A REPORT

ON

IMPLEMENTATION AND VERIFICATION OF CUSTOM FFT MODULE WITH PYQN FLOW

BY

LAKSHAY	**2023A3PS0173G**	**EEE**
HEMANT	**2023A3PS0292H**	**EEE**
BALA	**2023A8PS0424H**	**ENI**

Prepared in partial fulfillment of the

Practice School-I

AT

CSIR-CEERI

A Practice School-I Station of

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(June, 2025)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

(RAJASTHAN)

Practice School Division

Station: CSIR **Centre: CEERI**

Duration: 8 weeks **Date of Start: 26 May 2025**

Date of Submission: 16 July 2025

Title of the Project: IMPLEMENTATION AND VERIFICATION OF CUSTOM FFT MODULE WITH PYQN FLOW

2023A3PS0173G	**Lakshay**	**EEE**
2023A3PS0292H	**Hemant**	**EEE**
2023A8PS0424H	**Bala**	**ENI**

Name of Expert: Dr. Gaurav Purohit (Sr. Scientist)

Name of PS Faculty: Dr. Meetha.V. Shenoy

Dr. Hari Om Bansal

Keywords: Fast Fourier Transform (FFT), Cooley-Tukey Algorithm, Verilog HDL, FPGA Implementation, ZedBoard, Shakti Processor, AXI Interface, Hardware-Software Co-design, RTL Design

Project Areas: Digital Signal Processing, FPGA-Based System Design, Hardware-Software Co-design, Embedded Systems, VLSI Design

Abstract: This project presents the design and FPGA implementation of an 8-point Fast Fourier Transform (FFT) processor using the radix-2 Cooley-Tukey algorithm in Verilog HDL. The design was synthesized and verified on the ZedBoard using Vivado, ensuring efficient resource usage and timing closure. An AXI wrapper is being developed to interface the FFT core with the Processing System (PS), enabling verification via C code. This work serves as a foundation for future deployment on the Shakti RISC-V processor, demonstrating a complete hardware-software co-design approach.

Signature(s) of Student(s) **Signature of PS Faculty**

Date **Date**

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to all those who supported and guided me throughout the successful completion of this project.

First and foremost, I sincerely thank **Dr. Gaurav Purohit**, my project mentor, for his invaluable guidance, continuous encouragement, and insightful suggestions at every stage of this work. His expertise and mentorship were instrumental in shaping the direction and depth of this project.

I also extend my appreciation to my fellow team members, **Hemant** and **Bala**, for their collaboration, support, and active participation throughout the development and verification phases. Working alongside them has been an enriching experience filled with learning and problem-solving.

This work was carried out at **CSIR–Central Electronics Engineering Research Institute (CEERI)**, under the **Advanced Information Technology Group**. I am grateful to the institute for providing the necessary technical infrastructure, FPGA development platforms, and a research-oriented environment that enabled hands-on learning and innovation.

I would also like to thank the faculty and staff at CEERI for their timely assistance and for facilitating access to resources crucial for the project's success. Their support was key in overcoming both technical and logistical challenges.

Additionally, I acknowledge the open-source communities and technical forums that provided valuable documentation and references for Verilog design, FFT algorithms, and AXI protocol integration. These resources significantly contributed to the functional and implementation aspects of the project.

Lastly, I am deeply thankful to my family for their unwavering support, patience, and encouragement throughout this journey. Their belief in me has been a constant source of motivation.

This project has not only strengthened my understanding of digital design and embedded systems but also instilled a deeper appreciation for collaborative research and development.

TABLE OF CONTENTS

INTRODUCTION

The Fast Fourier Transform (FFT) is a foundational algorithm in digital signal processing (DSP) used to analyse frequency components of discrete signals. Efficient computation of the Discrete Fourier Transform (DFT) is essential in a wide range of applications, such as wireless communications, audio and video signal processing, image compression, biomedical signal analysis, and radar systems. Traditional computation of the DFT is computationally expensive, having time complexity of $O(N^2)$. The FFT, specifically the Cooley-Tukey algorithm, optimizes this process by reducing the complexity to O(NlogN), making real-time processing feasible on resource-constrained hardware.

This project aims to design and implement a synthesizable 8-point FFT module using the Cooley-Tukey algorithm in Verilog. The design targets FPGA platforms, specifically the ZedBoard (Xilinx Zynq-7000 SoC), and is also intended to be portable to the Shakti processor platform, a RISC-V based open-source initiative from India. The objective was not only to develop a correct and optimized FFT architecture but also to make it FPGA-synthesisable and integrable with software through AXI interfacing.

