UNIVERSITÉ DE MONTRÉAL

POWER-EFFICIENT HARDWARE ARCHITECTURE FOR COMPUTING SPLIT-RADIX FFT ON HIGHLY SPARSE SPECTRUM

HANIEH ABDOLLAHIFAKHR

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L’OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

AOÛT 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

POWER-EFFICIENT HARDWARE ARCHITECTURE FOR COMPUTING SPLIT-RADIX FFT ON HIGHLY SPARSE SPECTRUM

présenté par : ABDOLLAHIFAKHR HANIEH

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. DAVID Jean-Pierre, Ph. D., président

M. SAVARIA Yvon, Ph. D., membre directeur de recherche

M. GAGNON François, Ph. D., membre et codirecteur de recherche

M. LANGLOIS Pierre, Ph. D., membre

DEDICATION

To My beloved, Jaber

To My Lovely Mother, Malihe

To My Patient and Sympathetic Father, Mahmoud

And to My Kind Brother, Hamon

Thank you for being with me in my life, my life is meaningful only with you.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to professors Yvon Savaria and François Gagnon and to Normand Bélanger for all their support, patience, generosity and advices during my master's studies in Polytechnique Montréal.

RÉSUMÉ

Le problème du transfert des signaux du domaine temporel au domaine fréquentiel d'une manière efficace, lorsque le contenu du spectre de fréquences a une faible densité, est le sujet de cette thèse. La technique bien connue de la transformée de Fourier rapide (FFT) est l'algorithme de traitement de signal privilégié pour observer le contenu fréquentiel des signaux entrants à des émetteurs-récepteurs de télécommunication, tels que la radio cognitive, ou la radio définie par logiciel qu'on utilise habituellement pour l'analyse du spectre dans une bande de fréquences. Cela peut représenter un lourd fardeau de calcul sur des processeurs lorsque la FFT ordinaire est mise en œuvre, ce qui peut impliquer une consommation d'énergie considérable. L'alimentation en énergie est une ressource limitée dans les appareils mobiles et, par conséquent, cette ressource peut être critique pour des dispositifs de télécommunications mobiles.

Dans le but de développer un processeur économe en énergie pour les applications de transformation temps-fréquence, un algorithme de transformée de Fourier plus efficace, en termes du nombre de multiplications et d'additions complexes, est sélectionné. En effet, la Split-Radix Fast Fourier Transform (SRFFT) offre une performance meilleure que la FFT classique en termes de réduction du nombre de multiplications complexes nécessaires et elle peut donc conduire à une consommation d'énergie réduite.

En appliquent le concept d'élagage des calculs inutiles, c'est-à-dire des multiplications complexes avec entrées ou sorties à zéro, tout au long de l'algorithme, on peut réduire la consommation d'énergie.

Ainsi, une architecture matérielle énergétiquement efficace est développée pour le calcul de la SRFFT. Cette architecture est basée sur l'élagage des calculs inutiles. En fait, pour tirer parti du potentiel de la SRFFT, une nouvelle architecture d'un processeur de SRFFT configurable est d'abord conçue, puis l'architecture est développée afin d'éliminer les calculs inutiles. Cela se fait par l'utilisation appropriée d'une matrice d'élagage. Le processeur proposé peut trouver des applications dans le multiplexage orthogonal en fréquence (OFDM) dans des émetteurs-récepteurs de communication, où les signaux transmis peuvent n'occuper qu'une petite partie de l'ensemble du spectre opérationnel. Le processeur est mis en œuvre sur un circuit programmable (FPGA) et sa fonctionnalité ainsi que sa performance sont validées par des simulations.

En outre, la consommation de puissance des processeurs SRFFT avec ou sans moteur d’élagage est analysée via le simulateur de puissance du simulateur. Prenant le processeur SRFFT sans élagage comme référence, les simulations de consommation d'énergie montrent que l'économie d'énergie maximale obtenue est de l'ordre de 20% lors de l'élagage d’une FFT de 1024 points appliquée à des signaux de spectre très clairsemé.

ABSTRACT

The problem of transferring a time domain signal into the frequency domain in an efficient manner, when the frequency contents are sparsely distributed, is the research topic covered in this thesis. The well-known Fast Fourier Transform (FFT) is the most common signal processing algorithm for observing the frequency contents of incoming signals in telecommunication transceivers. It is notably used in cognitive or software defined radio which usually demands for monitoring the spectrum in a wide frequency band. This may imply a heavy computation burden on processors when the ordinary FFT algorithm is implemented, and hence yield considerable power consumption. Power and energy supply is a limited resource in mobile devices and therefore, efficient execution of the Fourier transform has turned out to be critical for mobile telecommunication devices.

With the purpose of developing a power-efficient processor for time-frequency transformation, the most computationally efficient Fourier transform algorithm is selected among the existing Fourier transform algorithms upon studying them in terms of required arithmetic operations, i.e. complex multiplications and additions. Indeed, the Split-Radix Fast Fourier Transform (SRFFT) offers a performance that is better than conventional FFT in terms of reduced number of complex multiplications and hence, can reduce power consumption.

Appling the concept of pruning of the unnecessary computations, i.e. complex multiplications with either zero inputs or outputs, throughout the whole algorithm may reduce the power consumption even further.

Thus, a power efficient hardware architecture implementing the SRFFT is developed through pruning unnecessary computations. In fact, leveraging the potential in SRFFT algorithm, a new architecture of a configurable SRFFT processor is first devised and then the architecture is developed further so that unnecessary computations, which yield zeros at the output, are pruned. This is done through stalling butterfly computations with appropriate use of a pruning matrix. The proposed processor may find applications in orthogonal frequency division multiplexing (OFDM) communication transceivers, where transmitted signals may occupy only a small portion of the whole operational spectrum. The processor is implemented on a field-programmable gate array (FPGA) and its performance is validated via simulations.

In addition, the power consumption analysis of the SRFFT processors with and without the pruning engine is performed using a power analysis simulation tool.

Taking the SRFFT processor without pruning engine as a reference, the power consumption simulations show that a maximum power saving of around 20% is achieved when the pruning engine outputs are actively used in to the SRFFT processor computing the 1024-point Fourier Transform of signals with a very sparse spectrum.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

FFT	*Fast Fourier Transform*
SRFFT	*Split Radix Fast Fourier Transform*
OFDM	*Orthogonal Frequency Division Multiplexing*
NC-OFDM	*Non-Contiguous Orthogonal Frequency Division Multiplexing*

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Background and motivation

Along with the increasing demand for radio technology related services, the issue of developing techniques for utilizing available radio-frequency bands optimally is currently an important research topic. Furthermore, the techniques are being developed with the purpose of preventing interferences, which may reduce the data throughput and channel reliability in wireless networks [1, 2, 3, 4].

On the other hand, the licensed part of the radio spectrum is often poorly used and therefore practical means must be developed to improve spectrum utilization. Therefore, a growing attention has been given to configurable and multi-standard radios [5, 6]. Software defined radios (SDRs) that integrate various radio architectures in the back-end for different standards [5] or cognitive radios (CRs) that can reconfigure the architecture and optimize the operational parameters in a reconfigurable manner [6] are good examples of the recent developments in this regard.

Spectrum sensing plays an important role nowadays in several wireless communication techniques such as backhaul small cell communication [7] or CR systems [4]. Cognitive radio is explored as a means to improve wireless communications and it could become a reliable solution to the spectrum underutilization problem. Furthermore, the ultimate objective of CR systems is to provide highly reliable and available communication means between all users and to also facilitate more efficient utilization of the radio spectrum. CRs are actually able to sense the spectrum in the environment and recognize unused frequency bands so that they can make best use of available resources. In CR, several unused frequency bands could be located in different segments of some reserved spectrum. Furthermore, spectrum allocation can be done while preventing interference with other frequency bands that are actively used. This technique is called spectrum pooling [8].

Theoretically, an orthogonal frequency division multiplexing OFDM-based Cognitive Radio system can optimally approach the Shannon capacity in the segmented spectrum using adaptive resource allocation on each subcarrier.

In these OFDM transceivers, the radio frequency (RF) signals may not include all subcarriers, implying that the transceivers in cognitive radio devices try to remain aware of the spectrum segments that are used and unused. A common technique for spectrum sensing is Digital Signal Processing (DSP), specifically the Fast Fourier Transform (FFT) algorithm. Indeed, OFDM transmits a large number of narrowband carriers, closely spaced in the frequency domain. In order to avoid a large number of modulators and filters at the transmitter and complementary filters and demodulators at the receiver, it is desirable to be able to use modern digital signal processing techniques, such as the FFT. Thus, one consequence of maintaining orthogonality is that the OFDM signal can be defined using Fourier transform procedures. But, the FFT of the operational signal may yield many zero-valued frequency bins or bins that are "Don't Cares" for some applications. This means that a processor performing those calculations may carry out many unnecessary computations, which increase power consumption. Considering that mobile devices must minimize power consumption in order to ensure that the batteries of those devices will power the device for a sufficiently long period of time on a single charge, our goal is to propose means to minimize the number of computations performed.

The rest of this thesis is organized as follows. In Chapter 2, the reported research in this area is reviewed with the purpose of contextualizing this thesis work. Subsequently, efficient fast Fourier transform algorithms are studied and compared in order to develop the most appropriate algorithm to be applied to sparse spectrum in Chapter 3. Then, the architecture of the proposed FFT processor is explained in detail in the following Chapter 4. Also, the pruning engine which is added on purpose is presented and finally, the overall power consumption of the proposed FFT core with and without the pruning engine is compared through power simulation analysis upon implementation on FPGA. A power savings on the order of around 20% is concluded in Chapter 5.

CHAPTER 2 PREVIOUS CONTRIBUTIONS ON FAST FOURIER TRANSFORM PRUNING

The considerable demand for computationally efficient FFT algorithms and processors in wireless technology, in particular for NC-OFDM transceivers, has driven a research in this direction [12, 17]. Furthermore, developing FFT algorithms with minimum number of computations when a significant number of zeros exist in either the input or the output of the FFT has been subjected to intensive research [13-25]. Reconfigurable algorithms that can adapt to the conditions at the input or output may be used to reduce power consumption. Because means of effectively computing the Fourier transform may find applications in today's mobile wireless communication devices, development and implementation of power efficient FFT cores have come highly relevant. In this chapter, the papers that are the basis of the current research in this area are categorized and briefly explained in order to contextualize this thesis work.

Except for work by [12] on the development of low-power FFT architectures, which was achieved through optimization of an FFT core, the majority of the published research was pursuing a specific direction, i.e. omitting unnecessary computations. This concept, which is termed "pruning", has been introduced to eliminate useless computations [13-17]. The idea of pruning was first suggested in [13] with the purpose of increasing the computational efficiency for the decimation-in-frequency (DIF) structured FFT performed on signals whose number of non-zeros is considerably less than the length of the FFT. The main objective was to increase the computational efficiency by decreasing the total number of arithmetic operations. The technique is proposed with the purpose of reducing the computation time. Indeed, all computational operations that yield zeros are not carried out.

A very similar algorithm was developed [14] for computing the FFT of a decimated-in-time (DIT) signal. Like the pruning algorithm proposed by Markel, this allowed reducing the overall processing time. However, Skinner's pruning algorithm for DIT achieves a better reduction factor on the total number of multiplications. The major limitation of Markel's or Skinner's pruning algorithms is that they can only compute a contiguous series of output starting from the first one.

Therefore, a pruning technique was proposed that can prune an FFT according to zeros at either the input or output stages [19, 20]. Moreover, two efficient algorithms were proposed for simultaneous DIT and DIF, in [19] and [20] respectively. As a matter of fact, these methods combine Markel's pruning method to the output of Skinner's algorithm.

Another pruning technique was proposed [15] by which a matrix could be established in order to determine whether a specific computation must be conducted or not. Furthermore, depending on where pruning is to be implemented (i.e. at input or output), two different matrices of size $N \times log_2N$ are formed to specify which computations are useful. Input and output pruning matrices are obtained from the first and last column of the FFT computation diagram, respectively. In the pruning matrix, the non-zero terms at both input and output vectors are denoted by unity and the rest of the matrix is obtained from these vectors as the first or last column from the input or output pruning matrices, respectively. The main contribution of this algorithm is that the number of non-zero inputs or non-zero outputs as well as their location can be arbitrary. It also works equally well for both DIT and DIF architectures.

In [16], a pruning algorithm is proposed, which is mostly built upon previous algorithms. A considerable reduction in computation time was reported by employing the proposed matrix. It is claimed that the algorithm can efficiently and quickly prune the FFT for NC-OFDM transceivers. Contrary to the general FFT pruning proposed by [15], it avoids using conditional terms and therefore brings a considerable reduction in computation time. Unlike previous work, the size of the pruning matrix proposed by [16] for an N-point FFT is $(\mathrm{N}/2) \times \log_2\mathrm{N}$, which is half of the size reported in [15].

The main drawback of existing pruning algorithms is their high resources utilization, i.e. data memory for storage of the configuration matrix. In [17], FFT pruning for NC-OFDM transceivers is considered again with the purpose of reducing its inevitable resource requirements when implemented. Partial pruning allows the pruning of the input until a given depth in the FFT structure, which yields a reduction in memory requirements. Moreover, through partial pruning, a compromise is made between the computational time saved through pruning and the required size of the memory for data storage. Through the developed partial pruning algorithm, a time saving close to the full pruning algorithms was achieved after implementation while 20% less memory was required in comparison to previously reported algorithms [15, 16]. This made the algorithm more suitable for embedded-system applications with limited memory resources.

In [21], a pruning algorithm called input-zero-traced FFT pruning (IZTFFTP) was proposed. It is based on the Cooley-Tukey radix-2 FFT algorithm for the applications in NC-OFDM transceiver, where the number of non-zeros outnumbers the number of zeros at either the input or the output. Through C++ simulations, the performance of the algorithm was compared with the non-pruned FFT and the reduction in the overall arithmetic operations was reported in a table. For example, 5120 complex multiplication in ordinary 1024 point FFT was reduced down to 2448 with the proposed pruned FFT with the same length. Throughout comparison of the execution time of operation of the proposed pruned FFT with the ordinary FFT, it was concluded that the computational complexity was reduced considerably.

In [13, 22], Sorensen et al. proposed an efficient algorithm called transform decomposition. Transform decomposition can be seen as a modified Cooley-Tukey FFT where the DFT is decomposed into two smaller DFTs. Transform decomposition is more efficient and flexible than FFT pruning when it is applied on the SRFFT algorithm but it is less efficient when applied on the FFT algorithm. Note that no work was found in the literature about pruning the transform decomposition.

Qiwei Zhang et al. developed a computationally efficient IFFT/FFT for OFDM-based cognitive radio on a reconfigurable architecture [23]. The proposed algorithm was based on transform decomposition [24].

As already mentioned, the fundamental strategy to reduce both overall computational time and power is to perform as few multiplications as possible. Hence, the FFT algorithm with minimum number of arithmetic operations should be the optimum algorithm for hardware implementations. Therefore, the theoretical total number of required complex multiplications in different FFT formats is compared in Table 2-1, and the values are plotted in Figure 2-1 for an easier comparison for a typical 1024-point FFT. One can observe that, among several FFT formats, including the radix-2 and the split-radix (SR), the latter requires the smallest number of multiplications and additions. Also, different formats of FFT pruning techniques are compared in accordance with the equations provided in Table 2-1. The superior performance of SRFFT pruning in comparison to transform decomposition can be observed in Figure 2-1. Note that, unlike the Radix-2 diagram with in-place computation, the computations in the SRFFT are not carried out stage by stage, but its L-shaped butterfly advances with computations stage by stage at only its top half, while combining all computations of two subsequent stages at once in its lower half, much as in a radix-4 implementation [24]. Therefore, SRFFT is a promising form of FFT formulation, assuming the objective is to reduce power consumption with pruning.

Table 2-1: Total number of complex multiplications in different FFT structures [25, 37]
N is the length of FFT, L is the number of non-zeros and $p = 2^L$

FFT Algorithms	Complex Multiplication	
Radix-2 FFT	$N/2\times\log_2(N)$	
SRFFT	$N/3\times\log_2(N)-8/9\times(N-1)$	
Markel's FFT pruning	$N/2\times\log_2(L)+N-L$	
Skinner's FFT pruning	$N/2\times\log_2(L)+N-L$	
Transform Decomposition	Based on radix-2	$N/2\times\log_2(P)+L\times(N/P-1)$
	Based on SRFFT	$N/3\times\log_2(P)-8/9\times N\times(1-1/P)+L\times(N/P-1)$
SRFFT pruning	Continuous L	$N/18\times(1/L+6\times\log_2(L)-1)-L+1$ Where $L<N$, L is power of 4
	Arbitrary L	$L/N\times(N/3\times\log_2(N)-8/9\times(N-1))$
Radix-4 FFT	$(3N/8)\times\log_2(N)$ [37]	

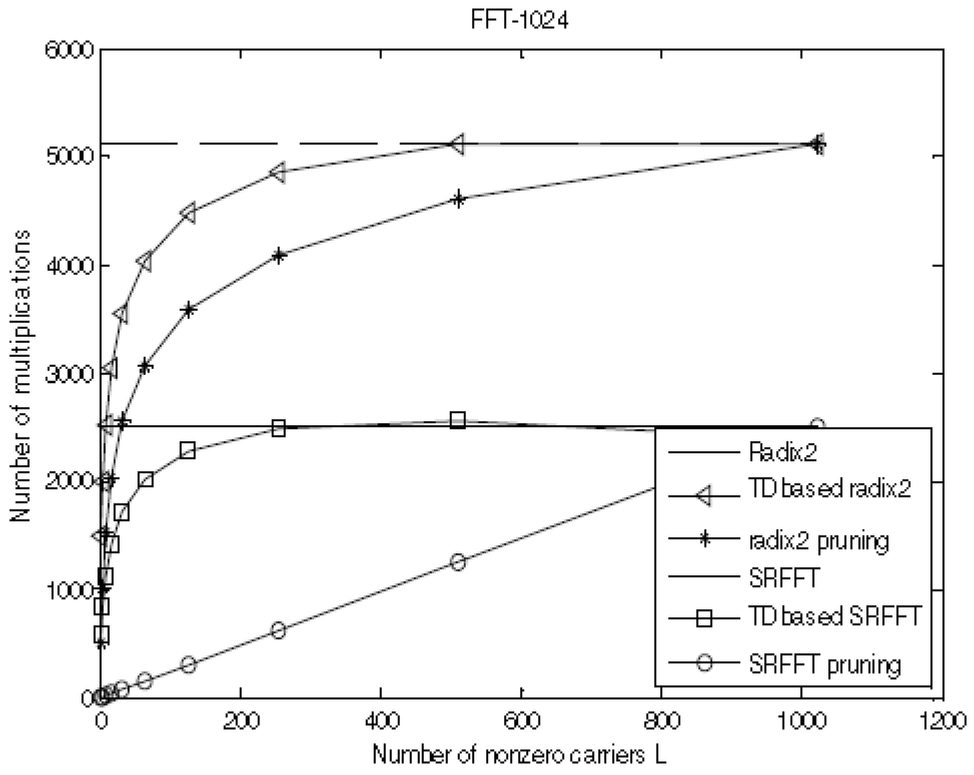


Figure 2-1: Total number of complex multiplications in different FFT architectures [25]

In [17], an FFT core architecture is described. It comprises a pruning engine that helps reducing the power consumption. Their FFT core was implemented on a Field Programmable Gate Array (FPGA) and a saving of 10% in power consumption due to the pruning engine was reported.

In [25], a binary pruning matrix similar to the one proposed in [15] for output pruning is applied to 64-point SRFFT architecture. However, no analysis is provided on power consumption and resource utilization.

In this research work, a new architecture of a SRFFT core is developed that includes a pruning engine with the purpose of reducing power consumption. The combination of SRFFT with pruning has already been introduced. However, it has never been studied and analyzed for possible reduction in power consumption. Therefore, to the best of our knowledge, a power efficient SRFFT core with a pruning engine is developed and analyzed for the first time in this research.

CHAPTER 3 APPLICATION OF FFT IN WIRELESS COMMUNICATION TRANSCEIVERS

In order to highlight the importance of power efficient FFT processor for OFDM-based wireless communication transceiver in the CR platform, a brief review of its fundamental architecture and operational principles is presented in this chapter. Subsequently, the FFT algorithm in its various formats including Radix-2, Radix-4 and SR-FFT is introduced. Finally, a study of pruning techniques is presented.

3.1 OFDM transceivers

As was already mentioned, spectrum sensing is the task of autonomous monitoring of the spectrum resources or sensing the radio spectrum in the local neighbourhood of the radio system. Thereby, spectrum holes or rather specifically those sub-bands of the radio spectrum that are underutilized at a particular instant of time and specific geographic location, may be detected and used efficiently. This may make more bandwidth available and consequently increase data throughput. A common part in many proposed techniques is the feature of real-time spectrum sensing or differently termed, spectrum sniffing. This allows finding available frequency bands and opportunistically use these bands for signals to be communicated in a fair and cost-effective manner. Spectrum sensing may be conducted through digital signal processing (DSP) [1, 2], radio meters [9] or, more recently, analog signal processing (ASP) [10, 11].

Through DSP techniques, the received analog signals are converted to the digital domain using analog-to-digital converters (ADCs) and then further processed to transform them into the frequency domain to obtain their power spectral density (PSD). Fast Fourier Transform (FFT) is one of the most common algorithms for obtaining the PSD of the signals.

Although DSP suffers fundamental drawbacks such as high-cost ADCs, high power consumption and poor performance at high frequencies, emphasis has recently been placed on further developments in this area. One important direction of research has been to reduce the computational burden and hence the DSP power consumption, especially for mobile devices with limited power sources.

ASP addresses some of the aforementioned challenges and is considered to be a reliable technique for signal processing at millimeter-wave frequencies for the near future [10]. However, renewed interest in ASP is recent and it still needs improvements to be a viable solution.

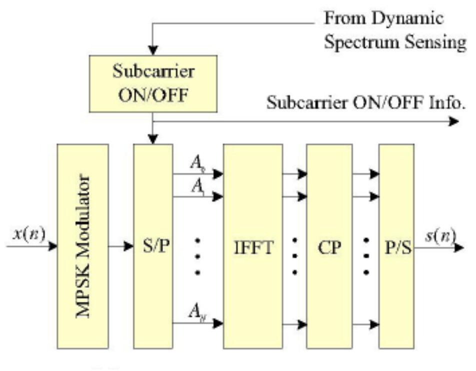
Nevertheless, FFT and IFFT are widely applied through digital signal processing algorithms in many communication transceivers, because, in essence, the modulation or demodulation techniques, applied to them, are based on the Fourier transform. For instance, orthogonal frequency division multiplexing (OFDM), in particular the non-contiguous OFDM (NC-OFDM), is a technology of this kind, through which modulated subcarriers may be distributed non-uniformly so that maximum bandwidth might be used for data communication. Thus, NC-OFDM is a viable transmission technology for cognitive radio transceivers operating in Dynamic Spectrum Access (DSA) networks.

As shown in Figure 3-1, in communication transceivers with OFDM modulation, the FFT and IFFT are fundamental blocks in receiver (Rx) and transmitter (Tx) modules, respectively. Furthermore, symbols are first computed using one of the desired modulation formats, i.e. quadrature amplitude modulation (QAM) or phase shift keying (PSK), and then the modulated data stream is split into N slower data streams using a serial-to-parallel (S/P) converter. Modulation basically happens via the IFFT of these symbols when they are mapped onto several subcarriers. Thus, each OFDM symbol contains all subcarriers, each carrying modulated symbol. In order to prevent inter-symbol interference (ISI), a few fixed subcarriers are located in the beginning of the OFDM symbols using cyclic prefix (CP) blocks. The subcarriers that transmit the modulated symbols are then summed through a parallel to serial (P/S) block and transmitted upon up-conversion to the Radio Frequency (RF) range [35].

