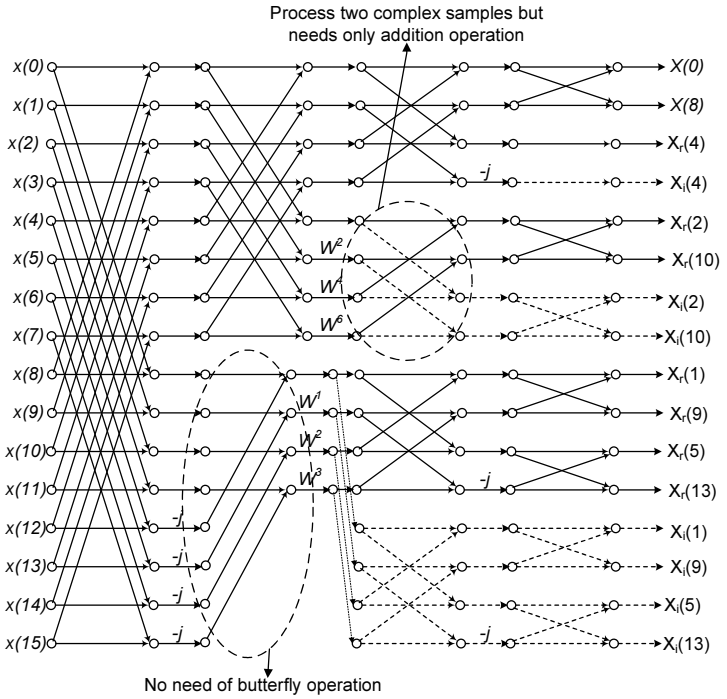


Figure 3.12: Modified flow graph of a 16-point radix-2^2 DIF FFT.

real and complex numbers. We propose new butterfly structures to handle these real and complex operations as the regular butterfly structure can handle only either the real or complex numbers. We identify three different possible scenarios in the modified flow graphs of radix-2^n algorithms. The first scenario is the straightforward one which involves only real inputs. Fig. 3.13 shows the basic butterfly structure (BFI) which operates on two real inputs, and thus, only consists of a real adder and real subtractor.

The second scenario is when the butterfly needs to compute a regular butterfly operation or just pass through the input samples without any processing. This scenario can be observed in the second stage of the flow graph in Fig. 3.12. We can observe that the butterfly unit in this stage needs to work as regular butterfly half the time and transfer the input samples to the output for remaining half the time. Fig. 3.14 shows the new butterfly structure (BFII) which operates on real inputs along with the possible outputs. BFII butterfly processes two real inputs and generates two real outputs (real and/or imaginary components). Thus, it requires only two real adders.

The third scenario is when the butterfly needs to operate on real inputs and complex inputs at different time instances. The example of such a scenario can be observed in the third stage of the flow graph in Fig. 3.12. The third stage of operations consists of complex data in the second quarter. During this time, the butterfly needs to add only two complex inputs. The subtraction is not required in contrast to the regular butterfly due to the removal of redundant operations. Two real adders are sufficient to add two complex numbers. The remaining operations in the third stage are regular butterfly operations which process only real or imaginary data. Fig. 3.15 shows the butterfly structure (BFIII) which can operate as regular butterfly with real inputs or a modified butterfly with complex inputs. BFIII also requires only two real adders. The possible outputs of the butterfly are also shown in the Fig. 3.15. The control signal to the multiplexers will define the output.

3.4.3 Folding

The data flow graph (DFG) can be derived by identifying the correct butterflies in the modified flow graph. As an example, consider the modified flow graph shown in Fig. 3.12. The corresponding DFG is shown in Fig. 3.16. The DFG consists of both real and complex datapaths. The bold lines represent the complex datapath while the rest

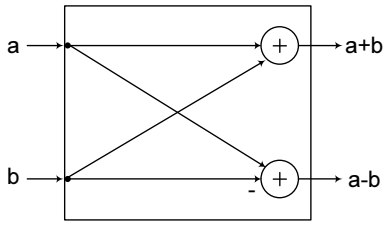


Figure 3.13: Butterfly structure BFI for the proposed FFT architecture

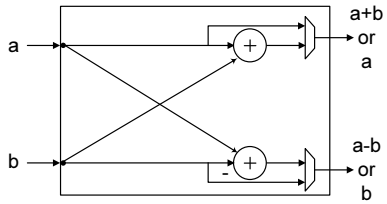


Figure 3.14: Butterfly structure BFII for the proposed FFT architecture

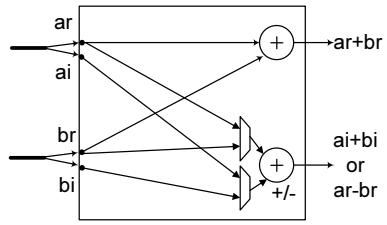


Figure 3.15: Butterfly structure BFIII for the proposed FFT architecture

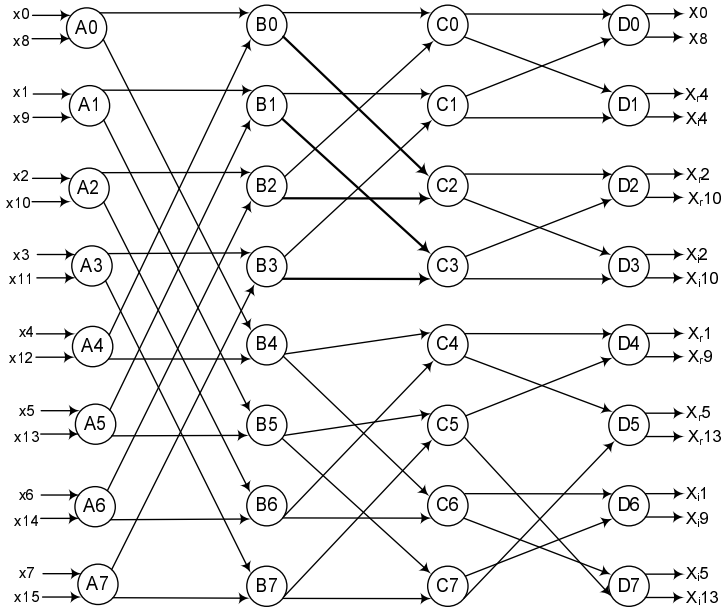


Figure 3.16: Modified flow graph of a 16-point radix-2^2 DIF FFT.

represent real datapath. The nodes in the DFG can be mapped to different butterfly structures described earlier. The nodes A, B, C, D can be mapped to BFI, BFIV, BFIII, BFI respectively. Now, folding technique can be used to derive the pipelined architecture from the data flow graph. The folding approach is used in [11] to optimize the datapath. A similar approach is followed in this paper to derive the proposed architectures.

2-parallel architecture To derive the 2-parallel architecture, consider the folding sets:

$$\begin{aligned} A &= \{A0, A2, A4, A6, A1, A3, A5, A7\}, \\ B &= \{B5, B7, B0, B2, B4, B6, B1, B3\}, \\ C &= \{C3, C5, C7, C0, C2, C4, C6, C1\}, \\ D &= \{D2, D4, D6, D1, D3, D5, D7, D0\} \end{aligned} \tag{3.2}$$

4-parallel architecture To derive the 4-parallel architecture divide the nodes into two groups. The nodes in the same group are processed by the same computation unit. A

4-parallel architecture can be derived using the following folding sets.

$$\begin{aligned} A &= \{A0, A2, A4, A6\} & A' &= \{A1, A3, A5, A7\} \\ B &= \{B1, B3, B0, B2\} & B' &= \{B5, B7, B4B6\} \\ C &= \{C2, C1, C3, C0\} & C' &= \{C6, C5, C7, C4\} \\ D &= \{D3, D0, D2, D1\} & D' &= \{D7, D4, D6, D5\} \end{aligned} \tag{3.3}$$

The mapping of nodes to different butterfly structures can be different in the case of 4-parallel architecture. The nodes $\{B4, ..., B7\}$ can be implemented with only a complex multiplier instead of BFIV structure, as these nodes consists of only complex multiplication operation. The reader can refer to [11] for the complete folding equations and register minimization techniques needed to derive the pipelined architecture.

3.5 Proposed Architectures

In this section, we present 2-parallel and 4-parallel architectures for real FFT computation based on radix-2^3 and radix-2^4 algorithms using the proposed methodology. Further, folding sets can be modified to derive L-parallel architectures of any N-point RFFT. To illustrate the proposed architectures, we consider examples of $N = 64$, $N = 128$.

3.5.1 Radix-2^3

The proposed radix-2^3 based architectures are described using the example of $N = 64$ point FFT. The flow graph is modified by removing the redundant samples and rescheduling the imaginary operations in place of the removed operations. The modified flow graph is shown in Fig. 3.17. The broken lines represent the imaginary parts of the computation, while continuous lines represent the real part of the computation. We can observe that the flow graph is regular and leads to a pipelined architecture. The intermediate samples need to be reordered (swapped) so that the real and imaginary components are aligned correctly before multiplied by a twiddle factor. For example, the output samples at the 3rd stage should be multiplied by a twiddle factor, but the real and imaginary components are computed separately. This problem is solved by

reordering the samples before the multiplier. Fig. 3.18 shows the design of multiplier block used in the proposed architectures. The multiplexers on the left can either pass through the samples or swap the samples separated by n clock cycles. The multiplexers on the right will control the output depending on whether the input samples need to be multiplied by a twiddle factor.

The multiplication with the "-j" factor can be implemented in a similar way except that we do not need a multiplier. The real and imaginary parts need to be interchanged. We introduce a swap block whenever only a "-j" operation in required. We observed that "-j" twiddle factors occur only at the bottom outputs of the butterflies in radix-2^3 and radix-2^4 algorithms. Fig. 3.19 shows the design of the swap block used in the proposed architectures. The swapping of real and imaginary parts is required only at the bottom output of butterflies as shown in the Fig. 3.19.

2-parallel architecture

The next step is to determine the butterfly structures required to design the pipelined architecture. In a 2-parallel design, all the butterfly computations in one stage are executed using one computation unit. The folding sets for this example are similar to (3.2), except extending them for 64-point. The first stage is straightforward as it processes real inputs and BFI structure can be used. In the second stage, the butterflies process real samples for half the time and complex samples for the rest of the time. The complex samples consists of only either a real or a imaginary component and only passthrough operation is sufficient instead of a regular butterfly operation. BFII butterfly structure is used as the computation unit for the second stage. Similarly, BFII, BFIII, BFII, BFII butterfly structures are used for 3rd, 4th, 5th stage and 6th stages, respectively. Further, the multiplier after the 3rd stage is modified similar to that of a butterfly stage to enable the dataflow according the flow graph. An extra reordering circuit is inserted before the complex multiplier to reorder the samples into required order as shown in Fig. 3.18. The final architecture can be derived by folding the flow graph using the above computation units.

Fig. 3.20 shows the proposed 2-parallel pipelined architecture for a 64-point DIF FFT derived from the flow graph of Fig. 3.17. This radix-2 feedforward structure achieves 100% utilization of the hardware components, and achieves a throughput of

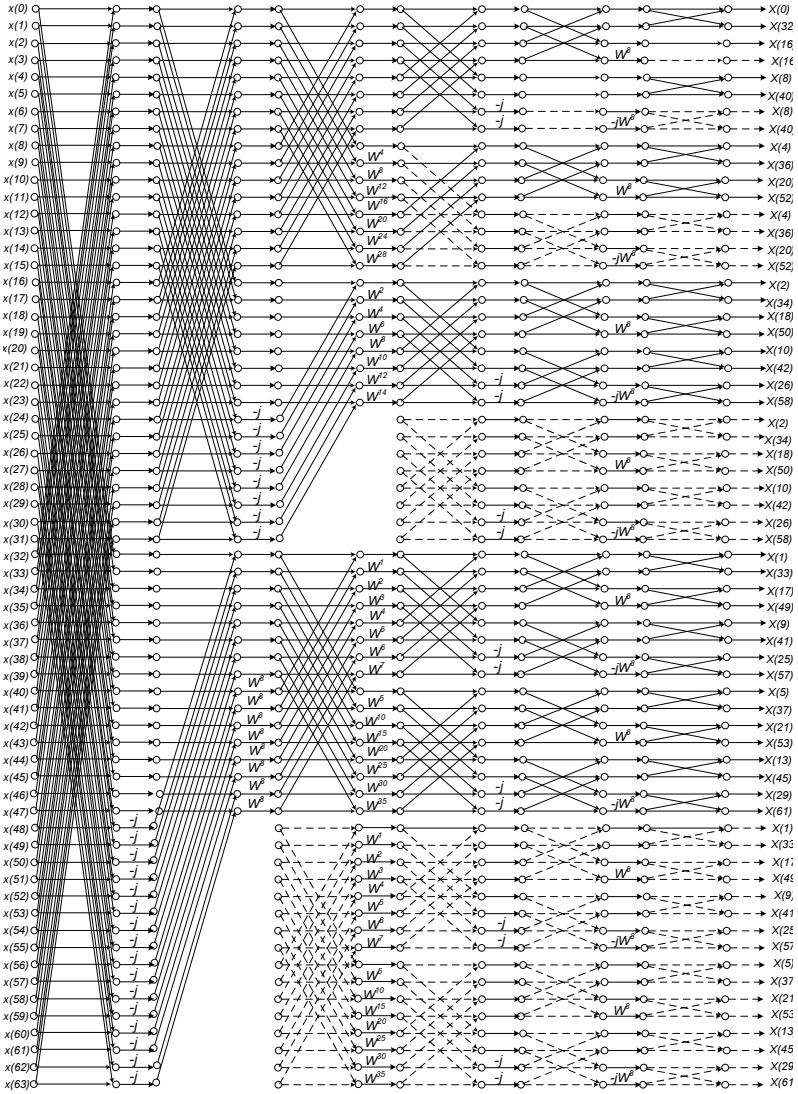


Figure 3.17: Modified flow graph of 64-point radix-2^3 DIF FFT. The samples at the continuous and broken lines indicate real components and imaginary components respectively.

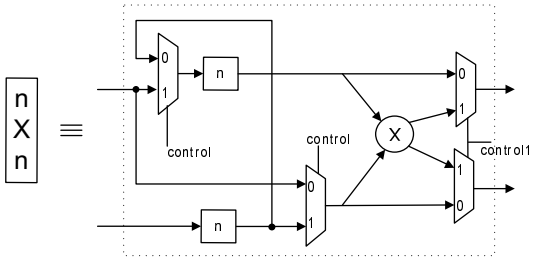


Figure 3.18: Structure of the multiplier block in the proposed architectures

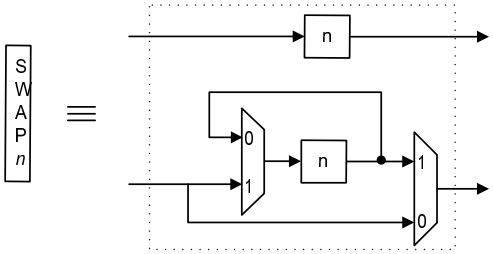


Figure 3.19: Structure of the swap block in the proposed architectures

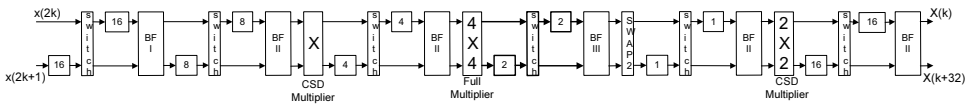


Figure 3.20: Proposed 2-parallel architecture for radix-2^3 64-point DIF FFT. The bold lines indicate complex data paths.

2 samples per clock cycle. The design consists of both real and complex datapaths as shown in Fig. 3.20. The bold lines represent the complex datapath while the rest represent real datapath. Three different butterfly structures (BFI, BFII and BFIII) are used to handle the complex and real datapaths. It requires only one full complex multiplier and two constant multipliers. The constant multipliers involve the multiplication of the input with the special twiddle factor W^8. It can be implemented with 2 addition operations and a scaling operation by a constant $(1/\sqrt{(2)})$. The constant scaling operation can be implemented using canonic signed digit (CSD) multiplication [55]. '$-j$' operation can be handled either in the multiplier or using a swap block shown in Fig. 3.19. Fig. 3.20 shows a swap block after 4th stage to handle "-j" operation. The swap block just interchange the corresponding real and imaginary components.

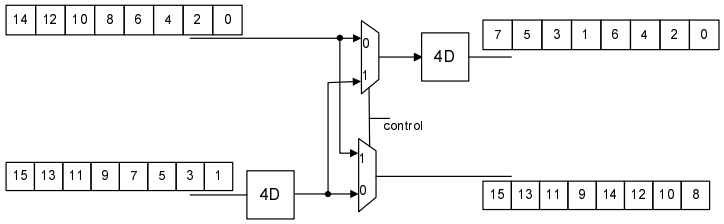


Figure 3.21: Reordering structure (switch and delay elements) of the pipelined architecture. This example shows the reordering in the first stage of a 16-point RFFT.

The rest of the datapath consists of delay elements and switch components (multiplexers) for reordering the samples according to the flow graph. The data flow of the switch and delay elements is shown in Fig. 3.21. The function of the switch and delay elements is to reorder the incoming samples to provide the corresponding samples at the input of each butterfly stage according to the data flow during every clock cycle.

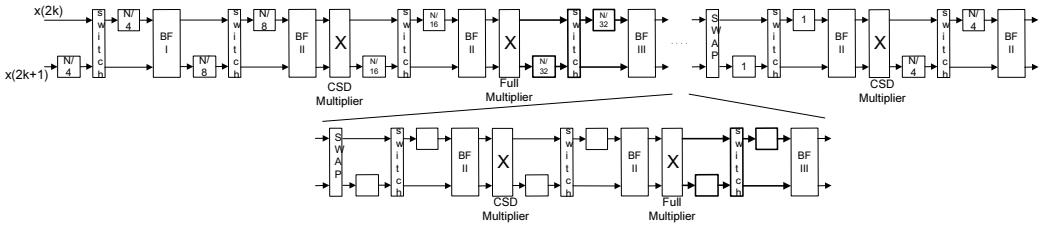


Figure 3.22: Proposed N-point 2-parallel architecture for radix-2^3 DIF FFT, where N is a power of 8. The bottom structure will be repeated depending on N.

The control signal controls the multiplexers which connects the input and output of the switch in two different ways (either straight or cross paths). The control signals for switches in different stages of the architecture can be generated by using simple counter logic. Fig. 3.21 shows the switch and delay elements in the third stage of the 64-point RFFT architecture along with the input and output order of the samples. The control signal should be 0 for 4 clock cycles and 1 for the next 4 clock cycles, i.e., signal needs to toggle every 4 clock cycles to generate the output order. Thus, 3^{rd}-bit of a 5-bit counter can be used as the control signal in this example. In a similar fashion, other bits in the 5-bit counter can be used as the control signals for the remaining switch components.

Fig. 3.22 shows a general N-point 2-parallel architecture based on radix-2^3 algorithm, where N is a power of 8. The bottom part will be repeated $log_8 N - 2$ times. In general, for an N-point RFFT, with N power of 2, the 2-parallel architecture requires $2log_2 N$ real adders, $log_8 N - 1$ complex multipliers, $2(log_8 N - 1)$ CSD (W^8) multipliers and $3N/2 - 4$ real delay elements. Few extra delays are required for interchanging real and imaginary components before twiddle factor multiplications.

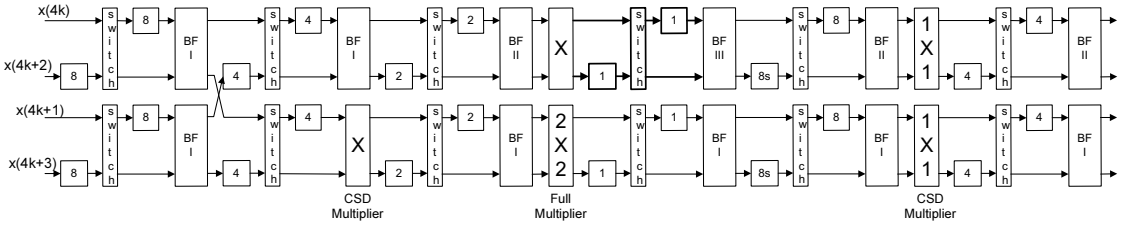


Figure 3.23: Proposed 4-parallel architecture for radix-2^3 64-point DIF FFT.

4-parallel architecture

A 4-parallel architecture can be derived from the flow graph shown in Fig. 3.17. We need to determine the folding sets and the computation units. Two computation units are available for each stage. The first stage can be computed using two BFI units since all the operations involve only real components. In the second stage, the top half can be computed using a BFI unit and the bottom half does not need a butterfly operation. Similarly, the rest of the computation units can be determined. The proposed 4-parallel architecture for radix-2^3 64-point FFT is shown in Fig. 3.23. This architecture processes

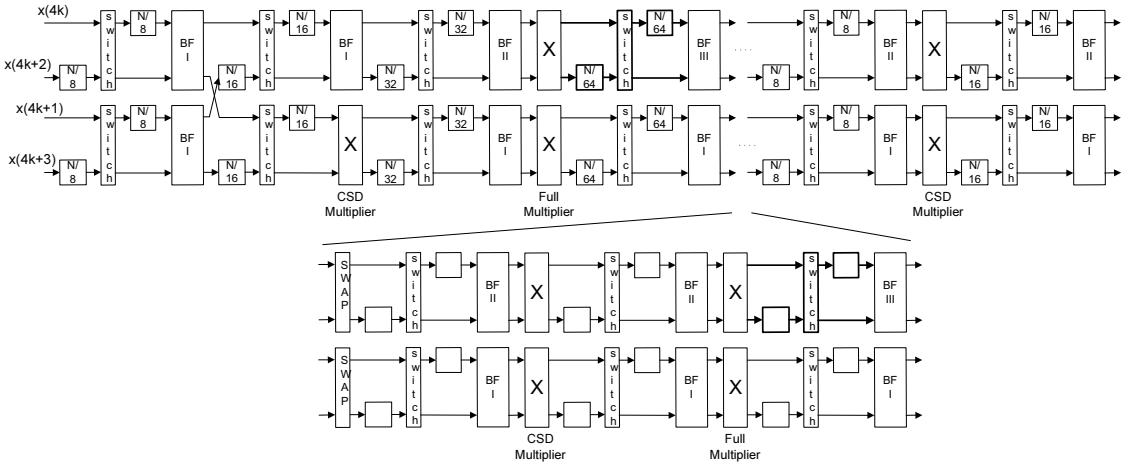


Figure 3.24: Proposed N-point 4-parallel architecture for radix-2^3 DIF FFT, where N is a power of 8. The bottom structure will be repeated depending on N.

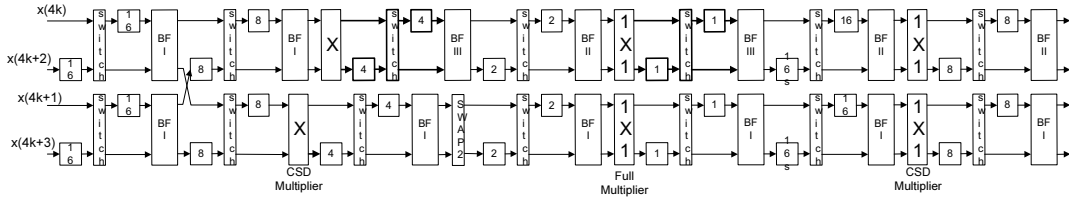


Figure 3.25: Proposed 4-parallel architecture for radix-2^4 128-point DIF FFT.

4 consecutive samples in parallel. The upper butterfly operates on even samples, i.e., (0, 32), (4, 36), (8, 40) and so on and the lower butterfly processes the odd samples, i.e., (1, 33), (5,37), (9, 41), etc. Similar to the 2-parallel design, this architecture consists of both real and complex data paths.