The ZedBoard offers a tightly-coupled processing system (PS) and programmable logic (PL), making it an ideal testbed for developing hardware accelerators that can be verified and controlled using software. By using Verilog HDL for design, the project ensures that low-level hardware optimizations are possible and can be mapped directly onto the FPGA fabric. Furthermore, integration through an AXI interface will allow the FFT module to be controlled from the PS using a C-based application. This forms the basis for a hardware-software co-design approach, enabling flexible validation and benchmarking.

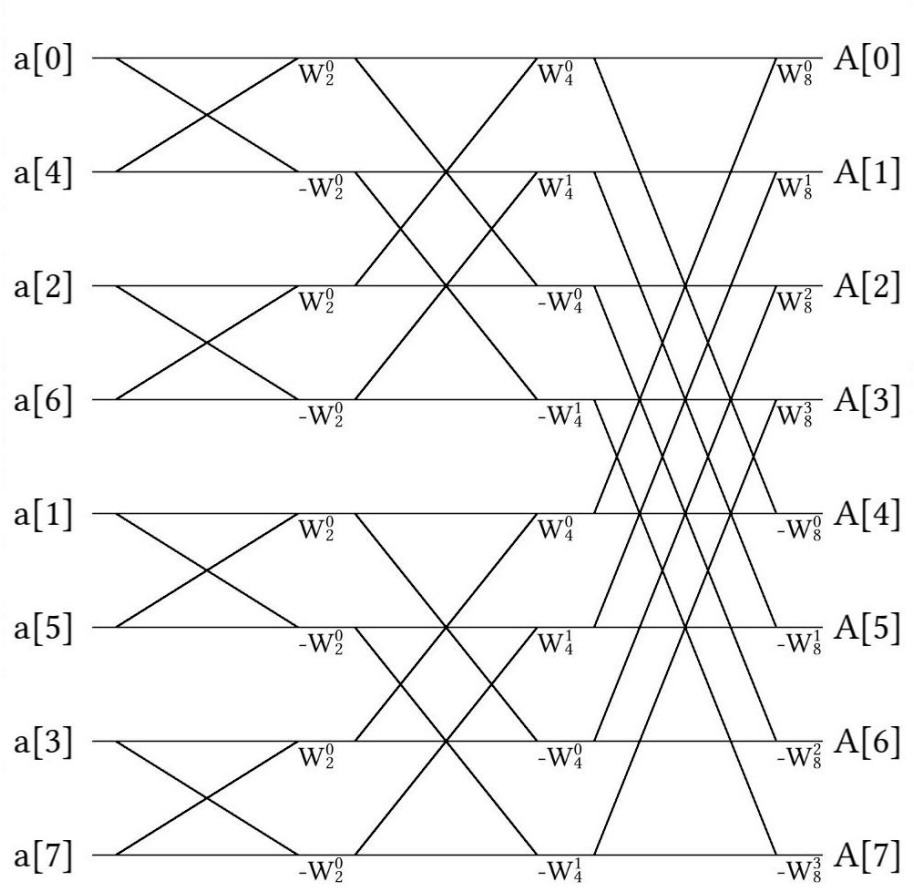
The project workflow involves several key phases: algorithm selection and analysis, RTL design and modularization, simulation and functional verification, synthesis and implementation using Vivado, and finally, system-level integration using AXI. Once the AXI wrapper is developed, test inputs can be provided by a C program running on the ARM Cortex-A9 processor of the ZedBoard, and FFT results can be retrieved and validated in software. This paves the way for further enhancements such as scaling to 16/32/64-point FFTs, implementing inverse FFTs, or extending compatibility to other platforms like the Shakti processor.

By completing this project, the goal is to gain hands-on experience in digital design, hardware acceleration, FPGA implementation, and hardware-software co-verification. The ability to offload computation-intensive DSP tasks to custom hardware accelerators can significantly boost system performance, reduce latency, and enhance power efficiency, making this project a valuable foundation for more complex embedded system developments.

MAIN TEXT

The core of this project lies in implementing the Cooley-Tukey radix-2 FFT algorithm using Verilog HDL. The Cooley-Tukey algorithm recursively divides an N-point DFT into two N/2 point DFTs, leveraging symmetry and periodicity properties of complex exponentials to reduce computation. For this project, an 8-point FFT was selected to simplify initial development and verification while providing a functional framework that can be scaled to larger transforms in future iterations.

