Designing 64-Bit FFT Structures for High-Speed DSP Applications

A Project report submitted to

JAWAHARLAL NEHRU TECHONOLOGICAL UNIVERSITY ANANTAPUR
ANANTHAPURAMU

in partial fulfilment of the requirements for
the award of the degree of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Gajula Amrutha Sai	**- 19121A0459**
Besta Divya Sree	**- 19121A0424**
Bharadwaj Karthik K	**- 19121A0426**
Bhandoth Uday Kiran Naik	**- 19121A0425**

Under the Guidance of

Ms. M. Bharathi, M.Tech., (Ph.D.)
Assistant Professor, Department of ECE

Department of Electronics and Communication Engineering

SREE VIDYANIKETHAN ENGINEERING COLLEGE

(AUTONOMOUS)

Sree Sainath Nagar, A. Rangampet, Tirupathi - 517102.

(2019-2023)

SREE VIDYANIKETHAN ENGINEERING COLLEGE

(AUTONOMOUS)

Sree Sainath Nagar, A.Rangampet - 517 102

VISION

To be one of the Nation's premier Engineering Colleges by achieving the highest order of excellence in Teaching and Research.

MISSION

- To foster intellectual curiosity, pursuit and dissemination of knowledge.
- To explore students' potential through academic freedom and integrity.
- To promote technical mastery and nurture skilled professionals to face competition in ever increasing complex world.

QUALITY POLICY

Sree Vidyanikethan Engineering College strives to establish a system of Quality Assurance to continuously address, monitor and evaluate the quality of education offered to students, thus promoting effective teaching processes for the benefit of students and making the College a Centre of Excellence for Engineering and Technological studies.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

Vision

To be a center of excellence in Electronics and Communication Engineering through teaching and research producing high quality engineering professionals with values and ethics to meet local and global demands.

Mission

- The Department of Electronics and Communication Engineering is established with the cause of creating competent professionals to work in multicultural and multidisciplinary environments.
- Imparting knowledge through contemporary curriculum and striving for development of students with diverse background.
- Inspiring students and faculty members for innovative research through constant interaction with research organizations and industry to meet societal needs.
- Developing skills for enhancing employability of students through comprehensive training process.
- Imbibing ethics and values in students for effective engineering practice.

B. Tech. (Electronics and Communication Engineering)

Program Educational Objectives

After few years of graduation, the graduates of B.Tech (ECE) will be:

PEO1. Enrolled or completed higher education in the core or allied areas of electronics and communication engineering or management.

PEO2. Successful entrepreneurial or technical career in the core or allied areas of electronics and communication engineering.

PEO3. Continued to learn and to adapt to the world of constantly evolving technologies in the core or allied areas of electronics and communication engineering.

Program Outcomes

On successful completion of the Program, the graduates of B.Tech. (ECE) Program will be able to:

PO1 **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2 **Problem analysis**: Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3 **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4 **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5 **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6 **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7 **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8 **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9 **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10 **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11 **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12 **Lifelong learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

On successful completion of the Program, the graduates of B. Tech. (ECE) will be able to

PSO1. Design and develop customized electronic circuits for domestic and industrial applications.

PSO2. Use specific tools and techniques to design, analyse and synthesize wired and wireless communication systems for desired specifications and applications.

PSO3. Apply suitable methods and algorithms to process and extract information from signals and images in Radar, Satellite, Fiber optic and Mobile communication systems.

Department of Electronics and Communication Engineering

SREE VIDYANIKETHAN ENGINEERING COLLEGE

(AUTONOMOUS)

Sree Sainath Nagar, A.Rangampet - 517102.

Certificate

This is to certify that the Project Report entitled

"DESIGNING 64-BIT FFT STRUCTURES FOR HIGH-SPEED DSP APPLICATIONS"

is the bona fide work done and submitted by

Gajula Amrutha Sai	**- 19121A0459**
Besta Divya Sree	**- 19121A0424**
Bharadwaj Karthik K	**- 19121A0426**
Bhandoth Uday Kiran Naik	**- 19121A0425**

in the Department of Electronics and Communication Engineering, ***Sree Vidyanikethan Engineering College, A.Rangampet*** *affiliated to* **Jawaharlal Nehru Technological University Anantapur, Ananthapuramu** *in partial fulfilment of the requirements for the award of degree of* **Bachelor of Technology** *in* **Electronics and Communication Engineering** *during 2019-2023.*

GUIDE

Ms. M. BHARATHI, M.Tech., (Ph.D.)
Assistant Professor, Dept. of ECE

HOD

Dr. V. VIJAYA KISHORE, M.Tech., Ph.D.,
Professor & Head, Dept. of ECE

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENTS

First of all, we are indebted to the GOD ALMIGHTY for giving us an opportunity to excel in our efforts to complete this project on time.

Our heartfelt gratitude to our seminar guide **Ms. M. Bharathi, M.Tech., (Ph.D.),** Assistant Professor, Department of Electronics and Communication Engineering, for her valuable suggestions and guidance in the preparation of the Project Thesis.

We express our heartfelt thanks to **Dr. V. Vijaya Kishore**, **M.Tech., Ph.D.,** Professor and Head, Department of Electronics and Communication Engineering, for his kind attention and valuable guidance to us throughout the Project Course.

We owe our gratitude to **Dr. B. M. Sathish**, **Ph.D.,** Principal, Sree Vidyanikethan Engineering College for permitting us to use the facilities available to accomplish the Project course.

We are also thankful to all the teaching and non-teaching staff of Electronics and Communication Engineering Department for their cooperation.

Gajula Amrutha Sai - 19121A0459
Besta Divya Sree - 19121A0424
Bharadwaj Karthik K - 19121A0426
Bhandoth Uday Kiran Naik - 19121A0425

ABSTRACT

Well-established technology like Bluetooth is used to provide wireless connectivity from wireless headphones to mobile phone, laptop and mice of the wireless computer to a lot of other devices which need a short-range connectivity. Our spotlight is to reduce complexity for Bluetooth to find channels free from the use of other devices (in 2.4 GHz ISM band). Bluetooth uses 79 RF channels in this band, each 1 MHz wide. Channels availability can be detected by comparing Power Spectral Density (PSD) with a threshold. To reduce the implementation complexity, the PSD is calculated by means of Fast Fourier Transform (FFT) and linear interpolation by exploiting spectrum characteristics. A PSD is computed by multiplying each frequency bin in an FFT by its complex conjugate. All the complex multiplications required for this FFT are implemented using Distributed Arithmetic (DA) technique. Distributed Arithmetic (DA) takes the value in bit serial fashion which suits for signal processing applications. DA is a technology for multiplier-less implementation of inner dot product of two vectors. DA recognizes some frequently used values of a Multiply and Accumulate (MAC) operation, precomputes these values and stores them in a Look-up Table (LUT).Several applications of FFT are medical imaging, signal processing and it is also used in digital formats such as JPEG, MP3, H.264. This project reveals a new integration approach for computing using DA. The proposed design will be implemented for 64-point Butterfly structure by incorporating DA technique. Finally, the results of the proposed Butterfly structure will be compared with the existing Butterfly structure of equal length in terms of power, area and delay parameters. The designs can be evaluated using Xilinx.

Keywords: Distributed Arithmetic (DA), Power Spectral Density (PSD), Fast Fourier Transform (FFT), Look-up Table (LUT)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Seismology, audio, speech processing, radar, sonar, voice recognition, and various financial signals are among the applications where DSP is most commonly utilised. For instance, speech transmission and compression for mobile phones are both done using digital signal processing. The same suppression and augmentation approach is equally crucial in this case. DSP is also utilised in high-end headphone equipment to prevent users from hearing damage. Digital signal processing is used by top businesses in the workplace communication and hearing protection sectors, like Sensear, to produce safe, high-quality communication. Other uses include CAT scans, MRI mp3 file manipulation, and even electric guitar amplifiers in some cases. The goal of digital signal processing is to remove analogue signals from the present moment in time and space. It is a crucial component of communication equipment that uses noise suppression and voice enhancement, though it is utilised in a wide range of modern devices.

Transferring data to and from memory is one of the main bottlenecks in the execution of DSP algorithms. This contains both programme instructions—the binary codes that are fed into the programme sequencer—and data—such as samples from the input signal and the filter coefficients. Consider the situation when we need to multiply two numbers that are stored in memory. To accomplish this, we must retrieve three binary values from memory, the multiplicands, and the programme instruction. This simple task is executed using a traditional microprocessor which is often called Von Neumann architecture.

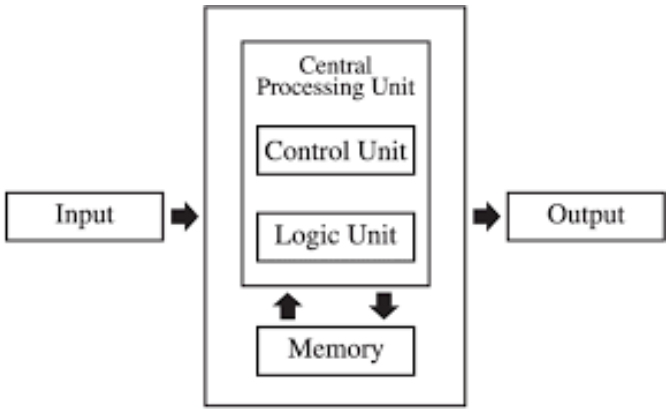


Figure 1.1 Von Neumann Architecture

A single memory and a single bus are used in Von Neumann architecture to transmit data into and out of the central processor unit (CPU). Other architectures are only necessary when extremely quick processing is required and the cost of added complexity is acceptable. This brings us to the Harvard architecture.

The Harvard Architecture is a type of computer architecture that has distinct instruction and data buses (signal paths) and storage. Basically, it was created to overcome the Von Neumann Architecture bottleneck. The fundamental benefit of having distinct buses for instructions and data is that the CPU can access instructions while simultaneously reading and writing data.

As mentioned above DSP processors are widely used in various applications. The overall chapter addresses different types of DSP processors that utilise VLSI and the Distributed Algorithm (DA), whose multiplierless implementation is carried out by these DSP processors. The structure of VLSI-DSP processors is detailed in section 1.2, while section 1.3 uses DSP algorithms for FFTs. Section 1.4 discusses the many challenges for FFT and how to overcome them, as well as the motivation for our project. The objectives of the DSP processor are described in part 1.5, and section 1.6 describes how the thesis is organised.

1.2 VLSI-DSP FRAMEWORK

The area-power-speed trade-offs for various DSP applications will be investigated using the VLSI architectural design approaches. The ability to process high-speed data is only one constraint for VLSI implementations; another is minimising the amount of space and power used. Digital signal processing in real time has just lately been practical. This is due to the high levels of computational throughput required for the implementation of digital processing techniques, especially for real-time applications. Due to this demand and the continuously rising performance levels of VLSI, devices like the TMS320 and the DSP56001, which are VLSI digital signal processors, have been created.

In DSP design, more algorithm-based structures are becoming more popular. In other words, the simplicity of current VLSI design encourages the designer to create more specialised designs. As was previously said, FFT computing, FIR, and IIR digital filters are among the most practical methods for digital signal processing. There are only three different operations needed to apply these techniques. Storage, multiplication, and addition are necessary processes. The usage of a repeated modular architecture is suggested by the few processes needed.

A number of VLSI-oriented special-purpose architectures for digital signal processing applications have been developed as a result of the few different operations that must be performed and the way that VLSI technology is manufactured. These architectures achieve extremely fast processing rates by using pipelining, array processors, reduced instruction set computers (RISC), multiprocessing, and parallel processing. In DSP design, more algorithm-based structures are becoming more popular. In other words, the simplicity of current VLSI design encourages the designer to create more specialised designs. As was previously said, FFT computing is the most practical methods for digital signal processing. There are only three different operations needed to apply these techniques. Storage, multiplication, and addition are necessary processes. These architectures achieve extremely fast processing rates by using pipelining, array processors, reduced instruction set computers (RISC), multiprocessing, and parallel processing.

Multiple processors work together in parallel processing, also known as multiprocessing, to solve problems through concurrent execution. Pipelining is merely a multiprocessing enhancement that maximises resource use and benefits from interdependencies between calculations. An array processor is typically a two-dimensional array of processors through which data is passed. The data is routed through the array in the most effective way possible using pipelines.

1.3 CHARACTERISTICS OF DSP

Devices together referred to as DSP hardware are used to process the various signals. The hardware for transmitting signals, several devices for enhancing or filtering the signals, analogue to digital and digital to analogue converters, and other processing devices like computers are all included here.

Microprocessors known as digital signal processors (DSPs) include the following features:

1. Capabilities for real-time digital signal processing. Since real-time data processing is the norm for DSPs, the timing of the data processing's completion has a significant impact on how accurately an operation is performed.
2. A lot of output. DSPs are capable of processing high-speed streaming data, including audio and multimedia.
3. Operational determinism. DSP programmes' execution times can be predicted with accuracy, ensuring a predictable, desired performance.

4. Software reprogrammability. Instead of making hardware changes, a different system behaviour could be achieved by re-coding the algorithm that the DSP uses.
5. They can function at their best even when streaming data is put into them since they are set up to handle real-time processing.
6. The memory used to store programmes and data are distinct from one another.
7. They don't offer hardware that can handle many tasks.
8. It can be used in supporting or host settings as a direct memory access device.
9. The signals are input in analogue form, converted to digital form, processed, and then specifically specified back into analogue form.

1.4 APPLICATIONS OF DSP

1. Analog signals are recorded while audio or music is played back. The signal is turned into a digital signal by the ADC. The digitised signal is input into the digital processor, where it is processed and stored. The digital processor decodes the recorded data on playback. For human hearing, a DAC converter converts the signal to analogue. The digital processor additionally enhances quality through equalisation, noise reduction, and volume enhancement.

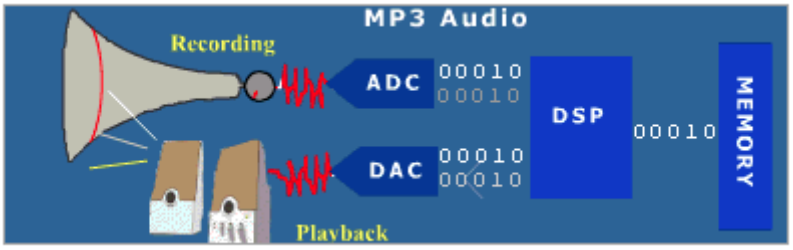


Figure 1.2 An application of DSP

2. Modern laptops and computers with digital processors offer greater flexibility, speed, quality, and portability. Through a cable, the graphics card transmits the digital signals from the computer to the digital display. For human sight, the graphics card transforms digital signals into analogue signals and sends them to an analogue display.
3. Smartphones, IPADs, iPods, and other digital devices all contain processors that accept user input, convert it to digital form, process it, and output the results in a way that is understandable to humans.

4. We use a variety of digital appliances every day, including refrigerators, microwave ovens, and washing machines. The dashboard, GPS, music player, and other devices in cars all depend on digital processors.

1.5 CHALLENGES AND MOTIVATIONS

The computational complexity of the Fast Fourier Transform (FFT) technique, which employs butterfly structures, is O(Nlog(N)), which is significantly less than O(N2). However, FFT computations still present significant difficulties in terms of computational complexity in terms of performance, time requirements, and energy consumption in light of the introduction of large data in many applications in recent years.

The twiddle factors must be stored in a considerable amount of space because they are kept in a ROM memory and used for additional multiplications. In order to overcome this a ROM-free twiddle factor generator, which uses straightforward accumulators, shifters, registers, and adders and is also a changeable architecture.

The path delay in current FFT architectures is growing as a result of multipliers in DSP processors. To get around this, we employ pre-calculated values in the lookup tables and this method is known as distributed arithmetic, which enables the multiplier-less implementation of various FFT structures.

Another motivation for FFT is exceedingly high computing load of calculating the DTFS is the primary driving force for the FFT. The quantity of calculations required is greatly decreased by taking use of the symmetry and periodicity characteristics of the twiddle factor.

Power consumption is currently one of the key components in the creation of DSP processors. Dynamic power and static power are the two parts of power usage. Depending on the operation being carried out, dynamic power consumption is extremely changeable while static power consumption is constant. When compared to the previous design, the dynamic power has been reduced in our proposed design. Consequently, the power as a whole decrease.

1.6 OBJECTIVES

1. To implement 64-bit Fast Fourier Transform (FFT) structures using LUT methodology for high-speed applications.
2. To develop Fast Fourier Transform (FFT) using Distributed Algorithm (DA).

3. To reduce the power, utilization and delay parameters for DSP processors compared with conventional FFT algorithm.
4. To design a novel 64-bit FFT structures using adder-based approach.

1.7 ORGANIZATION OF THESIS

This thesis is organized and presented in 8 chapters

Chapter 1: This chapter explains about the introduction and also the different types of architectures in DSP processors and the framework of VLSI-DSP and how the DSP processors are fabricated using VLSI. This chapter also introduces the characteristics and applications of DSP.

Chapter 2: This chapter explains about the detailed review of literature. A literature review is used to offer information that has been abstracted regarding the various methods currently in use. The literature evaluation is thorough since it includes all element required for the research inquiry regarding FFT and DA.

Chapter 3: This chapter explains the significance of FFT and different algorithms in FFT and type of algorithm that is used in FFT, it explains about the existing FFTs. It also explains about the different points in FFT in radix-2.

Chapter 4: This chapter explains the proposed method of the project. It explains about Distributed Arithmetic (DA). In the Look Up Tables (LUT) and LUTless two, four, eight, sixteen, thirty-two, sixty-four point FFT based structures are explained.

Chapter 5: In this chapter here, we compare the both existing and proposed of two, four, eight, sixteen, thirty-two, sixty-four point FFT. The proposed is FFT based on DA and the existing is normal FFT. Here the results are based on power and delay and also on the other parameters.

Chapter 6: In this chapter here, we explain about the project management. The time management explains about the literature survey, problem identification, implemenatation and completion of the project

Chapter 7: In this chapter we conclude the project by comparing the existing and proposed methods. The proposed results are better in terms of power and delay.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

A detailed literature review is presented in this chapter[1]. The literature review is conducted to address the issue of reducing Bluetooth's channel-finding complexity that are free from the use of other devices. Power spectral density (PSD) and a threshold can be compared to determine the availability of channels. PSD can be estimated using the Fast Fourier Transform (FFT) to simplify implementation. In this study, the Distributed Arithmetic (DA) technique is used to implement all of the complex multiplications required for the FFT. The Look-up Table contains the values that DA needs for a multiply and accumulate (MAC) operation on a regular basis. This DA approach implements this concept for a 64-bit butterfly structure.

2.2 FFT

Su Min Cho et al. (2021) suggested FIR filters, which offer 73.5% less area-delay product (ADP) than the current systolic designs, can enable very high throughput of over 1.85 Giga samples per second[2]. Through careful study, it was possible to approximate the propagation delay of the FIR filter in terms of unit gate delay, which made it possible to develop an effective pipelining approach at the gate level. Due to the fine-grained seamless pipelining, the suggested full-parallel system can achieve very high throughput. To evaluate the maximal throughput and the trade-off between complexity and speed, these FIR filters were created. In comparison to the current design, the suggested single multiplier-accumulator (MAC) based FIR filter has a 3 times greater throughput, a 26.0% smaller area, and 75.8% less ADP.

Seyed Hadi Mirfarshbafan et al. (2021) suggested beamspace processing is a new concept that as a way to simplify the technology in all-digital mmWave massive multiple-input multiple-output (MIMO) base stations[3]. This method takes advantage of the mmWave channels' sparsity yet calls for spatial discrete Fourier transforms (DFTs) performance across the antenna array, which is required at the sampling rate for baseband. We suggest a fully-unrolled DFT to lessen the associated DFT hardware implementation bottleneck. Fourier Transform with Streaming multiplierless (SMUL) speed (FFT) engine with one transformation every clock cycle. By limiting the twiddle factors to a sum-of-powers-of-two, the SMUL-FFT architecture avoids hardware multipliers, leading in significant power and space savings.

Vikas Kumar et al. (2021) designed a hardware-efficient flexible filtering and spectrum shifting design for UFMC transmitter[4]. The suggested design demonstrates versatility in that it can produce filter and spectrum shifting co-efficients for externally selected IDFT-size, number of subbands, number of subcarriers in a subband, and pulse shaping filter-length without modifying hardware resources. The results of the simulation are supported by the experimental baseband signal. Furthermore, the spectrally shifted filter coefficients produced by the suggested architecture agree exactly with the outcomes of the simulation. In comparison to state-of-the-art, the suggested flexible filtering filtering and spectrum shifting architectue gives a unique flexibility to UFMC transmitters through its flexibility, hardware efficiency, and reuse of a number of hardware components. Pipelining at the data and process levels can be utilised to conduct additional research and boost throughput.

Mario Garrido et al. (2021) proposed a method for calculating the total number of optimal multi-path delay commutator (MDC) fast fourier transform (FFT) designs in terms of delays and multiplexers[5]. This method is based on identifying the FFT stage order that results in the least number of delays and multiplexers. The findings indicate that there are several different optimum MDC FFTs. In the future, this vast design space can be investigated to create effective MDC architecture that not only optimises the number of delays and multiplexers, but also other figures of merit like the quantity of rotators or the order of the input/output data.

Sarita Chauhan et al. (2020) applied a high-performance, energy-efficient Fast Fourier Transform architecture with reduced delay and power usage[6]. The Time-Fast Fourier Transform's decimation architecture is discussed, along with the design and compute module improvements that are necessary to produce the desired low power and low delay outcomes. Using vedic multipliers, an optimal constant complex multiplier for twiddle factor multiplication is created. The suggested multiplier differs just slightly from the current complicated multipliers. When used with FFT, the provided complex multiplier's minimal complexity results in significantly reduced delay and simulation time, which slows down overall speed.

Jian Wang et al. (2020) deduced a conflict-free data-access mechanism for the memory-based radix-2k FFT processor[7]. Based on this, a unique FFT processor was presented that utilised single-port RAMs. To allow arbitrary power-of-2 computing parallelism, the memories were combined into four banks. By expanding I/O parallelism and enabling concurrent I/O operation, the proposed architecture could speed up computing, and its advantages in terms of area, throughput, and power had been thoroughly established through theoretical assessment

and hardware implementation. It should be noted that the FFT processor presented in this article operated under the assumption that the data length was a power of 2. We will broaden the debate to include the mixed-radix example for emerging cellular networks.

Kevin Bowlyn et al. (2020) proposed in the execution of numerous Digital Signal Processing (DSP) algorithms, complex integers are essential[8]. The techniques increased the quantity of computations because they are represented as two distinct components (real and imaginary). By combining the computation of the radix-2 Fast Fourier Transform (FFT) with the Distributed Arithmetic (DA) and Complex Binary Amount System and provided an improved method in this letter that minimises the number of computations (CBNS). Additionally, the proposed architecture drastically reduces the design space by switching out multipliers for adders and shifters. Results show that the suggested design produces better results than the conventional radix-2 FFT and DA or CBNS-based FFT algorithms, even though more adders are required to execute the novel DA-CBNS architecture.

Zhihao Li et al. (2020) developed as the outer parallelization framework, AutoFFT, which is based on the Cooley-Tukey FFT technique and takes advantage of the symmetric and periodic qualities of the DFTmatrix[9]. In order to further reduce the number of floating-point operations for butterflies of any arbitrary natural numbers, explore additional symmetric and periodic properties of the DFT matrix and develop numerous optimised calculation templates. Butterflies are the core operations of the Cooley-Tukey algorithm. We incorporate a number of optimizations in an assembly template optimizer to fully utilise hardware resources. With each DFT problem, AutoFFT automatically creates C FFT kernels using these calculation templates, then uses the template optimizer to transform them into effective assembly kernels.

Tong Xu et al. (2020) demonstrated the use of DA-based resampling filters for frequency translation and channel selection in digital subcarrier cross-connect (DSXC)[10]. The DA algorithm creates look-up tables, which require only digital memories, which are typically more plentiful, as opposed to classic FIR filters, which are based on multipliers and DSP slices are to be implemented in FPGA. In order to implement DSXC, which requires being able to switch multiple digital subcarrier channels from any input to any output port, these filters offer a hardware resource-efficient solution. Additionally, when compared to a multiplier-based FIR filter with the same transfer function, a DA-based resampling filter has lower processing latency.

Dingli Xue et al. (2019) proposed linear convolution technique built on the Discrete Hirschman Transform (DHT) was created. In terms of computing complexity, it outperforms the conventional convolution based on the Discrete Fourier Transform (DFT)[11]. For this DHT convolution algorithm, we suggest a linear convolution filter implementation utilising the Distributed Arithmetic (DA) method. In order to assess its hardware performance, we have realised it in FPGAs. Simulation findings show that our suggested DHT convolution filter has superior accuracy than the conventional DFT convolution filter using the same DA approach. Additionally, it can reduce latency by about 25%, real computations by more than 23%, and memory accesses by about 23%.

Safwat Mostafa Noor et al. (2019) proposed the concept of hashing and lookup table to effectively reduce the number of arithmetic operations required to perform the FFT of an electrocardiogram (ECG) signal[12]. Here, it lowers the FFT computation's overall energy footprint and dynamic power consumption. A 90-nm standard cell library is used to synthesise the design, and gate level switching activity is simulated to produce precise power consumption figures. When evaluated with genuine ECG data obtained from Physio Net, the proposed optimizations produced a low energy usage of 27.72 nJ per FFT, which is 14.22% lower than a conventional 128-point radix-2 FFT.

Shaohan Liu et al. (2019) designed a fast Fourier transform (FFT) processor with high throughput that supports 12- to 2400-point discrete Fourier transformations and 16- to 4096-point FFTs has been developed (DFTs)[13]. Several improvements are offered to construct a high-speed CPU that is hardware-efficient. A reconfigurable butterfly unit is suggested to provide computation that includes eight radix-2 in parallel, four radix-3/4 in parallel, two radix-5/8 in parallel, and a radix-16 in one clock cycle in order to maximise the reuse of the hardware resource. Twiddle factor multipliers are adjusted and compared using several strategies, and a modified coordinate rotation digital computer scheme is finally used to reduce hardware costs while enabling both FFTs and DFTs. Additionally, a conflict-free data access strategy that is optimised for many butterflies at all radices is suggested.

Xin-Yu Shih et al. (2018) proposed design concept is based mostly on integrated radix-5, radix-32, and radix24 single-path delay feedback FFT design methodologies and also created three design strategies to further our hardware design, including the reconfigurable processing kernel with seven types (RPK-ST), an effective FIFO management system, and the single-table approximation method[14]. This 46-mode reconfigurable FFT chip uses TSMC 40-nm CMOS

technology, has a core area of 0.36 mm2, dissipates 48.46 mW, and operates at a clock frequency of up to 500 MHz. The work provides high-quality design findings in the areas of area- and energy-related performance indices, giving a useful FFT design prototyping for 3GPP-LTE systems in comparison to another cutting-edge research.

Mayura Patrikar et al. (2017) simulated and synthesized of 2- and 8-point FFT designs utilising the Radix-2 algorithm are complete[15]. The design of the 2-bit and 8-bit radix-2 DIT FFT is demonstrated in this paper and is implemented in the FPGA. The simulation of the 2-bit radix-2 FFT is complete, and the results are available. The output of the 2 bit and 8-bit radix-2 FFT power results are obtained. Two sorts of power are the outcomes. They are power, both static and dynamic.

Á. Ordóñez, et al. (2017) proposed GPU-accelerated technique for the registration of hyperspectral images based on the Fourier transform (HYFM)[16]. The PCA, MLFFT, combination of log-polar maps, and peak processing-based technique was created to run totally in the GPU and effectively utilise the GPU architecture. The FFT computation-intensive steps are the most expensive in terms of execution time and GPU optimization. Phase correlation and high-pass filters have experienced the greatest speedups for MLFFT. According to the findings, even for huge photos, cheap GPUs provide a sufficient foundation for performing efficient registration.

Diego F.G. Coelho et al. (2017) suggested with arbitrarily high precision, the Gauss-Eisenstein representation converts complex numbers into 4-tuples of integers[17]. The representation allows the computation of the discrete Fourier transform (DFT) at any desired accuracy for 3, 6, and 12 points. Up until the last reconstruction step, which can be implemented in hardware as a multiplierless implementation, the associated fast algorithms based on the Gauss-Eisenstein integers are error-free. The arithmetic complexity of the introduced approaches and those of competing algorithms is contrasted.

Donald Donglong Chen et al. (2016) proposed the fundamental operation of public-key cryptography algorithms like RSA and the Diffie-Hellman algorithm is modular multiplication[18]. The effectiveness of the modular multiplier is essential to how well these cryptographic techniques work. This study presents carry-save arithmetic and pre-computation strategies to enhance the FFT-based Montgomery Modular Multiplication (FFTM3). Additionally, the permitted operand sizes for the FFTM3 are enriched using the pseudo-Fermat number transform. This approach has the same asymptotic complexity as the Schonhage-

Strassen multiplication algorithm, which is O(l log l log log l) (SSA). A methodical method for choosing a good parameter set for the FFTM3 is offered. It offers better computation latency and area-latency product compared to the state-of-the-art methods for operand size of 3072-bit and above.

CHAPTER 3

FFT IN DSP PROCESSORS

3.1 SIGNIFICANCE OF FFT IN DSP

Fast Fourier Transform is a computer algorithm used in digital signal processing (DSP) to modify, filter and decode digital audio, video and images. FFTs are also used for fault analysis, quality control, and condition monitoring of machines or systems. In the highly networked, digital world of today, the FFT is employed to process data. It enables computers to quickly determine the various frequency components present in time-varying signals as well as to reconstruct those signals using a selection of frequency components.

Most efficient method for performing DFT is Fast Fourier Transform (FFT). No algorithm for computing the DFT could have a smaller complexity than FFT. With the introduction of specialized DSP blocks embedded into programmable architectures the efficiency of FFT is limited by the speed of hardware multipliers of DSP modules. Programmable architectures provide possibility to increase the performance of digital system by parallelism (bit serial fashion algorithm).

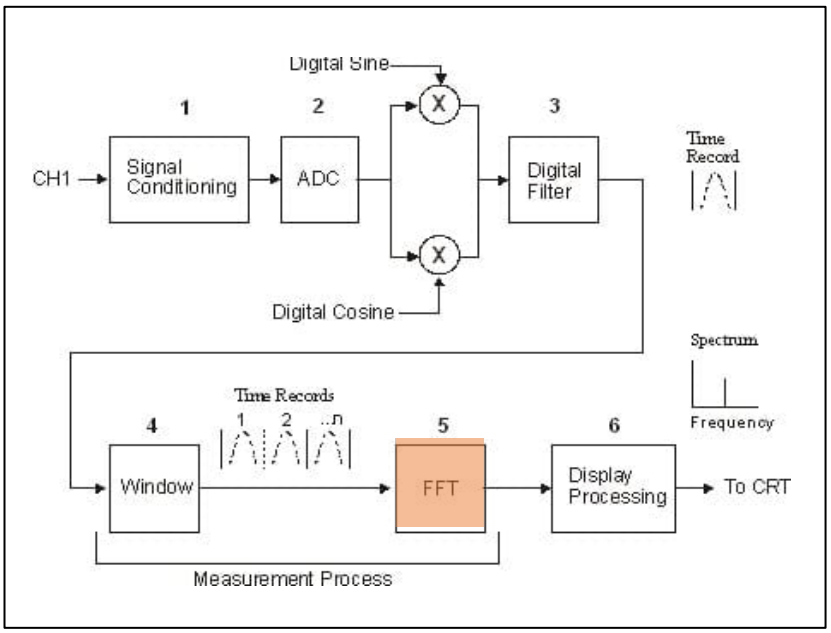


Figure 3.1 Block diagram of FFT analyser

The two endpoints of a time waveform are interpreted by the FFT computation as though they are connected and have the same value at both ends of the time period. Random signals aren't frequently equal at either end of the time period, though. This presumption leads to undesirable noise in the frequency spectrum and breaks up the time domain.

The computational portion of older DFT approaches is excessively lengthy. So, to reduce that fast Fourier transform, sometimes known as FFT, can be used for this. We can therefore conclude that FFT is nothing more than the computation of the discrete Fourier transform in an algorithmic manner, where the computational part will be minimised. The ability to create FIR filters is the main benefit of having FFT.

The FFT works by breaking down a time domain signal with N points into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum.

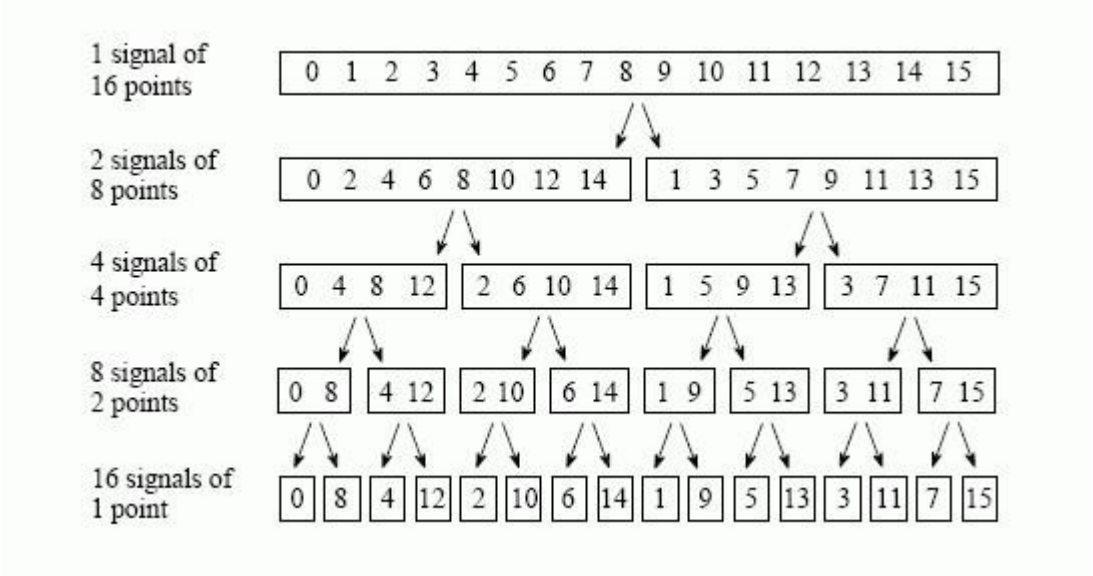


Figure 3.2 Time Domain decomposition used in FFT

Here a 16-point signal is decomposed through four separate stages. The first stage breaks the 16-point signal into two signals each consisting of 8 points. The second stage decomposes the data into four signals of 4 points. This pattern continues until there are N signals composed of a single point. An interlaced decomposition is used each time a signal is broken in two, that is, the signal is separated into its even and odd numbered samples. The best way to understand this is by inspecting Figure. 3.1. There are Log2N stages required in this

decomposition, i.e., a 16-point signal requires 4 stages, a 512-point signal requires 7 stages, a 4096-point signal requires 12 stages, etc.

3.2 ALGORITHMS FOR IMPLEMENTING FFT

There are two different algorithms of implementing the Fast Fourier Transform (FFT).

- Decimation in Time
- Decimation in Frequency

In DIT the input is bit reversal and the output is in natural order whereas in DIF the input is natural order and the output is bit reversal.

DIT algorithm is used to calculate the DFT of a N-point sequence. The idea is to break the N-point sequence into two sequences, the DFTs of which can be obtained to give the DFT of the original N-point sequence. Initially the N-point sequence is divided into N/2-point sequences Xe(n) and Xo(n), which have even and odd numbers of x(n) respectively. The N/2-point DFTs of these two sequences are evaluated and combined to give the N-point DFT. Similarly, the N/2-point DFTs can be expressed as a combination of N/4-point DFTs. This process is continued until we are left with two-point DFT. This algorithm is called decimation-in-time because the separate x(n) is often split into smaller sequences.

Figure 3.3 Butterfly diagram of DIF FFT and DIT FFT

In DIF the original sequence s(n) is divided into two sub sequences as the first half and second half of a sequence in the Radix-2 decimation-in-frequency (DIF) FFT algorithm. As with the Radix-2 decimation-intime (DIT) FFT algorithm, the original sequence does not need to be rearranged (shuffled). The N-point time domain sequence is split into two N/2 sequences using this approach. Then, two numbers of N/4-point sequences are created from each N/2 point sequence. Thus, four N/4-point sequences are obtained. We repeat this method until we get N/2 2-point sequences. It can be demonstrated that two sets of N/2 point DFTs can be used to create the N-point DFT of x(n). From two different sets of N/4-point DFTs, the N/2 point DFTs can be realised. Here we use DITFFT method in our project.

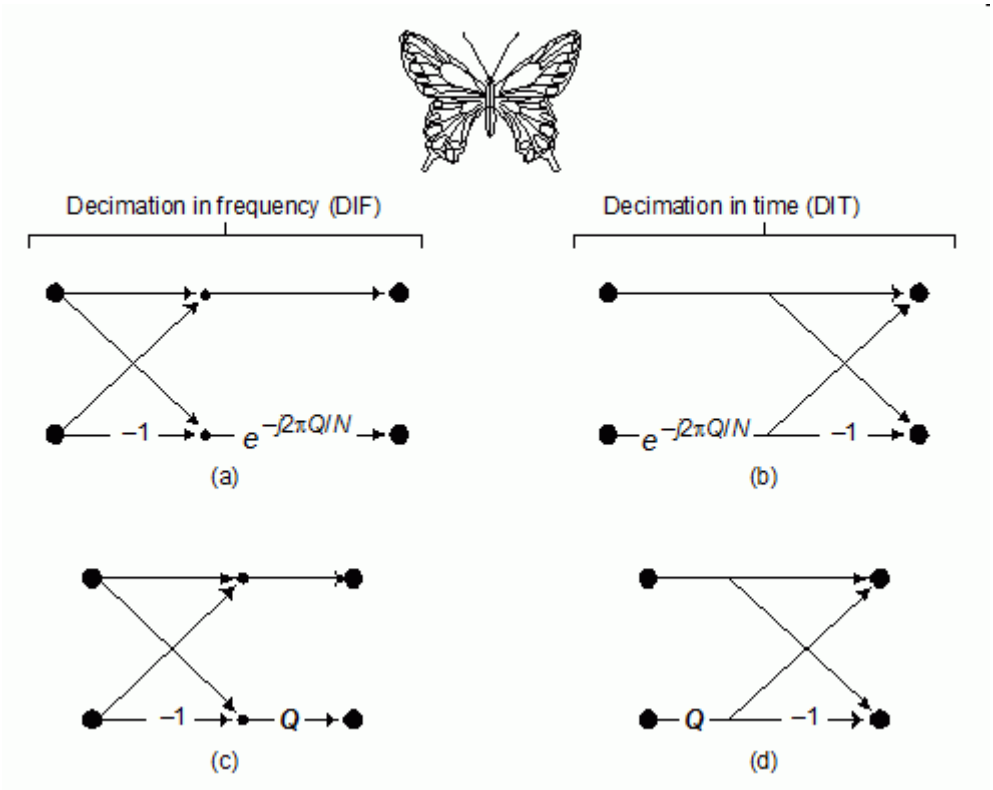


Figure 3.4 Computing FFT Twiddle Factors

3.3 DECIMATION IN TIME FFT

Radix 2 FFT: If N is a power of 2 and N=2m, then it is a radix-2 FFT. For example, if N = 4 = 22, the sequence a(n) above can be decimated into two N/2 sequences. The first sequence contains samples a (0), a (2) and the second sequence contain sample a (1) (3). Starting from the basics, we can create a radix-2 DIT algorithm to evaluate the N=4 DFT.

Decimation in Time (DIT) Radix 2 FFT algorithm converts the time domain N point sequence x(n) to a frequency domain N-point sequence X(k). In Decimation in Time algorithm the time domain sequence x(n) is decimated and smaller point DFT are performed. The results of smaller point DFTs are combined to get the result of N-point DFT. In DIT radix -2 FFT the time domain sequence is decimated into 2-point sequences. For each 2-point sequence,2 -point DFT can be computed. From the result of 2-point DFT the 4-point DFT can be calculated. From the result of 4-point DFT the 8- point DFT can be calculated. This process is continued until we get N point DFT. This FFT algorithm is called radix-2 FFT. In decimation in time algorithm the N point DFT can be realized from two numbers of N/2 point DFTs, The N/2 point DFT can be calculated from two numbers of N/4-point DFTs and so on.

Let x (n) be N sample sequence, we can decimate x (n) into two sequences of N/2 samples. Let the two sequences be f1 (n) and f2 (n). Let f1 (n) consists of even numbered samples of x (n) and f2(n) consists of odd numbered samples of x(n).

f 1(n) = x (2n) for n=0,1,2,3……N 2 – 1

f 2(n) = x (2n +1) for n=0,1,2,3……N 2 - 1

Let X (k) = N-point DFT of x (n)

F1 (k) = N/2 point DFT of f1(n)

F2 (k) = N/2 point DFT of f2(n)

By definition of DFT the N/2 point DFT of f1(n) and f2(n) are given by

F1 (k) = $\sum f1\ N\ 2 - 1\ n$=0 $(n)WN/2\ kn$;

F2 (k) = $\sum f2\ N\ 2 - 1\ n$=0 $(n)WN/2\ kn$;

Now-point DFT X (k), in terms of N/2 point DFTs F1 (k) and F2 (k) is given by

X (k) = F1 (k) + Wk N F2(k), where, k= 0,1, 2……. (N-1)

Having performed the decimation in time once, we can repeat the process for each of the sequences f1 (n) and f2 (n). Thus f1(n) would result in the two N/4-point sequences and f2(n) would result in another two N/4-point sequences. Let the decimated N/4-point sequences of f1 (n) be V11 (n) and V12 (n).

V11 (n) = f1 (2n); for n= 0, 1, 2…. N 4 – 1

V12 (n) = f1 (2n+1); for n= 0,1, 2…. N 4 – 1

Let the decimated N/4-point sequences of f2(n) be V21(n) and V22(n).

V21(n) = f1 (2n); for n= 0,1, 2... N 4 - 1

V22(n) = f1 (2n+1); for n= 0,1, 2…. N 4 – 1

Let V11 (k) = N/4-point DFT of V11 (n);

V12 (k) = N/4-point DFT of V12 (n)

V21(k) = N/4-point DFT of V 21(n)

V22(k) = N/4-point DFT of V 22(n)

Then like earlier analysis we can show that,

F1 (k) = V11 (k) + Wk N/2 V12 (k); for k = 0,1,2, 3……*N* 2 -1

F2 (k) = V21 (k) + Wk N/2 V 22(k); for k = 0,1,2, 3……*N* 2 -1

Hence the N/2 point DFTs are obtained from the results of N/4-point DFTs. The decimation of the data sequence can be repeated again and again until the resulting sequences are reduced to 2-point sequences.

Radix 4 FFT: If N is a power of 4 and N= 4m, it will be a radix-4 FFT. For example, if N = 16 = 42, then the given sequence a(n) can be decimated into four N/4 sequences. The first sequence has samples a(0), a(4), a(8), a(12), the second has samples a(1), a(5), a(9), a(13), the third contains samples a(2), a(6), a(10), a(14), the fourth of these contains samples a(3), a(7), a(11), containing a(15). Starting from the basics, we can create a radix-4 DIT and DIF algorithm to evaluate the N=16 DFT.

Radix 8 FFT: If N is a power of 8 and N=8m, it will be a radix-8 FFT. For example, if N = 64 = 72, then the given sequence a(n) can be decimated into eight N/8 sequences. The first sequence has samples of a (0), a (8), a(16), a(24), a(32), a(40), a(48), a(56), the second sequence has samples of a(1), a(9), a(17), a(25), a(33), a(41), a(49), a(57) and the third sequence has samples of a(2), a(10), a(18), a(26), a(34), a(42), a(50), a(58) and the fourth sequence sample of a(3), a(11), a(19), a(27), a(35), a(43), a(51), a(59) and the fifth sequence sample of a(4), a(12), a(20), a(28), a(36), a(44), a(52), a(60) and the sixth sequence sample of a(5), a(13), a(21), a(29), a(37), a(45), a(53), a(61) and the seventh sequence sample of a(6), a(14), a(22), a(30), a(38) , a(46), a(54), a(62) and the eighth sequence sample of a(7), a(15), a(23), a(31), a(39), a(47), a(55), a(63). Starting from the basics, we can create a radix-8 DIT and DIF algorithm to evaluate the N = 64 DFT.

3.4 EXISTING RADIX-2 DIT AND DIF FFT ALGORITHMS

If N is a power of 2 and N=2m, then it is a radix-2 FFT. For example, if $N = 4 = 2^2$, the sequence a(n) above can be decimated into two N/2 sequences. The first sequence contains samples a(0), a(2) and the second sequence contains sample a(1). a(3). Starting from the basics, we can create a radix-2 DIT and DIF algorithm to evaluate the N=4 DFT.

THE 8-POINT DFT USING RADIX-2 DIT FFT

There are three stages to the computation of 8-point DFT using radix-2 FFT. Therefore, r = 2 and m = 3 for $N = 8 = 2^3$. The 8-point sequence is divided into four 2-point sequences. The 2-point DFT is computed for each 2-point sequence. Two 4-point DFTs are obtained from four 2-point DFTs, while the 8-point DFT is obtained from two 4-point DFTs.

Let {a (0), a (1), a (2), a (3), a (4), a (5), a (6), a (7)} be the 8-sample sequence a(n). The eight samples should be divided into two-sample sequences.

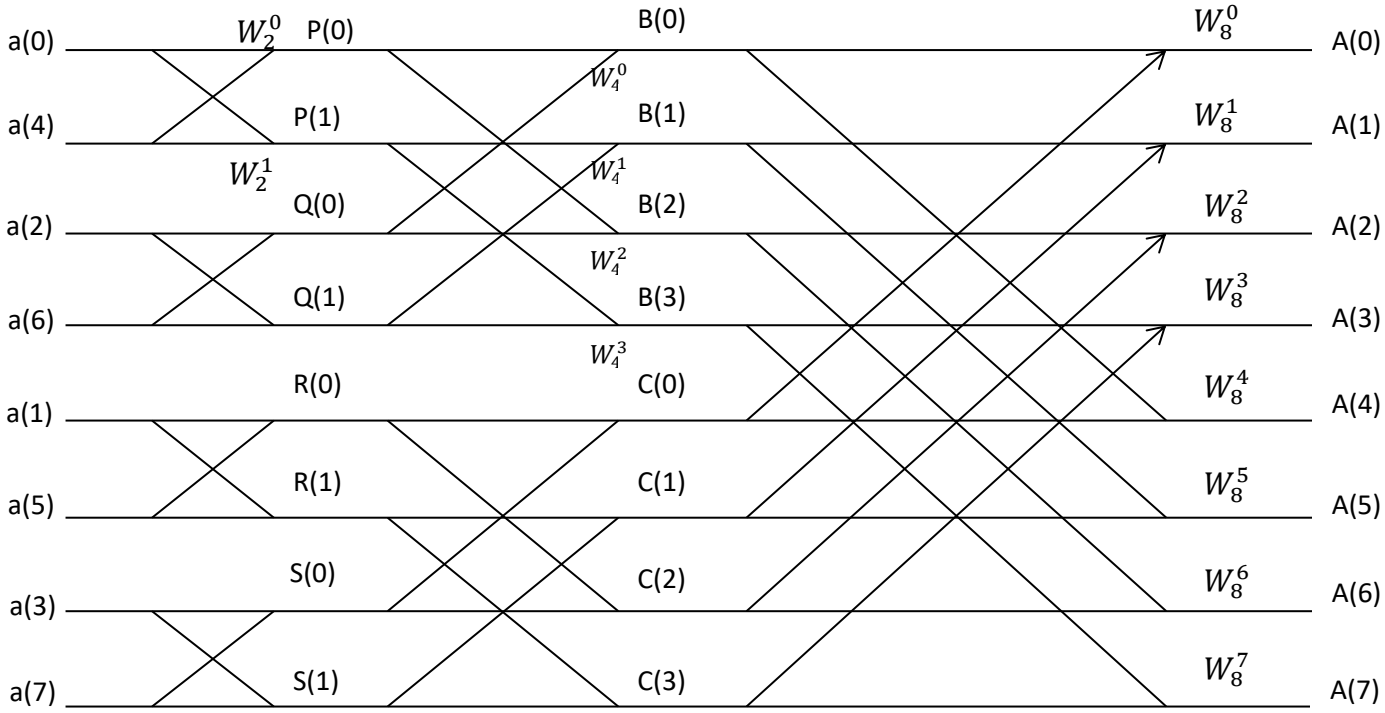


Figure 3.5 Illustration of complete flow graph obtained by combining all stages for N=8

Table 3.1: Normal and bit reversed order for N=8

Normal order		Bit reversed order	
a(0)	a(000)	a(0)	a(000)
a(1)	a(001)	a(4)	a(100)
a(2)	a(010)	a(2)	a(010)
a(3)	a(011)	a(6)	a(110)
a(4)	a(100)	a(1)	a(001)
a(5)	a(101)	a(5)	a(101)

a(6)	a(110)	a(3)	a(011)
a(7)	a(111)	a(7)	a(111)

As seen below, a(n) is decimated into 4 numbers of 2-point sequences in bit reversed order.

1. a (0) and a (4)
2. a (2) and a (6)
3. a (1) and a (5)
4. a (3) and a (7)

The 8-point sequence is a(n) = a(0), a(1), a(2), a(3), a(4), a(5), a(6), a(7).

4-point sequences b(n) = a(0), a(2), a(4), a(6); c(n) = a(1), a(3), a(5), a(7) derived from a(n).

2-point sequences produced from b(n): p(n) = a(0), a(4); q(n) = a(2), a(6)

2-point sequences produced from c(n), r(n) = a(1), a(5); s(n) = a(3), a(7)

Below are the relationships between the samples of distinct sequences.

p(0) = b(0) = a(0) r(0) = c(0) = a(1)

p(1) = b(2) = a(4) r(1) = c(2) = a(5)

q(0) = b(1) = a(2) s(0) = c(1) = a(3)

q(1) = b(3) = a(6) s(1) = c(3) = a(7)

The first stage of calculation

The 2-point DFTs of the 2-sample sequences are computed in the first stage of the computation**.**

Let P(k)=DFT{p(n)}. The 2-point DFT of p(n) is given by

$$P(k) = \sum_{n=0}^{\frac{N}{4}-1} a(n) w_{N/4}^{kn} \; ; 0 \le k \le N/4\text{-}1 \text{ and } P(k) = \sum_{n=0}^{1} a(n) w_2^{kn} \; ; 0 \le k \le 1$$

For; k = 0 ➔ P (0) = a (0) + W_2^0a (4) and k = 1 ➔ P (1) = a (0) + W_2^1a (4)

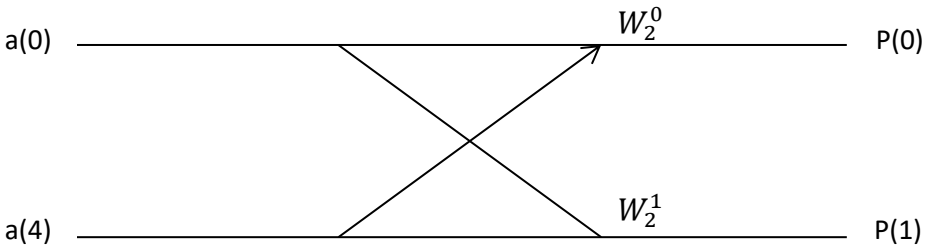


Figure 3.6 Butterfly structure of first stage for N=8

The second stage of computation

4-point DFTs are computed in the second step using the 2-point DFTs produced in the first stage as inputs.

Let B(k)=DFT{b(n)}. The equation can be used to calculate the 4-point DFT of b(n).

$$B(k) = P(k) + W_{N/2}^{k} Q(k); \text{ for } k = 0,1,2,3,4$$

$$B(0) = P(0) + W_4^0 Q(0) = P(0) + W_4^0 Q(0)$$

$$B(1) = P(1) + W_4^1 Q(1) = P(1) + W_4^1 Q(1)$$

$$B(2) = P(2) + W_4^2 Q(2) = P(0) - W_4^0 Q(0)$$

$$B(3) = P(3) + W_4^3 Q(3) = P(1) - W_4^1 Q(1)$$

Let C(k)=DFT{c(n)}. The equation can be used to calculate the 4-point DFT of c(n).

$$C(k) = R(k) + W_{N/2}^{k} S(k); \text{ for } k = 0,1,2,3$$

$$C(0) = R(0) + W_4^0 S(0) = R(0) + W_4^0 S(0)$$

$$C(1) = R(1) + W_4^1 S(1) = R(1) + W_4^1 S(1)$$

$$C(2) = R(2) + W_4^2 S(2) = R(0) - W_4^0 S(0)$$

$$C(3) = R(3) + W_4^2 S(3) = R(1) - W_4^1 S(1)$$

Here P(k) and Q(k) are periodic with periodicity of 2.

P (K+2) = P (K) and Q (K+2) = Q (2)

Here R(k) and S(k) are periodic with periodicity of 2.

R (K+2) = R (K) and S (K+2) = S (2)

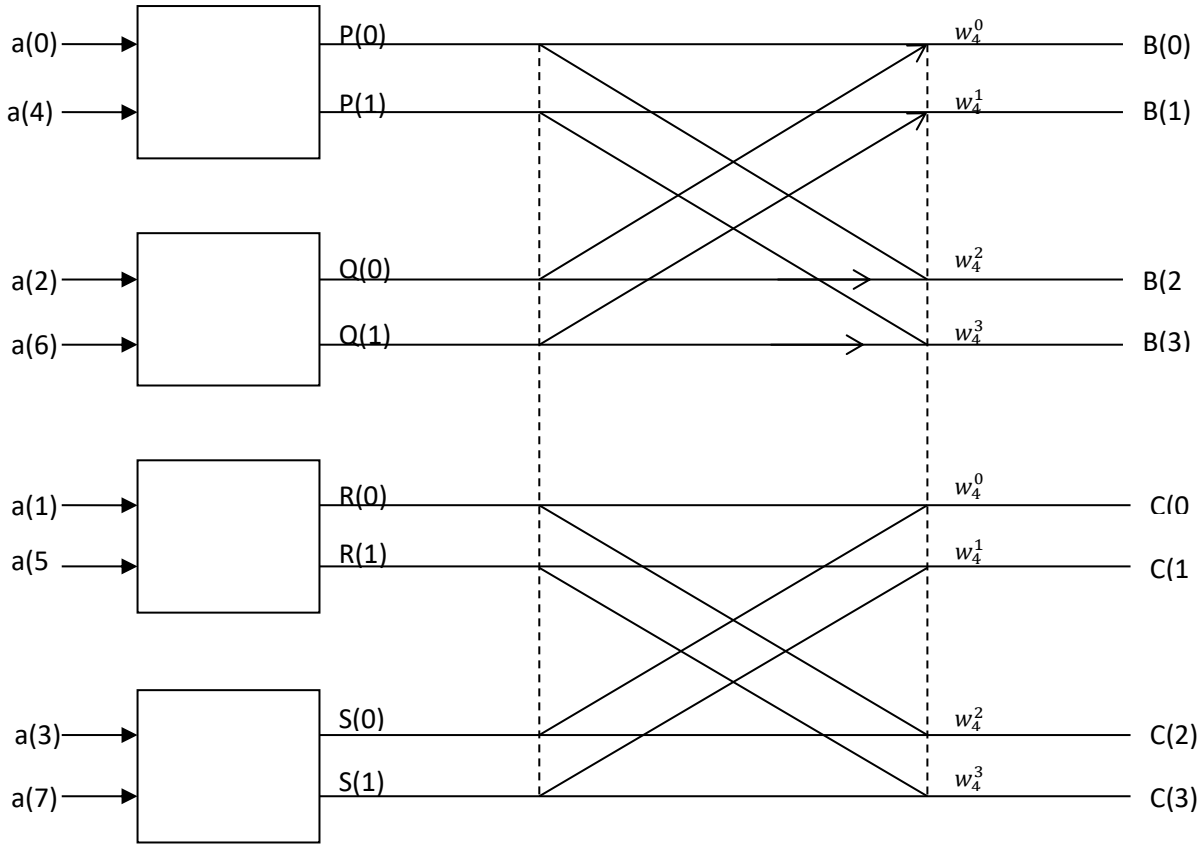


Figure 3.7 Signal Flow Graph of Second Stage for N=8

The third stage of computation

The 8-point DFT is computed in the third step using the 4-point DFTs generated in the second stage as inputs.

Let A(K) = DFT{a(n)}. The equation can be used to get the 8-point DFT of a(n).

A (k) = $\sum_{n=0}^{N-1} a(n) W_N^{kn}$; for k=0,1,2,3,4,5,6,7

A (0) = B (0) + W_8^0 C (0)

A (1) = B (1) + W_8^1 C (1)

A (2) = B (2) + W_8^2 C (2)

A (3) = B (3) + W_8^3 C (3)

A (4) = B (0) + W_8^4 C (0) = B (0) - W_8^0 C (0)

A (5) = B (1) + W_8^5 C (1) = B (1) - W_8^1 C (1)

A (6) = B (2) + W_8^6 C (2) = B (2) - W_8^2 C (2)

A (7) = B (3) + W_8^7 C (3) = B (3) - W_8^3 C (3)

Here B(k) and C(k) are periodic with periodicity of 4.

B(K+4) = B(K) and C(K+4) = C(K).