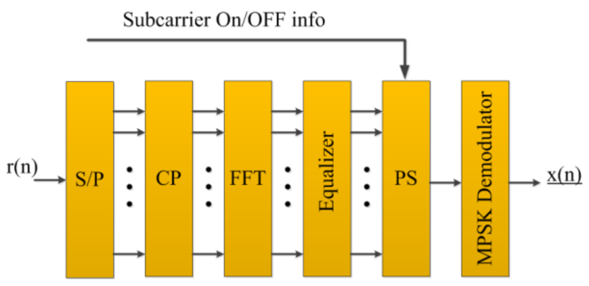
On the other hand, in the Tx module, the incoming RF signal is first down converted to baseband and then sampled and converted to digital signal through analog to digital converters. Subsequently, the OFDM symbols are de-modulated through FFT and the symbols detected using an FFT of the received signal. NC-OFDM is a special format of OFDM modulation in which the subcarriers of the OFDM symbol may be used arbitrarily and in a non-contiguous shape.

Figure 3-1 shows the architecture of the NC-OFDM transceiver along with the spectrum utilization. The architecture remains the same as that of an OFDM transceiver, which is explained above, but the subcarrier utilization would be different. Indeed, some subcarriers may be nullified in the NC-OFDM transmitter in accordance with used or unused spectrum frequency bands. This information is obtained via spectrum sensing block in Tx and is applied when data symbols are mapped on subcarriers. Users in a CR network may determine their operational frequency bands in a frequency agile and non-contiguous manner. Furthermore, in NC-OFDM, all subcarriers do not need to be active as in OFDM system, and indeed active subcarriers are located in unoccupied bands.

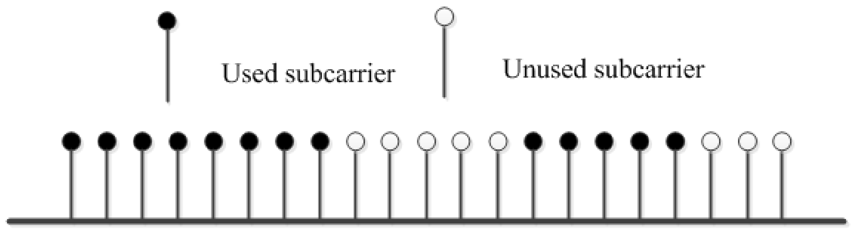
The information about occupied spectrum can be sent over the channel to the Rx. This information about the shape of occupied band may be used in Rx for pruning the FFT algorithm throughout demodulation and hence allow avoiding demodulation of zero subcarriers. Typical non-contiguous spectrum utilization is shown in Figure 3-1 (b).



(a) NC-OFDM Transmitter



(b) NC-OFDM Receiver



(c) Spectrum utilization

Figure 3-1: General NC-OFDM Transceiver Architecture [16]

3.2 FFT formats

3.2.1 Radix-2 FFT

The radix-2 decimation-in-frequency (DIF) and decimation-in-time (DIT) FFT formats are the simplest FFT algorithms that basically compute the discrete Fourier transform (DFT) [30]:

$$x(k)=\sum_{n=0}^{N-1}x(n)e^{-(\frac{i2\pi nk}{N})}=\sum_{n=0}^{N-1}x(n)W_N^{nk} \tag{3.1}$$

where $W_N^K = e^{-(j\frac{2\pi k}{N})}$.

The DIF radix-2 FFT computation is indeed a recursive combination of two distinct partitions of the DFT computation, which can each be computed by shorter-length DFTs of different combinations of input samples. The samples (X (k)) are separated into even-indexed (k = [0, 2, 4,…,N - 2]) and odd-indexed (k = [1,3,5,…,N-1]) outputs.

Considering only even indexed samples, we can reformulate the related part of the algorithm as follows:

$$\begin{aligned}
\mathrm{X}(2r) &= \sum_{n=0}^{N-1}(x(n)\,\mathrm{W}_N^{2rn}) \\
\mathrm{X}(2r) &= \sum_{n=0}^{\frac{N}{2}-1}(x(n)\,\mathrm{W}_N^{2rn})+\sum_{n=0}^{\frac{N}{2}-1}(x(n+\frac{N}{2})W_N^{2r(n+\frac{N}{2})}) \\
\mathrm{X}(2r) &= \sum_{n=0}^{\frac{N}{2}-1}(x(n)\,\mathrm{W}_N^{2rn})+\sum_{n=0}^{\frac{N}{2}-1}(x(n+\frac{N}{2})\,\mathrm{W}_N^{2rn})(1) \\
\mathrm{X}(2r) &= \sum_{n=0}^{\frac{N}{2}-1}((x(n)+x(n+\frac{N}{2}))\,\mathrm{W}_{\frac{N}{2}}^{rn})
\end{aligned} \tag{3.2}$$

which is basically the $\frac{N}{2}$ point DFT of $y(n) = x(n) + x(n+\frac{N}{2})$.

For the odd termed samples we have

$$X(2r+1)=\sum_{n=0}^{N-1}(x(n)\,\mathrm{W}_N^{(2r+1)n})$$
$$X(2r+1)=\sum_{n=0}^{\frac{N}{2}-1}((x(n)+W_N^{\frac{N}{2}}x(n+\frac{N}{2}))W_N^{(2r+1)n}) \qquad (3.3)$$
$$X(2r+1)=\sum_{n=0}^{\frac{N}{2}-1}(((x(n)-x(n+\frac{N}{2}))\,\mathrm{W}_N^{n})\,\mathrm{W}_{\frac{N}{2}}^{rn})$$

which is basically the $\frac{N}{2}$ point DFT of $y(n) = x(n) - x(n+\frac{N}{2})$.

The mathematical simplifications in (3.2) and (3.3) reveal that both the even-indexed and odd-indexed frequency outputs X(k) can each be computed by a length-N/2 DFT. The inputs to these DFTs are sum or subtraction of the first and second halves of the input signal, respectively, where the input to the short DFT producing the odd-indexed frequencies is multiplied by a so-called twiddle factor terms $W_N^K = e^{-(j\frac{2\pi k}{N})}$. This is called decimation in frequency because the frequency samples are computed separately in alternating groups, and a radix-2 algorithm because there are two groups. Figure 3-2 graphically illustrates this form of the DFT computation. This conversion of the full DFT into a series of shorter DFTs with a simple pre-processing step gives the decimation-in-frequency FFT its computational savings.

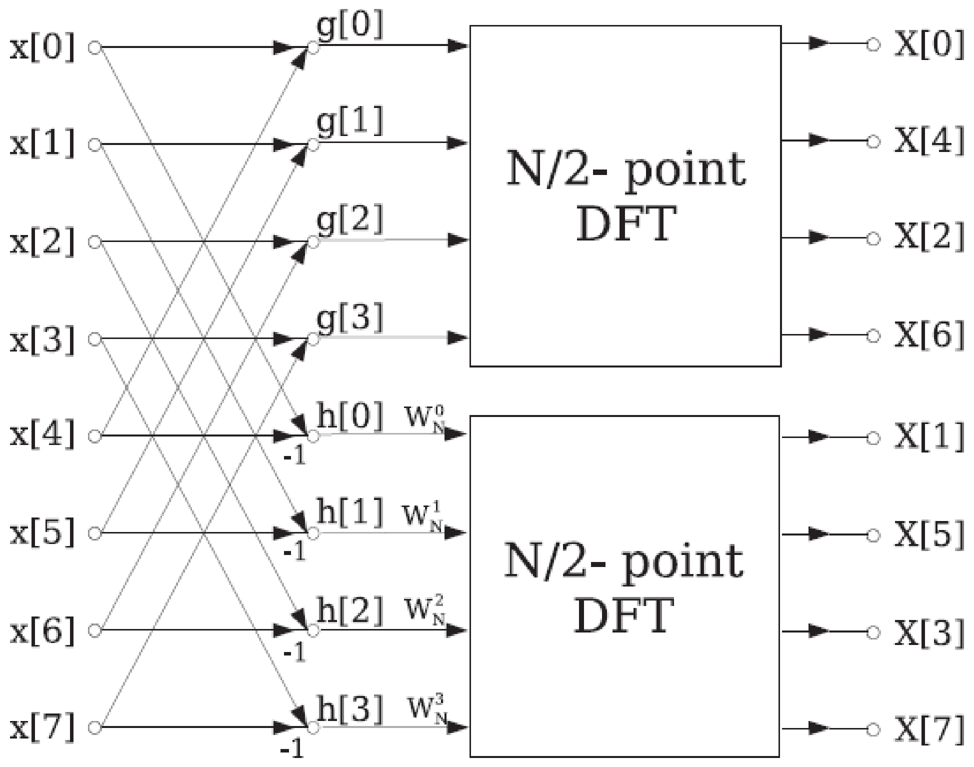


Figure 3-2: Decimation in frequency of a length-N DFT into two length-$N/2$ DFTs preceded by a preprocessing stage [30].

The butterfly operation is illustrated in Figure 3-3, which is basically the main computation block in the FFT. Each butterfly includes a complex addition and subtraction followed by one twiddle-factor multiplication by $W_N^n = e^{-\frac{i2\pi n}{N}}$ on the lower output branch. There are N/2 butterflies per stage. It is worthwhile to note that the length-2 FFT is just one butterfly block where the twiddle factor is 1.

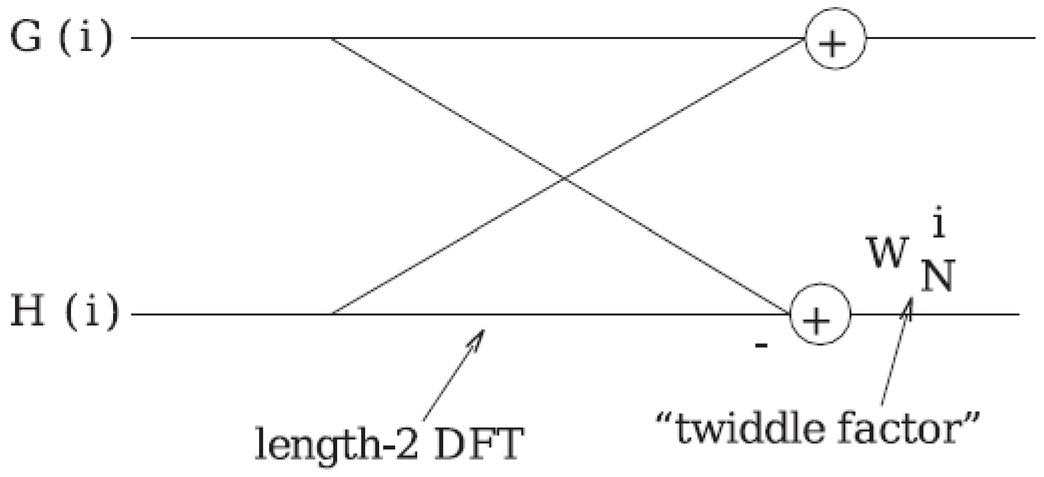


Figure 3-3: The butterfly, a basic DIF computation unit

3.2.2 Radix-4 FFT algorithm

The butterfly of a radix-4 algorithm consists of four inputs and four outputs. The FFT length is 4^M, where M is the number of stages. A stage has half the number of butterflies of a radix-2 FFT stage. The radix-4 DIF FFT divides an N-point discrete Fourier transform (DFT) into four N/4 - point DFTs, then into 16 N/16-point DFTs, and so on. In the radix-2 DIF FFT, the DFT equation is expressed as the sum of two calculations: one for the first half and one for the second half of the input sequence. Similarly, the radix-4 DIF FFT expresses the DFT equation as four summations and then divides it into four equations, each of which computes every fourth output sample. The following equations illustrate radix-4 decimation in frequency [31].

$$X(k)=\sum_{n=0}^{N-1}x(n)\,\mathrm{W}_N^{nk}$$

$$X(k)=\sum_{n=0}^{\frac{N}{4}-1}x(n)W_N^{nk}+\sum_{n=\frac{N}{4}}^{\frac{2N}{4}-1}x(n)W_N^{nk}+\sum_{n=\frac{2N}{4}}^{\frac{3N}{4}-1}x(n)W_N^{nk}+\sum_{n=\frac{3N}{4}}^{N-1}x(n)W_N^{nk}$$

$$X(k)=\sum_{n=0}^{\frac{N}{4}-1}x(n)W_N^{nk}+\sum_{n=0}^{\frac{N}{4}-1}x(n+\frac{N}{4})W_N^{(n+\frac{N}{4})k}+\sum_{n=0}^{\frac{N}{4}-1}x(n+\frac{N}{2})W_N^{(n+\frac{N}{2})k}+\sum_{n=0}^{\frac{N}{4}-1}x(n+\frac{3N}{4})W_N^{(n+\frac{3N}{4})k} \tag{3.4}$$

$$X(k)=\sum_{n=0}^{\frac{N}{4}-1}[x(n)+x(n+\frac{N}{4})\,\mathrm{w}_N^{(\mathrm{n}+\frac{N}{4})}+x(n+\frac{N}{2})\,\mathrm{w}_N^{(\mathrm{n}+\frac{N}{2})K}+x(n+\frac{3N}{4})W_N^{(n+\frac{N}{2})K}]W_N^{nk}$$

The three twiddle factor coefficients can be expressed as follows:

$$W_N^{(\frac{N}{4})K}=[\cos(\frac{\pi}{2})-j\sin(\frac{\pi}{2})]^K=(-j)^k \tag{3.5}$$

$$W_N^{(\frac{N}{2})K}=[\cos(\pi)-j\sin(\pi)]^K=(-1)^k \tag{3.6}$$

$$W_N^{(\frac{N}{4})K}=[\cos(\frac{\pi}{2})-j\sin(\frac{\pi}{2})]^K=(-j)^k \tag{3.7}$$

$$W_N^{(\frac{3N}{4})\mathrm{K}}=[\cos(\frac{3\pi}{2})-\mathrm{j}\sin(\frac{3\pi}{2})]^k=j^k \tag{3.8}$$

Equation (3.4) can thus be expressed as

$$X(k)=\sum_{n=0}^{\frac{N}{4}-1}[\mathrm{x(n)}+(-j)^k\,\mathrm{x(n+}\frac{N}{4})+(-1)^k x(n+\frac{N}{2})+(j)^k x(n+\frac{3N}{4})]W_N^{nk} \tag{3.9}$$

To arrive at a four-point DFT decomposition, let $W_{N/4}^{4} = W_{N/4}$. Equation (3.9) can then be written as four N/4 point DFTs, or

$$
\begin{aligned}
X(4k) &= \sum_{n=0}^{\frac{N}{4}-1}[\mathrm{x(n)} + \mathrm{x(n+}\frac{N}{4}) + x(n+\frac{N}{2}) + x(n+\frac{3N}{4})]W_{\frac{N}{4}}^{nk} \\
X(4k+1) &= \sum_{n=0}^{\frac{N}{4}-1}[\mathrm{x(n)} - j\,\mathrm{x(n+}\frac{N}{4}) - x(n+\frac{N}{2}) + jx(n+\frac{3N}{4})]\mathrm{W}_N^{n}\mathrm{W}_{\frac{N}{4}}^{nk} \\
X(4k+2) &= \sum_{n=0}^{\frac{N}{4}-1}[\mathrm{x(n)} - \mathrm{x(n+}\frac{N}{4}) - x(n+\frac{N}{2}) - x(n+\frac{3N}{4})]\mathrm{W}_N^{2n}\mathrm{W}_{\frac{N}{4}}^{nk} \\
X(4k+3) &= \sum_{n=0}^{\frac{N}{4}-1}[\mathrm{x(n)} + \mathrm{jx(n+}\frac{N}{4}) - x(n+\frac{N}{2}) - jx(n+\frac{3N}{4})]\mathrm{W}_N^{3n}\mathrm{W}_{\frac{N}{4}}^{nk}
\end{aligned}
\qquad (3.10)
$$

For k= 0 to N/4 -1.

$X(4K)$, $X(4K+1)$, $X(4K+2)$ and $X(4K+3)$ are $N/4$ point DFTs. Each of the their $N/4$ points is a sum of four input samples $x(n)$, $x(n+N/4)$, $x(n+N/2)$ and $x(n+3N/4)$, each multiplied by either 1, -1, j or –j. The sum is multiplied by a twiddle factor $(W_N^0, W_N^n, W_N^{2n}, W_N^{3n})$. The four N/4 point DFTs is divided into four $N/16$-point DFT. Then, each of these N/16 point DFT is further divided into four N/64-point DFTs, and so on, until the final decimation produces four-point DFTs. The four-point DFT equation makes up the butterfly calculation of the radix-4 FFT. A radix-4 butterfly is shown graphically in Figure 3-4.

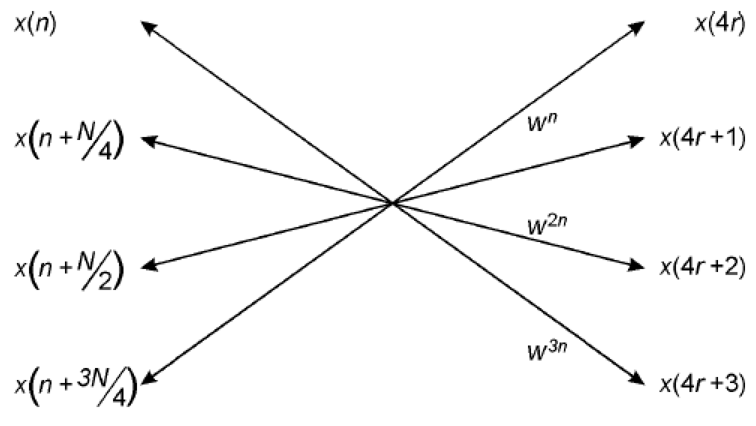


Figure 3-4: Radix4 decimation in frequency FFT butterfly [31]

3.2.3 Split radix FFT (SRFFT)

SRFFT was first introduced by Duhmel *et al* [27] and further developed by the same authors who presented an implementation [28]. The algorithm originates basically from the radix-2 algorithm diagram which is transformed quite straightforwardly into a radix-4 algorithm by changing the exponents of the twiddle factors. When doing so, it can be observed that a radix-4 is best for the odd terms of the DFT, while radix-2 is best for the even terms of the DFT in each stage of the diagram. Therefore, restricting the above-mentioned transformation locally to only the lower part of the diagram might improve the algorithm. This leads to a decomposition of the DFT into

$$
\begin{aligned}
x(2k) &= \sum_{n=0}^{N/2-1} [x(n) + x(n+\frac{N}{2})] W_N^{2nk} \\
x(4k+3) &= \sum_{n=0}^{\frac{N}{4}-1} \{[x(n) - x(n+\frac{N}{2})] + j[x(n+\frac{N}{4}) - x(n+\frac{3N}{4})]\} W_N^{3n} W_N^{4nk} \\
x(4k+1) &= \sum_{n=0}^{\frac{N}{4}-1} \{[x(n) - x(n+\frac{N}{2})] - j[x(n+\frac{N}{4}) - x(n+\frac{3N}{4})]\} W_N^{n} W_N^{4nk}
\end{aligned}
\tag{3.11}
$$

We can see that the fundamental N-point DFT equation is decomposed into one length-N/2 DFT and two length-N/4 DFTs with twiddle factors.

The first stage of a split-radix decimation in frequency decomposition then replaces a DFT of length N by one DFT of length N/2 and two DFTs of length N/4 at the cost of [(N/2) - 4] general complex multiplications (3 real multiplications + 3 additions), and 2 multiplications by the eighth root of unity (2 real multiplications + 2 additions). The N-point DFT is then obtained by successive use of such decompositions up to the last stage, where some usual radix-2 butterflies (without twiddle factors) are needed.

The implementation of the algorithm is considerably simplified by noting that at each stage i of the algorithm, the butterflies are always applied in a repetitive manner to blocks of length $N/2^i$. Then, only one test is needed to decide whether the butterflies are applied to the block or not (2^i tests are needed at each stage i).

Some specific features SRFFT are listed and discussed in [27]. Within this thesis framework, the most interesting advantage of split radix FFT in comparison to other existing FFT techniques is the lowest number of required multiplications and additions.

3.3 Pruning

The primary pruning technique, proposed by [13], which is already introduced in Chapter 2 with the literature review, is briefly explained hereafter to provide insight on its principles.

From Figure 3-5, it can be seen that there are 4 stages, 8 butterflies per stage and 3 operations per butterfly. The structure of a butterfly computation is depicted in Figure 3-6. In general, within the diagram of an M-stage FFT, N/2 butterflies are present in each stage and each butterfly includes 3 complex arithmetic operations; i.e. one addition, one subtraction, and one multiplication.

Figure 3-7 shows a 16-point FFT with the DIF structure where the input contains only two non-zeros. Comparing the FFT with the same structure but in a un-pruned implementation, one can observe that only the computational operations originating from these two non-zero inputs that yield non-zero values in every stage were computed.

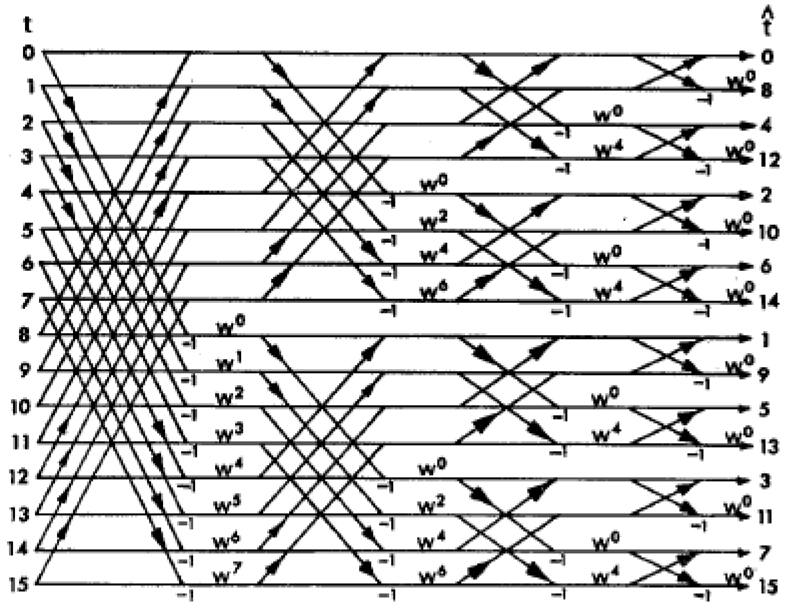


Figure 3-5:FFT computed according to the DIF structure (no pruning) [13]

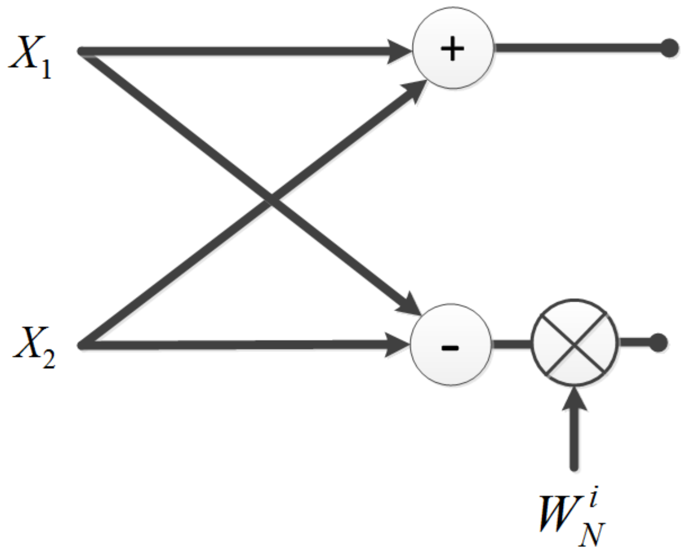


Figure 3-6: Representation of a single “butterfly”; the twiddle factors are defined by $W^k = e(k/16) = \exp(-j2\pi k/16)$.