The Verilog module accepts 8-bit real-valued input samples, which are fed into the system one at a time. Internally, the module computes the FFT using a staged pipeline of butterfly units. Each butterfly unit performs complex arithmetic operations, including addition, subtraction, and multiplication with twiddle factors (complex exponentials). The design is modular, with parameterized components for easy extension. Fixed-point arithmetic is used throughout to maintain synthesis compatibility and resource efficiency while approximating real-world DSP applications. Q1.15 format is used to store twiddle factors in the memory. All the calculation are performed on integers and the result we after FFT calculations is the nearest integer value.



- Figure showing the mathematical algorithm of Cooley Tukey Method

The twiddle factors defined are as follows:

Real_part = {32767, 23167, 0, -23167}

Imag_part = {0, -23167, -32767, -23167}

These are values corresponding to

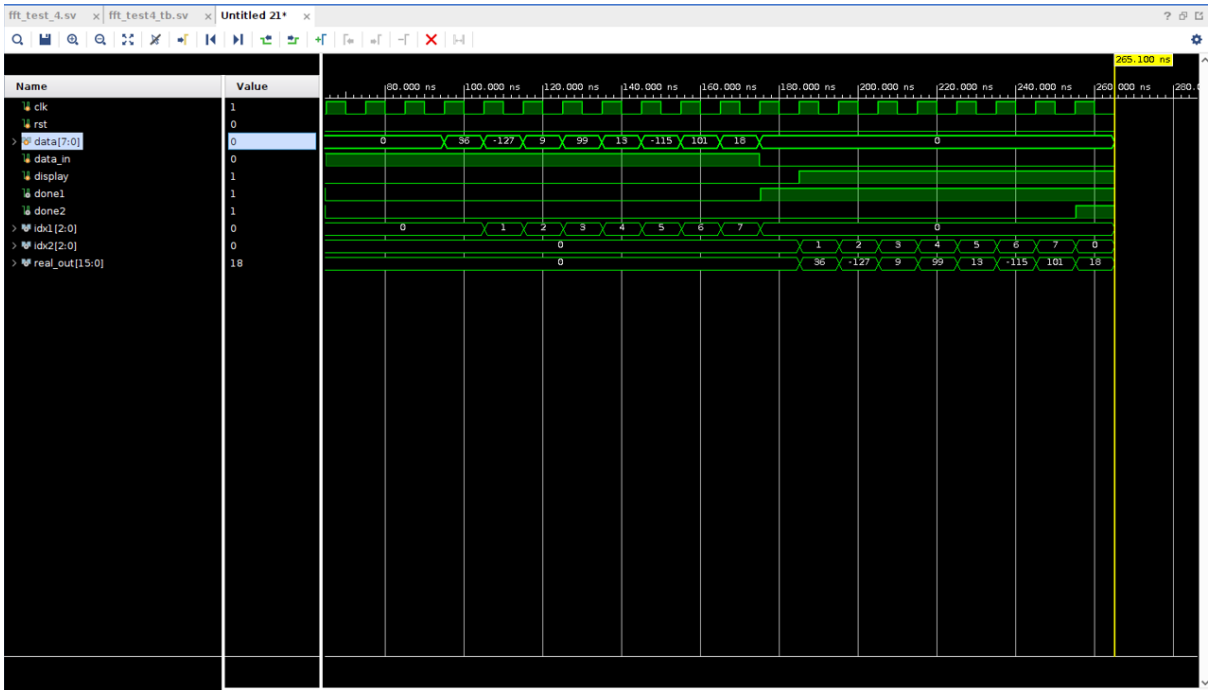
$$W_8^0 = e^0 = 1$$

$$W_8^1 = \frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}$$

$$W_8^2 = 0 - j \cdot 1 = -j$$

$$W_8^3 = -\frac{1}{\sqrt{2}} - j\frac{1}{\sqrt{2}}$$

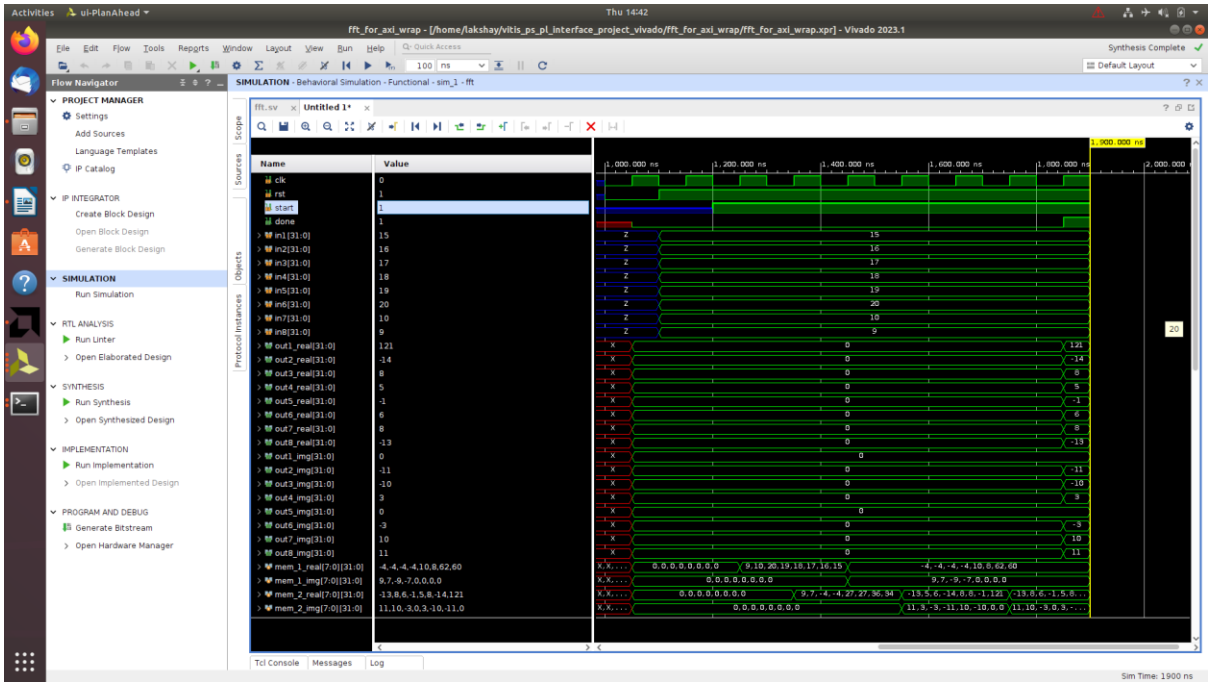
The module was first functionally verified using testbenches written in Verilog. First of all the memory model was written and verified. After its successful verification and implementation further code of the algorithm implementation was written. These testbenches applied known inputs and compared the outputs against reference FFT results calculated in MATLAB. The simulation confirmed the correctness of both the intermediate