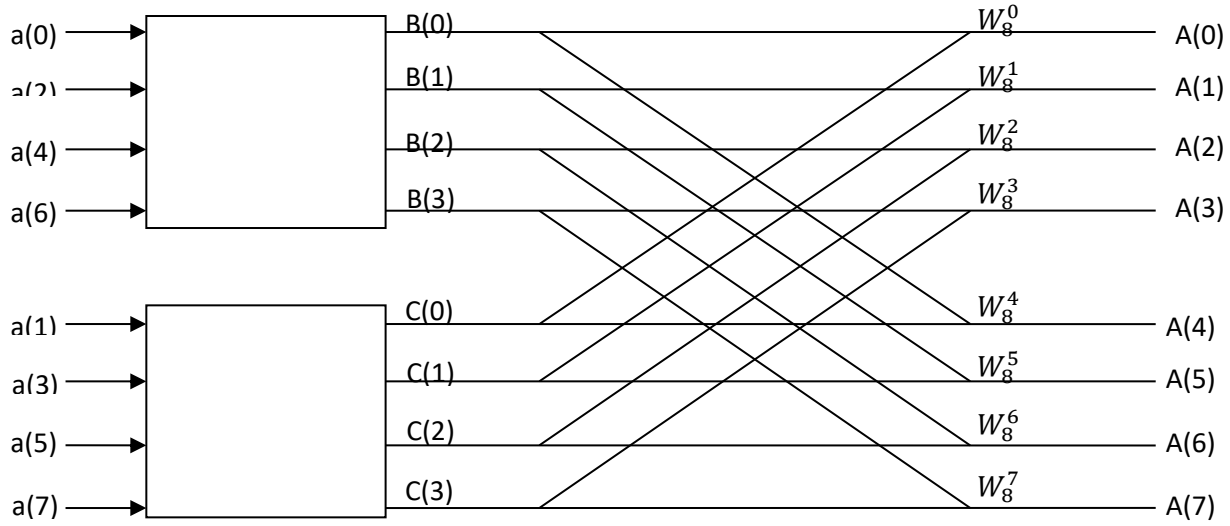


Figure 3.8 Signal flow graph of third stage for N=8

THE 4-POINT DFT USING RADIX-2 DIT FFT

There are two stages to the computation of 4-point DFT using radix-2 FFT. Therefore, r = 2 and m = 2 for N = 4 = 2^2. The 4-point sequence is divided into two 2-point sequences. The 2-point DFT is computed for each 2-point sequence. While the 4-point DFT is obtained from two 2-point DFTs.

Let {a (0), a (1), a (2), a (3)}be the 4-sample sequence a(n). The four samples should be divided into two-sample sequences

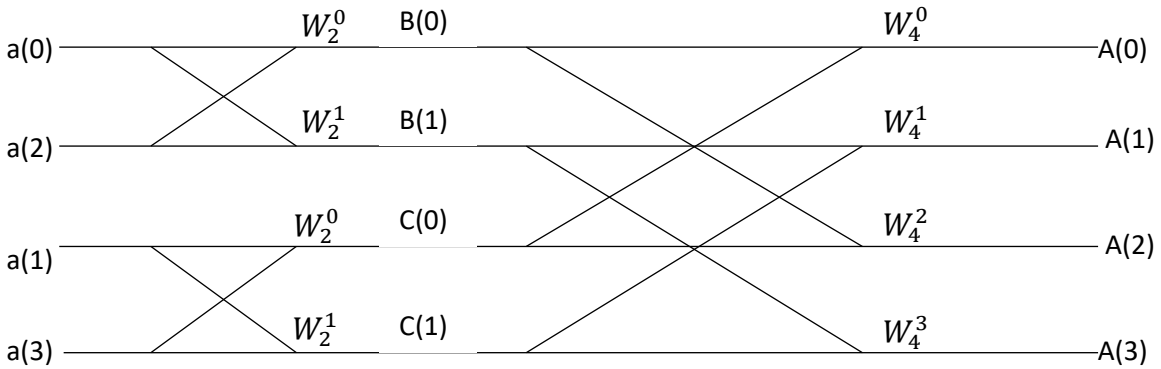


Figure 3.9 Illustration of complete flow graph obtained by combining all stages for N=4

Table 3.2: Normal and bit reversed order for N=4

Normal order		Bit reversed order	
a(0)	a(00)	a(0)	a(00)
a(1)	a(01)	a(2)	a(10)
a(2)	a(10)	a(1)	a(01)
a(3)	a(11)	a(3)	a(11)

The first stage of calculation

The 2-point DFTs of the 2-sample sequences are computed in the first stage of the computation**.**

Let P(k)=DFT{p(n)}. The 2-point DFT of p(n) is given by

$$B(k) = \sum_{n=0}^{\frac{N}{4}-1} a(n) w_{N/4}^{kn} \; ; \; 0 \le k \le N/4\text{-}1$$

$$B(k) = \sum_{n=0}^{1} a(n) w_2^{kn} \; ; \; 0 \le k \le 1$$

For; k = 0 ➔ B (0) = a (0) + W_2^0a (2)

k = 1 ➔ B (1) = a (0) + W_2^1a (2)

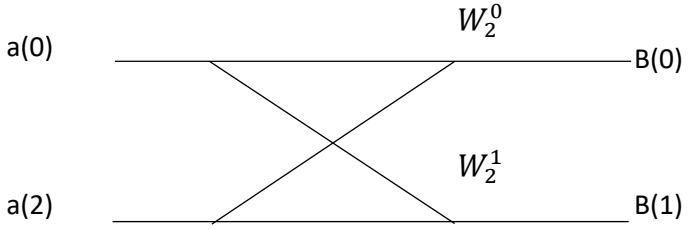


Figure 3.10 Butterfly structure of first stage for N=4

The second stage of computation

4-point DFTs are computed in the second step using the 2-point DFTs produced in the first stage as inputs.

Let A(k)=DFT{a(n)}. The equation can be used to calculate the 4-point DFT of a(n).

$A(k) = B(k) + W_N^k C(k)$; for k=0,1,2,3

$A(0) = B(0) + W_4^0 C(0)$

$A(1) = B(1) + W_4^1 C(1)$

$A(2) = B(0) + W_4^2 C(0) = B(0) - W_4^0 C(0)$

$A(3) = B(0) + W_4^3 C(1) = B(1) - W_4^1 C(1)$

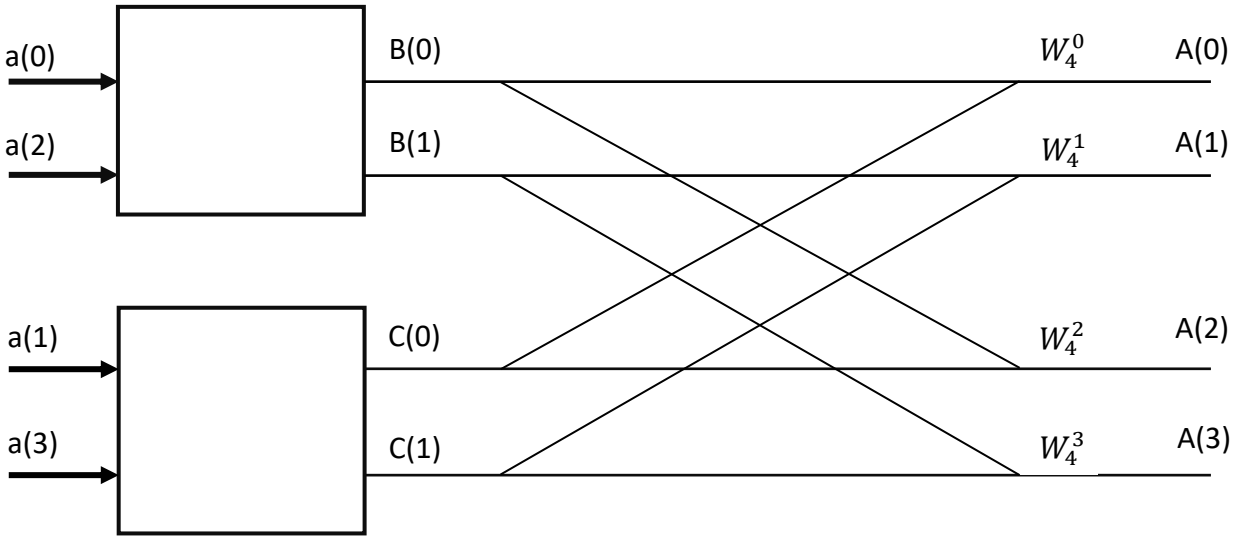


Figure 3.11 Signal flow graph of second stage for N=4

THE 2-POINT DFT USING RADIX-2 DIT FFT

There is one stage to the computation of 4-point DFT using radix-2 FFT. Therefore, r = 2 and m = 1 for $N = 2 = 2^1$. Let {a (0), a (1)}be the 2-sample sequence a(n).

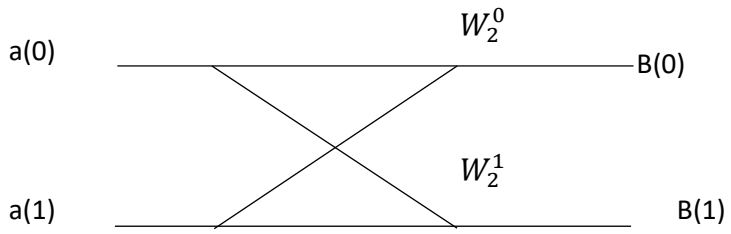


Figure 3.10 Butterfly structure for N=2

The 2-point DFTs of the 2-sample sequences are computed.

Let P(k)=DFT{p(n)}. The 2-point DFT of p(n) is given by

$$B(k) = \sum_{n=0}^{\frac{N}{4}-1} a(n) w_{N/4}^{kn} \; ; 0 \leq k \leq N/4\text{-}1$$

$$B(k) = \sum_{n=0}^{1} a(n) w_2^{kn} \; ; 0 \leq k \leq 1$$

For; k = 0 ➔ B (0) = a (0) + W_2^0a (2)

k = 1 ➔ B (1) = a (0) + W_2^1a (2)

CHAPTER 4

PROPOSED METHOD

4.1 DISTRIBUTED ARITHMETIC (DA)

It is a multiplier-less implementation technique that computes inner dot product between two signals – fixed and varying by using shifters and adders. DA takes the values in bit-serial fashion which suits for real-time processing.

$$y = \sum_{k=1}^{k} A_k x_k$$

Distributed Arithmetic (DA) is a computational algorithm that performs multiplication using precomputed Lookup Tables (LUTs) instead of logic. It is well suited to implementation on homogenous FPGAs because of its high utilization of the available LUTs.

A numerical example

address = {a3, a2, a1, a0} = {1,2,3,4}

inputs = {b3, b2, b1, b0} = {2,4,6,8}

Step-1: y=0(2) + 1(4) + 0(6) + 0(8) = 4

Step-2: y=0(2) + 0(4) + 1(6) + 1(8) = 14 + 4 = 18

Step-3: y=0(2) + 0(4) + 1(6) + 0(8) = 6 + 18 = 24

Step-4: y=0(2) + 0(4) + 0(6) + 1(8) = 8 + 24 = 32

4.2 FORMULATION OF DA

Consider

$$\boldsymbol{y = \sum_{k=1}^{k} A_k x_k} \qquad \ldots\ldots\ldots (\mathbf{1})$$

let x_k be an n-bit scaled two's complement number. in other words,

$$|x_k| < 1$$

$$x_k : \{b_{k0}, b_{k1}, b_{k2} \ldots\ldots\ldots, b_{k(n-1)}$$

where b_{k0} is the sign bit

We can express x_k as

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \qquad \ldots\ldots (2)$$

Substituting (2) in (1),

$$y = \sum_{k=1}^{k} A_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn}\, 2^{-n}\right]$$

$$y = \sum_{k=1}^{k} (b_{kn} * A_k) + \sum_{k=1}^{k} \sum_{n=1}^{N-1} (A_k * b_{kn}) 2^{-n} \qquad \ldots\ldots\ldots (3)$$

$$\boldsymbol{y = -\sum_{k=1}^{k} A_k * (b_{k0}) + \sum_{n=1}^{N-1} \sum_{k=1}^{k} [A_k * b_{kn}] 2^{-n}} \qquad \ldots. (4)$$

Consider the equation (4) rewritten as:

$$y = \sum_{n=1}^{N-1} \left[\sum_{k=1}^{k} A_k\, b_{kn}\right] 2^{-n} + \sum_{k=1}^{k} A_k\, (-b_{k0})$$

$\left[\sum_{k=1}^{k} A_k\, b_{kn}\right]$ has only 2^k possible values *(if k=4 we need $2*2^4$ =32 ROM size)*

$\left[\sum_{k=1}^{k} A_k\, b_{k0}\right]$ has only 2^k possible values

with the sign bit as an input, we can store it in a ROM of size=$2*2^k$

$Y = \sum_{k=1}^{K} A_k x_k$ has only 2^k possible values

4.3 LUT BASED FFT STRUCTURE

Lookup tables (LUTs) are a great way to speed up the evaluation of functions that cost a lot to calculate but don't cost much to cache. An interpolation algorithm can produce accurate approximations for data requests that lie between the samples of the table by averaging adjacent samples. Getting the result of a trigonometry calculation, such as the sine of a value, is a typical illustration of reducing run-time computations using lookup tables. Trigonometric function calculations can significantly slow down a computing programme. When an application first calculates the sine of a number of values, it can complete much more quickly.

Instead of using a mathematical formula to determine the sine of a value, the programme can use the lookup table to obtain the closest sine value from a memory address and/or interpolate to the desired sine. Therefore, mathematics coprocessors in computer

systems use lookup tables. Intel's notorious floating-point divide problem was caused by an error in a lookup table.

A straightforward array can be used to create functions with a single variable, such as sine and cosine. Multidimensional array indexing methods are necessary for functions with two or more variables. In the latter case, a function to compute xy for a finite set of x and y values may be replaced by a two-dimensional array of power[x][y]. Lookup tables, which are arrays of structures, can be used to create functions that have multiple results.

4.3.1 TWO POINT 64-BIT FFT BASED ON DA

In 2-point FFT the two inputs are taken and multiplied with the Twiddle factors.

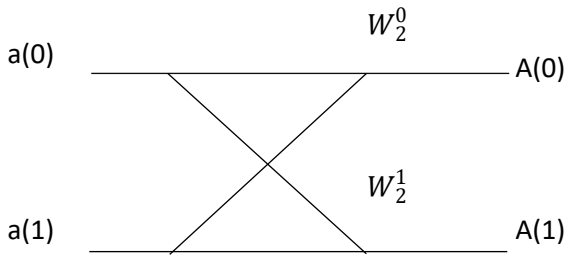


Figure 4.1 Butterfly diagram of 2-point FFT

The Formula for FFT is

$$\mathbf{A(k) = \sum_{n=0}^{N-1} a(n) W_N^{kn} \; ; \; 0 \le k \le N\text{-}1} \qquad \ldots(1)$$

Here for 2-point FFT N =2 and k ranges from 0 to N-1,

Here N is the number of points used to calculate the FFT.

When we substitute N=2 in the above equation then we get

A(k) = $\sum_{n=0}^{1} a(n) W_2^{kn}$; 0 ≤ k ≤ 1 (since N=2)

A(k) = $a(0) W_2^{k(0)} + a(1) W_2^{k(1)}$

As k ranges from k=0 to k=1 on substituting we get,

At k=0; A(0) = $a(0) + a(1) W_2^0$

At k=1; A(1) = $a(0) + a(1) W_2^1$

By the symmetric property twiddle factor changes to $W_2^1 = -W_2^0$,

Now the above equation changes to

A(1) = a(0) $- W_2^0$ a(1)

W_2^0= W1, W2 bits

In this proposed method we use a 2 bit select line (w1, w2). If the select line is 0 the bit representation is [0 0],

Similarly for 1 the bit representation is [0 1]

For 2 the bit representation is [1 0]

And for 3 the bit representation is [1 1]

If the select line is 0 then bit representation is [0 0] and by substituting the values in the above equation we get

A(0) = **0***a(0) + **0***a(1) =0

Similarly, when the select line is 1 then the equation is

A(0) = **0***a(0) + **1***a(1)= W_2^0a(1)

Similarly, when the select line is 2 then the equation is

A(0) = **1***a(0) + **0***a(1)= a(0)

Similarly, when the select line is 3 then the equation is

A(0) = **1***a(0) + **1***a(1)=a(0) + W_2^0a(1) (W_2^1 = - W_2^0)

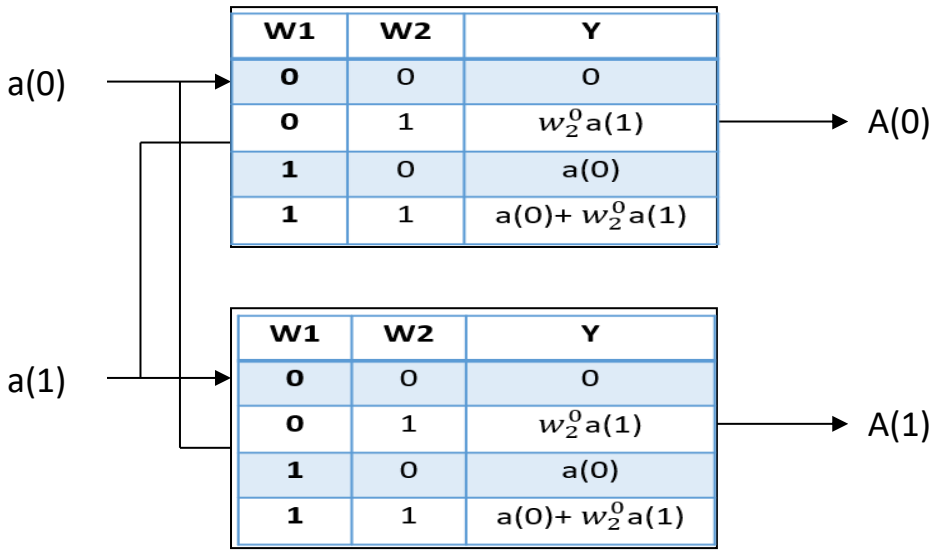


Figure 4.2 LUT based representation of 2-point FFT

4.3.2 FOUR POINT 64-BIT FFT BASED ON DA

A 4-points FFT can be computed in 2 stages where each stage consists of 1 operation. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 4-points DFT can be divided into two 2-point DFTs. Then these can be combined finally to get eight 4-point DFTs.

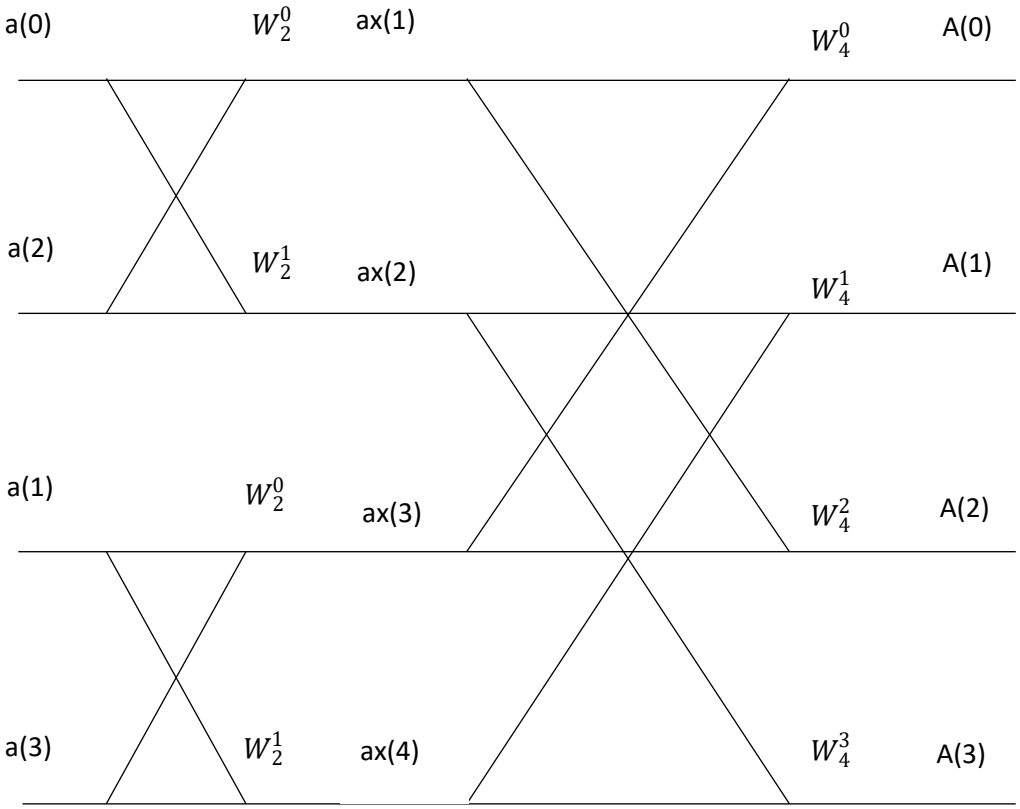


Figure 4.3 Butterfly diagram of 4-point FFT

As we know A(k) = $\sum_{n=0}^{N-1} a(n)W_N^{kn}$; 0 ≤ k ≤ N-1 …(1)

Here for 4-point FFT N =4 and k ranges from 0 to N-1,

When we substitute N=4 in the above equation then we get

A(k) = $\sum_{n=0}^{3} a(n)W_4^{kn}$; 0 ≤ k ≤ 3 (since N=4)

A(k) = $a(0)W_4^{k(0)} + a(1)W_4^{k(1)} + a(2)W_4^{k(2)} + a(3)W_4^{k(3)}$

As k ranges from k=0 to k=3 on substituting we get,

A(0) = $a(0) + a(1) + a(2) + a(3)$

A(1) = $a(0) + a(1)W_4^1 + a(2)W_4^2 + a(3)W_4^3$

A(2) = $a(0) + a(1)W_4^2 + a(2) + a(3)W_4^2$

A(3) = $a(0) + a(1)W_4^3 + a(2)W_4^2 + a(3)W_4^1$

Eq-1 decimation into even, odd sequence

Here we divide the equation-1 in an even and odd sequence then the obtained equations are

A(k) = B(k) + W_N^KC(k); $0 \leq k \leq N/2-1$ …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2); $N/2 \leq k \leq N-1$ …(3)

(Symmetry and periodicity)

As we know substituting the k=0 to k=3 in the equ-2, we get

k=0; A(0) = B(0) + W_4^0 C(0)

k=1; A(1) = B(1) + W_4^1 C(1)

k=2; A(2) = B(0) − W_4^0 C(0)

k=3; A(3) = B(1) − W_4^1 C(1)

W_4^1= W1, W2 bits

In this proposed method we use a 2 bit select line (w1, w2). If the select line is 0 the bit representation is [0 0],

Similarly for 1 the bit representation is [0 1]

For 2 the bit representation is [1 0]

And for 3 the bit representation is [1 1]

If the select line is 0 then bit representation is [0 0] and by substituting the values in the above equation we get

A(1) = **0***B(1) + **0***C(1) =0

Similarly, when the select line is 1 then the equation is

A(1) = **0***B(1) + **1***C(1)= W_4^1C(1)

Similarly, when the select line is 2 then the equation is

A(1) = **1***B(1) + **0***C(1)= B(1)

Similarly, when the select line is 3 then the equation is

A(1) = **1***B(1) + **1***C(1)= B(1) + W_4^1C(1)

(W_4^1 = -j = 1)

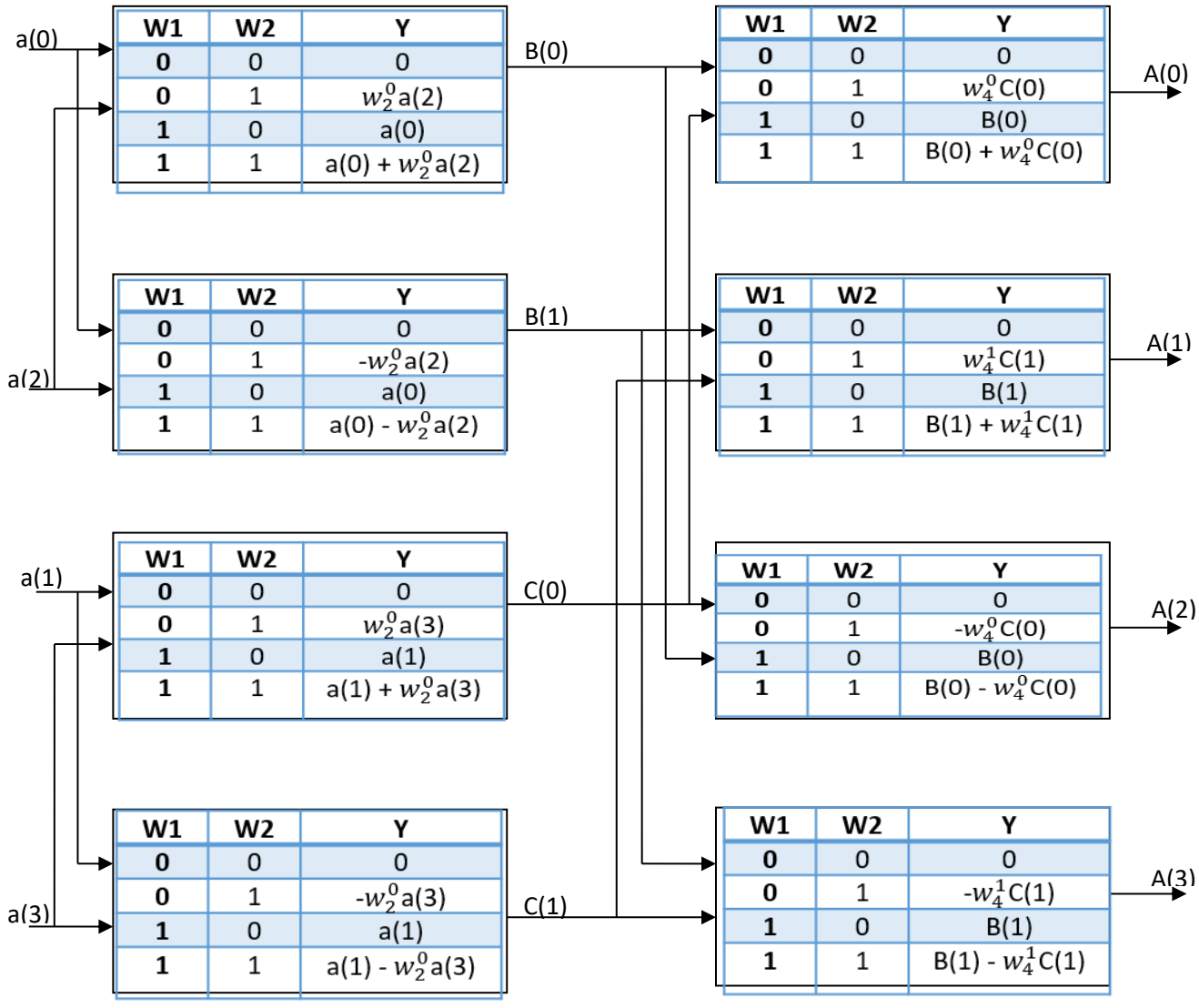


Figure 4.4 LUT based representation of 4-point FFT

4.3.3 EIGHT POINT 64-BIT FFT BASED ON DA

A 8-points FFT can be computed in 3 stages where each stage consists of 2 operation. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 8-points DFT can be divided into four 2-point DFTs. Then these can be combined to get two 4-point DFTs. Finally, we get 8- point DFT.

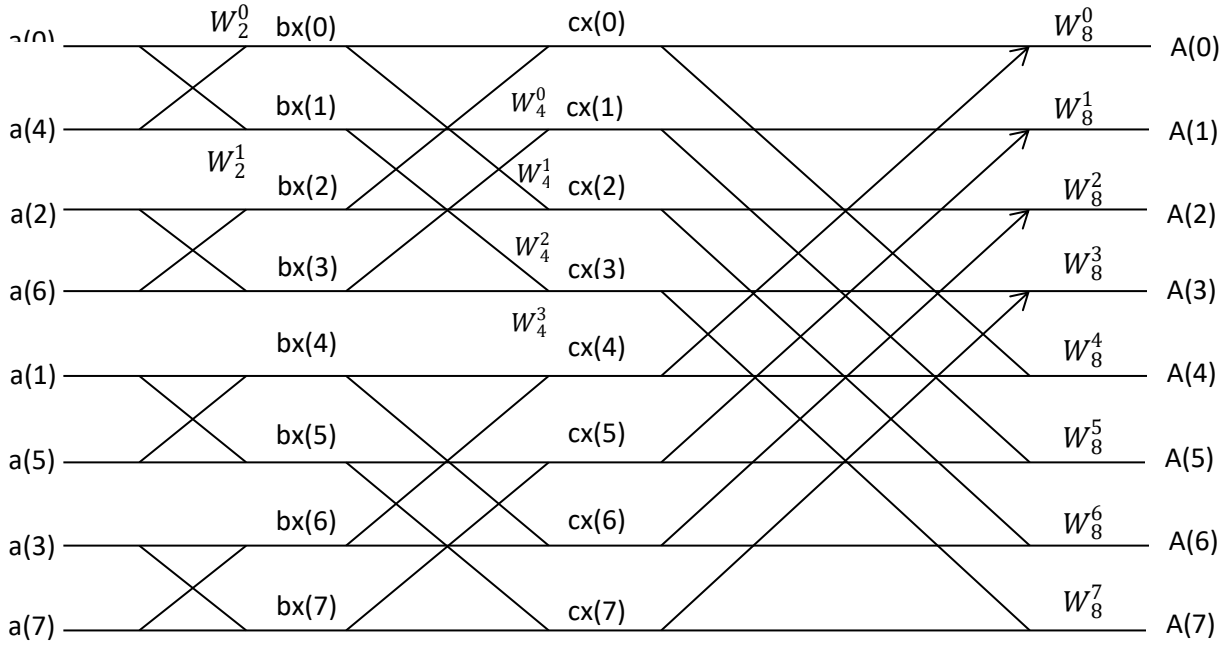


Figure 4.5 Butterfly diagram of 8-point FFT

As we know A(k) = $\sum_{n=0}^{N-1} a(n)W_N^{kn}$; $0 \le k \le N-1$ …(1)

Here for 8-point FFT N =8 and k ranges from 0 to N-1,

When we substitute N=8 in the above equation then we get

A(k) = $\sum_{n=0}^{7} a(n)W_8^{kn}$; $0 \le k \le 7$ (since N=8)

A(k)=$a(0)W_8^{k(0)}+a(1)W_8^{k(1)}+a(2)W_8^{k(2)}+a(3)W_8^{k(3)}+a(4)W_8^{k(4)}+a(5)W_8^{k(5)}+a(6)W_8^{k(6)}$
+ $a(7)W_8^{k(7)}$

Eq-1 decimation into even, odd sequence

Here we divide the equation-1 in an even and odd sequence then the obtained equations are

A(k) = B(k) + W_N^KC(k); $0 \le k \le N/2-1$ …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2); N/2 ≤ k ≤ N-1 …(3)

(Symmetry and periodicity)

As we know substituting the k=0 to k=7 in the above even and odd equations, we get

A(0) =$a(0) + a(1) + a(2) + a(3) + a(4) + a(5) + a(6) + a(7)$

A(1) = $a(0) + a(1)W_8^1 + a(2)W_8^2 + a(3)W_8^3 + a(4)W_8^4 + a(5)W_8^5 + a(6)W_8^6 + a(7)W_8^7$

A(1) =$a(0) + a(1)W_8^1 + a(2)W_8^2 + a(3)W_8^3 - a(4) - a(5)W_8^1 - a(6)W_8^2 - a(7)W_8^3$

A(2) =$a(0) + a(1)W_8^2 + a(2)W_8^4 + a(3)W_8^6 + a(4)W_8^8 + a(5)W_8^{10} + a(6)W_8^{12} + a(7)W_8^{14}$

A(2) =$a(0) + a(1)W_8^2 - a(2) - a(3)W_8^2 + a(4) + a(5)W_8^2 - a(6) - a(7)W_8^2$

A(3) =$a(0) + a(1)W_8^3 + a(2)W_8^6 + a(3)W_8^9 + a(4)W_8^{12} + a(5)W_8^{15} + a(6)W_8^{18} + a(7)W_8^{21}$

A(3) =$a(0) + a(1)W_8^3 + a(2)W_8^2 + a(3)W_8^1 - a(4) - a(5)W_8^3 - a(6)W_8^2 - a(7)W_8^1$

A(4) =$a(0) + a(1)W_8^4 + a(2)W_8^8 + a(3)W_8^{12} + a(4)W_8^{16} + a(5)W_8^{20} + a(6)W_8^{24} + a(7)W_8^{28}$

A(4)=$a(0) - a(1) + a(2) - a(3) + a(4) - a(5) + a(6) - a(7)$

A(5)=$a(0) + a(1)W_8^5 + a(2)W_8^{10} + a(3)W_8^{15} + a(4)W_8^{20} + a(5)W_8^{25} + a(6)W_8^{30} + a(7)W_8^{35}$

A(5)=$a(0) - a(1)W_8^1 + a(2)W_8^2 - a(3)W_8^3 - a(4) + a(5)W_8^1 - a(6)W_8^2 + a(7)W_8^3$

A(6)=$a(0) + a(1)W_8^6 + a(2)W_8^{12} + a(3)W_8^{18} + a(4)W_8^{24} + a(5)W_8^{30} + a(6)W_8^{36} + a(7)W_8^{42}$

A(6) =$a(0) - a(1)W_8^2 - a(2) + a(3)W_8^2 + a(4) - a(5)W_8^2 - a(6) + a(7)W_8^2$

A(7)=$a(0) + a(1)W_8^7 + a(2)W_8^{14} + a(3)W_8^{21} + a(4)W_8^{28} + a(5)W_8^{35} + a(6)W_8^{42} + a(7)W_8^{49}$

A(7) = $a(0) - a(1)W_8^3 - a(2)\,W_8^2 - a(3)W_8^1 - a(4) + a(5)W_8^3 + a(6)W_8^2 + a(7)W_8^3$

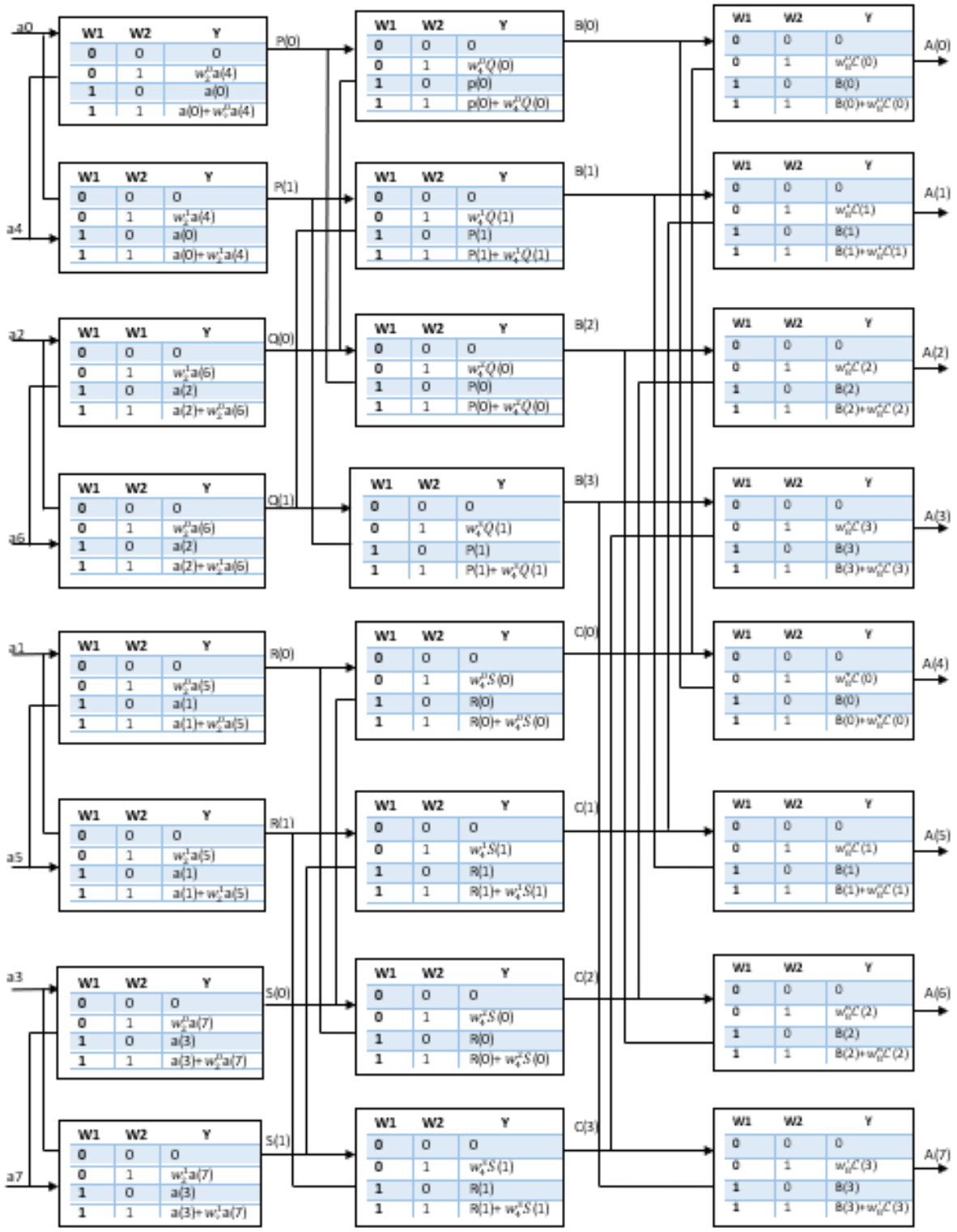


Figure 4.6 LUT based representation of 8-point FFT

4.3.4 SIXTEEN POINT 64-BIT FFT BASED ON DA

A 16-points FFT can be computed in 4 stages where each stage consists of 8 operation. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 16-points DFT can be divided into eight 2-point DFTs. Then these can be combined to get four 4-point DFTs. Then these can be combined to get two 8-points DFTs. Finally, we get 16- point DFT.

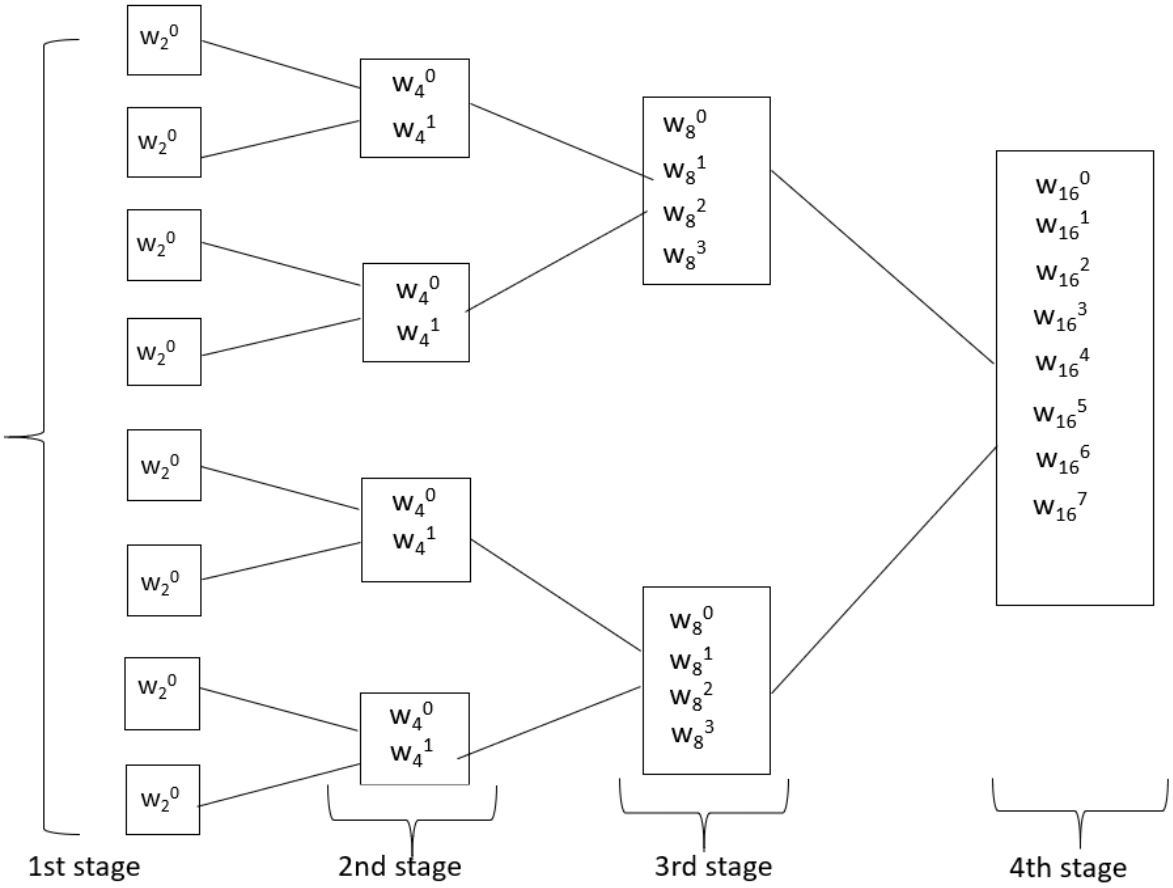


Figure 4.7 Stage-wise representation of 16-point FFT

As we know A(k) = $\sum_{n=0}^{N-1} a(n)W_N^{kn}$; 0 ≤ k ≤ N-1 …(1)

Here for 16-point FFT N =16 and k ranges from 0 to N-1,

When we substitute N=16 in the above equation then we get

A(k) = $\sum_{n=0}^{15} a(n)W_{16}^{kn}$; 0 ≤ k ≤ 15 (since N=16)

As we know substituting the k=0 to k=15 in the above even and odd equations, we get

A(k) = $a(0)W_{16}^{k(0)} + a(1)W_{16}^{k(1)} + a(2)W_{16}^{k(2)} + a(3)W_{16}^{k(3)} + - - - a(14)W_{16}^{k(14)} + a(15)W_{16}^{k(15)}$

A(0) = $a(0) + a(1) + a(2) + a(3)$ + - - - $a(14) + a(15)$

A(1) = $a(0)$ + $a(1)W_{16}^{1}$ + $a(2)W_{16}^{2}$ + $a(3)W_{16}^{3}$ + - - - $a(14)W_{16}^{14}$ + $a(15)W_{16}^{15}$

.

.

A(15) = $a(0)$ + $a(1)W_{16}^{15}$ + $a(2)W_{16}^{30}$ + $a(3)W_{16}^{45}$ + - - - $a(14)W_{16}^{210}$ + $a(15)W_{16}^{225}$

Eq-1 decimation into even, odd sequence

A(k) = B(k) + W_N^KC(k) ; 0 ≤ k ≤ N/2-1 …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2) ; N/2 ≤ k ≤ N-1 …(3)

(Symmetry and periodicity)

4.3.5 THIRTY-TWO POINT 64-BIT FFT BASED ON DA

A 32-points FFT can be computed in 5 stages where each stage consists of 16 operations. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner.

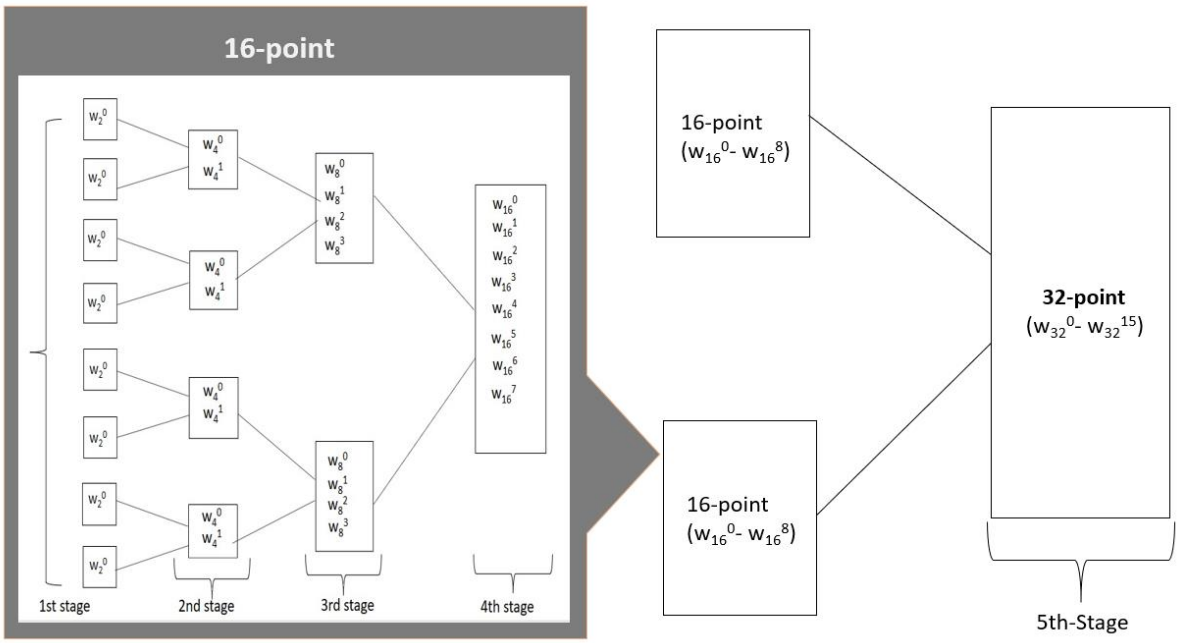


Figure 4.8 Stage-wise representation of 32-point FFT

In this case 32-points DFT can be divided into sixteen 2-point DFTs. Then these can be combined to get eight 4-point DFTs. Then these can be combined to get four 8-points DFTs. These can be combined to get two 16-points DFTs. Finally, we get 32- point DFT.

A(k) = $\sum_{n=0}^{31} a(n)W_{32}^{kn}$; 0 ≤ k ≤ 31 (since N=32)

As we know substituting the k=0 to k=31 in the above even and odd equations, we get

A(k) = $a(0)W_{32}^{k(0)} + a(1)W_{32}^{k(1)} + a(2)W_{32}^{k(2)} + a(3)W_{32}^{k(3)} + \text{- - -} + a(31)W_{32}^{k(31)}$

A(0) = $a(0) + a(1) + a(2) + a(3) + \text{- - -}\ a(30) + a(31)$

A(1) = $a(0) + a(1)W_{32}^{1} + a(2)W_{32}^{2} + a(3)W_{32}^{3} + \text{- - -}\ a(30)W_{32}^{30} + a(31)W_{32}^{31}$

.

.

A(31) = $a(0) + a(1)W_{32}^{31} + a(2)W_{32}^{62} + a(3)W_{16}^{93} + \text{- - -}\ a(30)W_{16}^{930} + a(15)W_{16}^{961}$

Eq-1 decimation into even, odd sequence

A(k) = B(k) + W_N^KC(k) ; 0 ≤ k ≤ N/2-1 …(1)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2) ; N/2 ≤ k ≤ N-1 …(2)

(Symmetry and periodicity)

4.3.6 SIXTY-FOUR POINT 64-BIT FFT BASED ON DA

A 64-points FFT can be computed in 6 stages where each stage consists of 32 operations. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 64-points DFT can be divided into thirty-two 2-point DFTs. Then these can be combined to get sixteen 4-point DFTs. Then these can be combined to get eight 8-points DFTs. These can be combined to get four 16-points DFTs. Then these four 16-points are combined to two 32-points DFTs. Finally, we get 64- point DFT.

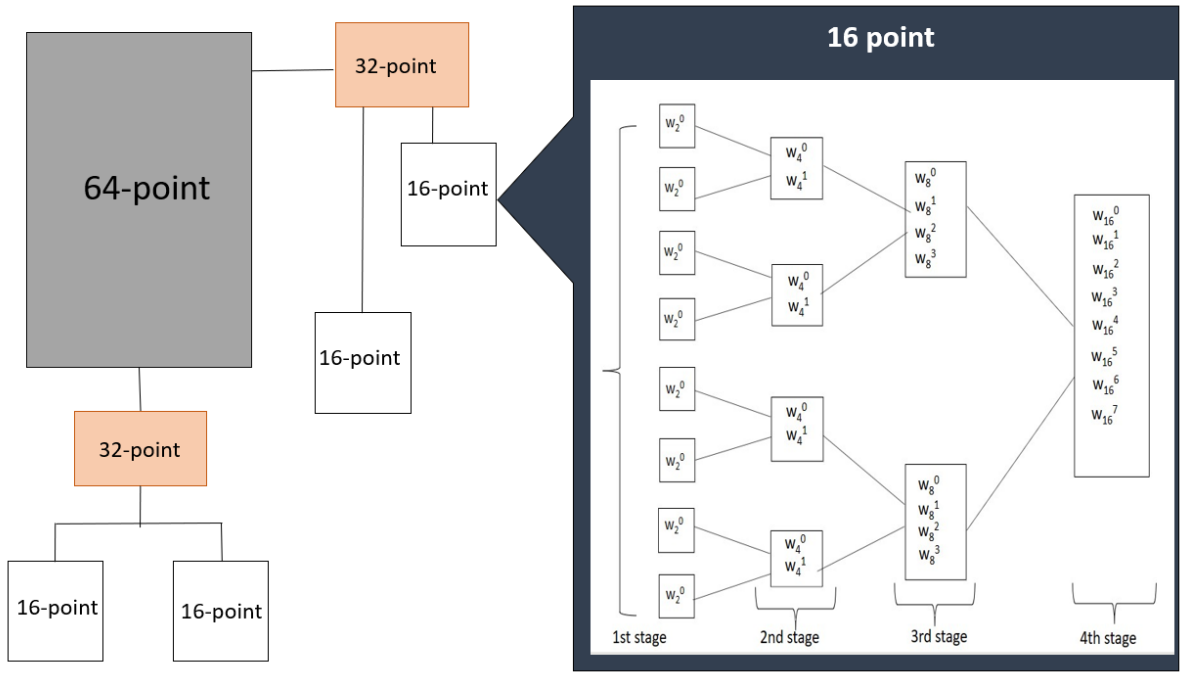


Figure 4.9 Stage-wise representation of 64-point FFT

As we know A(k) = $\sum_{n=0}^{N-1} a(n)W_N^{kn}$; 0 ≤ k ≤ N-1 …(1)

Here for 64-point FFT N =64 and k ranges from 0 to N-1,

When we substitute N=64 in the above equation then we get

A(k) = $\sum_{n=0}^{63} a(n)W_{64}^{kn}$; 0 ≤ k ≤ 63 (since N=64)

As we know substituting the k=0 to k=63 in the above even and odd equations, we get

A(k) = $a(0)W_{64}^{k(0)} + a(1)W_{64}^{k(1)} + a(2)W_{64}^{k(2)} + a(3)W_{64}^{k(3)} + \text{- - -} + a(63)W_{64}^{k(63)}$

A(0) = $a(0) + a(1) + a(2) + a(3) + \text{- - -}\ a(62) + a(63)$

A(1) = $a(0) + a(1)W_{64}^{1} + a(2)W_{64}^{2} + a(3)W_{64}^{3} + \text{- - -}\ a(62)W_{64}^{62} + a(63)W_{64}^{63}$

.

.

A(63) = $a(0) + a(1)W_{64}^{63} + a(2)W_{64}^{126} + a(3)W_{64}^{189} + \text{- - -}\ a(62)W_{64}^{3906} + a(63)W_{64}^{3969}$

Eq-1 decimation into even, odd sequence

A(k) = B(k) + W_N^KC(k) ; 0 ≤ k ≤ N/2-1 …(2) B(k) and C(k) → Periodic with N/2

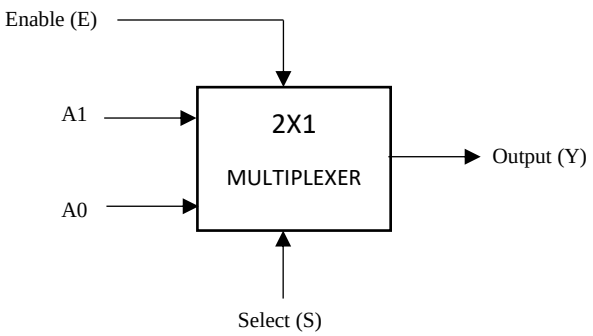
4.4 LUTLESS BASED FFT STRUCTURE

A LUT less based FFT structure using distributed arithmetic (DA) is a type of DA that does not use lookup tables (LUTs) to perform the computation. Instead, it uses other techniques, such as arithmetic and logic operations, to perform the computation. One of the technique to perform a LUT less operation is using MULTIPLEXER (MUX).A MUX based Distributed Arithmetic(DA) is a technique used in digital signal processing (DSP) to efficiently implement multiplication of digital signals by a constant coefficient. It is a combination of two techniques: Distributed Arithmetic and Mux-based implementation.

Mux-based implementation, uses a multiplexer to select and combine multiple reference voltages to generate an output voltage that corresponds to the input signal. It is commonly used to implement Digital-to-Analog converters. These models or algorithms can be designed to provide a more flexible and adaptable system, as they can be updated or modified without requiring changes to the LUT values.

A Mux-based Distributed Arithmetic combines these techniques to perform multiplication in a digital circuit using a precomputed table of products stored in the form of reference voltages. The input signal is used to select one or more of these reference voltages using a multiplexer, and the selected reference voltages are then combined to generate the output signal. The advantage of this approach is that it reduces the memory requirement of the precomputed table by using a multiplexer to select the required values, making it more efficient in terms of memory usage.

LUT less based DA have several advantages over LUT-based DA systems. They can be more power-efficient, as they do not require the large memory resources needed to store LUT values. They can also be more flexible and adaptable, as the coefficient vector can be generated using mathematical models or algorithms, which can be updated or modified as needed.



In this LUT less technique, a 2x1 multiplexer(MUX) is used to implement each point FFT.

4.5 STRUCTURE OF LUTLESS

A Mux-based Fast Fourier Transform (FFT) is a variation of the FFT algorithm that uses a multiplexer (mux) to perform the butterfly operations that are at the core of the FFT algorithm. The butterfly operation is the basic building block of the FFT algorithm, which involves combining pairs of complex numbers to compute the DFT of a signal.

The Mux-based FFT uses a Mux to select the correct pairs of complex numbers that are to be combined in the butterfly operation, thereby eliminating the need for a complex twiddle factor multiplication, which is typically required in conventional FFT algorithms. This reduces the computational complexity of the FFT algorithm and makes it more efficient in terms of power consumption and hardware requirements .An example for a MUX based 4-point FFT structure is shown below,

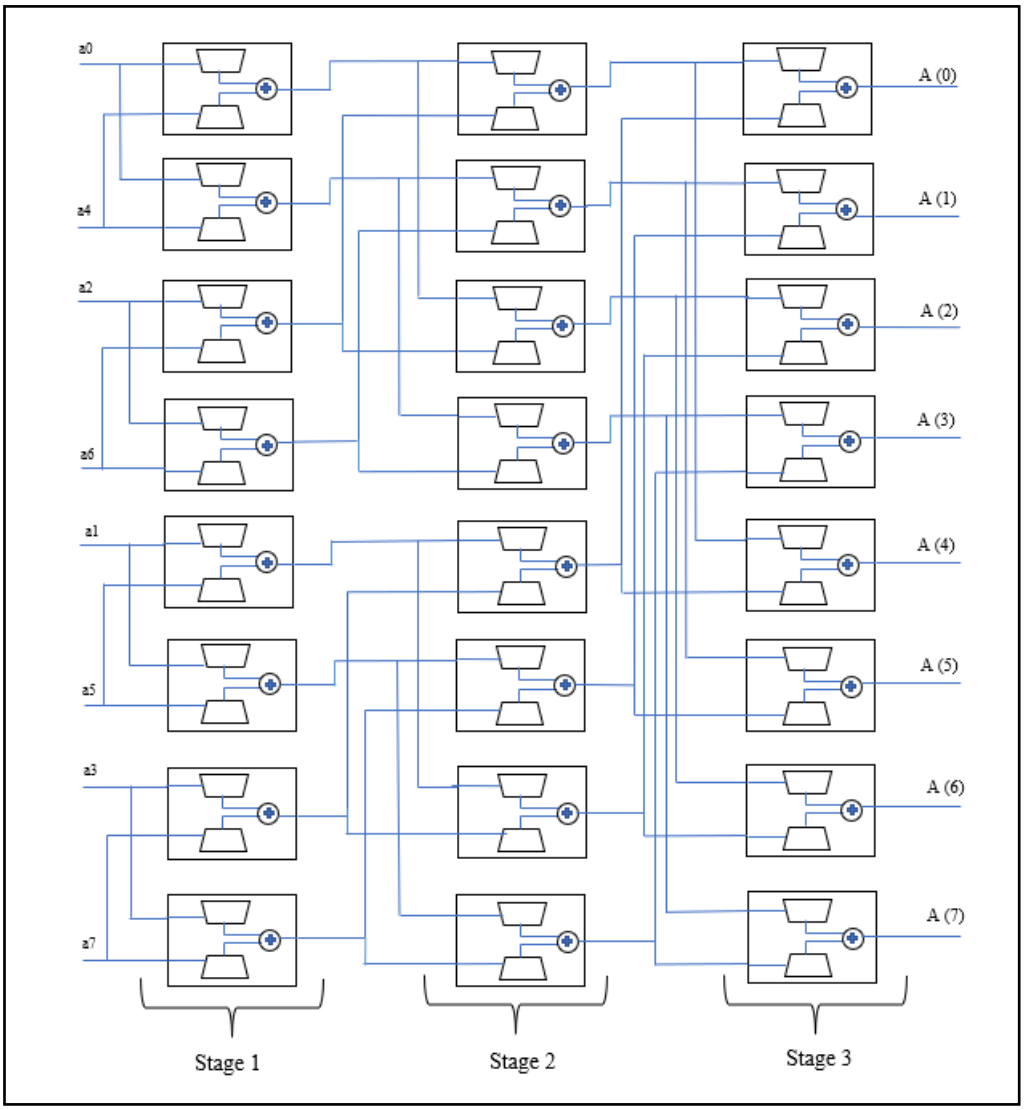


Figure 4.10 Example of MUX based FFT structure

Each stage uses a 2x1 multiplexers to implement a 8-point FFT structure. In the above figure, there are four 2-point, two 4-point and one 8-point FFT structures are used. The basic structure of a Mux-based FFT consists of multiple stages of butterfly operations, where each stage is implemented using a Mux to select the appropriate pairs of complex numbers. The input data is first divided into smaller sub-sequences that are processed independently in each stage of the FFT algorithm. The output of each stage is then combined to form the final output of the FFT.

4.5.1 TWO POINT 64 BIT LUTLESS FFT BASED ON DA

A two-point 64-bit LUT less FFT based on distributed arithmetic (DA) is a type of FFT algorithm that can perform a 2-point FFT computation using DA techniques, without relying on lookup tables (LUTs). The 64-bit refers to the precision of the input and output data.