The 4-parallel architecture requires three types of butterfly structures (BFI, BFII, and BFIII) to handle different combinations of real and complex data. Only two full multipliers are required in this design. The rest of the three multipliers can be implemented using CSD logic. Fig. 3.22 shows a general N-point 2-parallel architecture based on radix-2^3 algorithm, where N is a power of 8. The bottom part will be repeated $log_8 N - 2$ times. In a general case of N-point RFFT, the 4-parallel architecture requires $4log_2 N$ real adders, $2(log_8 N - 1)$ complex multipliers, $4log_8 N - 5$ CSD (W^8) multipliers and $7N/4 - 8$ delay elements.

3.5.2 Radix-2^4

The proposed radix-2^4 2-parallel architecture is explained using $N = 128$ point FFT. The advantage of radix-2^4 algorithm is that it needs only one full multiplier every four stages. The radix-2^4 algorithms are described in detail in [23]. We can modify the flow graph similar to the other radices. We schedule the imaginary operations of the required frequencies in place of redundant operations.

4-parallel architecture

The folding sets similar to (3.2) are used to derive the 4-parallel architecture. Three different butterfly structures are necessary to handle the real and complex datapaths. Similar to radix-2^3 architectures, complex multipliers need to operate on samples computed at different time instances. Multipliers with reordering circuits shown in Fig. 3.18 are required at these stages to have corresponding samples at the input of the multiplier.

Fig. 3.25 shows the 4-parallel architecture for computing RFFT based on radix-2^4 algorithm. We can observe that the architecture needs three different butterfly structures (BFI, BFII, BFIII). It consists of two parallel data paths processing two input samples. Each data path consists of seven butterfly units, four constant and two full complex multipliers, delay elements and multiplexers. The function of delay elements and switches is to store and reorder the input data until the other available data is received for the butterfly operation. The four output data values generated after the first stage are multiplied by constant twiddle factors ($W_8^1 = e^{-j2\pi/8}, W_8^3 = e^{-j2\pi 3/8}$). These twiddle factors can be implemented efficiently using canonic signed digit (CSD) approach. The outputs after the third stage are multiplied by the nontrivial twiddle factor. Another constant multiplier stage is required before the sixth butterfly stage. The CSD complex constant multiplier processes the multiplication of twiddle factors $W^8, W^{16}, W^{24}, W^{48}$. These twiddle factors correspond to $cos(\pi/8), sin(\pi/8)$, and $cos(\pi/4)$.

Table 3.1: Comparison of architectures for the computation of N-point RFFT

Architecture	# C.M	# Real Adders	# Real Delays	Throughput
R2MDC	$2(log_4N-1)$	$4log_2N$	$2(3N/2-2)$	1
R2SDF	$2(log_4N-1)$	$4log_2N$	$2(N-1)$	1
R4SDC	(log_4N-1)	$3log_2N$	$2(2N-2)$	1
R4MDC	(log_4N-1)	$3log_2N$	$2(2N-2)$	1
R2^2SDF	(log_4N-1)	$4log_2N$	$2(N-1)$	1
R2^3SDF	(log_8N-1)	$4log_2N$	$2(N-1)$	1
radix-2 [52]	$2(log_4N-1)$	$4log_2N$	$2N$	4
radix-2^2 [54]	log_4N-1	$4log_2N-2$	$2(N-1)$	2
Proposed Architectures				
radix-2^3 (2-parallel)	log_8N-1	$2log_2N$	$<2N$	2
radix-2^3 (4-parallel)	$2(log_8N-1)$	$4log_2N-2$	$<2N$	4
radix-2^4 (4-parallel)	$2(log_{16}N-1)$	$4log_2N-2$	$<2N$	4

C.M - Complex Multipliers

3.6 Comparison and Analysis

Table 7.3 compares the hardware complexity and the throughput of the previous architectures and the proposed ones for computing an N-point RFFT. The hardware complexity of the architectures depend on the required number of multipliers, adders, delay elements. The performance is represented by throughput. Three different multipliers are necessary to implement the proposed architectures. The proposed design and the designs in [52] and [54] are the only specific approaches for the computation of RFFT. The other approaches are not specific and can be used to compute FFT with complex inputs. The number of delays in [52] is N/2 complex delays and requires another N/2 delays for the input buffers. The proposed architectures include input buffers, for fair comparison, we add extra N/2 complex delays in the Table 7.3. The proposed architectures are feed-forward. Further pipelining stages can be added as necessary to increase the frequency of operation. We can observe that the number of multipliers required in the radix-2^3 and radix-2^4 architectures are less compared to the previous designs. Further to get a better picture of the hardware complexities, we present the

Table 3.2: Multipliers required for the computation of different N-point RFFTs

Architecture	64-point	128-point	1024-point
[54](2-parallel)	2	3	4
radix-2^3 (2-parallel)	1.2	2.2	3.3
[52](4-parallel)	4	5	8
radix-2^3 (4-parallel)	2.3	4.3	6.5
radix-2^4 (4-parallel)	-	2.6	5

* Fractional parts represent the complexity of CSD multipliers compared to one full complex multiplier

examples of 64, 128 and 1024-point RFFT computations to get a better comparison of the hardware complexity. Table 3.2 show the comparison of multipliers required in the proposed architectures and the prior designs for the case of 64, 128 and 1024-point RFFT, respectively. We can observe that the proposed radix-2^3 architectures have less hardware complexity compared to the previous designs. Compared to [52], the proposed radix-2^3 and radix-2^4 4-parallel designs save 2 and 3 complex multipliers, respectively, for 64 and 1024 point FFTs.

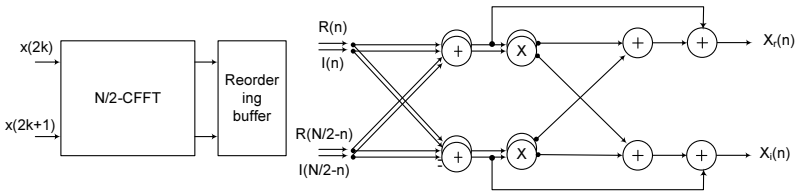


Figure 3.26: RFFT computation using packing algorithm along with the additional stage required.

Many parallel pipelined architectures to compute FFT with complex inputs have been proposed in the literature based on radix-2^n algorithms. Packing algorithm can be used to compute N-point RFFT using an N/2-point CFFT architecture. To compute the outputs of the RFFT from N/2-complex FFT, extra addition and multiplication operations are required. Fig. 3.26 shows the additional stage required to compute the RFFT outputs from the N/2-point CFFT. It can be observed that it requires 8 real adders (equivalent to 2 complex butterflies) and 4 real multipliers. Further, extra delays and multiplexers are required for reordering the CFFT outputs before the additional stage. The number of delays depends on the CFFT architecture. In general, the

RFFT computation using packing algorithm requires log_2N+1 complex butterfly stages compared to log_2N real butterfly stages in the proposed architectures which leads to a saving of more than 50% of the adders in the proposed architecture. Further 4 extra real multipliers are also required for the packing algorithm. Thus, the proposed RFFT architectures lead to low hardware complexity compared to the architectures based on the packing algorithm.

Table 3.3: Synthesis Results

Algorithm	radix-2^4
FFT size	128
Frequency	500MHz
Area	$57565\mu m^2$
Power	11.9mW

The proposed 4-parallel 128-point radix-2^4 architecture was implemented using STMicroelectronics 65nm CMOS technology. Verilog RTL descriptions were synthesized using Synopsys Design Compiler to obtain a gate-level netlist. The functionality is verified both at the verilog and gate-level simulations. The synthesis results for a 128-point RFFT are presented in Table 3.3.

The hardware complexity overhead due to the multiplixers in the new butterfly structures will be nominal. The number of additional multiplexers in the proposed architectures and prior CFFT architectures will be almost similar. The CFFT architectures based on radix-2^n algorithms consists of multiplication "-j" factor, which is computed in the butterfly stage by exchanging the real and imaginary components using extra multiplexers [13].

3.7 Conclusion

We proposed a generalized approach to design efficient architectures for the computation of RFFT. The approach can be extended to radix-2^5 and higher radix algorithms. In particular, novel 2-parallel and 4-parallel pipelined architectures are developed based on the modified flow graph and hybrid datapath design using radix-2^3 and radix-2^4 algorithms. Folding methodology is used to optimize the data path. The proposed architectures lead to low hardware complexity compared to the prior designs. Further

higher parallel architectures can be developed using the proposed approach. Parallel pipelined architectures can also be derived for decimation-in-time (DIT) algorithms. In case of complex FFT, it was observed that for higher-level parallelism, the DIT architectures require less delay elements than DIF [56]. While not proven, the same observation is expected to hold good for RFFT as well.

Chapter 4

In-Place FFT Architectures

This chapter presents memory-based FFT architectures for both complex and real-valued signals. Section 4.2 discusses the proposed architecture for complex signals using radix-2^2 butterfly as the processing element. In Section 4.3, the proposed architecture for real-valued signals is presented. Afterwards, Section 4.4 presents the addressing scheme to compute inverse FFT of Hermitian-symmetric data. The proposed architectures are compared with prior designs in Section 4.5.

4.1 Introduction

Much research has been carried out on designing pipelined architectures for computation of FFT of complex (CFFT) and real-valued signals (RFFT) for high throughput applications [25] - [32]. The hardware complexity of the pipelined architectures depends on the size of FFT. In general, pipelined architectures relatively consume large area due to multiple butterfly and multiplier units. Numerous memory-based architectures have been proposed to achieve smaller area [57]-[62]. These architectures require many computation cycles to complete one whole FFT computation. Higher radix butterfly units and/or parallel processing can be utilized to increase the throughput [60], [62]. Very few architectures have been proposed for real-valued signals [63], [64]. However, these architectures are developed based on the packing algorithm, which computes a complex FFT and requires additional operations for post-processing.

We propose a novel conflict free memory access scheme for mixed-radix $2/2^2$ algorithm. This scheme can be extended to the scenario with parallel processing elements. A continuous flow FFT processor is proposed based on the new memory addressing. Further, we propose a novel memory-based FFT architecture which computes the RFFT based on the modified radix-2 algorithm in [52]. The algorithm computes only half of the output samples and removes the redundant operations from the flow graph. The modified flow graph contains only real data paths as opposed to complex data paths in a regular flow graph. Therefore, the word length in the memory units of the proposed FFT processor is W, where W is the word length chosen to represent either real/imaginary component. A new processing element is proposed to efficiently implement the operations in the modified flow graph. The processing element consists of two radix-2 butterflies and can process four samples in parallel. A new addressing scheme is proposed for conflict-free memory accesses. Further, it is shown that the proposed scheme can be extended to support parallel processing elements.

4.2 Complex-FFT Architecture

Fig. 5.12 shows the high level architecture of the proposed FFT processor with L processing elements (PE). Each memory module can store N words of length W. The number of banks with in a memory module depends on L as shown in the figure. The data streams are multiplexed between the I/O interface and the processing element. When I/O interface communicates with memory module 1, the PE reads and writes to Memory Bank 2. When Memory Bank 1 is full of new data, the PE can begin reading and working on the data stored there. The I/O must now write to Memory Bank 2. The PE thus alternates between the memory banks, and as long as new data are sent to the memory bank not being used by PE, memory conflicts are avoided.

The PE block consists of mixed-radix FFT butterfly as shown in Fig. 4.2. The PE computes four butterfly outputs based on the radix-2^2 algorithm. The PE unit has three complex multipliers and eight complex adders. The multiplexers are used to support radix-2 computation. The PE can either compute two radix-2 butterflies or one radix-4 butterfly in once clock cycle. Dual-port SRAMs are necessary to read input data and write output data simultaneously for the butterfly operation.

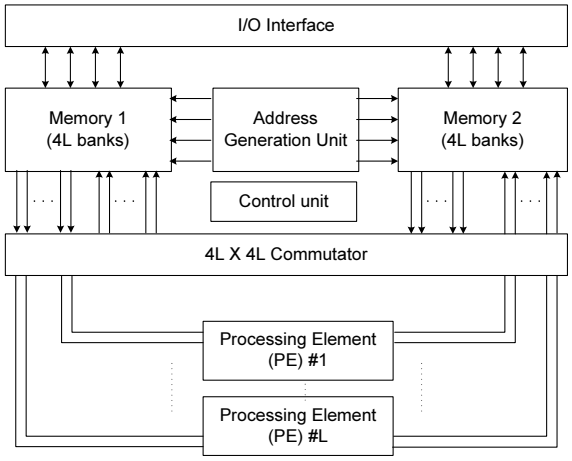


Figure 4.1: Proposed memory-based FFT architecture.

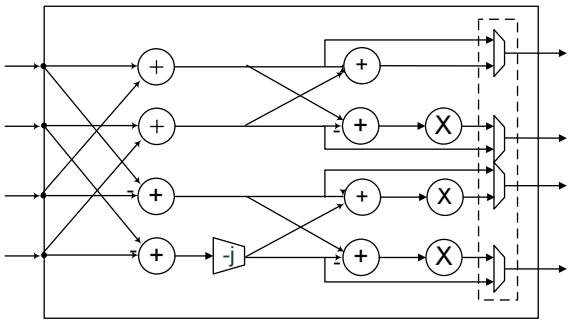


Figure 4.2: Processing element with mixed radix-$2/2^2$ butterfly

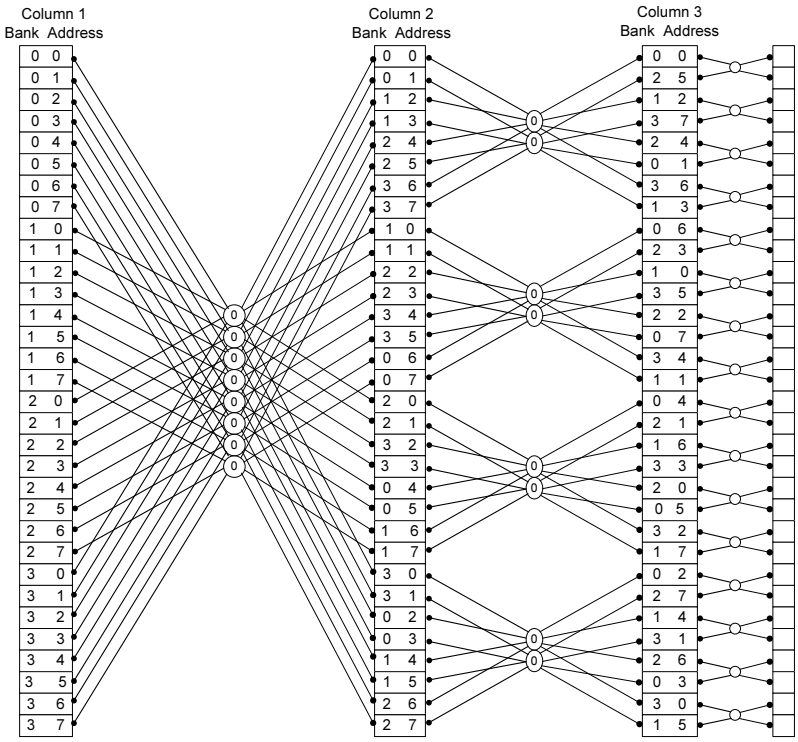


Figure 4.3: Data flow graph of mixed radix 32-point FFT.

4.2.1 Proposed Addressing Scheme

In this section a novel addressing scheme is proposed for mixed radix FFT computation. We will illustrate the solution of conflict free memory addressing scheme for one processing element. The proposed addressing strategy works similar to a pipelined FFT architecture. In a pipelined architecture, the intermediate values between the stages are stored in the registers which act as FIFOs. The number of registers depends on the stage of the computation. As an example, in the feed-forward pipelined architecture of a radix-2 16-point FFT computation, after the 1st stage the intermediate computations are stored in groups of 4 in two different memory units. Further, the size of the group goes down by 2 with every stage of computation. In a similar way, the logic can be extended to radix-4 algorithm. The size of the group changes by a factor of 4 in the case of radix-4 algorithm.

Fig. 4.3 shows the data flow graph of 32-point mixed radix FFT. The memory

is partitioned into four banks for concurrent read and write operations. Four inputs can be read concurrently and four outputs can be written concurrently to compute the radix-4 butterfly. The proposed addressing scheme can provide these concurrent reads and writes without bank conflicts. Each column at every stage in Fig. 4.3 indicate the memory bank and address assignments. The data is allocated into four banks as shown in the figure. As a result, four inputs can be read from different banks and four outputs can be written to different banks in all the butterfly operations.

Column 1 shows the order in which the host processor writes the input data. The input data is written in the natural order into the memory banks. The data is read one by one from four banks concurrently to compute each butterfly. After the computation, the output data is written to the same locations, but in a different order. This is to avoid bank conflicts during the computation of butterflies in the next stage. The order should be changed after the computation of 4 butterflies. Column 2 shows the order in which the data is written after the first stage of computation. We can observe that there are no memory conflicts, i.e., four inputs of all the butterflies are stored in different banks which can be read concurrently. Similarly, column 3 shows the corresponding address and memory bank of the data written after stage 2.

Fig. 4.4 shows the proposed addressing map for a 64-point FFT. The three columns show the locations at which the data is stored at each stage. The numbers correspond to the indices of the input/output at each stage. For example, the data with indices (0, 16, 32, 48), (1, 17, 33, 49) and soon are read to compute the butterflies in the first stage. Similarly, data with the index patterns (0,4,8,12) and (0,1,2,3) are read to compute the butterflies in second and third stage respectively.

We can observe that the first stage of FFT is composed on 16 radix-4 butterflies within a single group. The address pattern takes the form of $b_3b_2b_1b_0$. The second stage consists of 4 radix-4 butterflies within a single group and address pattern looks like $g_1g_0b_1b_0$. The third stage also consists of 4 groups, each consisting of 4 butterflies. The address pattern is similar to the second stage but with an offset. Table 4.1 shows the address patterns of all stages for computing different N-point FFTs. It can be seen that each stage has a different group counter and a butterfly counter. The values of these address are derived from a primary counter. Addresses rotates in groups of four after the first stage. The butterfly inputs are addressed in such a way as to avoid memory

Stage 1

Bank0	Bank1	Bank2	Bank3
0	16	32	48
1	17	33	49
2	18	34	50
3	19	35	51
4	20	36	52
5	21	37	53
6	22	38	54
7	23	39	55
8	24	40	56
9	25	41	57
10	26	42	58
11	27	43	59
12	28	44	60
13	29	45	61
14	30	46	62
15	31	47	63

Stage 2

Bank0	Bank1	Bank2	Bank3
0	16	32	48
1	17	33	49
2	18	34	50
3	19	35	51
52	4	20	36
53	5	21	37
54	6	22	38
55	7	23	39
40	56	8	24
41	57	9	25
42	58	10	26
43	59	11	27
28	44	60	12
29	45	61	13
30	46	62	14
31	47	63	15

Stage 3

Bank0	Bank1	Bank2	Bank3
0	16	32	48
13	29	45	61
10	26	42	58
7	23	39	55
52	4	20	36
49	1	17	33
62	14	30	46
59	11	27	43
40	56	8	24
37	53	5	21
34	50	2	18
47	63	15	31
28	44	60	12
25	41	57	9
22	38	54	6
19	35	51	3

Figure 4.4: Addressing scheme for computing 64-point FFT.

Table 4.1: Address patterns for different N-point FFT computation

Stage	$N = 32$	$N = 64$	$N = 128$	$N = 256$
1	$b_2b_1b_0$	$b_3b_2b_1b_0$	$b_4b_3b_2b_1b_0$	$b_5b_4b_3b_2b_1b_0$
2	$g_1g_0b_0$	$g_1g_0b_1b_0$	$g_1g_0b_2b_1b_0$	$g_1g_0b_3b_2b_1b_0$
3	$g_1g_0b_0$	$g_1g_0b_1b_0$	$g_1g_0f_1f_0b_0$	$g_1g_0f_1f_0b_1b_0$
4			$g_1g_0f_1f_0b_0$	$g_1g_0f_1f_0b_1b_0$

access conflicts.

4.2.2 Address generation unit

The address generation scheme employed in the architecture is described in Section II. Fig. 4.5 shows the block diagram of address generation unit based on the proposed scheme. The core units are n-bit counter, where $n = log_2N - 2$, up-to-down converter and modulo-2 adder ($\oplus$). The schematic here shows the example of address generation unit for 1024-point FFT computation. The up-to-down converter unit converts the original counter value to a corresponding down counter value at that particular instant. This conversion is required from the 2nd stage of FFT computation. We can observe from Fig. 4.4 that a down counter is required to generate the group number. The up-to-down counter simply requires a two's complement function which can be realized with few gates in the hardware.

The module-2 adders are required to add the offset between the addresses of different banks. The offset values depend on the stage of the FFT computation. As an example, the offset for the group counter is 1, while the butterfly counter is 0 during the 2^{nd} stage computation of 64-point FFT. The control unit generates the offset values depending on the computation stage.

4.3 Real-FFT Architecture

4.3.1 RFFT and Prior Work

The Fourier transform of the real valued signals exhibit conjugate symmetry giving rise to redundancies. Due to this only half the samples need to be computed. This property could be exploited to reduce both arithmetic and memory complexities. Specific

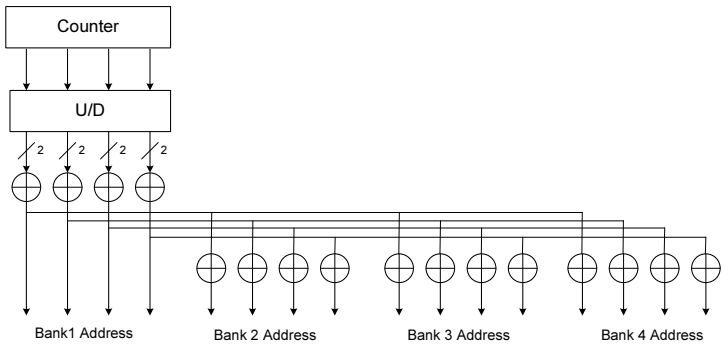


Figure 4.5: Address generation unit.

algorithms for the computation of RFFT have been proposed to reduce the number of operations. The basic idea of the algorithm is to compute only one half of the intermediate outputs, while the rest can be obtained by conjugating them. This idea has been extended to higher radices as well as split-radix [49]. In [50], an algorithm for RFFT is proposed for decimation in frequency (DIF) decomposition making use of linear phase sequences.

Different methods have been proposed to compute the FFT of real-valued signals based on a complex FFT architecture. The FFT architecture for complex inputs can be efficiently used to compute FFT for real signals using two different algorithms. One such algorithm is the doubling algorithm [45], where an existing architecture is used to calculate two real FFTs simultaneously. Another one is the packing algorithm [45], which forms a complex sequence of length $N/2$ taking the even and odd indexed samples of a real input sequence of length N, and computes the $N/2$-point FFT of the complex sequence. In these algorithms, additional operations for post-processing are necessary to obtain the final results. In-place architectures have been proposed using the packing algorithm in [63], [64]. In [63], in-place computation and memory schemes are developed to implement a variable size radix-4 RFFT processor.