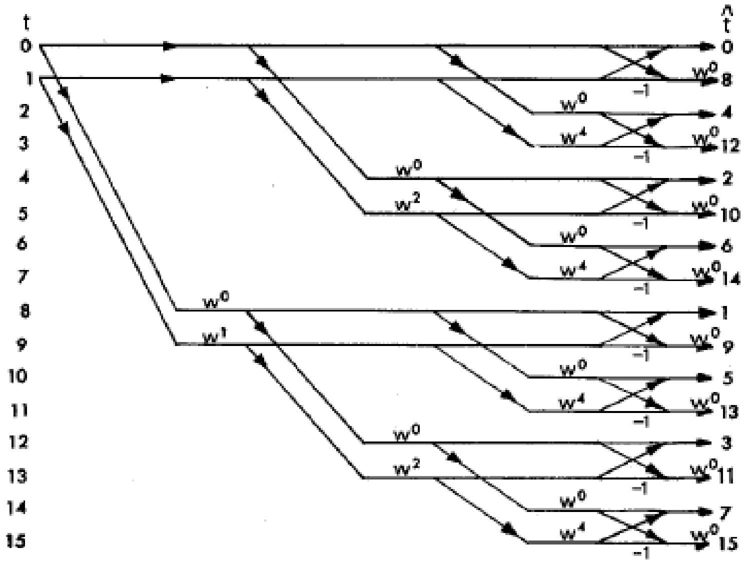


Figure 3-7: FFT pruning scheme proposed by Markel [13]

Figure 3-7 shows pruning in the FFT through which operations that do not contribute to the outputs are eliminated. Hence, only the data paths that affect the result at the output are kept. In fact, it shows graphically a pruned 16-point FFT where only two nonzero input points are present and all arithmetic operations originating from or yielding zero values are omitted. In this diagram, we can see that pruning is possible in only three stages and pruning in the last stage is not feasible.

Since there are 2^L nonzero data points and 2^{M-L} zero value points, L=1 corresponds to the number of stages in which nothing can be pruned and M-L=3 corresponds to the number of stages in which pruning can be applied. More specifically, L is equal to the number of stages in which no pruning is allowable and thus M-L is equal to the number of stages in which pruning can be employed [13]. The implementation procedure uses careful inner-loop nesting. In essence, the algorithm is nothing more than a clever nesting of loops to provide proper indexing for the butterfly calculations.

3.4 The efficient FFT algorithm

Among the reported FFT pruning algorithms, the one proposed by Ranjbanshi *et al* [16] was proven to have superior performance in terms of the total processing time due to a lower number of arithmetic operations when a large number of data points at the input are zero. Moreover, the proposed algorithm works for any pattern of zeroes at the input. Such algorithm can be used in the FFT or IFFT blocks in the NC-OFDM transceivers where a significant number of subcarriers remain null and the spectrum is sparsely utilized. Indeed, for highly sparse spectrum, the number of zero-valued inputs is quite large, and therefore considerable processing time can be saved through pruning the FFT algorithm. Therefore, the performance of this algorithm is studied as a possible candidate of further development for power saving purposes. The principles of this pruning algorithm are briefly explained next.

Figure 3-8 shows the developed data flow FFT pruning architecture proposed by Ranjbanshi.

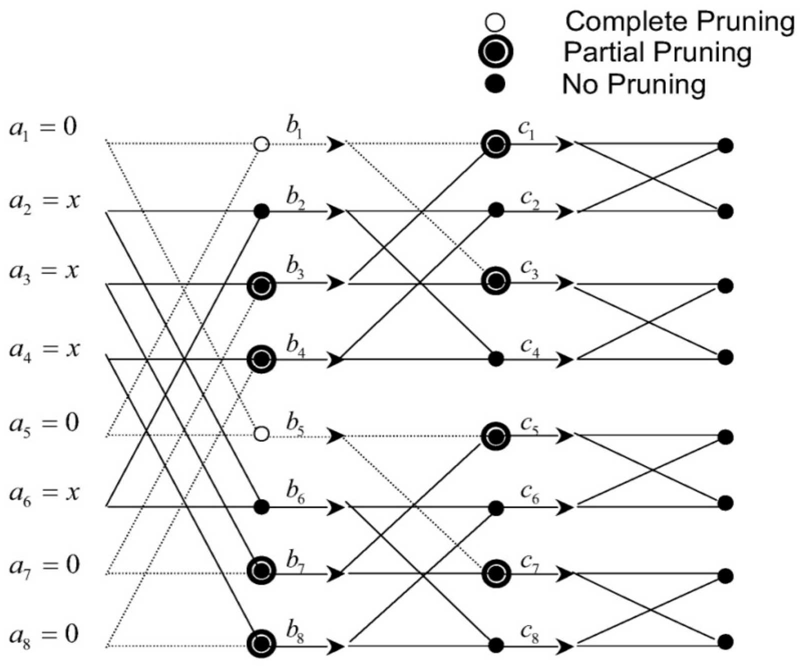


Figure 3-8: The data flow FFT pruning architecture [16]

In this 8-point FFT diagram, inputs with zero and non-zero value are denoted by 0 and x, respectively. In computing the butterflies, if both inputs are zero, they will be completely pruned. If one of the inputs is non-zero the corresponding butterfly is partially pruned while the whole computation in the corresponding node would be carried out if none of the data inputs were null.

3.5 Software implementation and analysis of FFT and pruned FFT

In order to count and monitor the arithmetic operations throughout several FFT algorithms, we have implemented them with the C programming language. As a reference point, the ordinary FFT proposed by Cooley and Tukey [29] as well as the FFT pruning algorithm proposed by Ranjbanshi [16] were analyzed and implemented in C using Eclipse Helios [36].

For code validation, a combination of single frequency sinusoidal signals, which are sampled with an appropriate sampling frequency, is used as input signals. The sampled input test signal is a single frequency signal oscillating at frequency f_1 and denoted by:

$$x[n] = Acos(2\pi f_1(n \times T_s)) \tag{3.12}$$

Where T_s is the sampling period and is the inverse of sampling frequency, i.e. $T_s = 1/fs$.

At the output of the FFT with $x[n]$ input data, a peak appears at the index K_1 which corresponds to the frequency content f_1 of the input signal, sampled at f_s, through

$$f_1 = \frac{K_1}{N} \times f_s \tag{3.13}$$

The sampling frequency should be selected in a way that respects the Nyquist sampling period or more specifically speaking it should be at least twice as large as the largest frequency in the spectrum of the input signal.

In order to validate the correctness of the programmed algorithms, a typical signal composed of three sinusoidal functions at three distinct frequencies is used as in:

$$X[n] = \cos(2\pi f_1 n T_s) + 6\cos(2\pi f_2 n T_s) + \ 2\cos(2\pi f_3 n T_s) \tag{3.14}$$

where $f_1 = 500$, $f_2 = 700$, $f_3 = 300$. $X[n]$, which has a sampling frequency of 4 KHz, is plotted in Figure 3-9.

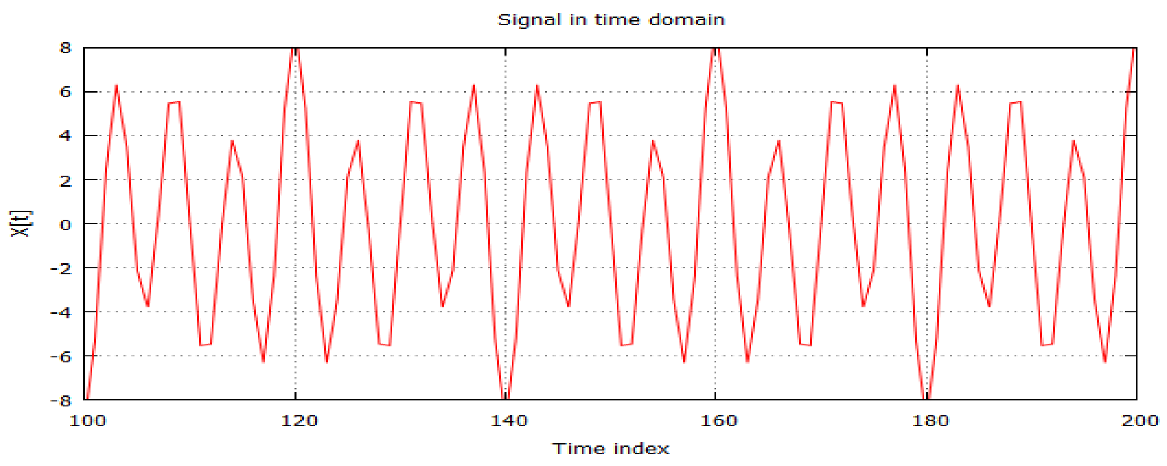


Figure 3-9: Input signal

The output signal, which is in the frequency domain, is plotted in Figure 3-10. This frequency domain plot shows the existence of three peaks that correspond to three frequency components.

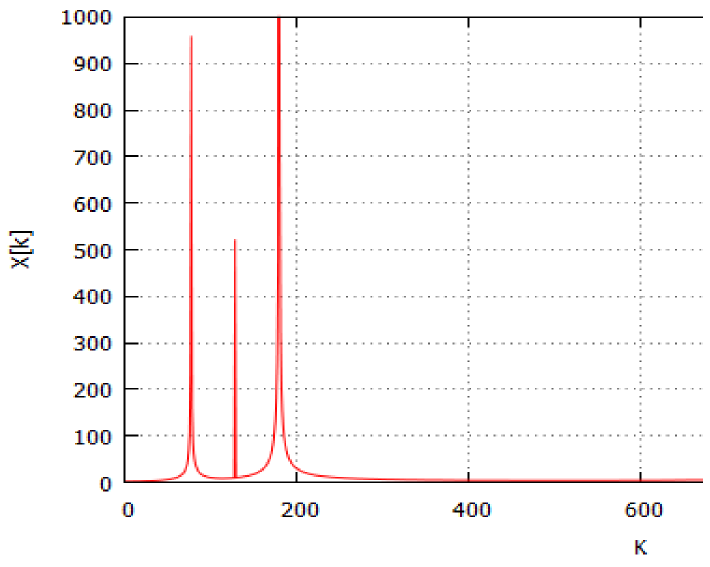


Figure 3-10: Result computed with the implemented software FFT

In another test, the IFFT pruning code is assessed using a data input that is the frequency components or the subcarriers of the transmitting signal of an OFDM transmitter. This set of subcarriers is selected so that three quarter of them are null, i.e. the sparseness factor is 75%. As shown in Figure 3-11, the subcarriers within the range of 256 to 511 are active, and the rest are zeros.

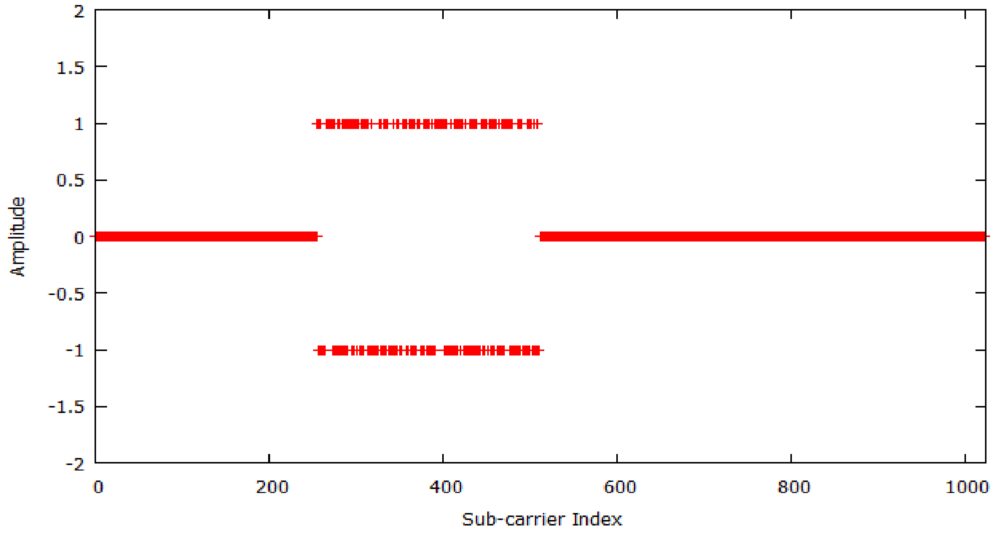


Figure 3-11: The input signal used to test the IFFT; 25% of the carriers are non-zero

In order to compare the total number of multiplications required in the ordinary FFT (as originally proposed by Cooley and Tukey) with those required by the SRFFT, the number of multiplications and summations with various implementations of a 256-point FFT was profiled and the results are shown in Figure 3-12. One can observe that the number of multiplications and summations with the SRFFT is much less than that required with the Cooley-Tukey FFT when not using any pruning technique.

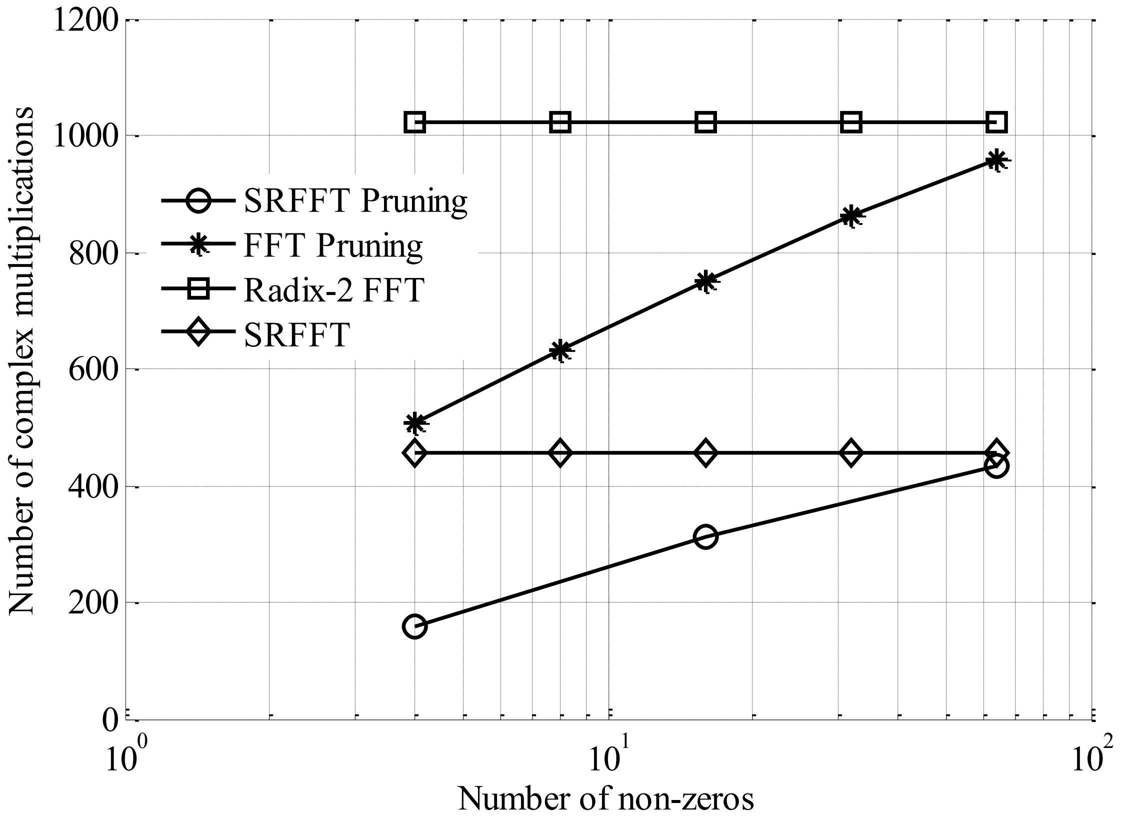


Figure 3-12: Comparison of the number of operations required, i.e. summations and multiplications, for a 256-point FFT computed using different algorithms

As one may anticipate, the SRFFT with pruning requires fewer operations than others and hence it can complete the transform process within a shorter time. Obviously, if this algorithm was implemented in FPGA, it could consume much less power. Thus, SRFFT pruning, which theoretically completes the frequency transform with minimum arithmetic operations, was selected for hardware implementation. First, an implementation of the SRFFT processor targeting an FPGA was developed and then improved for minimizing the overall power consumption by adding the pruning engine. The architecture of this processor is explained in further details in the next chapter.

CHAPTER 4 STUDY AND DEVELOPMENT OF A POWER EFFICIENT SRFFT PROCESSOR

SRFFT, which was proven to have less computation burden in comparison to other FFT algorithms, was selected for implementation in combination with the pruning technique. Therefore, a SRFFT processor is first developed and then a pruning engine is added with the purpose of increasing savings. In this chapter, the SRFFT processor as well as the pruning engine are explained in detail along with presenting the simulation results as a proof of their correct functionality. At the end, the performance of the processor with or without the pruning engine is assessed in terms of power consumption.

4.1 Implementation of SRFFT processor

4.1.1 Architecture of the proposed SRFFT processor

Figure 4-1 shows the architecture of the generic SRFFT processor proposed in this thesis. This processor comprises a finite state machine (FSM) as a control unit, a butterfly engine, two pairs of RAMs with single address ports for real and imaginary components of the signal, a ROM for storing the twiddle factors and several interconnect units. The functions of each block are explained further in the following.

4.1.2 L-shaped butterflies

The main difference between FFT and SRFFT is the shape of butterflies and the number of butterflies that are calculated in each stage. In Radix-2 FFT in VHDL, only one butterfly with 2 inputs is calculated while in SRFFT, 2 butterflies with 4 inputs and 1 butterfly with 2 inputs at the following stage are calculated in each round. Secondly, in the first two butterflies, we do not have multiplications with twiddle factors and thirdly, twiddle factors are arranged differently. The structure of the resulting L-shaped butterfly is sketched in Figure 4-2.

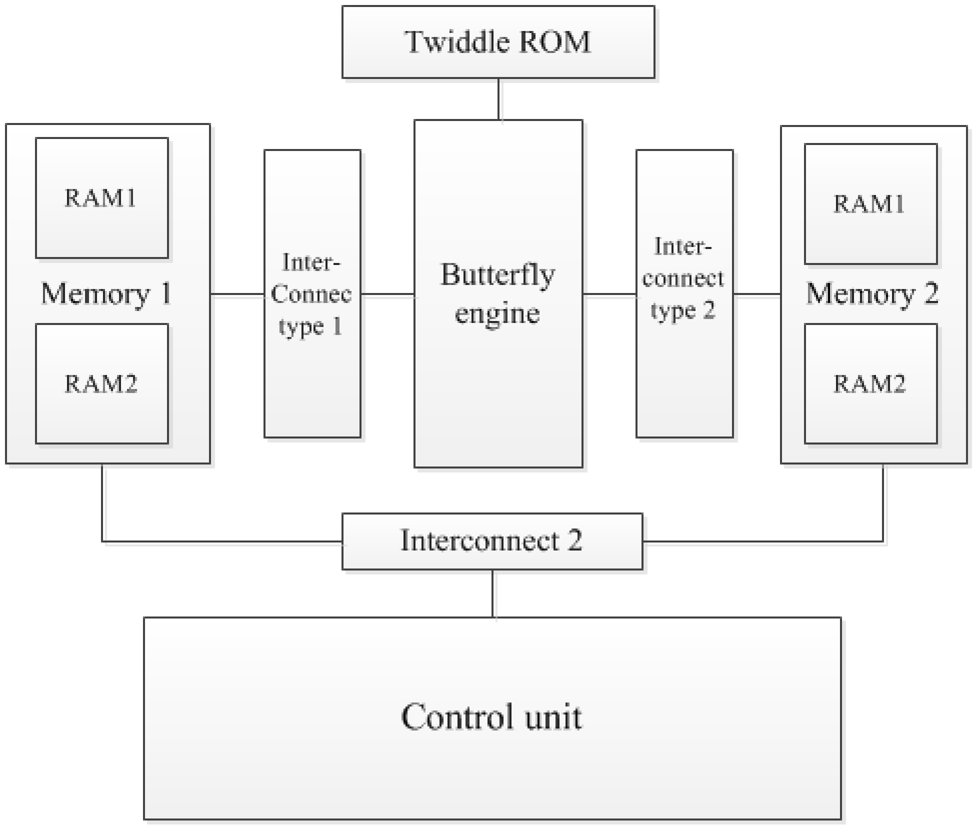


Figure 4-1: Architecture of the proposed SRFFT

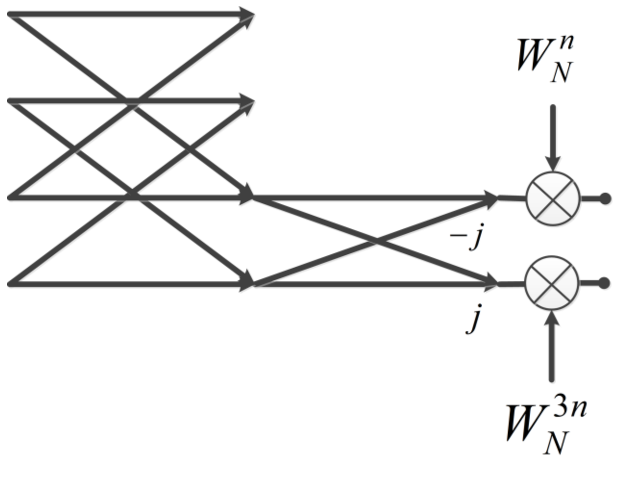


Figure 4-2: L shape butterfly

Furthermore, in SRFFT, a combination of Radix-2 and Radix-4 index mapping is used for even and odd indexed data in each stage, respectively. This decomposition yields arithmetic computations grouped in L-shape butterflies. Indeed, unlike the ordinary FFT in which computations are carried out stage by stage, those in SRFFT involve two adjacent stages. This adds to the complexity of control-signal generation and design. Figure 4-2 shows a typical L-shaped butterfly needed in the SRFFT. This butterfly can be decomposed into three distinct blocks, and in our proposed architecture, each of these blocks carry out a portion of the computations as highlighted in Figure 4-3.

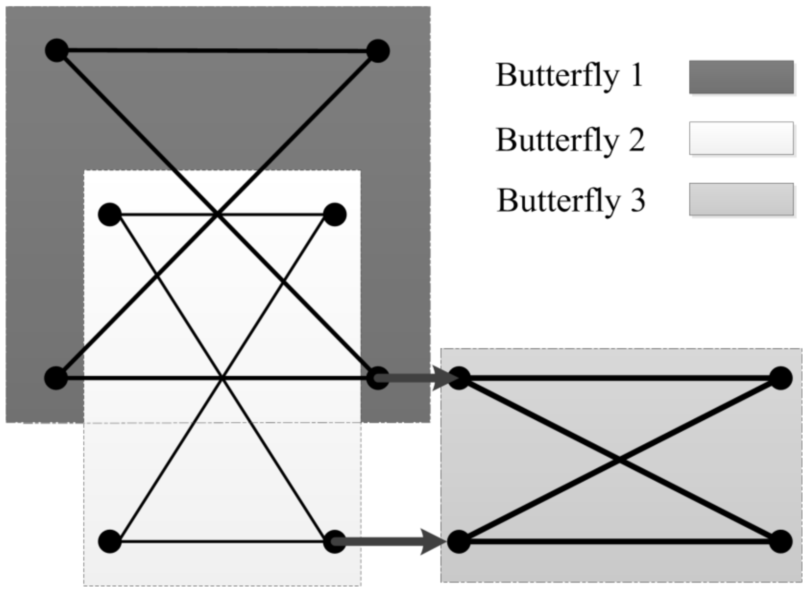


Figure 4-3: Butterfly engine

4.1.3 Control unit

Figure 4-4 shows the diagram of a 16-point SRFFT and the forward flow of processing from the input stage until the output stage. Each L-shape section may contain at least one L-shape butterfly. The computation diagram of an N-point SRFFT contains $\log_2 N$-1 L-shaped stages and one ordinary or non-L-shaped stage in the last one. In this diagram, one can see three L-shaped stages, which are numbered and may contain several L-shaped butterflies.

In the proposed architecture of the SRFFT processor, a control unit, which is a finite state machine, is used to accomplish several main tasks, including 1) address generation, 2) control of data and address paths, and 3) assignment of read and write operations for the RAMs at the appropriate time.