The Formula for FFT is

A(k) = $\sum_{n=0}^{N-1} a(n) W_N^{kn}$; 0 ≤ k ≤ N-1. …(1)

Here for 2-point FFT N =2 and k ranges from 0 to N-1(N is the number of points used)

When we substitute N=2 in the above equation then we get

A(k) = $\sum_{n=0}^{1} a(n) W_2^{kn}$; 0 ≤ k ≤ 1 (since N=2)

A(k) = $a(0) W_2^{k(0)} + a(1) W_2^{k(1)}$

As k ranges from k=0 to k=1 on substituting we get,

At k=0; A(0) = $a(0) + a(1) W_2^0$

At k=1; A(1) = $a(0) + a(1) W_2^1$

By the symmetric property twiddle factor changes to $W_2^1 = -W_2^0$,

Now the above equation changes to

A(1) = a(0) $- W_2^0$ a(1) ….. (2)

In Multiplexer based technique, the select lines are the inputs i.e., a(0), a(1) and the input for multiplexer is precomputed with the Complex twiddle factors. For a 2-point Mux based FFT structure the twiddle factor is W_0^2. Equation (2) is implemented using select lines separately as shown below. First mux using a(0) as select line and second mux using select line a(1).

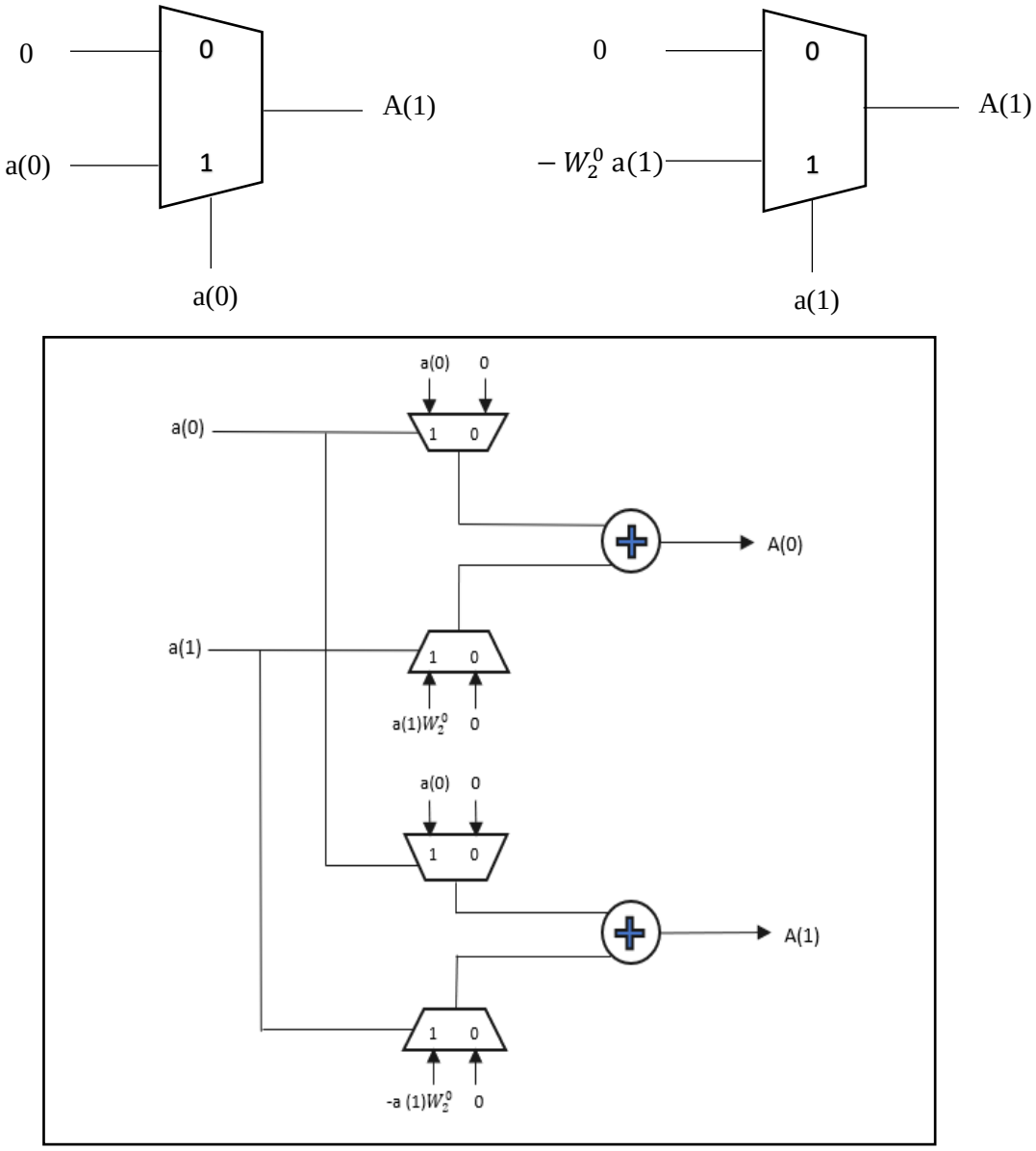


Figure 4.11 2-point MUX based FFT structure

In this proposed method we use a 2 bit select line (w1, w2). If the select lines are represented in bit representation. The detailed explanation for each select line is given below

If the select line is 0 then bit representation is [0 0] and by substituting the values in the above equation we get

A(0) = **0***a(0) + **0***a(1) =0

Similarly, when the select line is 1 [0 1] then the equation is

A(0) = **0***a(0) + **1***a(1)= W_2^0a(1)

Similarly, when the select line is 2 then the equation is

A(0) = **1***a(0) + **0***a(1)= a(0)

Similarly, when the select line is 3 then the equation is

A(0) = **1***a(0) + **1***a(1)=a(0) + W_2^0a(1)

4.5.2 FOUR POINT 64 BIT LUTLESS FFT BASED ON DA

A 4 point 64 bit MUX based FFT structure with two stages is shown below,

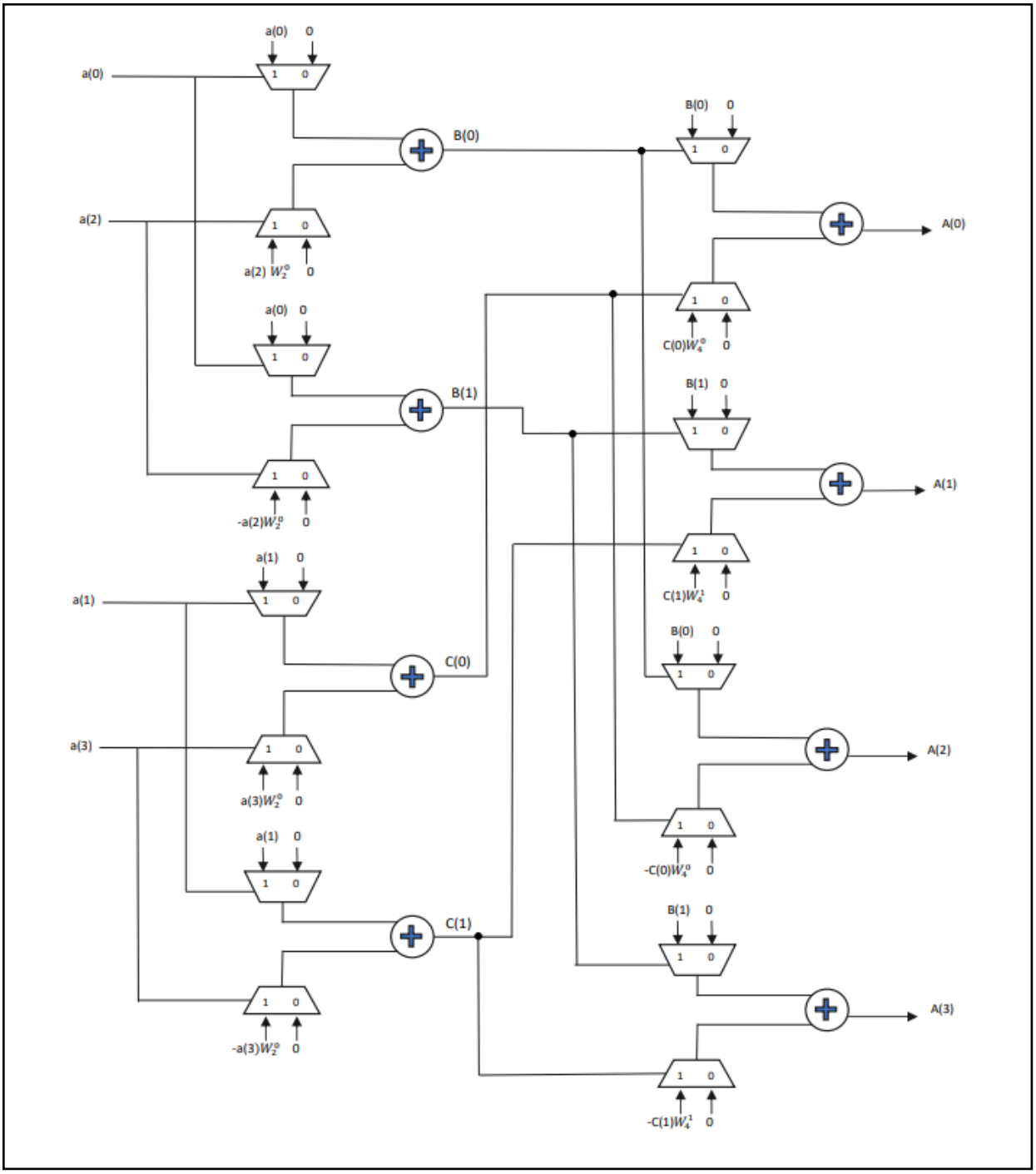


Figure 4.12 4-point MUX based FFT structure

As we know $A(k) = \sum_{n=0}^{N-1} a(n) W_N^{kn}$; $0 \le k \le N-1$ … (1)

Here for 4-point FFT N =4 and k ranges from 0 to N-1,

When we substitute N=4 in the above equation then we get

$A(k) = \sum_{n=0}^{3} a(n) W_4^{kn}$; $0 \le k \le 3$ (since N=4)

$A(k) = a(0)W_4^{k(0)} + a(1)W_4^{k(1)} + a(2)W_4^{k(2)} + a(3)W_4^{k(3)}$

As k ranges from k=0 to k=3 on substituting we get,

$A(0) = a(0) + a(1) + a(2) + a(3)$

$A(1) = a(0) + a(1)W_4^1 + a(2)W_4^2 + a(3)W_4^3$

$A(2) = a(0) + a(1)W_4^2 + a(2) + a(3)W_4^2$

$A(3) = a(0) + a(1)W_4^3 + a(2)W_4^2 + a(3)W_4^1$

Eq-1 decimation into even, odd sequence

Here we divide the equation-1 in an even and odd sequence then the obtained equations are

$A(k) = B(k) + W_N^K C(k)$; $0 \le k \le N/2-1$ … (2)

B(k) and C(k) → Periodic with N/2

$A(k) = B(k-N/2) + W_N^K C(k-N/2)$; $N/2 \le k \le N-1$ … (3)

(Symmetry and periodicity)

As we know substituting the k=0 to k=3 in the equ-2, we get

k=0; $A(0) = B(0) + W_4^0 C(0)$

k=1; $A(1) = B(1) + W_4^1 C(1)$

k=2; $A(2) = B(0) - W_4^0 C(0)$

k=3; $A(3) = B(1) - W_4^1 C(1)$

W_4^1= W1, W2 bits

In this proposed method we use a 2 bit select line (w1, w2). Select lines are represented in bits.

For select line =1, the bit representation is [0 1]

For 2 the bit representation is [1 0]

And for 3 the bit representation is [1 1]

If the select line is 0 then bit representation is [0 0] and by substituting the values in the above equation we get

A(1) = **0***B(1) + **0***C(1) =0

Similarly, when the select line is 1 then the equation is

A(1) = **0***B(1) + **1***C(1)= W_4^1C(1)

Similarly, when the select line is 2 then the equation is

A(1) = **1***B(1) + **0***C(1)= B(1)

Similarly, when the select line is 3 then the equation is

A(1) = **1***B(1) + **1***C(1)= B(1) + W_4^1C(1) ……($W_4^1 = -j$)

Twiddle factors in the proposed method are stored in MUX so the memory storage is decreased when compared to the conventional/existing FFT structure. The Butterfly structure for a 4-point conventional FFT is shown below which includes complex twiddle factors during computations

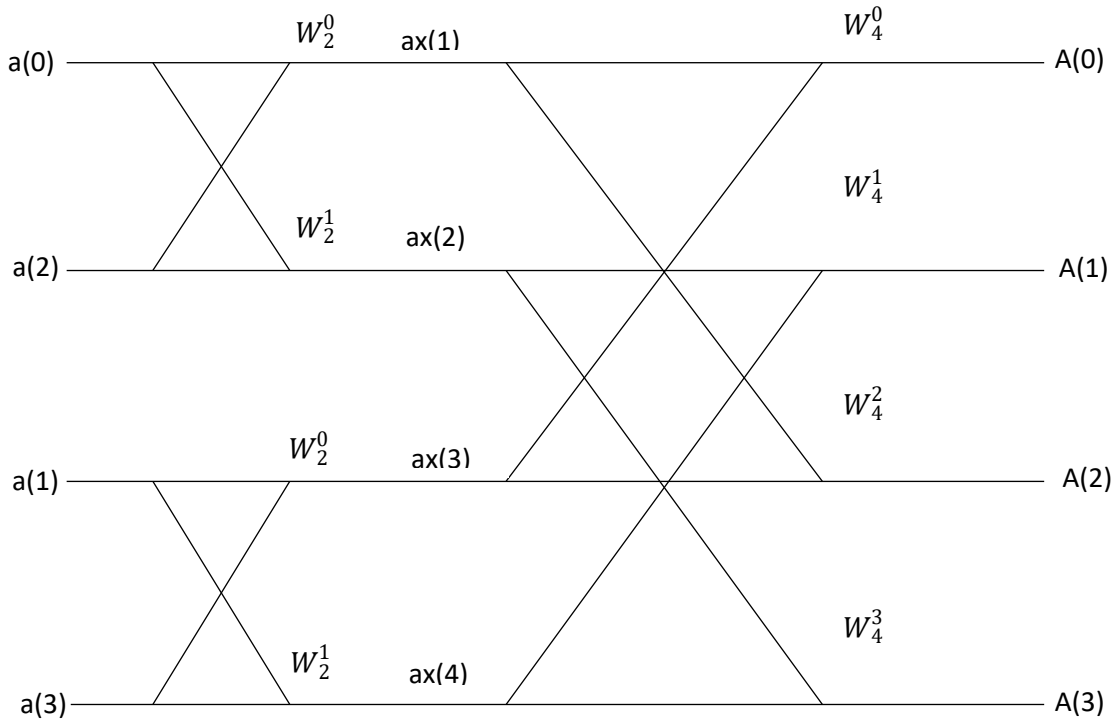


Fig 4.13 Butterfly diagram of 4-point FFT

4.5.3 EIGHT POINT 64 BIT LUTLESS FFT BASED ON DA

Eight point FFT can be computed in 3 stages where each stage consists of 2 operation. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 8-point DFT can be divided into four 2-point DFTs. Then these can be combined to get two 4-point DFTs. Finally, we get 8- point DFT.

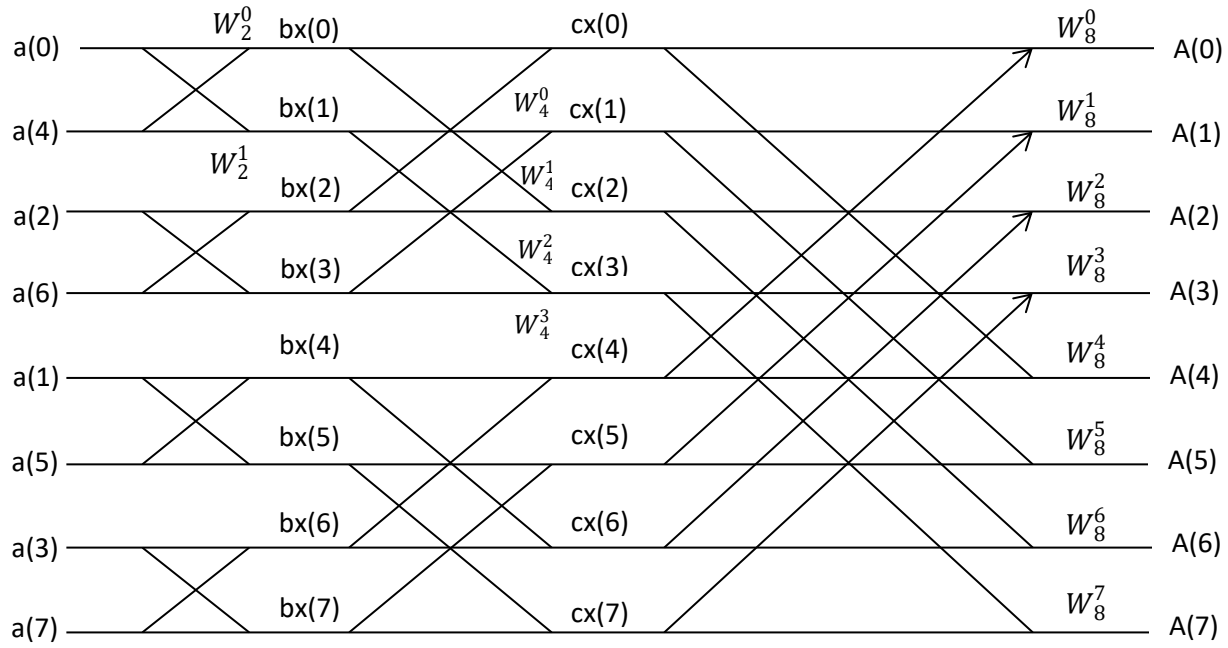


Fig 4.14 Butterfly diagram of 8-point FFT

An eight-point MUX (multiplexer) based FFT (Fast Fourier Transform) structure is a method for computing the FFT of a sequence of eight complex numbers using a multiplexer-based circuit. In an eight-point MUX based FFT structure, the input sequence of eight complex numbers is first arranged in a specific order and then passed through a series of multiplexers. The multiplexers select and combine specific elements of the input sequence to perform the required butterfly operations (a basic building block of the FFT algorithm) and produce the output sequence of eight complex numbers.

In an eight-point MUX-based FFT using distributed arithmetic, the butterfly operation is implemented using a distributed arithmetic circuit that computes the required twiddle factors using only binary addition and shift operations.

As we know A(k) = $\sum_{n=0}^{N-1} a(n) W_N^{kn}$; 0 ≤ k ≤ N-1 …(1)

Here for 8-point FFT N =8 and k ranges from 0 to N-1,

When we substitute N=8 in the above equation then we get

A(k) = $\sum_{n=0}^{7} a(n)W_8^{kn}$; 0 ≤ k ≤ 7 (since N=8)

A(k) = $a(0)W_8^{k(0)}+a(1)W_8^{k(1)}+a(2)W_8^{k(2)}+a(3)W_8^{k(3)}+a(4)W_8^{k(4)}+a(5)W_8^{k(5)}+a(6)W_8^{k(6)}$

+ $a(7)W_8^{k(7)}$

Eq-1 decimation into even, odd sequence

Here we divide the equation-1 in an even and odd sequence then the obtained equations are

A(k) = B(k) + W_N^KC(k); 0 ≤ k ≤ N/2-1 …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2); N/2 ≤ k ≤ N-1 …(3)

(Symmetry and periodicity)

k=0; A(0) = cx(1) + W_8^0 cx(5) k=4; A(4) = cx(1) − W_8^0 cx(5)

k=1; A(1) = cx(2) + W_8^1 cx(6) k=5; A(5) = cx(2) − W_8^1 cx(6)

k=2; A(2) = cx(3) + W_8^2 cx(7) k=6; A(6) = cx(3) − W_8^2 cx(7)

k=3; A(3) = cx(4) + W_8^3 cx(8) k=7; A(7) = cx(4) − W_8^3 cx(8)

As we know substituting the k=0 to k=7 in the above even and odd equations, we get

A(0) = $a(0) + a(1) + a(2) + a(3) + a(4) + a(5) + a(6) + a(7)$

A(1) = $a(0) + a(1)W_8^1 + a(2)W_8^2 + a(3)W_8^3 - a(4) - a(5)W_8^1 - a(6)W_8^2 - a(7)W_8^3$

A(2) = $a(0) + a(1)W_8^2 - a(2) - a(3)W_8^2 + a(4) + a(5)W_8^2 - a(6) - a(7)W_8^2$

A(3) = $a(0) + a(1)W_8^3 + a(2)W_8^2 + a(3)W_8^1 - a(4) - a(5)W_8^3 - a(6)W_8^2 - a(7)W_8^1$

A(4) = $a(0) - a(1) + a(2) - a(3) + a(4) - a(5) + a(6) - a(7)$

A(5) = $a(0) - a(1)W_8^1 + a(2)W_8^2 - a(3)W_8^3 - a(4) + a(5)W_8^1 - a(6)W_8^2 + a(7)W_8^3$

A(6) = $a(0) - a(1)W_8^2 - a(2) + a(3)W_8^2 + a(4) - a(5)W_8^2 - a(6) + a(7)W_8^2$

A(7) = $a(0) - a(1)W_8^3 - a(2)W_8^2 - a(3)W_8^1 - a(4) + a(5)W_8^3 + a(6)W_8^2 + a(7)W_8^3$

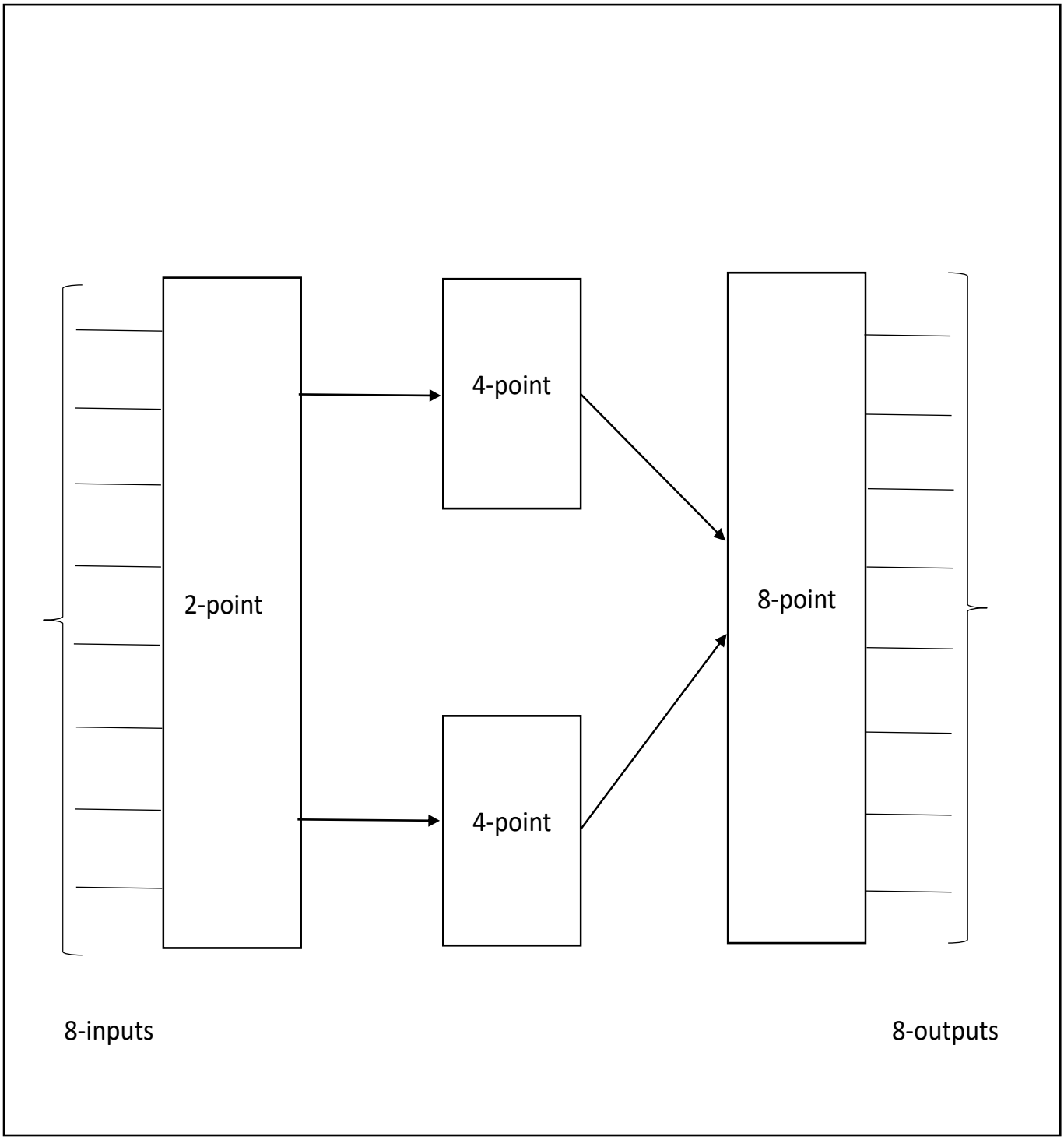


Fig 4.15 8-point 64-bit MUX based

4.5.4 SIXTEEN POINT 64 BIT LUTLESS FFT BASED ON DA

A sixteen-point 64-bit LUT less FFT based on distributed arithmetic (DA) is a type of FFT algorithm that can perform a 16-point FFT computation using DA techniques, without relying on lookup tables (LUTs). The 64-bit refers to the precision of the input and output data.

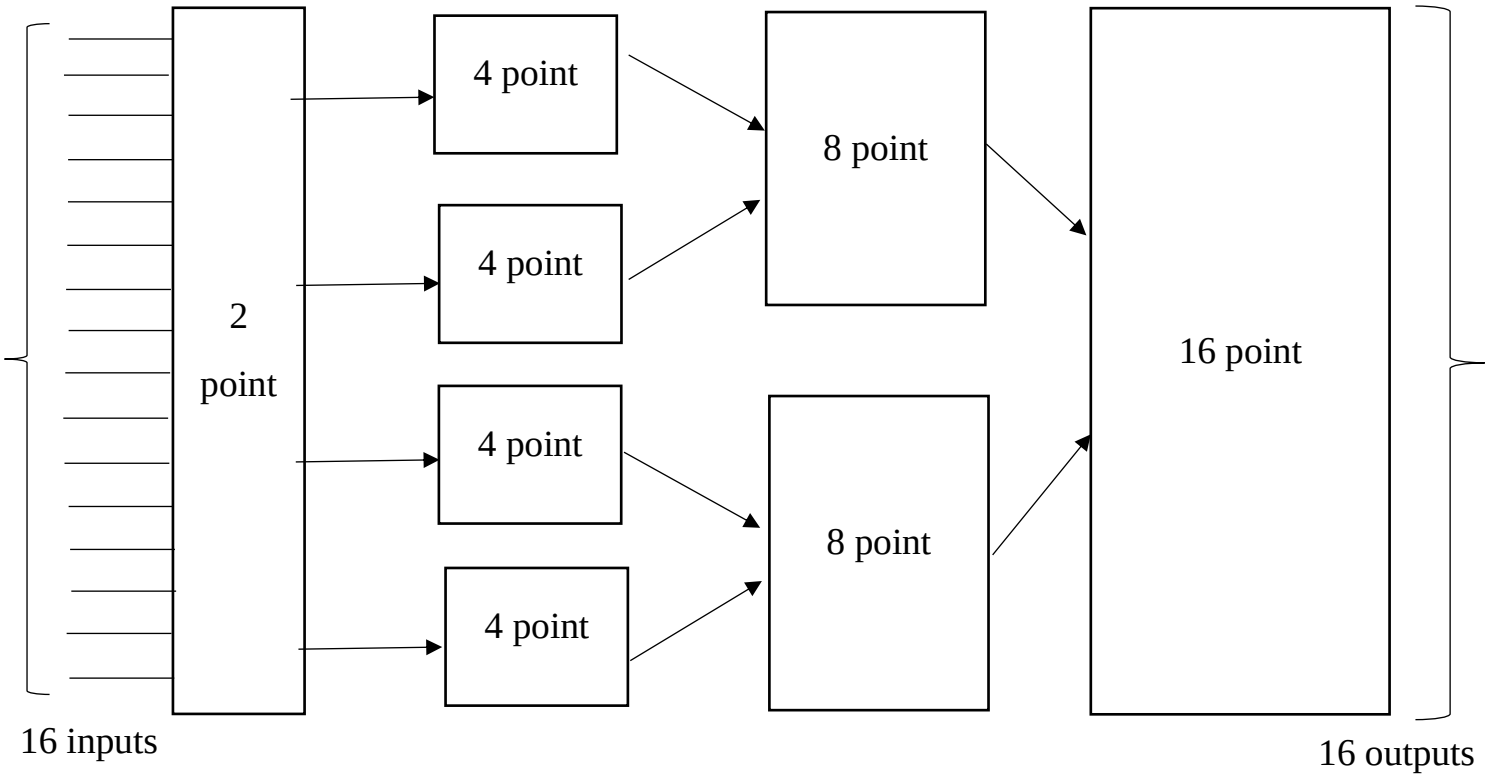


Fig 4.16 16-point MUX based FFT structure

A 16-point Multiplexer (MUX)-based FFT (Fast Fourier Transform) is a type of implementation of the FFT algorithm that uses multiplexers for the butterfly operations. Here are the steps to perform a 16-point MUX-based FFT:

Step 1: Arrange the input data into a bit-reversed order. This means that the first element of the input data should be swapped with the eighth, the second with the ninth, the third with the tenth, and so on.

Step 2: Divide the input data into two groups of eight points each.

Step 3: Perform an 8-point MUX-based FFT on each group of eight points. This involves performing butterfly operations using multiplexers.

Step 4: Multiply the second group of 8-point FFT results with the appropriate twiddle factors. The twiddle factors are complex numbers that are used in the FFT computation. They are derived from the roots of unity.

Step 5: Combine the two groups of 8-point FFT results by adding them element-wise.

Step 6: Divide the resulting data into four groups of four points each.

Step 7: Perform a 4-point MUX-based FFT on each group of four points.

Step 8: Multiply the second group of 4-point FFT results with the appropriate twiddle factors.

Step 9: Combine the two groups of 4-point FFT results by adding them element-wise.

Step 10: Divide the resulting data into eight groups of two points each.

Step 11: Perform a 2-point MUX-based FFT on each group of two points.

Step 12: Multiply the second group of 2-point FFT results with the appropriate twiddle factors.

Step 13: Combine the two groups of 2-point FFT results by adding them element-wise.

Step 14: The final result is the 16-point FFT.

As we know A(k) = $\sum_{n=0}^{N-1} a(n)W_N^{kn}$; $0 \leq k \leq N-1$ …(1)

Here for 16-point FFT N =16 and k ranges from 0 to N-1,

When we substitute N=16 in the above equation then we get

A(k) = $\sum_{n=0}^{15} a(n)W_{16}^{kn}$; $0 \leq k \leq 15$ (since N=16)

As we know substituting the k=0 to k=15 in the above even and odd equations, we get

A(k) = $a(0)W_{16}^{k(0)} + a(1)W_{16}^{k(1)} + a(2)W_{16}^{k(2)} + a(3)W_{16}^{k(3)} + - - - a(14)W_{16}^{k(14)} + a(15)W_{16}^{k(15)}$

A(0) = $a(0) + a(1) + a(2) + a(3) + - - - a(14) + a(15)$

A(1) = $a(0) + a(1)W_{16}^{1} + a(2)W_{16}^{2} + a(3)W_{16}^{3} + - - - a(14)W_{16}^{14} + a(15)W_{16}^{15}$

.

.

A(15) = $a(0) + a(1)W_{16}^{15} + a(2)W_{16}^{30} + a(3)W_{16}^{45} + - - - a(14)W_{16}^{210} + a(15)W_{16}^{225}$

Eq-1 decimation into even, odd sequence

A(k) = B(k) + W_N^KC(k) ; 0 ≤ k ≤ N/2-1 …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2) ; N/2 ≤ k ≤ N-1 …(3)

(Symmetry and periodicity)

The MUX-based implementation of the FFT algorithm uses multiplexers to perform the butterfly operations, which are the basic building blocks of the FFT algorithm. This type of implementation requires less hardware than other implementations, making it suitable for low-cost applications.

4.5.5 THIRTY-TWO POINT 64 BIT LUTLESS FFT BASED ON DA

The main purpose of a 32-point 64-bit FFT (Fast Fourier Transform) structure is to efficiently compute the Fourier transform of a 32-point input signal using 64-bit fixed-point arithmetic.

The use of 64-bit fixed-point arithmetic in the 32-point 64-bit FFT structure provides high precision and accuracy in the computation of the FFT, making it suitable for applications that require high accuracy in the frequency domain. The use of fixed-point arithmetic also simplifies the hardware implementation of the FFT structure, making it more efficient in terms of hardware resources and power consumption.

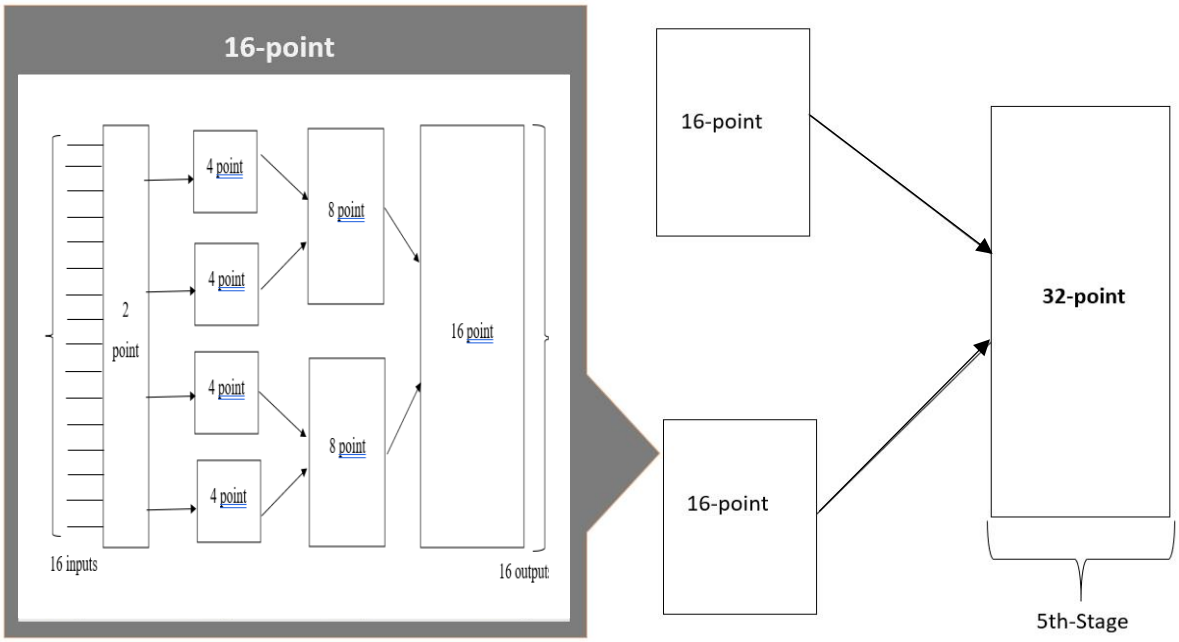


Fig 4.17 32-point 64 bit MUX based FFT structure

A 32 point FFT can be computed in 5 stages where each stage consists of 16 operations. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 32-points DFT can be divided into sixteen 2-point DFTs. Then these can be combined to get eight 4-point DFTs. Then these can be combined to get four 8-points DFTs. These can be combined to get two 16-points DFTs.

As we know A(k) = $\sum_{n=0}^{N-1} a(n)W_N^{kn}$; 0 ≤ k ≤ N-1 …(1)

Here for 16-point FFT N =32 and k ranges from 0 to N-1,

When we substitute N=32 in the above equation then we get

A(k) = $\sum_{n=0}^{31} a(n)W_{32}^{kn}$; 0 ≤ k ≤ 31 (since N=32)

As we know substituting the k=0 to k=31 in the above even and odd equations, we get

A(k) = $a(0)W_{32}^{k(0)}$ + $a(1)W_{32}^{k(1)}$+ $a(2)W_{32}^{k(2)}$+ $a(3)W_{32}^{k(3)}$+ - - - $a(30)W_{32}^{k(30)}$ + $a(31)W_{32}^{k(31)}$

A(0) = $a(0) + a(1) + a(2) + a(3)$ + - - - $a(30) + a(31)$

A(1) = $a(0)$ + $a(1)W_{32}^{1}$+ $a(2)W_{32}^{2}$ + $a(3)W_{32}^{3}$ + - - - $a(30)W_{32}^{30}$ + $a(31)W_{32}^{31}$

.

.

.

A(31) = $a(0)$ + $a(1)W_{32}^{31}$ + $a(2)W_{32}^{62}$ + $a(3)W_{16}^{93}$ + - - - $a(30)W_{16}^{930}$ + $a(15)W_{16}^{961}$

Eq-1 decimation into even, odd sequence

A(k) = B(k) + W_N^KC(k) ; 0 ≤ k ≤ N/2-1 …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2) ; N/2 ≤ k ≤ N-1 …(3)

(Symmetry and periodicity)

4.5.6 SIXTY-FOUR POINT 64 BIT LUTLESS FFT BASED ON DA

A 64-point 64-bit FFT (Fast Fourier Transform) structure can offer several advantages over other FFT structures with different point sizes and bit widths. Here are some of the main advantages:

1. **High precision**: The 64-bit data format allows for high precision in the calculations, which is particularly useful in applications that require accurate frequency analysis. The high precision can reduce errors due to quantization, resulting in better signal fidelity.
2. **Efficient computation**: The 64-point FFT structure is a good compromise between frequency resolution and computation complexity. It can provide sufficient resolution for most applications while being computationally efficient, meaning it can handle the FFT calculation in a reasonable amount of time.
3. **Compatibility with hardware**: 64-point FFT structures are often supported by a wide range of hardware platforms, including CPUs, DSPs, and FPGAs. This makes it easy to implement FFT algorithms in a variety of systems without the need for specialized hardware.
4. **Easy to analyze**: A 64-point FFT structure can provide a good balance between time and frequency domain resolution, making it easy to analyze the frequency content of signals in real-time applications.

A sixty-four point 64-bit LUTless FFT based on distributed arithmetic (DA) is a type of FFT algorithm that can perform a 64-point FFT computation using DA techniques, without relying on lookup tables (LUTs). The 64-bit refers to the precision of the input and output data.

A 64-points FFT can be computed in 6 stages where each stage consists of 32 operations. Butterfly operation in a Radix-2 algorithm is that part of FFT computation, that provides the Fourier transform of two-point sequence in a simplified manner. In this case 64-points DFT can be divided into thirty-two 2-point DFTs. Then these can be combined to get sixteen 4-point DFTs. Then these can be combined to get eight 8-points DFTs. These can be combined to get four 16-points DFTs. Then these four 16-points are combined to two 32-points DFTs. Finally, we get 64- point DFT.

As we know $A(k) = \sum_{n=0}^{N-1} a(n) W_N^{kn}$; $0 \leq k \leq N-1$ …(1)

Here for 64 point FFT N =64 and k ranges from 0 to N-1,

When we substitute N=64 in the above equation then we get

A(k) = $\sum_{n=0}^{63} a(n)W_{64}^{kn}$; 0 ≤ k ≤ 63 (since N=64)

As we know substituting the k=0 to k=63 in the above even and odd equations, we get

A(k) = $a(0)W_{64}^{k(0)}$ + $a(1)W_{64}^{k(1)}$+ $a(2)W_{64}^{k(2)}$+ $a(3)W_{64}^{k(3)}$+ - - - $a(62)W_{64}^{k(62)}$ + $a(63)W_{64}^{k(63)}$

A(0) = $a(0) + a(1) + a(2) + a(3)$ + - - - $a(62) + a(63)$

A(1) = $a(0)$ + $a(1)W_{64}^{1}$+ $a(2)W_{64}^{2}$ + $a(3)W_{64}^{3}$ + - - - $a(62)W_{64}^{62}$ + $a(63)W_{64}^{63}$

.

.

A(63) = $a(0)$ + $a(1)W_{64}^{63}$ + $a(2)W_{64}^{126}$ + $a(3)W_{64}^{189}$ + - - - $a(62)W_{64}^{3906}$ + $a(63)W_{64}^{3969}$

Eq-1 decimation into even, odd sequence

A(k) = B(k) + W_N^KC(k) ; 0 ≤ k ≤ N/2-1 …(2)

B(k) and C(k) → Periodic with N/2

A(k) = B(k-N/2) +W_N^K C(k-N/2) ; N/2 ≤ k ≤ N-1 …(3)

(Symmetry and periodicity)

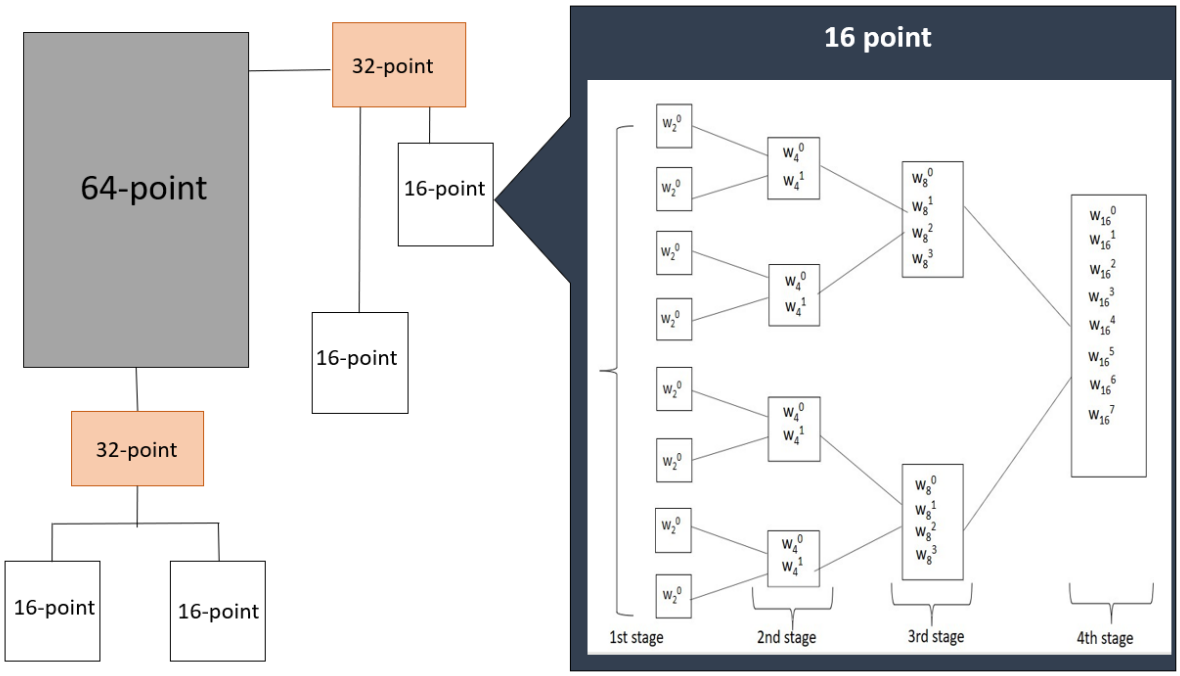


Fig 4.18 64-point 64-bit MUX based FFT structure

ADVANTAGES OF LUT

LUT stands for Look-Up Table, which is a type of data structure used in computer science and engineering. Here are some advantages of using LUTs:

1. Fast computation: LUTs can perform complex operations, such as mathematical functions or image processing algorithms, in a single clock cycle. This makes them much faster than traditional computation methods, which require multiple instructions and clock cycles.
2. Memory efficient: LUTs require very little memory compared to other algorithms, because they store the results of pre-computed operations. This makes them ideal for use in small embedded systems, where memory is often limited.
3. Flexibility: LUTs can be programmed to perform any type of operation, as long as it can be expressed as a function with a finite number of inputs and outputs. This means they can be used for a wide range of applications, from image processing to cryptography.
4. Accuracy: LUTs can provide very accurate results, because they store pre-computed values at high precision. This is particularly useful for applications where accuracy is critical, such as scientific simulations or financial modeling.
5. Ease of implementation: LUTs are relatively easy to implement in hardware or software, because they do not require complex algorithms or iterative processes. This makes them a popular choice for FPGA and ASIC designs, as well as software libraries and APIs.

DISADVANTAGES OF LUT

While LUTs have many advantages, they also have some disadvantages:

1. Limited range: LUTs can only perform operations that can be expressed as a function with a finite number of inputs and outputs. This means that they may not be suitable for complex operations that require iterative or recursive processing.
2. Memory requirements: While LUTs are memory-efficient compared to other algorithms, they still require some amount of memory to store the pre-computed values. This can be a limitation in systems with very limited memory, such as small, embedded devices.

ADVANTAGES OF MUX BASED DESIGN

Mux (multiplexer) based designs have several advantages, including:

1. Reduced gate count: Mux based designs can reduce the number of gates required for a given circuit, as they allow for multiple inputs to be selected by a single control signal. This can reduce the complexity and cost of the design.
2. Faster operation: Mux based designs can be faster than other designs, as the selection of an input can be done in a single clock cycle. This can be important for applications that require high speed operation.
3. Reduced power consumption: Mux based designs can consume less power than other designs, as the reduced number of gates can reduce the capacitance and switching activity in the circuit. This can be important for portable or battery-powered devices.
4. Simplified layout: Mux based designs can simplify the layout of a circuit, as the multiplexers can be placed close to the inputs they select. This can reduce the routing complexity and improve the signal integrity of the design.
5. Flexibility: Mux based designs are highly flexible, as they can be used for a wide range of applications. They can be used for data selection, address decoding, and even for implementing more complex logic functions. This makes them a popular choice for digital circuit design.

DISADVANTAGES OF MUX BASED DESIGN

While mux (multiplexer) based designs have several advantages, they also have some disadvantages, including:

1. Limited number of inputs: Muxes have a limited number of inputs, typically 2^n where n is the number of select lines. This means that they may not be suitable for designs that require more than a few inputs.
2. Limited functionality: Muxes can only select one input at a time, which limits their functionality compared to other types of logic gates. This means that they may not be suitable for more complex logic functions.
3. Increased complexity for complex designs: While muxes can simplify the layout of a circuit, they can increase the complexity of the circuit for more complex designs. This is because muxes require additional select lines and may require additional logic gates for decoding and control.

4.5 APPLICATIONS OF LUT AND LUTLESS

Lookup tables (LUTs) and LUT less techniques have various applications in digital signal processing, particularly in the design and implementation of signal processing algorithms. Some of the applications of LUT and LUT less techniques are:

1. **Digital signal processing**: LUTs are commonly used in digital signal processing (DSP) applications to store and retrieve precomputed data, such as sine and cosine values, that are used in signal processing algorithms. LUT less techniques, on the other hand, are used to reduce the memory requirements and power consumption of DSP algorithms, particularly in the implementation of FFT and FIR filters.

2. **FPGA and ASIC design**: LUTs are commonly used in field-programmable gate array (FPGA) and application-specific integrated circuit (ASIC) designs to implement digital logic functions, such as AND, OR, and NOT gates. LUT less techniques are also used in FPGA and ASIC designs to reduce the size and power consumption of the circuit.

3. **Image and video processing**: LUTs are used in image and video processing applications to store and retrieve precomputed lookup tables that are used in colour space conversion, gamma correction, and other image processing algorithms. LUT less techniques are also used in image and video processing applications to reduce the memory requirements and power consumption of the algorithms.

4. **Communications**: LUTs and LUT less techniques are used in various communication applications, such as digital modulation, demodulation, error correction coding, and decoding. LUTs are used to store and retrieve precomputed data that are used in these algorithms, while LUT less techniques are used to reduce the memory requirements and power consumption of the algorithms.

5. **Control systems**: LUTs and LUT less techniques are used in control system applications, such as PID controllers and Kalman filters, to store and retrieve precomputed data that are used in the control algorithms. LUT less techniques are also used to reduce the memory requirements and power consumption of the algorithms.

Overall, LUTs and LUT less techniques are essential tools in digital signal processing and various other applications that require efficient and optimized algorithms. The choice between using LUTs and LUT less techniques depend on the specific requirements of the application, such as performance, power consumption, and memory requirements.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 EXISTING TWO POINT 64-BIT FFT

On simulating the 2-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

{a0, a1} = {1, 2}, twiddle factor w02 = 1

The outputs obtained are {A0, A1} = {3, -1}

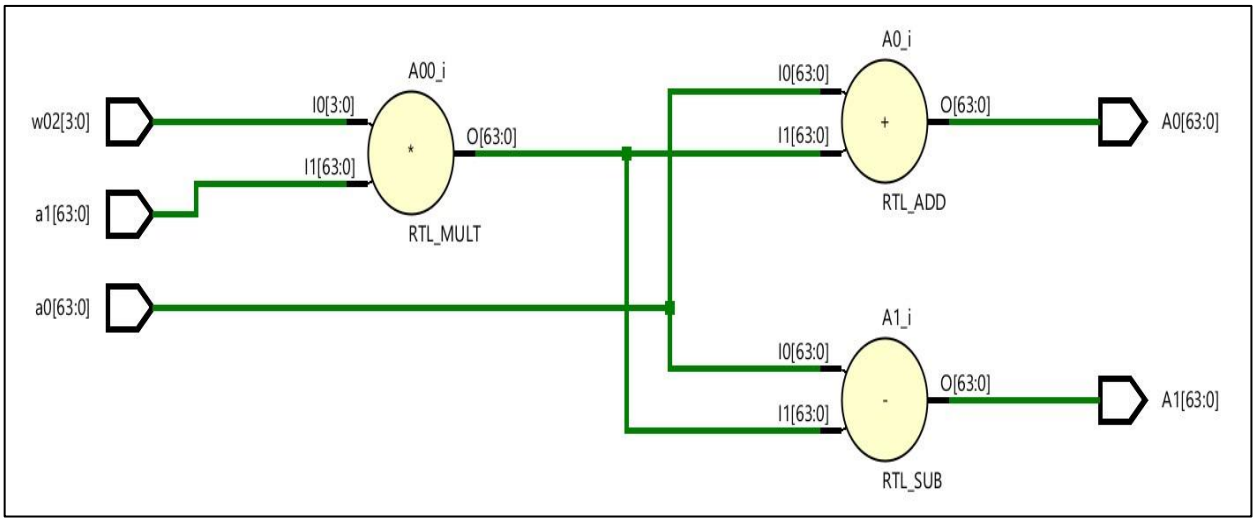


Figure 5.1.1 Schematic of 2-point 64-bit FFT in Xilinx Vivado 2022.2

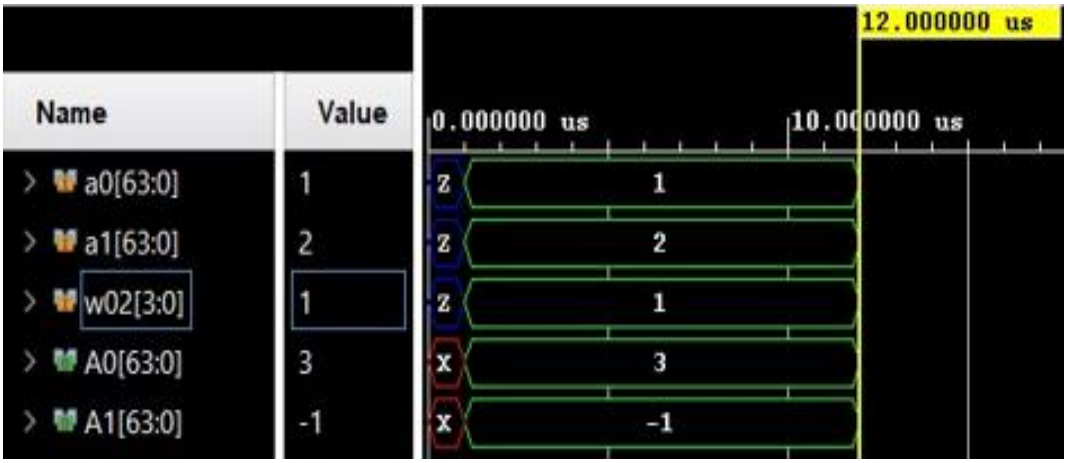


Figure 5.1.2 Simulation result of 2-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of 2-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

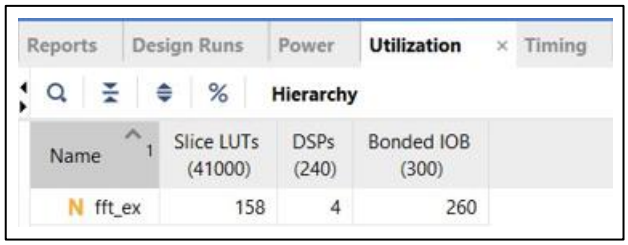
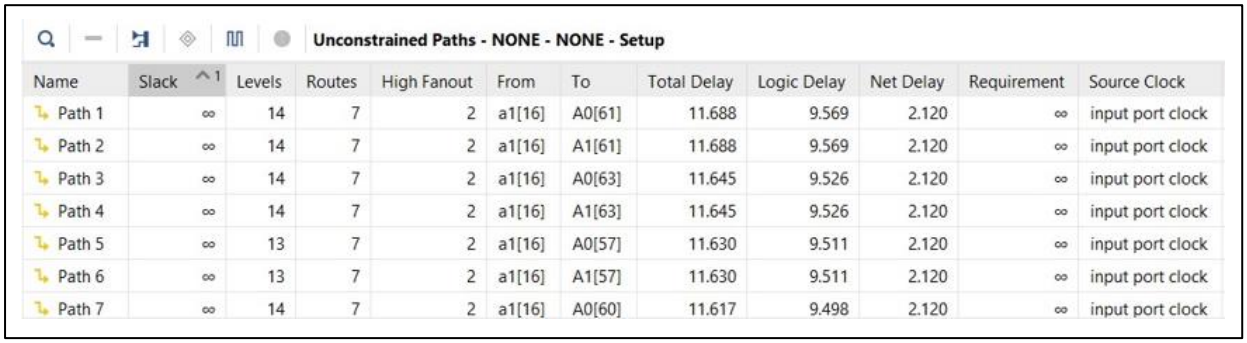
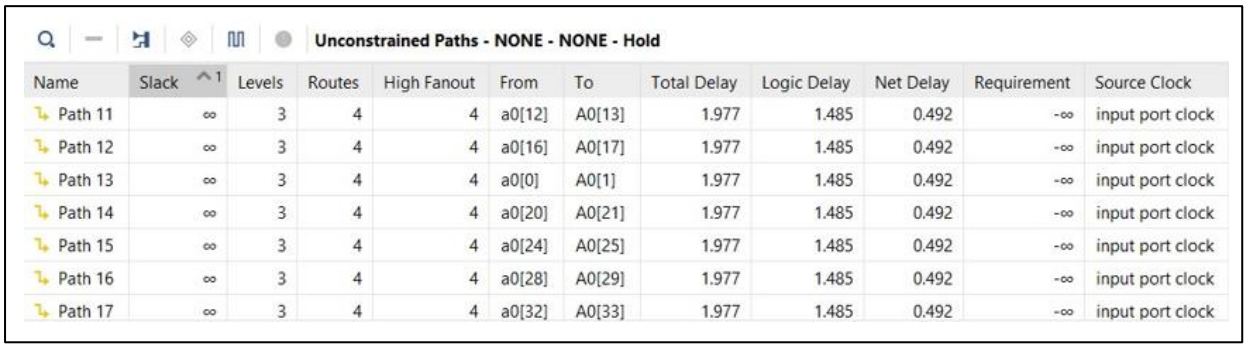


Figure 5.1.3 Utilization report of 2-point 64-bit FFT in Xilinx Vivado 2022.2



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	14	7	2	a1[16]	A0[61]	11.688	9.569	2.120	∞	input port clock
Path 2	∞	14	7	2	a1[16]	A1[61]	11.688	9.569	2.120	∞	input port clock
Path 3	∞	14	7	2	a1[16]	A0[63]	11.645	9.526	2.120	∞	input port clock
Path 4	∞	14	7	2	a1[16]	A1[63]	11.645	9.526	2.120	∞	input port clock
Path 5	∞	13	7	2	a1[16]	A0[57]	11.630	9.511	2.120	∞	input port clock
Path 6	∞	13	7	2	a1[16]	A1[57]	11.630	9.511	2.120	∞	input port clock
Path 7	∞	14	7	2	a1[16]	A0[60]	11.617	9.498	2.120	∞	input port clock



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	4	a0[12]	A0[13]	1.977	1.485	0.492	-∞	input port clock
Path 12	∞	3	4	4	a0[16]	A0[17]	1.977	1.485	0.492	-∞	input port clock
Path 13	∞	3	4	4	a0[0]	A0[1]	1.977	1.485	0.492	-∞	input port clock
Path 14	∞	3	4	4	a0[20]	A0[21]	1.977	1.485	0.492	-∞	input port clock
Path 15	∞	3	4	4	a0[24]	A0[25]	1.977	1.485	0.492	-∞	input port clock
Path 16	∞	3	4	4	a0[28]	A0[29]	1.977	1.485	0.492	-∞	input port clock
Path 17	∞	3	4	4	a0[32]	A0[33]	1.977	1.485	0.492	-∞	input port clock