Further, a low complexity algorithm is proposed in [52] to compute the RFFT. This algorithm requires less number of operations compared to the packing algorithm. The radix-2 flow graph is modified after removing the redundant operations to obtain a regular geometry. The modified data flow graph of 32-point RFFT is shown in Fig. 4.6. The flow graph computes only 17 output samples instead of all 32 samples. Further,

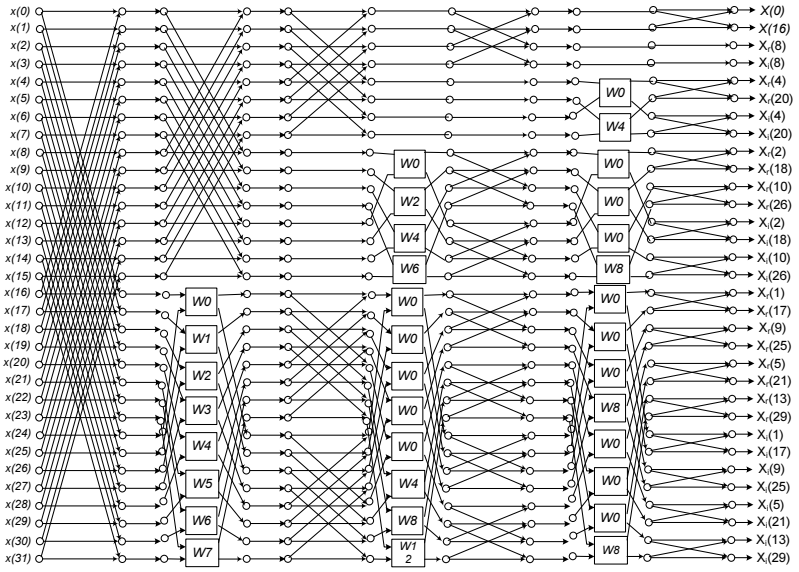


Figure 4.6: Data flow graph of 32-point FFT for real-valued signals.

entire datapath is real. We can observe that at any stage of computation, we need to store only N real samples instead of complex samples. Further, all the butterflies in the flow graph process only real samples which require two real adders instead of complex adders. We propose a novel memory-based FFT processor based on the algorithm in [52].

Fig. 5.12 shows the high level architecture of the proposed FFT processor with L processing elements (PE). Each memory module can store N words of length W. The number of banks with in a memory module depends on L as shown in the figure. The data streams are multiplexed between the I/O interface and the processing element. When I/O interface communicates with memory module 1, the PE reads and writes to Memory Bank 2. When Memory Bank 1 is full of new data, the PE can begin reading and working on the data stored there. The I/O must now write to Memory Bank 2. The PE thus alternates between the memory banks, and as long as new data are sent to the memory bank not being used by PE, memory conflicts are avoided.

A new processing element is proposed based on the modified FFT flow graph [52]. Fig. 4.7 shows the proposed processing element (PE) for the RFFT processor. The PE consists of two butterflies and one complex multiplier. The PE processes 4 samples in

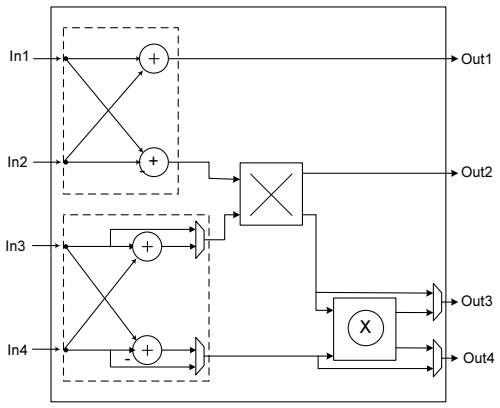


Figure 4.7: Processing element for real-valued signals.

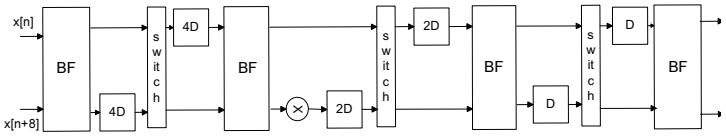


Figure 4.8: 2-parallel pipelined architecture for a 16-point FFT computation.

parallel to increase the throughput. The PE operates in three different modes depending on the computation stage. In mode 1, the multiplier unit is bypassed to compute the butterflies for the first and last stages of the FFT only. The bottom butterfly unit is bypassed for the computation of stage 2 of the FFT. The bottom butterfly will act as a pass through unit, i.e., the top butterfly unit processes the top two inputs and the multiplier processes the two bottom inputs. The two butterflies and the multiplier unit will be utilized in mode 3 to compute the butterfly and multiplication operations during the remaining computation stages. Dual-port SRAMs are necessary to read input data and write output data simultaneously for the butterfly operation.

4.3.2 Addressing Scheme

We present a novel addressing scheme for the RFFT computation similar to the one in [59]. We first illustrate the conflict-free memory addressing scheme for one processing element. Then, we extend the approach to an architecture with multiple processing elements.

The proposed addressing strategy works similar to a pipelined FFT architecture.

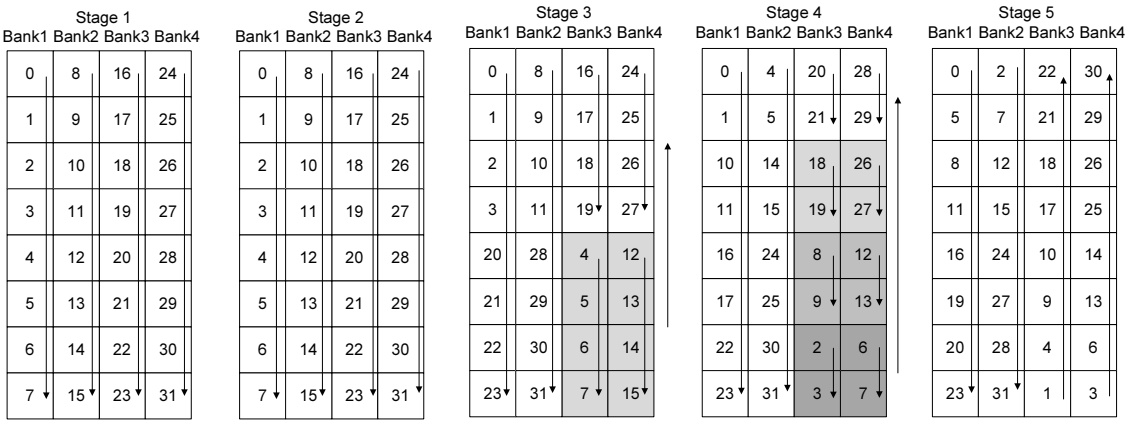


Figure 4.9: Illustration of proposed addressing scheme for one processing element. The arrows show how the data is read/write into the memory banks during the various stages of the RFFT computation.

In a pipelined architecture, the intermediate values between the stages are stored in the registers which act as FIFOs. The number of registers depends on the stage of the computation. As an example, the feed-forward pipelined architecture of a radix-2 16-point FFT computation is shown in Fig. 4.8. We can observe that after the 1st stage the intermediate computations are stored in groups of 4 in two different memory units. Further, the size of the group goes down by 2 with every stage of computation. We read/write the intermediate computations in groups in the proposed RFFT processor. The size of the group depends on the FFT computation stage.

The addressing scheme for the proposed RFFT architecture is shown in Fig. 4.9 for a 32-point RFFT. The memory is partitioned into four banks for concurrent read and write operations. Four inputs can be read concurrently and four outputs can be written concurrently to compute the two radix-2 butterflies. The proposed addressing scheme enables these concurrent reads and writes without bank conflicts. Each column corresponds to one memory bank and shows how the data is stored in the memory bank at that stage of computation. The numbers correspond to the indices of the input/output at each stage. For example, the data with indices (0, 8, 16, 24), and (1, 9, 17, 25) are read in order to compute the butterflies in the first stage. The stage 1 shows the order in which the I/O interface writes the input data. The input data is written in the natural order into the memory banks. The data is read one by one from the four

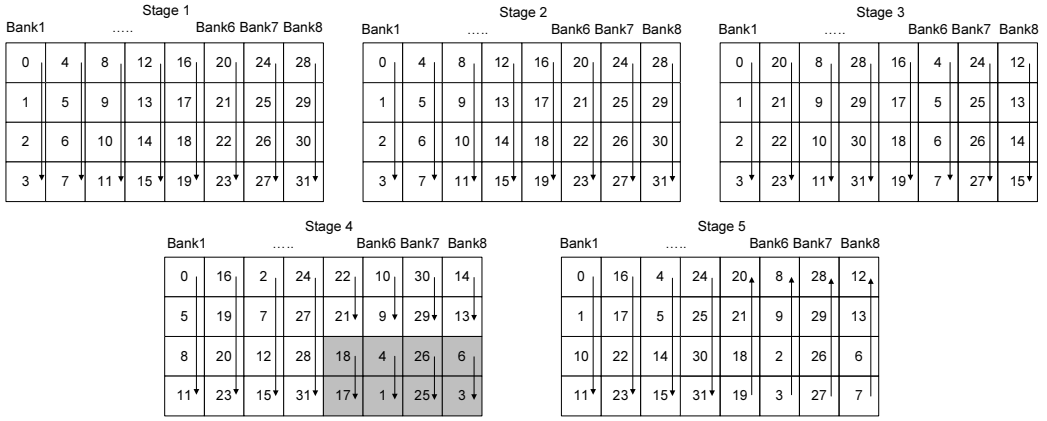


Figure 4.10: Illustration of proposed addressing scheme for two processing elements.

banks concurrently to compute each butterfly. After the computation, the output data are written to the same locations. The order in which they are written back depends on the computation stage. This is to avoid bank conflicts during the computation of butterflies in the next stage.

Fig. 4.9 also shows how the data are accessed with arrows inside the columns. We can observe that read/write patterns of memory banks 1 and 2 are the same for all stages. The data is read/write in a serial order from address location 0 to 7 at every stage and a simple counter can generate the read/write address for these banks. The address pattern can be represented as $b_2b_1b_0$. Further, the addressing patterns of memory banks 3 and 4 are also the same but the pattern varies across the stages. The data are stored in groups after stage 2, similar to a pipelined architecture. The size of the group depends on the FFT stage. The size of the group is 4 at stage 3 for a 32-point RFFT computation. The size of the group goes down by a factor of 2 with every subsequent stage as shown in the Fig. 4.9. The data is accessed from the first location of the bottommost group. Table 4.2 shows the read/write address patterns of memory banks 3 and 4 for different N-point FFTs. These patterns can be derived from the primary counter. It can be seen that each stage has a different group counter and a butterfly counter. The values of these addresses are derived from a primary counter. Recall that the outputs of the processing element are written to the same memory locations as the input data so the addresses for memory writes are the same as that for reads.

Table 4.2: Address patterns for different N-point RFFT computation

Stage	$N = 32$	$N = 64$	$N = 128$
1	$b_2b_1b_0$	$b_3b_2b_1b_0$	$b_4b_3b_2b_1b_0$
2	$b_2b_1b_0$	$b_3b_2b_1b_0$	$b_4b_3b_2b_1b_0$
3	$g_0b_1b_0$	$g_0b_2b_1b_0$	$g_0b_3b_2b_1b_0$
4	$g_1g_0b_0$	$g_1g_0b_1b_0$	$g_1g_0b_2b_1b_0$
5	$g_2g_1g_0$	$g_2g_1g_0b_0$	$g_2g_1g_0b_1b_0$
6		$g_3g_2g_1g_0$	$g_3g_2g_1g_0b_0$
7			$g_4g_3g_2g_1g_0$

Further, the proposed addressing scheme can be extended to an architecture with multiple processing elements. We demonstrate this with an example of 2 processing elements. Fig. 4.10 shows the proposed addressing scheme for two processing elements. Eight data samples need to be read/written in each clock cycle. The memory module consists of 8 memory banks to enable parallel processing of two PEs. We can observe that the read/write address patterns for the first three stages is the same. Similar to the scheme with one PE, the data will be divided into groups from the 4th stage of the FFT. As the addressing order for memory banks 0-3 and memory banks 4-7 are the same, and only two different addresses need to be generated. The address generation circuit for this case is similar to the circuit shown in Fig. 4.11 except for the offset values.

4.3.3 Address generation unit

Fig. 4.11 shows the block diagram of the address generation for the computation of 32-point RFFT. The $\oplus$ represents an XOR gate in this case. The XOR gates are required to add the offset values between the addresses of different banks. The offset values depend on the stage of the FFT computation. The memory banks 1 and 2 are accessed in a serial fashion and the counter output itself can be used as the address. For the memory banks 3 and 4, an offset needs to added depending on the computation stage. The offset values required for 1st to 5th stages are "000", "000", "100", "110" and "111", respectively. The control unit generates the offset values depending on the computation stage.

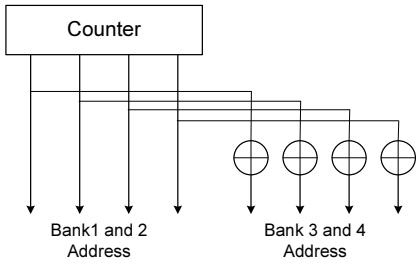


Figure 4.11: Address generation unit of the proposed RFFT processor.

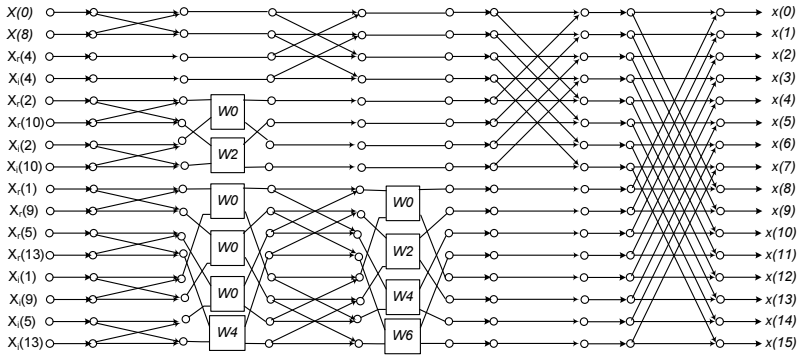


Figure 4.12: Flow graph of the 16-point inverse FFT for hermitian symmetric input.

4.4 Hermitian-symmetric IFFT processor

The IFFT of a hermitian-symmetric signal can be computed using only half the samples. The flow graph can be modified in a way similar to the RFFT flow graph. We can derive the IFFT flow graph by transposing the RFFT flow graph. Fig. 4.12 shows the flow graph of 16-point IFFT of a Hermitian-symmetric sequence. It can be seen that only 9 input samples are utilized to compute the 16-point real sequence. Further, all the data paths carry real signals. Due to this, the word length of the required memory can be W instead of 2W, where W is the word length of either real/imaginary component.

The IFFT can be computed using the proposed processing element shown in Fig. 4.7. The architecture of the IFFT processor will be same as the RFFT except the addressing strategy. Similar to the RFFT case, the processing element operates in three different modes. The units in the PE are fully utilized, i.e., all the butterfly and multiplier units are working during the first $(log_2 N - 3)$ computation stages. The multiplier unit is bypassed in the $(log_2 N - 2)$ and $log_2 N$th stages. In the $(log_2 N - 1)$th stage, the bottom

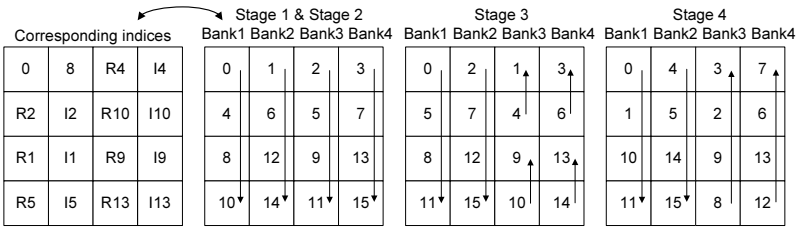


Figure 4.13: Addressing scheme for the proposed 16-point hermitian-symmetric IFFT processor.

butterfly unit acts as pass through unit to compute the only multiplication operation.

The addressing scheme for the computation of IFFT is also similar to the RFFT computation. Fig. 4.13 shows the addressing scheme for a 16-point IFFT computation. The second column shows the initial order in which the data needs to be stored. The corresponding data indices are shown in the first column. As an example, $R4$ and $I4$ correspond to the real and imaginary components of $X(4)$ sample. We can observe that the data are accessed in serial manner from memory banks 1 and 2 throughout the whole computation. The read/write patterns are the same for memory banks 3 and 4 but differ with the computation stage. We can see that the read/write strategy is the same in the first two stages. After that, the data are divided into groups similar to the addressing in the RFFT computation. Fig. 4.13 also shows the how the data are accessed with the arrows. The address patterns for the IFFT computation are similar to the RFFT as shown in the Table 4.2. The only difference comes in when adding the offset values to generate the correct address. The offset values required for 1st to 4th stages are "00", "00", "01" and "11", respectively. Therefore, by configuring the offset values, we can either compute RFFT or IFFT by using the same processor.

4.5 Comparisons

Table 7.3 compares the computation cycles of various memory-based FFT processors with their respective addressing schemes. The hardware complexity is also provided. We can observe that the proposed addressing scheme with two parallel processing elements reduces the computation cycles of the memory-based architecture compared to prior work. The reduction comes at the cost of increase in hardware complexity.

Table 4.3: Comparison of memory-based FFT architectures

	Proposed	Proposed (2 PEs)	[62]	[57]	[58]	[59]	[61]
Complex Adders	8	16	12		8	12	
Complex Multipliers	3	6	3*		3	3	
Computation Cycles	$\frac{N}{4}log_4N$	$\frac{N}{8}log_4N$	$\frac{N}{4}log_8N$		$\frac{N}{4}log_4N$	-	
256	256	128	256	512	256	271	256
512	640	320	512	1152	640	783	512
1024	1280	640	1024	2560	1280	1305	1024
2048	3072	1536	2048	5632	3072	3609	2048
4096	6144	3072	4096	12288	6144	6174	4096

In Table 4.4, we compare the hardware complexity and computation cycles of a prior memory-based RFFT processor with the proposed design. The radix and the number of memory banks required are also provided. It can be seen that the proposed design with a single PE requires the same number of computational cycles as the one in [40]. This is achieved with a PE of two real butterflies and one complex multiplier, while the prior design requires a complex radix-4 butterfly consisting of 12 complex adders and three complex multipliers. Further, the proposed design with 2 PEs outperforms the prior approach in terms of both computational cycles and hardware complexity.

Further, the proposed processor would be efficient in terms of energy consumption compared to [40]. In every computation cycle, the proposed processor reads/writes four real samples, while the prior design accesses four complex samples. The prior design computes a complex radix-4 butterfly while the proposed design computes two real radix-2 butterflies in every cycle. Based on these observations, we conclude that the proposed architecture consumes less energy compared the architecture in [40].

4.6 Conclusion

We proposed a novel continuous-flow FFT processor for real-valued signals. The proposed computation scheme is based on a modified algorithm which removes the redundant operations from the flow graph. A new processing element is proposed to reduce the hardware complexity and can process 4 samples in parallel. A conflict-free addressing scheme is developed which can be extended to support parallel processing elements.

Table 4.4: Comparison of the RFFT processors

	Proposed	Proposed (2-PEs)	[40]
Radix	radix-2	radix-2	radix-4
Complex Adders	2	4	12
Complex Multipliers	1	2	3
Memory	N/4 x W	N/8 x W	N/8 x 2W
# Memory modules	4	8	4
N	Computational Cycles		
256	512	256	512
512	1152	576	768
1024	2560	1280	2560
2048	5632	2816	3854
4096	12288	6144	12288

The proposed RFFT processor reduces the number of computational cycles with low hardware complexity. Further, the proposed processor consumes less energy compared to prior work based on packing algorithm.

Chapter 5

Power Spectral Density Computation

This chapter presents a low-complexity algorithm and architecture to compute power spectral density based on Welch method. Section 5.2 provides an introduction to Welch method to compute power spectral density. The proposed low-complexity algorithm for PSD computation is described in Section 5.3. In Section 5.4, the performance and complexity analysis of the proposed algorithm are presented. Section 5.5 presents a new architecture to compute PSD based on the proposed algorithm. Sections 5.6 and 5.7 presents an approach to reduce the computational complexity in overlapped block processing and short-time Fourier transform computation, respectively.

5.1 Introduction

Spectral analysis is widely used in signal processing for distinguishing and tracking signals of interest [65] [66] (e.g., analysis of radar and sonar signals [67]), spectrum sensing [68], [69] and for extracting information from the relevant data (e.g., biomedical signal analysis [44]). Power spectral density or spectral power represents the power of the input signal over a range of frequencies. Wiener-Khintchine theorem proves that the power spectral density of a signal is the Fourier transform of the auto-correlation of

the signal [70].

$$S(w) = \sum_{l=-\infty}^{\infty} \phi_{xx}[l] e^{-jwl}$$

However, the above equation is not useful in computing the PSD in practice. Different estimation methods have been developed in the literature to compute PSD which can be divided into two groups: parametric methods and non-parametric methods. Parametric methods assume a stationary stochastic process which generates the underlying signal. Some of the parametric approaches include autoregressive-moving average (ARMA) model identification, minimum-variance distortionless response method (MVDR) and eigen decomposition based methods [71]. Non-parametric methods do not assume any stochastic model. Two examples of non-parametric methods include periodogram based methods and multiple-window method. In general parametric approaches are computationally intensive compared to the non-parametric methods. The periodogram based methods involve the computation of the Fourier transform of the signals. The computation of the Fourier transform has been well explored in the literature. Fast Fourier transform (FFT) algorithms have been developed over the years. Due to the availability of computationally efficient FFT algorithms, the periodogram approach is often preferred to the parametric approaches.

This thesis presents novel modifications to the Welch PSD method to derive a low-complexity architecture suitable for low-power embedded systems. Biomedical signal analysis [44] is one such application where a dedicated hardware can be used in low-cost and low-power systems. Constraints like area, performance and power consumption should be taken into consideration while trying to incorporate PSD computation into biomedical monitoring systems. Such systems impose strict constraints on power consumption.

There has not been much research in developing architectures for PSD computation except for the designs on short-time Fourier transform [73] - [74]. A systolic architecture has been proposed in [67] based on the ARMA model approach. These designs are straightforward and no new optimizations were proposed. The widely used method to compute PSD is the Welch method [72] which is a modified periodogram approach. FFT is the core part of the Welch method. In general, an overlap of 50% is used when dividing the input signal into multiple segments. That is, half of the samples are the

same over two consecutive FFT operations.

We propose a PSD architecture based on this observation that leads to a reduction in the number of operations required to compute the PSD. The proposed architecture requires a $N/2$-point FFT instead of an N-point FFT block, where N is the length of the window. The main idea is to reuse the $N/2$-point FFT from the previous segment by moving the windowing operation into the frequency domain. This is only practical when the window functions are represented by raised cosine functions. The windows based on raised cosine functions, such as Hamming, Hanning etc., have only very few significant coefficients in the frequency domain. The window operation in frequency domain is a convolution operation which can be implemented by a *symmetric* 3-tap or 5-tap FIR filter. The low-complexity of the frequency domain convolution with a short filter is the key to reduction in complexity of the PSD computation. The proposed approach is extended to reduce the computational complexity in overlapped block processing of the signals like speech, when a rectangular window is used. Further, a low complexity architecture to compute a special case of the short-time Fourier transform (STFT) based on the proposed modifications is also presented.

5.2 PSD Computation

PSD estimation based on the Welch method involves dividing the signal into multiple segments, taking modified periodograms of these segments and averaging these modified periodograms. The complete algorithm is described as follows:

- The input signal $x[n]$ is divided into L overlapping segments.
- The specified window is applied to each segment.
- FFT is applied to the windowed data.
- The modified periodogram of each windowed segment is computed.
- The modified periodograms are averaged to obtain the spectral estimate $S(k)$.