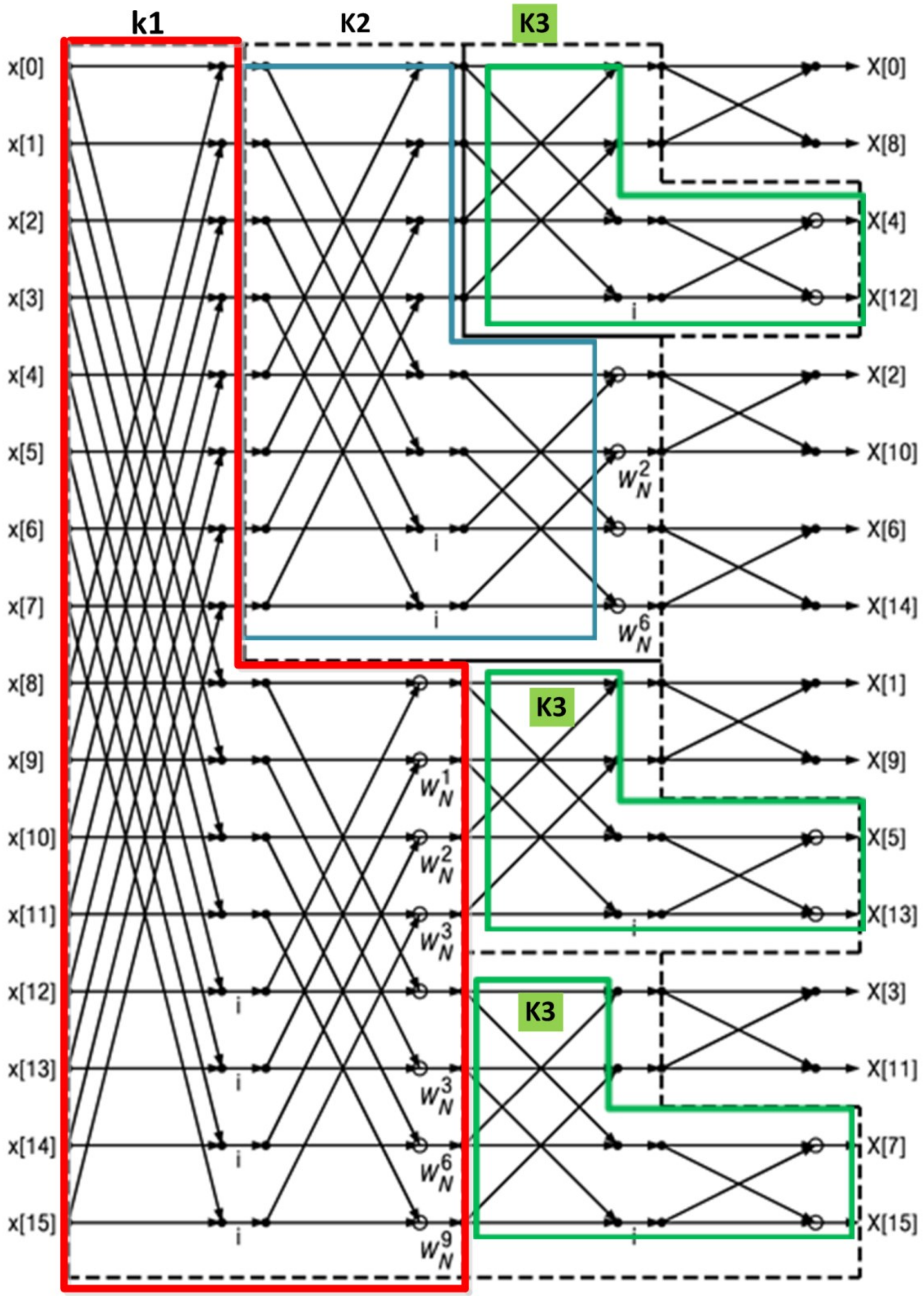


Figure 4-4: 16-point SRFFT and processing progress [33]

Figure 4-5 shows the simulation results of the initial architecture of the SRFFT processor. The addresses generated for calling the data points from RAM for an 8 point SRFFT can be observed on the signals i0, i1, i2 and i3. Also, the signal "Kport" shows the forward progress in L-shaped stages. There are two L-shape butterflies in the first stage, i.e. the first two, while there is only 1 L-shaped butterfly in the second stage. Furthermore, when the signal varies from 1 to 3. Two L-shape butterflies with index values 1, 3, 5, 7 and 2, 4, 6, 8 take part in the first stage when "Kport" is 1. In the second stage, when "Kport" is 2, one L-shape butterfly operates with index values 1,2,3,4, and finally in the last stage, two ordinary butterflies with index values 5, 6 and 7, 8 are carried out. In addition, address management to produce twiddle factors for 16 point SRFFT are shown in Figure 4-6. For each L-shape butterfly, two complex twiddle factors are sent to the multipliers in the butterfly units. Real and imaginary components with 16-bit-long twiddle factors are called "Wr1", "Wr2", "Wi1" and "Wi2" respectively.

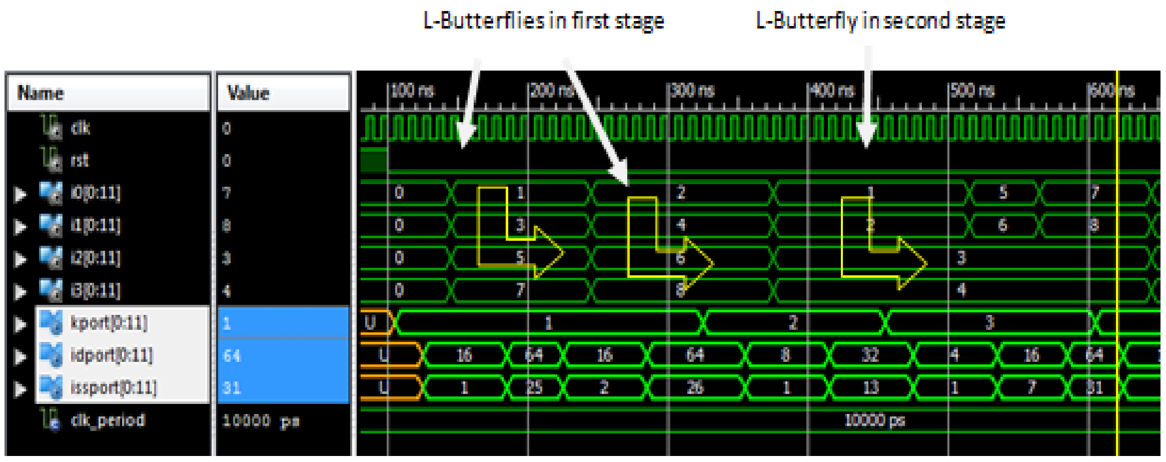


Figure 4-5: Addresses generated while computing an 8 point SRFFT

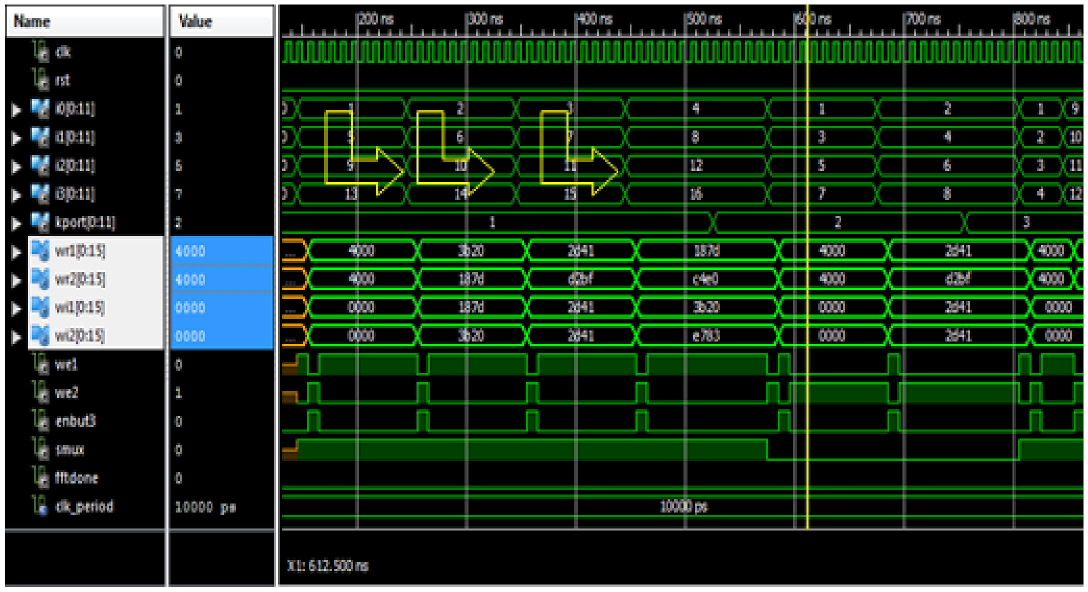


Figure 4-6: Addresses generated while processing a 16 point SRFFT

4.1.4 RAMs and interconnects

The SRFFT processor comprises two pairs of RAMs, which contain 16-bit words, taken from two memory blocks. They allow reading and writing the signal data in each stage. With the proposed butterfly engine, the conventional “ping-pong” style swapping between the RAMs for writing and reading the signals in each stage should not be employed in the usual manner, because all four outputs of the butterfly engine do not pertain to a single stage. For instance, the output of butterfly-3 in Figure 4-3 belongs to the second stage and in the proposed architecture, they must be rewritten in the same RAM from which the four input data of the butterfly engine were called.

The L-shape butterfly requires four data points to appear at its input ports simultaneously. Therefore, four addresses should be generated in the control unit for reading four data words from memory. However, the RAMs that are employed in the memory units possess only one address port, which makes the concurrent reading of four data words unfeasible. The use of single port RAMs is done on purpose, which allows avoiding complex logic and making use of block RAMs in the FPGA. Hence, the data words should be read one by one and then rearranged so that they appear at the butterfly engine all at once. It should be mentioned that use of RAMs with four address and input-output ports would of course be a better solution if they existed as block RAMs in FPGAs on which the processor is to be implemented.

Figure 4-7 shows the address and data word management at the input and output of a memory. In the control unit, four address words are generated and then pipelined to appear at the address port of the RAM upon being multiplexed onto a single path. Then, the data words at the output of the RAM are de-multiplexed onto four separate signal paths and then pipelined to appear in parallel at the butterfly blocks. A similar structure is also used for writing the result of computations (by the butterfly engine) back into the memory. Thereby, single-address port memory blocks are used in order to reduce the complexity and area usage.

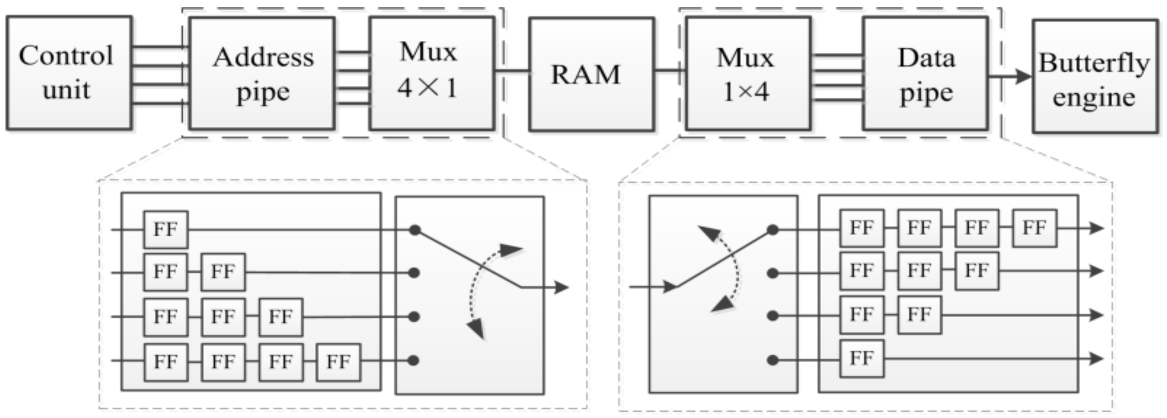


Figure 4-7: The architecture of pipeline SRFFT

As the data would appear at the output of the RAM with one clock cycle delay, these control signals should last for 2 clock periods. The data that goes through the block "Datapipe" will appear at the input of each butterfly at the same time.

Figure 4-8 shows the behaviour of interconnect control signals, which control the parallel-to-serial or serial-to-parallel units. These control signals have four states and therefore accept values 0, 1, 2 and 3.

Active low write-enable (we) and active-high Ram-Enable (en) signals handle read/write functions of the RAMs. There is one of each signal for each RAM (i.e. we1, en1, we2 and en2). Writing takes place before changing the addresses while the reading happens four clock-cycles after changes in addresses at the output of the FSM. The signals I0, I1, I2 and I3 are connected to the address pins of the control unit. More specifically, "Spipectrl" and "Spipedata" vary from 0 to 3 in order to map the appropriate signals at the address port of the RAM. The signal "Spipedata" helps mapping the data at the output of the RAM to the input ports of the butterfly. Both signals change simultaneously when the relevant enable signal (en) and write/read signal (wr) at any specific stage are active. The signal "SpipeResctrl" helps mapping the computation results onto the input port of the RAM where the computation results are to be written. The signal "Spipectrl" remaps the addresses again onto the address port.

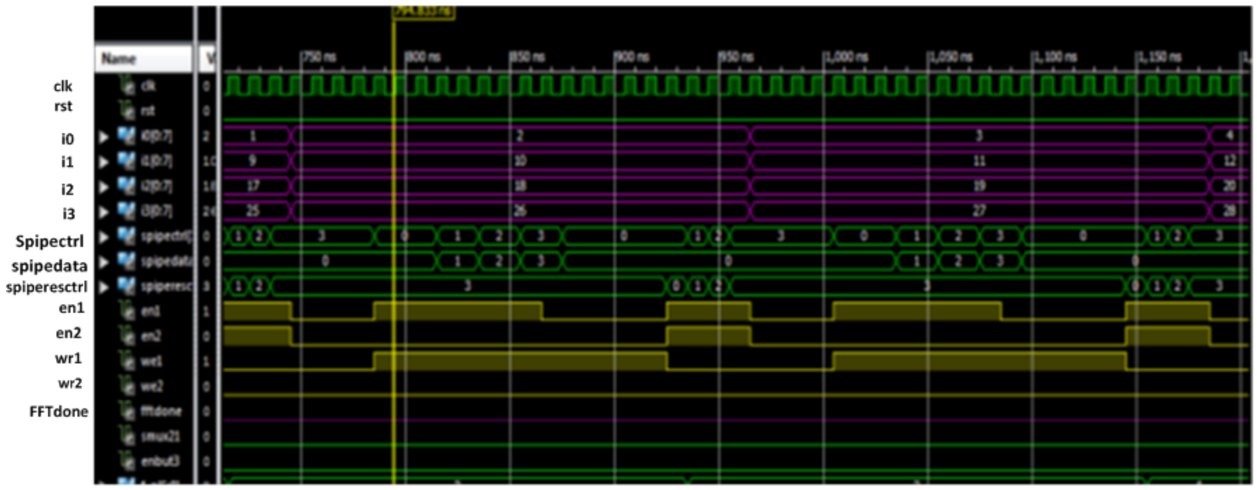


Figure 4-8: Address and control signals at the output of FSM block

Figure 4-9 shows a snapshot of a simulation of the SRFFT when 4 data-points are read out of the RAM and then sorted so that they appear all in parallel at the input ports of butterfly engine, simultaneously. The four address signals addi0, addi1, addi2 and addi3 are generated at the same time by the control unit. Since the addresses appear at the port one by one, we had to keep it active as long as the four addresses are passing. The signal "adi" is connected to the address ports of both RAMS. We can see that after 4 clock cycles from the time of generation of new set of addresses, they appear on "adi" consecutively. The control signal of the multiplexer that switches the addresses on "adi" is named as "SAddCtrl", which varies from 0 to 3 and, each time, it maps one address on the data path connected to the address port of the RAMs.

At the time the first address appears at the address port of the RAMs, one pair of signals from the RAM-enable signals "Sen1", "Sen2" and the write-enable signals "Swe1" and "Swe2" are activated simultaneously, so that the data are read out of the desired RAM. Note that the reading and writing control signal are active-high for both RAMs.

As already mentioned, the data should be located at the input ports of the butterfly blocks at the same time, and this is actually made through de-multiplexing. Indeed, the control signal of “SDataCtrl” switches the data at the output of the RAMs and sorts them at four data paths in parallel.

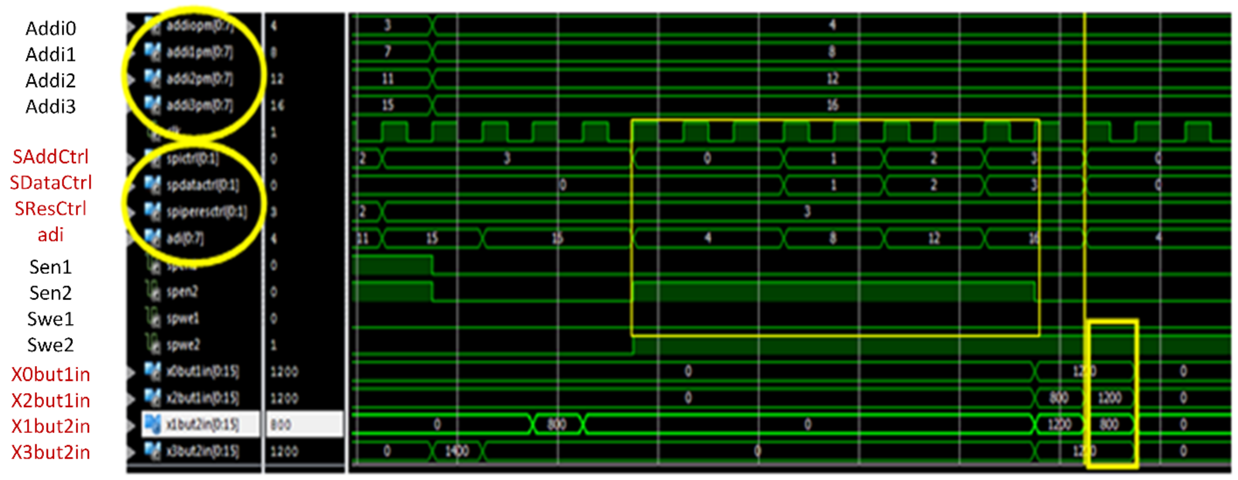


Figure 4-9: Reading 4 data points from RAM and sorting them at the input of butterfly engine

Figure 4-10 shows the procedure through which the results of the butterfly engine are written to RAM. The butterfly blocks compute the results at different clock ticks, which makes data management complicated. This problem is resolved by appropriate pipelining at their outputs upon computations. Thereby, the computation results appear at the output port of the butterfly engine, i.e. “Outbut1x0”, “Outbut2x1”, “Outbut3x2”, “Outbut3x3”, consecutively. Then, they are sorted in series on the data path of the input port of the RAM, i.e. “XinRam1” and “XinRam2”, where the computation results should be stored. In fact, the control signal of “SResCtrl” which controls a de-multiplexer changes from 0 to 3 and thereby the computation results from butterfly engine are sorted in series on the input data port of both RAMs. In Figure 4-10, the computation results at the output of butterfly engine are highlighted with an inclined ellipse. It should be mentioned that the addresses are regenerated and then synchronized with the computation results when they arrive at the input data port of the RAM. These changes are synchronized with the changes of the regenerated addresses on the input address port. During this data writing time, the write-enable signals are deactivated, while both RAM-enable signals, namely “Sen1” and “Sen2”, are activated. Thus, the computation results are written in both RAMs.

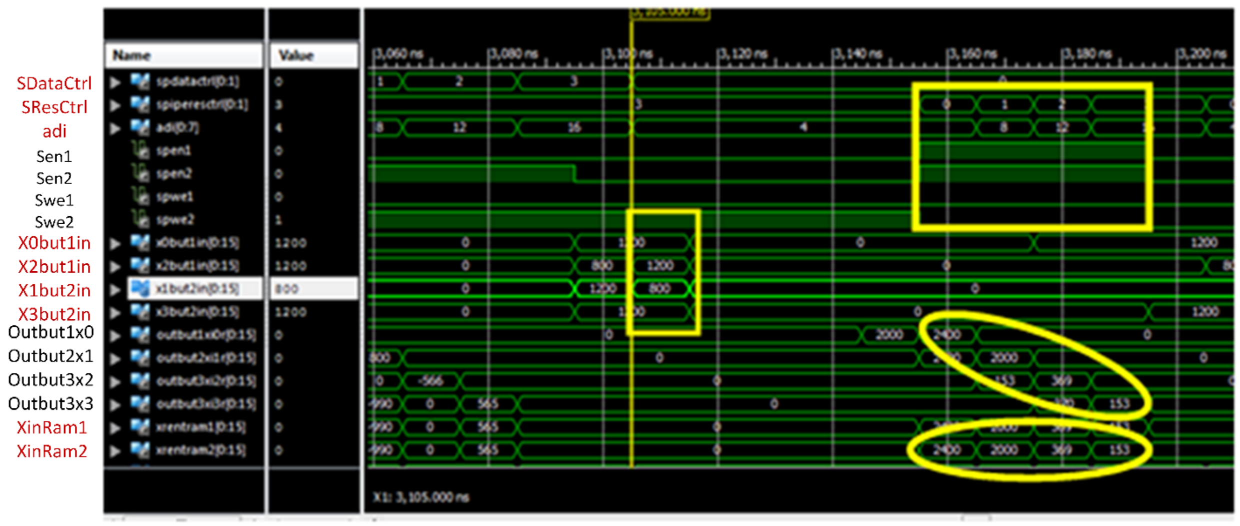


Figure 4-10: Writing the results into RAMs

4.1.5 Twiddle factor ROM

In all FFT architectures, multiplications take part at some fixed points with external parameters called twiddle factors. Depending on the size of the FFT or SRFFT, the number of twiddle factors and their value at each stage may vary. Usually, two different techniques are applied for generating and using the twiddle factors in FFT cores, i.e. twiddle factor bank or computing them during the FFT computations using a complex constant multiplication scheme [12] .

In order to reduce the power consumption, the first approach is used in our SRFFT core. Therefore, a bank of twiddle factors is built up. That bank can be used for generic N-point SRFFT covering from 8 to 1024 points.

Twiddle factors are complex numbers with a magnitude less or equal than unity. The size of the twiddle factor words chosen is 16 bits where the two most significant bits are assigned to sign and non-fractional value and the other 14 bits are assigned to fractional values. Figure 4-11 shows the bit configuration of the twiddle factors.

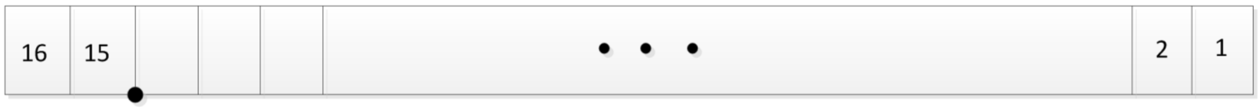


Figure 4-11: Twiddle factor binary word configuration

Table 4-1 shows examples of twiddle factors converted to binary and hexadecimal numbers.

Table 4-1: Examples of bit configuration of twiddle factor

Decimal	Binary	Hexadecimal
0.707	00.10110101000001	X"2D41"
1	01.00000000000000	X"4000"
-1	11.00000000000000	X"C000"
-0.707	11.01001010111111	X"D2BF"

4.2 Butterfly engine structure

The arithmetic operations within the butterfly 1 and 2 in the butterfly engine shown in Figure 4-12 can be expressed by the following equations, given the four inputs of D1, D2, D3 and D4:

$$D1 = X(i0) + jY(i0) \tag{4.1}$$

$$D2 = X(i1) + jY(i1) \tag{4.2}$$

$$D3 = X(i2) + jY(i2) \tag{4.3}$$

$$D4 = X(i3) + jY(i3) \tag{4.4}$$

In butterfly 1:

$$X(I0) = X(I0) + X(I2) \tag{4.5}$$

$$Y(I0) = Y(I0) + Y(I2) \quad \text{(Imaginary)} \tag{4.6}$$

$$R1 = X(I0) - X(I2); \tag{4.7}$$

$$S1 = Y(I0) - Y(I2) \quad \text{(Imaginary)} \tag{4.8}$$

In butterfly 2:

$$X(I1) = X(I1) + X(I3) \tag{4.9}$$

$$Y(I1) = Y(I1) + Y(I3) \quad \text{(Imaginary)} \tag{4.10}$$

$$R2 = X(I1) - X(I3) \tag{4.11}$$

$$S2 = Y(I1) - Y(I3) \quad \text{(Imaginary)} \tag{4.12}$$

These equations are implemented with the structure shown in Figure 4-12. The architecture of both butterflies 1 and 2 are identical.