Figure 5.1.4 Delay report of 2-point 64-bit FFT in Xilinx Vivado 2022.2

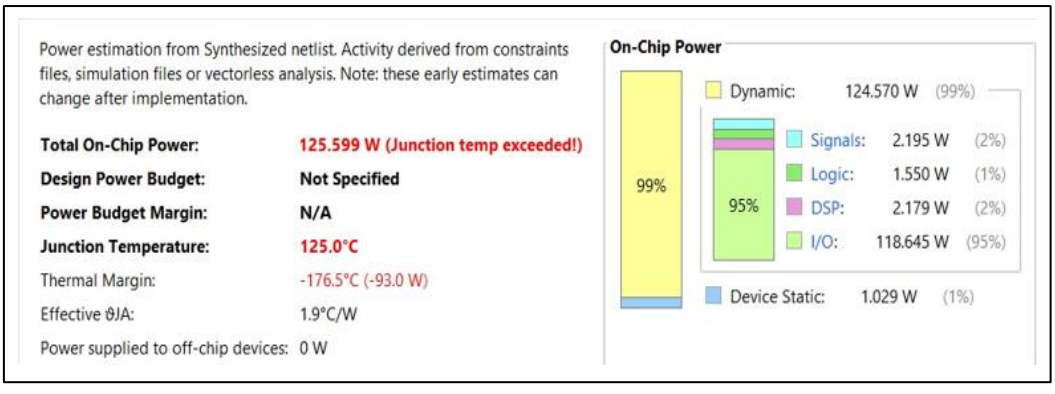


Figure 5.1.5 Power report of 2-point 64-bit FFT in Xilinx Vivado 2022.2

5.2 PROPOSED TWO POINT 64-BIT FFT BASED ON DA

On simulating the proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: {a1, a2} = {1, 2}, twiddle factor W20 = 1 and

- for select line w = 3, the outputs obtained are {bx0, bx1} = {3, -1}
- for select line w = 2, the outputs obtained are {bx0, bx1} = {1, 1}
- for select line w = 1, the outputs obtained are {bx0, bx1} = {2, -2}
- for select line w = 0, the outputs obtained are {bx0, bx1} = {0, 0}

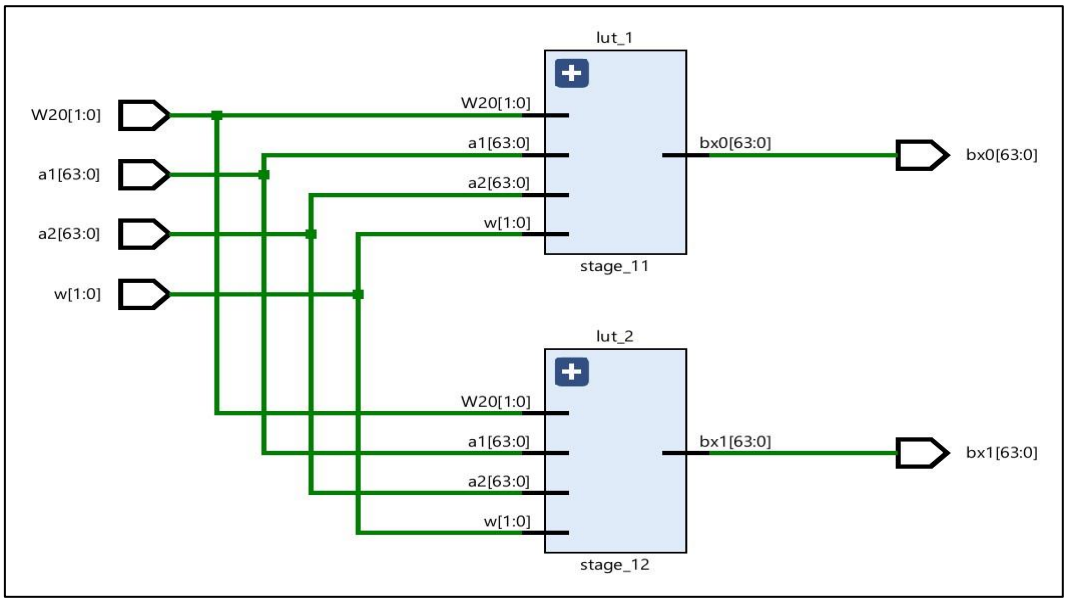


Figure 5.2.1 Schematic of proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2

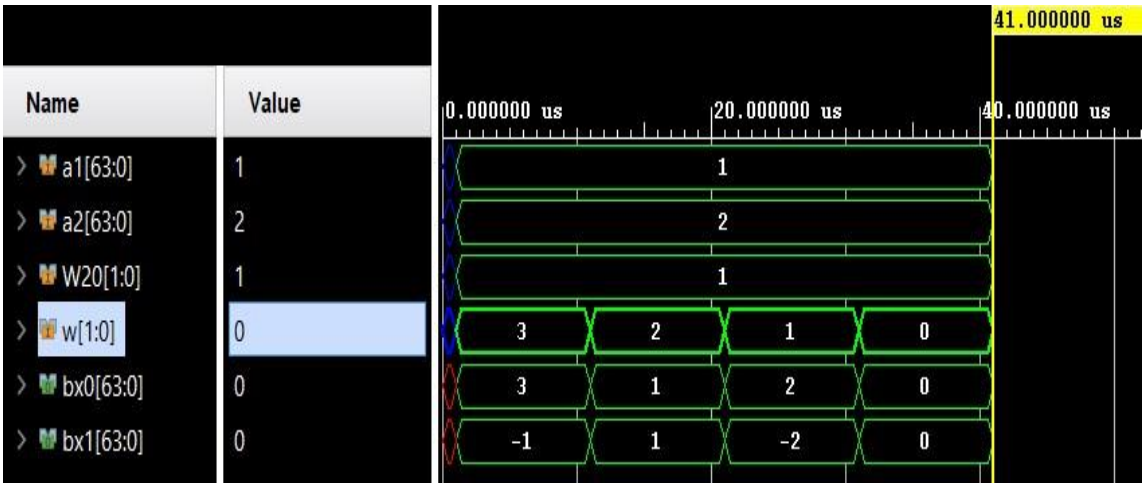


Figure 5.2.2 Simulation result of proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted

Reports | Design Runs | **Utilization** | Timing

Hierarchy

Name	Slice LUTs (41000)	Bonded IOB (300)
btf	56	36
m1 (stage_11)	35	0
m2 (stage_12)	21	0

Figure 5.2.3 Utilization report of proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	8	6	31	W20[0]	bx0[7]	6.359	4.576	1.783	∞	input port clock
Path 2	∞	8	6	31	W20[0]	bx1[7]	6.359	4.576	1.783	∞	input port clock
Path 3	∞	8	6	31	W20[0]	bx0[6]	6.316	4.552	1.764	∞	input port clock
Path 4	∞	8	6	31	W20[0]	bx1[6]	6.316	4.552	1.764	∞	input port clock
Path 5	∞	8	6	31	W20[0]	bx0[5]	6.274	4.491	1.783	∞	input port clock
Path 6	∞	8	6	31	W20[0]	bx1[5]	6.274	4.491	1.783	∞	input port clock
Path 7	∞	8	6	31	W20[0]	bx0[4]	6.200	4.415	1.785	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	6	a1[0]	bx0[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	6	a1[1]	bx0[1]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	6	a1[2]	bx0[2]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	6	a1[3]	bx0[3]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	6	a1[4]	bx0[4]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	6	a1[5]	bx0[5]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	6	a1[6]	bx0[6]	1.905	1.413	0.492	-∞	input port clock

Figure 5.2.4 Delay report of proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**9.034 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**42.0°C**
Thermal Margin:	43.0°C (22.6 W)
Effective θJA:	1.9°C/W

On-Chip Power

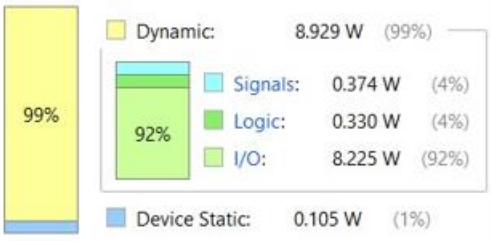


Figure 5.2.5 Power report of proposed 2-point 64-bit FFT in Xilinx Vivado 2022.2

5.3 EXISTING FOUR POINT 64-BIT FFT

On simulating the 4-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

{x0, x1, x2, x3} = {0, 1, 2, 3}, twiddle factors w20 = w40 = w41 = 1

The outputs obtained are {X1, X2, X3, X4} = {6, -4, -2, 0}

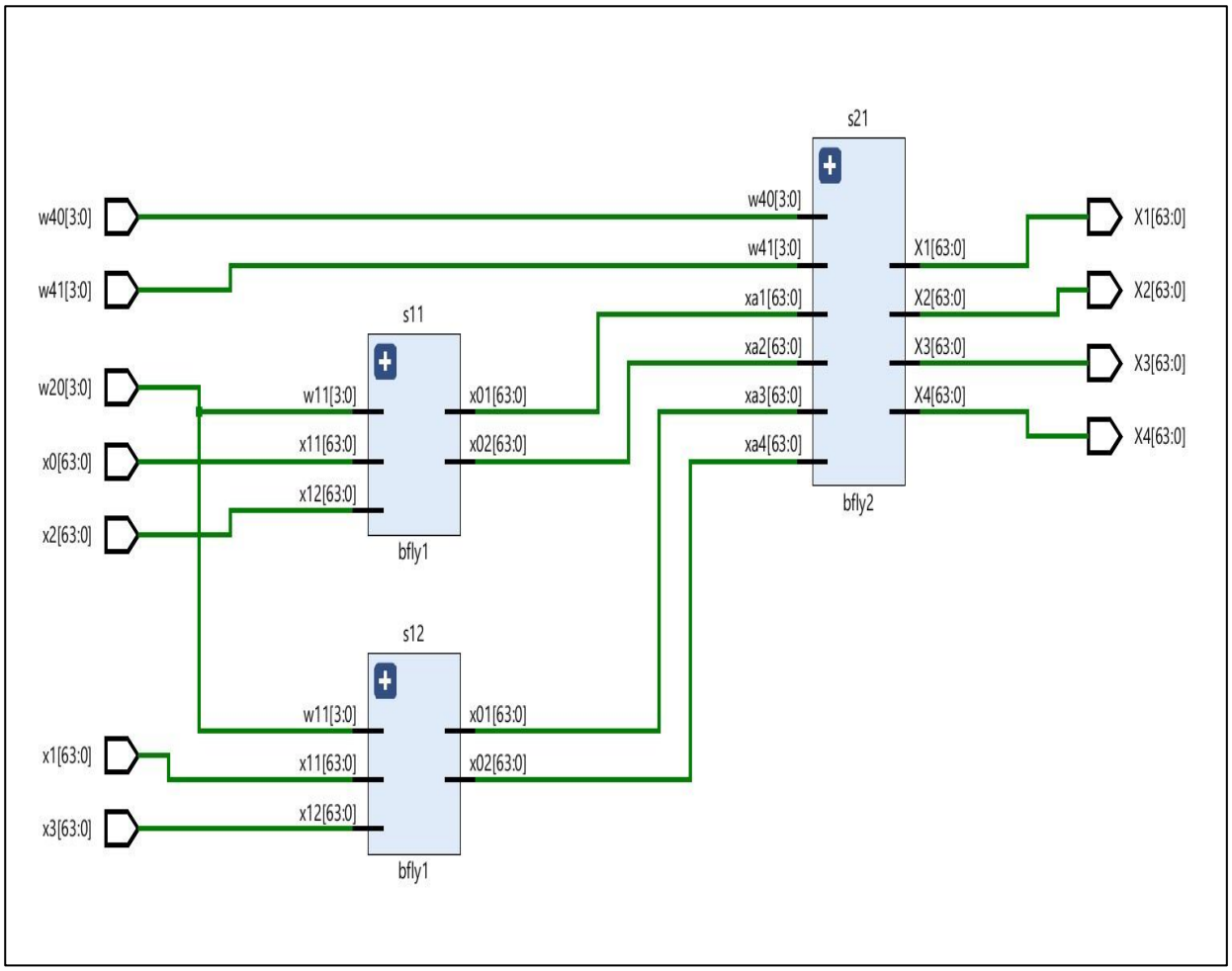


Figure 5.3.1 Schematic of 4-point 64-bit FFT in Xilinx Vivado 2022.2

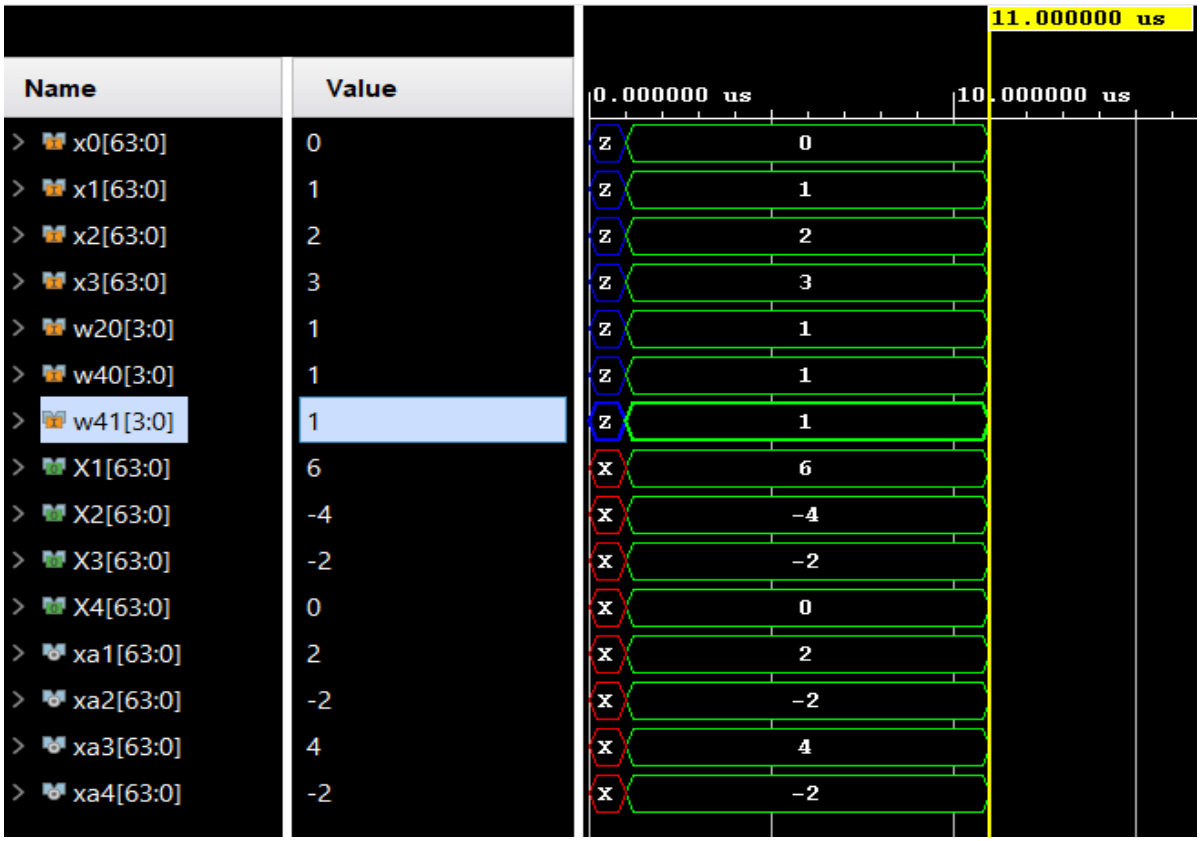


Figure 5.3.2 Simulation result of 4-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of 4-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

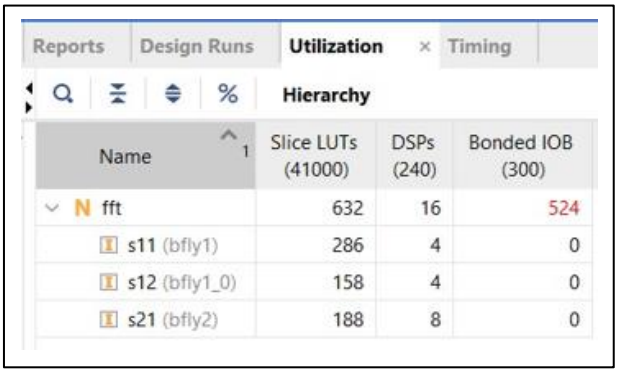


Figure 5.3.3 Utilization report of 4-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	19	11	2	x3[16]	X1[61]	19.157	15.408	3.750	∞	input port clock
Path 2	∞	19	11	2	x3[16]	X2[61]	19.157	15.408	3.750	∞	input port clock
Path 3	∞	19	11	2	x3[16]	X3[61]	19.157	15.408	3.750	∞	input port clock
Path 4	∞	19	11	2	x3[16]	X4[61]	19.157	15.408	3.750	∞	input port clock
Path 5	∞	19	11	2	x3[16]	X1[63]	19.114	15.365	3.750	∞	input port clock
Path 6	∞	19	11	2	x3[16]	X2[63]	19.114	15.365	3.750	∞	input port clock
Path 7	∞	19	11	2	x3[16]	X3[63]	19.114	15.365	3.750	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	5	5	4	x0[12]	X1[13]	2.205	1.602	0.603	-∞	input port clock
Path 12	∞	5	5	4	x0[16]	X1[17]	2.205	1.602	0.603	-∞	input port clock
Path 13	∞	5	5	4	x0[0]	X1[1]	2.205	1.602	0.603	-∞	input port clock
Path 14	∞	5	5	4	x0[20]	X1[21]	2.205	1.602	0.603	-∞	input port clock
Path 15	∞	5	5	4	x0[24]	X1[25]	2.205	1.602	0.603	-∞	input port clock
Path 16	∞	5	5	4	x0[28]	X1[29]	2.205	1.602	0.603	-∞	input port clock
Path 17	∞	5	5	4	x0[32]	X1[33]	2.205	1.602	0.603	-∞	input port clock

Figure 5.3.4 Delay report of 4-point 64-bit FFT in Xilinx Vivado 2022.2

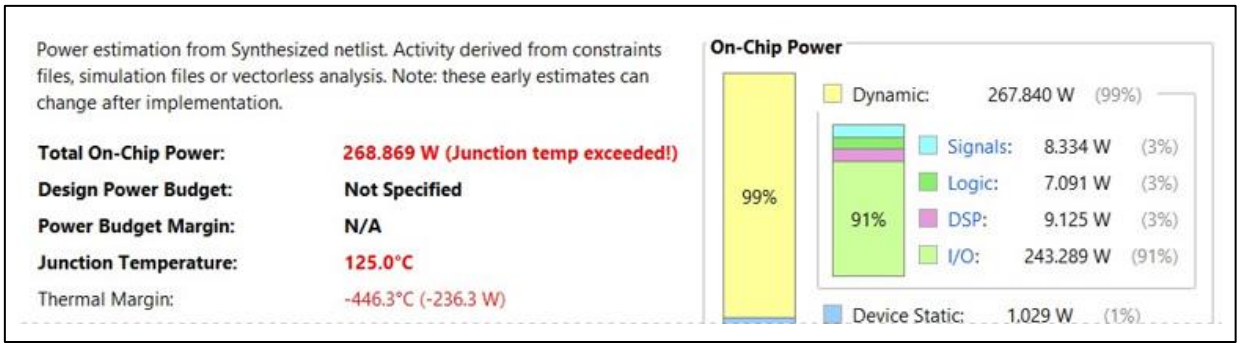


Figure 5.3.5 Power report of 4-point 64-bit FFT in Xilinx Vivado 2022.2

5.4 PROPOSED FOUR POINT 64-BIT FFT BASED ON DA

On simulating the proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: {a0, a1, a2, a3} = {0, 1, 2, 3}, twiddle factors W20 = W40 = W41 = 1 and

- for select line w = 3, the outputs obtained are {A0, A1, A2, A3} = {6, -4, -2, 0}
- for select line w = 2, the outputs obtained are {A0, A1, A2, A3} = {0, 0, 0, 0}
- for select line w = 1, the outputs obtained are {A0, A1, A2, A3} = {3, -3, -3, 3}
- for select line w = 0, the outputs obtained are {A0, A1, A2, A3} = {0, 0, 0, 0}

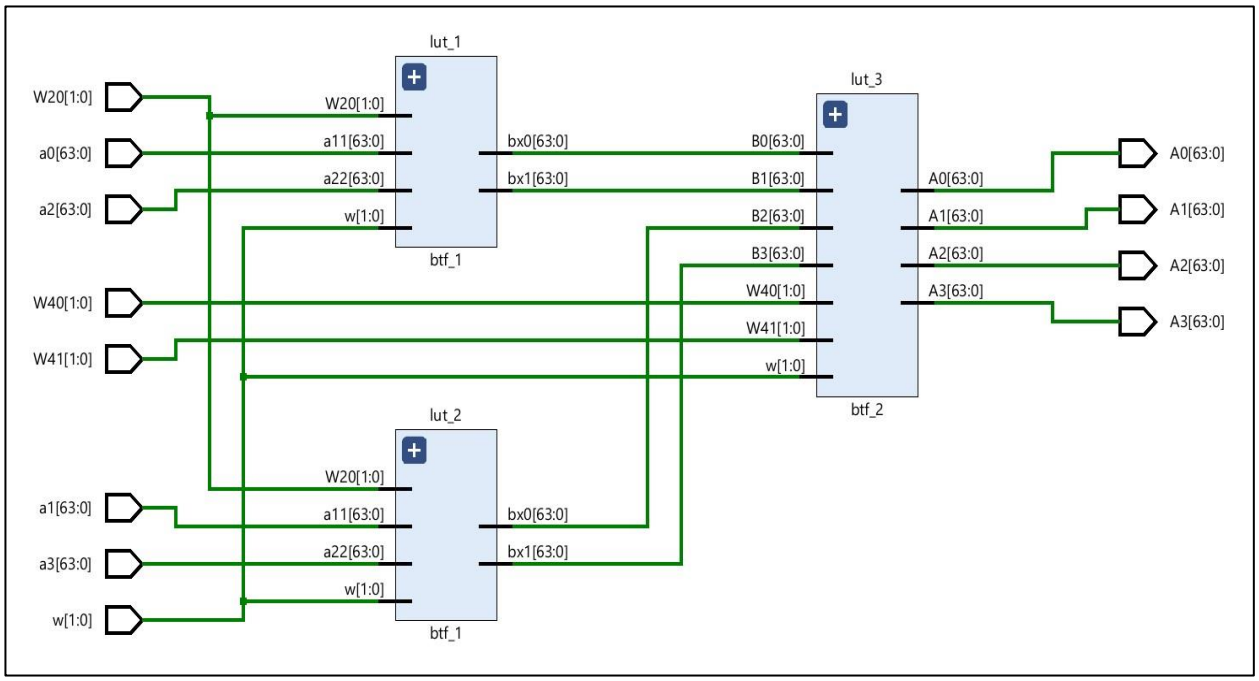


Figure 5.4.1 Schematic of proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2

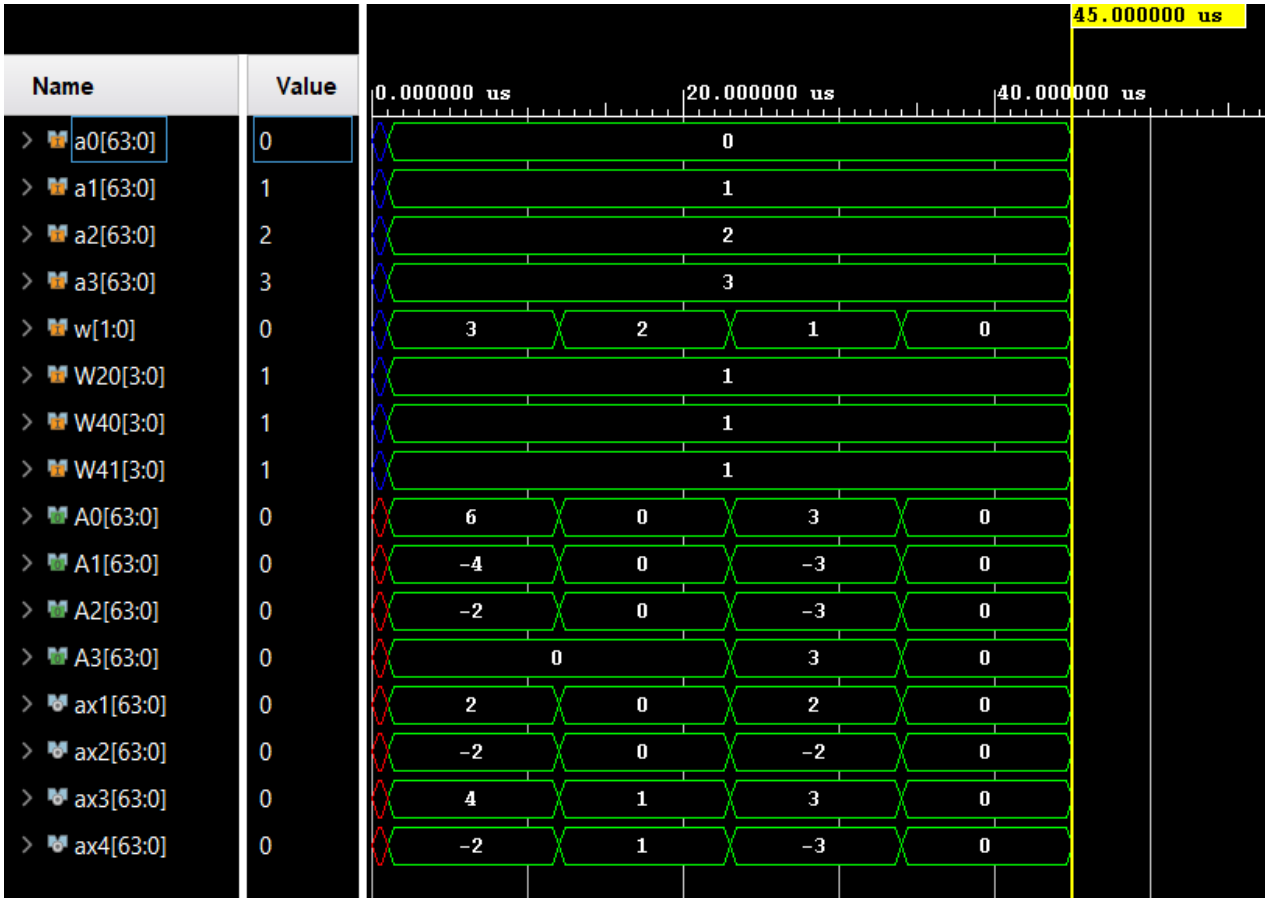
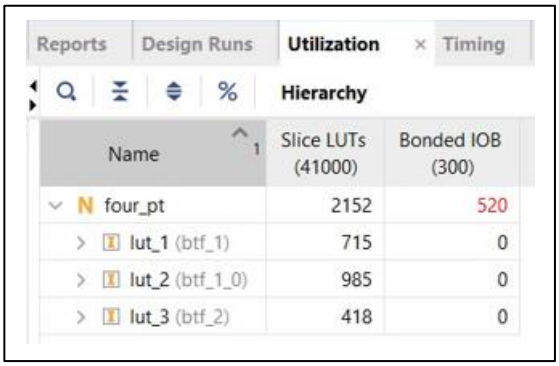


Figure 5.4.2 Simulation result of proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.



Reports | Design Runs | Utilization | Timing

Hierarchy

Name	Slice LUTs (41000)	Bonded IOB (300)
four_pt	2152	520
lut_1 (btf_1)	715	0
lut_2 (btf_1_0)	985	0
lut_3 (btf_2)	418	0

Figure 5.4.3 Utilization report of proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	31	13	4	a3[1]	A1[63]	13.873	8.873	5.000	∞	input port clock
Path 2	∞	31	13	4	a3[1]	A3[63]	13.862	8.873	4.989	∞	input port clock
Path 3	∞	30	12	4	a3[1]	A3[62]	13.844	8.969	4.875	∞	input port clock
Path 4	∞	31	13	4	a3[1]	A1[62]	13.830	8.849	4.981	∞	input port clock
Path 5	∞	30	13	4	a3[1]	A1[61]	13.792	8.821	4.971	∞	input port clock
Path 6	∞	30	13	4	a3[1]	A3[61]	13.781	8.821	4.960	∞	input port clock
Path 7	∞	30	12	4	a3[1]	A3[58]	13.733	8.850	4.883	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	1046	w[1]	A0[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	1046	w[1]	A0[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	1046	w[1]	A0[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	1046	w[1]	A0[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	1046	w[1]	A0[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	1046	w[1]	A0[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	1046	w[1]	A0[15]	1.905	1.413	0.492	-∞	input port clock

Figure 5.4.4 Delay report of proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2

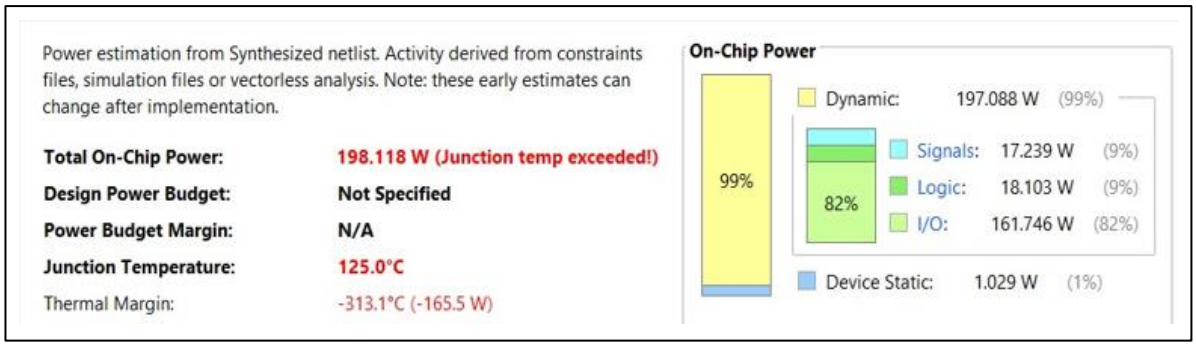


Figure 5.4.5 Power report of proposed 4-point 64-bit FFT in Xilinx Vivado 2022.2

5.5 EXISTING EIGHT POINT 64-BIT FFT

On simulating the 8-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

{x0, x1, x2, x3, x4, x5, x6, x7} = {a0, a1, a2, a3, a4, a5, a6, a7} = {1, 2, 3, 4, 4, 3, 2, 1}, twiddle factors W20 = W40 = W41 = W80 = W81 = W82 = W83 = 1

The outputs obtained are {X1, X2, X3, X4, X5, X6, X7, X8} = {A0, A1, A2, A3, A4, A5, A6, A7} = {20, 0, 0, -8, 0, -4, 0, 0}

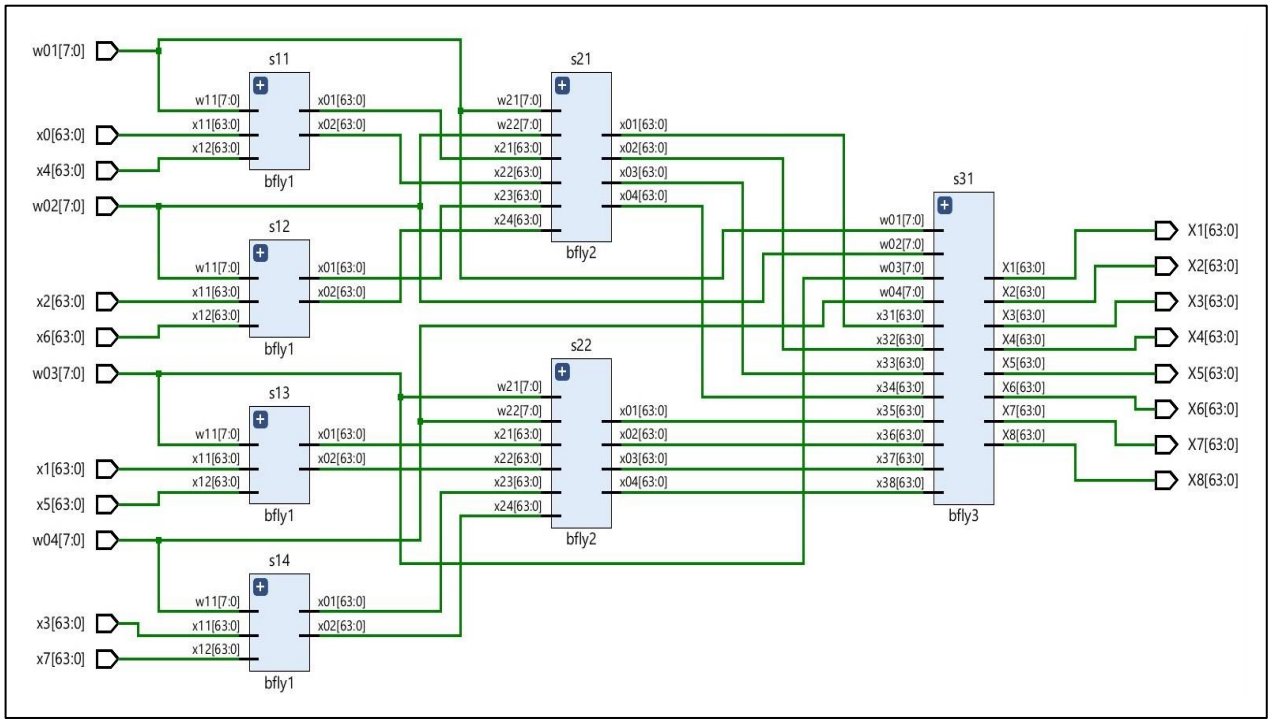


Figure 5.5.1 Schematic of 8-point 64-bit FFT in Xilinx Vivado 2022.2

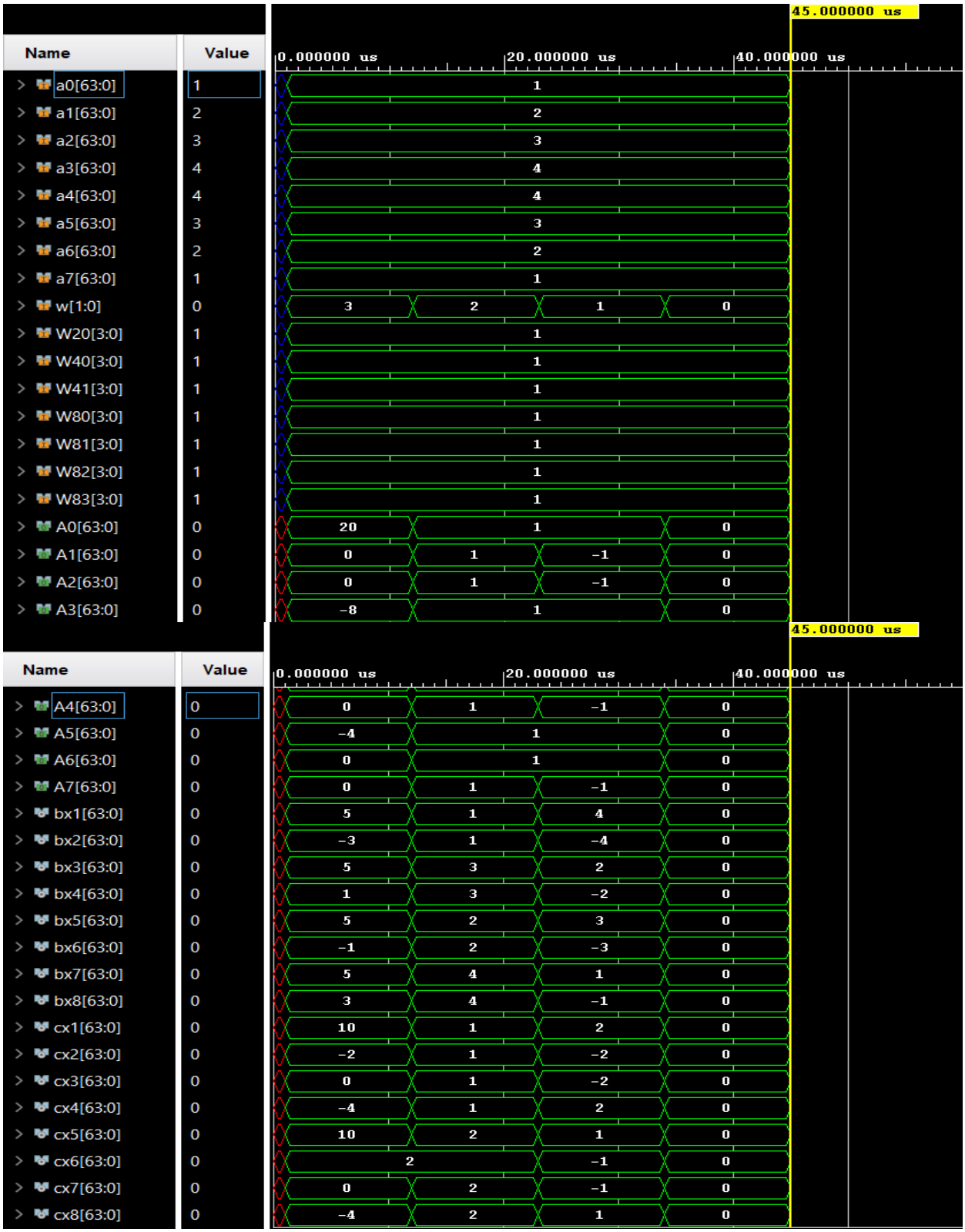


Figure 5.5.2 Simulation results of 8-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of 8-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Reports | Design Runs | Power | **Utilization** | Timing

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
fft	1896	48	1056
s11 (bfly1)	286	4	0
s12 (bfly1_0)	158	4	0
s13 (bfly1_1)	286	4	0
s14 (bfly1_2)	158	4	0
s21 (bfly2)	444	8	0
s22 (bfly2_3)	188	8	0
s31 (bfly3)	376	16	0

Figure 5.5.3 Utilization report of 8-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	28	14	2	x7[16]	X1[61]	25.465	20.085	5.380	∞	input port clock
Path 2	∞	28	14	2	x7[16]	X2[61]	25.465	20.085	5.380	∞	input port clock
Path 3	∞	28	14	2	x7[16]	X3[61]	25.465	20.085	5.380	∞	input port clock
Path 4	∞	28	14	2	x7[16]	X4[61]	25.465	20.085	5.380	∞	input port clock
Path 5	∞	28	14	2	x7[16]	X5[61]	25.465	20.085	5.380	∞	input port clock
Path 6	∞	28	14	2	x7[16]	X6[61]	25.465	20.085	5.380	∞	input port clock
Path 7	∞	28	14	2	x7[16]	X7[61]	25.465	20.085	5.380	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	7	6	4	x0[12]	X1[13]	2.433	1.719	0.715	-∞	input port clock
Path 12	∞	7	6	4	x0[16]	X1[17]	2.433	1.719	0.715	-∞	input port clock
Path 13	∞	7	6	4	x0[0]	X1[1]	2.433	1.719	0.715	-∞	input port clock
Path 14	∞	7	6	4	x0[20]	X1[21]	2.433	1.719	0.715	-∞	input port clock
Path 15	∞	7	6	4	x0[24]	X1[25]	2.433	1.719	0.715	-∞	input port clock
Path 16	∞	7	6	4	x0[28]	X1[29]	2.433	1.719	0.715	-∞	input port clock
Path 17	∞	7	6	4	x0[32]	X1[33]	2.433	1.719	0.715	-∞	input port clock

Figure 5.5.4 Delay report of 8-point 64-bit FFT in Xilinx Vivado 2022.2

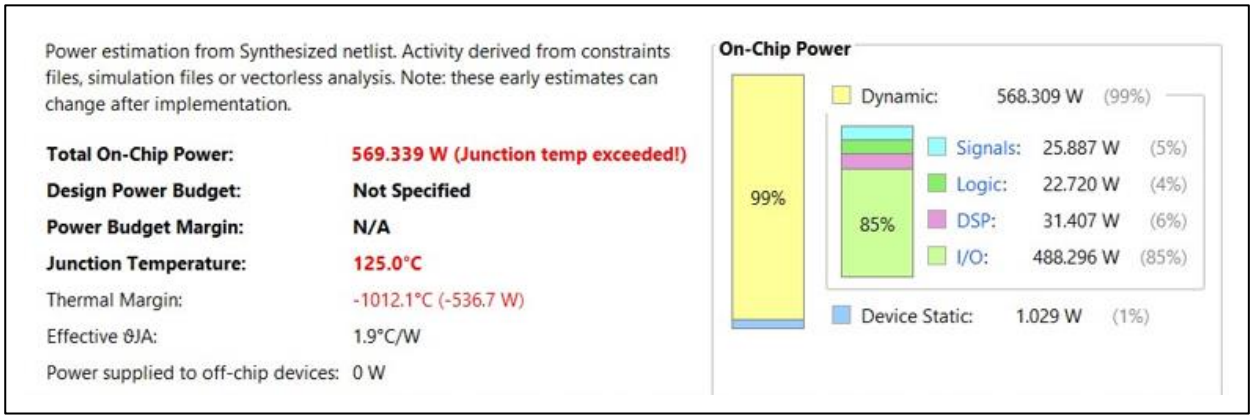


Figure 5.5.5 Power report of 8-point 64-bit FFT in Xilinx Vivado 2022.2

5.6 PROPOSED EIGHT POINT 64-BIT FFT BASED ON DA

On simulating the proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: {a0, a1, a2, a3, a4, a5, a6, a7} = {1, 2, 3, 4, 4, 3, 2, 1}, twiddle factors W20 = W40 = W41 = W80 = W81 = W82 = W83 = 1 and

- for select line w = 3, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {20, 0, 0, -8, 0, -4, 0, 0}
- for select line w = 2, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {1, 1, 1, 1, 1, 1, 1, 1}
- for select line w = 1, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {1, -1, -1, 1, -1, 1, 1, -1}
- for select line w = 0, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {0, 0, 0, 0, 0, 0, 0, 0}

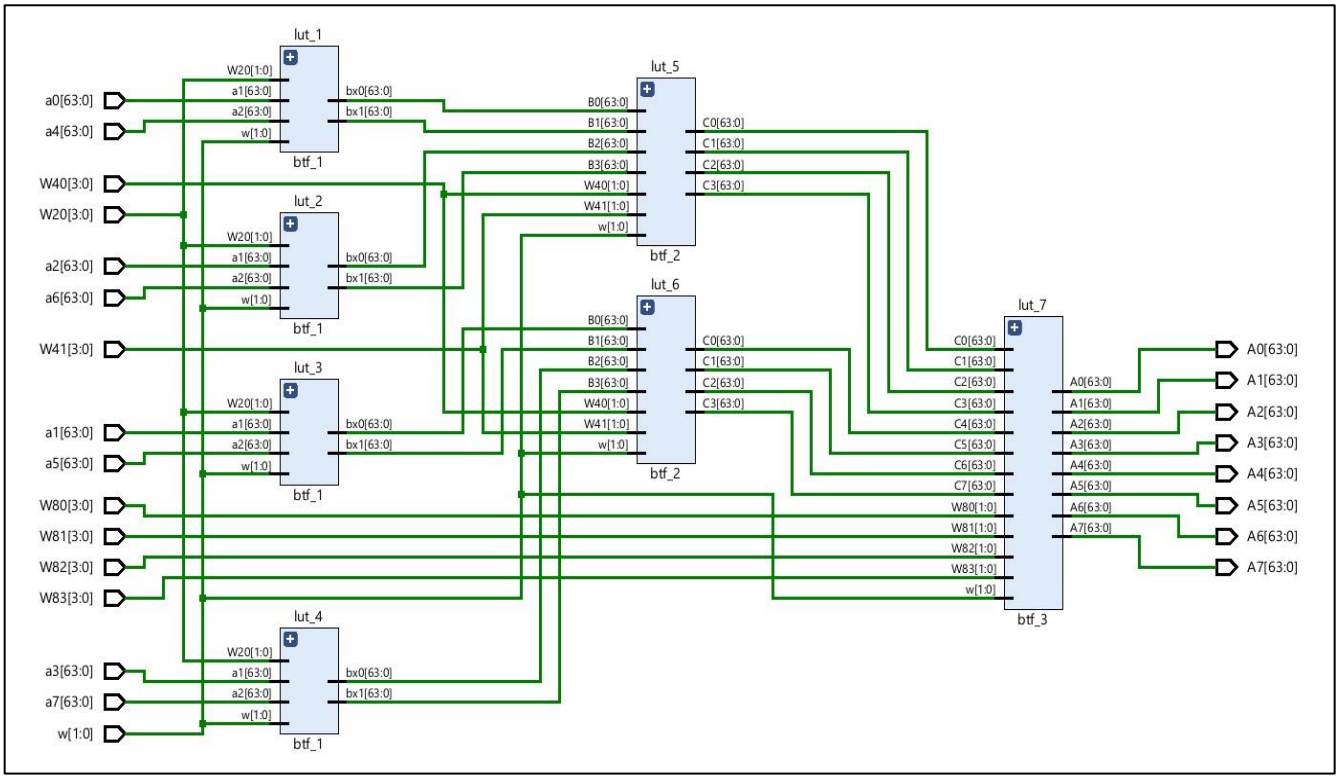
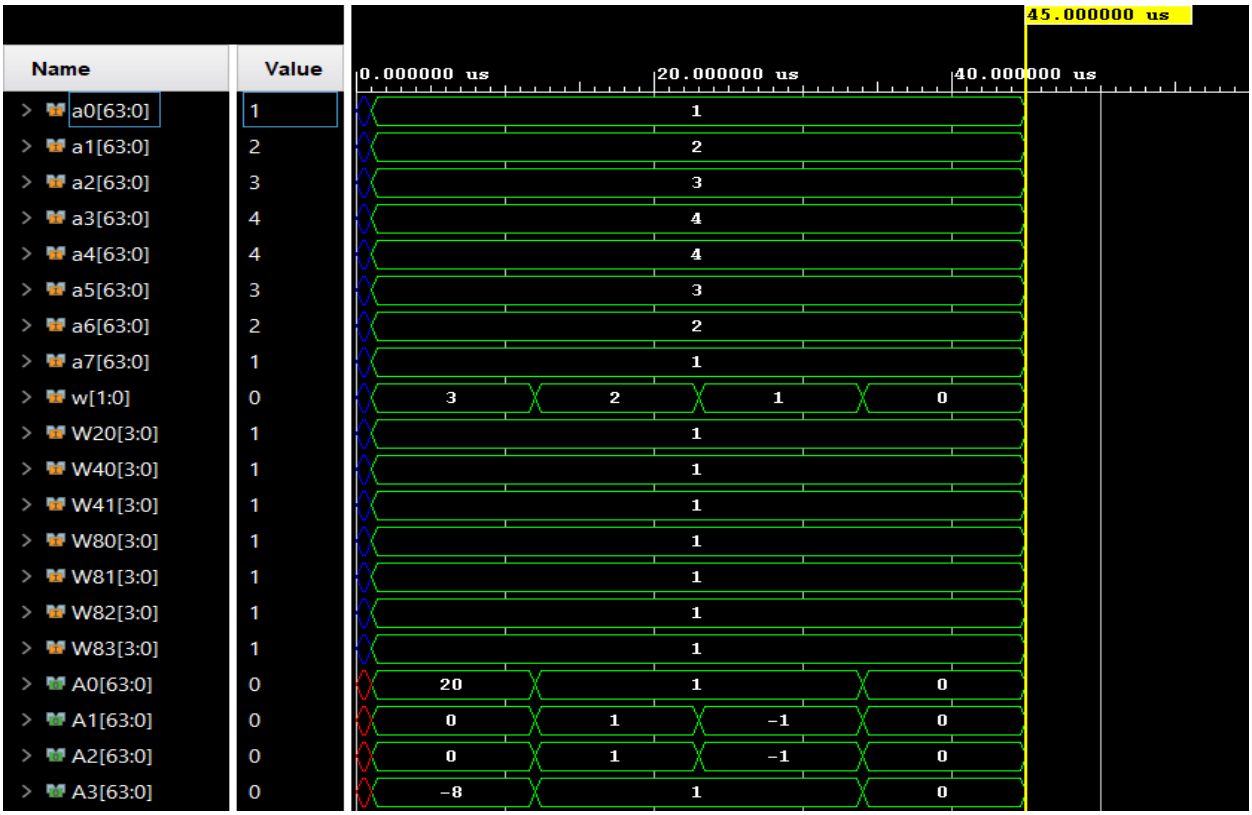


Figure 5.6.1 Schematic of proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2



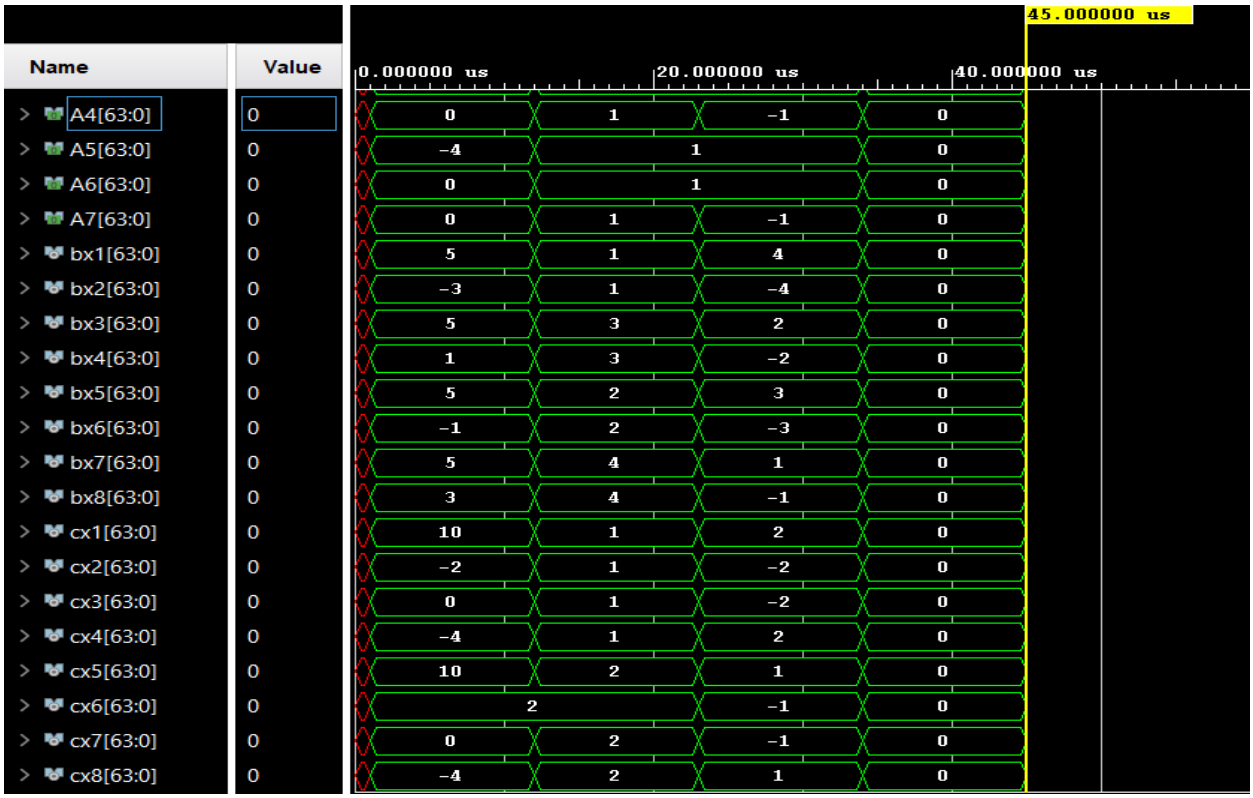


Figure 5.6.2 Simulation result of proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

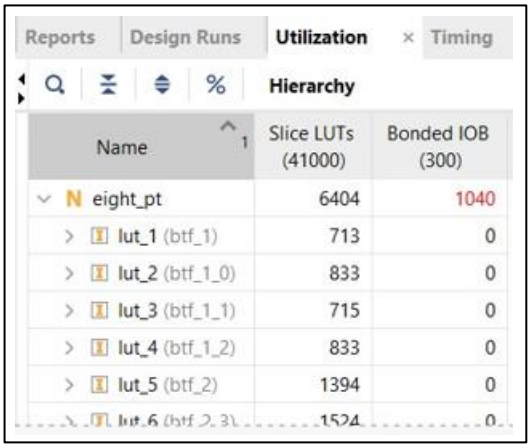


Figure 5.6.3 Utilization report of proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	38	17	4	a7[1]	A3[63]	17.817	10.801	7.016	∞	input port clock
Path 2	∞	38	17	4	a7[1]	A7[63]	17.806	10.801	7.005	∞	input port clock
Path 3	∞	39	17	4	a7[1]	A3[62]	17.769	10.967	6.802	∞	input port clock
Path 4	∞	39	17	4	a7[1]	A7[62]	17.758	10.967	6.791	∞	input port clock
Path 5	∞	38	17	4	a7[1]	A3[61]	17.731	10.939	6.792	∞	input port clock
Path 6	∞	38	18	4	a7[1]	A1[63]	17.729	10.531	7.198	∞	input port clock
Path 7	∞	38	17	4	a7[1]	A7[61]	17.720	10.939	6.781	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	3628	w[1]	A0[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	3628	w[1]	A0[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	3628	w[1]	A0[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	3628	w[1]	A0[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	3628	w[1]	A0[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	3628	w[1]	A0[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	3628	w[1]	A0[15]	1.905	1.413	0.492	-∞	input port clock

Figure 5.6.4 Delay report of proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2

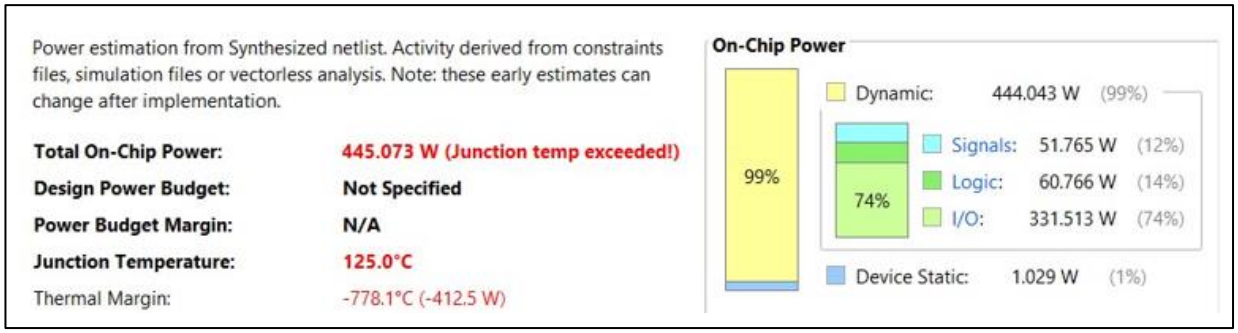


Figure 5.6.5 Power report of proposed 8-point 64-bit FFT in Xilinx Vivado 2022.2

5.7 EXISTING SIXTEEN POINT 64-BIT FFT

On simulating the 16-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1

The outputs obtained are X(n) = {24, 0, 0, 0, 0, 0, -16, 0, 0, 0, -8, 0, 0, 0, 0, 0}

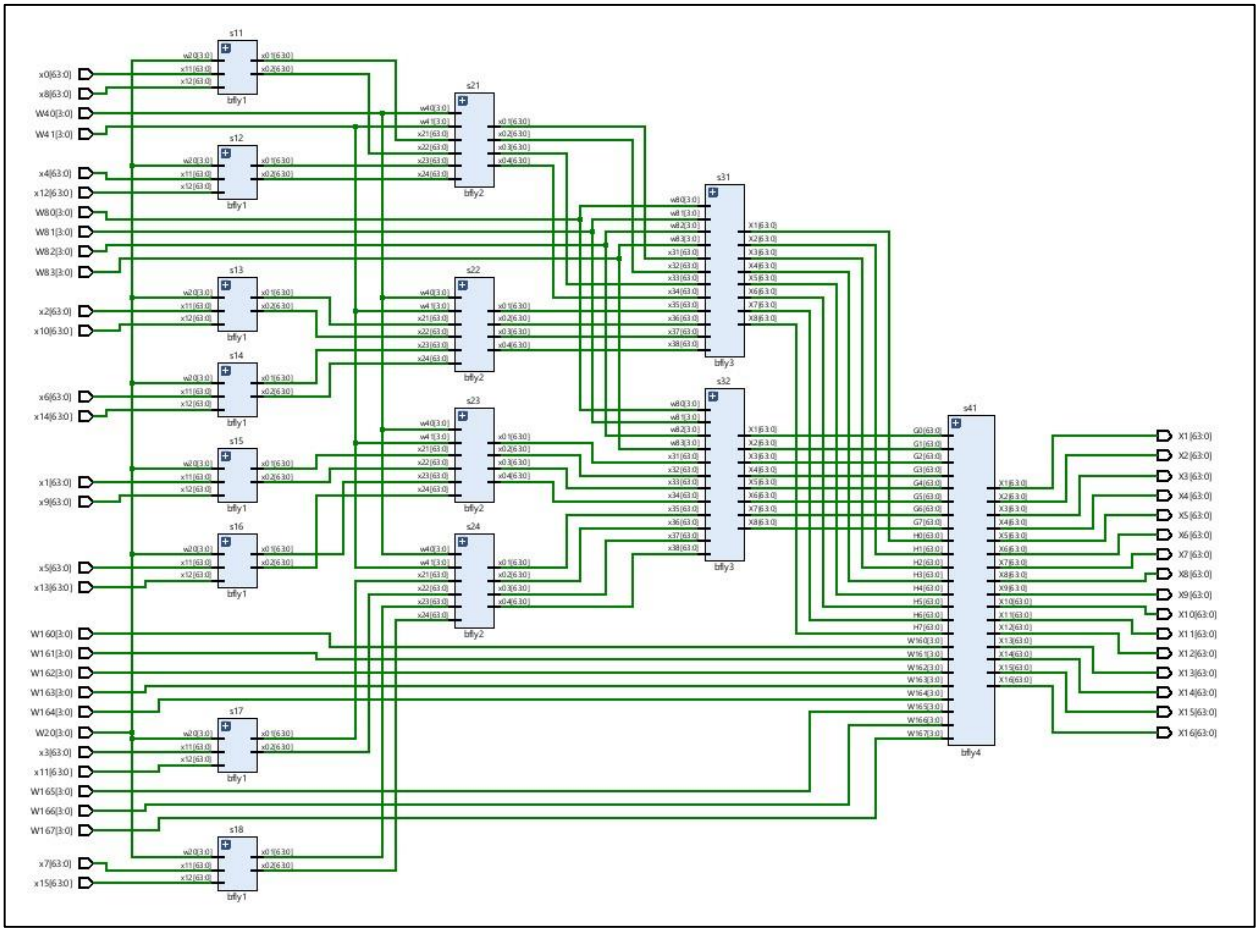
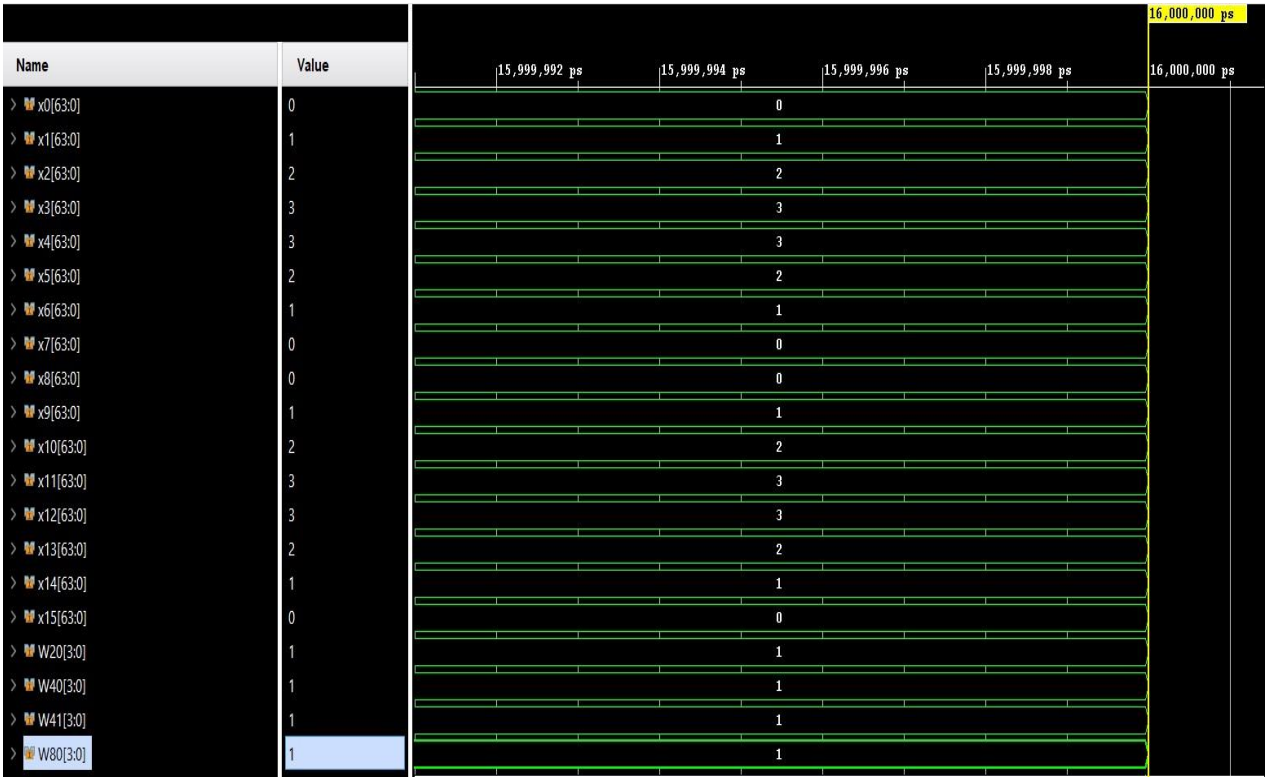