To describe the Welch method in a mathematical form, let

$$x_l(n) = x(n + (l-1)M), \quad n = 0, ..., N-1$$
$$l = 1, ..., L \tag{5.1}$$

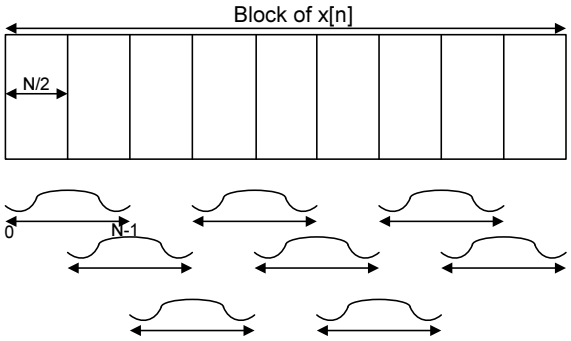


Figure 5.1: Illustration of the segmentation of a given block of $x[n]$ into 8 overlapping segments, i.e., $L = 8$.

denote the lth data segment. In (5.1), $(l-1)M$ is the starting point for lth sequence of observations. The value recommended for M in the Welch method is $M = N/2$, i.e., the data segments contain 50% overlap between successive segments [65]. The windowed periodogram corresponding to $x_l(n)$ is computed as

$$A_l(k) = \sum_{n=0}^{N-1} x_l(n)w(n)e^{-j\frac{2\pi}{N}nk}$$
$$\phi_l(k) = \frac{1}{NP}|A_l(k)|^2, l = 1, ..., L$$

where A_l is the FFT of windowed segment, ϕ_l is the periodogram, and P denotes the power of the window $(w(n))$:

$$P = \frac{1}{N}\sum_{n=0}^{N-1}|w(n)|^2.$$

The Welch estimate of PSD is the average of these periodograms, i.e.,

$$S(k) = \frac{1}{L}\sum_{l=1}^{L}\phi_l(k) \tag{5.2}$$

It can seen that the Welch method computes periodograms of overlapped segments with 50% overlap. We need to compute L N-point FFTs, assuming that the input signal is divided into L segments of length N. With an overlap of 50%, half the samples over two consecutive segments will be the same. If we can combine two consecutive $N/2$-point FFTs into one N-point FFT, the number of computations can be reduced to $(L+1)$ $N/2$-point FFTs. We propose a modified Welch algorithm based on this observation to reduce the number of computations required.

5.3 Low-complexity PSD Computation

The proposed low-complexity Welch algorithm is presented in this section. The main idea is to compute FFT of the individual non-overlapped parts (i.e., half of the original segments) and obtain the FFT of the overlapped segments by combining those of the non-overlapped segments. The window function for these non-overlapped parts will be different over two consecutive segments. Therefore, the windowing operation has to be performed in frequency domain.

The proposed modifications include two new steps in the algorithm: perform the windowing operation in the frequency domain, and merge two $N/2$-point FFTs into an N-point FFT.

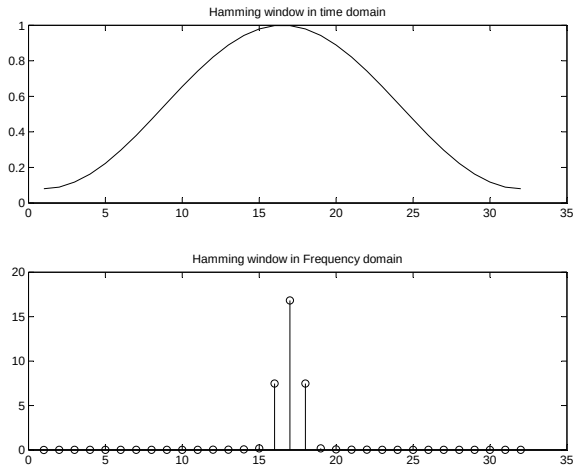


Figure 5.2: Hamming window in time and frequency domains.

5.3.1 Windowing in the frequency domain

The window function will be applied in the frequency domain instead of the time domain to take advantage of the overlapped segments. Mathematically, the multiplication operation in the time domain will be transformed into convolution in the frequency domain.

$$F\{x[n]w[n]\} <--> \frac{1}{2\pi}X(k) * W(k)$$

The convolution operation is computationally complex compared to simple multiplication operation. However, the window functions used in the PSD computation typically

Table 5.1: Non-zero coefficients in frequency domain for different window functions

Window	# Significant coefficients
Hamming	3
Hanning	3
Kaiser	3
Blackman	5

represent raised cosine functions and these can be represented by 3 or 5 non-zero coefficients in the frequency domain. Fig. 5.2 shows the 256-point Hamming window ($w[n]$) and its frequency domain counterpart ($W(k)$). We can observe that $W(k)$ has only 3 significant coefficients, while the rest of them are close to zero. The convolution operation can be implemented using a 3-tap FIR filter in the frequency domain. Further the filter is symmetric, i.e., two of the filter coefficients are equal. Therefore, the convolution operation can be implemented using 2 multiplication and 2 addition operations. Table 5.1 shows the number of non-zero coefficients in frequency domain for different window functions used in the PSD computation. It may be noted that the use of a short FIR filter in frequency domain is the key to the derivation of the low-complexity PSD computation.

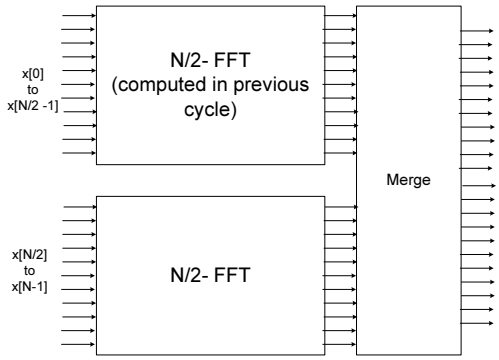


Figure 5.3: Illustration of the combining two consecutive N/2-point FFTs into an N-point FFT.

5.3.2 Merging of 2 $N/2$-point FFTs

Due to frequency domain windowing, half the samples are the same in the two consecutive windows. The computational complexity can be reduced if we can compute the

N-point FFT by combining two $N/2$-point FFTs, i.e., instead of computing an N-point FFT for each window, compute an $N/2$-point FFT and combine it with the previous $N/2$-point FFT. Using this approach, $(L+1)$ $N/2$-point FFTs need to be computed instead of L N-point FFTs for every frame of the signal. Fig. 5.3 illustrates the idea of merging two FFTs into one FFT to form an overlapped segment. The mathematical derivation to merge the FFTs is presented below.

Consider a signal $x[n], n = 0, ..., N-1$ of length N.

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk} \tag{5.3}$$

Substituting, $k = s + 2u$ and $n = l + mM$ in (5.3), where $M = N/2$, $s = \{0, 1\}$, $u = 0, ..., M-1$, $m = \{0, 1\}$ and $l = 0, 1, .., M-1$, we get,

$$X(s+2u) = \sum_{m=0}^{1} \sum_{l=0}^{M-1} x[l+mM] e^{-j\frac{2\pi}{N}ls} e^{-j\frac{4\pi}{N}lu} e^{-j\pi ms} \tag{5.4}$$

For $s = 0$,

$$X(2u) = \sum_{m=0}^{1} \sum_{l=0}^{M-1} x[l+mM] e^{-j\frac{2\pi}{M}lu} \tag{5.5}$$

and for $s = 1$,

$$X(2u+1) = \sum_{m=0}^{1} \sum_{l=0}^{M-1} x[l+mM] e^{-j\frac{2\pi}{N}l} e^{-j\frac{2\pi}{M}lu} e^{-j\pi m} \tag{5.6}$$

Define $X_1(k)$ and $X_2(k)$ as:

$$\begin{aligned} X_1(k) &= \sum_{n=0}^{M-1} x[n] e^{-j\frac{2\pi}{M}nk} \\ X_2(k) &= \sum_{n=0}^{M-1} x[n+M] e^{-j\frac{2\pi}{M}nk} \end{aligned} \tag{5.7}$$

From (5.5), (5.6) and (5.7), we get,

$$\begin{aligned} X(2u) &= X_1(u) + X_2(u) \\ X(2u+1) &= X_1(u+\frac{1}{2}) - X_2(u+\frac{1}{2}) \end{aligned} \tag{5.8}$$

Two $N/2$-point FFTs can be merged into one FFT using (5.8). We can observe that the even samples can be computed exactly, but the odd samples can only be approximated due to fractional delay shift. Fig. 5.4 shows the direct implementation of (5.8). The circuit requires a $z^{\frac{1}{2}}$ block which is non-causal. A pipeline stage as shown in the figure is required to the make it causal. We can implement a half sample delay filter to estimate the odd samples.

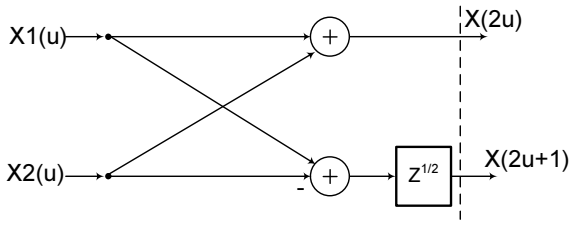


Figure 5.4: Implementation of (5.8) in hardware. A pipeline stage is required to make it a causal system.

5.3.3 Fractional delay filter

We briefly present the digital filter design techniques for the approximation of a fractional delay. There are plenty of design methods for both finite impulse response (FIR) and infinite impulse response (IIR) fractional delay (FD) filters [75], [76]. FD filters are widely utilized in digital signal processing like speech coding and synthesis, arbitrary sampling-rate conversion, timing adjustment in digital modems etc.

Consider a delay element whose purpose is to delay the incoming discrete time signal $x[n]$ by D samples, where D is a non-integer. The output signal can be expressed as

$$y[n] = x[n - D].$$

The transfer function of the system is

$$H(z) = z^{-D}.$$

The ideal impulse response for this system would be

$$h_{id}[n] = \frac{sin(\pi(n - D))}{\pi(n - D)} = sinc(n - D) \forall n.$$

It is evident that with a finite-order causal FIR or IIR filter the ideal response can only be approximated. Different approaches have been published in the literature [75]. The least-squared integral error design approach is considered in this work. The impulse response $h[n]$ of an N^{th} order least square FD FIR filter can be expressed as

$$h[n] = \{ \begin{array}{c} sinc(n-D)\ ,\ 0 \leq n \leq N \\ 0\ ,\ \text{otherwise} \end{array} \tag{5.9}$$

Fig. 5.5 shows the implementation of (5.8) based on least square FD filter design. A delay is added to make it a causal system. The figure also shows the illustration of filtering operation along with filter coefficients. The filter will estimate the output $X(2u-1)$ based on the current sample $Y(u)$ and previous samples $Y(u-1)$, $Y(u-2)$ and $Y(u-3)$.

We can observe from the ideal impulse response (h_{id}) of $z^{-1/2}$ that the system depends on both the current and future samples. We can get a better estimate by including the future samples in the computation. This is a non-causal system which is infeasible. This can be converted into a causal system by adding the required number of pipeline stages. The number of pipeline stages depends on the number of future samples required during the estimation. We propose a *bidirectional estimator* which processes both past and future samples to compute the output. Fig. 5.6 shows the implementation of the bidirectional estimator. Two extra delays were added to include two future samples in the estimation process. The figure also shows how the filter coefficients overlap during the estimation of the output. Here $X(2u-5)$ is computed using past samples $Y(u-3)$, $Y(u-4)$, $Y(u-5)$ and future samples $Y(u-2)$, $Y(u-1)$ and $Y(u)$. Fig. 5.7 shows the coefficients of bidirectional estimator of length 6 for $D = 1/2$.

The block diagram of the proposed bidirectional estimator of length 6 is shown in Fig. 5.8. The filter is given by $h[n] = \{0.1273, -0.2122, 0.6366, 0.6366, -0.2122, 0.1273\}$. We can observe that it is a symmetric filter and the number of multipliers required is half the length of the filter. Fig. 5.9 shows the result of merging two 128-point FFTs (consecutive samples) into a 256-point FFT. A 6-tap fractional delay FIR filter based on least squared error given in (5.9) is used to estimate the half sample delay. We can observe that fractional delay approximation introduces some errors. The error between the estimated and the original spectrum is less, as half the samples (even) are computed

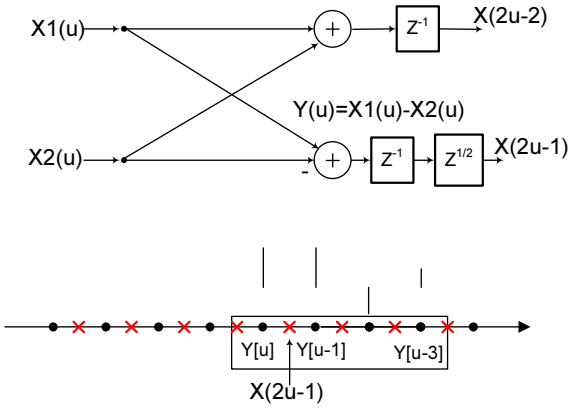


Figure 5.5: Merging two FFTs using half sample delay filter. The signal $X(2u-1)$ is computed using a 4-tap FIR filter.

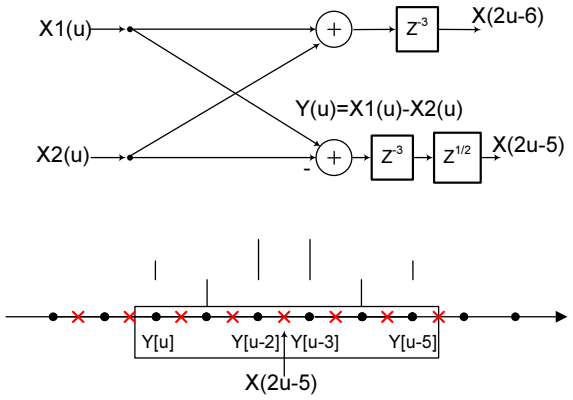


Figure 5.6: Bidirectional estimator for half sample delay.

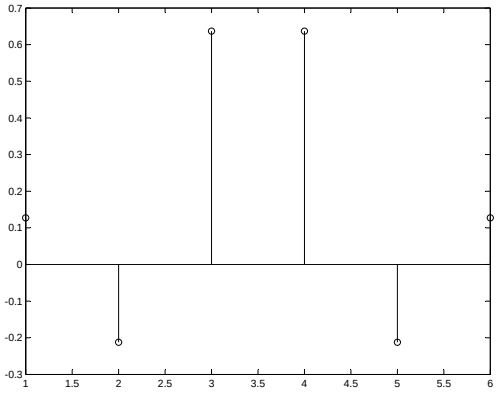


Figure 5.7: Coefficients of the filter $h[n]$ for $D = \frac{1}{2}$ of length 6.

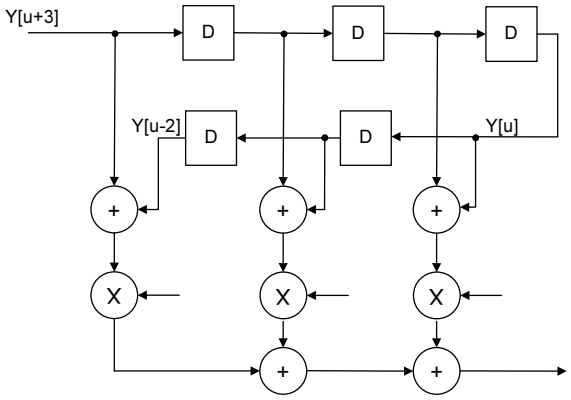


Figure 5.8: Proposed bidirectional estimator using 6-tap FIR filter.

exactly, while only the odd samples are estimated. The reader can refer to [75] for more details and derivations on fractional delay filter design.

5.3.4 Proposed PSD computation

The modified algorithm for computing power spectral density computation based on the proposed modifications is described as follows.

- The input signal vector $x[n]$ is divided into $(L+1)$ non-overlapping segments of length $N/2$.
- An $N/2$-point FFT is applied to each segment.
- N-point FFT is computed by merging two consecutive $N/2$-point FFTs.
- The specified window is applied in the frequency domain to form the overlapped segment.
- The (modified) periodogram of each windowed segment is computed.
- The modified periodograms are averaged to form the spectrum estimate $S(k)$.

The flow chart of the modified Welch algorithm is shown in Fig. 5.10 along with possible outputs at each stage. The final two steps of computing the periodograms and averaging are the same as in the original Welch method.

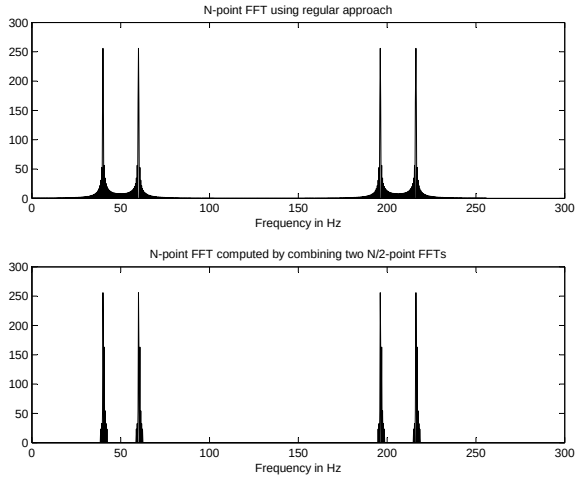


Figure 5.9: Effect of fractional delay approximation on merging two FFTs. The top one shows the 256-point FFT computed in a regular way. The bottom one is computed by merging two 128-point FFTs.

5.4 Analysis

In this section, we present the performance and complexity analysis of the proposed approach and compare it with the original Welch method.

5.4.1 Performance

The proposed approach is evaluated using EEG signals from the Freiburg database [89]. The performance of the proposed approach is gauged by percentage of change in spectral power defined as $\Delta PSD = \frac{|PSD_{orig} - PSD_{prop}|}{PSD_{orig}}$, where PSD_{orig} is defined as the total spectral power computed using Welch method and PSD_{prop} is the total spectral power computed using the proposed method. Table 5.2 summarizes the ΔPSD values simulated using least square FD filter for different lengths using both bidirectional and unidirectional filtering. The #Multiplications column represent the number of multiplication operations required to compute one output sample of the filter. As an example, unidirectional filter of length 4 and bidirectional filter of length 6 requires 3 multiplication operations. The PSD is computed by dividing the signal into 8 segments of length 1024. The original Welch method computes 8 1024-point FFTs, while the proposed method computes 9 512-point FFTs. We can observe that the bidirectional

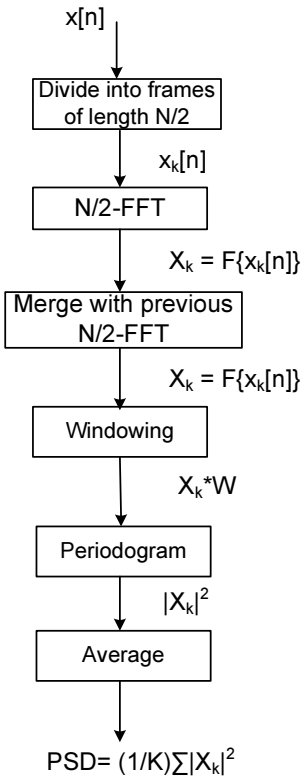


Figure 5.10: Flow chart of the proposed modified Welch algorithm.

estimator shown in Fig. 5.6 gives a better approximation compared to the uni-directional fractional delay filter shown in Fig. 5.5.

Fig. 5.11 shows the PSD of an EEG signal computed using both original and the modified method with a bidirectional estimator. A $4-$tap FD filter is used to estimate the half sample delay. We can observe from Table 5.2 that a small error is introduced in the proposed method. This is due to the fact that all the even samples can be computed exactly and the odd samples are computed using fractional delay filters.

5.4.2 Complexity

Table 5.3 shows the number of additions and multiplications required to compute PSD using the original Welch method and the proposed approach, where N is the size of the FFT and N' is the length of the fractional delay filter. The operations required in computing the periodogram and averaging are the same in the two approaches and are

Table 5.2: Performance of the proposed method using least squares FD filter of different lengths

#Multiplications	ΔPSD (%) (unidirectional)	ΔPSD (%) (bidirectional)
2	19.11	7.87
3	17.41	5.82
4	16.98	4.13
5	16.83	3.78

Table 5.3: Computational complexity of the proposed approach

	Original	Proposed
Additions	$NlogN$	$\frac{N}{2}log(\frac{N}{2}) + 3N + \frac{N}{2}(N' - 1)$
Multiplications	$\frac{N}{2}logN + N$	$\frac{N}{4}log(\frac{N}{2}) + 2N + \frac{N}{2}\frac{N'}{2}$

Table 5.4: Comparison of computational complexity for different N values

	Multiplications			Additions		
N	Original	Proposed (N'=2)	Proposed (N'=4)	Original	Proposed (N'=2)	Proposed (N'=4)
1024	6144 (100%)	4736 (77.08%)	4864 (79.17%)	10240 (100%)	8192 (80%)	9216 (90%)
2048	13312 (100%)	9856 (74.04%)	9984 (75%)	22528 (100%)	17408 (77.27%)	19456 (86.36%)
4096	28672 (100%)	20608 (71.88%)	20736 (72.32%)	49152 (100%)	36864 (75%)	40960 (83.33%)

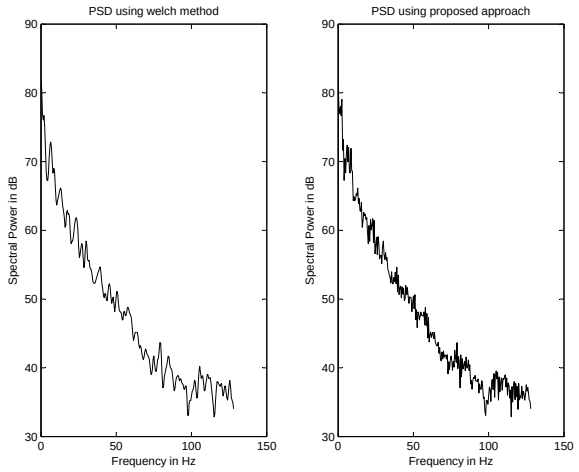


Figure 5.11: PSD computed using Welch method (left) and proposed method (right).

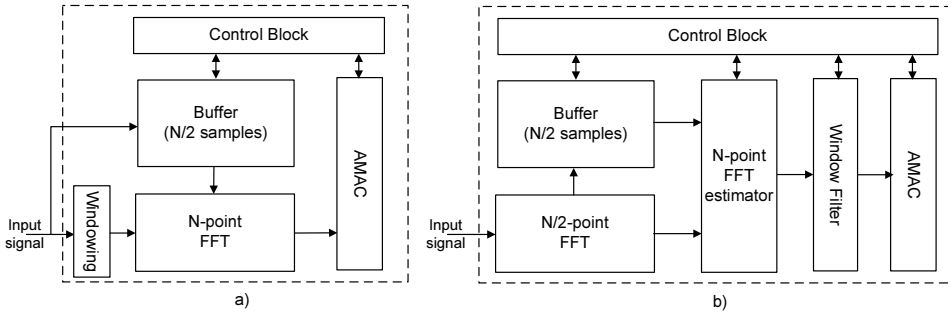


Figure 5.12: Block level architecture for PSD computation. a) Using original Welch method b) Using proposed approach.

not included in these results. We can observe that the proposed approach reduces the number of computations compared to the original Welch method.

Table 5.4 shows the comparison for different values of N to give a clear picture with the use of 2nd order and 4th order FD filters. The number of multiplications and additions are estimated using the radix-2 algorithm to compute the FFT. Only the number of multiplications will change using a different FFT algorithm. It can be seen that the proposed method requires 25% and 20% less multiplications and additions, respectively, for a 2nd order FD filter on an average. The overall power consumption will be less even though the number of addition operations are more for higher-order FD filters. This is due to the reduction in the number of multiplication operations

and the reduction in the size of the FFT that needs to be computed. The memory and multiplication operations are most power consuming compared to an addition operation. This is illustrated by comparing the energy estimates in the next section.

5.5 Proposed Architecture

Fig. 5.12a shows the block level architecture for PSD computation using the Welch method. We can observe that the core components include FFT and absolute-square multiple accumulator (AMAC) circuits. The function of AMAC circuit is to compute the periodograms and average them over L segments. AMAC is similar to the multiply multiple-accumulator (MMAC) in [78]. Fig. 5.13 illustrates an architecture for the AMAC block for a completely sequential computation. In this figure, the AMAC block stores the values of L different periodograms. The number of registers depends on the size of the FFT used in the PSD computation. The control block controls the address decoder and multiplexer to correctly accumulate the periodogram outputs over different segments. When N is large, an SRAM based design is efficient compared to shift register implementation in terms of both power and area.