stages and the final FFT output. Once verified, the module was synthesized and implemented using Xilinx Vivado targeting the ZedBoard. Timing reports and resource utilization were analysed to ensure the design met performance and area constraints. The design successfully passed synthesis and place-and-route stages, confirming it as a synthesisable, deployable hardware core.



Verification result of the memory created.

Verification results of the FFT module along with the memory.

Then, after completing the entire code and simulating an verifying all the stages involved while calculation the FFT using Cooley Tukey algorithm, the following simulation results were achieved.



- Figure showing simulation results of the FFT IP

The next phase involves integrating the FFT module with the ARM processor on the Zynq SoC via an AXI interface. To achieve this, an AXI wrapper will be developed around the FFT core. This wrapper will manage control signals, data ports, and memory-mapped register access to allow the ARM processor to initiate FFT operations, provide input samples, and retrieve output data. A C program running on the ARM processor will serve as the user-level interface for testing and verification, communicating with the FFT core over the AXI bus.

Creating AXI Wrap of the FFT IP

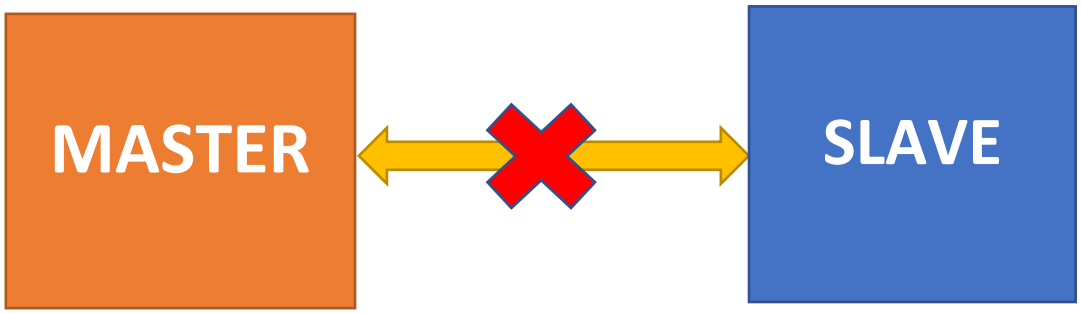
An **AXI wrap of an IP** refers to **encapsulating a custom or existing hardware IP (Intellectual Property) block with an AXI (Advanced eXtensible Interface) interface**, so that it can communicate with other components in a SoC (System on Chip), typically one that uses the AMBA AXI protocol — like ARM or Xilinx Zynq systems.