Figure 5.7.1 Schematic of 16-point 64-bit FFT in Xilinx Vivado 2022.2



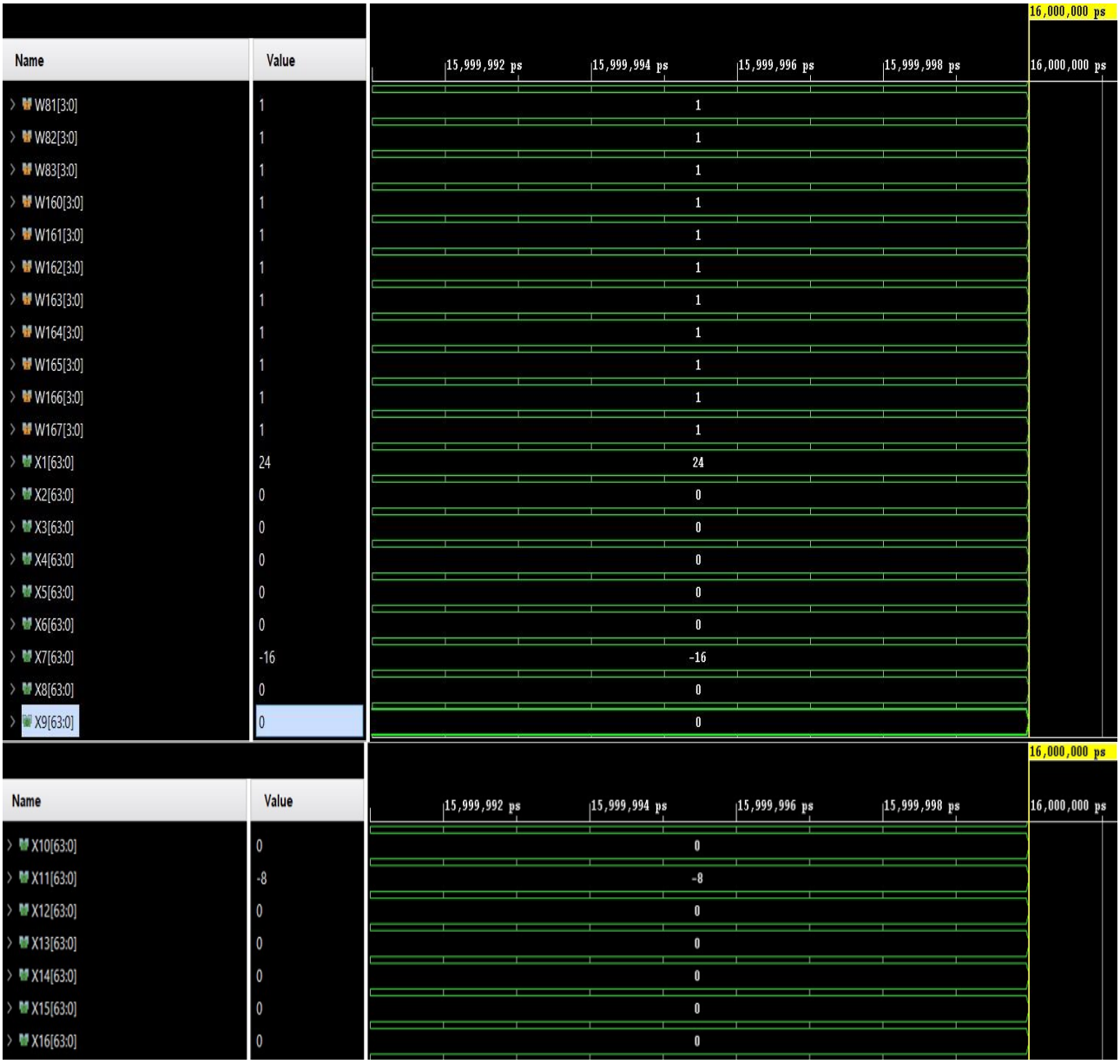


Figure 5.7.2 Simulation results of 16-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of 16-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Reports | Design Runs | Timing | Power | **Utilization**

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
fft	5056	128	2108
s11 (bfly1)	96	4	0
s12 (bfly1_0)	96	4	0
s13 (bfly1_1)	96	4	0
s14 (bfly1_2)	96	4	0
s15 (bfly1_3)	96	4	0
s16 (bfly1_4)	96	4	0

Figure 5.7.3 Utilization report of 16-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	37	17	2	x15[16]	X10[61]	31.772	24.762	7.010	∞	input port clock
Path 2	∞	37	17	2	x15[16]	X11[61]	31.772	24.762	7.010	∞	input port clock
Path 3	∞	37	17	2	x15[16]	X12[61]	31.772	24.762	7.010	∞	input port clock
Path 4	∞	37	17	2	x15[16]	X13[61]	31.772	24.762	7.010	∞	input port clock
Path 5	∞	37	17	2	x15[16]	X14[61]	31.772	24.762	7.010	∞	input port clock
Path 6	∞	37	17	2	x15[16]	X15[61]	31.772	24.762	7.010	∞	input port clock
Path 7	∞	37	17	2	x15[16]	X16[61]	31.772	24.762	7.010	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	9	7	4	x0[12]	X10[13]	2.662	1.836	0.826	-∞	input port clock
Path 12	∞	9	7	4	x0[16]	X10[17]	2.662	1.836	0.826	-∞	input port clock
Path 13	∞	9	7	4	x0[0]	X10[1]	2.662	1.836	0.826	-∞	input port clock
Path 14	∞	9	7	4	x0[20]	X10[21]	2.662	1.836	0.826	-∞	input port clock
Path 15	∞	9	7	4	x0[24]	X10[25]	2.662	1.836	0.826	-∞	input port clock
Path 16	∞	9	7	4	x0[28]	X10[29]	2.662	1.836	0.826	-∞	input port clock
Path 17	∞	9	7	4	x0[32]	X10[33]	2.662	1.836	0.826	-∞	input port clock

Figure 5.7.4 Delay report of 16-point 64-bit FFT in Xilinx Vivado 2022.2

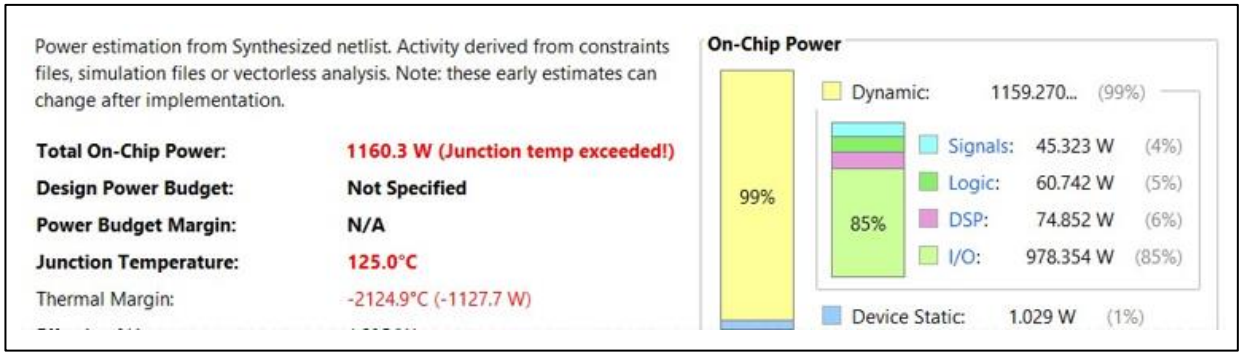


Figure 5.7.5 Power report of 16-point 64-bit FFT in Xilinx Vivado 2022.2

5.8 PROPOSED SIXTEEN POINT 64-BIT FFT BASED ON DA

On simulating the proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1 and for select line = 3 the outputs obtained are X(n) = {24, 0, 0, 0, 0, 0, -16, 0, 0, 0, -8, 0, 0, 0, 0, 0}

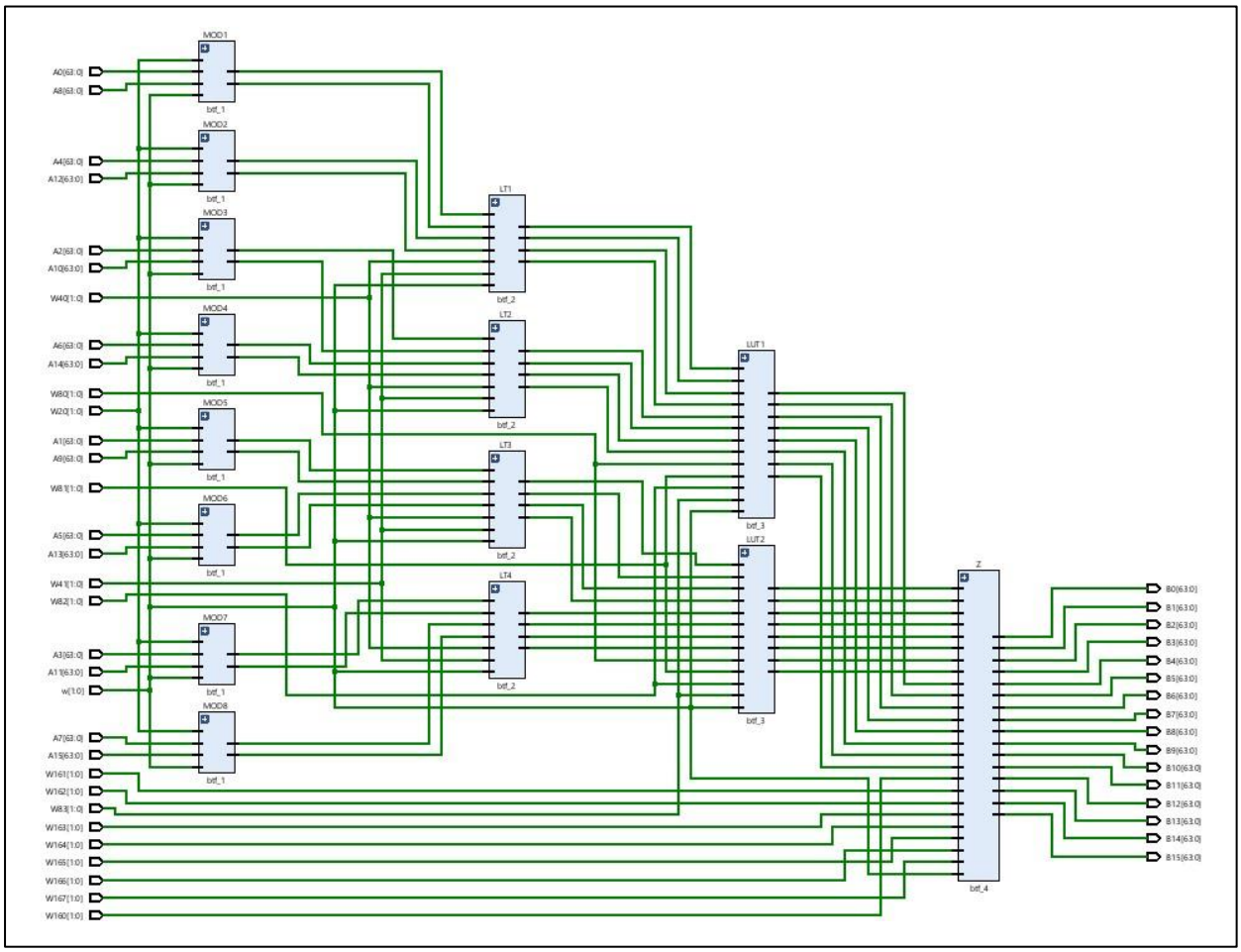
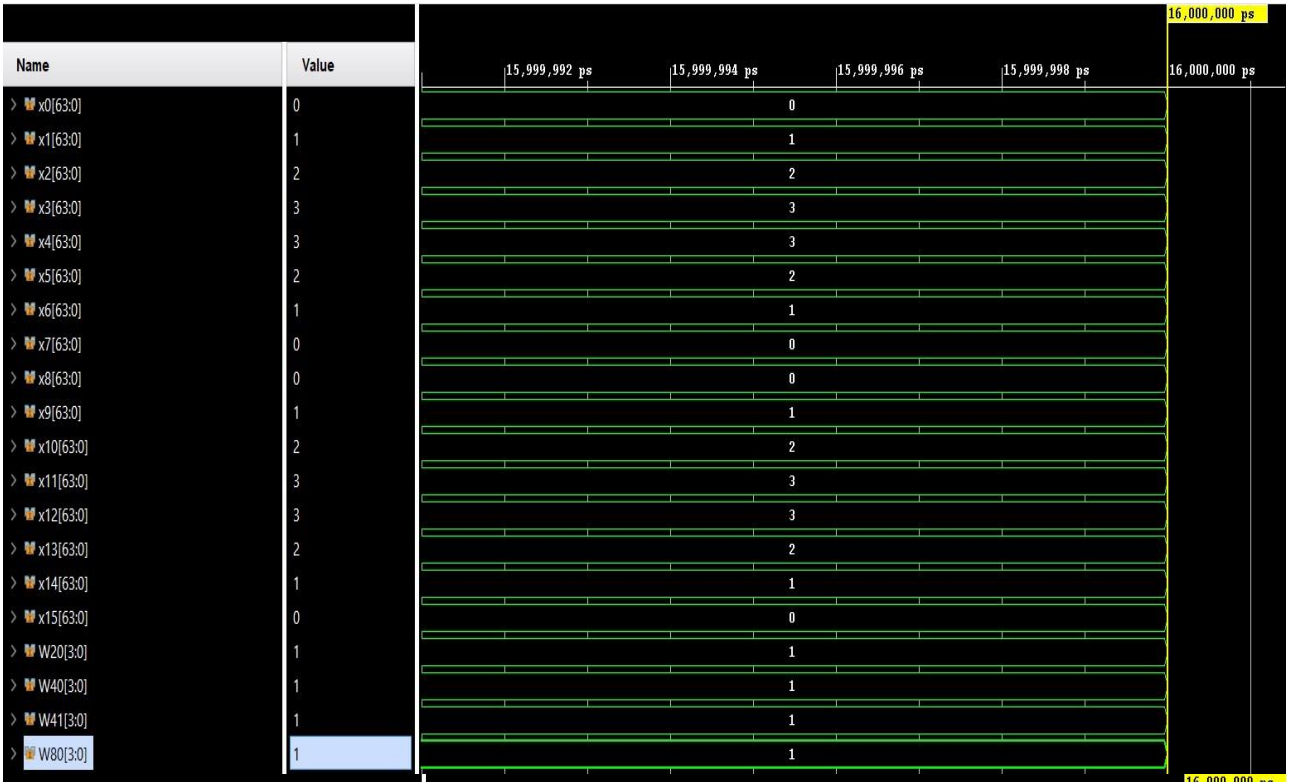
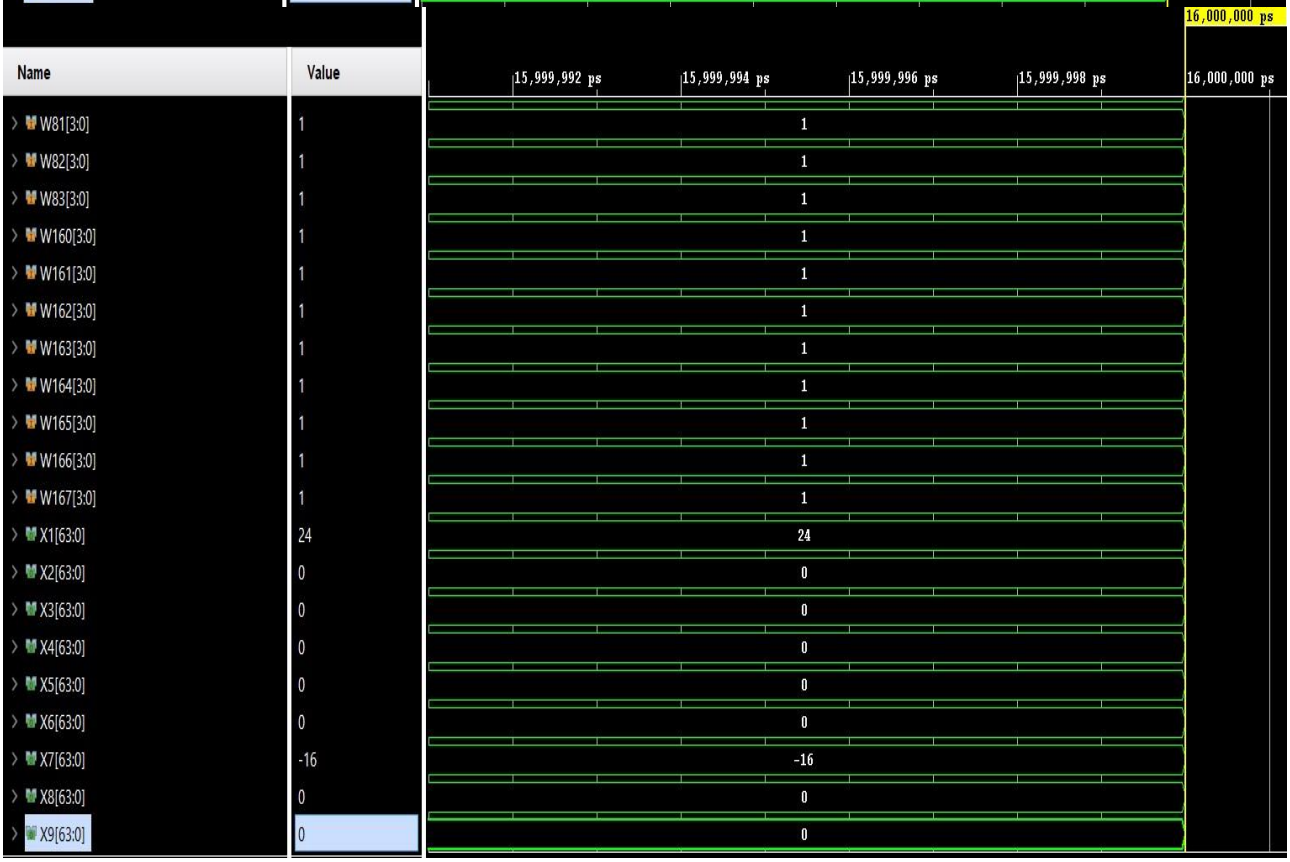


Figure 5.8.1 Schematic of proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2


Name
Value
x0[63:0] 0
x1[63:0] 1
x2[63:0] 2
x3[63:0] 3
x4[63:0] 3
x5[63:0] 2
x6[63:0] 1
x7[63:0] 0
x8[63:0] 0
x9[63:0] 1
x10[63:0] 2
x11[63:0] 3
x12[63:0] 3
x13[63:0] 2
x14[63:0] 1
x15[63:0] 0
W20[3:0] 1
W40[3:0] 1
W41[3:0] 1
W80[3:0] 1
15,999,992 ps
15,999,994 ps
15,999,996 ps
15,999,998 ps
16,000,000 ps


Name
Value
W81[3:0] 1
W82[3:0] 1
W83[3:0] 1
W160[3:0] 1
W161[3:0] 1
W162[3:0] 1
W163[3:0] 1
W164[3:0] 1
W165[3:0] 1
W166[3:0] 1
W167[3:0] 1
X1[63:0] 24
X2[63:0] 0
X3[63:0] 0
X4[63:0] 0
X5[63:0] 0
X6[63:0] 0
X7[63:0] -16
X8[63:0] 0
X9[63:0] 0
15,999,992 ps
15,999,994 ps
15,999,996 ps
15,999,998 ps
16,000,000 ps

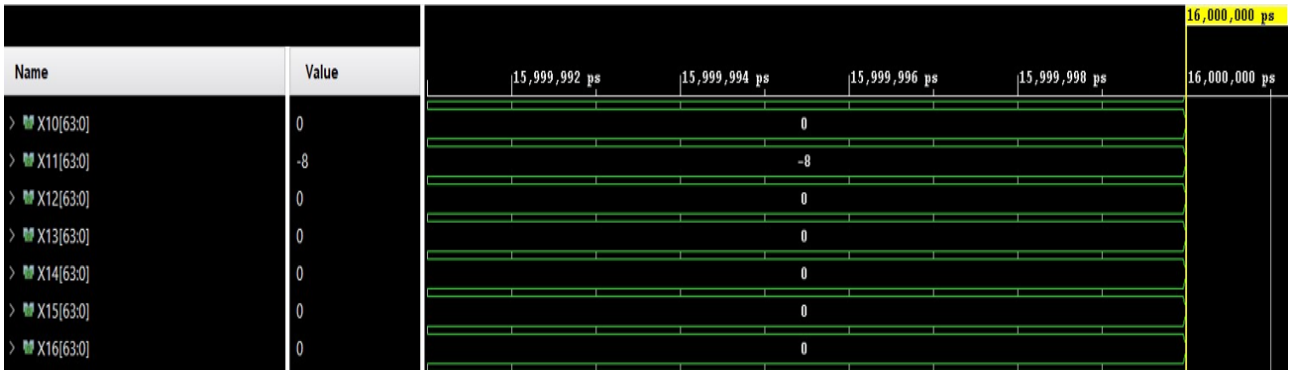


Figure 5.8.2 Simulation results of proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Reports | Design Runs | Timing | Power | **Utilization**

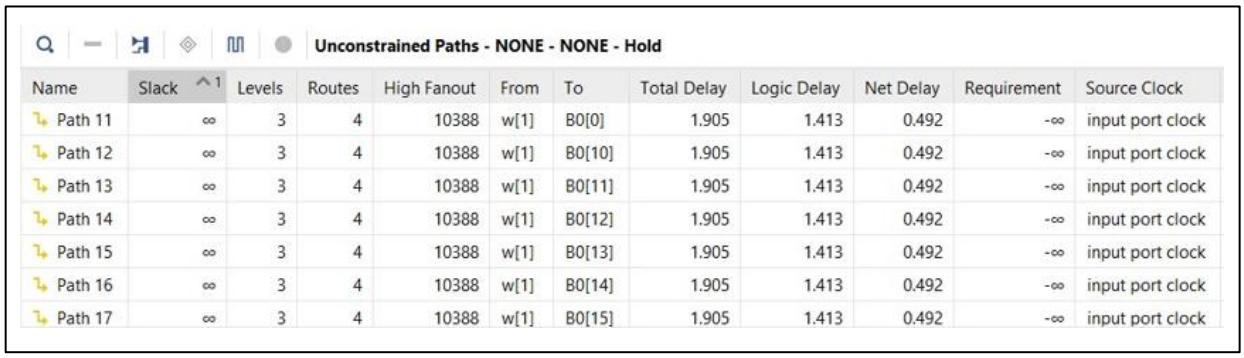
Hierarchy

Name	Slice LUTs (41000)	Bonded IOB (300)
sixteen_pt	17556	2080
LT1 (btf_2)	3286	0
LT2 (btf_2_0)	1512	0
LT3 (btf_2_1)	2030	0
LT4 (btf_2_2)	1768	0
MOD1 (btf_1)	721	0
MOD2 (btf_1_3)	833	0

Figure 5.8.3 Utilization report of proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	46	23	5	A15[1]	B3[63]	21.661	12.427	9.234	∞	input port clock
Path 2	∞	46	23	5	A15[1]	B11[63]	21.650	12.427	9.223	∞	input port clock
Path 3	∞	46	23	5	A15[1]	B7[63]	21.637	12.427	9.210	∞	input port clock
Path 4	∞	46	23	5	A15[1]	B15[63]	21.626	12.427	9.199	∞	input port clock
Path 5	∞	46	23	5	A15[1]	B3[62]	21.618	12.403	9.215	∞	input port clock
Path 6	∞	46	23	5	A15[1]	B11[62]	21.607	12.403	9.204	∞	input port clock
Path 7	∞	46	23	5	A15[1]	B7[62]	21.594	12.403	9.191	∞	input port clock



Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	10388	w[1]	B0[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	10388	w[1]	B0[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	10388	w[1]	B0[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	10388	w[1]	B0[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	10388	w[1]	B0[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	10388	w[1]	B0[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	10388	w[1]	B0[15]	1.905	1.413	0.492	-∞	input port clock

Figure 5.8.4 Delay report of proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2

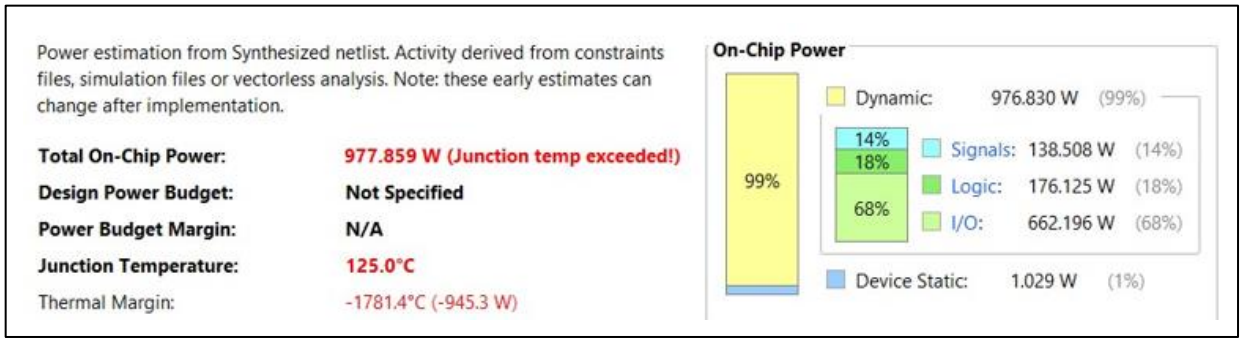


Figure 5.8.5 Power report of proposed 16-point 64-bit FFT in Xilinx Vivado 2022.2

5.9 EXISTING THIRTY-TWO POINT 64-BIT FFT

On simulating the 32-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1

The outputs obtained are X(n) = {48, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -32, 0, 0 0, 0, 0, 0, 0, -16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

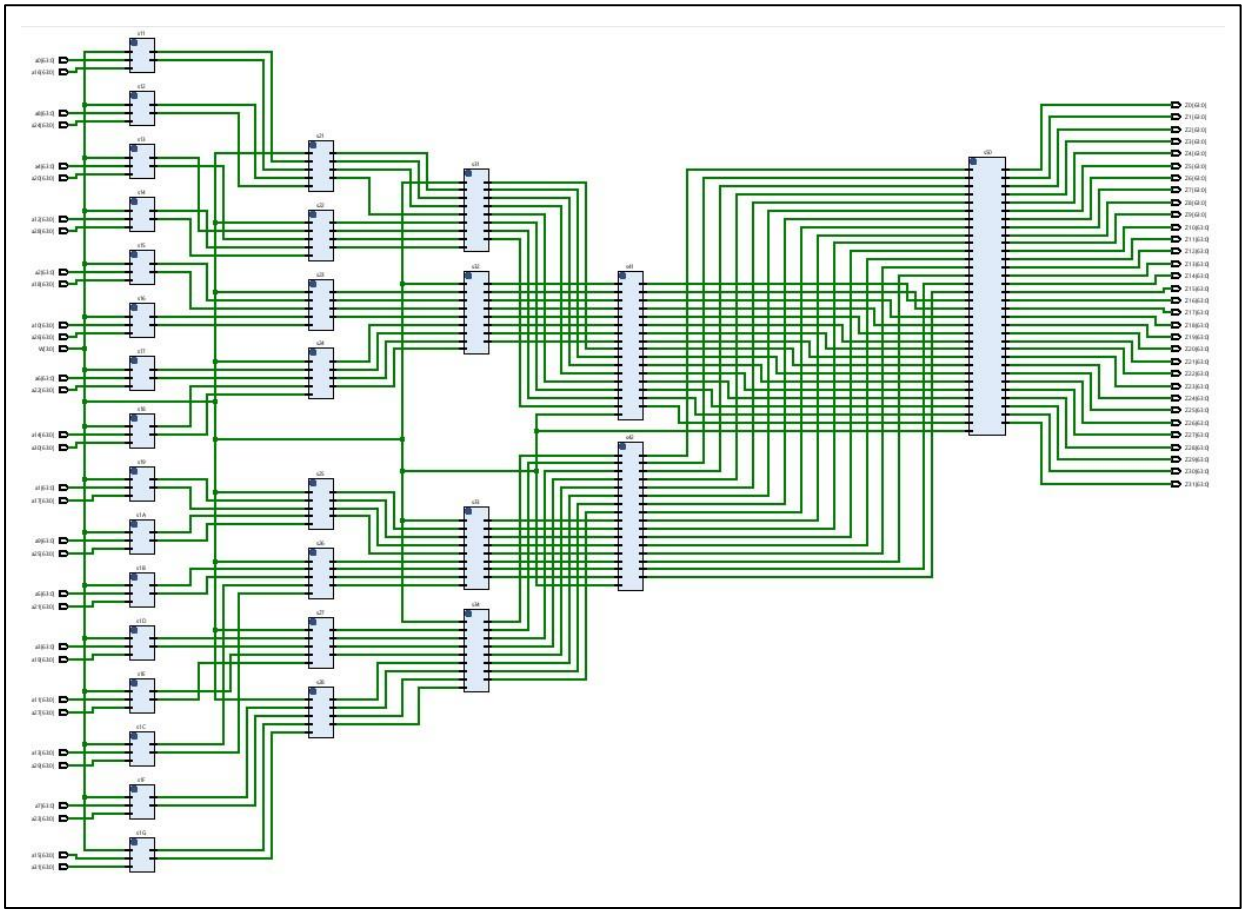
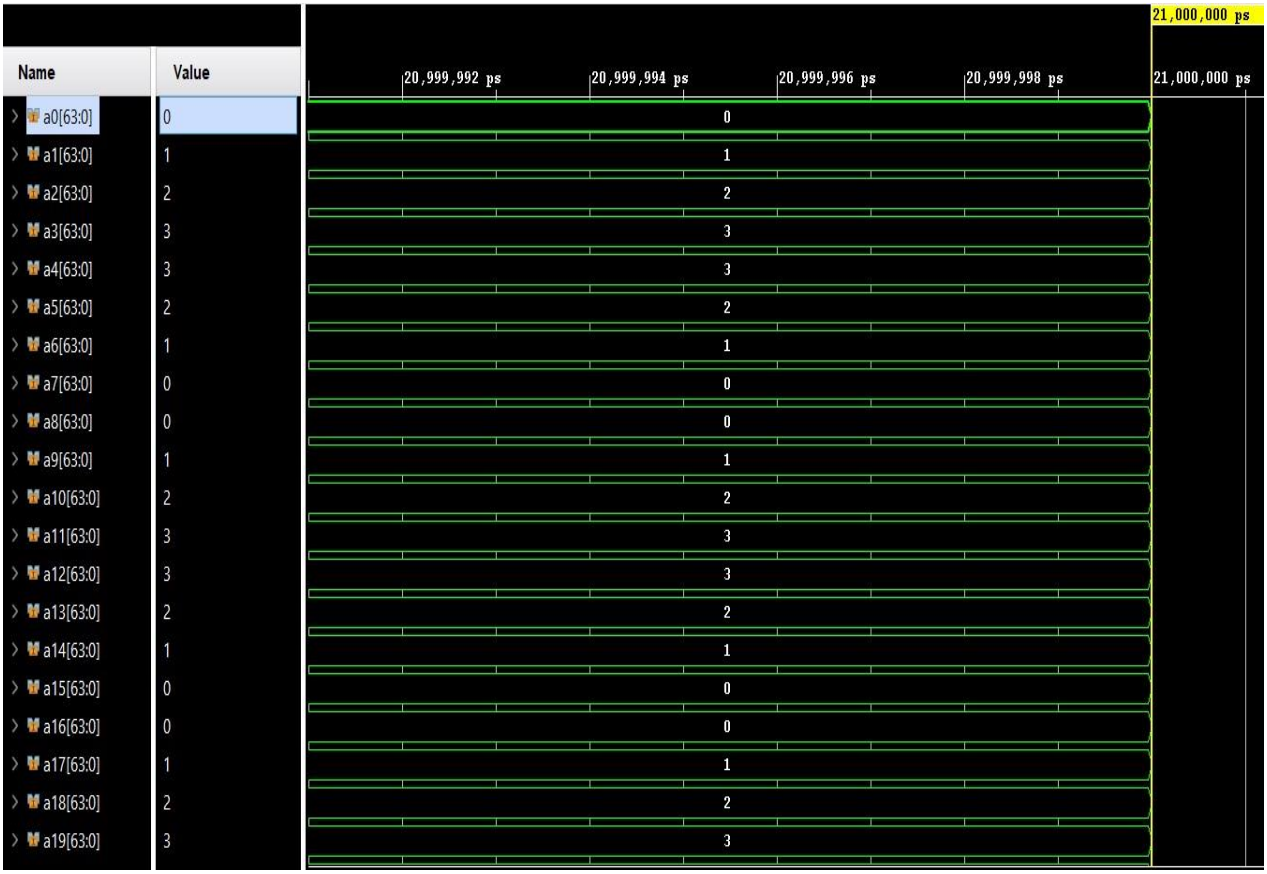
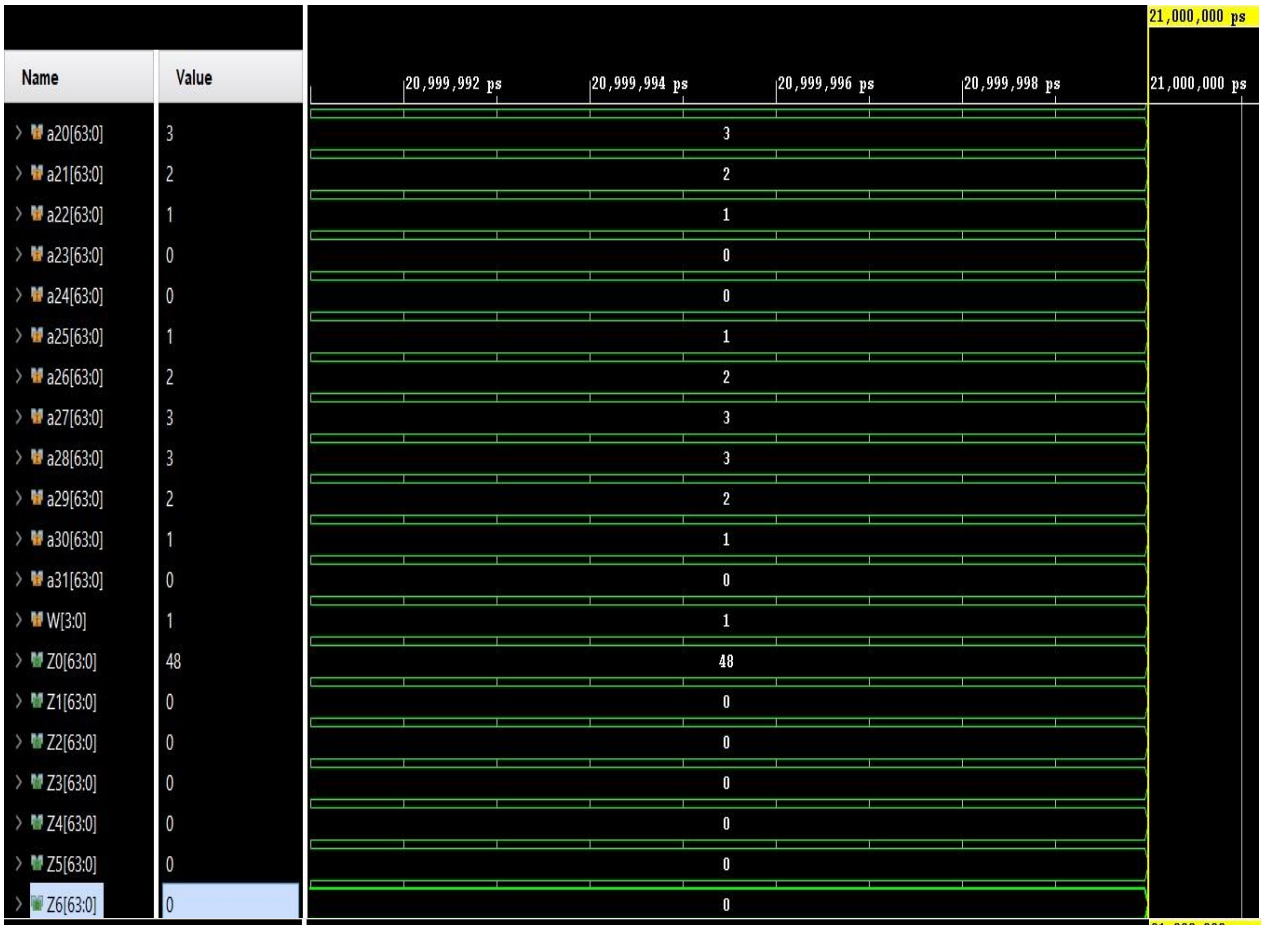
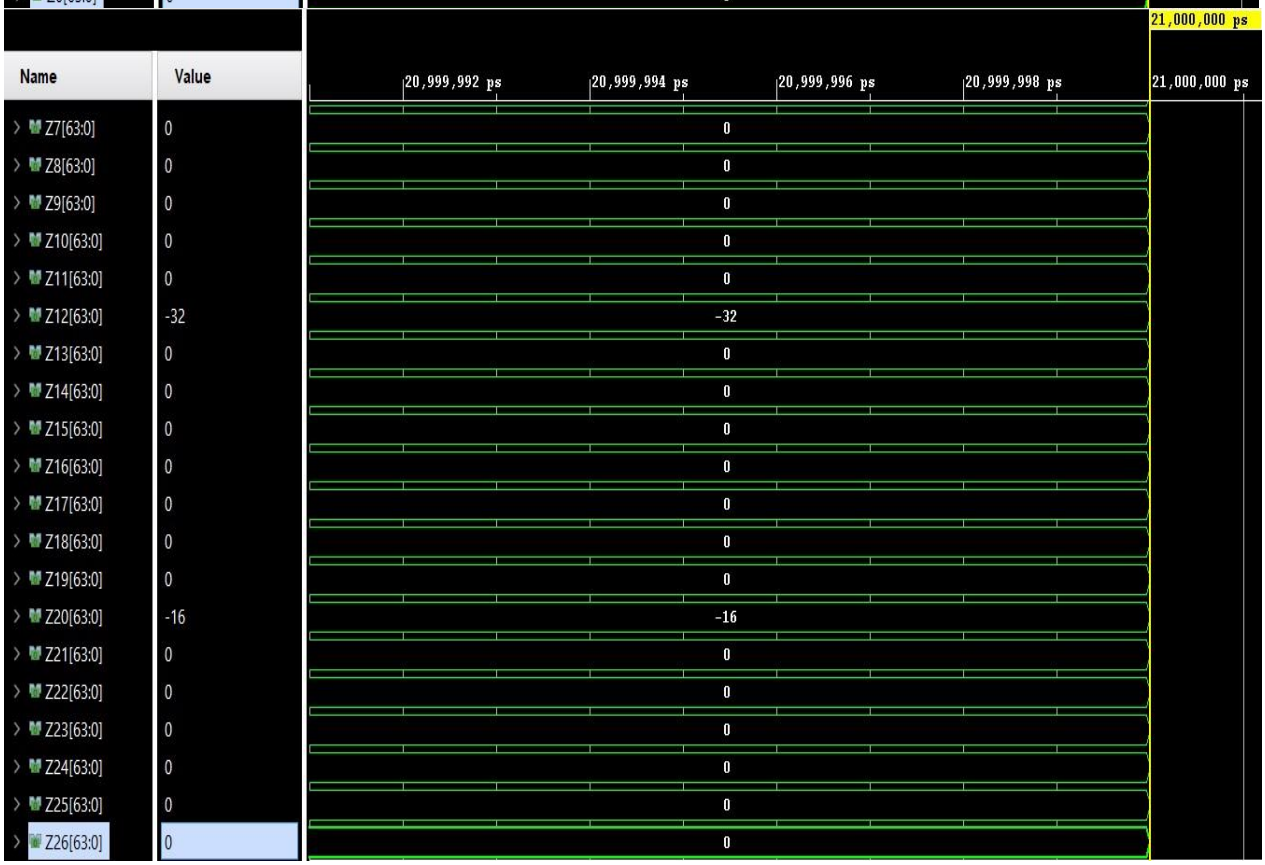


Figure 5.9.1 Schematic of 32-point 64-bit FFT in Xilinx Vivado 2022.2




Name
Value
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a20[63:0] 3
a21[63:0] 2
a22[63:0] 1
a23[63:0] 0
a24[63:0] 0
a25[63:0] 1
a26[63:0] 2
a27[63:0] 3
a28[63:0] 3
a29[63:0] 2
a30[63:0] 1
a31[63:0] 0
W[3:0] 1
Z0[63:0] 48
Z1[63:0] 0
Z2[63:0] 0
Z3[63:0] 0
Z4[63:0] 0
Z5[63:0] 0
Z6[63:0] 0


Name
Value
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Z7[63:0] 0
Z8[63:0] 0
Z9[63:0] 0
Z10[63:0] 0
Z11[63:0] 0
Z12[63:0] -32
Z13[63:0] 0
Z14[63:0] 0
Z15[63:0] 0
Z16[63:0] 0
Z17[63:0] 0
Z18[63:0] 0
Z19[63:0] 0
Z20[63:0] -16
Z21[63:0] 0
Z22[63:0] 0
Z23[63:0] 0
Z24[63:0] 0
Z25[63:0] 0
Z26[63:0] 0

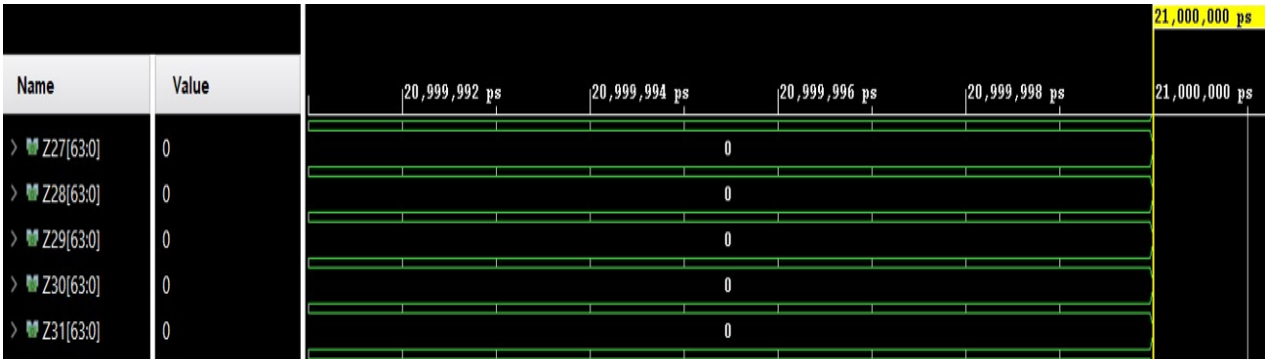


Figure 5.9.2 Simulation result of 32-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of 32-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

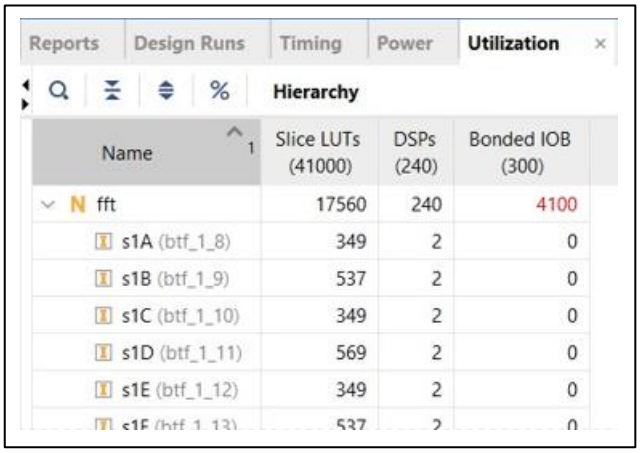


Figure 5.9.3 Utilization report of 32-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	44	19	2	a31[16]	Z0[61]	37.064	28.887	8.177	∞	input port clock
Path 2	∞	44	19	2	a31[16]	Z10[61]	37.064	28.887	8.177	∞	input port clock
Path 3	∞	44	19	2	a31[16]	Z11[61]	37.064	28.887	8.177	∞	input port clock
Path 4	∞	44	19	2	a31[16]	Z12[61]	37.064	28.887	8.177	∞	input port clock
Path 5	∞	44	19	2	a31[16]	Z13[61]	37.064	28.887	8.177	∞	input port clock
Path 6	∞	44	19	2	a31[16]	Z14[61]	37.064	28.887	8.177	∞	input port clock
Path 7	∞	44	19	2	a31[16]	Z15[61]	37.064	28.887	8.177	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	5	5	1640	W[3]	Z0[13]	2.722	1.984	0.738	-∞	input port clock
Path 12	∞	5	5	1640	W[3]	Z0[17]	2.722	1.984	0.738	-∞	input port clock
Path 13	∞	5	5	1640	W[3]	Z0[1]	2.722	1.984	0.738	-∞	input port clock
Path 14	∞	5	5	1640	W[3]	Z0[21]	2.722	1.984	0.738	-∞	input port clock
Path 15	∞	5	5	1640	W[3]	Z0[25]	2.722	1.984	0.738	-∞	input port clock
Path 16	∞	5	5	1640	W[3]	Z0[29]	2.722	1.984	0.738	-∞	input port clock
Path 17	∞	5	5	1640	W[3]	Z0[5]	2.722	1.984	0.738	-∞	input port clock

Figure 5.9.4 Delay report of 32-point 64-bit FFT in Xilinx Vivado 2022.2

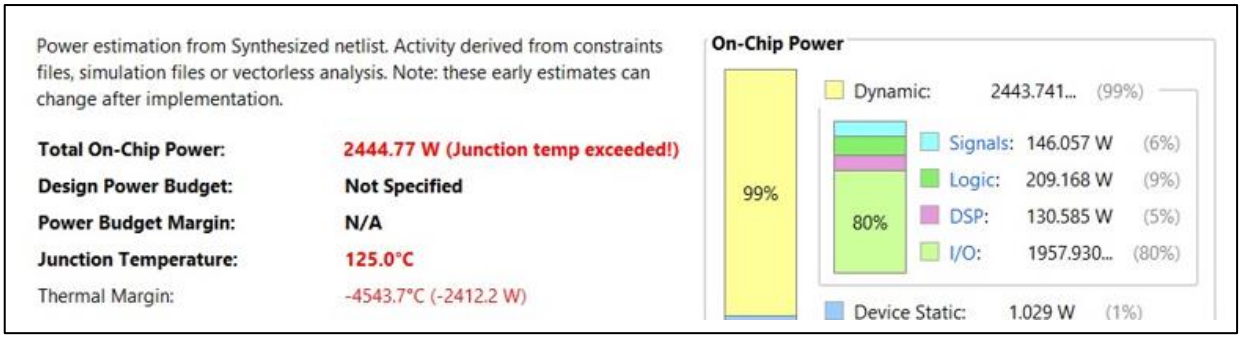


Figure 5.9.5 Power report of 32-point 64-bit FFT in Xilinx Vivado 2022.2

5.10 PROPOSED THIRTY-TWO POINT 64-BIT FFT BASED ON DA

On simulating the proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1 and for select line = 3 the outputs obtained are X(n) = {48, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -32, 0, 0 0, 0, 0, 0, 0, -16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

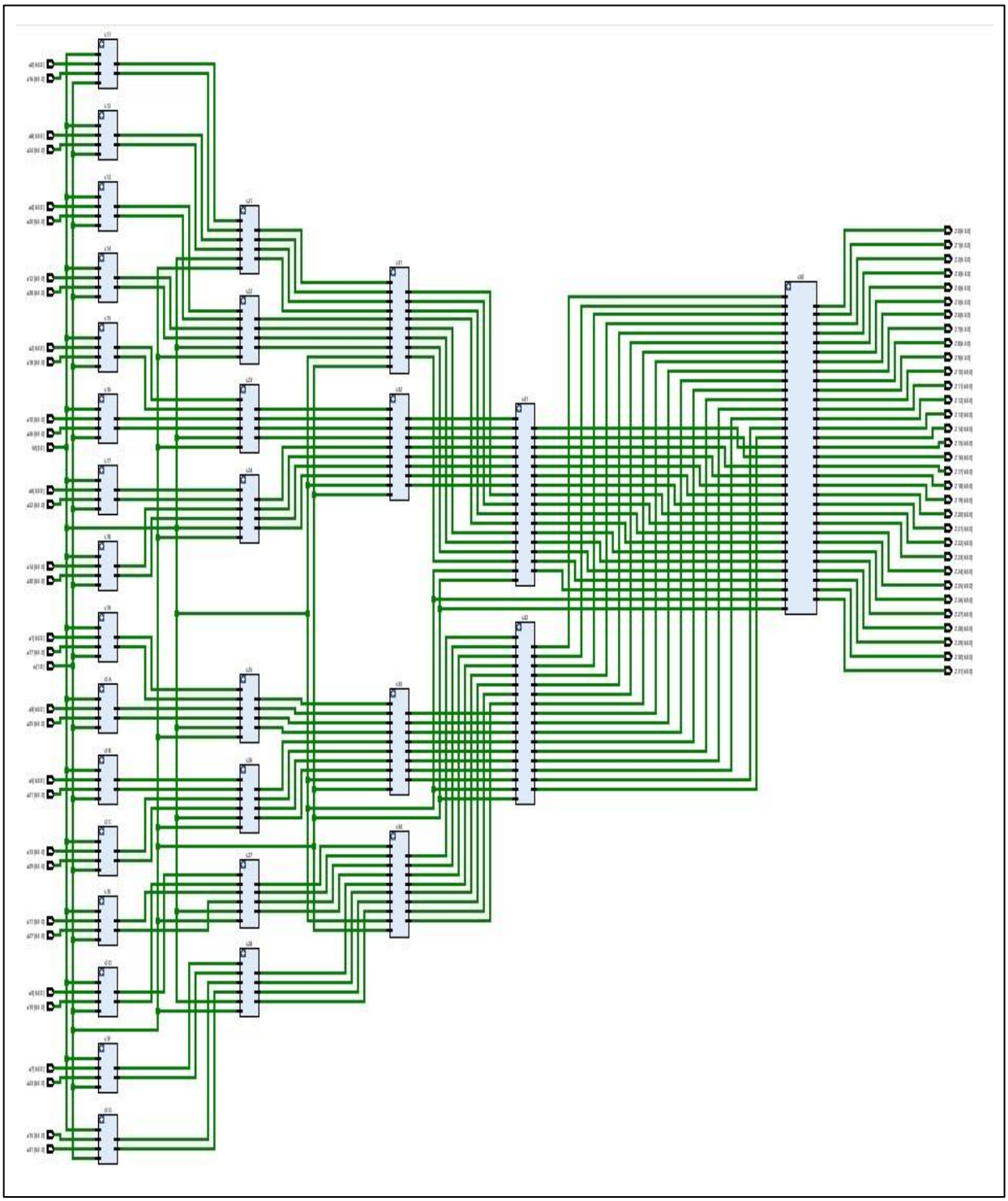
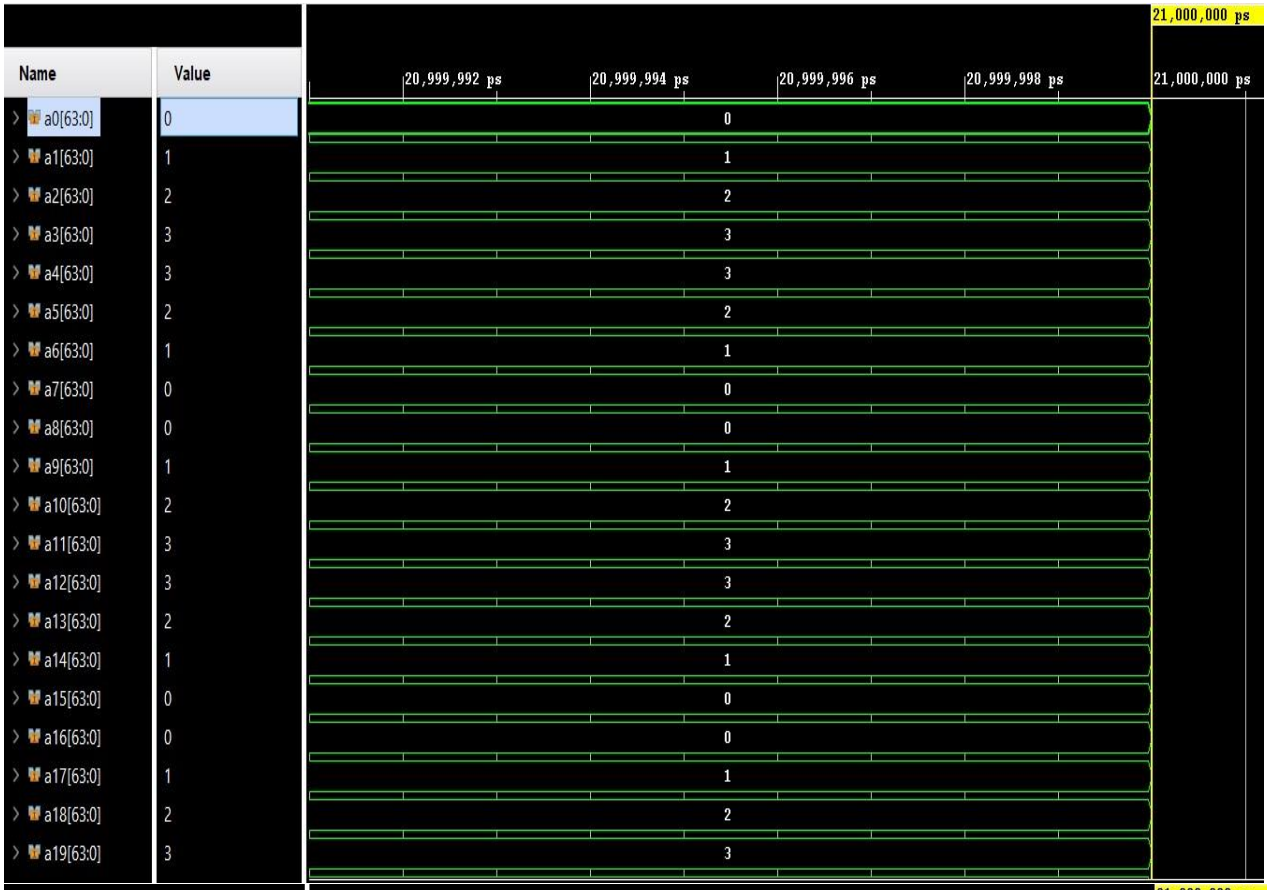


Figure 5.10.1 Schematic of proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2


Name
Value
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a0[63:0] 0
a1[63:0] 1
a2[63:0] 2
a3[63:0] 3
a4[63:0] 3
a5[63:0] 2
a6[63:0] 1
a7[63:0] 0
a8[63:0] 0
a9[63:0] 1
a10[63:0] 2
a11[63:0] 3
a12[63:0] 3
a13[63:0] 2
a14[63:0] 1
a15[63:0] 0
a16[63:0] 0
a17[63:0] 1
a18[63:0] 2
a19[63:0] 3