Fig. 5.12b shows the architecture based on the proposed approach. It can be seen that two new blocks, an N-point FFT estimator and a window filter, are required in contrast to the original method. Fig. 5.6 and Fig. 5.14 show the internal circuitry of the FFT estimator and window filter, respectively. The FFT estimator block consists of a butterfly structure and a FD FIR filter. The adder in the butterfly structure computes the even samples while the subtractor along with the FD filter estimates the odd samples. The window filter is a 3-tap symmetric FIR filter as shown in Fig. 5.14. The filter coefficients are the three significant samples of the Hamming or Hanning window represented in the frequency domain. These filters are given by $h_{Hamming} = \{-0.23 + 0.0007i, 0.54, -0.23 - 0.0007i\}$ and $h_{Hanning} = \{-0.25 + 0.0008i, 0.5, -0.25 - 0.0008i\}$, for $N = 1024$. The multipliers in these two blocks can be implemented using CSD approach to reduce the area and power consumption.

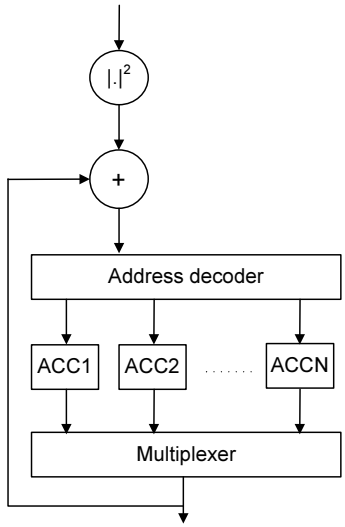


Figure 5.13: Absolute square-multiple accumulator (AMAC) circuit.

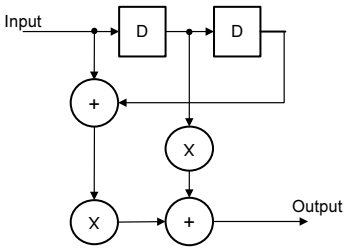


Figure 5.14: Filter circuit for windowing in frequency domain.

Energy Estimation

The energy consumption and area of the proposed architecture are estimated using 65nm technology. The logic components are implemented using STMicroelectronics 65nm libraries. The power and area of the SRAMs are estimated using CACTI tool [93]. Table 5.5 shows the area and energy estimates of the proposed architecture with wordlength of 12 bits. The energy is estimated assuming that the signal will be divided into 8 overlapping segments of length 1024 each, i.e., 1024-point FFT is computed for every segment and a 4-tap FD filter is used. We can observe that the proposed approach consumes 33% less energy compared to the original Welch method.

Most of the energy savings come from the FFT block, as the FFTs are dominant

Table 5.5: Energy estimates

	Energy		Area	
	Original	Proposed	Original	Proposed
FFT	430.56	224.55	125684	84884
Memory*	85.20	85.20	175200	175200
Logic #	12.53	43.62	2440	12654
Total	528.28 (100%)	353.36 (66.89%)	303324 (100%)	272738 (89.92%)

* memory in buffer and AMAC
# includes rest of the logic circuitry

Table 5.6: Energy estimates for real input

	Energy		Area	
	Original	Proposed	Original	Proposed
FFT	261.84	137.52	66789	46884
Memory*	42.59	50.79	87600	116800
Logic #	11.02	21.81	2440	12654
Total	315.4556 (100%)	210.12 (66.61%)	158769 (100%)	176338 (111.07%)

* memory in buffer and accumulator
# includes rest of the logic circuitry

power consuming operations. In the proposed design, we need to compute only 9 512-point FFTs instead of 8 1024-point FFTs. Various FFT architectures have been proposed in the literature based on different radix algorithms [26] - [32]. The FFT energy consumption is estimated based on the radix-2 algorithm. These estimates will change depending on the radix of the underlying algorithm and architecture. The two architectures require a buffer to store $N/2$ samples and an N word memory to store intermediate values during the averaging operation in the AMAC circuit. Further, we can observe the overhead in energy consumption due to the FFT estimator and windowing filter block. This is shown in "Logic" category in Table 5.5. The overhead is not very high as these blocks contain only adders and constant multipliers. The hardware complexity (area) of the proposed architecture is 10% less compared to the original method.

The energy estimates in Table 5.5 are presented assuming the input signal is complex. When the input signal is real, the spectrum of the signal is symmetric. Therefore, these estimates will be different as we need to compute only $N/2 + 1$ samples instead of N.

Further, the FFT architecture can also be optimized for the real signals. Different FFT architectures for real-valued signals have been proposed recently [52], [54]. We consider the algorithm in [52] to estimate the energy and area in this work. Table 5.6 shows the energy and area estimates for the case of real input. We can observe that an overall energy saving of 33% can be achieved using the proposed architecture. Most of the savings again come from the FFT block.

Further, we can observe that the complexity of the memory blocks is different in this case. In the original method, we need to save only the input samples (real in this case), while the proposed method requires storing the output samples from the FFT which are complex. The buffer size required in the proposed architecture will be doubled compared to the original method. Due to this, the overall hardware complexity of the proposed architecture is 11% more than the original architecture. We can observe from Table 5.2 that 33% energy reduction can be achieved with an error of 8%.

5.6 Overlapped Block Processing

It is common to process signals like speech, electroencephalogram (EEG), electrocardiogram (ECG) in blocks due to their slow varying nature in time. The properties of these waveforms is assumed to be constant over very short time intervals. These signals are processed block by block in overlapping intervals (50% overlap in general). The illustration of block processing and the segmentation of each block using the Welch method is shown in Fig. 5.15. The number of segments in the Welch computation should be an odd number for exactly half overlap between the two consecutive blocks. The figure shows the splitting of each block into 7 segments. The proposed modifications will further reduce the computational complexity in such overlapped block processing when rectangular windowing is used. We can observe from Fig. 5.15 that almost half the computations are common between the consecutive blocks. These common computations are circled in the figure and need to be evaluated only once over the two blocks.

Table 5.7 shows the computational complexity for processing one block. The computations for computing periodograms and averaging are not included which are common in both approaches. In this analysis, each block is divided into 7 overlapped segments

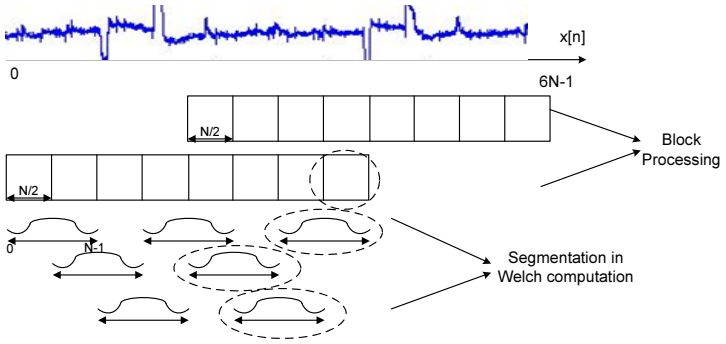


Figure 5.15: Illustration of block processing with 50% overlap. The circled computations are common over the two blocks. Each block contains 7 segments.

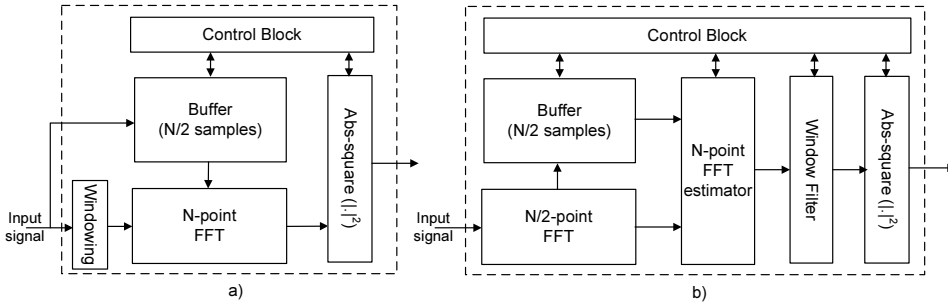


Figure 5.16: Block level architecture for STFT computation. a) Original method b) Proposed approach.

Table 5.7: Computational complexity of processing one block

	Regular	Proposed
Additions	$7NlogN$	$2Nlog(\frac{N}{2}) + 6N$
Multiplications	$\frac{7N}{2}logN + 7N$	$Nlog(\frac{N}{2}) + 4N$

and a 4-tap filter is used to compute the half sample delay in merging the FFTs. We can observe that for $N \geq 128$ almost 30% of the computations can be saved using the proposed method.

5.7 STFT computation

The short-time Fourier transform of a signal $x[n]$ is defined as

$$X(n,k) = \sum_{m=0}^{N-1} x[n+m]w[m]e^{-j\frac{2\pi}{N}km} \tag{5.10}$$

where $w[n]$ is a window sequence of length N, and is commonly a Hamming or Hanning window that represents a raised cosine function. The data to be transformed will be divided into segments which usually overlap with each other to reduce artifacts at the boundary. Not much research has been done on implementing STFT computation in hardware [73], [74]. In [73], a fully parallel STFT processor is presented based on radix-2^2 FFT algorithm. An optimization based on window symmetry is proposed to reduce the number of multipliers. A straightforward implementation of (5.10) is presented in [74]. The proposed modifications for PSD computation can also be applied for the STFT computation when n is moved $N/2$ samples, i.e., the overlap is 50%.

The short-time Fourier transform with a 50% overlap can be computed as follows:

- The input signal vector $x[n]$ is divided into non-overlapping segments of length $N/2$.
- An $N/2$-point FFT is applied to each segment.
- N-point FFT is computed by merging two consecutive $N/2$-point FFTs.
- The specified window is applied in the frequency domain to form the overlapped segment.

Fig. 5.16 shows the block level architecture of the short-time Fourier transform computation circuit. The PSD computation and STFT computation circuits are very similar except in the last stage. The short-time Fourier transform does not require averaging over multiple segments. Two blocks of FFT estimator and window filter are required in the proposed architecture as shown in Fig. 5.16b. The internal circuit of the FFT estimator and window filter are the same for STFT and PSD computation, and are shown in Fig. 5.6 and Fig. 5.14, respectively. The length of the FD filter is dictated by the application and depends on the input signal.

5.8 Conclusion

We proposed a low complexity approach to compute power spectral density using the Welch method. The computational complexity is reduced at the cost of slight performance degradation. The proposed approach computes the even samples exactly, while the odd samples need to be shifted by half sample delay. A fractional delay filter is used to estimate the odd samples which leads to a small error. The difference in the spectral power computed using proposed and original methods is approximately 8% with a 4-tap FD filter. A novel architecture is proposed based on the modified method. The proposed architecture with 4-tap FD filter consumes 33% less energy compared to the original method. Future work will be directed towards reducing the complexity of PSD computation with overlapped block processing and windowing with raised cosine windows.

Chapter 6

Support Vector Machines Computation

This chapter presents techniques for reducing the energy consumption in the architectures for support vector machines computation. Section 6.2 briefly discusses the mathematical background on support vector machines (SVM). Section 6.3 presents the complexity analysis of SVM computation with three popularly used kernels. Afterwards, Section 6.4 discusses the proposed techniques to reduce the energy consumption. Section 6.5 presents a new configurable SVM architecture based on the proposed optimizations. In Section 6.6, the proposed architecture is analyzed using simulation results.

6.1 Introduction

Support Vector Machine (SVM) is a powerful machine learning method that provides excellent performance for a wide range of regression and classification tasks based on the structural risk minimization induction principle [70]. In recent years, SVMs have been effectively used as a classification tool in a wide range of problems including pattern recognition, image analysis, and communications [8], [9]. While the SVM training can be considered as an offline task, the classification is mostly performed in real-time on newly obtained data. Face detection, speech recognition, biomedical signal analysis require online classification and have real-time constraints. However, the SVM classification is a computationally expensive task. The complexity is linearly dependent on the

classification data load, the population of the support vectors and the dimensionality of the feature set.

Even though SVMs are widely used on general purpose computer systems, the inherent complexity of the algorithm hinders its application in embedded systems. Biomedical signal classification [80], [81] is one such application where a dedicated hardware can be of great help because of the computational intensive load. Constraints, like area, performance and power consumption should be taken into consideration while trying to incorporate SVMs into biomedical monitoring systems. The power consumption should be at very low levels (e.g., 1-10mW for wearable devices and 10-100μW for implantable devices) for biomedical applications [82].

SVM architectures employing different kernels have been proposed in the recent literature either for FPGA or ASIC. Highly parallel designs have been proposed for FPGA platforms to speed up the computation [83], [84]. ASIC based designs have been limited to linear and polynomial kernels [8], [82], [85], [86]. SVM with linear kernel function requires only a dot-product computation which has straightforward implementation [8]. A programmable SVM platform with polynomial kernel has been presented in [82]. The design exploits parallelism and voltage scaling to minimize the energy consumption. Not much research has been published on optimizing the implementation of radial basis function (RBF) kernel either for area or energy minimization.

SVM computations are inherently error resilient as the decision function depends on the sign but not on the magnitude of the final value. We propose to reduce the precision of the computations for energy minimization which has not been exploited earlier in the case of SVM design. The advantage of the proposed methodology will also lead to low hardware complexity while methods based on VOS need extra hardware for compensating the errors.

In this thesis, we propose a general-purpose architecture for SVM computation that takes advantage of inherent error resiliency of the SVM algorithms. We consider the common SVM kernels used in biomedical signal analysis and multimedia recognition applications and propose design techniques and optimizations to minimize power consumption. We present the computational complexity analysis of SVM computation of commonly used kernels and optimize the kernel implementations using two techniques: fixed-width multiplier and non-uniform quantization. Fixed-width multiplier can reduce

both area and power consumption with little effect on the performance. A novel *non-uniform quantization* technique is also presented to optimize the implementation of the exponent function. Further, we propose a programmable architecture which can evaluate three different SVM kernels (linear, 2^{nd}order polynomial, and RBF). The hardware design is reconfigurable to vary the precision according to the needs.

6.2 SVM Theory

SVM is a binary classification algorithm where each sample in the data set is viewed as a d-dimensional vector. The SVM takes a data set that contains samples from two classes (labeled 1 and -1), and constructs separating hyperplanes between them. The separating hyperplane that best separates the two classes is called the maximum-margin hyperplane and forms the decision boundary for classification. The margin (shown in Fig. 6.1) is defined as the distance between two parallel hyperplanes that lie at the boundary of each class. The data samples, that lie at the boundary of each class and determine how the maximum-margin hyperplane is formed, are called support vectors (SVs). The SVs are obtained during the training phase and are then used for classifying new (unknown) data. When two data classes are not linearly separable, a kernel function is used to project data to a higher dimensional space (feature space), where linear classification is possible. This is known as the kernel trick and allows an SVM to solve nonlinear problems [71].

Given a set of training data $\{\mathbf{x}_i, y_i\}$, where $\mathbf{x}_i$ is an input pattern and $y_i \in \{-1, 1\}$ is the corresponding label. The training an SVM for classification (finding an optimal hyperplane) is equivalent to solving the following quadratic optimization problem.

$$\text{minimize } \frac{1}{2}||\mathbf{w}||^2, \tag{6.1}$$
$$\text{subject to the constraints } y_i(\mathbf{w}.\mathbf{x}_i + b) \geq 1, \forall i$$

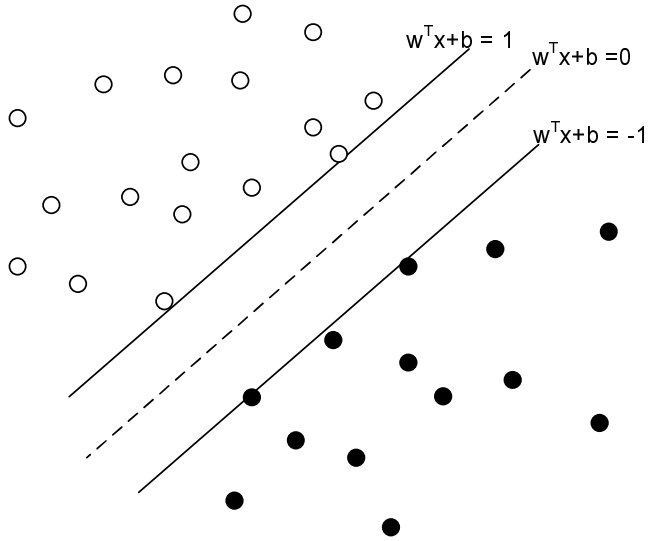


Figure 6.1: SVM separating hyperplanes

This dual of the quadratic optimization problem can be formulated as follows:

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{6.2}$$

$$\text{subject to the constraints } \sum_{i=1}^{n} y_i \alpha_i = 0,$$

$$0 \leq \alpha_i \leq C/n, \forall i$$

The final decision function is

$$f(\mathbf{x}) = sign(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b) \tag{6.3}$$

where $\mathbf{x}$ is the new feature vector, $\mathbf{x}_i$ are the support vectors, α_i are the Lagrangian coefficients and b is the threshold. The parameters x_i, α_i and b are computed during the training process. K is the kernel function. It can seen that the computational complexity depends on the kernel computation and the number of support vectors. We next describe how the computational complexity varies for three different kernels that are commonly used.

6.3 SVM Complexity Analysis

Linear Kernel: The linear kernel is given by $K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}_i^T \mathbf{x}$. The decision function can be simplified as follows:

$$f(x) = sign(\mathbf{w}^T \mathbf{x} + b), \text{ where } \quad \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i. \tag{6.4}$$

Polynomial Kernel: The polynomial kernel is given by,

$$K(\mathbf{x}, \mathbf{x}_i) = [\mathbf{x}_i^T \mathbf{x} + 1]^p,$$

where p is the degree of the polynomial. For the case of a second degree polynomial kernel, $p = 2$, $K(\mathbf{x}, \mathbf{x}_i)$ can be rewritten as

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}_i) &= [\mathbf{z}_i^T \mathbf{z}]^2, \text{ where } \quad \mathbf{z} = [\mathbf{x} \quad 1]^T \\ &= \mathbf{z}^T \mathbf{z}_i \mathbf{z}_i^T \mathbf{z}. \end{aligned} \tag{6.5}$$

The decision function can simplified as follows:

$$f(\mathbf{x}) = sign(\mathbf{z}^T \mathbf{W} \mathbf{z} + b), \text{ where } \quad \mathbf{W} = \sum_i \alpha_i y_i \mathbf{z}_i \mathbf{z}_i^T. \tag{6.6}$$

RBF Kernel: The non-linear RBF kernel is given by,

$$K(\mathbf{x}, \mathbf{x}_i) = exp\left(\frac{-||\mathbf{x} - \mathbf{x}_i||^2}{\sigma^2}\right).$$

The decision function can be written as,

$$f(\mathbf{x}) = sign\left(\sum_{i=1}^{N_{sv}} \alpha_i y_i exp(\frac{-||\mathbf{x} - \mathbf{x}_i||^2}{\sigma^2}) + b\right). \tag{6.7}$$

The decision function cannot be further simplified because of the exponent function in the kernel.

Table 7.1 show the computational complexity and memory requirements for evaluating the decision function with three kernel functions. We can observe that the complexity depends only on the feature set dimensionality for both linear and polynomial ($p = 2$) kernels. However, the complexity of the classifier with RBF kernel depends on both feature dimensionality (d) and the number of support vectors (N_{sv}). For large

Table 6.1: Computational complexity of SVM Classifier

Kernel	# ADD	# MUL	# WORDS
Linear	d	d	d
Polynomial (p=2)	d^2	$d(d+1)$	d^2
RBF	$2N_{sv}d$	$N_{sv}d$	$N_{sv}(d+1)$

*RBF kernel requires additional N_{sv} exponent operations

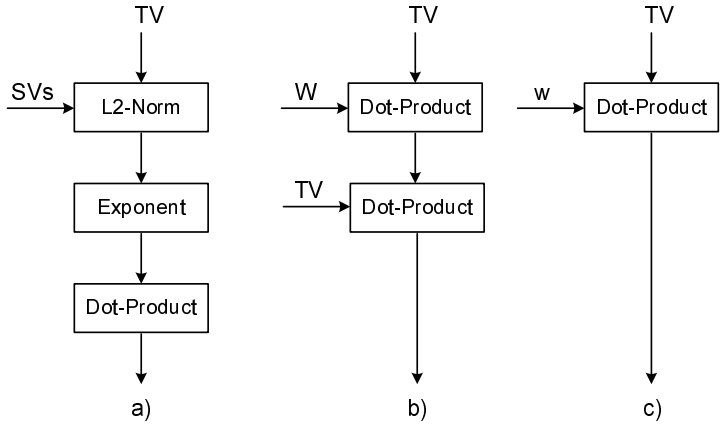


Figure 6.2: Overview of the SVM computation with (a) RBF, (b)2^{nd} order polynomial and (c) linear kernels

datasets, training will lead to large number of support vectors which in turn increases the computational complexity. The complexity of the classifier can be minimized by decreasing N_{sv} and/or d, i.e., number of support vectors and number of features per vector respectively. The reader can refer to reduced SVM (RSVM) algorithm [87] and feature selection algorithms [88] to reduce the N_{sv} and d, respectively, at the algorithm level.

6.4 Proposed Circuit Optimizations

Fig. 6.2 shows an overview of the SVM computation using different kernels. It can be observed that the core components of the SVM computation are dot-product, L2-norm and exponent (needed only for evaluating RBF kernel) operations (see Fig. 6.3). The power consumption of these individual circuits will dictate the total power consumption

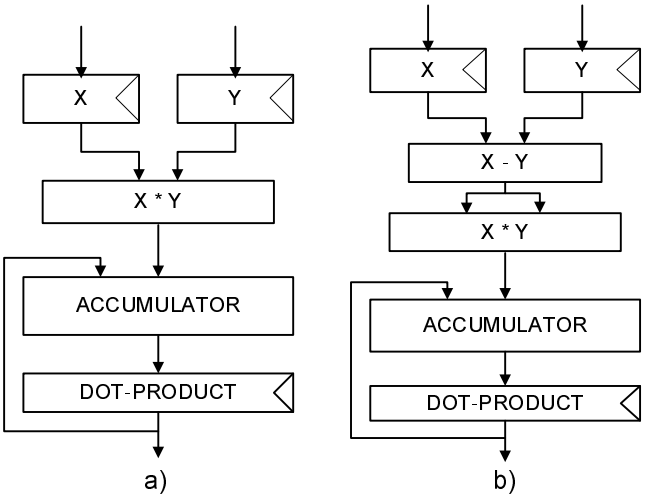


Figure 6.3: Architecture of (a) dot-product and (b) L2-norm

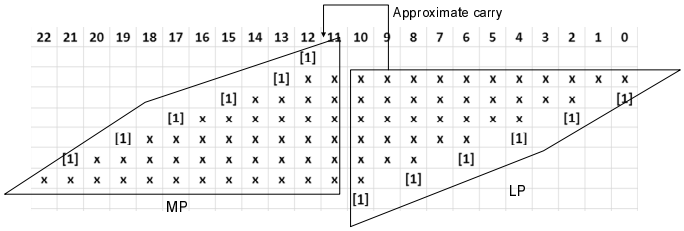


Figure 6.4: Structure of the 12-bit fixed-width multiplier

of the SVM architecture. In this section, we describe the proposed optimizations to scale the precision in these circuits which can effectively minimize the energy required. The performance analysis of the SVM computation is based on seizure prediction algorithm in [80] using the patient data from the Freiburg database [89].