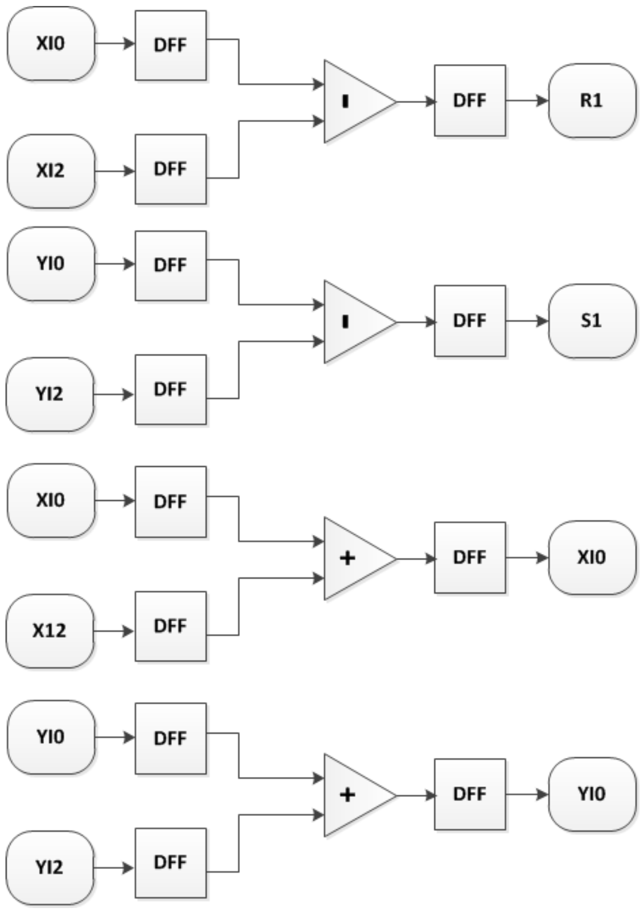


Figure 4-12: Pipelined butterfly 1 and 2

The results of the first and second butterflies are used as input data for further processing within butterfly 3 through the following equations.

$$\mathrm{S3} = \mathrm{R1} - \mathrm{S2} \tag{4.13}$$

(Real part at Index=I3), which includes the operation of "multiplication by *j*"

$$S2 = R2 - S1 \tag{4.14}$$

- (Imaginary part) at Index=I2)

$$\mathrm{R2} = \mathrm{R2} + \mathrm{S1} \tag{4.15}$$

(Imaginary part at Index=I3)

$$\mathrm{R3} = \mathrm{R1} + \mathrm{S2} \tag{4.16}$$

(Real part at Index=I2)

The implementation architecture of this butterfly is shown in Figure 4-13.

In the flow graph of the SRFFT, as it is shown in Figure 4-4 for length of 16 points, some frequency bins in the final stage are not part of the L-shape butterflies and they must be computed using conventional butterfly engines typically found in ordinary FFT hardware implementation. Therefore, the L-shape butterflies are stalled through gating when the computation reaches the last stage of the flow graph of our proposed SRFFT core. This allows reducing power consumption. These changes in the data path are depicted in Figure 4-14 where the final configuration of all butterfly blocks is depicted. We can see that the data flow throughout the butterfly blocks are managed using multiplexers and the computations at the final stage within the ordinary configuration of butterflies are carried out in the block "Butt Last".

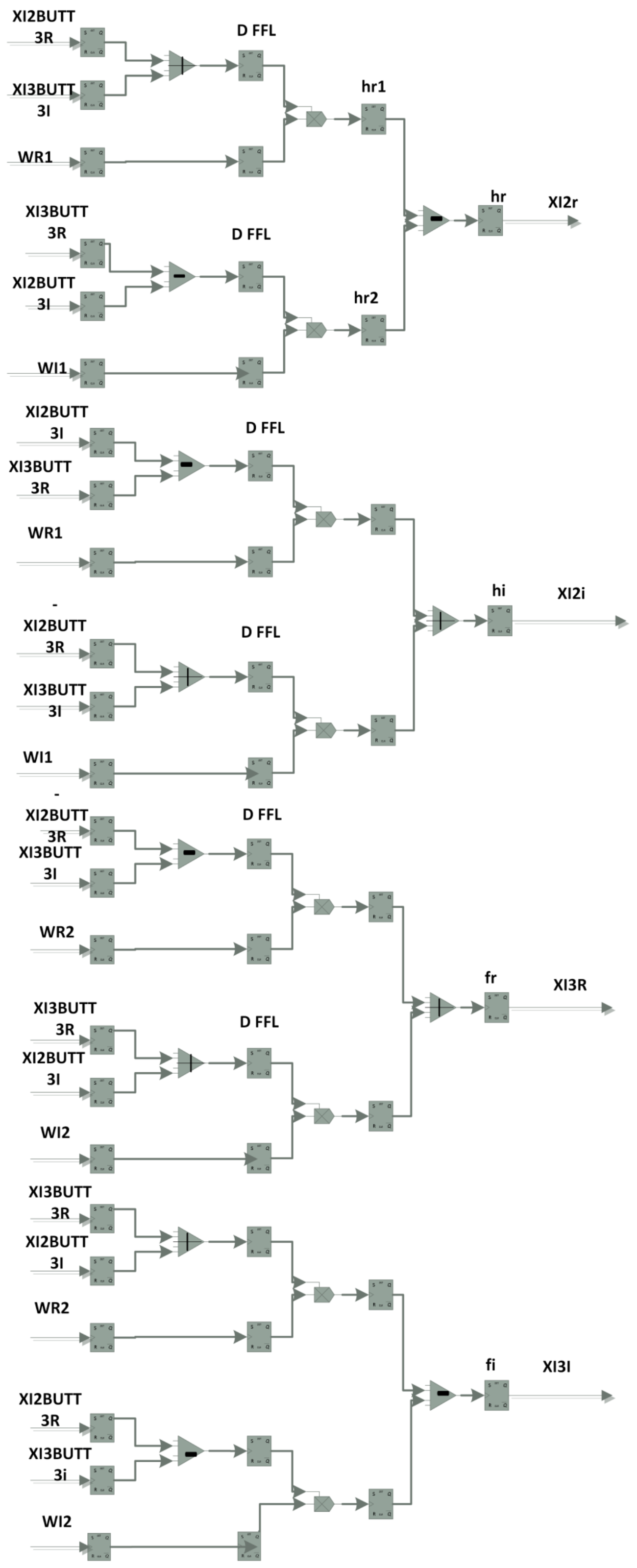


Figure 4-13: Pipelined butterfly 3

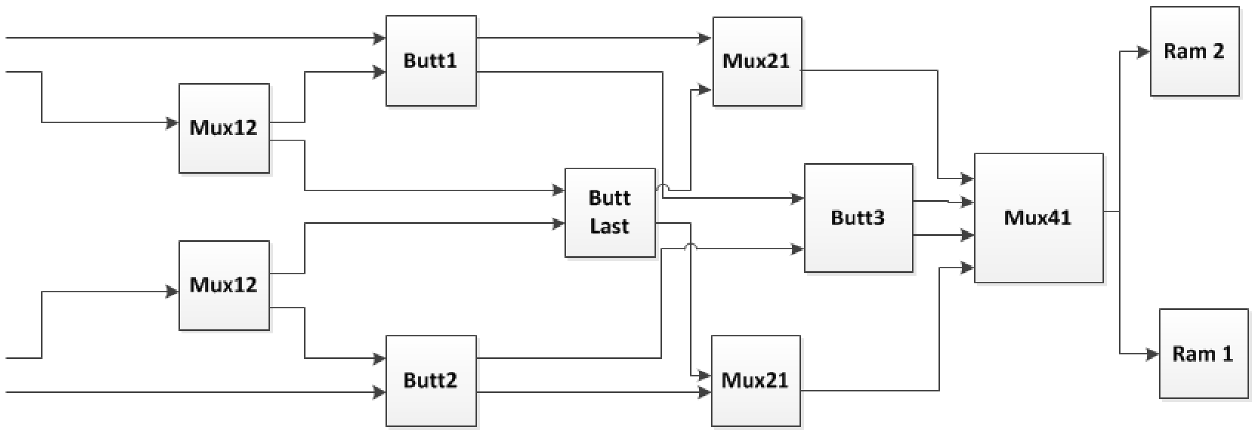


Figure 4-14: The modified butterfly engine

4.2.1 Simulation results

In order to validate the performance of the SRFFT processor, simulations were carried out and the output was compared with the FFT function in Matlab [32]. Three tests were done for SRFFT of sizes 8, 16 and 32. The input values used for the 8-point SRFFT were: 60, 23, 45, 90, 89, 87, 22, and 65. The excellent agreement between Matlab and VHDL simulation results reported in Table 4-2 shows the correct functionality of the developed SRFFT processing engine. The small observed differences in the results are easily explained by the differences in accuracy of the floating representation used in Matlab versus the 16 bit fixed point representation used in VHDL.

Table 4-2: The result of the 8-point SRFFT

Results / Index	**X1**	**X5**	**X3**	**X7**	**X2**	**X6**	**X4**	**X8**
Matlab/ Real	481	-49	82	82	-91.9	33.93	33.93	-91
Matlab/Imaginary	0	0	45	-45	4.5	-50	50.5	-4.5
VHDL/ Real	481	-49	82	82	-92	34	33	-91
VHDL / Imaginary	0	0	45	-45	4	-50	50	-4

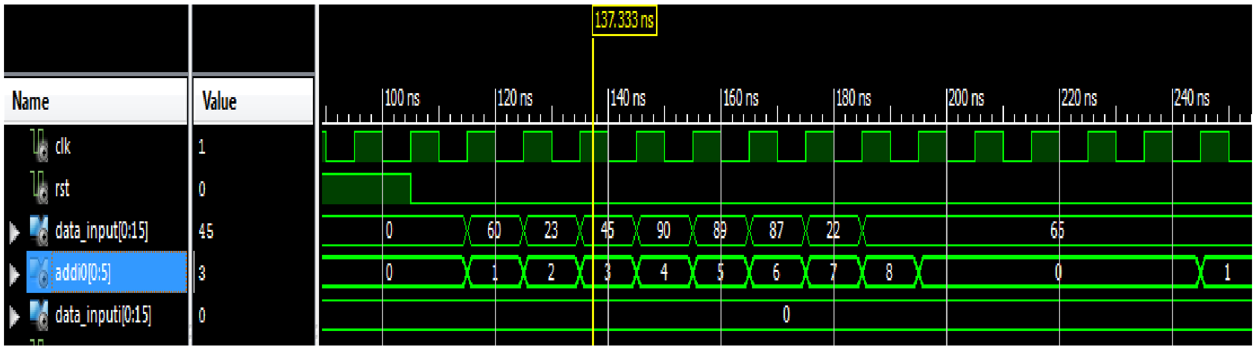


Figure 4-15: Input of the SRFFT

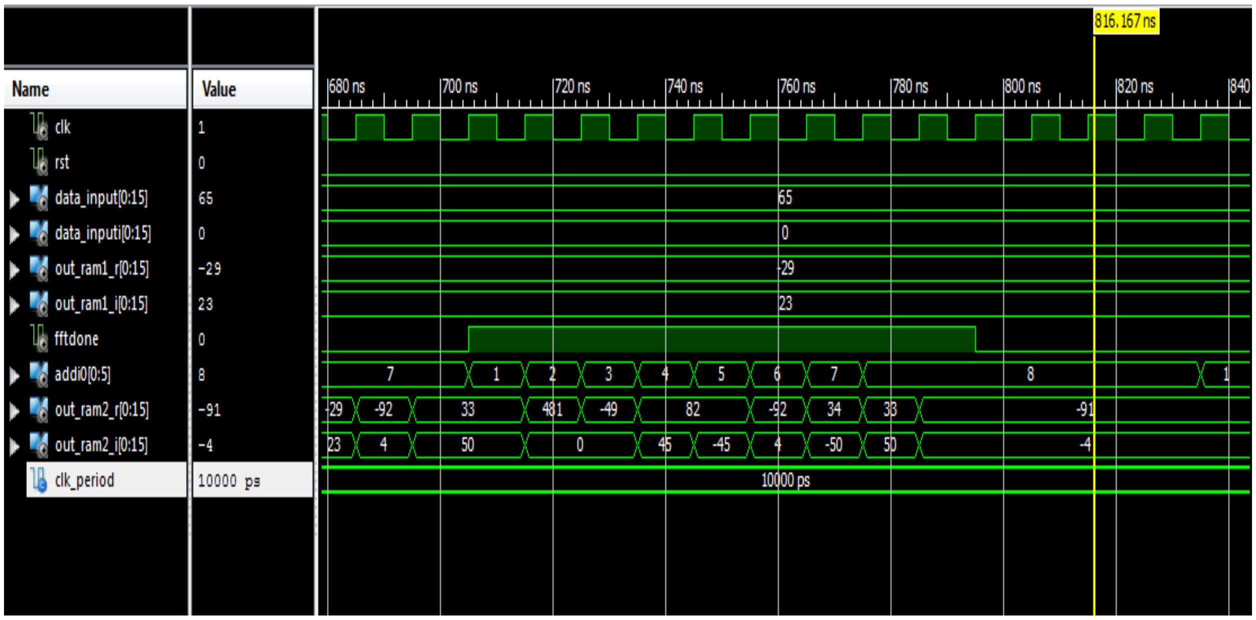


Figure 4-16: The result of the SRFFT

To validate the 16-point SRFFT, the inputs were: 200, 400, 600, 700, 800, 0, 400, 200, 100, 700, 600, 400, 300, 0, and 600. Again the results reported in Table 4-3 from our VHDL SRFFT processor operating with 16-bit words are in excellent agreement with the results from Matlab carried out on a PC in floating point. The outputs of the FFT in Table 4-3 are not sorted and are actually presented in the same order that appears in the last column of SRFFT computation diagram.

Table 4-3: Comparison between Matlab with proposed SRFFT

Index	Matlab/Real	VHDL/Real	Matlab/Imaginary	VHDL/Imaginary
1	6100	6100	0	0
9	-1100	-1100	0	0
5	300	300	800	800
13	300	300	-800	-800
3	-1507	-1508	-675	-676
11	-92	-92	-1524	-1524
7	-92.4	-93	1524	1524
15	-1507	-1507	675	676
2	-226	-228	511	513
10	-398	-398	335	337
6	-464	-464	-228	-229
14	688	690	-795	-795
4	688	688	795	794
12	-464	-464	228	230
8	-398	-398	-338	-336
16	-226	-226	511	512

A snapshot of simulation results for the test with 16 point SRFFT is shown in Figure 4-17 and Figure 4-18. The former shows the time when the input signal is being written in RAM before starting the computations, while the latter one shows the time when the SRFFT computation is done and the results are read out. Note that there is one clock cycle shift between the addresses and the corresponding stored results in RAM. It should be noted that the end of computations is signaled by activating the signal "FFTdone". Note that the read address in Figure 4-18, i.e.

“addi0”, helps reading the RAM contents which are the unsorted SRFFT outputs. For a correct comparison with the results of the FFT from Matlab, we have to note the true index of the FFT output. Indeed, the indexes shown in Table 4-3 are the true indexes of the SRFFT outputs for our 16-point SRFFT processor.

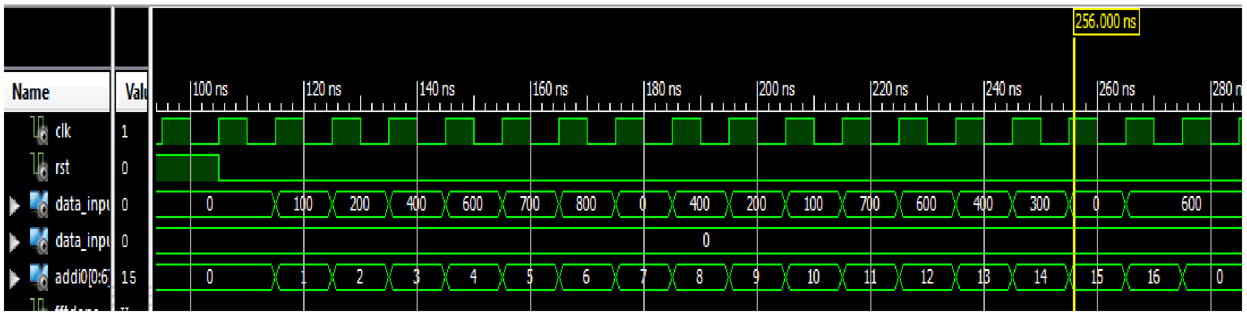


Figure 4-17: Inputs of the 16 point SRFFT

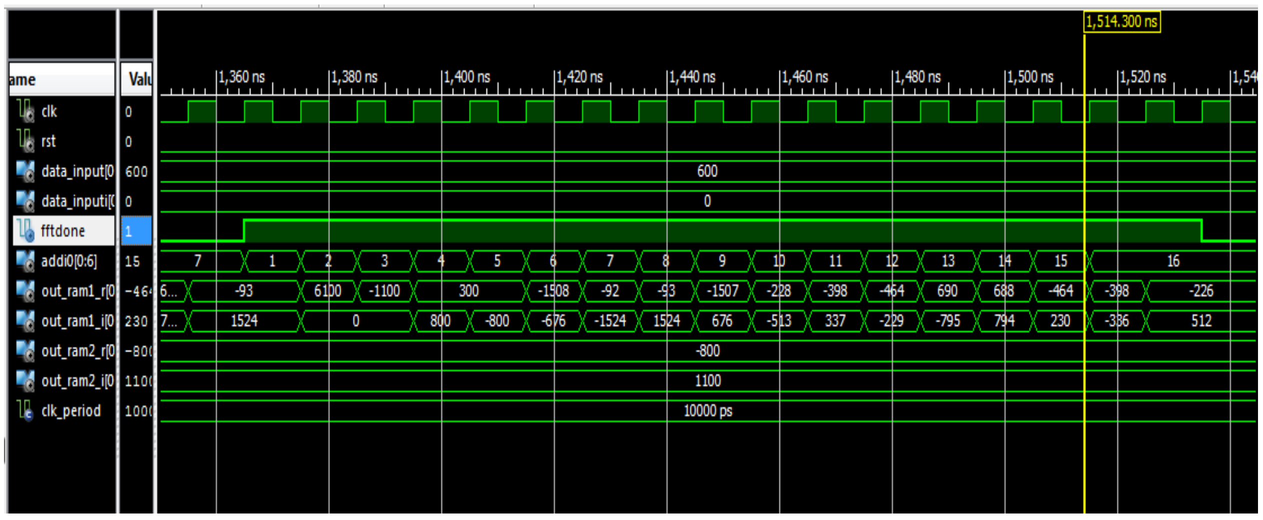


Figure 4-18: Outputs of the 16 point SRFFT

4.3 SRFFT pruning

Figure 4-19 shows the diagram of an 8-point SRFFT computation. It exhibits the L-shape repetitive pattern of computations. It also shows the case where it is pruned when used to process a signal for which only the two first output frequency bins are nonzero, which are depicted by solid circles at the output. Assuming that the presence of nonzero output frequency bins with their exact positions at the output is already known prior to computations, a matrix reflecting the necessity of arithmetic operations in each node can be obtained. This matrix, called the pruning matrix hereafter, has a size identical to the diagram, i.e. $N \times \log_2 N$, and comprises zeros and ones as its elements. In Figure 4-19, the solid circles are implemented as ones in the pruning matrix.

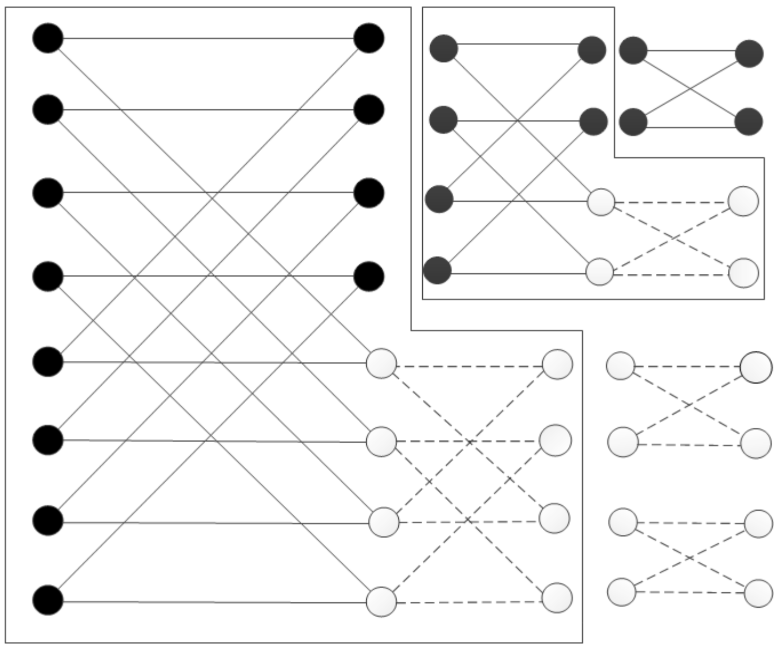


Figure 4-19: 8-point SRFFT diagram

Generation of the pruning matrix begins from the last column and the rest of the matrix can be computed given the last column. Furthermore, the elements in the 3rd column are obtained in groups of 2 from the corresponding elements in the 4th column through OR-gating of the elements used by the relevant butterfly. For example, in the case shown in Figure 4-19, only the two first elements in the 3rd column are non-zeros since the adjacent elements in the 4th column are non-zeros. For the second column, the elements are determined in groups of 4 that are the results of OR-gating of 4 elements in the 3rd column.

In general, for generating the kth column of the pruning matrix of an N-point SRFFT with m=$\log_2 N$ columns, 2^{m-k} elements in (k+1)th column are grouped and then OR gated. The main reason is that the size of butterflies at the last stage is only 2, i.e. they process on only two data inputs, and this size increases by a factor of 2 as we recede back to the first stage.

The result of this OR gating is then written in 2^{m-k} elements of kth column. This procedure continues until the first column of this matrix.

Figure 4-20 shows the pruning matrix for a 16-point SRFFT with an input data vector that contains only four non-zero frequency components. The position of non-zero frequency components at the output of SRFFT is shown to be the first 4 outputs. It is worthwhile to mention that the first column, which shows the position of data inputs in the SRFFT diagram, is not considered in this matrix, because no computation takes part at that column. Having a careful look at the SRFFT diagram, we can observe that it consists of ($\log_2 N$ +1) columns where the first column shows the position of the data inputs and actually no computation takes place in that column. Hence, allocating a column in the pruning matrix for the data inputs is obviously pointless.

Indeed, the last column of this matrix, which essentially corresponds to the FFT output, or the spectrum of the signal, should be known before commencing the FFT computations. Therefore, in FFT cores that are embedded on OFDM receivers, this matrix is first generated using a binary vector of length N that may be obtained from the signal provided by an additional circuitry in receiver block [25].

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 4-20: Pruning matrix of 16 point SRFFT with a typical data input

4.3.1 Matrix pruning generation

As described in the previous section, a pruning matrix is employed to eliminate unnecessary computations. It is composed of binary elements. Each element of this matrix pertains to the corresponding arithmetic computations in the SRFFT diagram. Hence, using the matrix elements, we can decide whether a computation should be carried out or not. We do this through activating/deactivating the three distinct butterfly blocks in the L-shape butterfly engine using enable signals. As was already explained, in each butterfly, a pair of complex computation is done, hence two matrix elements correspond to each butterfly. If one of the computations in this pair is to be done, a decision would be made on activating the butterfly. In other words, if one corresponding element in the pruning matrix is non-zero, the butterfly remains active. Consequently, the enable signal can be obtained through OR-gating the matrix elements. For a L-shape butterfly engine, we would consider 3 pairs of matrix elements.