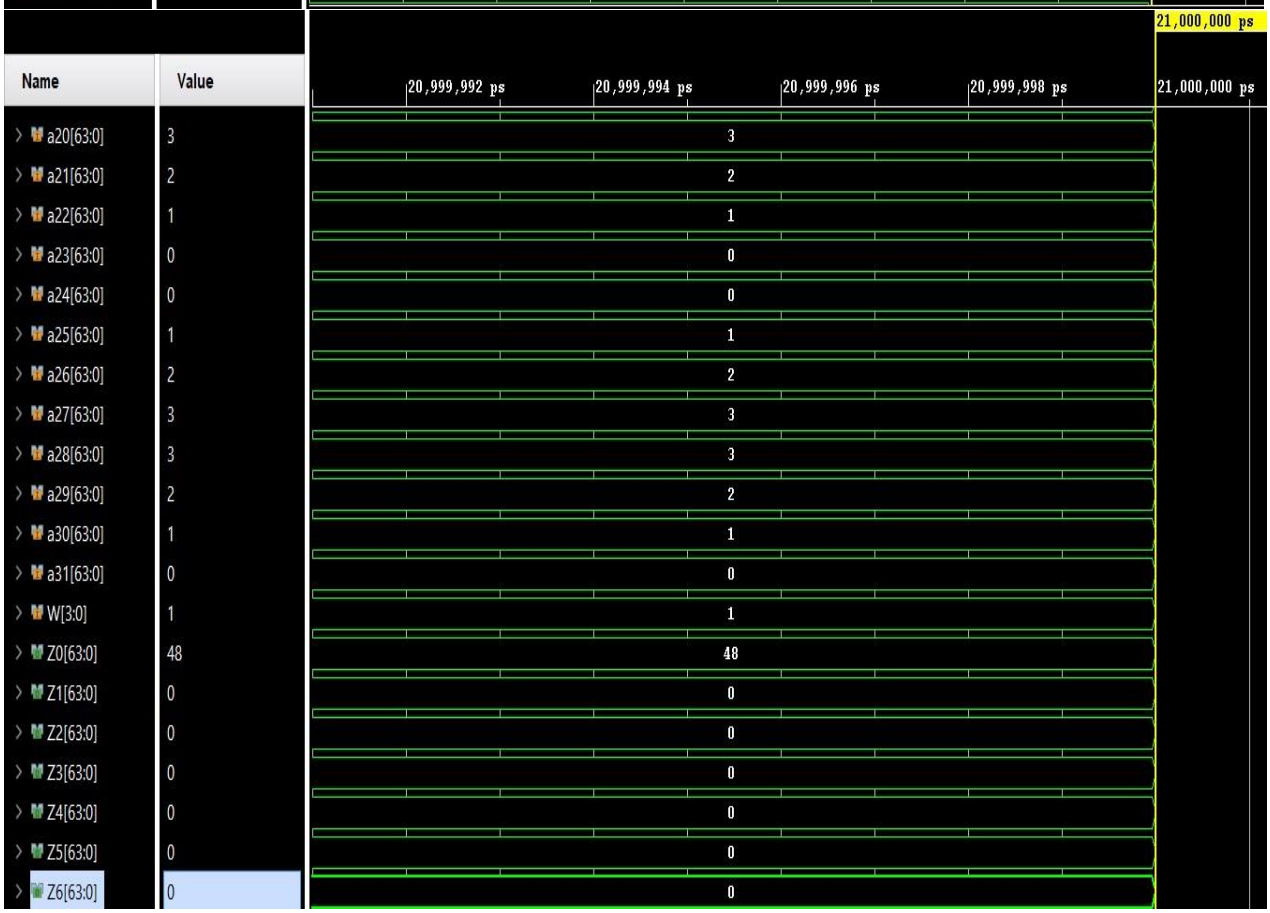

Name
Value
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a20[63:0] 3
a21[63:0] 2
a22[63:0] 1
a23[63:0] 0
a24[63:0] 0
a25[63:0] 1
a26[63:0] 2
a27[63:0] 3
a28[63:0] 3
a29[63:0] 2
a30[63:0] 1
a31[63:0] 0
W[3:0] 1
Z0[63:0] 48
Z1[63:0] 0
Z2[63:0] 0
Z3[63:0] 0
Z4[63:0] 0
Z5[63:0] 0
Z6[63:0] 0

Figure 5.10.2 Simulation results of proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

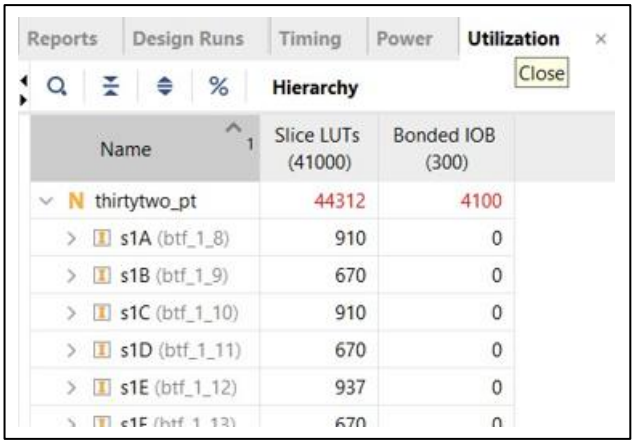


Figure 5.10.3 Utilization report of proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	55	28	7	a31[1]	Z31[63]	25.957	14.023	11.934	∞	input port clock
Path 2	∞	56	28	7	a31[1]	Z31[61]	25.826	14.123	11.703	∞	input port clock
Path 3	∞	55	28	7	a31[1]	Z15[63]	25.817	14.267	11.550	∞	input port clock
Path 4	∞	56	28	7	a31[1]	Z31[62]	25.773	13.937	11.836	∞	input port clock
Path 5	∞	55	28	7	a31[1]	Z23[63]	25.766	14.198	11.568	∞	input port clock
Path 6	∞	56	28	7	a31[1]	Z15[62]	25.758	14.254	11.504	∞	input port clock
Path 7	∞	55	28	7	a31[1]	Z29[63]	25.756	14.023	11.733	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	14536	w[1]	Z0[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	14536	w[1]	Z0[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	14536	w[1]	Z0[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	14536	w[1]	Z0[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	14536	w[1]	Z0[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	14536	w[1]	Z0[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	14536	w[1]	Z0[15]	1.905	1.413	0.492	-∞	input port clock

Figure 5.10.4 Delay report of proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2

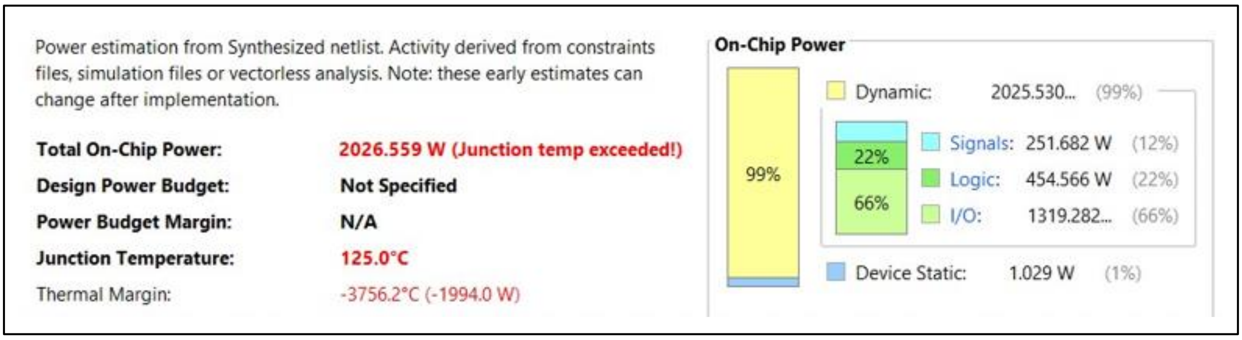


Figure 5.10.5 Power report of proposed 32-point 64-bit FFT in Xilinx Vivado 2022.2

5.11 EXISTING SIXTY-FOUR POINT 64-BIT FFT

On simulating the 64-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1

The outputs obtained are X(n) = {96, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

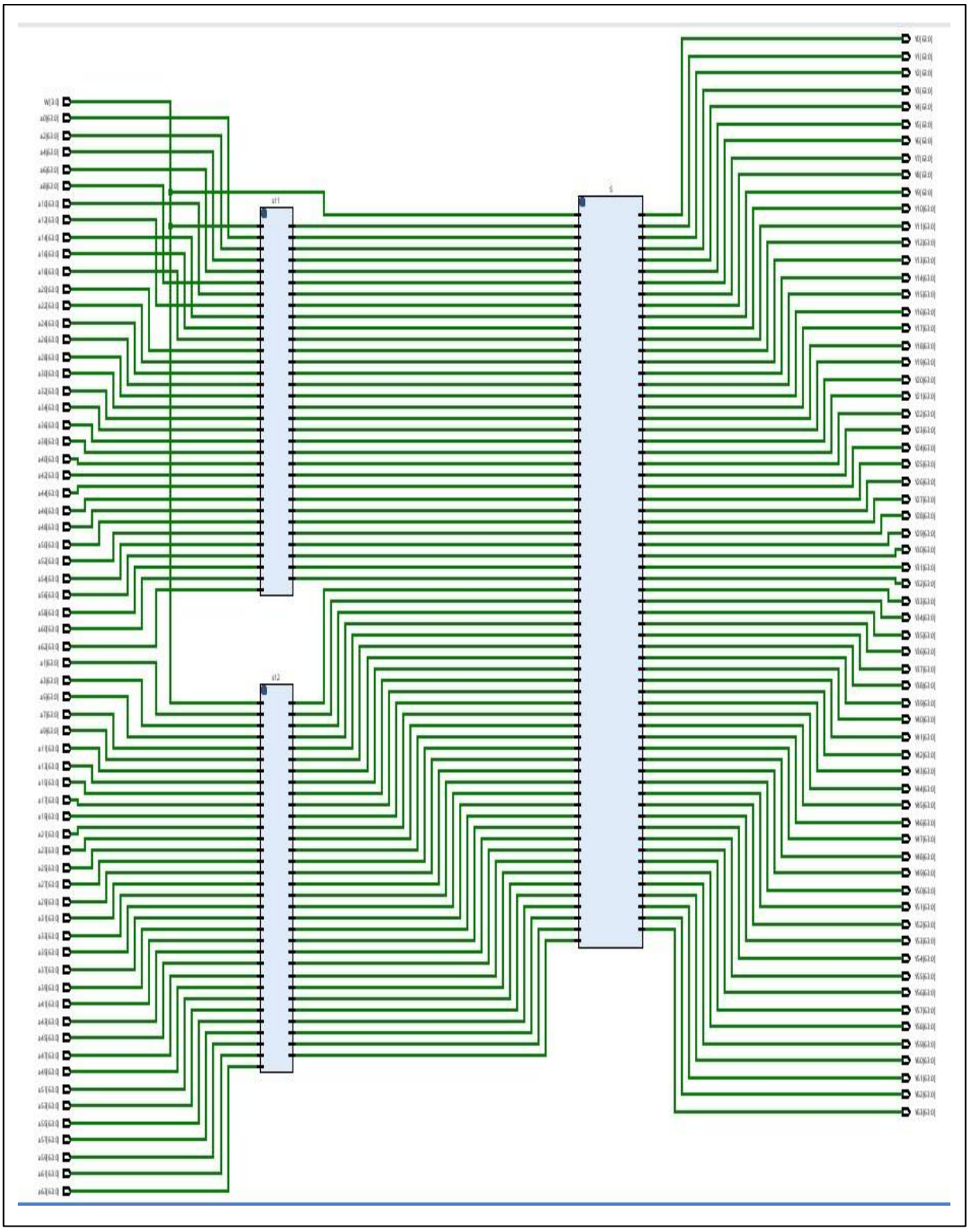
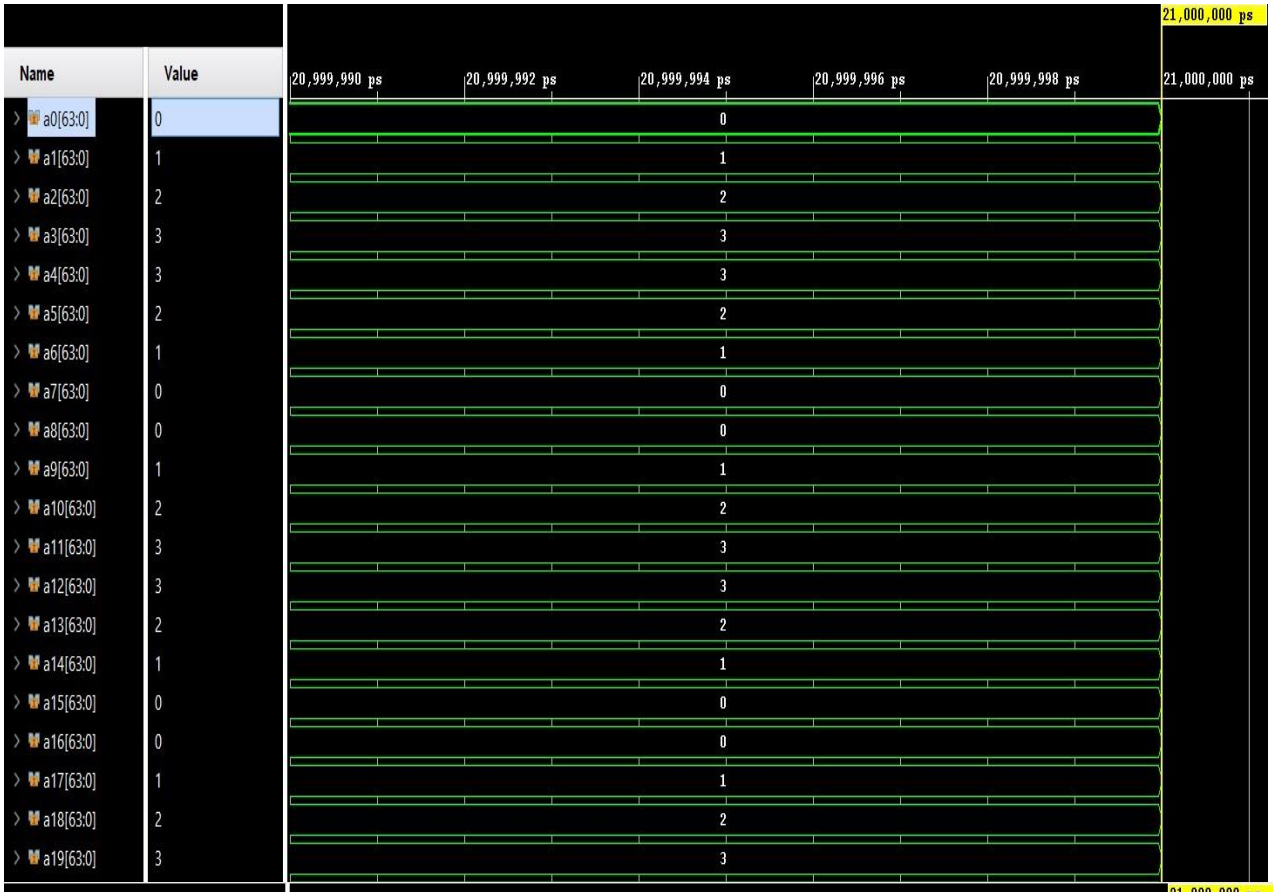
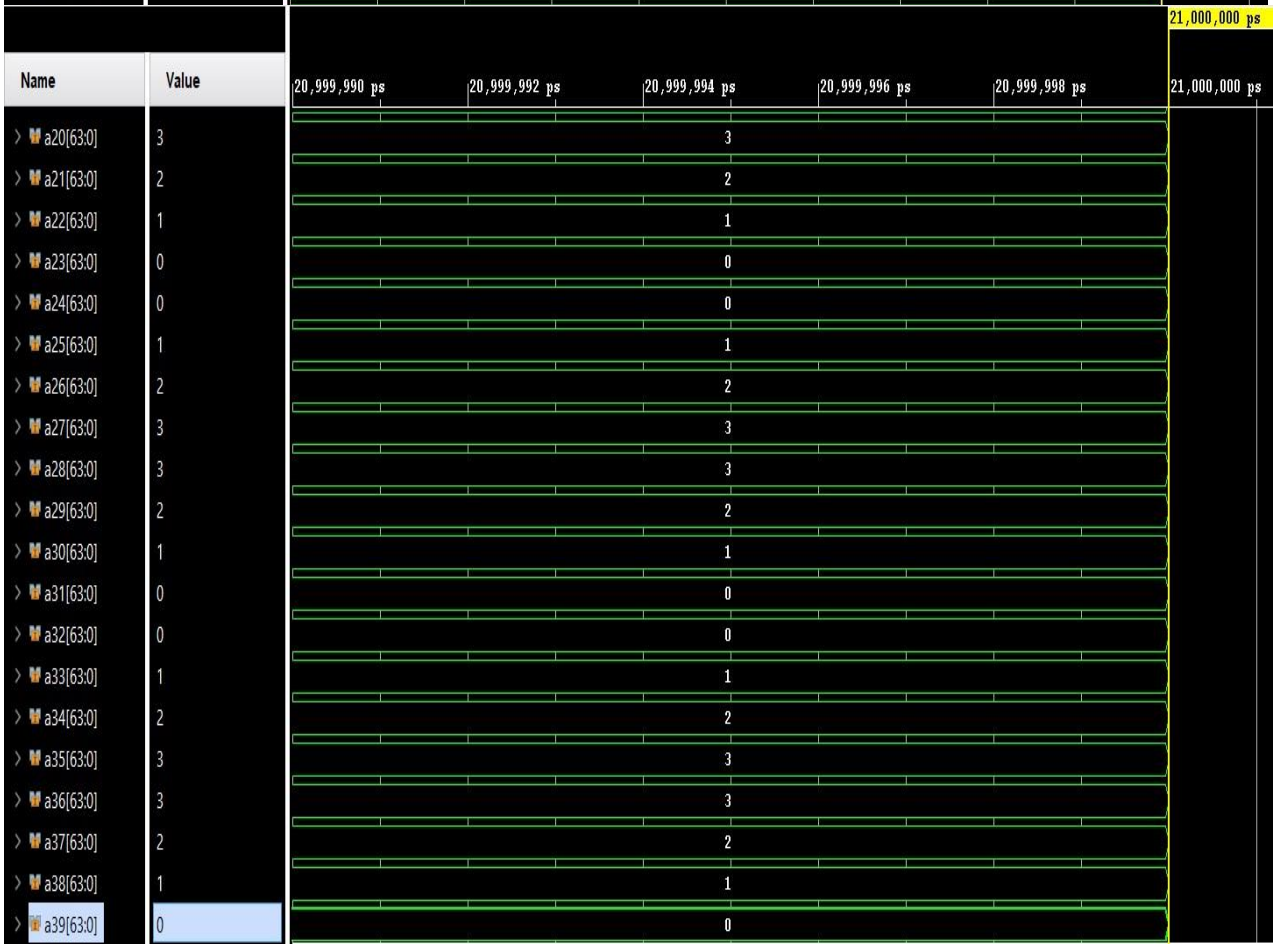
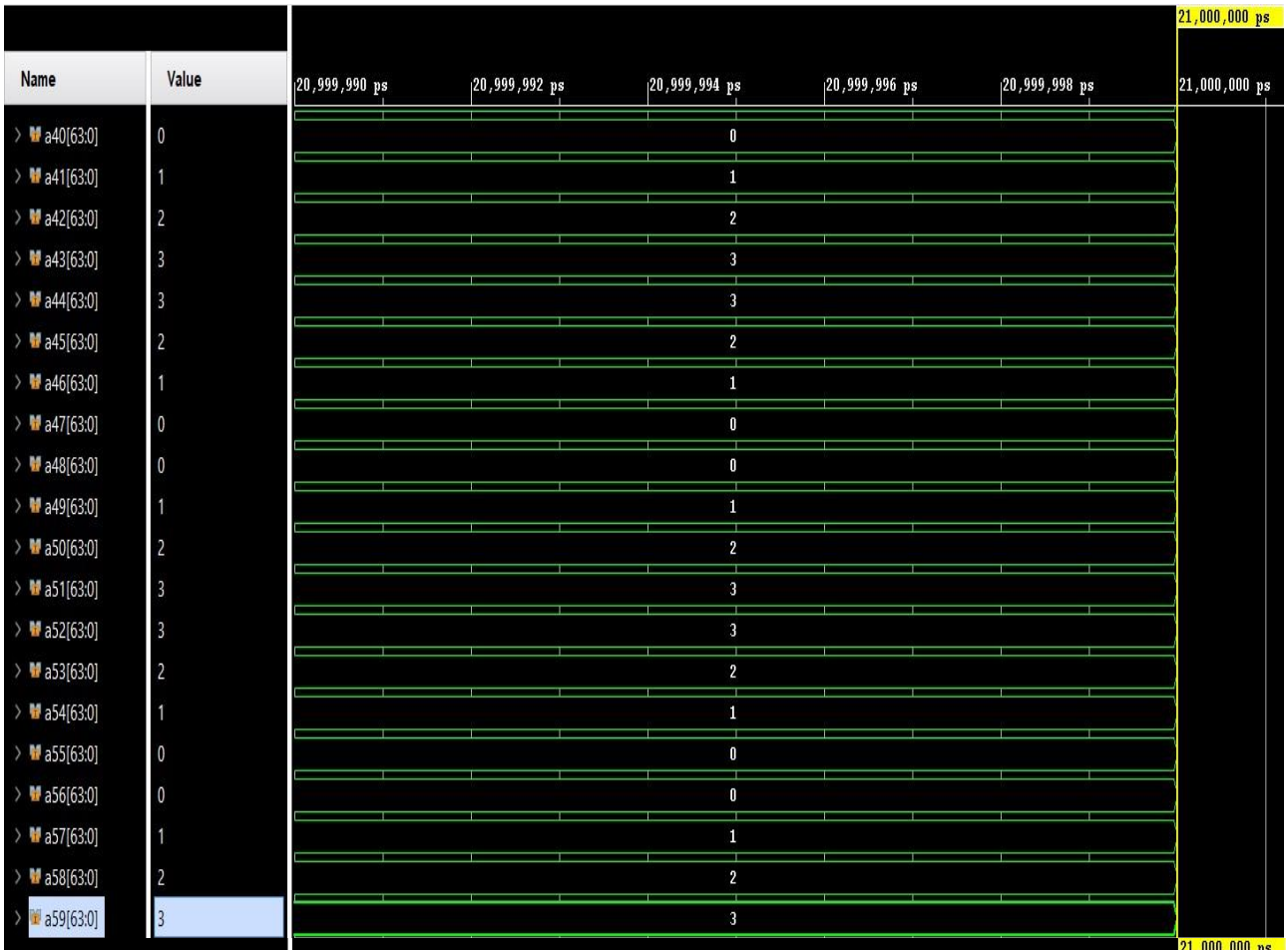
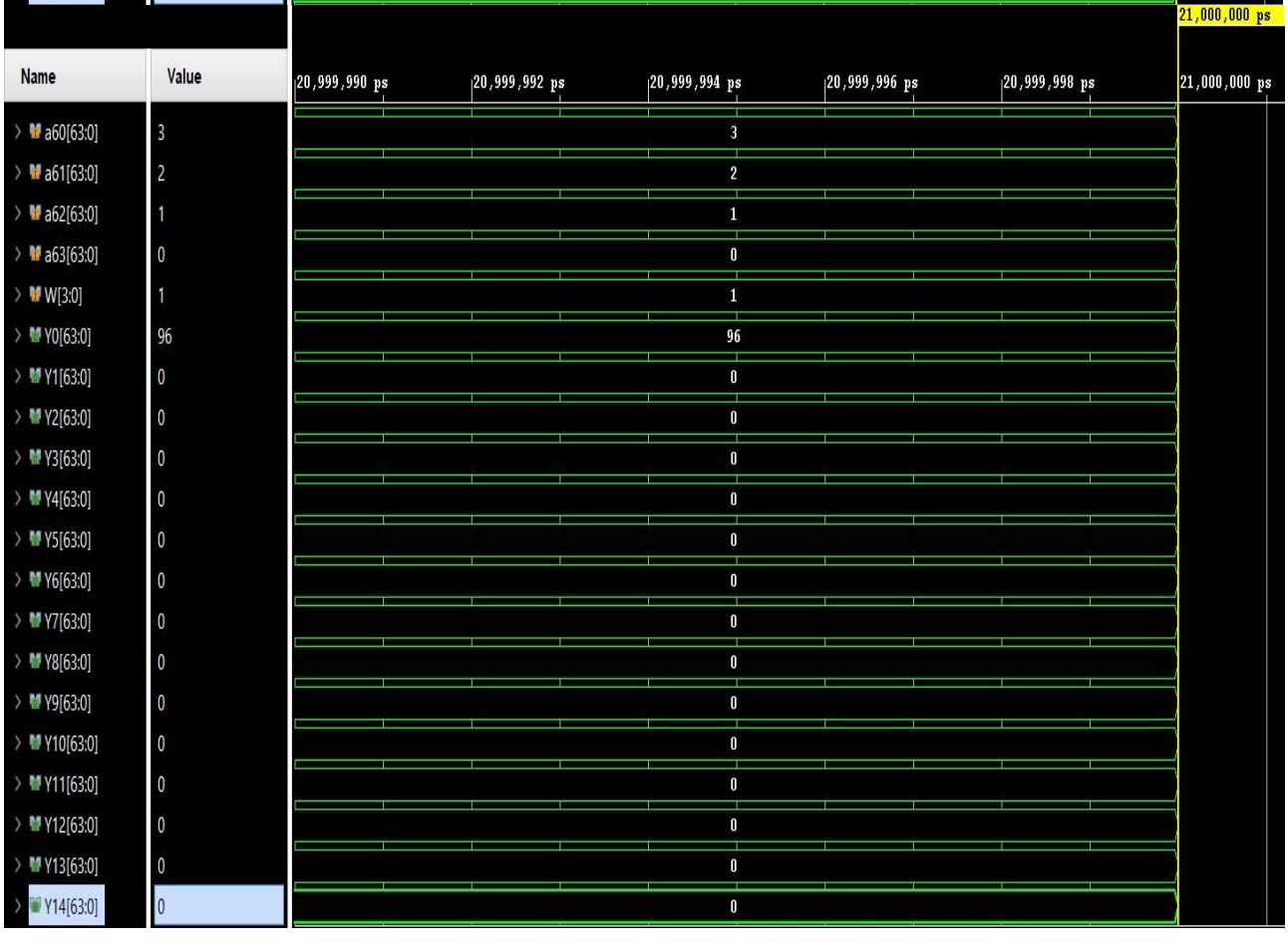


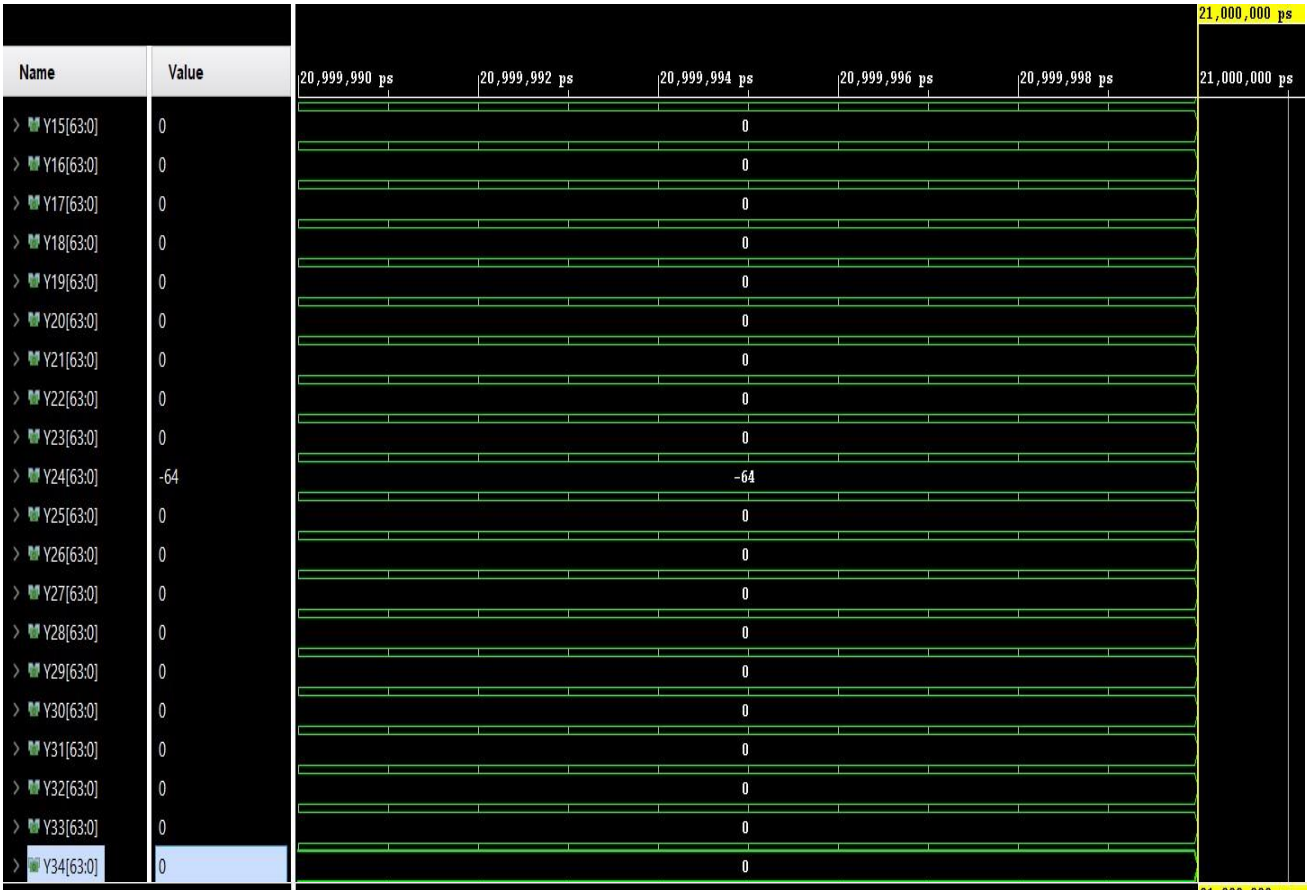
Figure 5.11.1 Schematic of 64-point 64-bit FFT in Xilinx Vivado 2022.2

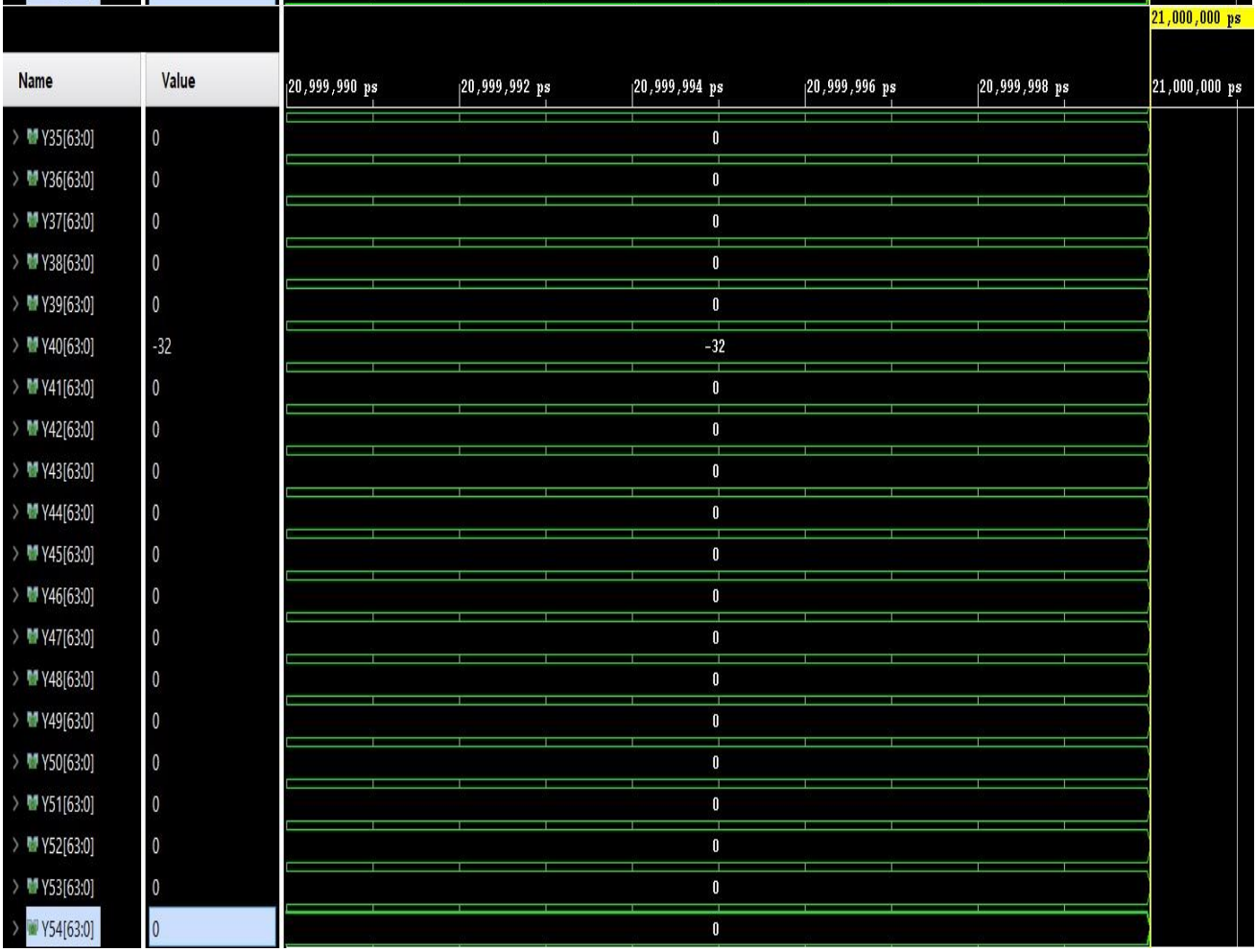

Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a0[63:0] 0
a1[63:0] 1
a2[63:0] 2
a3[63:0] 3
a4[63:0] 3
a5[63:0] 2
a6[63:0] 1
a7[63:0] 0
a8[63:0] 0
a9[63:0] 1
a10[63:0] 2
a11[63:0] 3
a12[63:0] 3
a13[63:0] 2
a14[63:0] 1
a15[63:0] 0
a16[63:0] 0
a17[63:0] 1
a18[63:0] 2
a19[63:0] 3


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a20[63:0] 3
a21[63:0] 2
a22[63:0] 1
a23[63:0] 0
a24[63:0] 0
a25[63:0] 1
a26[63:0] 2
a27[63:0] 3
a28[63:0] 3
a29[63:0] 2
a30[63:0] 1
a31[63:0] 0
a32[63:0] 0
a33[63:0] 1
a34[63:0] 2
a35[63:0] 3
a36[63:0] 3
a37[63:0] 2
a38[63:0] 1
a39[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a40[63:0] 0
a41[63:0] 1
a42[63:0] 2
a43[63:0] 3
a44[63:0] 3
a45[63:0] 2
a46[63:0] 1
a47[63:0] 0
a48[63:0] 0
a49[63:0] 1
a50[63:0] 2
a51[63:0] 3
a52[63:0] 3
a53[63:0] 2
a54[63:0] 1
a55[63:0] 0
a56[63:0] 0
a57[63:0] 1
a58[63:0] 2
a59[63:0] 3


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a60[63:0] 3
a61[63:0] 2
a62[63:0] 1
a63[63:0] 0
W[3:0] 1
Y0[63:0] 96
Y1[63:0] 0
Y2[63:0] 0
Y3[63:0] 0
Y4[63:0] 0
Y5[63:0] 0
Y6[63:0] 0
Y7[63:0] 0
Y8[63:0] 0
Y9[63:0] 0
Y10[63:0] 0
Y11[63:0] 0
Y12[63:0] 0
Y13[63:0] 0
Y14[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Y15[63:0] 0
Y16[63:0] 0
Y17[63:0] 0
Y18[63:0] 0
Y19[63:0] 0
Y20[63:0] 0
Y21[63:0] 0
Y22[63:0] 0
Y23[63:0] 0
Y24[63:0] -64
Y25[63:0] 0
Y26[63:0] 0
Y27[63:0] 0
Y28[63:0] 0
Y29[63:0] 0
Y30[63:0] 0
Y31[63:0] 0
Y32[63:0] 0
Y33[63:0] 0
Y34[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Y35[63:0] 0
Y36[63:0] 0
Y37[63:0] 0
Y38[63:0] 0
Y39[63:0] 0
Y40[63:0] -32
Y41[63:0] 0
Y42[63:0] 0
Y43[63:0] 0
Y44[63:0] 0
Y45[63:0] 0
Y46[63:0] 0
Y47[63:0] 0
Y48[63:0] 0
Y49[63:0] 0
Y50[63:0] 0
Y51[63:0] 0
Y52[63:0] 0
Y53[63:0] 0
Y54[63:0] 0

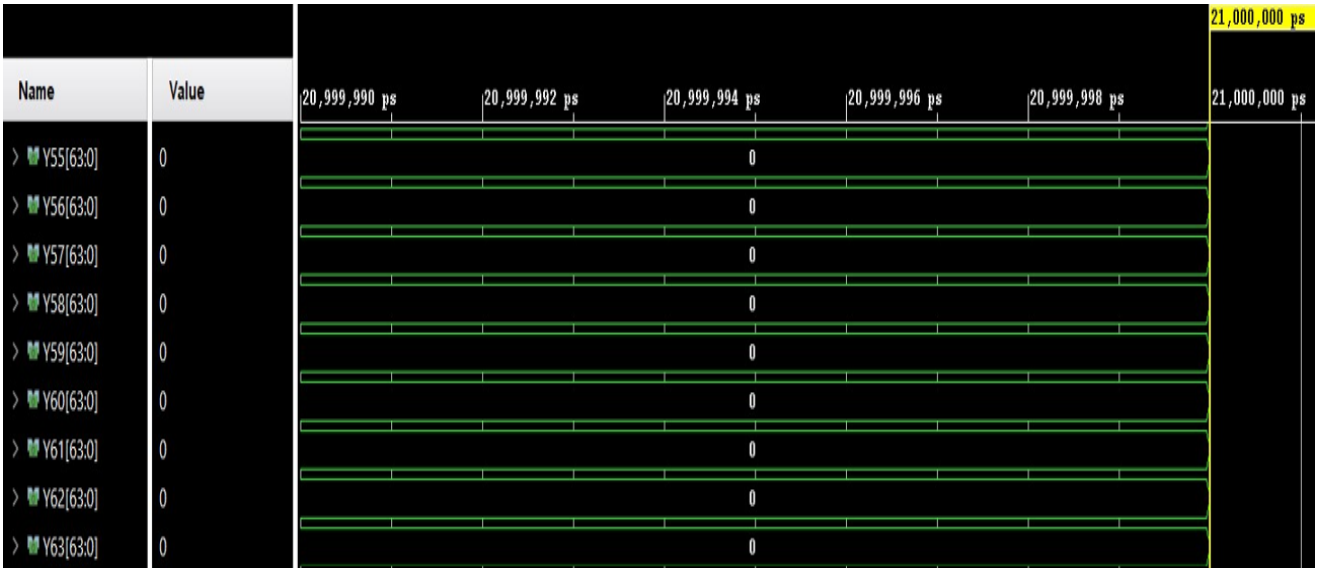
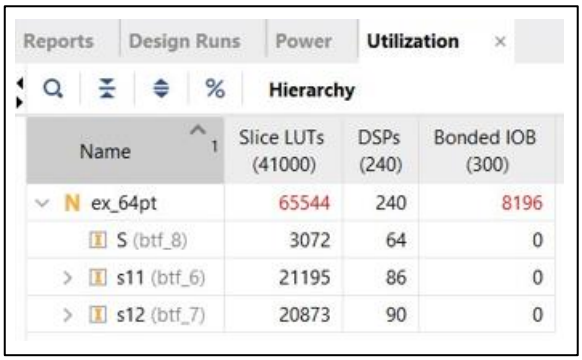


Figure 5.11.2 Simulation results of 64-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of 64-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.



Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
ex_64pt	65544	240	8196
S (btf_8)	3072	64	0
s11 (btf_6)	21195	86	0
s12 (btf_7)	20873	90	0

Figure 5.11.3 Utilization report of 64-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	64	29	24	a63[3]	Y0[61]	37.898	25.094	12.805	∞	input port clock
Path 2	∞	64	29	24	a63[3]	Y10[61]	37.898	25.094	12.805	∞	input port clock
Path 3	∞	64	29	24	a63[3]	Y11[61]	37.898	25.094	12.805	∞	input port clock
Path 4	∞	64	29	24	a63[3]	Y12[61]	37.898	25.094	12.805	∞	input port clock
Path 5	∞	64	29	24	a63[3]	Y16[61]	37.898	25.094	12.805	∞	input port clock
Path 6	∞	64	29	24	a63[3]	Y17[61]	37.898	25.094	12.805	∞	input port clock
Path 7	∞	64	29	24	a63[3]	Y18[61]	37.898	25.094	12.805	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	10	7	9840	W[3]	Y21[21]	2.671	1.859	0.812	-∞	input port clock
Path 12	∞	10	7	9840	W[3]	Y21[25]	2.671	1.859	0.812	-∞	input port clock
Path 13	∞	10	7	9840	W[3]	Y21[29]	2.671	1.859	0.812	-∞	input port clock
Path 14	∞	10	7	9840	W[3]	Y22[21]	2.671	1.859	0.812	-∞	input port clock
Path 15	∞	10	7	9840	W[3]	Y22[25]	2.671	1.859	0.812	-∞	input port clock
Path 16	∞	10	7	9840	W[3]	Y22[29]	2.671	1.859	0.812	-∞	input port clock
Path 17	∞	10	7	9840	W[3]	Y23[21]	2.671	1.859	0.812	-∞	input port clock

Figure 5.11.4 Delay report of 64-point 64-bit FFT in Xilinx Vivado 2022.2

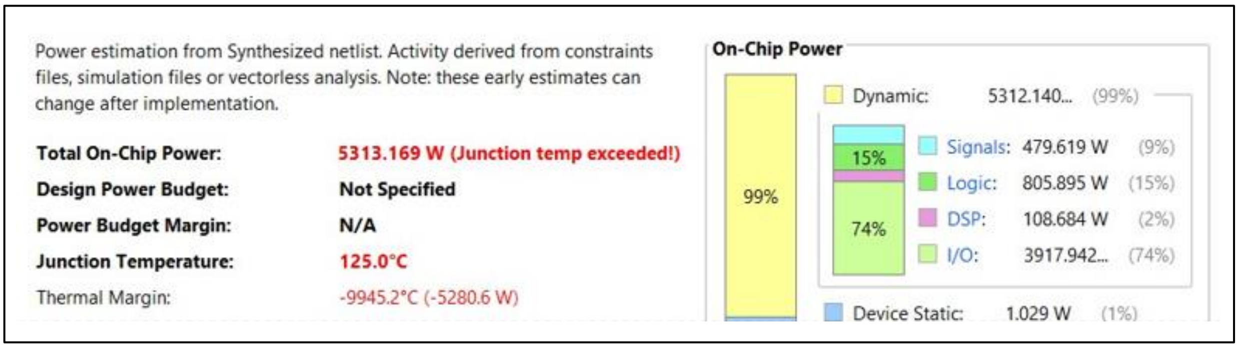


Figure 5.11.5 Power report of 64-point 64-bit FFT in Xilinx Vivado 2022.2

5.12 PROPOSED SIXTY-FOUR POINT 64-BIT FFT BASED ON DA

On simulating the proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1 and for select line = 3 the outputs obtained are X(n) = {96, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

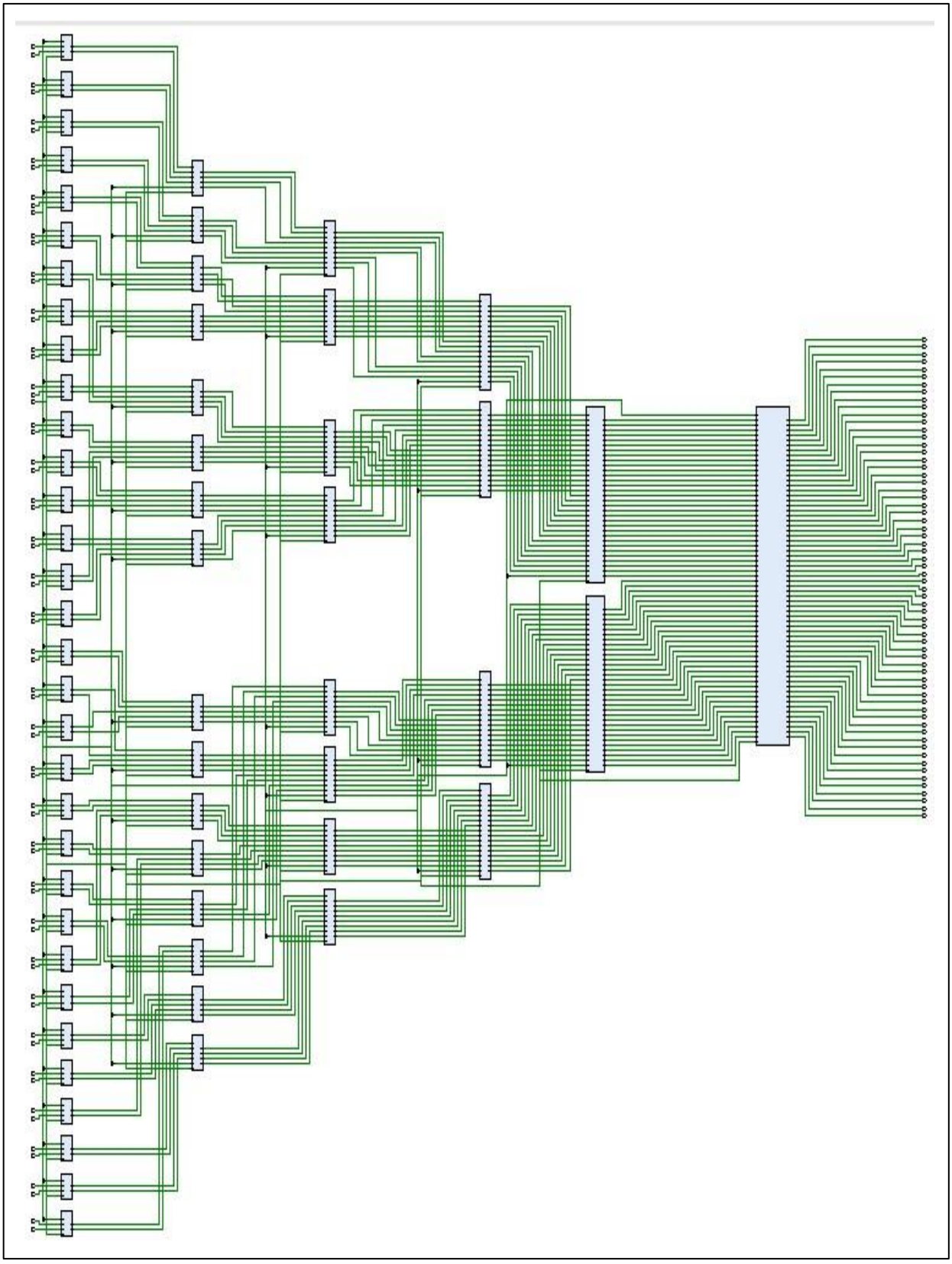
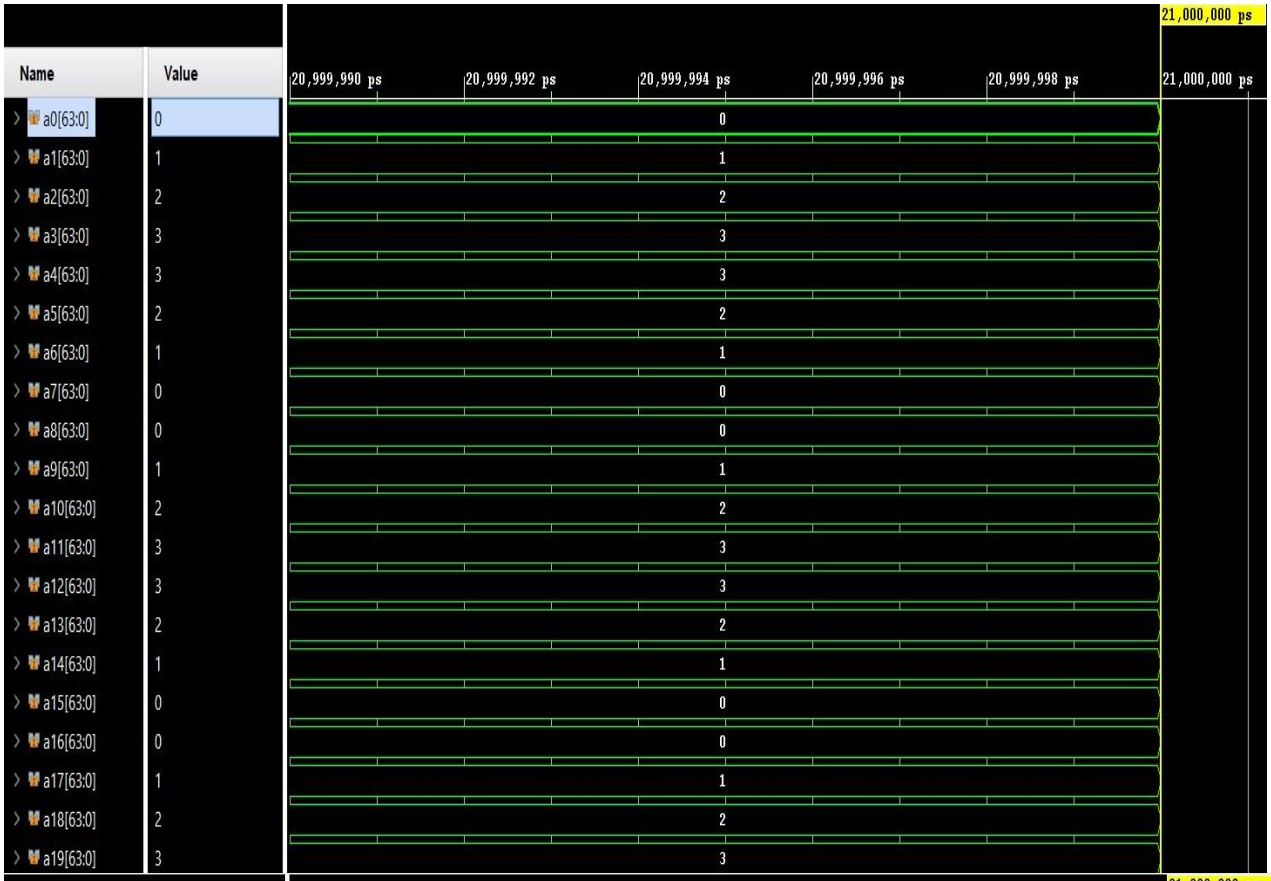
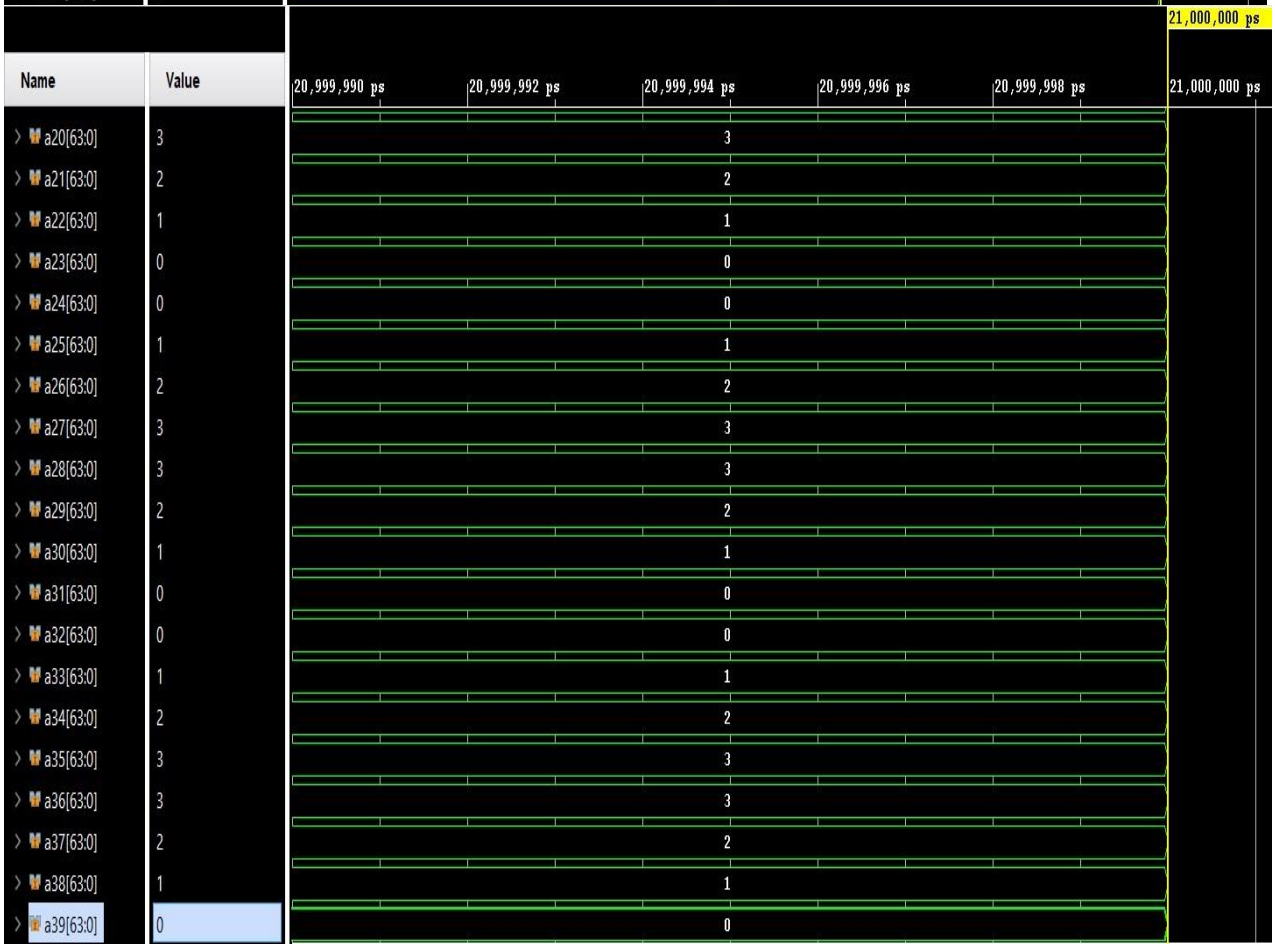
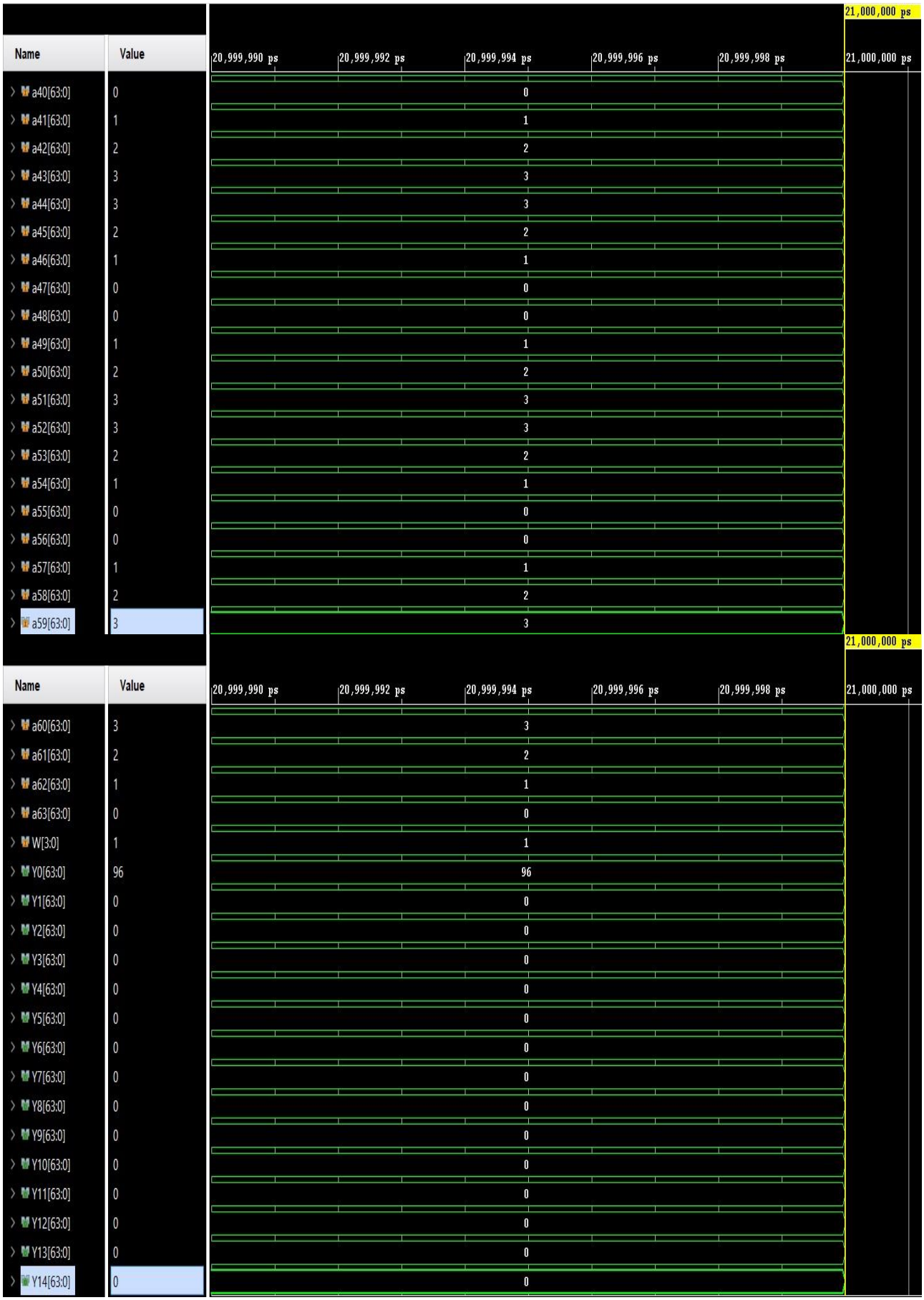
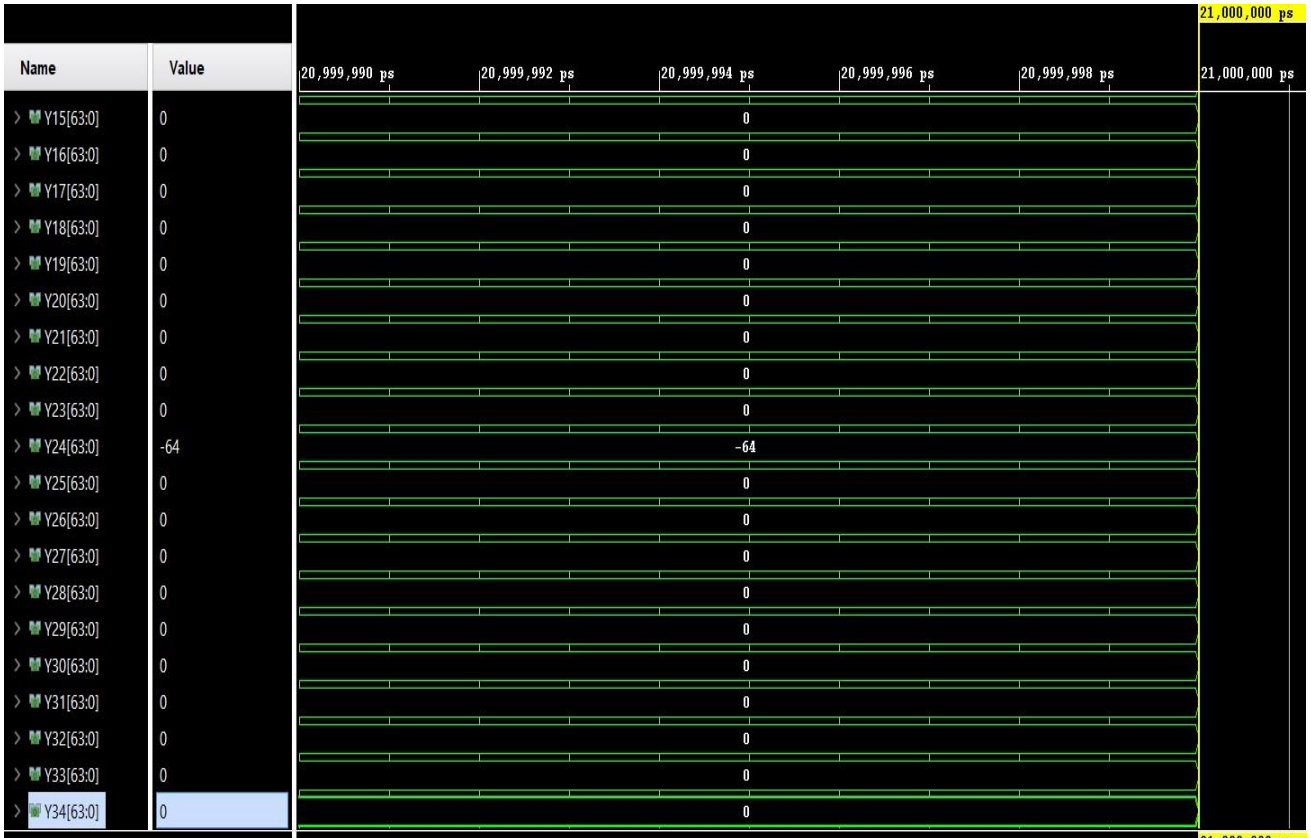