6.4.1 Fixed-width MAC

The multiply-accumulator is the core component of dot-product and L2-norm functions. It is known that the multiplier hardware dominates area and power consumption compared to an adder. The area and power consumption of a multiplier increase quadratically with the wordlength. Fig. 6.4 shows the design of a fixed-width multiplier which is an integral part of both dot-product and L2-norm functions. Fixed-width multiplier involves dividing the partial products of the multiplier into two groups (most

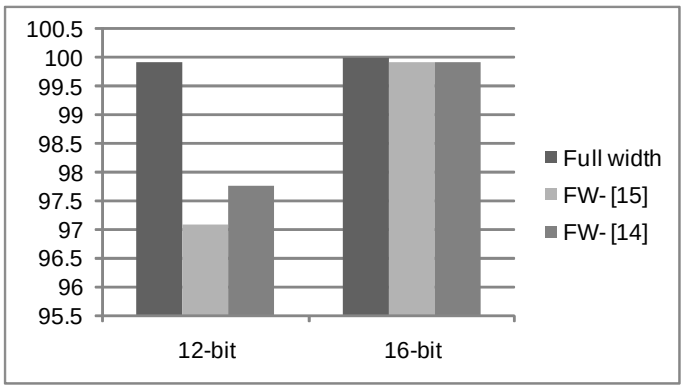


Figure 6.5: Effects of fixed-width multiplier on classifier accuracy using 12-bit and 16-bit wordlengths

significant part (MP) and least significant part (LP)) as shown in Fig. 6.4. The carry from the partial products in LP group can be approximated instead of computing the exact carry. The simplest method is to discard all the partial products, i.e., an approximate carry would be zero. However, this method may lead to significant errors. To overcome this problem, different approximation schemes [90], [91] have been proposed in the literature to generate the approximate carry from LP group with little hardware overhead. The partial products in MP group are added using Wallace or Dadda tree along with the compensation to get the final output. The fixed-width multiplier design serves two purposes: a)it reduces the area and power consumption, and b) it reduces the critical path of the MAC, allowing voltage scaling without aggressive scaling of clock frequency.

The main aspect that needs to be addressed in relation to fixed-width multiplier implementation is performance degradation of the overall SVM architecture. We analyzed the performance of the SVM computation using two existing compensation techniques. The performance of these techniques on seizure prediction algorithm is shown in Fig. 6.5 for 12-bit and 16-bit multipliers in terms of classification accuracy. We can observe that the the performance of two compensation techniques using 16-bit wordlength is same, while [90] performs better using 12-bit wordlength. The savings in area and power consumption of the 12-bit fixed-width MAC are shown in Fig. 6.6. The 12-bit fixed-width multiplier reduces the area and power consumption by 35% and 37%,

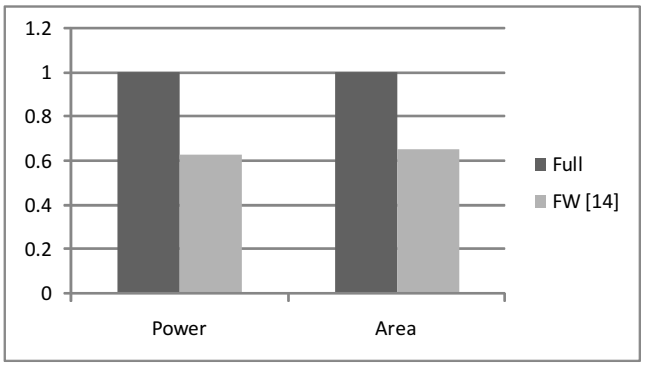


Figure 6.6: Area and power comparison of the 12-bit regular and fixed-width multiplier [90]

respectively, compared to a full multiplier.

6.4.2 Exponent Function

The LUT based implementation of the exponent function requires 2^q words of memory for a q-bit input. The memory requirement grows exponentially with the word length of the input. Therefore, we propose a novel scheme which can reduce the memory requirement by half. From Fig. 6.7, we can observe that e^{-x} function is non-linear and its slope decreases with the increase in x. The function saturates to zero, at $x = 8$ and the output of the function lies between 0 and 1 ($0 \leq e^{-x} \leq 1, \quad x > 0$). Therefore, only three bits are required for the integer part and the remaining bits can be used for fractional part. It is intuitive that the larger the slope, the smaller the quantization step should be used for a good performance. Thus, under the same total quantization bits q, a non-uniform quantization scheme will outperform the uniform one. Therefore, we propose a novel quantization scheme to implement a look-up table. The proposed scheme applies different quantization step sizes according to the magnitude of the input to the exponent function.

Consider a $(q : f)$ uniform quantization scheme in which totally q bits are used, of which f bits are used for its fractional part.The range and the step size that are capable of being expressed by $(q : f)$ quantization scheme are from $-2^{(q-f-1)}$ to $2^{(q-f-1)}$ and

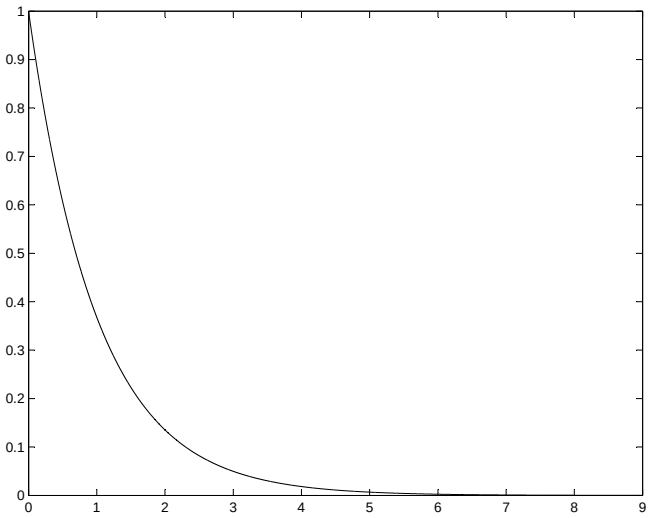


Figure 6.7: Curve of function $f(x) = e^{-x} (x > 0)$

2^{-f}, respectively for signed numbers. The proposed quantization scheme is as follows:

$$\begin{aligned} (q : f), \text{for } 0 \leq |x| < 2 \\ (q-1 : f-1), \text{for } 2 \leq |x| < 4 \\ (q-2 : f-2), \text{for } 4 \leq |x| < 8 \end{aligned} \tag{6.8}$$

Table 6.2 shows a conversion table between unsigned 4-bit uniformly quantized variables and 3-bit non-uniformly quantized variables. In this example, interval $[0, 2)$ is considered instead of $[0, 8)$ for demonstration.

It can be observed that the total number of different words quantized is reduced from 2^q to 2^{q-1}, which can be implemented with the $(q-1)$-bit word length by inserting additional conversion circuits. We need to design the conversion circuit to access the correct word in the LUT. From Table 6.2, non-uniform quantization data can be generated from uniform quantized data as described and can be implemented with a small size of combinational logic [92]. One possible combinational logic equation can be formulated as

$$\begin{aligned} y_2 &= x_3 + x_2 \\ y_1 &= x_3 + \overline{x}_2 x_1 \\ y_0 &= x_3 x_2 + x_2 x_1 + \overline{x}_3 \overline{x}_2 x_0 \end{aligned} \tag{6.9}$$

Table 6.2: Conversion between uniform quantization and proposed quantization scheme

Decimal	(4:3) uni-form quan.	non-uniform quant.	3-bit format
0	0000	0000	000
0.125	0001	0001	001
0.25	0010	0010	010
0.375	0011	0011	011
0.5	0100	0100	100
0.675	0101		
0.75	0110	0110	101
0.875	0111		
1	1000	1000	110
1.125	1001		
1.25	1010		
1.375	1011		
1.5	1100	1100	111
1.625	1101		
1.75	1110		
1.875	1111		

where $x_3x_2x_1x_0$ represents a 4-bit uniformly quantized number and $y_2y_1y_0$ represents the equivalent 3-bit data for accessing the LUT. The conversion circuit is shown in Fig. 6.8. This hardware overhead is small. Table 6.3 shows the area and power consumption of the conventional (requires only memory) and proposed look-up tables. The power and area of memory (SRAM) are estimated using CACTI tool [93]. We can observe that for a 10-bit quantization, the proposed design can save up to 35% and 37% of power consumption and area, respectively. Further, some errors will be introduced as only half of the required data are stored in the memory. It is revealed in our analysis that these errors will not affect the performance of the SVM classifier.

6.5 Proposed Architecture

In this section, we present the proposed programmable SVM architecture which has the flexibility in selecting the kernel function and precision of the computation. The

Table 6.3: Area and power comparison of conventional and proposed LUT schemes

	Power (μW)			Area (μm^2)		
Quantization	Memory	Overhead	Total	Memory	Overhead	Total
10-bit	2.77	0	2.77	49315	0	49315
10-bit Proposed	1.71	0.058	1.76	29226	54	30951
12-bit	6.07	0	6.07	162256	0	162256
12-bit Proposed	4.84	0.0722	4.91	98173	65	98238
16-bit	27.78	0	27.78	2.76e06	0	2.76e06
16-bit Proposed	18.94	0.1	19.04	1.59e06	88	1.59e06

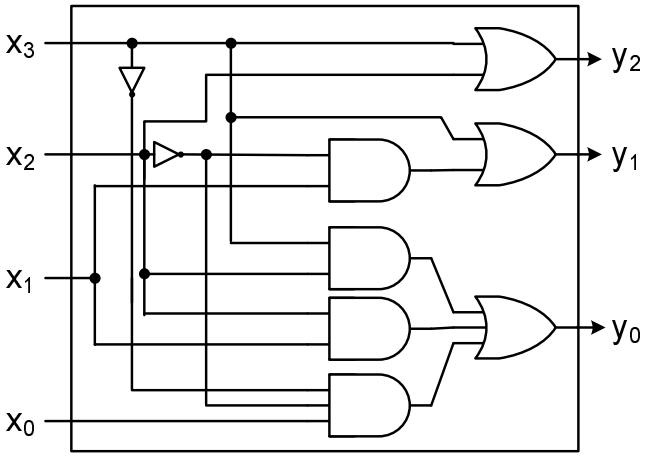


Figure 6.8: Conversion circuit for implementing the proposed quantization scheme

two design techniques, fixed width multiplier and proposed non-uniform look-up table (for exponent function) are employed to reduce the energy consumption. The proposed architecture for the SVM classifier is shown in Fig. 6.9. The proposed design mainly consists of an adder engine, a multiply-add engine and address conversion unit for exponent look-up table. Hardware parallelism is employed through an array of adder and MAC units, each of which is associated with a weight vector buffer (represents weights for linear and polynomial kernels and support vectors for RBF kernel). The weight vectors obtained during offline training and the test vector are loaded into buffers.

6.5.1 Variable precision MAC

The precision requirements of the classifier computation are assumed to be variable to support a wide range of support vectors across different applications. Different approaches have been reported for variable-precision multipliers [82]. We designed a new variable precision MAC with fixed-width multiplier. The architecture is shown in Fig. 6.10. The partial product generator only computes the necessary partial products using Booth encoding. The common partial products required for 12/14/16 precison bits are added using half and full adders in Dadda tree. The compensation scheme is based on the design in [90]. The vector merging operations after the Dadda tree require only 12/14/16-bit adders respectively, due to the fixed-width nature of the multiplier. The output of the selection multiplexer is accumulated into the output register using an 18-bit adder.

6.5.2 Programmable Kernel

Non-linear kernels have been shown to achieve good performance in biomedical signal classification and multimedia applications. The proposed design incorporates linear, polynomial (2nd order) and RBF kernels, which are widely used SVM kernels. To compute linear or polynomial kernel, the weight vectors and test vectors are fed to the MAC engine in order to perform the dot-product operations in equations (2) or (4). To compute the RBF kernel, the buffer outputs are routed to the adder engine to perform L2-norm operations. The final dot-product is computed once multiplication over all the dimensions is complete and is stored in a temporary register. This process is repeated for all the row vectors of the W matrix and support vectors in the case of polynomial and RBF kernel computations, respectively. The results are scaled and accumulated in a register. The sign of the final value determines the classification result.

6.6 Simulation Results

In this section, we present results of post-layout simulations that demonstrate the benefits of the proposed design techniques. We implemented the proposed design in STMicroelectronics 65nm CMOS technology. Verilog RTL descriptions were synthesized using

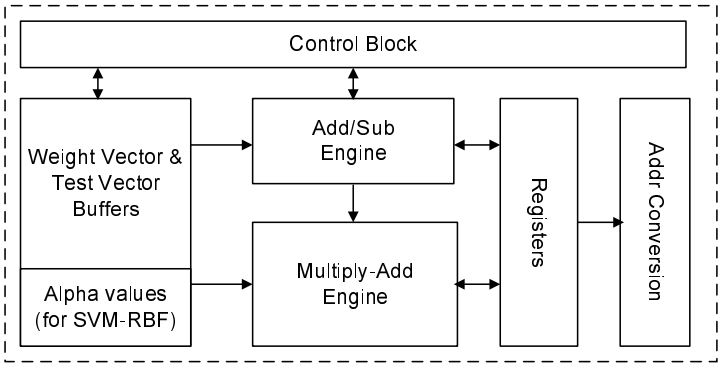


Figure 6.9: Proposed programmable SVM architecture

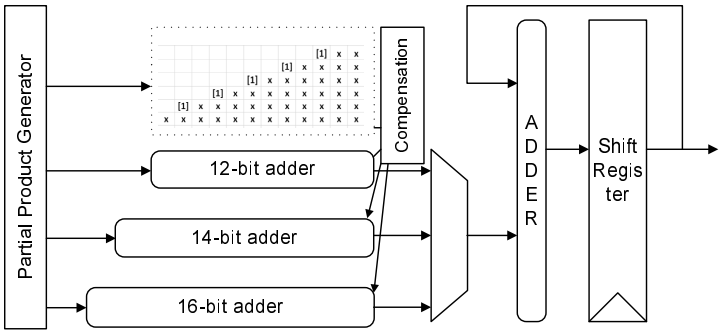


Figure 6.10: Proposed variable-precision MAC unit with fixed-width multiplier and compensation

Synopsys Design Compiler to obtain a gate-level netlist. Cadence Soc Encounter is used to place and route the gate-level design. The MAC engine consists of 4 multiply-add units. The memory (SRAM) required to store the look-up table is not implemented in this design. The energy consumption of the SRAM are estimated using CACTI tool [93]. Table 7.3 shows the energy consumption for linear and polynomial kernels from post-layout simulations using 16-bit wordlength. Voltage scaling and parallelism (adding more MAC units) can be used to save more energy. It can be seen that the proposed schemes reduce the energy consumption by 30%.

Table 6.5 shows the energy consumption using RBF kernel with different levels of optimizations. The non-uniform LUT and fixed-width MAC will save the energy by 13% and 20%, respectively, on average. The two optimizations when combined will reduce the energy consumption by 31%. Further, we can observe that the classification energy scales roughly linearly with N_{sv} and d. We can extrapolate these estimates for larger values of N_{sv}. The total energy for seizure prediction per test vector using spectral features ($d = 36$ and $N_{sv} = 5000$) is 109.2nJ at V_{dd} of 0.5 V. The bottom of the table shows the energy for various classifier kernels. It can be seen that the energy consumption of linear kernel is very low compared to polynomial and RBF kernels.

Table 6.4: Energy consumption of the proposed architecture per test vector

Vdd (V)	N_{sv}	d	Energy(pJ) w opt	Energy(pJ) w/o opt	Frequency
Linear					
1.0		32	49.1	70.92	10 MHz
		64	81.8	118.2	
		128	147.3	212.76	
2^{nd} order Polynomial					
1.0		32	1570.6	2269.44	10 MHz
		64	5235.2	7564.8	
		128	18847.14	27223.28	
Vdd (V)	N_{sv}	d	Energy (pJ)	Kernel	
1.0	100	64	81.8	118.2	Linear
			5235.2	7564.8	Poly
			10633.61	15523.67	RBF

Table 6.5: Energy consumption of the RBF kernel per test vector

N_{sv}	d	Energy(pJ)[1]	Energy(pJ)[2]	Energy(pJ)[3]	Energy(pJ)[4]
10	8	675.7	576.03	565.68	466.01
25	16	2002.37	1745.12	1635.61	1378.36
50	32	5257.1	4710.36	4156.83	3610.08

[1] with no optimizations [2] with only non-uniform LUT
[3] with only fixed-width MAC [4] with both optimizations

6.7 Conclusion

We proposed a low-energy architecture based on *approximate computing* exploiting the inherent error resilience in the SVM computation. The proposed design is programmable to evaluate three different kernels. Two design techniques, fixed-width multiply-add and non-uniform look-up table for exponent function are proposed to minimize energy consumption. A non-uniform quantization scheme is proposed to reduce the size of the look-up table to implement the exponent function. The memory required is reduced by 50% compared to the uniform quantization method while achieving the same classifier performance. This design is very suitable for applications in implantable and embedded systems because of its low power consumption and area.

Chapter 7

Low-power Seizure Prediction Algorithm

In this chapter we present a low-power seizure prediction algorithm using spectral power features and Adaboost algorithm. Section 7.1 provides an introduction of epilepsy and seizure prediction. In Section 7.2 we present the prior work on seizure prediction algorithm development. Section 7.3 presents a brief review of Adaboost algorithm. This is followed by Section 7.4 which presents the proposed seizure prediction algorithm. The performance and complexity analysis of the proposed algorithm are discussed in Section 7.6.

7.1 Introduction

Epilepsy is the one of the most common serious neurological disorders in the world. Approximately, 1% of the world's population experience sporadic seizures. The quality of lives of the epileptic patients will be significantly improved with an automated seizure prediction device. Recently, there has been great progress in seizure suppression methods. Some of these approaches include deep brain stimulation therapy, etc. A closed-loop therapy system can be developed, where a seizure prediction device monitors and triggers the seizure treatment.

Recently, much of the research has been carried out on predicting and detecting seizures based on real-time analysis of electroencephalogram (EEG) data from multiple

channels. Research on seizure prediction is focussed on finding features that discriminate between pre-ictal (period of time before the onset of seizure) and inter-ictal (period of time between the seizures) periods. These features include power spectral density, auto regressive coefficients, wavelets and cross-correlation measures.

Even though a lot of research has been done, these results cannot be used to realize an implantable device which can predict seizures in real-time due to their inability to achieve 1) High sensitivity and low false positive rate, and 2) Low power consumption and hardware complexity. The power consumption of a reliable seizure prediction device should be in the range of 50 μW [124]. High sensitivity and low false positive rates can be achieved using signal processing and machine learning techniques [125]. These cannot be adopted for a possible real-time implementation of seizure prediction device due to their high computational complexity. The number of features and the type of classifier used to make a prediction can have a dramatic effect on power consumption.

In this thesis, we propose a seizure prediction algorithm with low computational complexity, which achieves high sensitivity and low false positive rate at the same time. The proposed seizure prediction system will be suitable for real-time implementation on an implantable device.

7.2 Epilepsy and Seizure Prediction

Epilepsy is one of the most common neurological diseases. Approximately 5% of the population experiences a seizure within their lifetime, and 1% suffers from multiple seizures, classifying them as epileptic [95] - [97]. This disease affects nearly 3 million Americans with an estimated annual cost of $15.5 billion in direct and indirect costs per year [97]. A difficult aspect of epilepsy is the unpredictable nature of seizures. Many epileptics live in constant worry that a seizure could strike at an inopportune time resulting in humiliation, social stigma, and/or injury. Therefore, an implantable device that could predict a seizure by even a few seconds could dramatically change the lives of these patients by alerting them to the impending seizure or triggering a device to abate or suppress the seizure.

Deep brain stimulation therapy has been demonstrated to abate seizures in clinical trials, where electrical stimulation is applied to deep brain structures in open-loop

manner [94]. More experimental approaches have been used to suppress seizures by-optically uncaging inhibitory neurotransmitters [98] or by focal cooling of the cortex [99]. The efficacy of these methods may be improved by a closed-loop therapy, where a seizure prediction device monitors and triggers the seizure abatement. The design and implementation of responsive, closed-loop devices to treat seizures is an exciting new development in epilepsy therapy. Analogous to the feedback control in automatic implantable cardiac defibrillators, closed-loop devices actively record electroencephalogram (EEG) signals, process these signals in real time to detect evidence of imminent seizure onset, and then trigger an intervention. The central need of the closed-loop systems [100] is the ability to detect specific physiological states of interest from the signals such as EEG and electrocardiogram (ECG). It is challenging to find the signal correlations to relevant physiological states of interest and further these correlations often varies from patient to patient [101]. Machine learning based modeling is emerging as a powerful approach to overcome these challenges. This has been prompted by the recent large-scale availability of the data in healthcare domain and the development of learning techniques to efficiently model the data.

A closed-loop intervention system is expected to have higher efficacy and lower possibility for side effects than the open-loop approach [95], [102]. The close-loop intervention system is a future seizure control system, which can alarm a seizure occurrence in advance, generate warnings for the patient, and further trigger rescue therapy and abort the development of the seizure, such as by injecting fast-acting medication or turning on a neuro-stimulator [95], [96], [102]. The closed-loop systems are always superior to any open-loop systems in terms of efficacy and safety, because they can help the rescue therapy by turning on effectively at certain necessary times and thus will always reduce the amount of unnecessary medication or side-effects [95].

7.3 Prior Work

Seizure prediction based on EEG or intracranial EEG (iEEG) is complicated by two factors. The first is that preictal and interictal EEG/iEEG patterns across patients vary substantially; there may be no single generic algorithm that can be applied to all patients that can achieve high sensitivity and specificity. The second is that EEG/iEEG

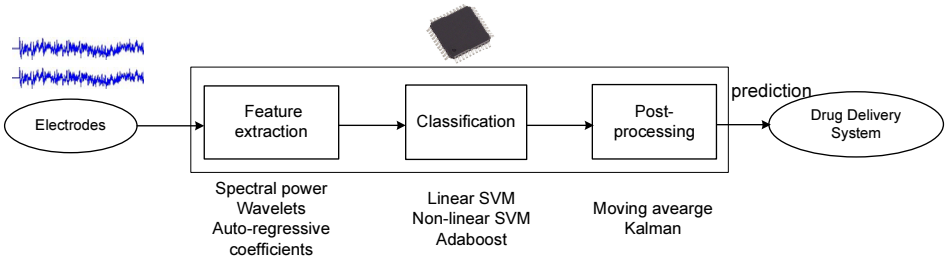


Figure 7.1: Block diagram of a seizure prediction device.

is highly complex and varies over time and no single measure of EEG/iEEG has yet been predictive on its own. It has been shown that a patient-specific classification method based on multiple features extracted from iEEG can achieve high sensitivity and specificity [103]. Fig. 7.1 shows a generalized block diagram of the seizure prediction device. The three major steps of the prediction system include feature extraction, classification and post-processing. Feature extraction step identifies (computes) features (indicators) which can differentiate normal and the abnormal signals. Classification step assigns the computed features into two classes (preictal and interictal). The outliers will be removed in the post-processing stage to make a final prediction.

Our patient-specific approach to seizure prediction is based on binary classification of iEEG using a machine learning algorithm. A machine learning algorithm classifies samples of iEEG as either preictal (immediately prior to a seizure) or interictal (between seizures) based on their multivariate features (see Fig. 7.2). When an epoch of iEEG is classified as preictal, the device can trigger an alarm or a seizure prevention device.

Seizure prediction using electroencephalogram (EEG) with high sensitivity and specificity has been elusive, despite numerous claims that a proposed algorithm or measure has provided significant predictive power. For example, nonlinear measures taken from chaos theory and applied to intracranial EEG (iEEG) demonstrated promising predictive power. However, when compared to linear features, the nonlinear features were not significantly better and their computational intensiveness made them prohibitive to be calculated in real-time. Furthermore, nonlinear features that seemed promising were not predictive at all when tested on long time series.