A pruning engine to generate this matrix was added to the SRFFT core. This engine uses a RAM with a single address port for storing the matrix elements and an FSM for generating the pruning matrix from the spectrum vector. The size of the pruning RAM for an N-point SRFTT is $N \times log_2 N$ and addresses to access these elements must be generated individually, as one row and one column should be addressed. However, viewing each row in this binary matrix as a $log_2 N$ bit word may demand for only one address port. Thereby, the words that compose the matrix elements in one row can be addressed via one address port. Thus, the pruning RAM is composed of N $log_2 N$-bit words, which are stacked up. Once a word is read from the pruning matrix RAM, we can have access to each bit readily. Furthermore, the pruning RAM is indeed structured in the form of N m-bit words as opposed to m N-bit words, because, in each L-shape butterfly, two adjacent columns of the pruning matrix are involved and hence access to the corresponding bits in two adjacent columns is essential. The second column comes into play when the computation of the bottom half of the L-shaped butterfly is carried out. By structuring the pruning RAM the other way around, i.e. m N-bit Words, we would not be able to have access to the pruning bits which correspond to the output of the bottom half of the L-shape butterfly, unless two columns are read at the same time, which is not possible with our single output port RAM. Reading two columns (N-bit words) from the pruning RAM at different times, i.e. one after another, is possible. However, reading and saving them into registers demands large registers (signals) in order to use them in the SRFFT FSM, which may increase power consumption. Also, a separate address signal from SRFFT FSM should be allocated for reading the pruning words, while with the m-bit words pruning matrix RAM, the same address line of data RAM can be shared and used for reading the pruning words.

Figure 4-22 shows the structure of the Split Radix pruning core that comprises one FSM for generating the pruning matrix and a RAM for storing the matrix elements. This FSM implements the algorithm explained in the previous section for computing the matrix elements. For any individual element of the matrix, the whole word is read out of RAM, modified and rewritten back in RAM. Once the matrix is completed, the address and the output ports of the RAM are given to the SRFFT FSM (control unit) to be used for pruning.

During the implementation of the SRFFT, the four inputs of each L-shape butterfly are read in each round from RAMs. The addresses for reading these data from RAMs are generated in control units. In order to take pruning into account, the main FSM (i.e. the FSM that implements the SRFFT) reads four rows (i.e. four words) of the pruning matrix in data memories. Then, the control signals that activate or deactivate the butterfly blocks in the butterfly engine are determined using the corresponding pruning elements, i.e. the kth bit of the read-out "m-bit Word" of the pruning matrix. Furthermore, if the Kth stage is computed, the Kth bits of two pairs of four different words are OR-gated and hence the resulting bits are the enable signals for butterflies in the butterfly engine. The pairs are selected from the addresses with the distance of $N/2^K$.

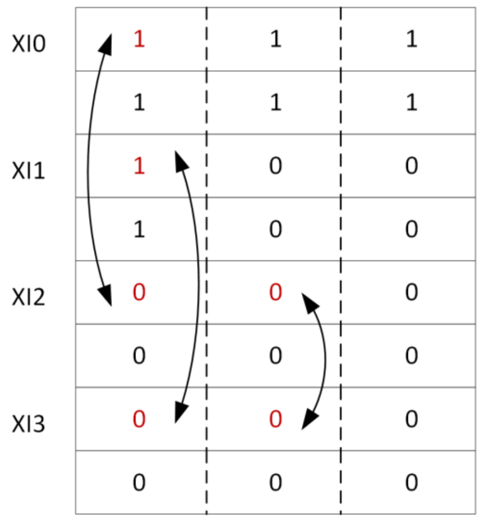
The pruning matrix of the case shown in Figure 4-19 that has only two non-zeros at the output consists of 8 3-bit Words in RAM that is shown in Figure 4-21 as an example. In order to better explain its operation, the SRFFT diagram is shown again in Figure 4-21 as a reference. The gray box shows the corresponding nodes in the pruning matrix. The nodes indicate the position of 0s and 1s, indeed.. The pruning matrix (Pruning Table) is the same as the one shown in the diagram of figure 4-19. The diagram (with lines and dots) in this figure is just to show the existence of several L-shape butterflies in a stage. Also, it highlights that each L-shape butterfly has four inputs. In addition, it intuitively shows the distance between the four inputs of each butterfly.

In every round of computation, four data points are read at four different addresses and denoted as XI0, XI1, XI2 and XI3. The words in the pruning matrix are also read at the same addresses. Each stage may contain several L-shape butterflies. In this example, the first stage includes two L-shape butterflies, each with four inputs. Red and black nodes in the first and second half of the second stage denotes the computation results of two L-shaped butterflies in this stage. The red marked binary numbers in the pruning matrix shown in Figure 4-21 pertain to the red nodes in the diagram shown in the same figure.

When the first L-shape butterfly (the one with red data nodes in Figure 4-21) in first stage is to be computed, four addresses indices of XI0 to XI3 help reading four words at the corresponding addresses. Pairs of data at addresses (XI0, XI2) and (XI1, XI3) are computed in butterflies 1 and 2 respectively. The two computed results at the bottom half of butterflies 1 and 2 make one pair for computing two other data points in the following stage (i.e. stage 2). The latter computations take part in butterfly 3.

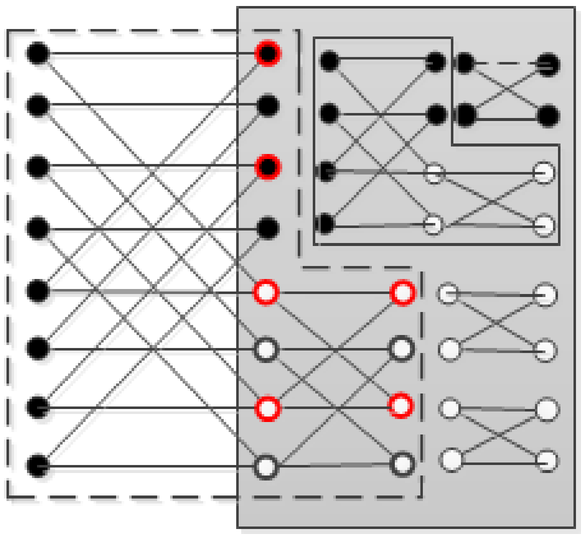
In order to determine whether butterflies 1 and 2 within the L-shape butterfly should be computed or not, the first bit of the words in two pairs of addresses, i.e., (XI0 and XI2), and (XI1 and XI3) are OR-gated. The resulting two signals of OR gates are used as the control signal in these two butterfly blocks as shown in Figure 4-23. In the given example, both butterflies 1 and 2 should remain active because their control signals are 1 as a result of OR gating 1 and 0.

For the second part of the L-shape butterfly, which is implemented in the butterfly 3 block, the 2nd bits of the Words in addresses of XI2 and XI3 in the pruning matrix should be used. They pertain to the second stage of the FFT diagram. In the example of Figure 4-21, the 2nd bit of Words in addresses XI2 and XI3 are OR gated and used as the control signal of buttefly-3, and hence this butterfly should be deactivated as both of its outputs are zeros. The control signal for butterfly 3 is zero as a result of OR-gating 0 and 0. It should be noted as long as the whole butterfly (butterfly 1 or 2 or 3) is controlled by only one control signal, partial pruning can be implemented and that is why the butterfly should remain active even with there is only one non-zero value at its output.



XI0	1	1	1
	1	1	1
XI1	1	0	0
	1	0	0
XI2	0	0	0
	0	0	0
XI3	0	0	0
	0	0	0

(a)Pruning Matrix of (b)



(b) 2 L shape butterflies in first stage of SRFFT diagram

Figure 4-21: Pruning matrix of the 8 point SRFFT with two non-zero values at the output; The matrix corresponds to the exemplary SRFFT diagram shown in Figure 4-19. The red and black circles in the SRFFT diagram on the right show two different L-shape butterflies (each with four data inputs) in the first stage.

In general, the butterflies are advancing in an L-shaped format. In fact, in SRFFT computations, each stage contains one complete column of butterflies that are computed in butterflies 1 and 2, and half of the subsequent column, which are computed in butterfly 3. Therefore, for controlling the butterfly-3 for pruning purposes, the $(K+1)^{th}$ bits of each word from the pruning matrix are OR-gated as the control signal for butterfly-3 while, for butterflies 1 and 2, the K^{th} bits of words in the pruning matrix are OR-gated and used as control signals. K is the stage number and, in the example of Figure 4-21, K equals 1.

These signals are used for stalling the data and the twiddle factors at the input of butterflies, so that unnecessary arithmetic operations are pruned. The changes that are made to our processor by adding the pruning engine are shown in Figure 4-22 and Figure 4-23. In Figure 4-22, it is shown that the pruning matrix is stored in a RAM, which is labeled "RAM Matrix". The address and data output ports of this RAM are shared between the control unit of the pruning matrix and the control unit of the SRFFT, using two distinct multiplexers. When the matrix is generated and is being used by the main control unit, the data outputs are sorted to appear in parallel at the input of the main control unit.

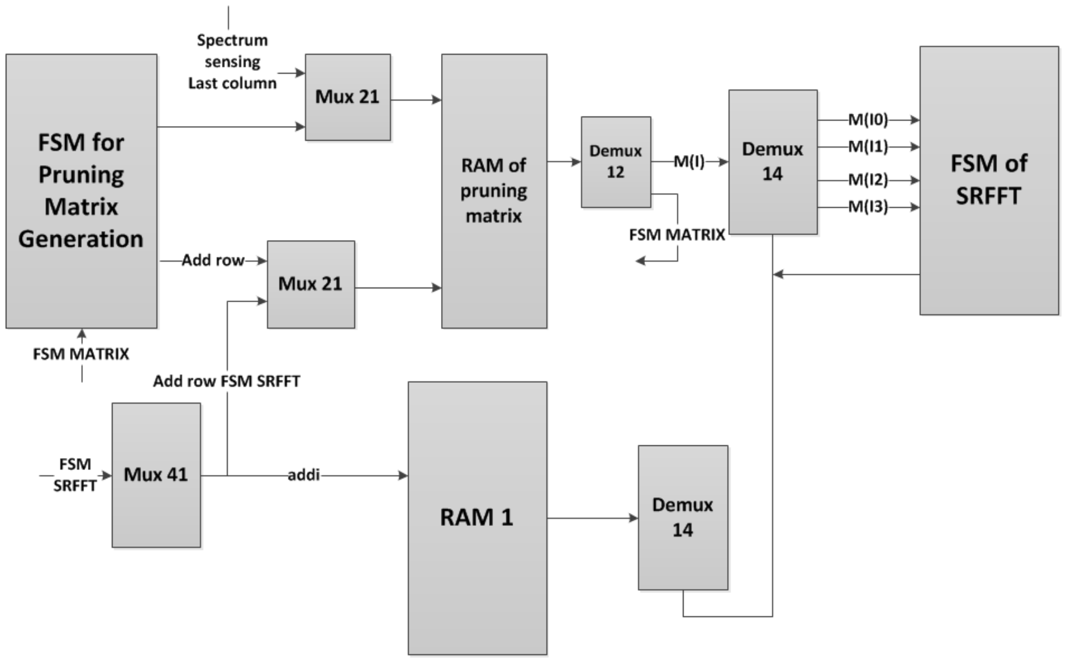


Figure 4-22: The modified SRFFT architecture with pruning engine

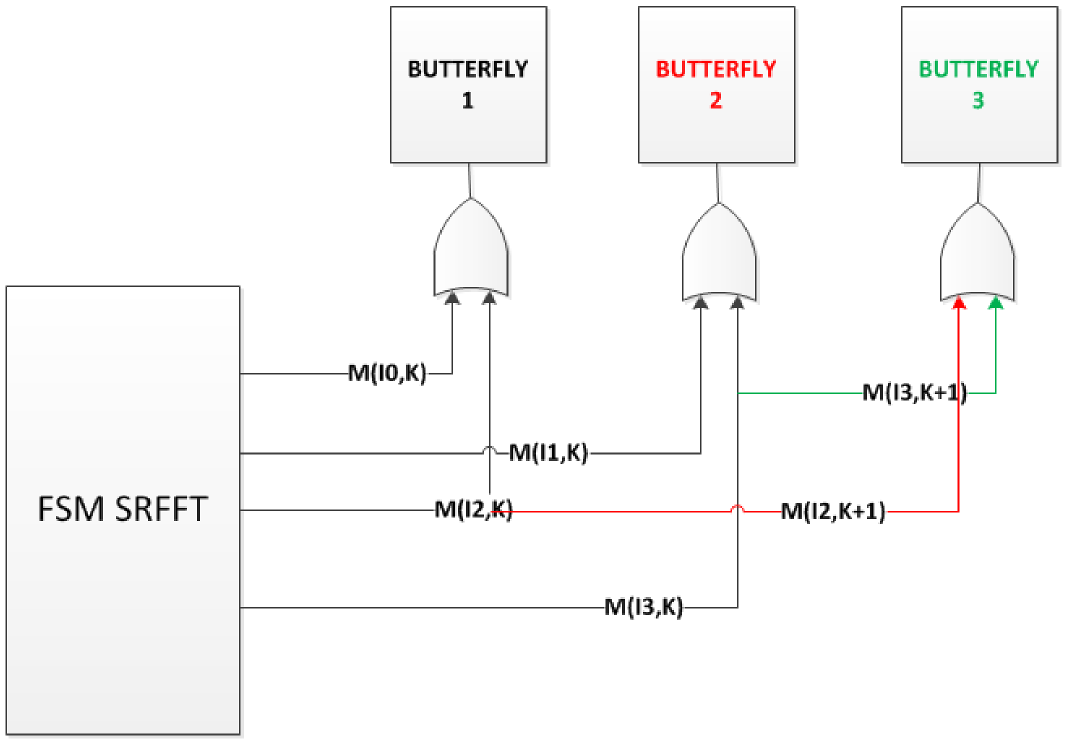


Figure 4-23: Structure used for controlling the butterflies

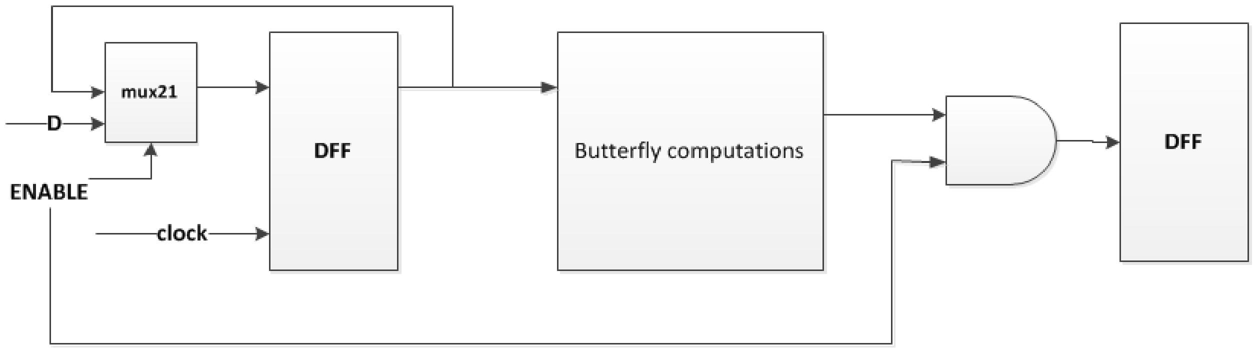


Figure 4-24: Structure used for butterfly stalling

For stalling the butterfly engine when the corresponding control signal is ‘0’, three strategic changes are made to the butterfly units: 1) Deactivate the DFFs at either input or output, 2) the computation results are AND-gated with the control signal and 3) the new incoming data is redirected at the input port of the butterfly unit and replaced with the last data, using a multiplexer at the input ports of the data path. These changes are shown in Figure 4-24.

4.4 Simulations

In order to validate the correct functionality of the proposed SRFFT pruning engine, a set of simulations were carried out and the results are presented next. In Figure 4-25 and Figure 4-26, the procedure for generating the pruning matrix is shown in a short time snapshot. A typical "last column vector" is considered as can be seen by the signal "input", which is connected to the RAM's input port, labeled "ramin". The RAM enable signal, i.e. "enram", is activated with the change of address in order to read data from RAM and also write the calculated element into the corresponding position in RAM. The changes in the signal "addcol" show the progress in computing the matrix elements from the last column (column 3) to the first column (column 1). In this configuration of the pruning matrix, the corresponding algorithm, explained in Section 4.3, is implemented. For example, two elements of the last column (3rd column of pruning matrix) in RAM are called and then the result of OR gating is written back into the RAM in its previous column (2nd column of pruning matrix shown by signal "addcol") at the same row address ("addrow"). The variation in the signal "we1" from "1" to "0" determines the time when the matrix elements are called and processed and then written back into the RAM.

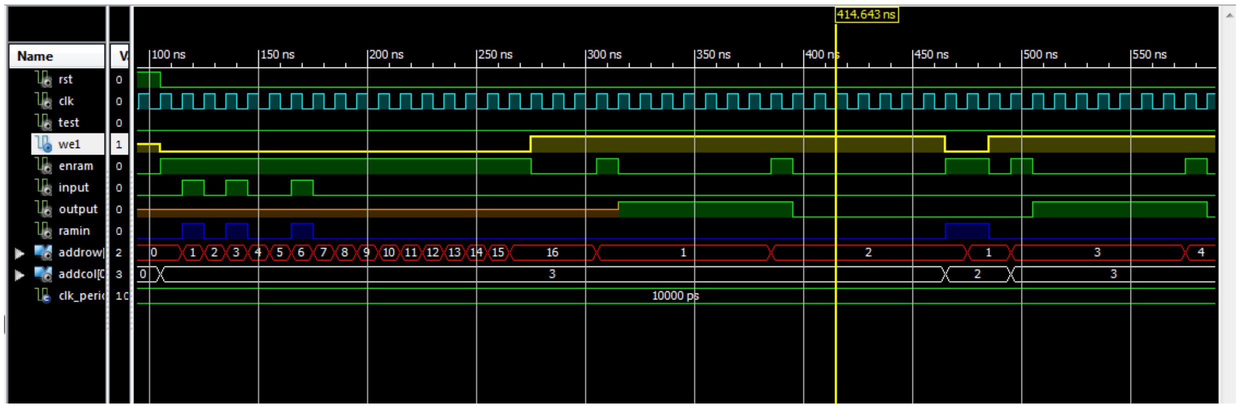


Figure 4-25: 16-point SRFFT pruning; matrix pruning generation for a typical last column input, Part I

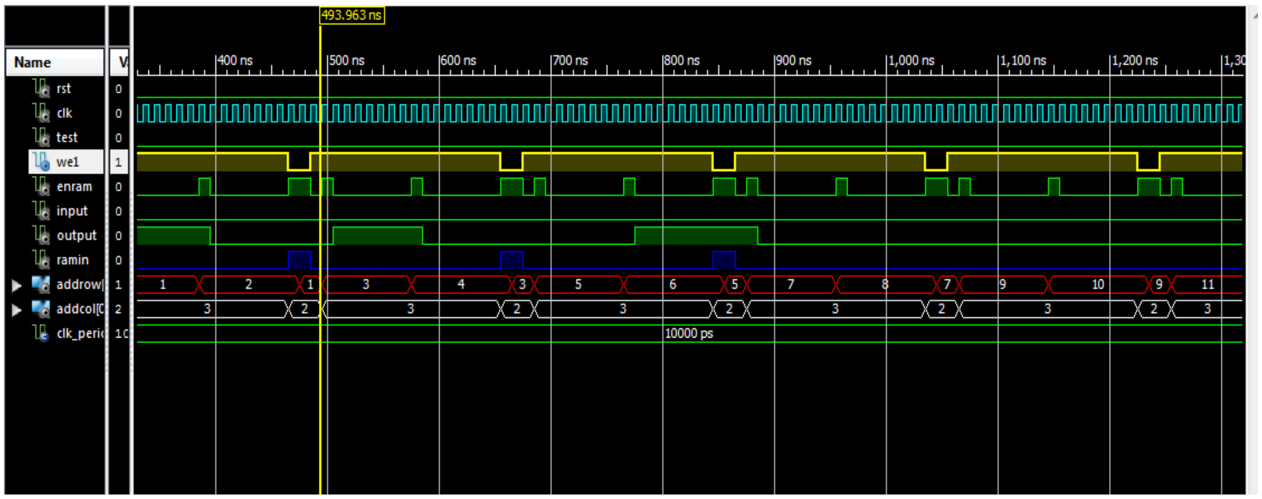


Figure 4-26: 16-point SRFFT pruning; matrix pruning generation for a typical last column input-Part II

The pruning engine is then added to the main SRFFT core and the functionality of both units are evaluated when cooperating. A typical test is carried out for a 16-point SRFFT with a data input that contains only four non-zero frequencies. This data with known frequency contents is formed using the "IFFT" function in Matlab and then used as input data. The input to the "IFFT" function in Matlab was the array [1000, 0, 0, 0, 1000, 0, 0, 0, 1000, 0, 0, 0, 1000, 0, 0, 0]. The corresponding input to a FFT processor that yields this array is [250, 0, 0, 0, 250, 0, 0, 0, 250, 0, 0, 0, 250, 0, 0, 0]. Figure 4-27 shows the last column vector labeled "spectrum", which is active for the four first points, implying that only the first four elements at the output of SRFFT should be nonzero. We should note that the indices at the output of SRFFT diagram are not sorted, hence the first four points at the non-sorted FFT output are not the first four frequency bins of the input signal but other frequency indices. The result of computation is read out of RAM and shown with addresses in Figure 4-28. Note that there is one clock cycle shift between the addresses and the SRFFT indices with respect to the data output. For example, "1000" is aligned to the input of address 2 and continues to the end of address 5, but they belong to address 1 to 4. Table 4-4 compares the result of FFT computation from Matlab and our pruned SRFFT core and confirms the excellent agreement between them.

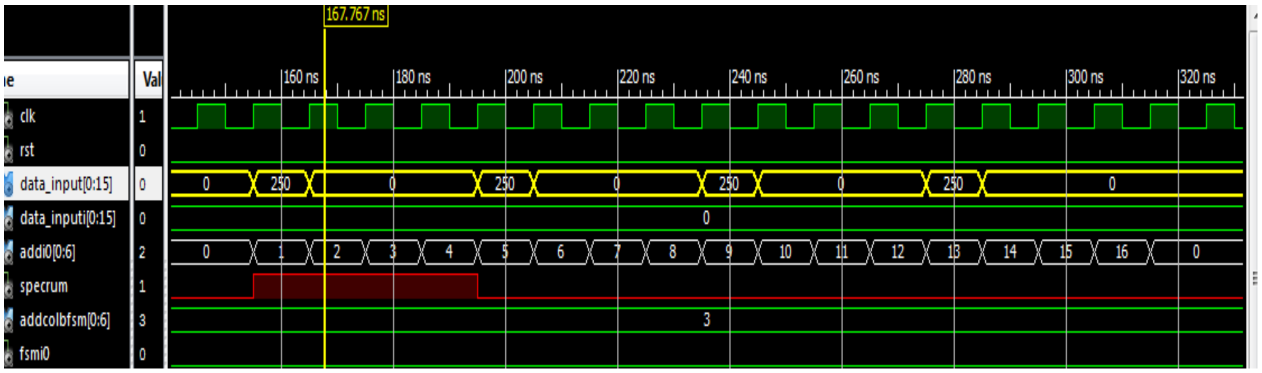


Figure 4-27: Validation test of the pruned split radix FFT; data input and the spectrum vector

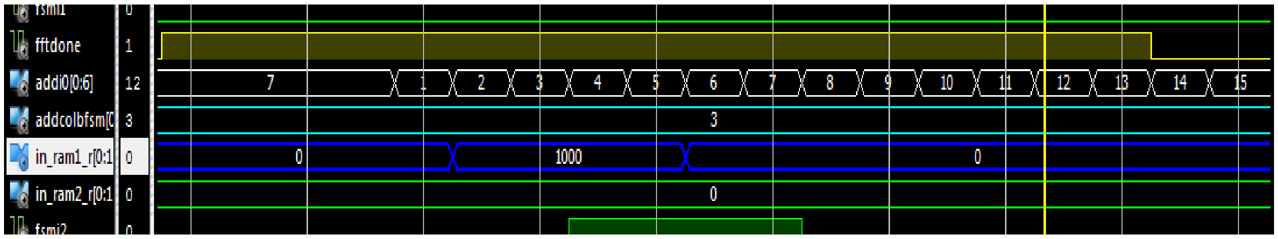


Figure 4-28:Results of the computation

Another test was also carried out for the 16-point FFT but with different frequency contents. Using Matlab, this data set is selected to have four non-zero frequency contents as in the array of [0, 0, 0, 0, 1000, 1000, 1000, 1000, 0, 0, 0, 0, 0, 0, 0, 0]. The result of computations, which is expected to be the unsorted input array of the "IFFT" function in MATLAB, is listed and compared with the true value in Table 4-5. The snapshot of the simulation results showing both inputs and outputs are demonstrated in Figure 4-29 and Figure 4-30. Good agreement between the computations with the pruned SRFFT core and the Matlab function validates the correct functionality of our processor.