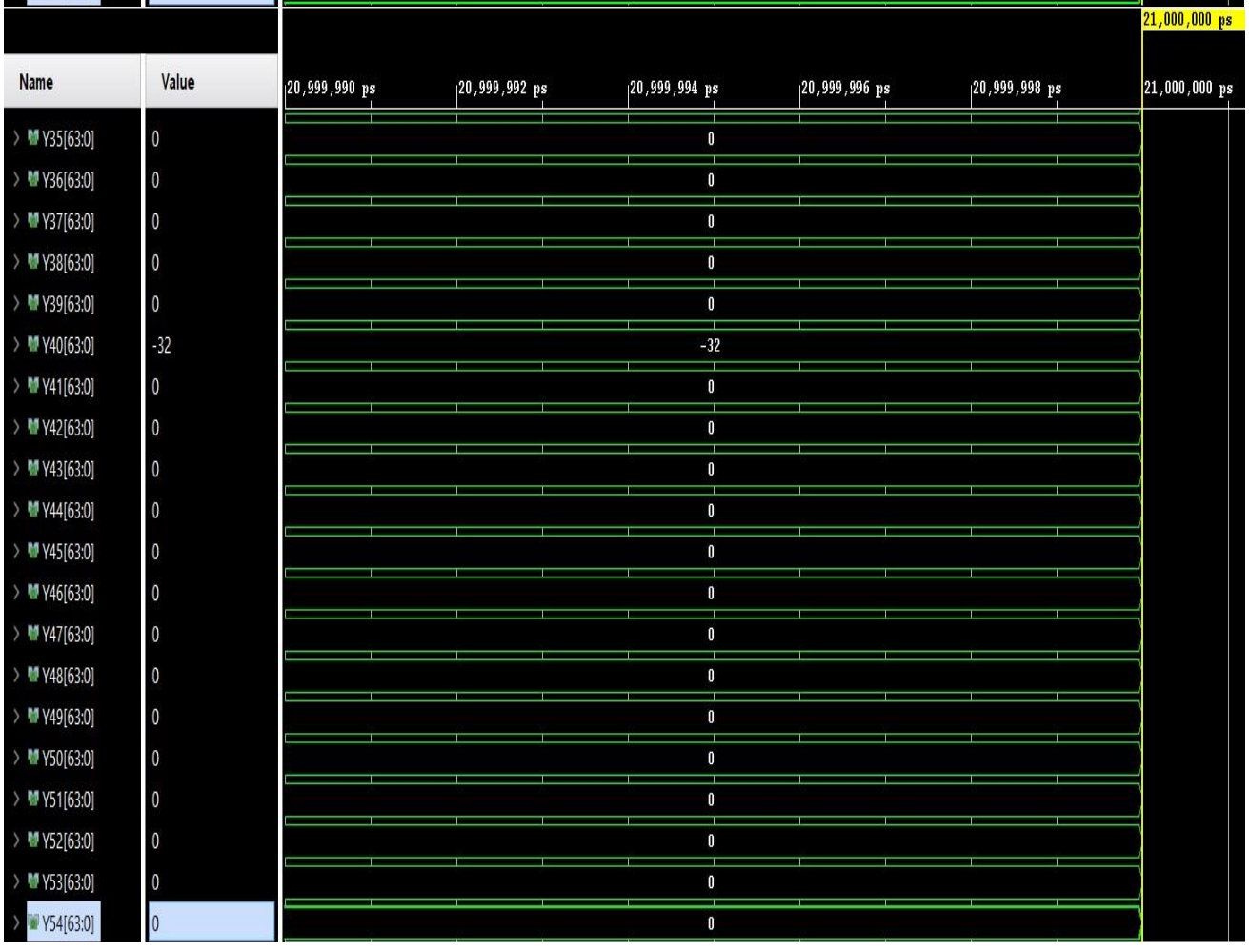
Figure 5.12.1 Schematic of proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a0[63:0] 0
a1[63:0] 1
a2[63:0] 2
a3[63:0] 3
a4[63:0] 3
a5[63:0] 2
a6[63:0] 1
a7[63:0] 0
a8[63:0] 0
a9[63:0] 1
a10[63:0] 2
a11[63:0] 3
a12[63:0] 3
a13[63:0] 2
a14[63:0] 1
a15[63:0] 0
a16[63:0] 0
a17[63:0] 1
a18[63:0] 2
a19[63:0] 3


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a20[63:0] 3
a21[63:0] 2
a22[63:0] 1
a23[63:0] 0
a24[63:0] 0
a25[63:0] 1
a26[63:0] 2
a27[63:0] 3
a28[63:0] 3
a29[63:0] 2
a30[63:0] 1
a31[63:0] 0
a32[63:0] 0
a33[63:0] 1
a34[63:0] 2
a35[63:0] 3
a36[63:0] 3
a37[63:0] 2
a38[63:0] 1
a39[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a40[63:0] 0
a41[63:0] 1
a42[63:0] 2
a43[63:0] 3
a44[63:0] 3
a45[63:0] 2
a46[63:0] 1
a47[63:0] 0
a48[63:0] 0
a49[63:0] 1
a50[63:0] 2
a51[63:0] 3
a52[63:0] 3
a53[63:0] 2
a54[63:0] 1
a55[63:0] 0
a56[63:0] 0
a57[63:0] 1
a58[63:0] 2
a59[63:0] 3
Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a60[63:0] 3
a61[63:0] 2
a62[63:0] 1
a63[63:0] 0
W[3:0] 1
Y0[63:0] 96
Y1[63:0] 0
Y2[63:0] 0
Y3[63:0] 0
Y4[63:0] 0
Y5[63:0] 0
Y6[63:0] 0
Y7[63:0] 0
Y8[63:0] 0
Y9[63:0] 0
Y10[63:0] 0
Y11[63:0] 0
Y12[63:0] 0
Y13[63:0] 0
Y14[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Y15[63:0] 0
Y16[63:0] 0
Y17[63:0] 0
Y18[63:0] 0
Y19[63:0] 0
Y20[63:0] 0
Y21[63:0] 0
Y22[63:0] 0
Y23[63:0] 0
Y24[63:0] -64
Y25[63:0] 0
Y26[63:0] 0
Y27[63:0] 0
Y28[63:0] 0
Y29[63:0] 0
Y30[63:0] 0
Y31[63:0] 0
Y32[63:0] 0
Y33[63:0] 0
Y34[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Y35[63:0] 0
Y36[63:0] 0
Y37[63:0] 0
Y38[63:0] 0
Y39[63:0] 0
Y40[63:0] -32
Y41[63:0] 0
Y42[63:0] 0
Y43[63:0] 0
Y44[63:0] 0
Y45[63:0] 0
Y46[63:0] 0
Y47[63:0] 0
Y48[63:0] 0
Y49[63:0] 0
Y50[63:0] 0
Y51[63:0] 0
Y52[63:0] 0
Y53[63:0] 0
Y54[63:0] 0

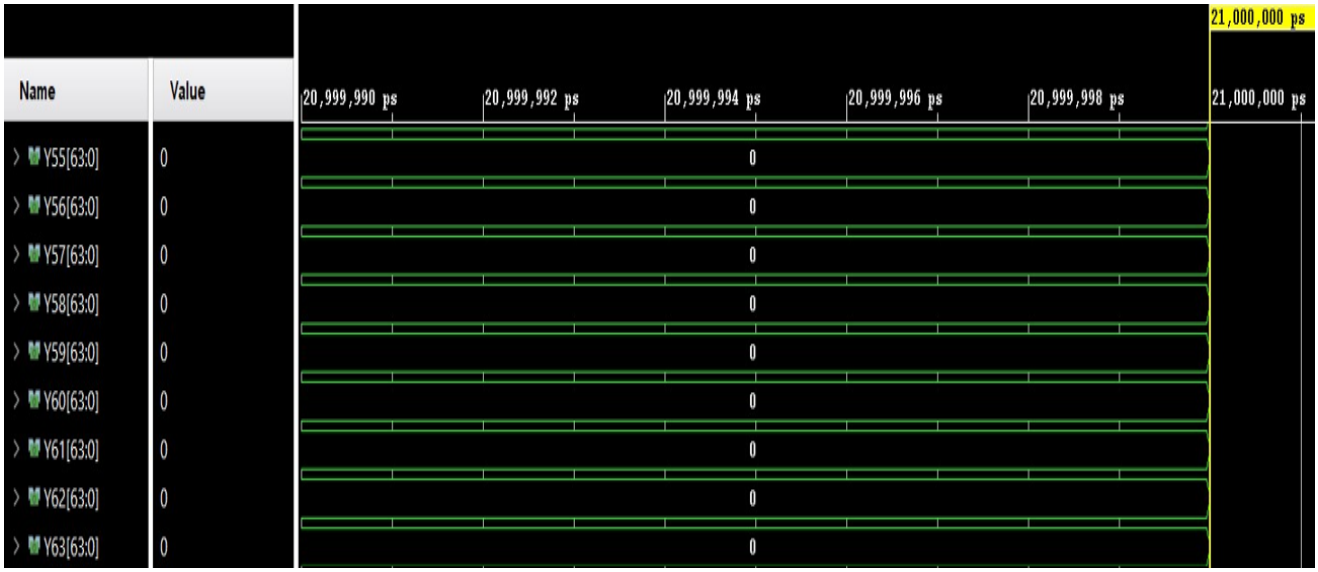


Figure 5.12.2 Simulation results of proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
lut_based	109952	235	8198
s1A (btf_a_8)	937	0	0
s1B (btf_a_9)	670	0	0
s1C (btf_a_10)	816	0	0
s1D (btf_a_11)	846	0	0
s1E (btf_a_12)	909	0	0
s1F (btf_a_13)	734	0	0

Figure 5.12.3 Utilization report of proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	63	32	8	a63[1]	Y47[63]	32.365	18.838	13.527	∞	input port clock
Path 2	∞	63	32	8	a63[1]	Y55[63]	32.314	18.769	13.545	∞	input port clock
Path 3	∞	63	32	8	a63[1]	Y47[62]	32.313	18.805	13.508	∞	input port clock
Path 4	∞	63	32	8	a63[1]	Y61[63]	32.304	18.594	13.710	∞	input port clock
Path 5	∞	63	32	8	a63[1]	Y47[61]	32.293	18.884	13.409	∞	input port clock
Path 6	∞	63	32	8	a63[1]	Y47[59]	32.276	18.749	13.527	∞	input port clock
Path 7	∞	63	32	8	a63[1]	Y55[62]	32.262	18.736	13.526	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	32158	w[1]	Y0[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	32158	w[1]	Y0[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	32158	w[1]	Y0[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	32158	w[1]	Y0[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	32158	w[1]	Y0[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	32158	w[1]	Y0[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	32158	w[1]	Y0[15]	1.905	1.413	0.492	-∞	input port clock

Figure 5.12.4 Delay report of proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2

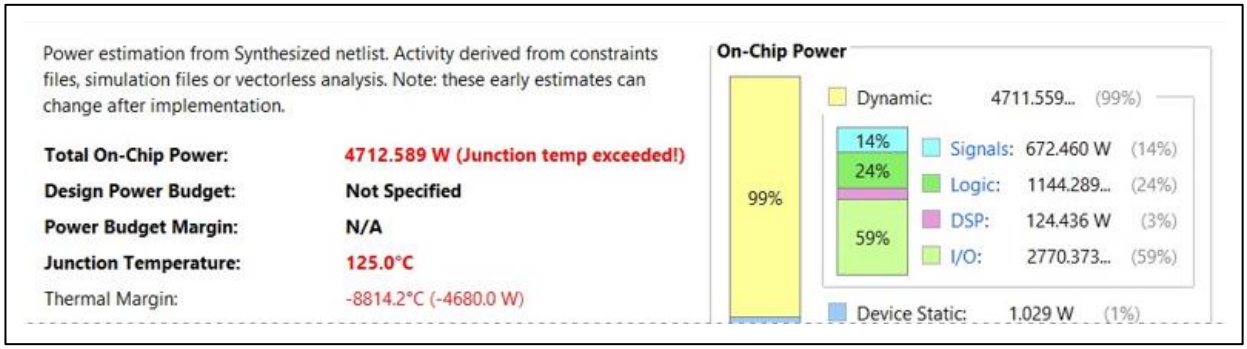


Figure 5.12.5 Power report of proposed 64-point 64-bit FFT in Xilinx Vivado 2022.2

Performance Evaluation of 64-bit 2, 4, 8, 16, 32, 64-point FFT

Table 5: For Kintex-7 xc7k70tfbv676-1

n-point (64-bit)	Utilization						Delay (in ns) (Setup + Hold)		Power (in W)	
	Existing			Proposed						
	LUTs (41000)	DSPs (240)	IOBs (300)	LUTs (41000)	DSPs (240)	IOBs (300)	Existing	Proposed	Existing	Proposed
2-point	158	4	260	56	-	36	13.665	8.264	125.599	9.034
4-point	632	16	524	2152	-	520	21.362	15.778	268.869	198.118
8-point	1896	48	1056	6404	-	1040	27.898	19.722	569.339	445.073
16-point	5056	128	2108	17556	-	2080	34.434	23.566	1160.3	977.859
32-point	17560	240	4100	44312	-	4100	39.786	27.862	2444.77	2026.559
64-point	65544	240	8196	109952	235	8198	40.569	34.270	5313.169	4712.589

5.13 PROPOSED TWO POINT 64-BIT FFT (MUX BASED)

On simulating the proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

{a11, a12} = {a1, a2} = {1, 2}, twiddle factor W20 = 1 and

- for select line w = 3, the outputs obtained are {bx0, bx1} = {3, -1}
- for select line w = 2, the outputs obtained are {bx0, bx1} = {1, 1}
- for select line w = 1, the outputs obtained are {bx0, bx1} = {2, -2}
- for select line w = 0, the outputs obtained are {bx0, bx1} = {0, 0}

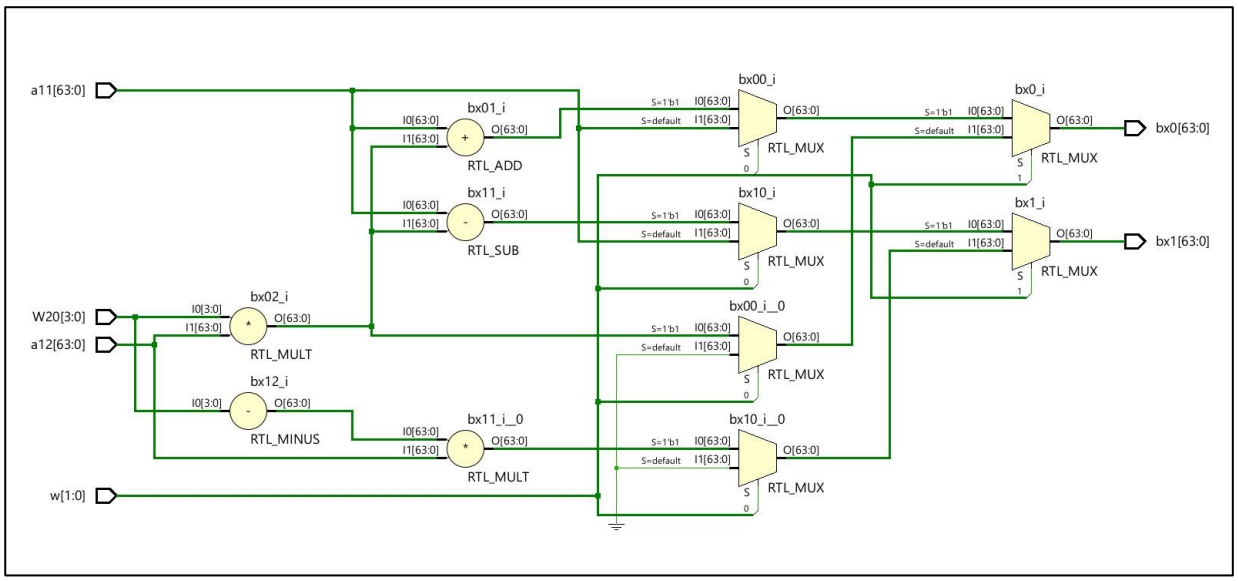


Fig 5.13.1 Schematic of proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2

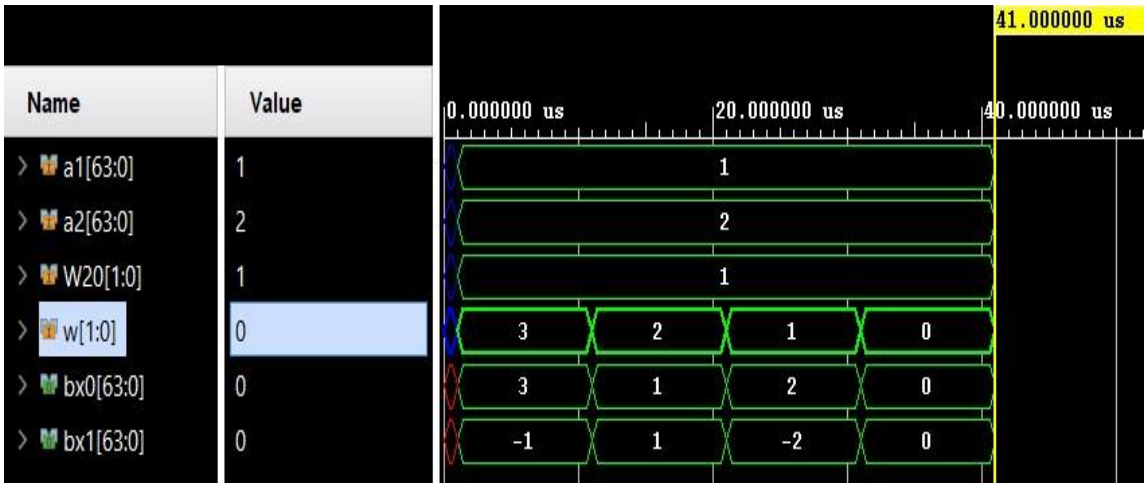


Fig 5.13.2 Simulation result of proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
btf1	259	14	262

Fig 5.13.3 Utilization report of proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	13	10	64	W20[1]	bx1[62]	15.066	11.937	3.129	∞	input port clock
Path 2	∞	12	10	64	W20[1]	bx1[58]	15.008	11.879	3.129	∞	input port clock
Path 3	∞	13	10	64	W20[1]	bx1[61]	14.888	11.861	3.027	∞	input port clock
Path 4	∞	13	10	64	W20[1]	bx1[63]	14.844	11.858	2.986	∞	input port clock
Path 5	∞	12	10	64	W20[1]	bx1[57]	14.830	11.803	3.027	∞	input port clock
Path 6	∞	12	10	64	W20[1]	bx1[60]	14.816	11.833	2.983	∞	input port clock
Path 7	∞	12	10	64	W20[1]	bx1[59]	14.786	11.800	2.986	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	193	w[0]	bx1[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	193	w[0]	bx1[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	193	w[0]	bx1[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	193	w[0]	bx1[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	193	w[0]	bx1[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	193	w[0]	bx1[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	193	w[0]	bx1[15]	1.905	1.413	0.492	-∞	input port clock

Fig 5.13.4 Delay report of proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**114.13 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125.0°C**
Thermal Margin:	-154.9°C (-81.5 W)
Effective θJA:	1.9°C/W

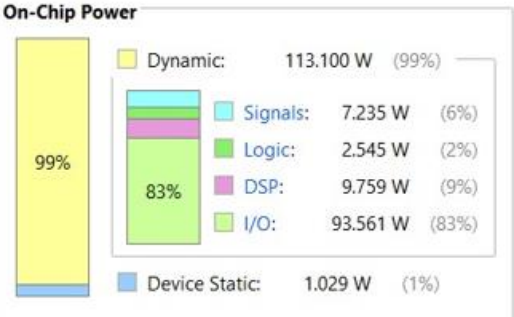


Fig 5.13.5 Power report of proposed MUX based 2-point 64-bit FFT in Xilinx Vivado 2022.2

5.14 PROPOSED FOUR POINT 64-BIT FFT (MUX BASED)

On simulating the proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs: {a0, a1, a2, a3} = {0, 1, 2, 3}, twiddle factors W20 = W40 = W41 = 1 and

- for select line w = 3, the outputs obtained are {A0, A1, A2, A3} = {6, -4, -2, 0}
- for select line w = 2, the outputs obtained are {A0, A1, A2, A3} = {0, 0, 0, 0}
- for select line w = 1, the outputs obtained are {A0, A1, A2, A3} = {3, -3, -3, 3}
- for select line w = 0, the outputs obtained are {A0, A1, A2, A3} = {0, 0, 0, 0}

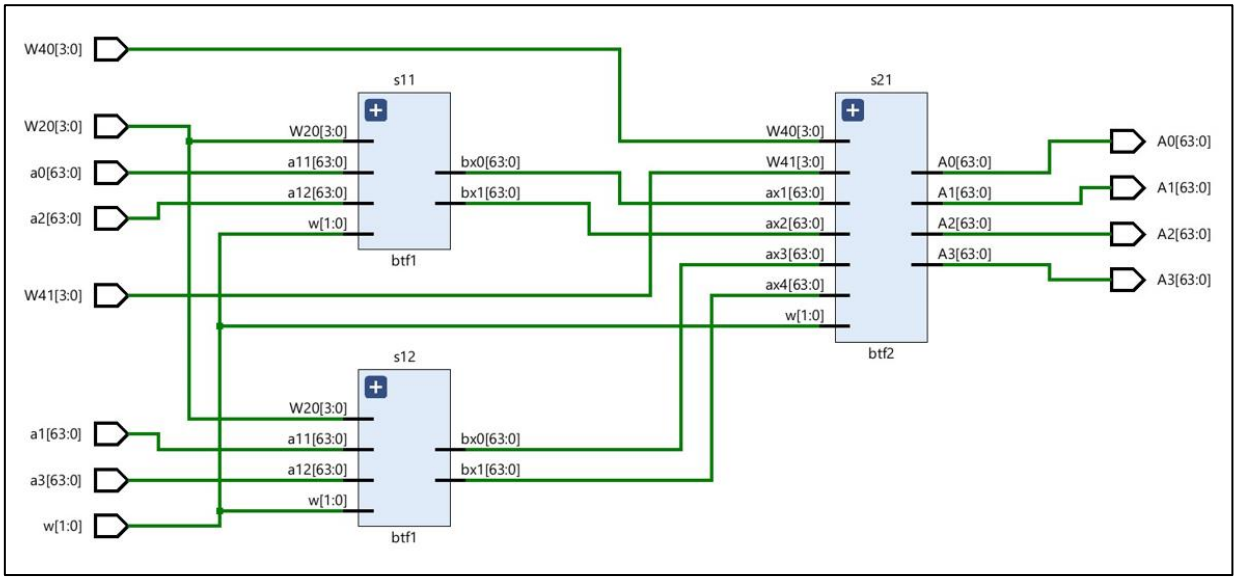


Fig 5.14.1 Schematic of proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2

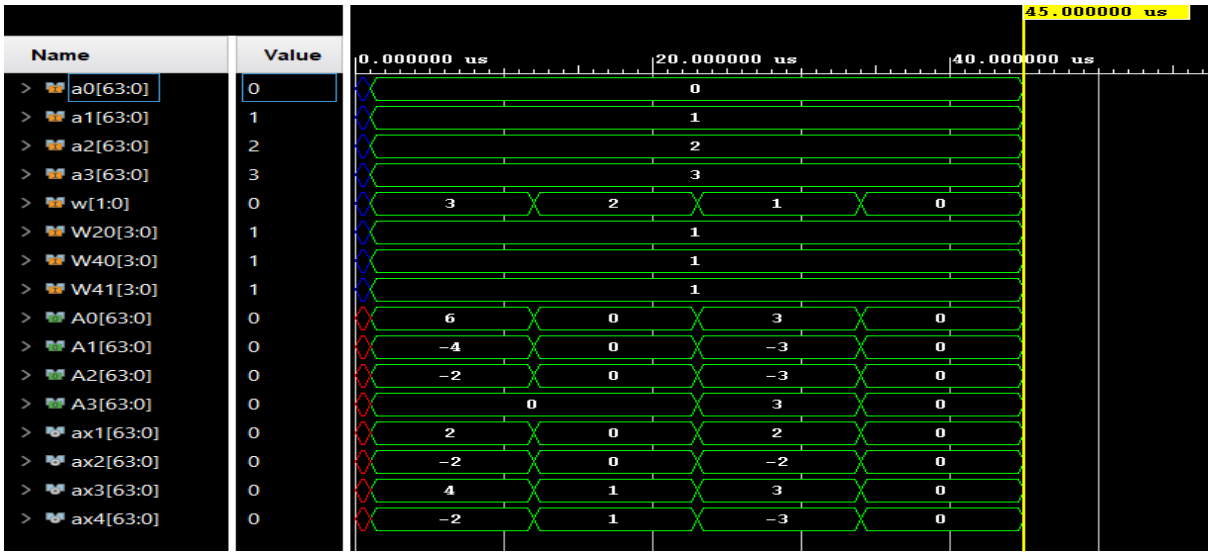


Fig 5.14.2 Simulation result of proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
fft	980	56	526
s11 (btf1)	451	14	0
s12 (btf1_0)	255	14	0
s21 (btf2)	274	28	0

Fig 5.14.3 Utilization report of proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	18	15	56	W20[1]	A1[62]	22.938	18.292	4.646	∞	input port clock
Path 2	∞	17	15	56	W20[1]	A1[58]	22.880	18.234	4.646	∞	input port clock
Path 3	∞	18	15	56	W20[1]	A1[61]	22.760	18.216	4.544	∞	input port clock
Path 4	∞	18	15	56	W20[1]	A1[63]	22.716	18.213	4.503	∞	input port clock
Path 5	∞	17	15	56	W20[1]	A1[57]	22.702	18.158	4.544	∞	input port clock
Path 6	∞	17	15	56	W20[1]	A1[60]	22.688	18.188	4.500	∞	input port clock
Path 7	∞	17	15	56	W20[1]	A1[59]	22.658	18.155	4.503	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	771	w[0]	A1[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	771	w[0]	A1[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	771	w[0]	A1[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	771	w[0]	A1[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	771	w[0]	A1[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	771	w[0]	A1[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	771	w[0]	A1[15]	1.905	1.413	0.492	-∞	input port clock

Fig 5.14.4 Delay report of proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2

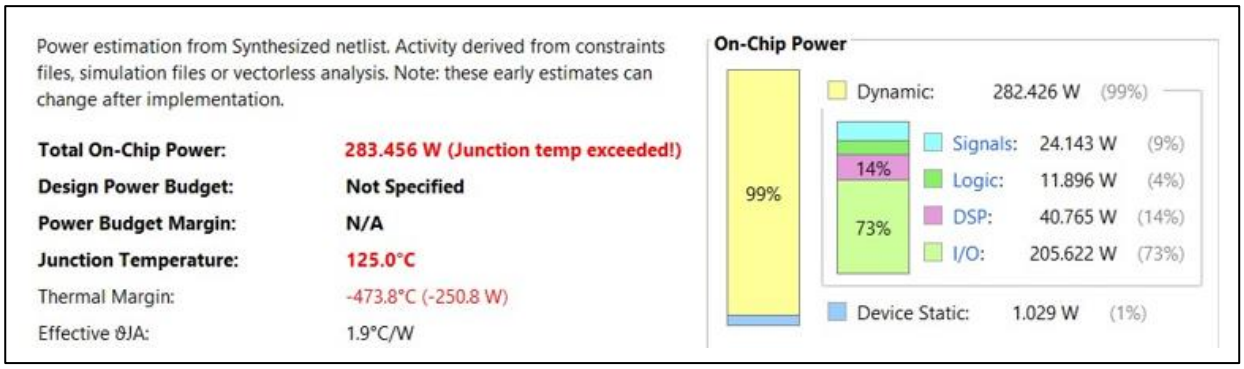


Fig 5.14.5 Power report of proposed MUX based 4-point 64-bit FFT in Xilinx Vivado 2022.2

5.15 PROPOSED EIGHT POINT 64-BIT FFT (MUX BASED)

On simulating the proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

{a0, a1, a2, a3, a4, a5, a6, a7} = {1, 2, 3, 4, 4, 3, 2, 1}, twiddle factors W20 = W40 = W41 = W80 = W81 = W82 = W83 = 1 and

- for select line w = 3, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {20, 0, 0, -8, 0, -4, 0, 0}
- for select line w = 2, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {1, 1, 1, 1, 1, 1, 1, 1}
- for select line w = 1, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {1, -1, -1, 1, -1, 1, 1, -1}
- for select line w = 0, the outputs obtained are {A0, A1, A2, A3, A4, A5, A6, A7} = {0, 0, 0, 0, 0, 0, 0, 0}

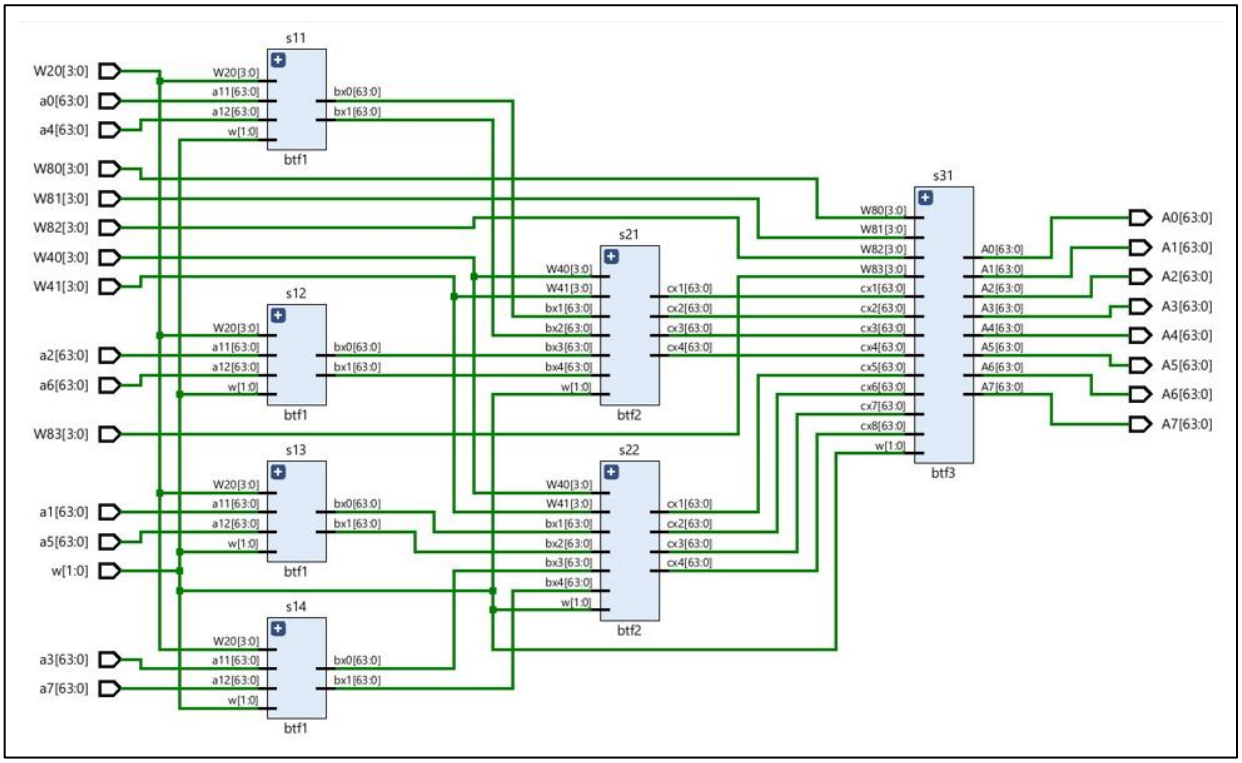


Fig 5.15.1 Schematic of proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2

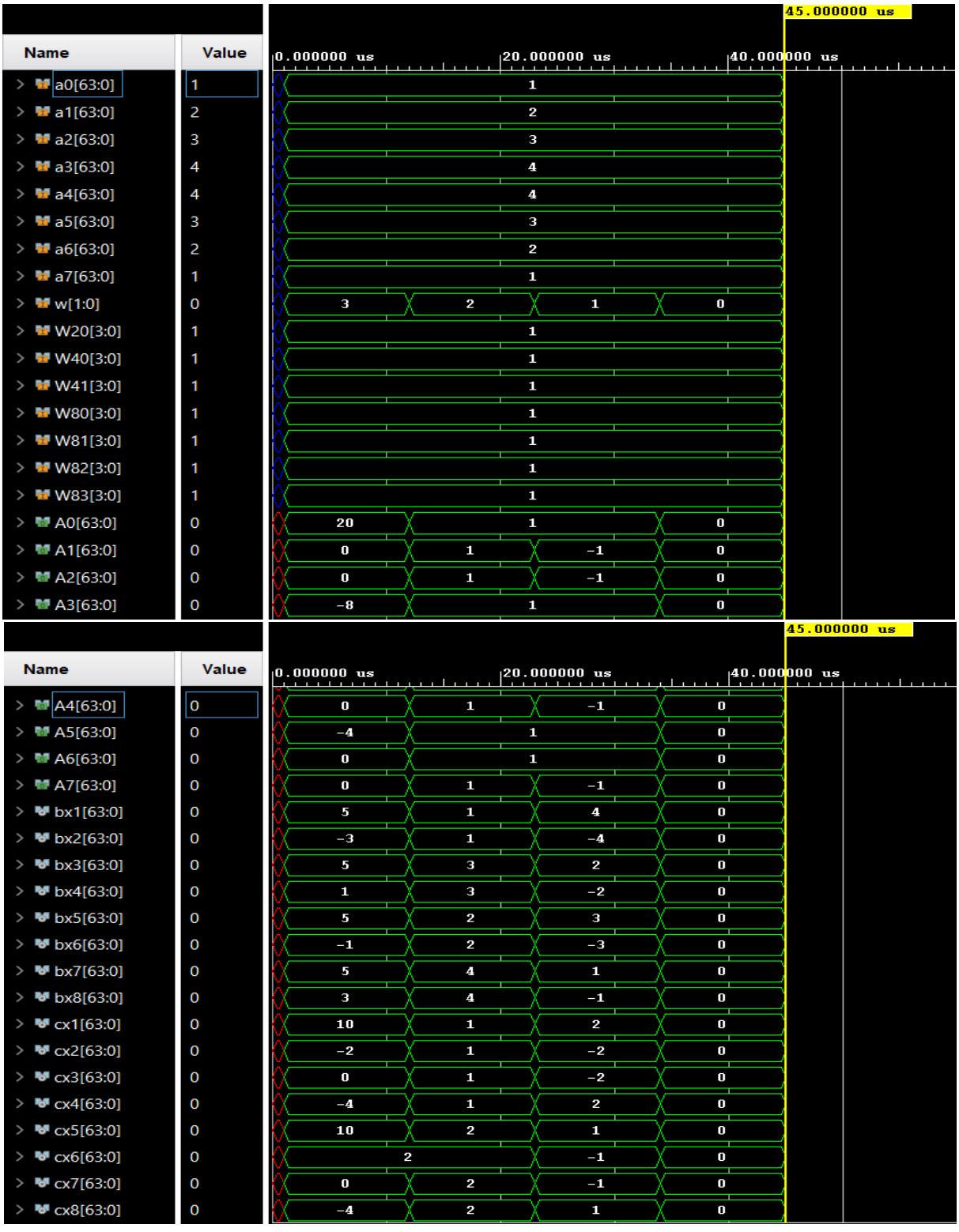


Fig 5.15.2 Simulation results of proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
fft	2928	168	1054
s11 (btf1)	209	14	0
s12 (btf1_0)	207	14	0
s13 (btf1_1)	206	14	0
s14 (btf1_2)	207	14	0
s21 (btf2)	461	28	0
s22 (btf2_3)	482	28	0
s31 (btf3)	881	56	0

Fig 5.15.3 Utilization report of proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	26	19	83	W20[1]	A5[62]	30.431	23.823	6.609	∞	input port clock
Path 2	∞	25	19	83	W20[1]	A5[58]	30.373	23.765	6.609	∞	input port clock
Path 3	∞	26	19	83	W20[1]	A5[61]	30.253	23.747	6.507	∞	input port clock
Path 4	∞	26	19	83	W20[1]	A5[63]	30.209	23.744	6.466	∞	input port clock
Path 5	∞	25	19	83	W20[1]	A5[57]	30.195	23.689	6.507	∞	input port clock
Path 6	∞	25	19	83	W20[1]	A5[60]	30.181	23.719	6.463	∞	input port clock
Path 7	∞	25	19	83	W20[1]	A5[59]	30.151	23.686	6.466	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	2312	w[0]	A4[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	2312	w[0]	A4[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	2312	w[0]	A4[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	2312	w[0]	A4[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	2312	w[0]	A4[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	2312	w[0]	A4[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	2312	w[0]	A4[15]	1.905	1.413	0.492	-∞	input port clock

Fig 5.15.4 Delay report of proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2

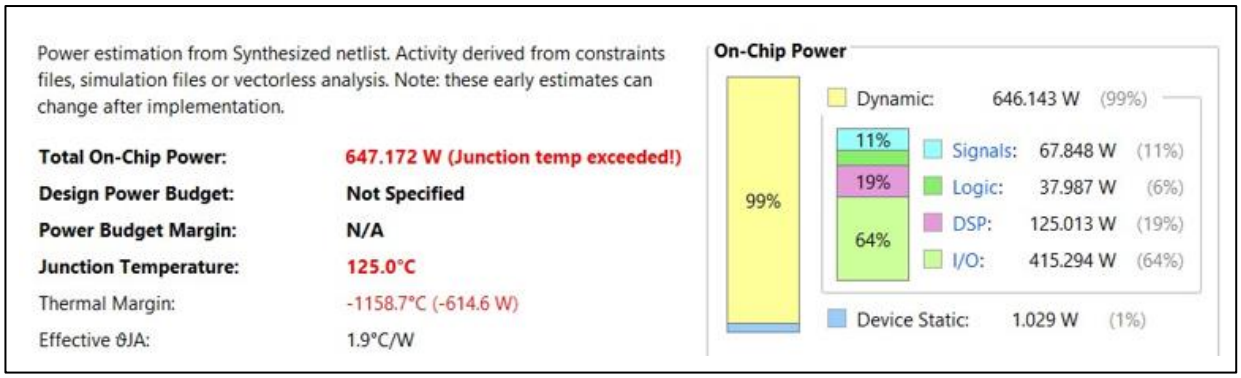


Fig 5.15.5 Power report of proposed MUX based 8-point 64-bit FFT in Xilinx Vivado 2022.2

5.16 PROPOSED SIXTEEN POINT 64-BIT FFT (MUX BASED)

On simulating the proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

x(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1 and for select line = 3 the outputs obtained are U(n) = X(n) = {24, 0, 0, 0, 0, 0, -16, 0, 0, 0, -8, 0, 0, 0, 0, 0}

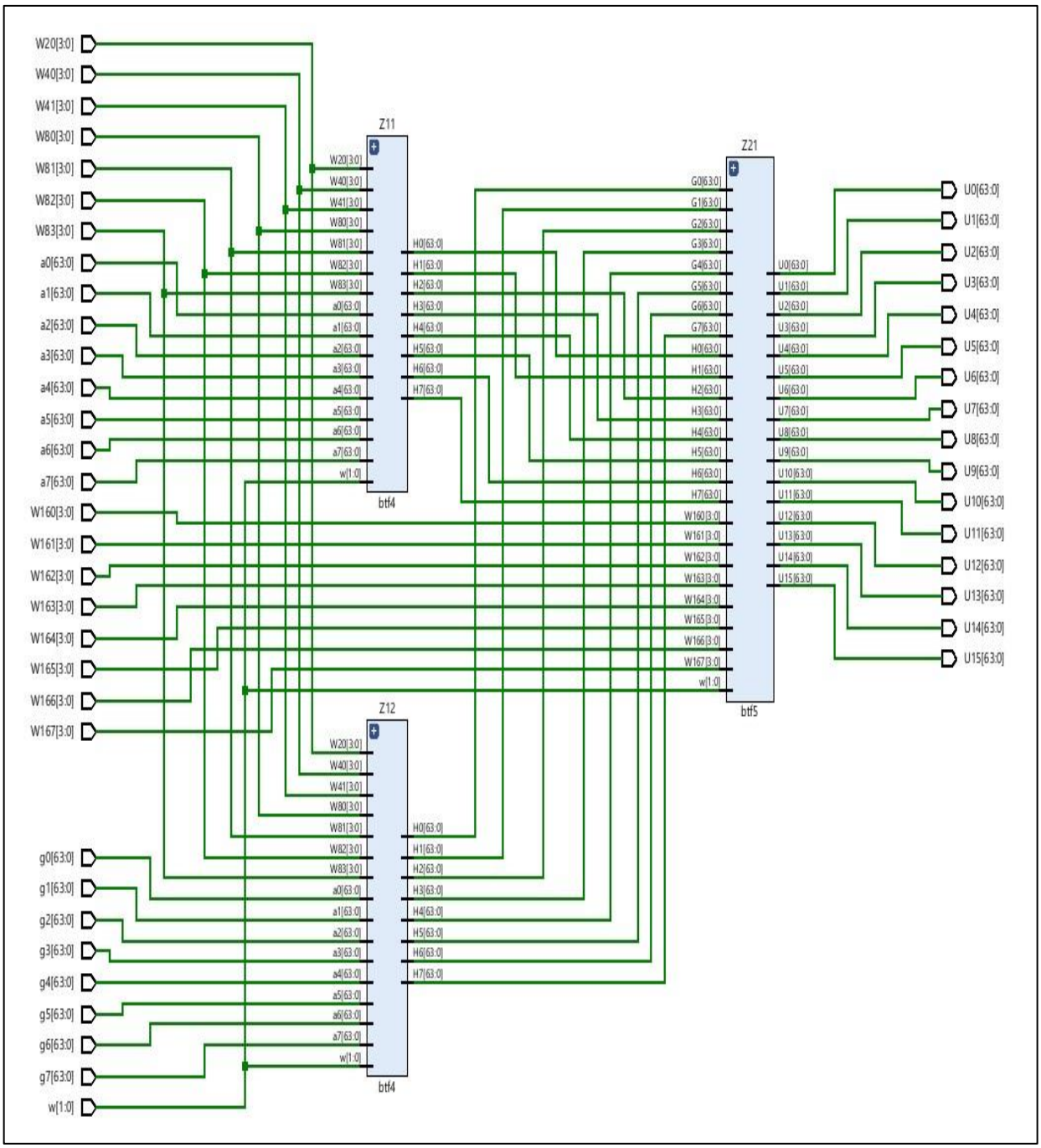
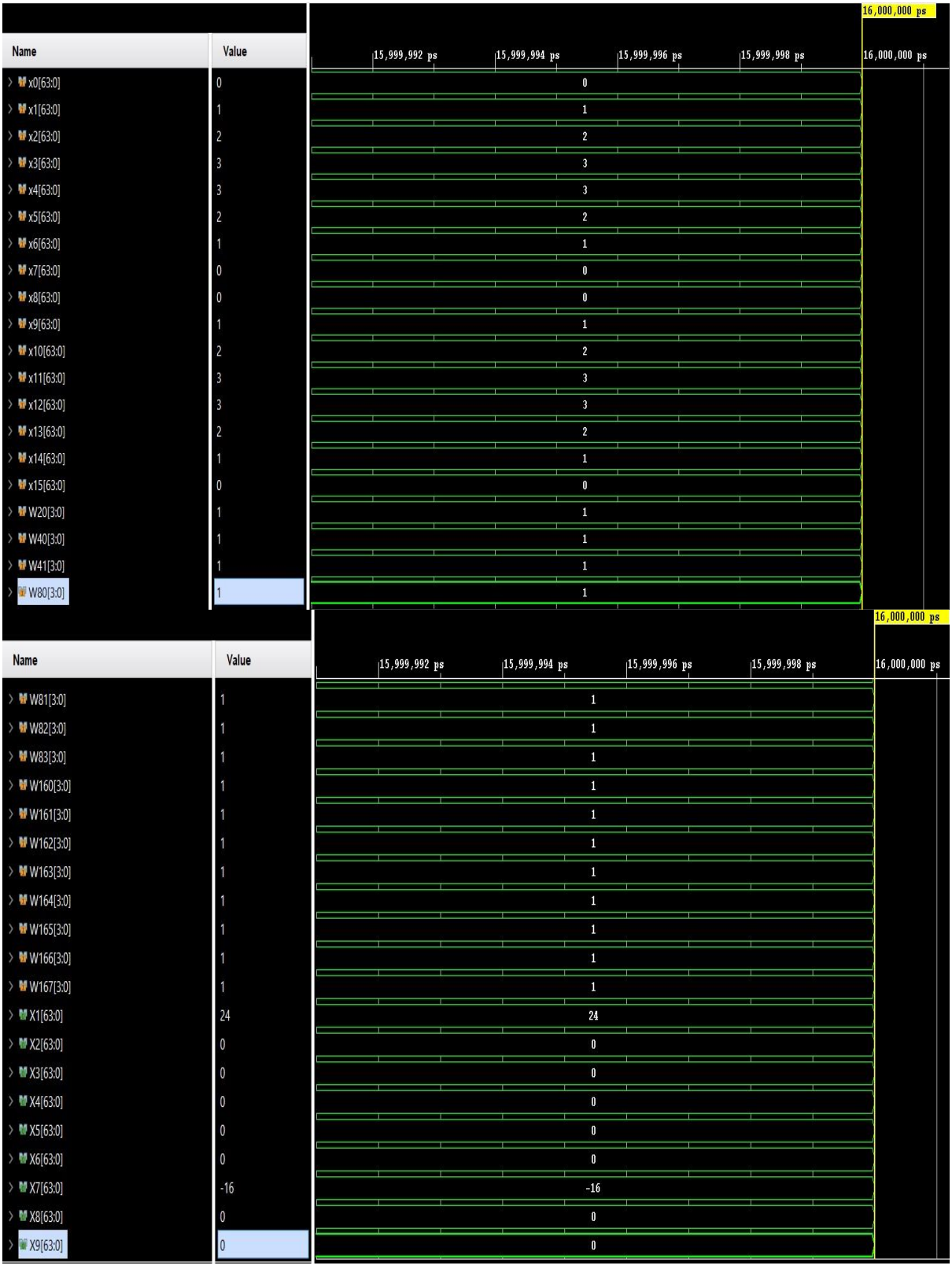


Fig 5.16.1 Schematic of proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2


Name
Value
15,999,992 ps
15,999,994 ps
15,999,996 ps
15,999,998 ps
16,000,000 ps
x0[63:0] 0
x1[63:0] 1
x2[63:0] 2
x3[63:0] 3
x4[63:0] 3
x5[63:0] 2
x6[63:0] 1
x7[63:0] 0
x8[63:0] 0
x9[63:0] 1
x10[63:0] 2
x11[63:0] 3
x12[63:0] 3
x13[63:0] 2
x14[63:0] 1
x15[63:0] 0
W20[3:0] 1
W40[3:0] 1
W41[3:0] 1
W80[3:0] 1
W81[3:0] 1
W82[3:0] 1
W83[3:0] 1
W160[3:0] 1
W161[3:0] 1
W162[3:0] 1
W163[3:0] 1
W164[3:0] 1
W165[3:0] 1
W166[3:0] 1
W167[3:0] 1
X1[63:0] 24
X2[63:0] 0
X3[63:0] 0
X4[63:0] 0
X5[63:0] 0
X6[63:0] 0
X7[63:0] -16
X8[63:0] 0
X9[63:0] 0

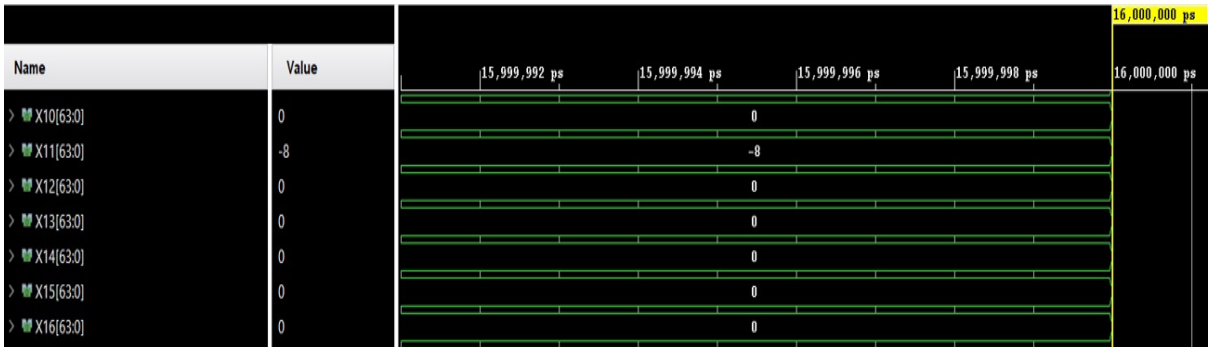


Fig 5.16.2 Simulation results of proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
fft	30424	144	2110
Z11 (btf4)	971	36	0
Z12 (btf4_0)	1190	36	0
Z21 (btf5)	1510	72	0

Fig 5.16.3 Utilization report of proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	60	29	1225	W20[2]	U5[62]	36.802	24.843	11.960	∞	input port clock
Path 2	∞	59	29	1225	W20[2]	U5[58]	36.744	24.785	11.960	∞	input port clock
Path 3	∞	60	29	1225	W20[2]	U5[61]	36.624	24.767	11.858	∞	input port clock
Path 4	∞	60	29	1225	W20[2]	U5[63]	36.580	24.764	11.817	∞	input port clock
Path 5	∞	59	29	1225	W20[2]	U5[57]	36.566	24.709	11.858	∞	input port clock
Path 6	∞	59	29	1225	W20[2]	U5[60]	36.552	24.739	11.814	∞	input port clock
Path 7	∞	59	29	1225	W20[2]	U5[59]	36.522	24.706	11.817	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	7764	w[0]	U10[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	7764	w[0]	U10[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	7764	w[0]	U10[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	7764	w[0]	U10[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	7764	w[0]	U10[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	7764	w[0]	U10[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	7764	w[0]	U10[15]	1.905	1.413	0.492	-∞	input port clock

Fig 5.16.4 Delay report of proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2

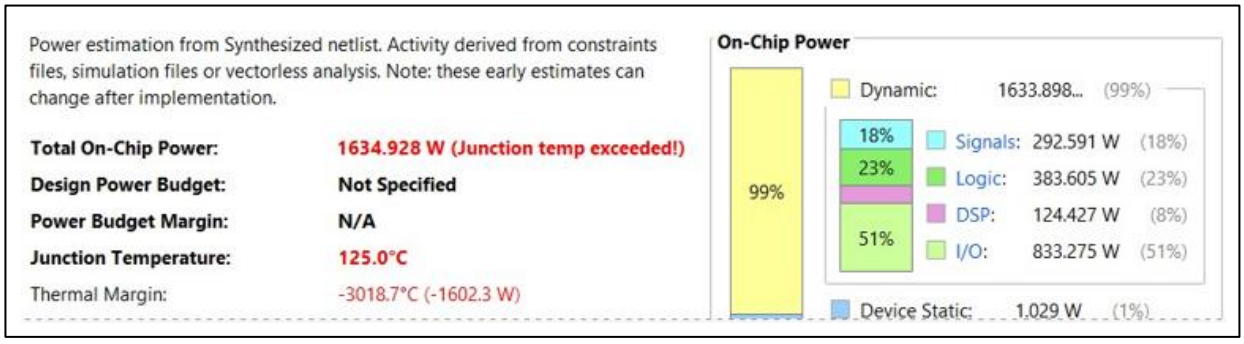


Fig 5.16.5 Power report of proposed MUX based 16-point 64-bit FFT in Xilinx Vivado 2022.2

5.17 PROPOSED THIRTY-TWO POINT 64-BIT FFT (MUX BASED)

On simulating the proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

a(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1 and for select line = 3 the outputs obtained are Z(n) = {48, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -32, 0, 0 0, 0, 0, 0, 0, -16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