In [104] Viglione et. al. did early work on seizure prediction in 1975. They attempted

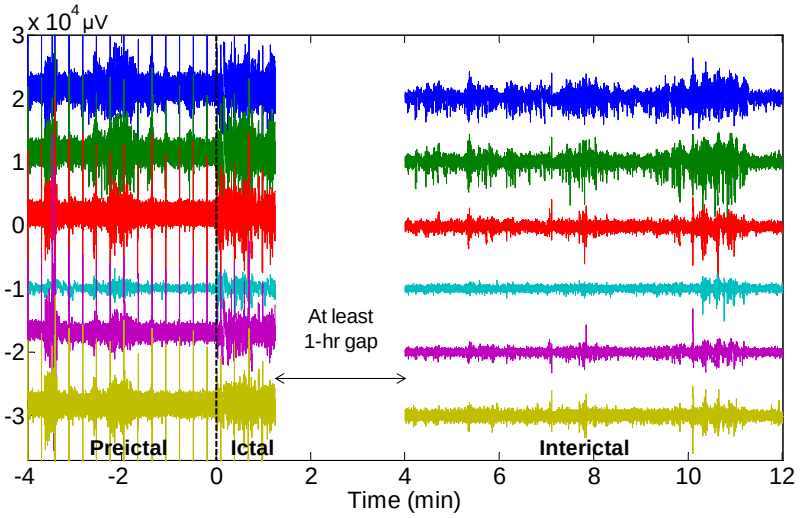


Figure 7.2: Pre-ictal, ictal and inter-ictal iEEG signals.

to extract seizure symptoms from EEG recordings using linear approaches. Another group monitored the spike occurrence rate of EEG, and tried to extract the predictive value [105]. Different kinds of estimation indices ([106] - [111]) and nonlinear dynamics system analyses series ([112] - [117]) have been proposed to model the algorithmic system in seizure prediction. In ([106], [107]) a method to measure the dynamical similarity between different parts of the time series in long-term non-stationary EEG signals is presented. Mormann et. al. [108] measured the phase synchronization of EEG in different brain and proved that the procedure of seizure occurrence is associated with an abnormal synchronization of neurons. In [109], a prediction rule is defined based on comparing energy accumulation and baseline in EEG signals. Direito et al. [110] presented a prediction method based on energy measurement using wavelet coefficients. Alessandro et al. [111] proposed an intelligent genetic search process for EEG signals, which is trained and validated on both series of baseline and pre-seizure records.

In [112], the short-term largest Lyapunov exponent (STLmax) is used as an indicator of chaos system from temporal lobe epilepsy (TLE) patients, and it was found that the chaotic level would decrease minutes prior to seizure. Furthermore, in ([112] - [114]) EEG or electrocorticogram (ECoG) data are analyzed as multi-dimensional systems on time, frequency and space domain with chaotic estimation. However, their study's focus

of interest was entirely limited to the preictal period without processing the interictal recordings. Iasemidis et al. [118] described an adaptive seizure prediction algorithm (ASPA) based on the convergence and divergence of STLmax.

Further in the recent years, features based on power spectral density [103], auto regressive coefficients [119], wavelets [120], [121] and cross-correlation measures [121], [122]. In [119] coefficients of auto-regressive (AR) models are used as features. The prediction is based on SVM classifier. Williamson et. al., [122] presented a method based on multivariate signal coherence. The algorithm uses space-delay correlation and covariance matrices to extract the spatiotemporal correlation structure from multichannel electrocorticogram (ECoG) signals. Wavelet decomposition and cross-correlation techniques are used to predict a seizure event in [120] and [121]. Features based on spectral power in different subbands along with SVM classifier is proposed in [123] to predict a seizure onset.

7.4 Adaboost

Adaboost is a machine learning algorithm, formulated by Freund and Schapire [128]. It is a meta-algorithm, and can be used in conjunction with many other learning algorithms to improve their performance. Adaboost is adaptive in the sense that subsequent classifiers built are tweaked in favor of those instances misclassified by previous classifiers. It can be less susceptible to the overfitting problem than most learning algorithms. The classifiers it uses can be weak (i.e., display a substantial error rate), but as long as their performance is not random (resulting in an error rate of 0.5 for binary classification), they will improve the final model.

The algorithm takes as input a training set where each $\mathbf{x}_i$ belongs to some domain or instance space X of dimension d, and each label y_i is in some label set Y. Pseudo code for Adaboost is given below. Adaboost calls a given weak or base learning algorithm

repeatedly over T iterations.

Adaboost Algorithm:

Given: $(x_1, y_1), ..., (x_m, y_m)$

where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$

For $t = 1, ..., T$:

- Train weak learner using distribution D_t
- Get weak hypothesis $h_t : X \rightarrow \{-1, +1\}$ with error

  $\epsilon_t = Pr_{D_t}[h_t(x_i) \neq y_i]$
- Choose $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
- Update $D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

  where Z_t is normalization factor (chosen such that $D_{t+1}(i)$ will be a distribution)

Output the final hypothesis: $F(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

7.5 Proposed Algorithm

7.5.1 Dataset

The proposed algorithm is evaluated on the Freiburg database [126] which contains electrocorticogram (ECoG) or intracranial electroencephalogram (iEEG) recordings from 21 patients who suffer from epilepsy. The data consists of six channels sampled at 256 Hz with 16 bit analog-to-digital converters. The data records for each patient are divided into ictal and interictal records by certified epileptologists. We have chosen 16 out of the available datasets of 20 patients, who have four or more seizures. Each 20-second long window of iEEG recordings has been categorized as interictal and preictal. Fig. 7.3 shows the steps involved in training and testing phases.

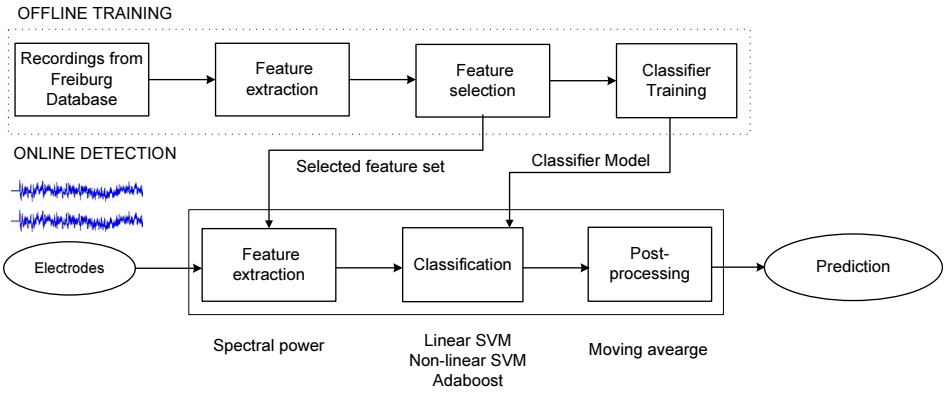


Figure 7.3: Flow chart of the proposed seizure prediction algorithm

7.5.2 Feature Extraction

Preprocessing step is done to remove the artifacts such as line noise, electrical noise, and movement artifacts in iEEG data. When features are extracted, the spectral power in the bands of 47-53 Hz and 97-103Hz are excluded to remove the power line noise. Further, bipolar and time-differential methods have been used to reduce the effect of other types of artifacts in iEEG recordings [123]. The space-differential measurement provides common-mode rejection to reduce line noise and movement artifacts that are common to all the electrodes. Time-differential method is used to normalize the spectral power in high and low frequency bands (refer to Fig. 7.4).

Feature extraction consists of computing the spectral power in different frequency bands in a 20-second long window of iEEG with 50% overlap. This provides a prediction of a seizure every 10 seconds. Spectral bands are selected based on standard iEEG frequency bands but the wide gamma band is split into four bands: delta (0.5-4Hz), theta (4-8Hz), alpha (8-13Hz), beta (13-30Hz), gamma bands (30-47Hz, 53-75Hz, 75-97Hz, 103-128Hz). The power in each band is normalized by the total power and is included as the last feature. Features are extracted from 4 space-differential signals, which makes a total of 36 features for a 20-second window.

7.5.3 Feature Selection

The complexity of the seizure prediction algorithm is proportional to the number of features required to make a prediction. We propose to find a subset of most discriminating

features out of 36 features through the process of feature selection. Feature selection has been known in the machine learning community for many decades. It finds a subset of features which contain essentially most of the relevant information for making decisions. This subset of features may result in an increase in the performance accuracy of the model as irrelevant features are not taken into consideration [127]. Adaboost [128] based feature selection algorithms have been proposed in the literature [129]-[131]. Most of these algorithms did not provide a method to sort the features, but instead used the implicitly selected features in each iteration. We propose a simple criterion based on which features will be ranked.

Algorithm 1 Adaboost

Given : $(x_1, y_1), ..., (x_N, y_N)$
where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize: T, h_t
model = Adaboost(X,y,h_t, T)
model contains $\alpha_t, 1 \leq t \leq T$
decision = $sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

Adaboost algorithm takes input data (training set: $(x_i, y_i), 1 \leq i \leq N$), where each x_i belongs to some domain or instance space X, and each label y_i is in the label set $Y = \{-1, +1\}$. Adaboost calls a given weak or base learning algorithm repeatedly over T iterations. Algorithm 1 shows the basic flow of the Adaboost algorithm. The reader may refer to [128] for the complete algorithm. The base classifier for feature ranking process [132] is defined as follows:

$$h(x) = \{ \begin{array}{c} -1 \text{ if } x_k < v \\ 1 \text{ if } x_k \geq v \end{array} \tag{7.1}$$

where k is a parameter indicating the input variable used to create the split and v is the splitting value. That is, k indicates the feature and v denotes the threshold to differentiate between the two classes. This base classifier is called a "decision stump" as it consists of a classification tree with tree depth of one (a single split decision and two terminal nodes). Parameters k and v are selected to minimize the cost function using a greedy optimization strategy.

Ranking Algorithm

Adaboost with decision stumps as the base classifier inherently performs feature selection. In each iteration, the algorithm selects the most discriminating feature (one with the lowest weighted error) for the corresponding weights. Further, adaboost algorithm generates weight (α) for that particular classifier which signifies the performance of that individual base classifier. In this case, as the base classifier is a decision stump, α signifies the discriminating power of that particular feature used in that iteration. We propose a ranking algorithm based on this observation. Features are sorted by assigning a weight (wt) for each feature. The weight of each feature is computed using α values the algorithm has generated. The outline of the ranking algorithm in presented in Algorithm 2. If a particular feature is not selected at all, then the corresponding weight will be zero.

Algorithm 2 Feature ranking

Given : $(x_1, y_1), ..., (x_m, y_m)$
where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize: T, h_t to be decision stump
model = Adaboost(X,y)
for $f = 1 \rightarrow$d **do**
  id = index of the iteration feature(f) is selected
  $wt(f) = \alpha_t(id)^2$
end for
rank = index(sort(wt))

7.5.4 Classification

In this step, the computed (selected) features are classified into two classes, preictal (+1) and interictal (-1) using a machine learning algorithm. Even though SVMs have demonstrated impressive performance in seizure prediction [123], [119], [122], the computational complexity of the final decision function depends on the type of kernel used during the training process. Table 7.1 presents the complexity analysis of SVMs with three popular kernels [132] (linear, 2_{nd} order polynomial and RBF), where d denotes the number of features and N_{sv} denotes the number of support vectors generated during the training process. We can observe that among three kernels, RBF kernel requires

the highest number of computations while linear kernel requires the lowest. The high computational complexity of the RBF kernel makes it unsuitable for implementing in an implantable device. The best choice would be linear-SVM for reducing the complexity of the seizure prediction algorithm.

Even though, the complexity of the linear-SVM is low, the performance may be degraded if the features used are not linearly separable. We propose to build a non-linear decision function using a combination of linear decision functions (in general linear decision functions are less computationally complex). The decision stumps can act as linear classifiers and can be boosted using the Adaboost algorithm. The main motivation of using Adaboost is its low complexity hardware implementation and the final decision boundary can be non-linear as well. Further, model selection is not required for the Adaboost algorithm which is an advantage compared to the SVM.

Table 7.1: Complexity Analysis of SVM and Adaboost classifiers

Classifier	# ADD	# MUL	# WORDS
SVM Linear	d	d	d
SVM Polynomial (p=2)	d^2	$d(d+1)$	d^2
SVM RBF	$2N_{sv}d$	$N_{sv}d$	$N_{sv}(d+1)$
Adaboost (decision stumps)	T	0	$2T$
*SVM-RBF requires extra N_{sv} exponent operations Adaboost requires extra T comparison operations			

7.5.5 Post-processing

We have observed some isolated false positives at the end of classification step as shown in Fig. 7.5. Post-processing is applied to eliminate these isolated false positives and false negatives. We applied 5-tap moving average filter to smoothen out these isolated events. The final prediction will be made using the filter output.

7.6 Results and Discussion

Fig. 7.3 shows the block diagram for seizure prediction system along with training process. During the training process, features are ranked using the proposed ranking algorithm. Different classification models are built using SVM-Linear and Adaboost classifier with decision stumps for different sizes of feature sets (ranging from 1 to 36). We use double-cross validation to ensure in-sample optimization and out-of-sample testing.

Table 7.2: Comparison of Seizure Prediction Algorithms

	# Pat	# Sz	Sens (%)	FP/hr	Feature	# Features	Classifier
[122]	19	83	90.8	0.094	Corr.	20	SVM-RBF
[119]	9	18	100	0.17	AR	36	SVM-RBF
[123]	18	80	97.5	0.27	S.P	36	SVM-RBF
Proposed	16	71	94.375	0.13	S.P	36	Adaboost
Proposed	16	71	91.25	0.27	S.P	36	SVM-Linear
Proposed	16	71	**94.375**	**0.14**	S.P	**4.8125**	**Adaboost**
Proposed	16	71	67.1825	0.15	S.P	5	SVM-Linear
Proposed	16	71	85.625	0.19	S.P	10	SVM-Linear

Corr. - Correlation
AR - Autoregressive coefficients
S.P - Spectral Power

7.6.1 Performance Analysis

Table 7.2 lists sensitivity, false positive % and number of false positives per hour for 16 patient data sets containing 4 or more seizures. Using all the 36 features and the Adaboost classifier, we achieved an average sensitivity of 94.375 and average FP% of 6.48 as measured by on-duration with 30-min on period for each prediction [123]. The results reported here compare favorably to previously published results. The algorithms from the prior literature requires SVM-RBF which is very computationally complex while the proposed algorithm achieves similar results with low complexity Adaboost classifier.

We can observe that using an average of 5 features, the proposed algorithm is able to achieve high sensitivity and low false positive rate. The detailed feature selection results are presented in Table 7.3. The # Features column represent the number of features

it required to achieve the given sensitivity and false positive rate. Further, Table 7.2 also shows the results with SVM-Linear using best 5 and 10 features. It can seen that the performance using SVM classifier with linear kernel degrades the performance. The chosen features may not be linearly separable which leads to the lower performance with linear kernel.

7.6.2 Complexity Analysis

Table 7.2 also analyzes the seizure prediction algorithms from the recent literature in terms of number, type of features and the classifier used. The number of features used is more than 20 in the prior art using SVM-RBF classifier. The hardware complexity of the SVM-RBF classifier is proportional to $N_{sv} * d$, where N_{sv} is the number of support vectors and d is the dimensionality of the feature vector. The number of support vectors (N_{sv}) depends on the size of the training data set. We observed N_{sv} varying anywhere between 1000 to 3000 during the training process. The computational complexity of SVM-Linear and Adaboost with decision stumps are similar except that the former requires multiplication operation and the later requires comparison operation. The power consumption and the hardware area of these two classifiers are shown in Fig. 7.6. We can observe that Adaboost is a better option between the two when operating at same conditions.

7.6.3 Power estimation

We estimated the power consumption of the proposed algorithm using 65nm technology libraries and HP cacti tool. Table 7.4 summarizes the energy estimates for processing one 20sec window of all 6 channels to make a prediction. The proposed seizure prediction requires only three ADC as we are processing only bipolar signals and using only 5 features. Therefore, only three analog front-end processors as opposed to six. The total energy estimated to make a prediction on one window is 26uJ. A new window is processed every 10s, which leads to a power consumption of 2.6uW on an average.

Table 7.3: Perfomance of the proposed seizure prediction algorithm

Patient #	Sen%	FP/hr	FP %	# Features
1	100	0	0	3
3	100	0	0	4
4	100	0	0	2
5	60	0.7917	36.3657	8
6	100	0.0833	4.1667	4
7	100	0	0	2
9	100	0.25	12.5	4
10	100	0.1667	8.33	4
11	75	0	0	4
12	100	0	0	5
14	75	0.0833	4.1667	8
15	100	0.1667	8.333	8
16	100	0.1667	8.333	8
17	100	0.125	6.25	5
18	100	0.1667	8.33	3
21	100	0.1667	8.333	5
Mean	94.375	0.135	6.487	4.8125

7.7 Conclusion

A low complexity patient specific algorithm is proposed that extracts spectral power based features from EEG recordings. A new feature ranking algorithm is proposed to rank the features. Non-linear classifier is built using Adaboost with decision stumps as base classifier, which makes it computationally less expensive compared to non-linear SVMs. The algorithm achieves high sensitivity and low false positive rate comparable to the previously published results but at a much lower computational complexity. Future

Table 7.4: Energy estimates

Supply Voltage	1V
Technology	65nm
Frequency	1MHz
Digital Processing	$2\mu J$
Analog Front-end	$8\mu J$
Bipolar channels	3
Total energy	$26\mu J$

work will focus on reducing the complexity in the feature extraction step.

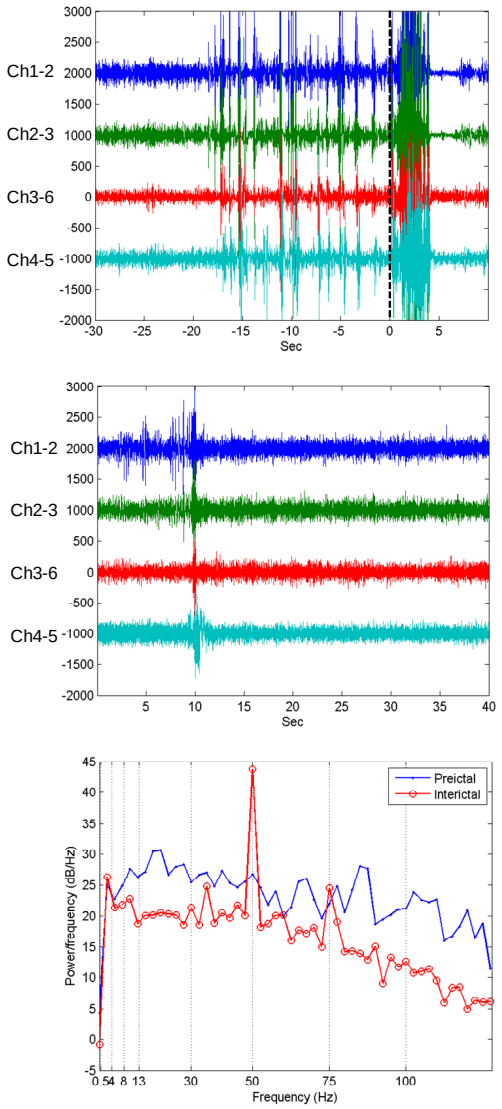


Figure 7.4: Pre-ictal (top), inter-ictal (middle) and their power spectral density (bottom).

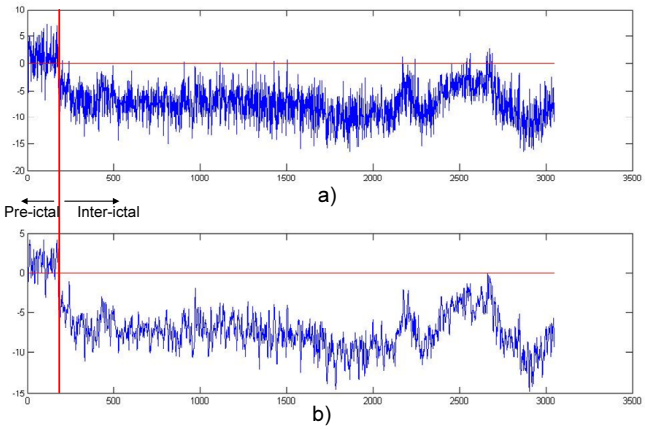


Figure 7.5: Post processing the classifier output with 5-tap moving average filter. a) Classifier output b) After post-processing.

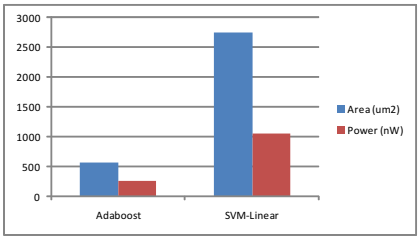


Figure 7.6: Comparison of power and area of SVM-Linear and Adaboost circuits. Circuits are synthesized using 65nm technology operating at 1V Vdd and 1MHz clock frequency.

References

[1] A. Csavoy, G. Molnar, and T. Denison, "Creating support circuits for the nervous system: Considerations for brain-machine interfacing," *International Symposium on VLSI Ciruits,* pp. 4-7, Jun. 2009.

[2] A. Shoeb, B. Bourgeosis, S. T. Treves, S. C. Schachter, and J. Guttag, "Impact of patient-specificity on seizure onset detection performance," *IEEE Int. Conf. on EMBS,* pp. 4110-4114, Aug. 2007.

[3] G. Meyfroidt, F. Guiza, J. Ramon, and M. Bruynooghe, "Machine learning techniques to examine large patient databases," *Best Practive and Research Clinical Anaestiesiology,* vol. 23, pp. 127-143, Mar. 2009.

[4] K. Hung, Y. T. Zhang, B. Tai, "Wearable devices for tele-home healthcare", IEEE Conference on Engineering in Medicine and Biology Society, vol.2, no., pp.5384-5387, Sept. 2004

[5] M. Scheffler, E. Hirt, "Wearable devices for emerging healthcare applications," IEEE Conference on Engineering in Medicine and Biology Society, vol.2, no., pp. 3301- 3304, Sept. 2004

[6] V. N. Vapnik, *The Nature of Statistical Learning Theory*, Springer Verlag, 1995.

[7] C. Burges, "A tutorial on support vector machines for pattern recognition," Data Mining and Knowledge Discovery 2, 121-167 (1998)

[8] C. F. Hsu, M.-K. Ku, L.-Y. Liu, "Support vector machine FPGA implementation for video shot boundary detection application," *IEEE International SOC Conference,* pp.239-242, Sept. 2009.

[9] K. Cao, H. Shen, "Scalable SVM Processor and Its Application to Nonlinear Channel Equalization," *Pacific-Asia Conference on Circuits, Communications and Systems,* pp.206-209, May 2009.

[10] T. Netoff, Yun Park, K. K. Parhi, "Seizure prediction using cost-sensitive support vector machine," Ann. Int. Conf. EMBS, pp. 3322 - 3325, Sept. 2009.

[11] M. Ayinala, M. Brown, K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Transactions on VLSI Systems*,pp. 1068-1081, vol. 20, no. 6, June 2012.

[12] J. W Cooley and J. Tukey, "An algorithm for machine calculation of complex fourier series," *Math. Comput.,* vol. 19, pp. 297-301, Apr. 1965

[13] S. He and M. Torkelson, "A new approach to pipeline FFT processor," *Proc. of IPPS,* 1996, pp. 766 - 770.

[14] S. He and M. Torkelson, "Design and Implementation of 1024-point pipeline FFT processor," *in Proc. Custom Integr. Circuits Conf.,* SantaClara, CA, May 1998, pp. 131-134.

[15] A. V. Oppenheim, R. W. Schafer, J. R. Buck, *Discrete-Time Singal Processing,* 2nd ed. Englewood Cliffs, NJ: Prentice Hall 1998.