Table 4-4: Results of pruned SRFFT computations

FFT Index	Matlab/Real	VHDL/Real	Matlab/Imaginary	VHDL/Imaginary
1	1000	1000	0	0
9	1000	1000	0	0
5	1000	1000	0	0
13	1000	1000	0	0
3	0	0	0	0
11	0	0	0	0
7	0	0	0	0
15	0	0	0	0
2	0	0	0	0
10	0	0	0	0
6	0	0	0	0
14	0	0	0	0
4	0	0	0	0
12	0	0	0	0
8	0	0	0	0
16	0	0	0	0

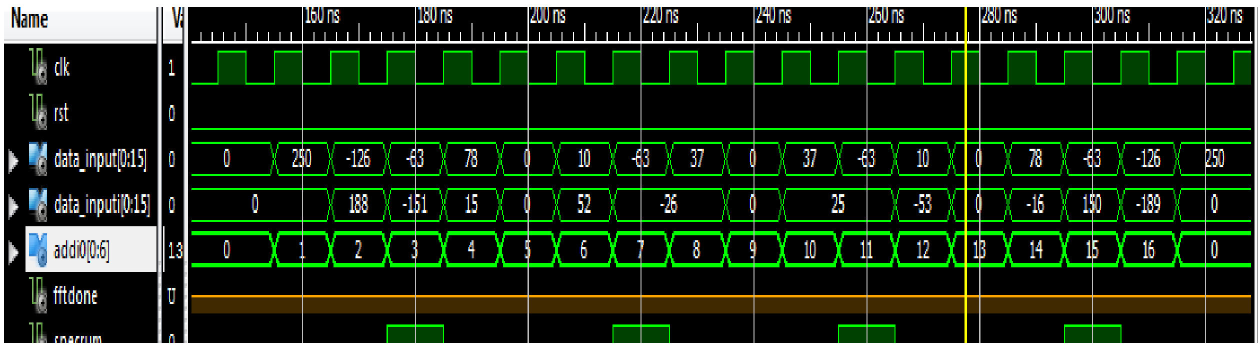


Figure 4-29: split radix pruning with 16-point inputs

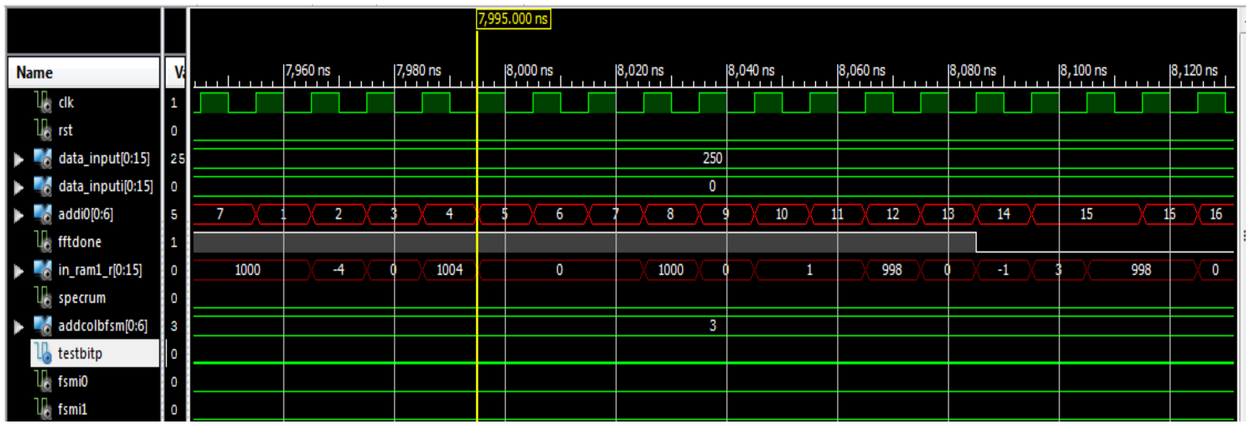


Figure 4-30: Results of a 16 point SRFFT with pruning

Table 4-5: Comparison of the computed results

FFT Index	Matlab/Real	VHDL/Real	Matlab/Image	VHDL/Image
1	0	-4	0	0
9	0	0	0	0
5	1000	1004	0	0
13	0	0	0	0
3	0	0	0	0
11	0	1	0	0
7	1000	1000	0	0
15	0	3	0	0
2	0	0	0	0
10	0	0	0	0
6	1000	998	0	0
14	0	0	0	0
4	0	-1	0	0
12	0	3	0	0
8	1000	998	0	0
16	0	0	0	0

4.5 Synthesis and assessment of power consumption

In order to estimate the power savings, the SRFFT core was used to implement two different FFT lengths, 1024 and 512, on a Xilinx Virtex-6 FPGA. According to the synthesis report of the core targeting an FPGA implementation, presented in Appendix A, the critical delay path of 3.74 ns imposes a maximum operating clock frequency of 268 MHz. The performance of core in terms of power consumption is assessed using the Xpower Analyzer available in Xilinx ISE. When the pruning engine is not actively working, it consumes power for the task of pruning matrix generation. For instance, for 1024-point FFT, the dynamic power consumption for the SRFFT core without a pruning engine and with an inactive one (outputs not used; computations are carried out for power characterisation) is 39.6 mW and 46.0 mW, respectively. The analysis reports are presented in the Appendices E and F. This implies an overhead of around 15% in power consumption when the pruning engine is not actively used. Nevertheless, the active pruning engine can save a considerable amount of overall power in comparison to the SRFFT without a pruning engine. Table 2-1 shows the power analysis result for two different cases when the core operates with a clock frequency of 100 MHz. The comparison is made between the power consumption of an SRFFT core with active pruning engine with reference to a SRFFT core without pruning engine. Furthermore, the reference for comparison is indeed an SRFFT core without any additional logic unit for pruning or disabling the butterflies. The inputs are identical for both cores and for the pruning-SRFFT core, the last column of pruning matrix is provided to be used for generating the pruning matrix in the pruning engine prior to SRFFT computation. One can see that using the pruning engine in maximum sparseness scenarios, i.e. when only four nonzero frequency bins exist, helped reducing the power consumption by 20%. It should be mentioned that in the case of 256 non-zero elements reported in Table 2-1, the non-zero elements are located at the first quarter of the spectrum. Having a closer look at the power simulation results reported in Table 4-7, we can conclude that maximum reduction in power consumption is limited to around 20%. In the second test, when the number of non-zero elements is changed to 256, which is 25% and 50% of non-zero elements for 1024 and 512 point SRFFT, the amount of saving reaches 14% on average. For the latter case, this saving is slightly better than the former one, even though the number of non-zeros is 25% more. This is due to the fact that there is a nonlinear relationship between number of complex multiplications and the power consumption. Indeed, the total power consumption also depends on the number of signals, the size of the RAMs, the number of I/Os and so on. This implies a complicated relationship that explains the nonlinear behaviour with respect to the amount of energy consumption reduction along with the increase in the number of non-zeros with different FFT lengths. Thus, a large reduction in the number of complex multiplications reported in Figure 2-1 does not guarantee the same amount of power saving. Even with the previously reported works on FFT pruning, a difference can be seen from the achieved power saving [17] compared to what can seemingly be expected from Figure 2-1. Thus, with reference to previously reported work, our analysis proves a considerable reduction in power consumption through pruning technique with the SRFFT algorithm.

Table 4-6: Power analysis of SRFFT with pruning engine

Number of nonzero	**Power Saving (%)**	
	1024	**512**
4	20.46	20.87
256	13.62	15.86

The synthesis and also place & route reports of the developed SRFFT core with and without pruning engine are reported in Appendices I to III.

4.5.1 Discussion

In [34], power savings around 11.2% with non-pruned SRFFT over their previous work on ordinary FFT was reported. As already mentioned, around 10% saving on power consumption was achieved through applying pruning technique on ordinary FFT [17]. We should note these works are done on the classic FFT algorithm, which is not optimum as even a non-pruned SRFFT outperforms the pruned FFT in terms of computational complexity, and hence power consumption. To the best of our knowledge, this is the first time that a SRFFT architecture that includes a pruning engine is proposed and its performance is analyzed in terms of power consumption. Comparing the results of this research with the literature, the 20% power consumption saving that was obtained appears to be a significant improvement.

CHAPTER 5 CONCLUSIONS AND FUTURE WORK

In this thesis, an architecture implementing an FFT engine implementing the SRFFT algorithm and including a pruning engine is described.

A SRFFT core is first developed, simulated and synthesized in FPGA and then a pruning engine is added in order to reduce the power consumption. Upon validation of the functionality of the developed core, the performance of the processor is assessed in terms of power consumption. In order to see the amount of reduction in power consumption, the SRFFT core with and without a pruning engine are compared.

In comparison to an SRFFT reference core, the pruning engine can reduce power consumption by more than 20% when the spectrum of the signal is very sparse. This makes the proposed scheme interesting for future communication systems, in particular the cognitive radios based on OFDM.

For further improvements of the current achievements, the authors would like to propose the following options as possible directions of future research:

1) The matrix generation engine consumes a considerable amount of power that yields 15% overhead. Thus, removing this generation step and replacing it with a RAM where the pruning matrix can be stored and ready to use may improve savings on power consumption.
2) Improve the pruning matrix and the algorithm for its generation. Indeed, the current pruning matrix is not able to control each part of a butterfly block and the decision on operation of a butterfly block is made based on the existence of at least one non-zero value at the output of the butterfly. All matrix elements could be considered individually, thus providing a more precise control on the unnecessary computations.
3) Improve the pruning engine so that it consumes less power for matrix pruning generation. Moreover, if the power overhead that exists due to the pruning engine is reduced, more power savings may be anticipated.

REFERENCES

[1] F. Harris, "A coupled multichannel filterbank and sniffer spectrum analyzer," Proc. Cognitive Radio Oriented Wireless Network Communications (CROWNCOM), pp. 1-5, 2010.

[2] L. Berlemann and S. Mangold, "Cognitive Radio for Dynamic Spectrum Access," John Wiley & Sons Ltd, pp. 1-70, 2009.

[3] M. Hunter, A. Kourtellis, C. Ziomek, and W. Mikhael, "Fundamentals of modern spectral analysis", Proc. AUTOTESTCON'10, pp. 1-5, 2010.

[4] S. Haykin, D. J. Thomson, and J. H. Reed. "Spectrum sensing for cognitive radio," Proceedings of the IEEE. vol. 7, no. 5, 2009.

[5] J. Brakensiek, B. Oelkrug, M. Bucker, D. Uffmann, A. Droge, M. Darianian, M. Otte, "Software radio approach for re-configurable multi-standard radios," IEEE International Symp. on Personal, Indoor and Mobile Radio, vol. 1, pp. 110-114, Sept. 2002.

[6] F.K. Jondral, "Software-defined radio: basics and evolution to cognitive radio," EURASIP Journal of Wireless Communications and Networking, vol. 3, pp. 275–283, 2003.

[7] [Available Online]: http://electronicdesign.com/communications/understanding-small-cell-wireless-backhaul.

[8] T.A. Weiss and F.K. Jondral, "Spectrum pooling: an innovative strategy for the enhancement of spectrum efficiency," IEEE Communications Magazine, vol. 42, no. 3, pp. Mar 2004.

[9] R. Tandra, S. M. Mishra, and A. Sahai, B, "What is a spectrum hole and what does it take to recognize one," Proc. IEEE, vol. 97, Mar. 2009.

[10] C. Caloz, S. Gupta, Q. Zhang, B. Nikfal, "Analog signal processing", Microwave magazine, vol. 14, no. 6, pp. 87-103, 2013.

[11] B. Nikfal, D. Badiere, M. Repeta, B. Deforge, S. Gupta, C. Caloz, "Distortion-Less Real-Time Spectrum Sniffing Based on a Stepped Group-Delay Phaser", IEEE Microwave & Wireless Component Letters, vol. 22, no. 11, pp. 601-603. 2013.

[12] C. Yu; M.H. Yen; P.A Hsiung; S. J Chen, "A low-power 64-point pipeline FFT/IFFT processor for OFDM applications," IEEE Transactions on Consumer Electronics, vol. 57, no. 1, p. 40, February 2011.

[13] J. Markel, "FFT pruning," IEEE Transactions on Audio and Electroacoustics,, vol. 19, no. 4, pp. 305-311, Dec 1971.

[14] D.P. Skinner, “Pruning the decimation in-time FFT algorithm," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 24, no. 2, pp.193-194, Apr 1976.

[15] R.G. Alves, P.L. Osorio, M.N.S Swamy, "General FFT pruning algorithm," Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, vol. 3, pp. 1192-1195, 2000

[16] R. Rajbanshi, Wyglinski, M. Alexander; G.J. Minden, "An Efficient Implementation of NC-OFDM Transceivers for Cognitive Radios," 1st International Conference on Cognitive Radio Oriented Wireless Networks and Communications, pp. 1-5, June 2006.

[17] R. Airoldi, F. Campi, M. Cucchi, D. Revanna, O. Anjum, and J. Nurmi, “Design and Implementation of a Power-aware FFT Core for OFDM-based DSA-enabled Cognitive Radios,” Journal of Signal Processing Systems, pp. 1-9, 2014.

[18] S. Holm, "FFT pruning applied to time domain interpolation and peak localization," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process., vol. 35, pp. 1776-1778, Dec. 1987.

[19] T. V. Sreenivas and P. V. S. Rao, "High-resolution narrow band spectra by FFT pruning," in Proc. IEEE Int. Conf. Acoust.Speech, Signal Process., vol. 28, pp. 254-257, Apr. 1980.

[20] T. V. Sreenivas and P. Rao, "FFT algorithm for both input and output Pruning," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process., vol. 27, pp. 291-292, June 1979.

[21] N. Mandal and H. Mandal, "Implementation and performance analysis of an efficient FFT pruning algorithm for NC-OFDM based cognitive radio system," in Communication and Industrial Application (ICCIA), 2011 International Conference on, 2011, pp. 1-6.

[22] H. V. Sorensen and C. S. Burrus, "Efficient computation of the DFT with only a subset of input or output points," Signal Processing, IEEE Transactions on, vol. 41, pp. 1184-1200, 1993.

[23] Q. Zhang, A. B. Kokkeler, and G. J. Smit, "An efficient FFT for OFDM based cognitive radio on a reconfigurable architecture," in Communications, 2007. ICC'07. IEEE International Conference on, 2007, pp. 6522-6526.

[24] H.V. Sorensen, M. Heideman, C.S. Burrus, "On computing the split-radix FFT," Acoustics, IEEE Transactions on Speech and Signal Processing, vol. 34, no. 1, pp. 152-156, Feb 1986.

[25] X. Yihu, L. Myong-Seob, "An efficient design of split-radix FFT pruning for OFDM based Cognitive Radio system," International SoC Design Conference (ISOCC), pp. 368-372, Nov. 2011.

[26] Johnson, H., and Burrus, C. S.: Twiddle factors in the radix-2 FFT'. Proceedings 1982 Asilomar Conference on circuits, systems and computers, pp. 413-416.

[27] P. Duhamel and H. Hollmann, "'Split radix'FFT algorithm," Electronics letters, vol. 20, pp. 14-16, 1984.

[28] P. Duhamel, "Implementation of "split-radix" FFT algorithms for complex, real, and real-symmetric data," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 34, pp. 285-295, 1986.

[29] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Mathematics of computation, vol. 19, pp. 297-301, 1965.

[30] D. L. Jones, "Decimation-in-frequency (DIF) Radix-2 FFT," Connexion project, Version, vol. 1, 2006.

[31] C. Wu, "Implementing the Radix-4 Decimation in Frequency (DIF) Fast Fourier Transform (FFT) Algorithm Using a TMS320C80 DSP," Application report: SPRA152, SC Sales & Marketing–TI Taiwan, Digital signal Processing Solutions, 1998.

[32] http:// www.mathworks.com/products/simulink

[33] J. Y. Kwong and M. Goel, "Constant geometry split radix FFT," ed: Google Patents, 2014.

[34] Z. Qian, N. Nasiri, O. Segal, and M. Margala, "FPGA implementation of low-power split-radix FFT processors," in Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, 2014, pp. 1-2.

[35] M. Cucchi, "Design and implementation of a FFT pruning engine for DSA-enabled cognitive radios," 2013.

[36] http://www.eclipse.org/helios/

[37] P. Duhmel, M. Vetterli, “Fast Fourier transform: a tutorial review and a state of the art”, Elsevier, Signal Processing, vol. 19, pp. 259-299, Oct. 1989

APPENDIX A SRFFT SYNTHESIS REPORT

```
Report of the SRFFT
Device utilization summary:
---------------------------
Selected Device : 6vlx75tff484-3
Slice Logic Utilization:
Number of Slice Registers:           2113  out of  93120     2%
Number of Slice LUTs:                1804  out of  46560     3%
Number used as Logic:              1604  out of  46560     3%
Number used as Memory:              200  out of  16720     1%
Number used as SRL:             200
Slice Logic Distribution:
Number of LUT Flip Flop pairs used:   2686
Number with an unused Flip Flop:     573  out of   2686    21%
Number with an unused LUT:          882  out of   2686    32%
Number of fully used LUT-FF pairs:  1231  out of   2686    45%
Number of unique control sets:       22
IO Utilization:
Number of IOs:                     66
Number of bonded IOBs:               66  out of    240    27%
Specific Feature Utilization:
Number of Block RAM/FIFO:               5  out of    156     3%
Number using Block RAM only:          5
Number of BUFG/BUFGCTRLs:              1  out of     32     3%
Number of DSP48E1s:                  8  out of    288     2%
Timing Summary:
---------------
Speed Grade: -3
Minimum period: 3.738ns (Maximum Frequency: 267.551MHz)
Minimum input arrival time before clock: 2.717ns
Maximum output required time after clock: 1.892ns
Maximum combinational path delay: No path found
Timing Details:
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 3.738ns (frequency: 267.551MHz)
Total number of paths / destination ports: 42325 / 2665
-------------------------------------------------------------------------
Delay:              3.738ns (Levels of Logic = 1)
Source:            Inst_butterfly3/sum1_0 (FF)
Destination:       Inst_butterfly3/hi2_r_0 (FF)
Source Clock:      clk rising
Destination Clock: clk rising
```

```
Data Path: Inst_butterfly3/sum1_0 to Inst_butterfly3/hi2_r_0
       Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
 FDC:C->Q            13   0.280   0.335  Inst_butterfly3/sum1_0 (Inst_butterfly3/sum1_0)
DSP48E1:A15->P29                              1                    2.843                    0.279
Inst_butterfly3/Mmult_wi1_r2[0]_sum2[0]_MuLt_16_OUT
(Inst_butterfly3/wi1_r2[0]_sum2[0]_MuLt_16_OUT<29>)
     FDC:D                 -0.012          Inst_butterfly3/hi2_r_29
    ----------------------------------------
    Total                  3.738ns (3.123ns logic, 0.615ns route)
                              (83.6% logic, 16.4% route)
```

APPENDIX B SRFFT PLACE & ROUTE REPORT

```
Device Utilization Summary:
Slice Logic Utilization:
Number of Slice Registers:                                         2,113 out of  93,120    2%
Number used as Flip Flops:                                         2,113
Number used as Latches:                                            0
Number used as Latch-thrus:                                        0
Number used as AND/OR logics:              0
Number of Slice LUTs:                  1,641 out of  46,560    3%
Number used as logic:                1,288 out of  46,560    2%
Number using O6 output only:            941
Number using O5 output only:             10
Number using O5 and O6:                 337
Number used as ROM:                      0
Number used as Memory:                 200 out of  16,720    1%
Number used as Dual Port RAM:            0
Number used as Single Port RAM:          0
Number used as Shift Register:         200
Number using O6 output only:           200
Number using O5 output only:             0
Number using O5 and O6:                  0
Number used exclusively as route-thrus:    153
Number with same-slice register load:    148
Number with same-slice carry load:         5
Number with other load:                  0

Slice Logic Distribution:
Number of occupied Slices:               466 out of  11,640    4%
Number of LUT Flip Flop pairs used:        1,757
Number with an unused Flip Flop:          258 out of   1,757   14%
Number with an unused LUT:                116 out of   1,757    6%
Number of fully used LUT-FF pairs:       1,383 out of   1,757   78%
Number of slice register sites lost
to control set restrictions:             0 out of  93,120    0%


IO Utilization:
  Number of bonded IOBs:                    66 out of     240   27%

+---------------------+--------------+------+------+------------+-------------+
|       Clock Net    |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+--------------+------+------+------------+-------------+
|         clk_BUFGP | BUFGCTRL_X0Y0| No   |  425 |  0.127     |  1.486      |
+---------------------+--------------+------+------+------------+-------------+

Timing Score: 0 (Setup: 0, Hold: 0, Component Switching Limit: 0)

-------------------------------------------------------------------------------------------------------
  Constraint                                |    Check    | Worst Case |  Best Case | Timing |   Timing
                                        |             |    Slack   | Achievable | Errors |    Score
-------------------------------------------------------------------------------------------------------
  TS_clk = PERIOD TIMEGRP "clk" 20 ns HIGH  | SETUP       |    15.036ns|     4.964ns|       0|
0
  50%                                   | HOLD        |     0.092ns|            |       0|          0
-------------------------------------------------------------------------------------------------------
```