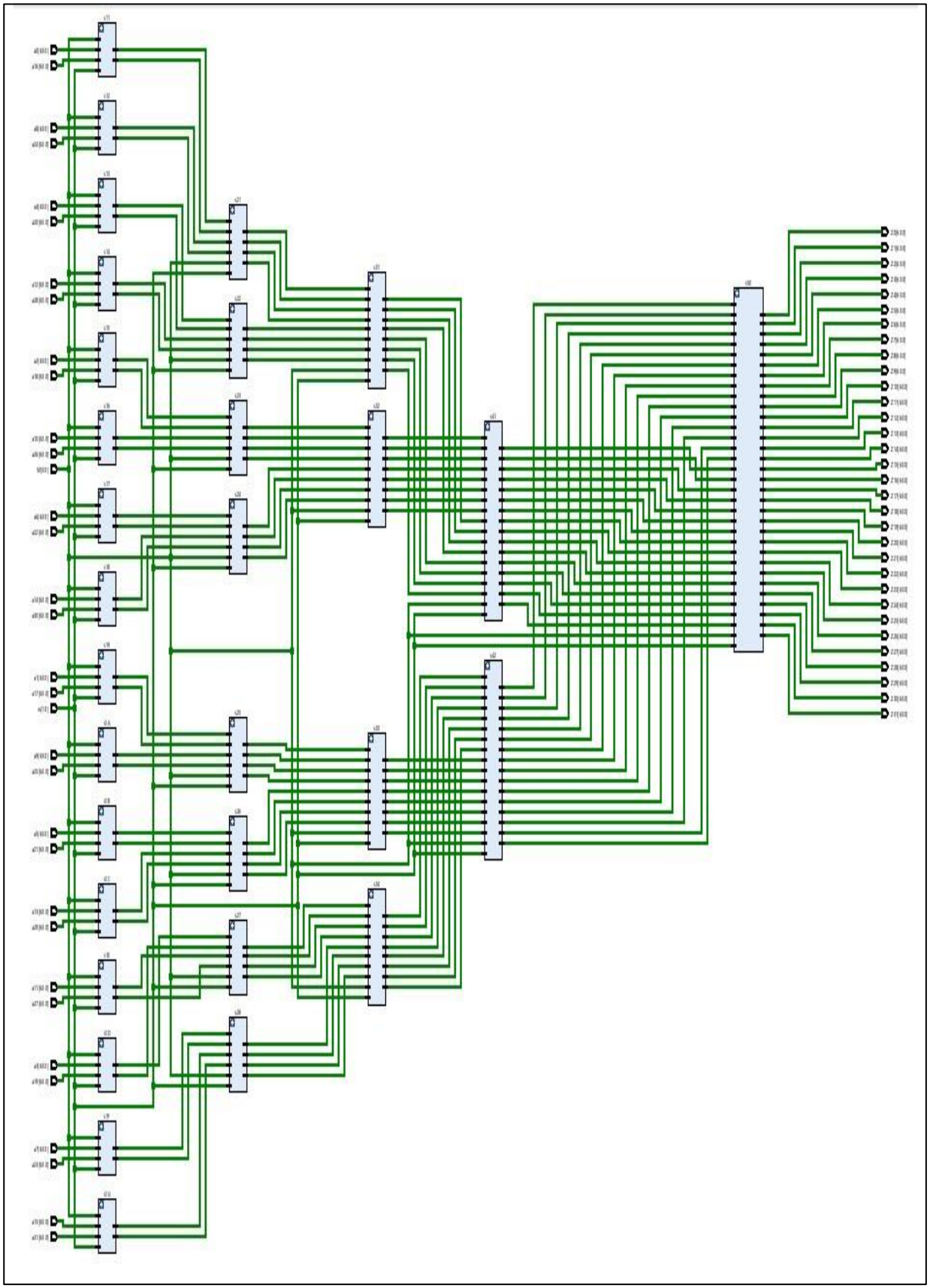
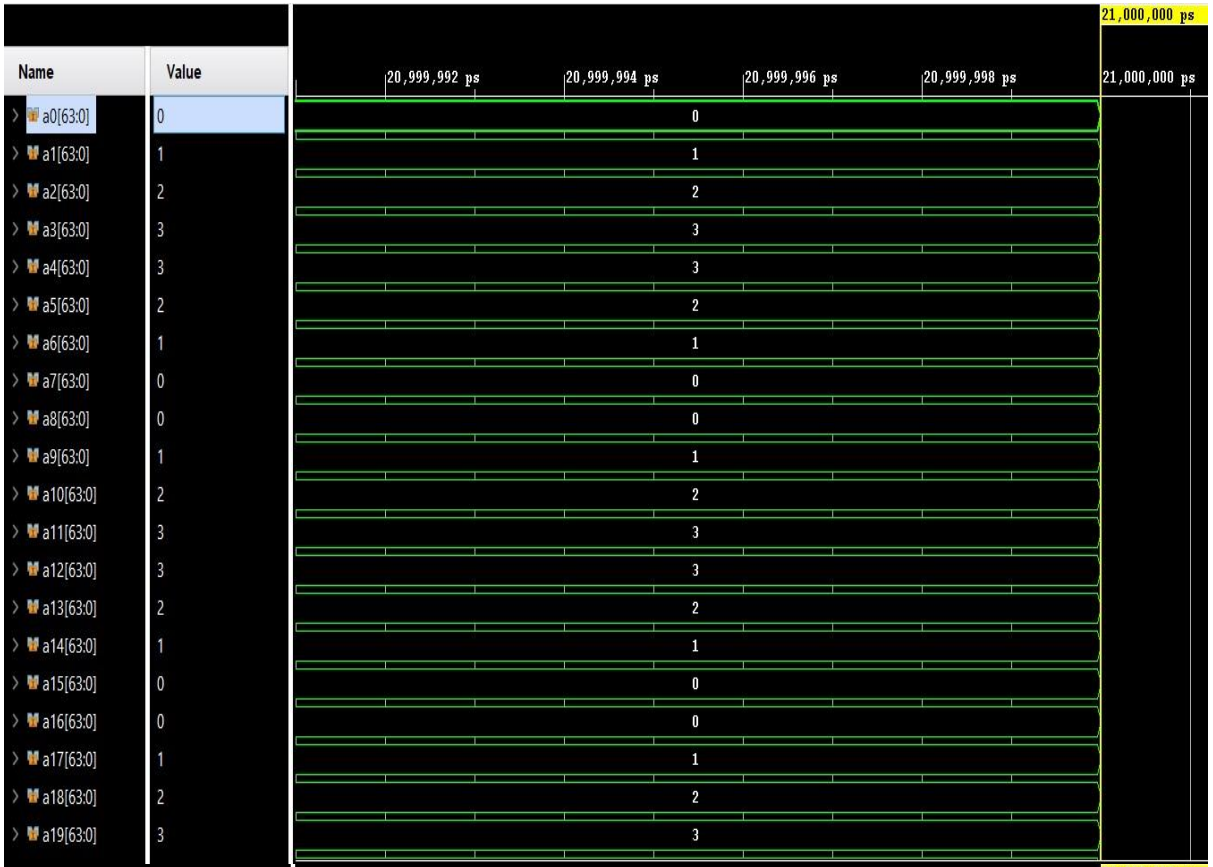


Fig 5.17.1 Schematic of proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2


Name
Value
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a0[63:0] 0
a1[63:0] 1
a2[63:0] 2
a3[63:0] 3
a4[63:0] 3
a5[63:0] 2
a6[63:0] 1
a7[63:0] 0
a8[63:0] 0
a9[63:0] 1
a10[63:0] 2
a11[63:0] 3
a12[63:0] 3
a13[63:0] 2
a14[63:0] 1
a15[63:0] 0
a16[63:0] 0
a17[63:0] 1
a18[63:0] 2
a19[63:0] 3

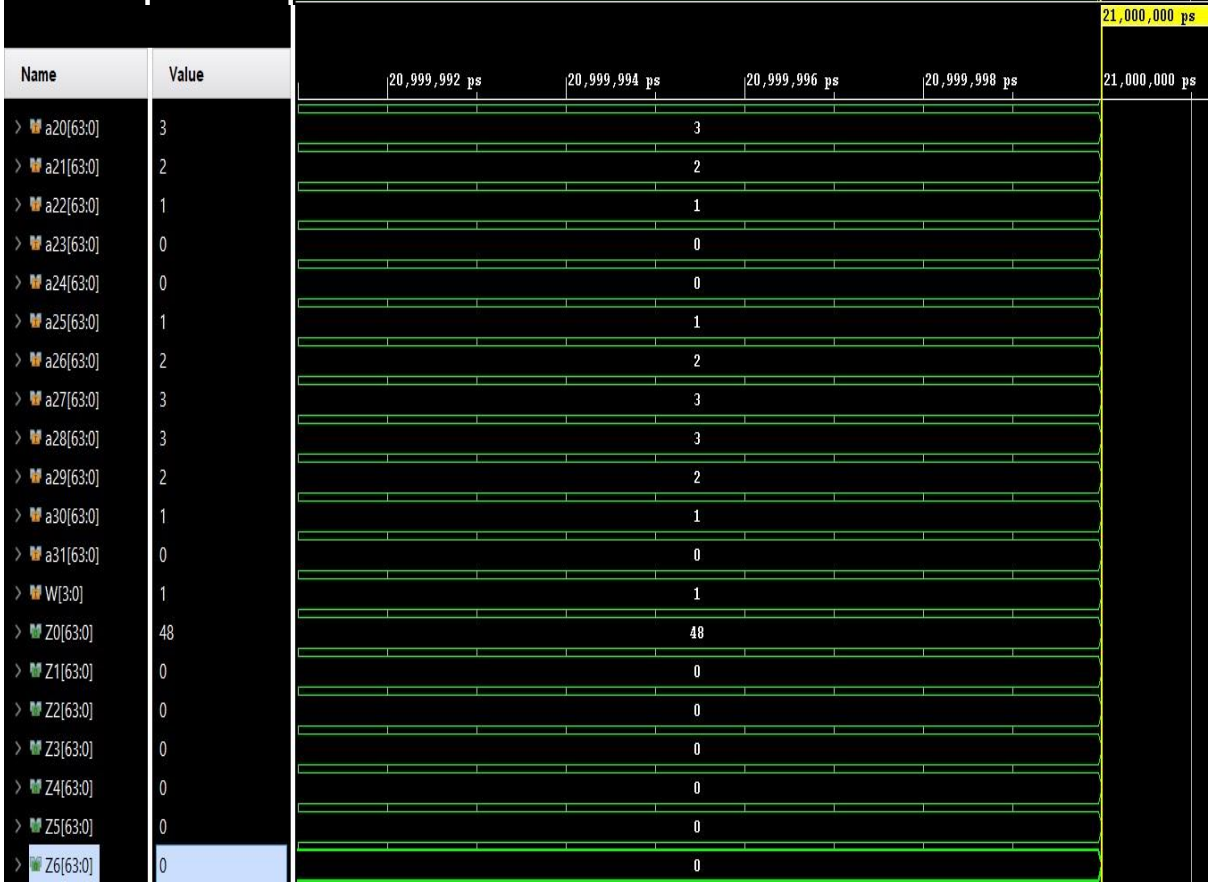

Name
Value
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a20[63:0] 3
a21[63:0] 2
a22[63:0] 1
a23[63:0] 0
a24[63:0] 0
a25[63:0] 1
a26[63:0] 2
a27[63:0] 3
a28[63:0] 3
a29[63:0] 2
a30[63:0] 1
a31[63:0] 0
W[3:0] 1
Z0[63:0] 48
Z1[63:0] 0
Z2[63:0] 0
Z3[63:0] 0
Z4[63:0] 0
Z5[63:0] 0
Z6[63:0] 0

Fig 5.17.2 Simulation results of proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
thirtytwo_pt	71402	180	4100
s1A (btf_1_0)	1380	0	0
s1B (btf_1_1)	1378	0	0
s1C (btf_1_2)	1380	0	0
s19 (btf_1)	2509	0	0
s21 (btf_2)	536	6	0
s22 (btf_2_3)	1268	6	0

Fig 5.17.3 Utilization report of proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	79	34	25	a31[9]	Z31[63]	41.306	26.281	15.026	∞	input port clock
Path 2	∞	79	34	25	a31[9]	Z31[62]	41.286	26.360	14.927	∞	input port clock
Path 3	∞	79	34	25	a31[9]	Z31[61]	41.212	26.284	14.929	∞	input port clock
Path 4	∞	78	34	25	a31[9]	Z31[60]	41.078	26.034	15.045	∞	input port clock
Path 5	∞	78	35	25	a31[9]	Z31[59]	41.009	25.993	15.017	∞	input port clock
Path 6	∞	78	35	25	a31[9]	Z31[58]	40.989	26.072	14.918	∞	input port clock
Path 7	∞	78	35	25	a31[9]	Z31[57]	40.915	25.996	14.920	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	16110	w[0]	Z16[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	16110	w[0]	Z16[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	16110	w[0]	Z16[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	16110	w[0]	Z16[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	16110	w[0]	Z16[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	16110	w[0]	Z16[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	16110	w[0]	Z16[15]	1.905	1.413	0.492	-∞	input port clock

Fig 5.17.4 Delay report of proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2

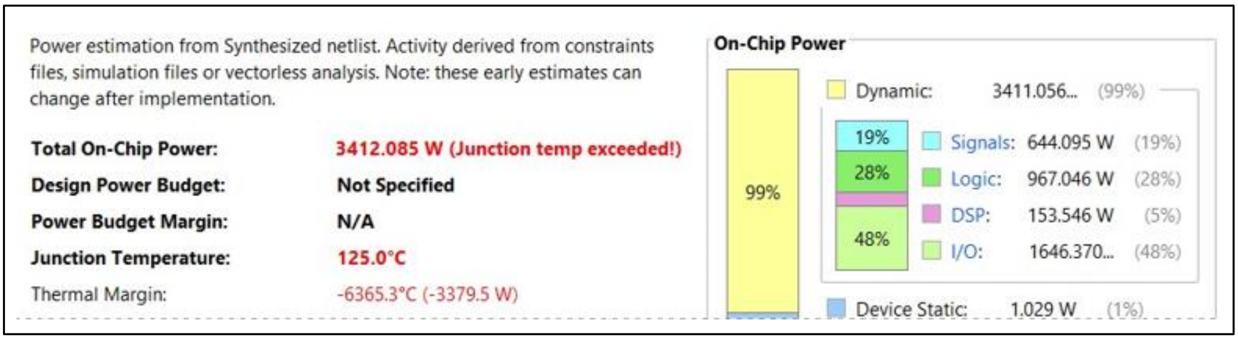


Fig 5.17.5 Power report of proposed MUX based 32-point 64-bit FFT in Xilinx Vivado 2022.2

5.18 PROPOSED SIXTY-FOUR POINT 64-BIT FFT (MUX BASED)

On simulating the proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2 with the inputs:

a(n) = {0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0, 0, 1, 2, 3, 3, 2, 1, 0}, twiddle factors W_N^{kn} = 1 and for select line = 3 the outputs obtained are Y(n) = {96, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -64, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

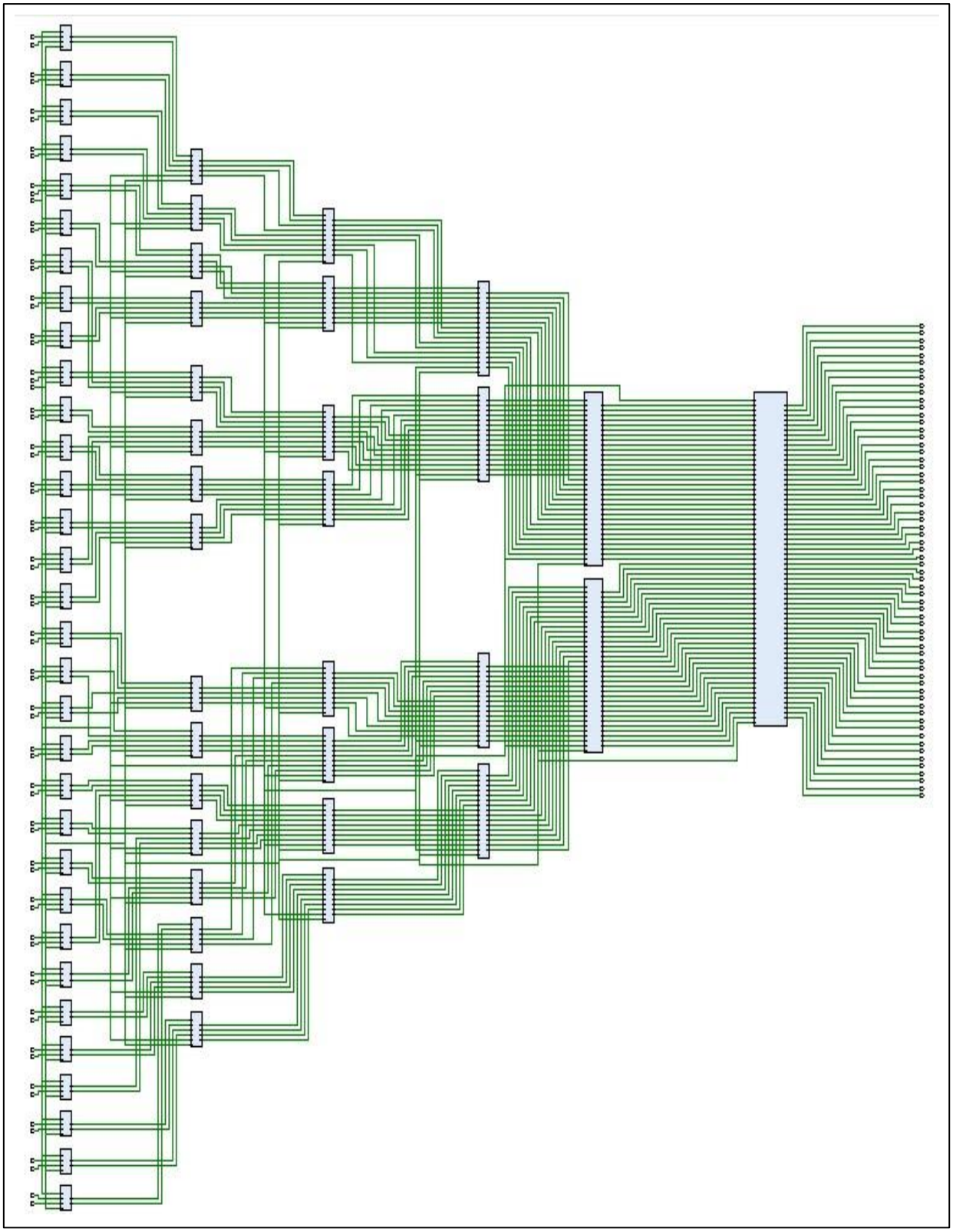
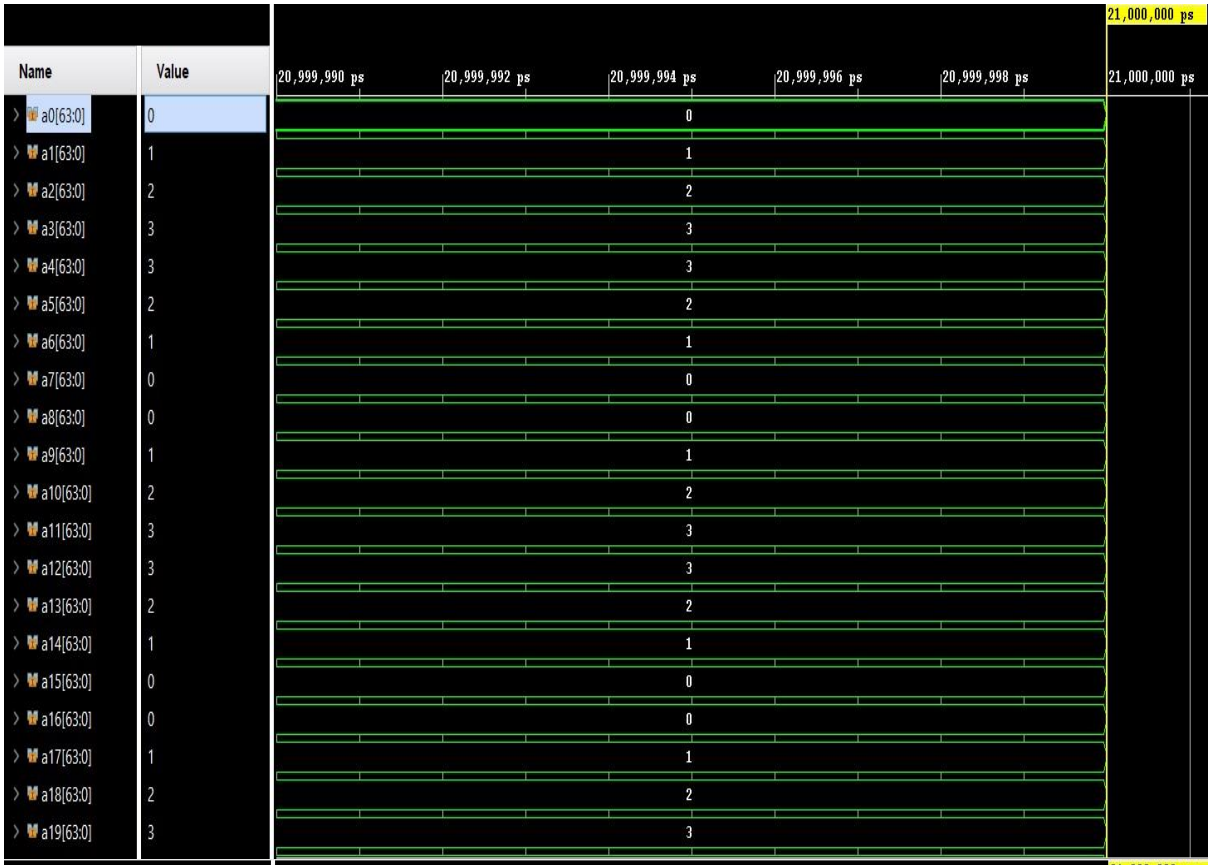
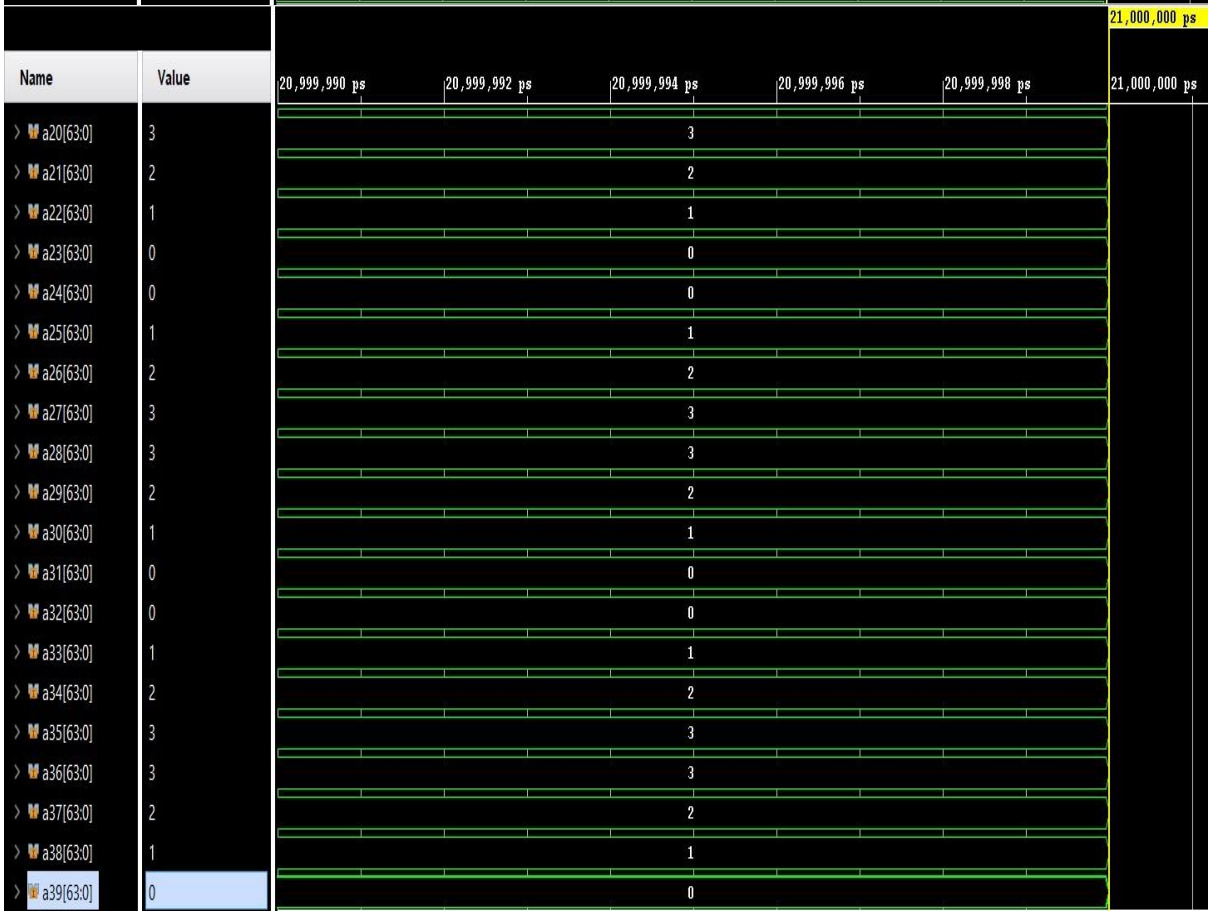
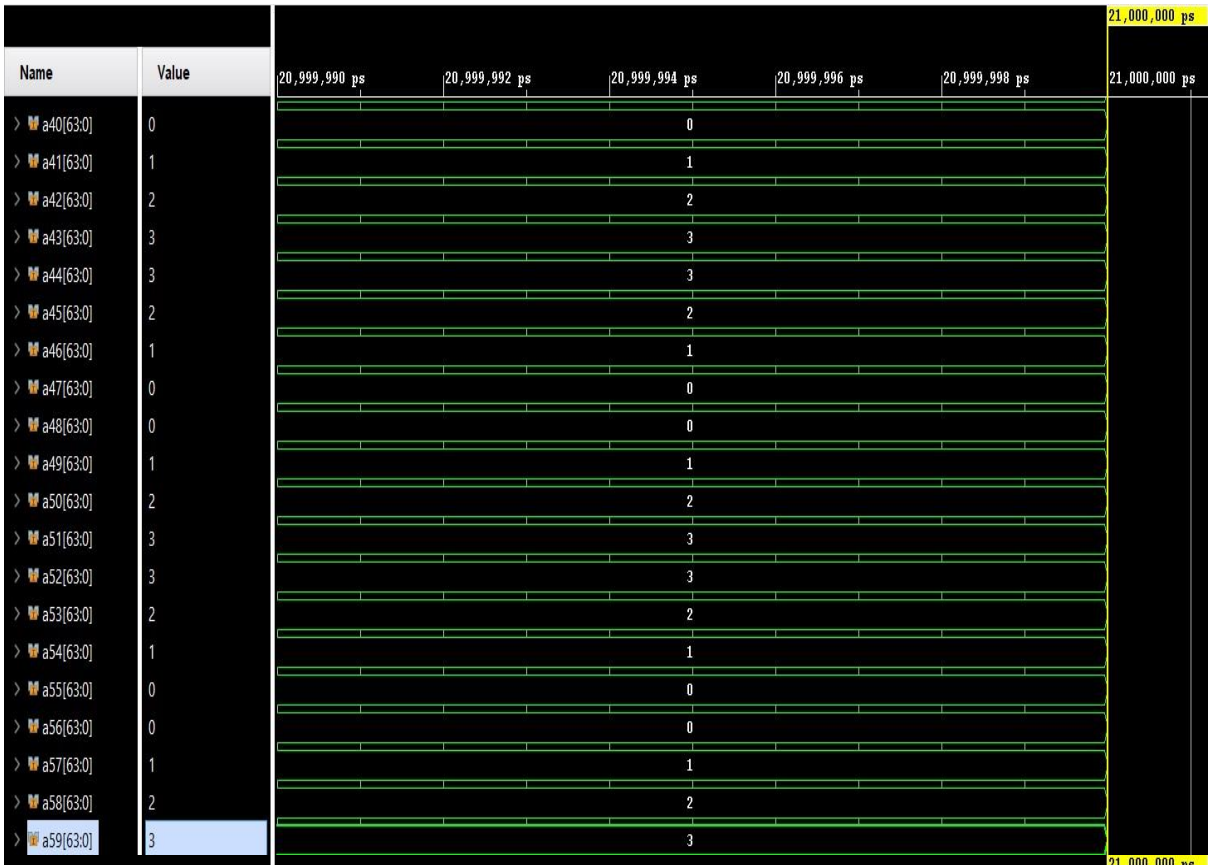
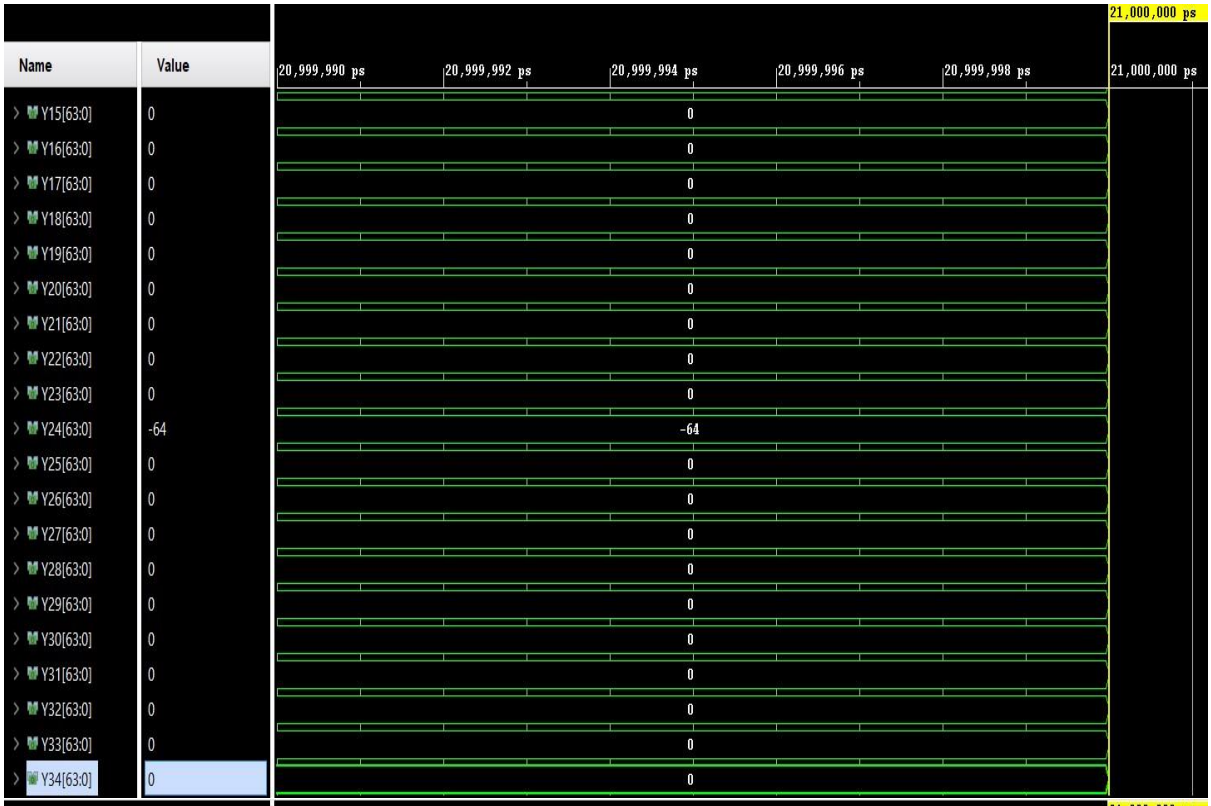


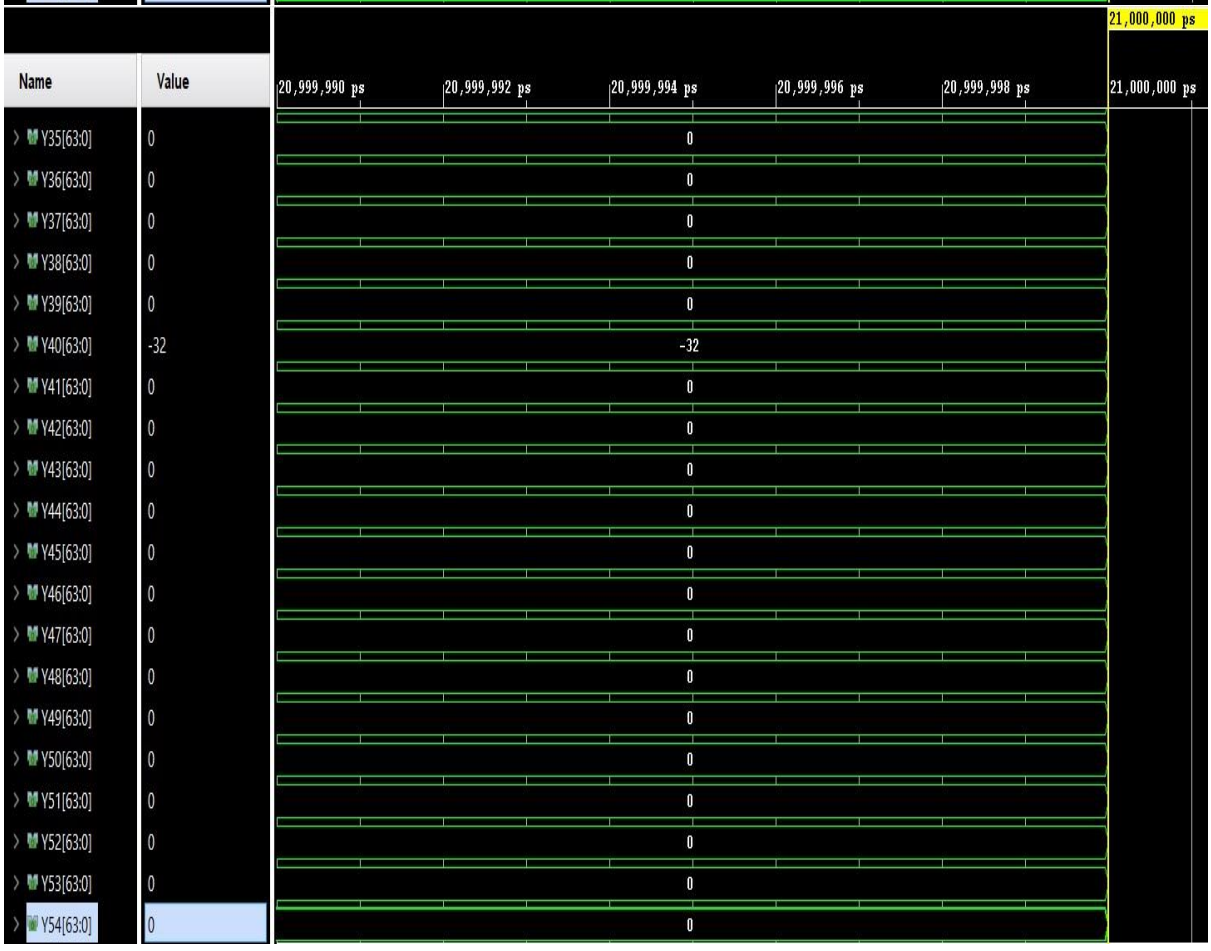
Fig 5.18.1 Schematic of proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a0[63:0] 0
a1[63:0] 1
a2[63:0] 2
a3[63:0] 3
a4[63:0] 3
a5[63:0] 2
a6[63:0] 1
a7[63:0] 0
a8[63:0] 0
a9[63:0] 1
a10[63:0] 2
a11[63:0] 3
a12[63:0] 3
a13[63:0] 2
a14[63:0] 1
a15[63:0] 0
a16[63:0] 0
a17[63:0] 1
a18[63:0] 2
a19[63:0] 3


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a20[63:0] 3
a21[63:0] 2
a22[63:0] 1
a23[63:0] 0
a24[63:0] 0
a25[63:0] 1
a26[63:0] 2
a27[63:0] 3
a28[63:0] 3
a29[63:0] 2
a30[63:0] 1
a31[63:0] 0
a32[63:0] 0
a33[63:0] 1
a34[63:0] 2
a35[63:0] 3
a36[63:0] 3
a37[63:0] 2
a38[63:0] 1
a39[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
a40[63:0] 0
a41[63:0] 1
a42[63:0] 2
a43[63:0] 3
a44[63:0] 3
a45[63:0] 2
a46[63:0] 1
a47[63:0] 0
a48[63:0] 0
a49[63:0] 1
a50[63:0] 2
a51[63:0] 3
a52[63:0] 3
a53[63:0] 2
a54[63:0] 1
a55[63:0] 0
a56[63:0] 0
a57[63:0] 1
a58[63:0] 2
a59[63:0] 3
a60[63:0] 3
a61[63:0] 2
a62[63:0] 1
a63[63:0] 0
W[3:0] 1
Y0[63:0] 96
Y1[63:0] 0
Y2[63:0] 0
Y3[63:0] 0
Y4[63:0] 0
Y5[63:0] 0
Y6[63:0] 0
Y7[63:0] 0
Y8[63:0] 0
Y9[63:0] 0
Y10[63:0] 0
Y11[63:0] 0
Y12[63:0] 0
Y13[63:0] 0
Y14[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Y15[63:0] 0
Y16[63:0] 0
Y17[63:0] 0
Y18[63:0] 0
Y19[63:0] 0
Y20[63:0] 0
Y21[63:0] 0
Y22[63:0] 0
Y23[63:0] 0
Y24[63:0] -64
Y25[63:0] 0
Y26[63:0] 0
Y27[63:0] 0
Y28[63:0] 0
Y29[63:0] 0
Y30[63:0] 0
Y31[63:0] 0
Y32[63:0] 0
Y33[63:0] 0
Y34[63:0] 0


Name
Value
20,999,990 ps
20,999,992 ps
20,999,994 ps
20,999,996 ps
20,999,998 ps
21,000,000 ps
Y35[63:0] 0
Y36[63:0] 0
Y37[63:0] 0
Y38[63:0] 0
Y39[63:0] 0
Y40[63:0] -32
Y41[63:0] 0
Y42[63:0] 0
Y43[63:0] 0
Y44[63:0] 0
Y45[63:0] 0
Y46[63:0] 0
Y47[63:0] 0
Y48[63:0] 0
Y49[63:0] 0
Y50[63:0] 0
Y51[63:0] 0
Y52[63:0] 0
Y53[63:0] 0
Y54[63:0] 0

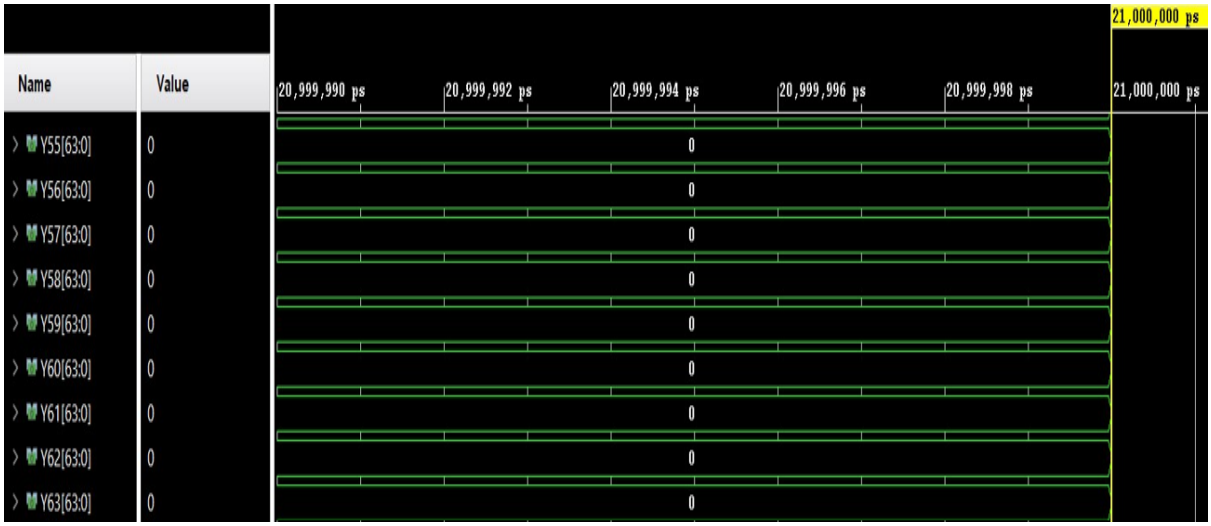


Fig 5.18.2 Simulation results of proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2

Further, after the synthesis of proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2, reports such as Utilization, Delay, Power are generated and the values are noted.

Hierarchy

Name	Slice LUTs (41000)	DSPs (240)	Bonded IOB (300)
mux_based	235038	180	8196
s1A (btf_a_0)	1404	0	0
s1B (btf_a_1)	1094	0	0
s1C (btf_a_2)	1690	0	0
s1D (btf_a_3)	1587	0	0
s1H (btf_a_4)	1221	0	0
s1I (btf_a_5)	1404	0	0

Fig 5.18.3 Utilization report of proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	93	45	41	a63[0]	Y63[63]	44.703	26.728	17.976	∞	input port clock
Path 2	∞	93	45	41	a63[0]	Y63[62]	44.683	26.807	17.877	∞	input port clock
Path 3	∞	93	45	41	a63[0]	Y63[61]	44.609	26.731	17.879	∞	input port clock
Path 4	∞	92	45	41	a63[0]	Y63[60]	44.604	26.492	18.113	∞	input port clock
Path 5	∞	92	45	41	a63[0]	Y63[59]	44.552	26.459	18.094	∞	input port clock
Path 6	∞	92	45	41	a63[0]	Y63[58]	44.532	26.538	17.995	∞	input port clock
Path 7	∞	92	45	41	a63[0]	Y63[57]	44.458	26.462	17.997	∞	input port clock

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	4	37800	w[0]	Y32[0]	1.905	1.413	0.492	-∞	input port clock
Path 12	∞	3	4	37800	w[0]	Y32[10]	1.905	1.413	0.492	-∞	input port clock
Path 13	∞	3	4	37800	w[0]	Y32[11]	1.905	1.413	0.492	-∞	input port clock
Path 14	∞	3	4	37800	w[0]	Y32[12]	1.905	1.413	0.492	-∞	input port clock
Path 15	∞	3	4	37800	w[0]	Y32[13]	1.905	1.413	0.492	-∞	input port clock
Path 16	∞	3	4	37800	w[0]	Y32[14]	1.905	1.413	0.492	-∞	input port clock
Path 17	∞	3	4	37800	w[0]	Y32[15]	1.905	1.413	0.492	-∞	input port clock

Fig 5.18.4 Delay report of proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2

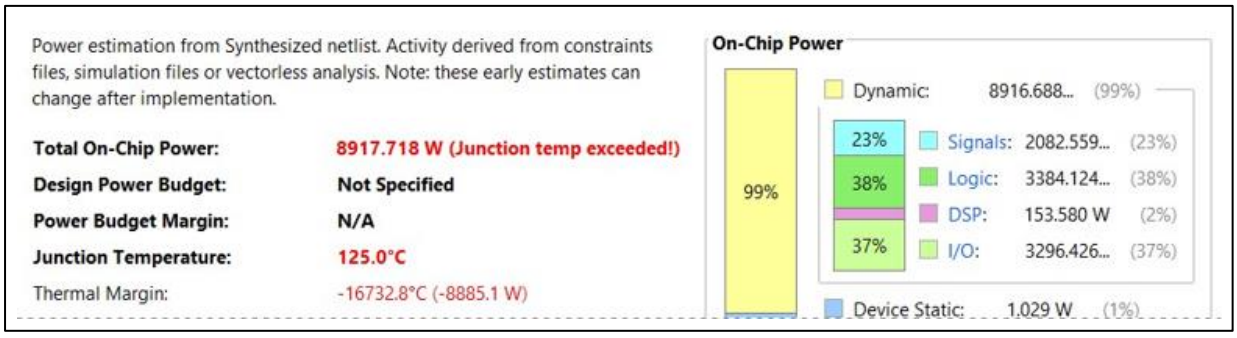


Fig 5.18.5 Power report of proposed MUX based 64-point 64-bit FFT in Xilinx Vivado 2022.2

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

In conclusion, our project is aimed to design high-speed DSP applications as they are essential for many modern technologies, including telecommunications, multimedia, medical imaging, and more. Their ability to process signals in real-time, handle high-frequency signals, and perform complex mathematical operations make them a critical component of many advanced systems.

However, modern DSPs commonly support FFTs with 16-bit or 32-bit precision with multipliers. So, the next higher bit 64-bit is necessary for High-speed applications. We have successfully developed two multipliers less methods that supports FFT with 64-bit precision. The two proposed methods are LUT based FFT structure and MUX based FFT structure. Among two LUT based FFT structures gives more optimizing results when compared to the other method. As a result, the parameters power and delay obtained for proposed 64-point 64 bit using LUT method based FFT are 11.03% less and 15.52% less compared to existing 64-point 64 bit conventional FFT method in Xilinx Vivado 2022.2 tool.

A 64-bit FFT structure using LUTs (Look-Up Tables) is a viable approach for high-speed DSP applications as the structure contains Reduced complexity, High accuracy, Faster processing, scalability, and Reduced power consumption. Overall, these advantages make a 64-point 64-bit FFT structure using LUTs a viable approach for high-speed DSP applications that require high accuracy and low latency.

FUTURE SCOPE

The future scope for 64-bit FFT structures is quite promising, as the demand for high-speed digital signal processing continues to grow in various fields such as telecommunications, image processing, wireless communication systems (Wi-Fi, LTE and 5G) and so on which require high accuracy and low error rates. One of the significant advantages of using 64-bit FFT structures is their ability to handle larger amount of data with higher precision. Furthermore, with the emergence of new technologies such as Machine Learning and Artificial Intelligence, there is a growing need for fast and efficient FFT implementations for tasks such as image and audio recognition, natural language processing and more.

CHAPTER 7

PROJECT MANAGEMENT

Since it is a simulation-based project a commercial PC and Xilinx Vivado 2022 tool are used.

Time Management

The total duration of the project work is 15 weeks with 30 hours spent each week. This entire duration is divided in to four phases with each phase taking Four weeks. The workdone in each phase is shown in the below table

Table 7 Time Management of project to build up the system

Phases	Work done
Phase 1	**Literature Survey:** Spent on searching the area on which the project is to be done. Decided to do on DA based FFT processors.
Phase 2	**Problem Identification and Study:** Searched for reference papers and found a problem. The next three days was spent on analyzing the problem & studied the theoretical aspects of the project.
Phase 3	**Implementation of the proposed work:** Spent on analyzing the algorithms to be used for the best results and the programming tools for the development of code. The project is broken down into small sub sections such that each section was programmed and tested.
Phase 4	**Completion of project and Manuscript submission:** Spent on the analyzing the results and arriving at conclusions. The rest of the time in this phase is spent on Manuscript submissions and final analysis of the work.

REFERENCES

[1] D. -C. Oh and Y. -H. Lee, *"Low Complexity FFT Based Spectrum Sensing in Bluetooth System," VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, 2009, pp. 1-5, doi: 10.1109/VETECS.2009.5073654.

[2] S. M. Cho, P. K. Meher, L. T. Nhat Trung, H. J. Cho and S. Y. Park, "*Design of Very High-Speed Pipeline FIR Filter Through Precise Critical Path Analysis,"* in *IEEE Access*, vol. 9, pp. 34722-34735, 2021, doi: 10.1109/ACCESS.2021.3061759.

[3] S. H. Mirfarshbafan, S. Taner and C. Studer, "*SMUL-FFT: A Streaming Multiplierless Fast Fourier Transform,"* in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1715-1719, May 2021, doi: 10.1109/TCSII.2021.3064238.

[4] V. Kumar, M. Mukherjee, J. Lloret, Z. Ren and M. Kumari, "*A Joint Filter and Spectrum Shifting Architecture for Low Complexity Flexible UFMC in 5G,"* in *IEEE Transactions on Wireless Communications*, vol. 20, no. 10, pp. 6706-6714, Oct. 2021, doi: 10.1109/TWC.2021.3076039.

[5] J. Wang, S. Li and X. Li, "*Scheduling of Data Access for the Radix-2k FFT Processor Using Single-Port Memory,"* in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 7, pp. 1676-1689, July 2020, doi: 10.1109/TVLSI.2020.2992021.

[6] M. Garrido and P. Paz, "*Optimum MDC FFT Hardware Architectures in Terms of Delays and Multiplexers,"* in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 3, pp. 1003-1007, March 2021, doi: 10.1109/TCSII.2020.3022528.

[7] K. Bowlyn and S. Hounsinou, "*An Improved Distributed Multiplier-Less Approach for Radix-2 FFT,"* in *IEEE Letters of the Computer Society*, vol. 3, no. 2, pp. 54-57, 1 July-Dec. 2020, doi: 10.1109/LOCS.2020.3014354.

[8] Z. Li, H. Jia, Y. Zhang, T. Chen, L. Yuan and R. Vuduc, "*Automatic Generation of High- Performance FFT Kernels on Arm and X86 CPUs,"* in *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 8, pp. 1925-1941, 1 Aug. 2020, doi: 10.1109/TPDS.2020.2977629.

[9] T. Xu, A. Fumagalli and R. Hui, "*Efficient Real-Time Digital Subcarrier Cross-Connect (DSXC) Based on Distributed Arithmetic DSP Algorithm,"* in *Journal of Lightwave Technology*, vol. 38, no. 13, pp. 3495-3505, 1 July1, 2020, doi: 10.1109/JLT.2019.2937787.

[10] D. Xue, V. DeBrunner and L. S. DeBrunner, "*Hardware Implementation of Discrete Hirschman Transform Convolution Using Distributed Arithmetic*," 2019 53rd Asilomar Conference on Signals, Systems, and Computers, 2019, pp. 1587-1590, doi: 10.1109/IEEECONF44664.2019.9048809.

[11] S. M. Noor, E. John and M. Panday, "*Design and Implementation of an Ultralow-Energy FFT ASIC for Processing ECG in Cardiac Pacemakers,"* in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 4, pp. 983-987, April 2019, doi: 10.1109/TVLSI.2018.2883642.

[12] S. Liu and D. Liu, "*A High-Flexible Low-Latency Memory-Based FFT Processor for 4G, WLAN, and Future 5G," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 3, pp. 511-523, March 2019, doi: 10.1109/TVLSI.2018.2879675.

[13] X. -Y. Shih, H. -R. Chou and Y. -Q. Liu, "*VLSI Design and Implementation of Reconfigurable 46-Mode Combined-Radix-Based FFT Hardware Architecture for 3GPP-LTE Applications,"* in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 1, pp. 118-129, Jan. 2018, doi: 10.1109/TCSI.2017.2725338.*T*

[14] Á. Ordóñez, F. Argüello and D. B. Heras, "*GPU Accelerated FFT-Based Registration of Hyperspectral Scenes,"* in *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 10, no. 11, pp. 4869-4878, Nov. 2017, doi: 10.1109/JSTARS.2017.2734052.

[15] D. F. G. Coelho, R. J. Cintra, N. Rajapaksha, G. J. Mendis, A. Madanayake and V. S. Dimitrov, "*DFT Computation Using Gauss-Eisenstein Basis: FFT Algorithms and VLSI Architectures,"* in *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1442-1448, 1 Aug. 2017, doi: 10.1109/TC.2017.2677427.

[16] D. D. Chen, G. X. Yao, R. C. C. Cheung, D. Pao and Ç. K. Koç, "*Parameter Space for the Architecture of FFT-Based Montgomery Modular Multiplication,"* in *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 147-160, 1 Jan. 2016, doi: 10.1109/TC.2015.2417553.

SREE VIDYANIKETHAN ENGINEERING COLLEGE

(AUTONOMOUS)

Sree Sainath Nagar, A.Rangampet - 517 102

Department of Electronics and Communication Engineering

PROJECT TITLE: DESIGNING 64-BIT FFT STRUCTURES FOR HIGH-SPEED DSP APPLICATIONS

ABSTRACT: Well-established technology like Bluetooth is used to provide wireless connectivity from wireless headphones to mobile phone, laptop and mice of the wireless computer to a lot of other devices which need a short-range connectivity. Our spotlight is to reduce complexity for Bluetooth to find channels free from the use of other devices (in 2.4 GHz ISM band). Bluetooth uses 79 RF channels in this band, each 1 MHz wide. To reduce the implementation complexity, the PSD is calculated by means of Fast Fourier Transform (FFT) and linear interpolation by exploiting spectrum characteristics. A PSD is computed by multiplying each frequency bin in an FFT by its complex conjugate. All the complex multiplications required for this FFT are implemented using Distributed Arithmetic (DA) technique. Distributed Arithmetic (DA) takes the value in bit serial fashion which suits for signal processing applications. DA is a technology for multiplier-less implementation of inner dot product of two vectors. Several applications of FFT are medical imaging, signal processing and it is also used in digital formats such as JPEG, MP3, H.264. This project reveals a new integration approach for computing using DA. The proposed design will be implemented for 64-point Butterfly structure by incorporating DA technique. Finally, the results of the proposed Butterfly structure will be compared with the existing Butterfly structure of equal length in terms of power, area and delay parameters. The designs can be evaluated using Xilinx Vivado.

PROJECT BATCH: 22A04

GAJULA AMRUTHA SAI - 19121A0459

BESTA DIVYA SREE - 19121A0424

BHARADWAJ KARTHIK K - 19121A0426

BHANDOTH UDAY KIRAN NAIK - 19121A0425

GUIDE: **Ms. M. BHARATHI**, M.Tech., (Ph.D.)

Assistant Professor, Dept. of ECE.

POs Attained:

	Program Outcomes												Program Specific Outcomes			
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
DESIGNING 64-BIT FFT STRUCTURES FOR HIGH-SPEED DSP APPLICATIONS	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

Signature of the Supervisor

Appendix - I

Base Paper

An Improved Distributed Multiplier-Less Approach for Radix-2 FFT

Kevin Bowlyn , Member, IEEE

Sena Hounsinou, Member, IEEE

Abstract

Complex numbers play a vital role in the implementation of a wide number of Digital Signal Processing (DSP) algorithms. Since they are represented as two separate components (real and imaginary), the algorithms increased the number of computations. In this letter, we present an improved approach that reduces the number of computations by integrating the computation of the radix-2 Fast Fourier Transform (FFT) with the Distributed Arithmetic (DA) and Complex Binary Number System (CBNS). Further, the proposed architecture replaces multipliers with the use of adders and shifters, which considerably decrease the design area. Although more adders are needed to implement the new DA-CBNS proposed architecture, when compared to the traditional radix-2 FFT and DA or CBNS -based FFT algorithms, results show that the proposed design yields an increase in operating frequency, and a reduction in size, memory usage, and power consumption.

Purpose

In an effort to reduce size, weight, and power and still keep up with the performance requirements of today’s technologies, it is imperative to devise more efficient techniques, especially for com- mutationally intensive domains like DSP. In this domain, most algorithms rely on the implementation of the Fourier Series (FS) and Fourier Transform (FT), where complex number arithmetic plays an essential role. However, in these implementations, complex numbers are treated as two separate entities (real and imaginary), yielding dedicated

multipliers to process each entity. Although dedicated multipliers are fast, such an approach often results in an increase in computations, hardware, costs, and power consumption.

Methods

Computing the Radix-2 FFT algorithm using a DIT and DIF butter-fly structure that are similar requires two multiplications and a subtraction each for Real($X_2[n]*W^k_N$) and for Imag($X_2[n]*W^k_N$), respectively. In addition, $X_1[k]$ and $X_2[K+N/2]$ necessitate two additions and two subtractions, respectively. By integrating the DA-CBNS technique within the FFT algorithm, each butterfly structure will only require one complex multiplication, an addition, and two complex additions/subtractions. To accomplish this, each input of the butterfly structure must first be converted to (-1+j) CBNS base algorithm as discussed in Section 2.3. Next, the multiplication circuit must be designed using a multiplier-less DA implementation. Here, we considered ROM-based DA structures with no dedicated multiplier and low memory utilization such as ROM decomposition, adder transformation, and offset binary coding.

Results

The results show that when considering the power and design area between the traditional FFT structure and the proposed DA-CBNS FFT structure, the DA-CBNS design utilized considerably less memory resources (LUT), consumed the lowest amount of power, and required the least design area. Compared to its 16-bit-real/16-bit-imaginary traditional FFT counterpart, the 32-bit 8-point Radix-2 DA-CBNS FFT showed a 32 percent area reduction, 41 percent power reduction, 59 percent reduction in run-time, 42 percent reduction in LUT count, and 66 percent increase in speed.

Conclusion

In this work, we presented a DA-CBNS based multiplier-less approach for computing Radix-2 FFT. Although dedicated multi- pliers yield a much-needed fast design for DSP applications, they are also very costly components that consume a large design area. Instead, our proposed approach replaces the dedicated multipliers by successive shifting and addition operations.

Next, it uses less memory resources by pre-computing and incorporating the twiddle values into each stage of the butterfly. Finally, the DA-CBNS based design integrates a conversion of complex numbers components into a single entity, thereby allowing an efficient representation of the data. Although the number of additions operations increased as a result of these modifications, the proposed design shows an overall increase in performance, compared to FFT implementations using dedicated multipliers, DA-only or CBNS-only approaches. These findings make the design suitable for DSP applications such as imaging in the medical field where efficiency and performance are vital features.

Appendix - II

Conference Paper Certificate

12th IEEE International Conference on
Communication Systems and Network Technologies
(CSNT-2023)
Sponsored by: M.P. Council of Science & Technology, Bhopal
08-09 April 2023
Organised by:
Technocrats Institute of Technology (Excellence) Bhopal
Madhya Pradesh Section Flagship Conference
Estd. : 2007

IEEE MADHYA PRADESH SECTION

Certificate of Appreciation

This is to certify that

Prof./Dr./Ms./Mr Besta Divya Sree, M.Bharathi, G.Amrutha Sai, Bharadwaj Karthik K, B.Uday Kiran Naik

of Mohan Babu University Tirupati, Andhra Pradesh

has Participated/Presented/Contributed a research paper in 12th IEEE International Conference CSNT-2023 held at Technocrats Institute of Technology (Excellence) Bhopal, Madhya Pradesh (India).

The paper was presented on 09-04-2023 Paper ID: 1570885848 Paper Title:

'Designing 64-Bit LUT Based FFT Structure for High-Speed DSP Applications'

Prof. (Dr.) K.K. Dwivedi
General Chair, CSNT 2023

Prof. G.S. Tomar
Chairman, IEEE MP Section

BIODATA

NAME	**:**	GAJULA AMRUTHA SAI
FATHER NAME	**:**	G SHANKAR MOHAN
DATE OF BIRTH	**:**	20-04-2001
NATIONALITY	**:**	INDIAN

CONTACT DETAILS

Contact No.	**:**	9390534656
Email	**:**	amruthasai20401@gmail.com
Contact Address	**:**	12-2-490, Ashok Nagar, Near Vivekananda Park, Anantapur-515001.

NAME	**:**	BESTA DIVYA SREE
FATHER NAME	**:**	B SREEHARI
DATE OF BIRTH	**:**	13-01-2002
NATIONALITY	**:**	INDIAN

CONTACT DETAILS

Contact No.	**:**	7702317953
Email	**:**	divyasreebesta@gmail.com
Contact Address	**:**	1-5b, BC Colony, Perur(V), Ramagiri(M), Anantapur(D) 515621.

NAME : BHARADWAJ KARTHIK K

FATHER NAME : BALASUBRAMANYAM K

DATE OF BIRTH : 15-08-2002

NATIONALITY : INDIAN

CONTACT DETAILS

Contact No. : 6304803725

Email : bharadwajk01@gmail.com

Contact Address : 29-84/2, Gobbilla Kotur Road, Near Anjaneya Temple, Palamaner-517408.

NAME : B. UDAY KIRAN NAIK

FATHER NAME : B. GOPAL NAIK

DATE OF BIRTH : 09-04-2001

NATIONALITY : INDIAN

CONTACT DETAILS

Contact No. : 6281227150

Email : bhandothuday2001@gmail.com

Contact Address : 7-11-43, BJP Colony, Kakkalapalli Panchayat, Anantapur-515002.