[16] P. Duhamel, "Implementation of split-radix FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. on Acoust., Speech Signal Process.,* vol. 34, no. 2, pp. 285-295, Apr. 1986

[17] L. R. Rabiner, B. Gold, *Theory and Application of Digital Signal Processing.* Prentice Hall Inc., 1975.

[18] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementation," *IEEE Trans. Computers,* C-33(5): 414-426, May 1984.

[19] A. M. Despain, "Fourier transfom using CORDIC iterations," *IEEE Trans. Comput.,* C-233(10): 993-1001, Oct. 1974.

[20] E. E. Swartzlander, W. K. W. Young, S.J. Joseph, "A radix-4 delay commutator for fast Fourier transform processor implementation," *IEEE Journal of Solid-state Cir.,* SC-19(5): 702-709, Oct. 1984.

[21] E. E. Swartzlander, V.K. Jain, H. Hikawa, "A radix-8 wafer scale FFT processor," *Journal. VLSI Signal Process.,* 4(2,3): 165-176, May 1992.

[22] G. Bi, E.V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Trans. Acoust., Speech, Signal Process.,* 37(12):1982-1985, Dec. 1989.

[23] J. Lee, H. Lee, S. I. Cho, S. S. Choi, "A High-Speed two parallel radix-2^4 FFT/IFFT processor for MB-OFDM UWB systems," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences,* pp. 1206-1211, April 2008.

[24] J. Palmer, B. Nelson, "A parallel FFT architecture for FPGAs", *Lecture Notes in Computer Science,* vol. 3203, pp. 948-953, 2004.

[25] Y. W. Lin, *et al.*, "A 1-GS/s FFT/IFFT processor for UWB applications," *IEEE Journal of Solid-state Circuits,* vol. 40, no.8 pp. 1726-1735, Aug. 2005.

[26] M. Shin and H. Lee, "A high-speed four parallel radix-2^4 FFT/IFFT processor for UWB applications", *IEEE ISCAS 2008,* pp. 960 - 963, May 2008.

[27] Z. Wang *et al.*, "A Novel FFT Processor for OFDM UWB Systems," *Proc. IEEE APCCAS,* pp. 374-377, Dec. 2006.

[28] S. Qiao *et al.*, "An Area and Power Efficient FFT Processor for UWB Systems," *Proc. IEEE WICOM*, Sept. 2007, pp. 582-585.

[29] C.-P. Fan, M.-S. Lee, and G.-A. Su, "A Low Multiplier and Multiplication Costs 256-Point FFT Implementation with Simplified Radix-24 SDF Architecture," *Proc. IEEE APCCAS*, Dec. 2006, pp. 1935-1938.

[30] Y. Jung, H. Yoon, and J. Kim, "New Efficient FFT Algorithm and Pipeline Implementation Results for OFDM/DMT Applications," *IEEE Trans. Consumer Elect.*, vol. 49, no. 1, Feb. 2003, pp. 14-20.

[31] S.-N Tang, J. Tsai, T.-Y. Chang, "A 2.4-GS/s FFT Processor for OFDM-Based WPAN Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol.57, no.6, pp.451-455, June 2010.

[32] T. Cho, H. Lee, J. Park, C. Park, "A high-speed low-complexity modified radix-2^5 FFT processor for gigabit WPAN applications," *ISCAS*, pp. 1259-1262, 2011.

[33] K. K. Parhi, C. Y. Wang, A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal Solid State Circuits,* vol. 27, no. 1, pp. 29-43, 1992.

[34] K. K. Parhi, "Systematic synthesis of DSP data format converters using lifetime analysis and forward-backward register allocation," *IEEE Trans. on Circuits and Systems - II,* vol. 39, no. 7, pp. 423-440, July 1992.

[35] K. K. Parhi, "Calculation of minimum number of registers in arbitrary life time chart," *IEEE Trans. on Circuits and Systems - II,* vol. 41, no. 6, pp. 434-436, June 1995.

[36] C. Cheng, K. K. Parhi, "High-Throughput VLSI Architecture for FFT Computation," *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol.54, no.10, pp.863-867, Oct. 2007.

[37] Y.-N. Chang and K.K. Parhi, "An Efficient Pipelined FFT Implementation", *IEEE Trans. on Circuits and Systems: Part-II: Analog and Digital Signal Processing*, Vol. 50(6), pp. 322-325, June 2003.

[38] N. Weste, D. J. Skellern, "VLSI for OFDM," *IEEE Communications Magazine,* vol.36, no.10, pp.127-131, Oct 1998.

[39] J. A. C. Bingham, "Multicarrier modulation for data transmission: an idea whose time has come," *IEEE Communications Magazine,* vol.28, no.5, pp.5-14, May 1990.

[40] H. Chi, Z. Lai, "A cost-effective memory-based real-valued FFT and Hermitian symmetric IFFT processor for DMT-based wire-line transmission systems," *IEEE International Symposium on Circuits and Systems*, pp. 6006- 6009 vol. 6, May 2005.

[41] W. Ko, J. Kim, Y. Park, T. Koh, D. Youn, "An efficient DMT modem for the G.LITE ADSL transceiver," *IEEE Transactions on VLSI Systems*, vol.11, no.6, pp.997-1005, Dec. 2003.

[42] A. Wang, A. Chandrakasan, "Energy-efficient DSPs for wireless sensor networks," *IEEE Signal Processing Magazine,* vol.19, no.4, pp.68-78, Jul 2002.

[43] S. F. Reddaway, P. Bruno, P. Rogina, R. Pancoast, "Ultra-high performance, low-power, data parallel radar implementations," *IEEE Aerospace and Electronic Systems Magazine*, vol.21, no.4, pp. 3- 7, April 2006.

[44] Y. Park, L. Luo, K. Parhi, T. Netoff, "Seizure Prediction with Spectral Power of EEG using Cost-Sensitive Support Vector Machines," *Epilepsia*, vol. 52, pp. 1761-1770, Oct. 2011 .

[45] W. W. Smith and J. M. Smith, *Handbook of Real-Time Fast Fourier Transforms,* :Wiley-IEEE Press, 1995.

[46] H. Ziegler, "A fast Fourier transform algorithm for symmetric real-valued series," *IEEE Trans. Audio Electroacoust.* vol. AU-20, pp. 353-356, Dec. 1972.

[47] J. B. Martens, "Discrete Fourier transform algorithms for real valued sequence," *IEEE Trans. Acoust., Speech, Sig. Proc.* , vol. ASSP-32, pp. 390-396, Apr. 1984.

[48] H. Sorensen, D. Jones, M. Heideman, C. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol.35, no.6, pp. 849- 863, Jun 1987.

[49] J. Chen, H. Sorensen, "An efficient FFT algorithm for real-symmetric data," *IEEE International Conference on Acoustics, Speech, and Signal Processing,* vol.5, pp.17-20 Mar 1992.

[50] B. R. Sekhar, and K. M. M. Prabhu, "Radix-2 decimation in frequency algorithm for the computation of the real-valued FFT," *IEEE Trans, Signal Process.,* vol. 47, no. 4, pp. 1181-1184, Apr. 1999.

[51] B. M. Baas, "A low-power, high-performance, 1024-point FFT processor," *IEEE Journal of Solid-State Circuits,* vol. 34, n0. 3, pp. 380-387, Mar. 1999.

[52] M. Garrido, K. K. Parhi, J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Cir. Sys. I, Reg. Papers,* vol. 56, no. 12, pp. 2634 - 2643, Dec. 2009.

[53] A. Wang, A. P. Chandrakasan, "Energy-aware architectures for a Real-Valued FFT implementation," *International Symposium on Low Power Electronics and Design*, pp. 360- 365, Aug. 2003.

[54] M. Ayinala and K. K. Parhi, "Parallel-Pipelined radix-2^2 FFT architecture for real valued signals," *Asilomar conference on signals, systems and computers,* pp. 1274 - 1278, Nov. 2010.

[55] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation.* Hoboken, NJ: Wiley, 1999.

[56] M. Ayinala and K.K. Parhi, "Parallel Pipelined FFT Architectures with Reduced Number of Delays," *ACM Great Lakes Symp. on VLSI*, pp. 63-66, May 2012.

[57] R. Radhouane, P. Liu, and C. Modin, "Minimizing the memory requirement for continuous flow FFT implementation: Continuous flow mixed mode FFT (CFMM-FFT)," *IEEE Int. Symp. Circuits and Systems,* pp. 116-119, May 2000.

[58] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Trans. Circuits and Systems I, Reg. Papers,* vol. 52, no. 5, pp. 911-919, May 2005.

[59] A. T. Jacobson, D. N. Truong, and B. M. Baas, "The design of a reconfigurable continuous-flow mixed-radix FFT processor," *IEEE Int. Symp. Circuits and Systems,* pp. 1133-1136, May 2009.

[60] W. Chao and H. Y. Peng, "Twin butterfly high throughput parallel architecture FFT algorithm," *Int. Symp. on Information Science and Engineering,* pp. 637-640, Dec. 2008.

[61] C. F. Hsiao, Y. Chen and C. Y. Lee, "A generalized mixed-radix algorithm for memory-based FFT processors," *IEEE Trans. on Circuits and Systems II, Exp. Briefs,* vol. 57, no. 1, pp. 26-30, Jan. 2010.

[62] P.-Y Tsai and C.-Y Lin, "A Generalized Conflict-Free Memory Addressing Scheme for Continuous-Flow Parallel-Processing FFT Processors With Rescheduling," *IEEE Transactions on Very Large Scale Integration Systems,* vol.19, no.12, pp.2290-2302, Dec. 2011.

[63] H. Chi and Z. Lai, "A cost-effective memory-based real-valued FFT and Hermitian symmetric IFFT processor for DMT-based wire-line transmission systems," *IEEE International Symposium on Circuits and Systems*, pp. 6006- 6009 vol. 6, May 2005.

[64] A. Wang and A. P. Chandrakasan, "Energy-aware architectures for a Real-Valued FFT implementation," *International Symposium on Low Power Electronics and Design*, pp. 360- 365, Aug. 2003.

[65] P. Stoica, and R.L. Moses, *Introduction to Spectral Analysis*, Prentice-Hall, 1997.

[66] M. Hayes, *Statistical Digital Signal Processing and Modeling*, John Wiley & Sons, 1996.

[67] F. El-Hawary, T. Richards, "A Systolic Computer Architecture For Spectrum Analysis," *Proceedings OCEANS*, vol.4, pp.1061-1065, Sep 1989.

[68] T.-H. Yu, C. -H. Yang; D. Cabric, D. Markovic, "A 7.4mW 200MS/s wideband spectrum sensing digital baseband processor for cognitive radios," *Symposium on VLSI Circuits*, pp.254-255, June 2011.

[69] T. Yucek, H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *IEEE Communications Surveys & Tutorials*, vol.11, no.1, pp.116-130, 2009.

[70] A. Oppenheim, R. Schafer, *Discrete Time Signal Processing*, 2nd edition, Prentice-Hall.

[71] S. Haykin, *Adaptive Filter Theory*, 4th edition, Prentice-Hall.

[72] Welch, P.D, "The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms," *IEEE Trans. Audio Electroacoustics*, Vol. AU-15 (June 1967), pp.70-73.

[73] S. Zhang, D. Yu, S. Sheng, "A Discrete STFT Processor for Real-time Spectrum Analysis," *IEEE Asia Pacific Conference on Circuits and Systems*, pp.1943-1946, 2006.

[74] A. Sanchez, M. Garrido, L. Vallejo, J. Grajal, C. Lopez-Barrio, "Digital channelised receivers on FPGAs platforms," *IEEE International Radar Conference*, pp. 816-821, 2005.

[75] T. I. Laakso, V. Valimaki, M. Karjalainen, U. K. Laine, "Splitting the unit delay [FIR/all pass filters design]," *IEEE Signal Processing Magazine*, vol.13, no.1, pp.30-60, Jan 1996.

[76] E. Hermanowicz, "Explicit formulas for weighing coefficients of maximally flat tunable FIR delayers", *Electron. Letters,* vol. 28, no. 20, pp. 1936-1937, Sept. 1992.

[77] Freiburg EEG database. Available from: https://epilepsy.uni-freiburg.de/freiburg-seizureprediction-project/eeg-database.

[78] V. Sundararajan and K.K. Parhi, "A Novel Multiply Multiple Accumulator Component for Low Power PDSP Design", *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Vol. 6, pp. 3247-3250, June 2000.

[79] S. Thoziyoor, N.Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5, Advanced architecture laboratory HP Laboratories," HPL-2007-167, 2007. [Online]. Available: http://www.hpl.hp.com/ techreports/2008/HPL-200820.html

[80] Y. Park, L. Luo, K.K. Parhi and T. Netoff, "Seizure Prediction with Spectral Power of EEG Using Cost-Sensitive Support Vector Machines," *Epilepsia*, 52(10), pp. 1761-1770, Oct. 2011.

[81] Y. Can, M. T. Coimbra, B. Vijaya Kumar, "Arrhythmia detection and classification using morphological and dynamic features of ECG signals," *IEEE EMBC,* pp.1918-1921, 2010.

[82] M. Shoaib, N. Jha, N. Verma, "A low-energy computation platform for data-driven biomedical monitoring algorithms," *ACM/IEEE Design Automation Conference (DAC)*, pp.591-596, June 2011.

[83] S. Cadambi, I. Durdanovic, V. Jakkula, et. al, "A Massively Parallel FPGA-Based Coprocessor for Support Vector Machines," *17th IEEE Symposium on FCCM*, pp.115-122, April 2009.

[84] M. Papadonikolakis and C. Bouganis, "A novel FPGA-based SVM classifier," *Intl. Conf. on Field-Programmable Technology,* pp.283-286, Dec. 2010.

[85] D. Mohapatra, V.K. Chippa, A. Raghunathan and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," *Design, Automation & Test in Europe Conference (DATE)*, pp.1-6, March 2011.

[86] V .Chippa, D. Sohapatra, A. Raghunathan, K. Roy, S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," *ACM/IEEE Design Automation Conference*, pp.555-560, June 2010.

[87] Y. Lee and O. L. Mangasarian, "RSVM: Reduced support vector machines," *Data Mining Institute, Computer sciences Department, University of Wisconsin,* 2001.

[88] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," *Intl. Conf. on Machine Learning,* pp. 121-129, 1994.

[89] Freiburg EEG database. Available from: https://epilepsy.uni-freiburg.de/freiburg-seizureprediction-project/eeg-database.

[90] K.-J. Cho, K.-C. Lee, J.-G. Chung, and K.K. Parhi, "Design of Low-Error Fixed Width Modified Booth Multiplier", *IEEE Trans. on VLSI Systems*, 12(5), pp. 522-531, May 2004.

[91] S. J. Jou and H. H.Wang, "Fixed-width multiplier for DSP application," *Int. Conf. Computer Design (ICCD)*, Sept. 2000, pp. 318-322.

[92] D. Oh, K. K. Parhi, "Min-Sum Decoder Architectures With Reduced Word Length for LDPC Codes," *IEEE Transactions on Circuits and Systems I: Regular Papers,* vol.57, no.1, pp.105-115, Jan. 2010.

[93] S. Thoziyoor, N.Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5, Advanced architecture laboratory HP Laboratories," HPL-2007-167, 2007. [Online]. Available: http://www.hpl.hp.com/ techreports/2008/HPL-200820.html

[94] R. Fisher, V. Salanova, T. Witt, R. Worth, T. Henry, R. Gross, K. Oommen, I. Osorio, J. Nazzaro, and D. Labar, "Electrical stimulation of the anterior nucleus of thalamus for treatment of refractory epilepsy," *Epilepsia*, vol. 51, pp. 899-908, 2010.

[95] B. Schelter, J. Timmer, and A. Schulze-Bonhage, *Seizure prediction in epilepsy: from basic mechanisms to clinical applications*: Vch Pub, 2008.

[96] F. Mormann, R. Andrzejak, C. Elger, and K. Lehnertz, "Seizure prediction: the long and winding road," *Brain*, vol. 130, p. 314, 2007.

[97] J. F. Annegers, *The epidemiology of epilepsy*, 1996. Available: http://www.epilepsyfoundation.org/about/statistics.cfm

[98] X. Yang, B. Schmidt, D. Rode, and S. Rothman, "Optical suppression of experimental seizures in rat brain slices," *Epilepsia*, vol. 51, pp. 127-135, 2009.

[99] S. Rothman, M. Smyth, X. Yang, and G. Peterson, "Focal cooling for epilepsy: an alternative therapy that might actually work," *Epilepsy & Behavior*, vol. 7, pp. 214-221, 2005.

[100] A. Csavoy, G. Molnar, and T. Denison, "Creating support circuits for the nervous system: considerations for brain- machine interfaceing," *Proc. Int. Symp. VLSI Circuits*, pp. 4-7, 2009.

[101] A. Shoeb, et. al., "Impact of patient-specificty on seizure onset detection performance, " *IEEE EMBS*, pp. 4110-4114, Aug. 2007.

[102] B. Litt and J. Echauz, "Prediction of epileptic seizures," *Lancet Neurology*, vol. 1, pp. 22-30, 2002.

[103] Y. Park, T. Netoff, K. Parhi, "Seizure prediction with spectral power of time/space-differential EEG signals using cost-sensitive support vector machine," *IEEE ICASSP*, pp.5450-5453, March 2010.

[104] S. S. Viglione and G. O. Walsh, "Proceedings: Epileptic seizure prediction," *Electroencephalogr Clin Neurophysiol*, vol. 39, pp. 435-6, Oct 1975.

[105] H. H. Lange, J. P. Lieb, J. Engel, Jr., and P. H. Crandall, "Temporospatial patterns of pre-ictal spike activity in human temporal lobe epilepsy," *Electroencephalogr Clin Neurophysiol*, vol. 56, pp. 543-55, Dec 1983.

[106] M. L. Van Quyen, J. Martinerie, M. Baulac, and F. Varela, "Anticipating epileptic seizures in real time by a non-linear analysis of similarity between EEG recordings," *Neuroreport*, vol. 10, pp. 2149-2155, 1999.

[107] V. Navarro, J. Martinerie, M. L. V. Quyen, S. Clemenceau, C. Adam, M. Baulac, and F. Varela, "Seizure anticipation in human neocortical partial epilepsy," *Brain*, vol. 125, pp. 640-655, March 1, 2002 2002.

[108] F. Mormann, T. Kreuz, R. G. Andrzejak, P. David, K. Lehnertz, and C. E. Elger, "Epileptic seizures are preceded by a decrease in synchronization," *Epilepsy Research*, vol. 53, pp. 173-185, Mar 2003.

[109] B. Litt, R. Esteller, J. Echauz, M. D'Alessandro, R. Shor, T. Henry, P. Pennell, C. Epstein, R. Bakay, M. Dichter, and G. Vachtsevanos, "Epileptic Seizures May Begin Hours in Advance of Clinical Onset: A Report of Five Patients," *Neuron*, vol. 30, pp. 51-64, 2001.

[110] B. Direito, A. Dourado, M. Vieira, and F. Sales, "Combining Energy and Wavelet Transform for Epileptic Seizure Prediction in an Advanced Computational System," *in International Conference on BioMedical Engineering and Informatics*, pp. 380-385, 2008.

[111] M. D'Alessandro, R. Esteller, G. Vachtsevanos, A. Hinson, J. Echauz, and B. Litt, "Epileptic seizure prediction using hybrid feature selection over multiple intracranial EEG electrode contacts: a report of four patients," *IEEE Transactions on Biomedical Engineering*, vol. 50, pp. 603-615, 2003.

[112] L. D. Iasemidis, J. Chris Sackellares, H. P. Zaveri, and W. J. Williams, "Phase space topography and the Lyapunov exponent of electrocorticograms in partial seizures," *Brain Topography*, vol. 2, pp. 187-201, 1990.

[113] L. D. Iasemidis, S. Deng-Shan, J. C. Sackellares, P. M. Pardalos, and A. Prasad, "Dynamical resetting of the human brain at epileptic seizures: application of

nonlinear dynamics and global optimization techniques," *IEEE Transactions on Biomedical Engineering*, vol. 51, pp. 493-506, 2004.

[114] L. D. Iasemidis, J. C. Principe, and J. C. Sackellares, "Measurement and quantification of spatio-temporal dynamics of human epileptic seizures," *in Nonlinear Biomedical Signal Processing*, vol. 2, pp. 294-318, 2000.

[115] K. Lehnertz and C. E. Elger, "Spatio-temporal dynamics of the primary epileptogenic area in temporal lobe epilepsy characterized by neuronal complexity loss," *Electroencephalography and Clinical Neurophysiology*, vol. 95, pp. 108-117, 1995.

[116] I. Osorio, M. A. Harrison, Y. C. Lai, and M. G. Frei, "Observations on the application of the correlation dimension and correlation integral to the prediction of seizures," *Journal of Clinical Neurophysiology*, vol. 18, pp. 269-74, May 2001.

[117] L. D. Iasemidis, "Epileptic seizure prediction and control," *IEEE Transactions on Biomedical Engineering*, vol. 50, pp. 549-558, 2003.

[118] L. D. Iasemidis, S. Deng-Shan, W. Chaovalitwongse, J. C. Sackellares, P. M. Pardalos, J. C. Principe, P. R. Carney, A. Prasad, B. Veeramani, and K. Tsakalis, "Adaptive epileptic seizure prediction system," *IEEE Transactions on Biomedical Engineering*, vol. 50, pp. 616-627, 2003.

[119] L. Chisci, A. Mavino, G. Perferi, et. al., "Real-Time epileptic seizure prediction using AR models and support vector machines," *IEEE Transactions on Biomedical Engineering*, vol.57, no.5, pp.1124-1132, May 2010.

[120] C. C. B. Suarez, E. T. Fonoff, et. al., "Wavelet transform and cross-correlation as tools for seizure prediction," *IEEE EMBC*, pp.4020-4023, Sept. 2010.

[121] S. Hung, C. Chao, S. Wang, et. al., "VLSI implementation for epileptic seizure prediction system based on wavelet and chaos theory," *IEEE Region Conference TENCON*, pp.364-368, Nov. 2010.

[122] J. R. Williamson, D. W. Bliss, D.W. Browne, "Epileptic seizure prediction using the spatiotemporal correlation structure of intracranial EEG," *IEEE ICASSP*, pp.665- 668, May 2011.

[123] Y. Park, L. Luo, K.K. Parhi and T. Netoff, "Seizure prediction with spectral power of EEG using cost-sensitive support vector machines," *Epilepsia*, 52(10), pp. 1761-1770, Oct. 2011.

[124] A. Shoeb, et al., "A micro support vector machine based seizure detection architecture for embedded medical devices," *IEEE EMBS*,pp. 4202-4205. Sept 2009.

[125] P. W. Mirowski, et al., "Comparing SVM and convolutional networks for epileptic seizure prediction from intracranial EEG," *IEEE Workshop on MLSP,* pp.244-249, Oct. 2008.

[126] Freiburg EEG database. Available from: https://epilepsy.uni-freiburg.de/freiburg-seizureprediction-project/eeg-database.

[127] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," *International Conference on Machine Learning*, pp. 121-129, 1994.

[128] Y. Freund, R. Schapire, "Experiments with a new boosting algorithm," *International Conference on Machine Learning*, pp. 148-156, 1996.

[129] L. Furst, S. Fidler, A. Leonardis, "Selecting features for object detection using an Adaboost-compatible evaluation function," *Patter recognition letters*, pp. 1603-1612, 2008.

[130] P. Silapachote, D. R. Karuppiah, A. R. Hanson, "Feature selection using adaboost for face expression recognition," *Intl. Conference on Visualization, Image, and Image Processing*, Spain, Sept. 2004.

[131] D. D. Le, "Feature selection by adaBoost for SVM-based face detection," *Forum on Information Technology*, pp. 183-186, 2004.

[132] V. Cherkassky, F. Mulier, *Learning from data, Concepts, Theory and Methods*, Wiley.