APPENDIX C SRFFT PRUNING SYNTHESIS REPORT

```
Device utilization summary:

---------------------------

Selected Device : 6vlx75tff484-3


Slice Logic Utilization:
Number of Slice Registers:              2360  out of  93120     2%
Number of Slice LUTs:                   2291  out of  46560     4%
Number used as Logic:                   2095  out of  46560     4%
Number used as Memory:                   196  out of  16720     1%
Number used as SRL:                      196

Slice Logic Distribution:
Number of LUT Flip Flop pairs used:     3208
Number with an unused Flip Flop:         848  out of   3208    26%
Number with an unused LUT:               917  out of   3208    28%
Number of fully used LUT-FF pairs:      1443  out of   3208    44%
Number of unique control sets:            40

IO Utilization:
Number of IOs:                            66
Number of bonded IOBs:                    66  out of    240    27%

Specific Feature Utilization:
Number of Block RAM/FIFO:                  6  out of    156     3%
Number using Block RAM only:               6
Number of BUFG/BUFGCTRLs:                  1  out of     32     3%
Number of DSP48E1s:                        8  out of    288     2%

---------------------------

-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 2567  |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
----------------------------------------
-----------------------------------+----------------------------------------+-------+
Control Signal                     | Buffer(FF name)                        | Load  |
----------------------------------+---------------------------------------+-------+
write_ctrl(write_ctrl_INV_0:O)     | NONE(Inst_mamory1_Imag/Mram_ram_11)     | 8     |
write_ctrl2(write_ctrl2_INV_0:O)   | NONE(Inst_memory2/Mram_ram_11)         | 8     |
write_ctrl1(write_ctrl1_INV_0:O)   | NONE(Inst_Ram_matrixpruning/Mram_ram_12)| 4     |
----------------------------------+---------------------------------------+-------+

Timing Summary:
---------------
Speed Grade: -3

Minimum period: 3.738ns (Maximum Frequency: 267.551MHz)
Minimum input arrival time before clock: 1.943ns
Maximum output required time after clock: 1.892ns
Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
====
Timing constraint: Default period analysis for Clock 'clk'
Clock period: 3.738ns (frequency: 267.551MHz)
Total number of paths / destination ports: 70991 / 4318
-------------------------------------------------------------------------
Delay:              3.738ns (Levels of Logic = 1)
Source:            Inst_butterfly3/sum1_0 (FF)
Destination:       Inst_butterfly3/hi2_r_29 (FF)
Source Clock:      clk rising
Destination Clock: clk rising

Data Path: Inst_butterfly3/sum1_0 to Inst_butterfly3/hi2_r_29
Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
FDCE:C->Q           13   0.280   0.335  Inst_butterfly3/sum1_0 (Inst_butterfly3/sum1_0)
DSP48E1:A15->P29      1   2.843   0.279
Inst_butterfly3/Mmult_wi1_r2[0]_sum2[0]_MuLt_16_OUT
(Inst_butterfly3/wi1_r2[0]_sum2[0]_MuLt_16_OUT<29>)
FDCE:D                 -0.012          Inst_butterfly3/hi2_r_29
----------------------------------------
Total                     3.738ns (3.123ns logic, 0.615ns route)
                                   (83.6% logic, 16.4% route)
```

```
==========================================================================
====
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
Total number of paths / destination ports: 2204 / 2198
-------------------------------------------------------------------------
Offset:              1.943ns (Levels of Logic = 4)
Source:              rst (PAD)
Destination:         Inst_statemachine_Mram__n33331 (RAM)
Destination Clock: clk rising

Data Path: rst to Inst_statemachine_Mram__n33331
Gate      Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
IBUF:I->O         1967   0.003   0.581  rst_IBUF (rst_IBUF)
LUT3:I1->O           4   0.053   0.310  Inst_statemachine/_n198821 (Inst_statemachine/_n19882)
LUT5:I4->O           1   0.053   0.279  Inst_statemachine/_n29361 (Inst_statemachine/_n2936)
INV:I->O             2   0.070   0.284  Inst_statemachine__n2936_inv_INV_0
(Inst_statemachine__n2936_inv)
RAMB36E1:ENARDEN         0.310          Inst_statemachine_Mram__n33331
----------------------------------------
Total                    1.943ns (0.489ns logic, 1.454ns route)
(25.2% logic, 74.8% route)

==========================================================================
====
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
Total number of paths / destination ports: 64 / 32
-------------------------------------------------------------------------
Offset:              1.892ns (Levels of Logic = 1)
Source:              Inst_mamory1/Mram_ram_12 (RAM)
Destination:         in_ram1_r<0> (PAD)
Source Clock:        clk rising

Data Path: Inst_mamory1/Mram_ram_12 to in_ram1_r<0>
Gate      Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
RAMB18E1:CLKARDCLK->DOADO7    5   1.591   0.298  Inst_mamory1/Mram_ram_12
(sigz1DATAr<0>)
OBUFT:I->O               0.003          in_ram1_r_0_OBUFT (in_ram1_r<0>)
----------------------------------------
Total                    1.892ns (1.594ns logic, 0.298ns route)
                                          (84.2% logic, 15.8% route)
```

```
=========================================================================

Cross Clock Domains Report:
--------------------------

Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |    3.738|         |         |         |
---------------+---------+---------+---------+---------+

=========================================================================


Total REAL time to Xst completion: 36.00 secs
Total CPU time to Xst completion: 36.97 secs

-->

Total memory usage is 331572 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :  305 (   0 filtered)
Number of infos    :   62 (   0 filtered)
```

APPENDIX D SRFFT PRUNING PLACE & ROUTE REPORT

```
Device Utilization Summary:

Slice Logic Utilization:
Number of Slice Registers:              2,360 out of  93,120    2%
Number used as Flip Flops:            2,360
Number used as Latches:                  0
Number used as Latch-thrus:                0
Number used as AND/OR logics:              0
Number of Slice LUTs:                   2,050 out of  46,560    4%
Number used as logic:               1,660 out of  46,560    3%
Number using O6 output only:          1,121
Number using O5 output only:             33
Number using O5 and O6:               506
Number used as ROM:                     0
Number used as Memory:                   196 out of  16,720    1%
Number used as Dual Port RAM:             0
Number used as Single Port RAM:           0
Number used as Shift Register:          196
Number using O6 output only:          196
Number using O5 output only:            0
Number using O5 and O6:               0
Number used exclusively as route-thrus:    194
Number with same-slice register load:    187
Number with same-slice carry load:        7
Number with other load:                  0

Slice Logic Distribution:
Number of occupied Slices:                 667 out of  11,640    5%
Number of LUT Flip Flop pairs used:        2,393
Number with an unused Flip Flop:           520 out of   2,393   21%
Number with an unused LUT:                343 out of   2,393   14%
Number of fully used LUT-FF pairs:       1,530 out of   2,393   63%
Number of slice register sites lost
to control set restrictions:              0 out of  93,120    0%


IO Utilization:
Number of bonded IOBs:                     66 out of     240   27%

Specific Feature Utilization:
Number of RAMB36E1/FIFO36E1s:                1 out of     156    1%
Number using RAMB36E1 only:                  1
Number using FIFO36E1 only:              0
Number of RAMB18E1/FIFO18E1s:             10 out of     312    3%
Number using RAMB18E1 only:               10
Number using FIFO18E1 only:              0
Number of BUFG/BUFGCTRLs:                 1 out of      32    3%
Number used as BUFGs:                   1
Number used as BUFGCTRLs:                0
Number of ILOGICE1/ISERDESE1s:              0 out of     360    0%
Number of OLOGICE1/OSERDESE1s:              0 out of     360    0%
Number of BSCANs:                       0 out of       4    0%
Number of BUFHCEs:                       0 out of      72    0%
Number of BUFOs:                        0 out of      18    0%
Number of BUFIODQSs:                     0 out of      36    0%
Number of BUFRs:                        0 out of      18    0%
Number of CAPTUREs:                      0 out of       1    0%
Number of DSP48E1s:                      8 out of     288    2%
Number of EFUSE_USRs:                     0 out of       1    0%
Number of FRAME_ECCs:                     0 out of       1    0%
Number of GTXE1s:                       0 out of       8    0%
Number of IBUFDS_GTXE1s:                   0 out of       6    0%
Number of ICAPs:                        0 out of       2    0%
Number of IDELAYCTRLs:                    0 out of       9    0%
Number of IODELAYE1s:                     0 out of     360    0%
Number of MMCM_ADVs:                      0 out of       6    0%
Number of PCIE_2_0s:                     0 out of       1    0%
Number of STARTUPs:                      1 out of       1  100%
Number of SYSMONs:                       0 out of       1    0%
Number of TEMAC_SINGLEs:                   0 out of       4    0%

+---------------------+--------------+------+------+------------+-------------+
|      Clock Net      |   Resource   |Locked|Fanout|Net Skew(ns)|Max Delay(ns)|
+---------------------+--------------+------+------+------------+-------------+
|         clk_BUFGP | BUFGCTRL_X0Y0| No   | 549 |  0.215     | 1.570      |
+---------------------+--------------+------+------+------------+-------------+


------------------------------------------------------------------------------------------------------
  Constraint                          |    Check    | Worst Case |  Best Case | Timing |   Timing
                                      |             |    Slack   | Achievable | Errors |    Score
------------------------------------------------------------------------------------------------------
  TS_clk = PERIOD TIMEGRP "clk" 5 ns HIGH 5 | SETUP       |     0.247ns|     4.753ns|       0|
0
  0%                                  | HOLD        |     0.088ns|            |       0|         0
```

APPENDIX E POWER REPORT FOR SRFFT WITHOUT PRUNING ENGINE FOR 1024

```
Release              | 13.1 - O.40d (nt64)                |
| Command Line          | Generated from Graphical User Interface |
-------------------------------------------------------------------

--------------------------------
|      Table of Contents       |
--------------------------------
| 1.  Settings                |
| 1.1.  Project               |
| 1.2.  Device                |
| 1.3.  Environment           |
| 1.4.  Default Activity Rates |
| 2.  Summary                 |
| 2.1.  On-Chip Power Summary  |
| 2.2.  Thermal Summary        |
| 2.3.  Power Supply Summary   |
| 2.4.  Confidence Level       |
| 3.  Detailed Reports         |
| 3.1.  By Hierarchy           |
| 3.2.  By Clock Domain        |
| 3.3.  By Resource Type       |
| 3.3.1.  Core Dynamic         |
| 3.3.1.1.  Logic              |
| 3.3.1.2.  Signals            |
| 3.3.1.3.  BRAM               |
| 3.3.1.4.  DSP                |
| 3.3.2.  IO                   |
--------------------------------

1.  Settings
1.1.  Project
------------------------------------------
|               Project                 |
------------------------------------------
| Design File               | portmap.ncd |
| Settings File             | NA          |
| Physical Constraints File | portmap.pcf |
| Simulation Activity File  | NA          |
| Design Nets Matched       | NA          |
| Simulation Nets Matched   | NA          |
------------------------------------------

1.2.  Device
-----------------------------------------------
|                  Device                     |
-----------------------------------------------
| Family          | Virtex6                   |
| Part            | xc6vlx75t                 |
| Package         | ff484                     |
| Grade           | C-Grade                   |
| Process         | Typical                   |
| Speed Grade     | -3                        |
| Characterization | Production,v1.2,2010-12-16 |
-----------------------------------------------

1.3.  Environment
-------------------------------------------
|              Environment                |
-------------------------------------------
| Ambient Temp (C)      | 50.0            |
| Use custom TJA?       | No              |
| Custom TJA (C/W)      | NA              |
| Airflow (LFM)         | 250             |
| Heat Sink             | None            |
| Custom TSA (C/W)      | NA              |
| Board Selection       | Medium (10"x10") |
| # of Board Layers     | 12 to 15        |
| Custom TJB (C/W)      | NA              |
| Board Temperature (C) | NA              |
-------------------------------------------

1.4.  Default Activity Rates
---------------------------------
|    Default Activity Rates     |
---------------------------------
| FF Toggle Rate (%)     | 12.5  |
| I/O Toggle Rate (%)    | 12.5  |
| Output Load (pF)       | 5.0   |
| I/O Enable Rate (%)    | 100.0 |
| BRAM Write Rate (%)    | 50.0  |
| BRAM Enable Rate (%)   | 25.0  |
| DSP Toggle Rate (%)    | 12.5  |
---------------------------------
```

```
2.  Summary
2.1.  On-Chip Power Summary
-----------------------------------------------------------------------------
|                    On-Chip Power Summary                    |
-----------------------------------------------------------------------------
|     On-Chip      | Power (mW) |  Used  | Available | Utilization (%) |
-----------------------------------------------------------------------------
| Clocks           |    16.54 |     1 |   ---   |     ---     |
| Logic            |     1.47 |  1641 |   46560 |           4 |
| Signals          |     2.73 |  3016 |   ---   |     ---     |
| IOs              |    14.17 |    66 |     240 |          28 |
| BlockRAM/FIFO    |     4.45 |  ---  |   ---   |     ---     |
|   18K BlockRAM   |     4.35 |     8 |     312 |           3 |
|   36K BlockRAM   |     0.11 |     1 |     156 |           1 |
| DSPs             |     0.23 |     8 |     288 |           3 |
| Quiescent        |  1326.06 |       |         |             |
| Total            |  1365.65 |       |         |             |
-----------------------------------------------------------------------------

2.2.  Thermal Summary
------------------------------
|     Thermal Summary      |
------------------------------
| Effective TJA (C/W) | 5.6  |
| Max Ambient (C)     | 77.4 |
| Junction Temp (C)   | 57.6 |
------------------------------

2.3.  Power Supply Summary
--------------------------------------------------------
|              Power Supply Summary              |
--------------------------------------------------------
|                |  Total  | Dynamic | Quiescent |
--------------------------------------------------------
| Supply Power (mW)    | 1365.65 | 39.60   | 1326.06   |
--------------------------------------------------------

---------------------------------------------------------------------------------------------------------------
|                                  Power Supply Currents                                  |
---------------------------------------------------------------------------------------------------------------
|    Supply Source     | Supply Voltage | Total Current (mA) | Dynamic Current (mA) | Quiescent
Current (mA) |
---------------------------------------------------------------------------------------------------------------
| Vccint           |       1.000 |          678.97 |              26.67 |              652.30 |
| Vccaux           |       2.500 |           45.27 |               0.27 |               45.00 |
| Vcco25        |     2.500 |        5.90 |        4.90 |        1.00 |
| MGTAVcc       |     1.000 |      303.43 |        0.00 |      303.43 |
| MGTAVtt       |     1.200 |      212.78 |        0.00 |      212.78 |
----------------------------------------------------------------------------------------------------------------
```

2.4. Confidence Level

```
-----------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
|                                                                                                      Confidence Level
|
```

APPENDIX F POWER REPORT WITH PRUNING FOR 1024 NON-ZERO

```
                Xilinx XPower Analyzer                |
--------------------------------------------------------------------
| Release              | 13.1 - O.40d (nt64)                |
| Command Line          | Generated from Graphical User Interface |
--------------------------------------------------------------------

--------------------------------
|      Table of Contents       |
--------------------------------
| 1.  Settings                 |
| 1.1.  Project                |
| 1.2.  Device                 |
| 1.3.  Environment            |
| 1.4.  Default Activity Rates |
| 2.  Summary                  |
| 2.1.  On-Chip Power Summary  |
| 2.2.  Thermal Summary        |
| 2.3.  Power Supply Summary   |
| 2.4.  Confidence Level       |
| 3.  Detailed Reports         |
| 3.1.  By Hierarchy           |
| 3.2.  By Clock Domain        |
| 3.3.  By Resource Type       |
| 3.3.1.  Core Dynamic         |
| 3.3.1.1.  Logic              |
| 3.3.1.2.  Signals            |
| 3.3.1.3.  BRAM               |
| 3.3.1.4.  DSP                |
| 3.3.2.  IO                   |
--------------------------------

1.  Settings
1.1.  Project
------------------------------------------
|               Project                  |
------------------------------------------
| Design File               | portmap.ncd |
| Settings File             | NA          |
| Physical Constraints File | portmap.pcf |
| Simulation Activity File  | NA          |
| Design Nets Matched       | NA          |
| Simulation Nets Matched   | NA          |
-----------------------------------------

1.2.  Device
-----------------------------------------------
|                 Device                      |
-----------------------------------------------
| Family          | Virtex6                   |
| Part            | xc6vlx75t                 |
| Package         | ff484                     |
| Grade           | C-Grade                   |
| Process         | Typical                   |
| Speed Grade     | -3                        |
| Characterization | Production,v1.2,2010-12-16 |
-----------------------------------------------

1.3.  Environment
------------------------------------------
|              Environment               |
------------------------------------------
| Ambient Temp (C)      | 50.0           |
| Use custom TJA?       | No             |
| Custom TJA (C/W)      | NA             |
| Airflow (LFM)         | 250            |
| Heat Sink             | None           |
| Custom TSA (C/W)      | NA             |
| Board Selection       | Medium (10"x10") |
| # of Board Layers     | 12 to 15       |
| Custom TJB (C/W)      | NA             |
| Board Temperature (C) | NA             |
------------------------------------------

1.4.  Default Activity Rates
--------------------------------
|    Default Activity Rates    |
--------------------------------
| FF Toggle Rate (%)    | 12.5  |
| I/O Toggle Rate (%)   | 12.5  |
| Output Load (pF)      | 5.0   |
| I/O Enable Rate (%)   | 100.0 |
| BRAM Write Rate (%)   | 50.0  |
| BRAM Enable Rate (%)  | 25.0  |
| DSP Toggle Rate (%)   | 12.5  |
--------------------------------
```

```
2.  Summary
2.1.  On-Chip Power Summary
------------------------------------------------------------------------
|                        On-Chip Power Summary                         |
------------------------------------------------------------------------
|       On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
------------------------------------------------------------------------
| Clocks               |      19.64 |      1 |    ---    |       ---       |
| Logic                |       1.60 |   1977 |     46560 |               4 |
| Signals              |       2.75 |   3571 |    ---    |       ---       |
| IOs                  |      16.16 |     66 |       240 |              28 |
| BlockRAM/FIFO        |       5.61 |  ---   |    ---    |       ---       |
|   18K BlockRAM       |       5.59 |     10 |       312 |               3 |
|   36K BlockRAM       |       0.01 |      1 |       156 |               1 |
| DSPs                 |       0.23 |      8 |       288 |               3 |
| Quiescent            |    1326.37 |        |           |                 |
| Total                |    1372.37 |        |           |                 |
------------------------------------------------------------------------

2.2.  Thermal Summary
----------------------------
|     Thermal Summary      |
----------------------------
| Effective TJA (C/W) | 5.6  |
| Max Ambient (C)     | 77.4 |
| Junction Temp (C)   | 57.6 |
----------------------------

2.3.  Power Supply Summary
----------------------------------------------------
|               Power Supply Summary               |
----------------------------------------------------
|                   |  Total  | Dynamic | Quiescent |
----------------------------------------------------
| Supply Power (mW)    | 1372.37 | 45.99   | 1326.37   |
----------------------------------------------------
```

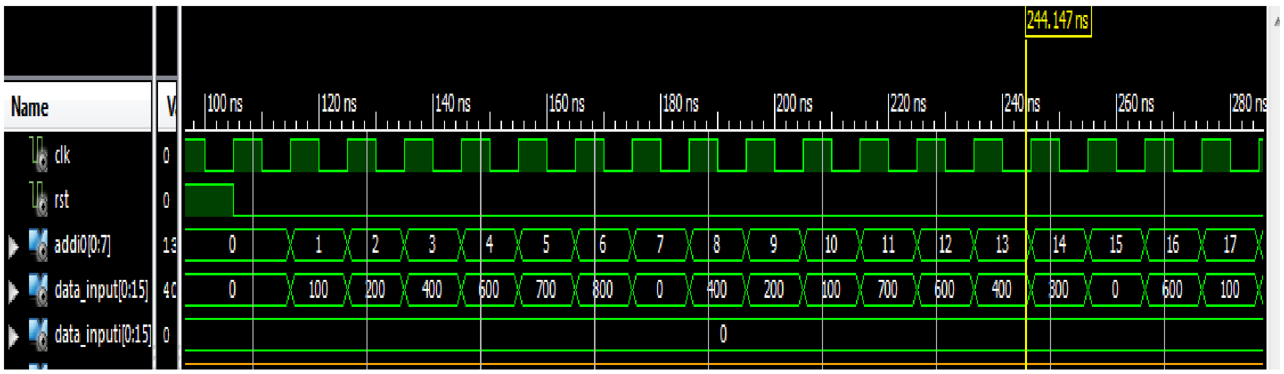
APPENDIX G FOR VALIDATING THE SRFFT PERFORMANCE

Inputs: 100, 200, 400, 600, 700, 800, 0, 400, 200, 100, 700, 600, 400, 300, 0, 600, 100, 200, 400, 600, 700, 800, 0, 400, 200, 100, 700, 600, 400, 300, 0, 600.

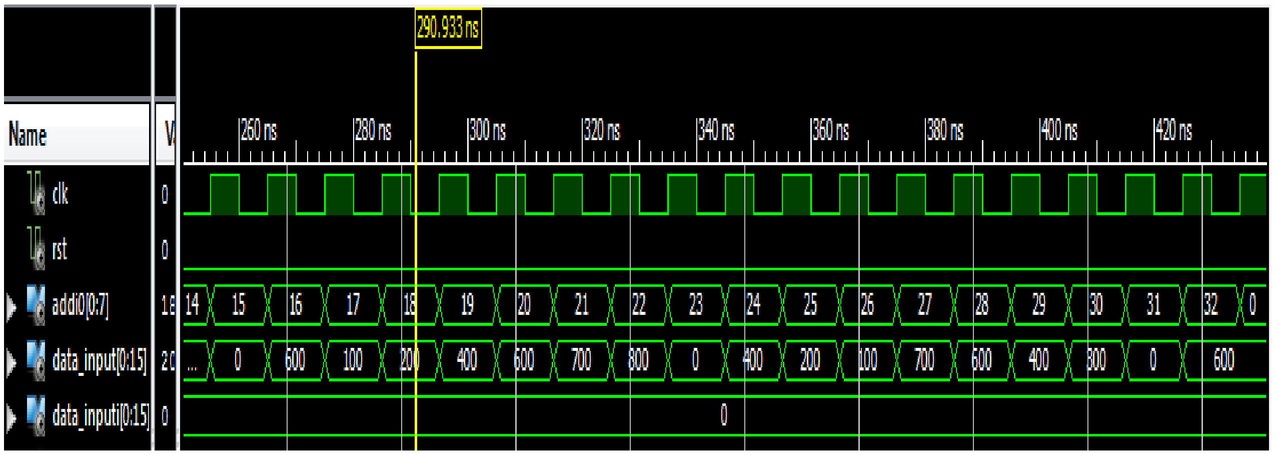
Table 1: The comparison between Matlab and proposed SRFFT

Matlab/Real	VHDL/Real	Matlab/Image	VHDL/Image
12200	12200	0	0
-2200	-2200	0	0
600	600	1600	1600
600	600	-1600	-1600
-3014	-3015	-1351	-1352
-186	-185	-3049	-3048
-186	-186	3049	-3048
-3014	-3014	1351	1352
-453	-454	1023	-1024
79.64	-796	672	672
-929	-929	-457	-467
1378	1379	-1592	-1591
1378	1377	1592	1590
-929	-929	457	458
-796	-796	-672	-671
-453	-452	1023	1023
0	0	0	0
0	0	0	0
0	0	0	0

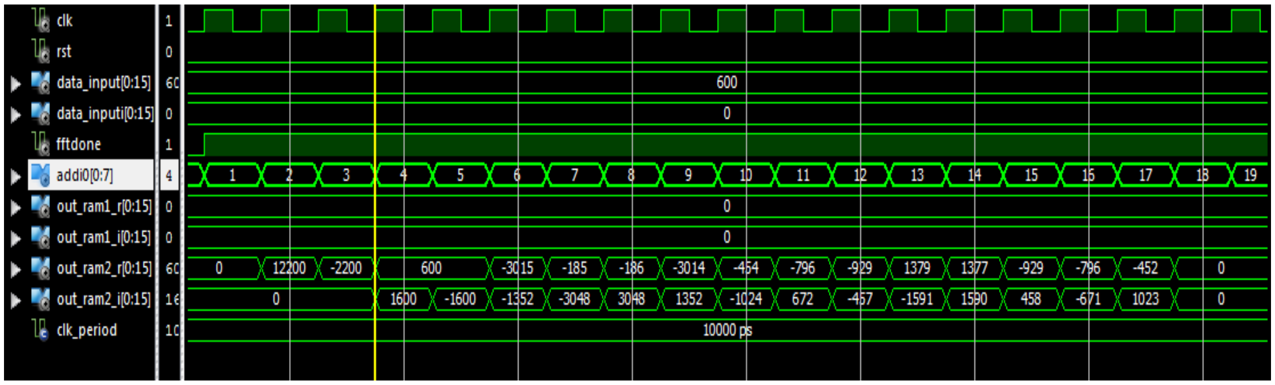
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0



Input of the 32 point SRFFT



Rest of the 32 point input SRFFT



Output of the 32 point

APPENDIX H PUBLICATIONS

[C1] **H. Abdollahifakhr**, N. Bélanger, Y. Savaria, F. Gagnon, “Power-Efficient Hardware Architecture for Computing Split-Radix FFTs on Highly Sparsed Spectrum”, IEEE International NEW Circuits And Systems, Grenoble, France, June 2015, (*Submitted and Accepted*)