To facilitate processor-level control and integration with the Zynq SoC architecture, the custom-designed FFT IP core—implemented in Verilog using the 8-point Cooley-Tukey algorithm—was encapsulated within an AXI4-Lite wrapper. The AXI (Advanced eXtensible Interface) protocol is widely used in modern SoC designs for high-performance, memory-mapped communication between the processing system (PS) and programmable logic (PL). Wrapping the FFT IP with an AXI interface enabled seamless communication with the ARM Cortex-A9 processor on the ZedBoard platform.

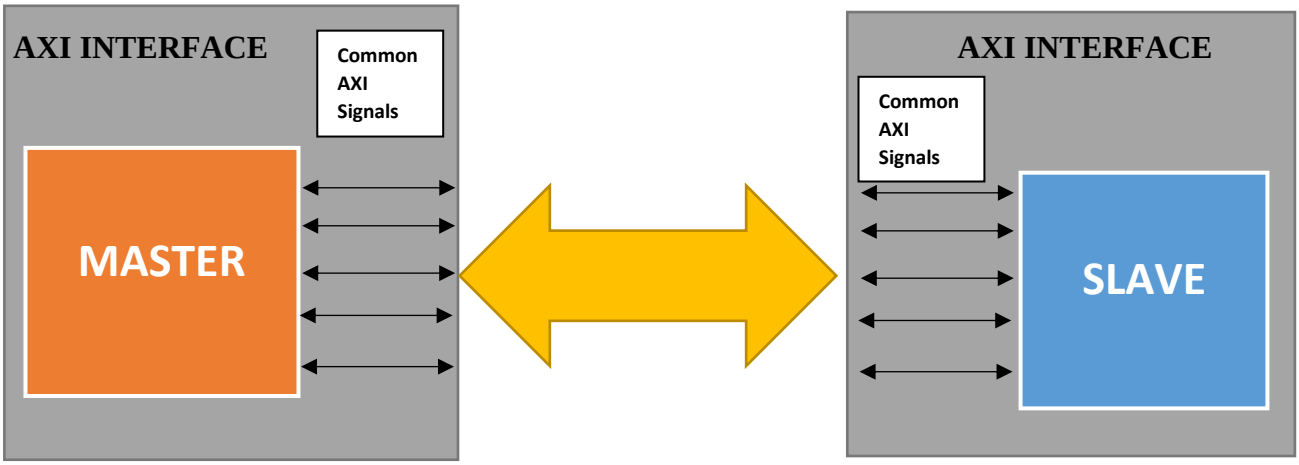
The AXI wrapper consists of addressable control and data registers mapped to memory space accessible from the software. Input values were written into these registers through AXI transactions initiated by the PS. Internally, the wrapper translated these AXI write transactions into signal-level inputs for the FFT module. Upon computation, the FFT output values were stored in output registers, which could then be read back by the processor using AXI read transactions.

This wrapping enabled high-level programmability, allowing the FFT IP to be verified and operated via a C program running in Vitis. Inputs were fed through a UART interface, and results were retrieved using standard memory-mapped I/O functions (`Xil_Out32`, `Xil_In32`). The AXI protocol's handshake mechanisms and well-defined timing model ensured robust communication and synchronization between hardware and software layers.

Overall, the AXI wrapping of the FFT IP allowed the module to be integrated into a larger embedded system and tested in a real-time environment, validating its correctness, flexibility, and reusability in modern SoC-based designs.



- No communication possible between master and slave without AXI wrapping.



- Communication established after AXI wrapping of both master and slave.

Different types of AXI4 Signals

1. AXI4 (Full AXI)

Supports: High-performance memory-mapped interfaces

- Burst-based transactions (up to 256 data transfers per burst)
- Separate read and write channels
- Out-of-order transaction support
- Ideal for high-throughput peripherals like memory controllers, DMA

2. AXI4-Lite

Supports: Simplified, low-bandwidth memory-mapped interface

- No burst transactions (only single data transfers)
- Lightweight and easier to implement
- Used for register-level access and control/status interfaces
- Commonly used to wrap custom IP cores (e.g., FFT, Adder)

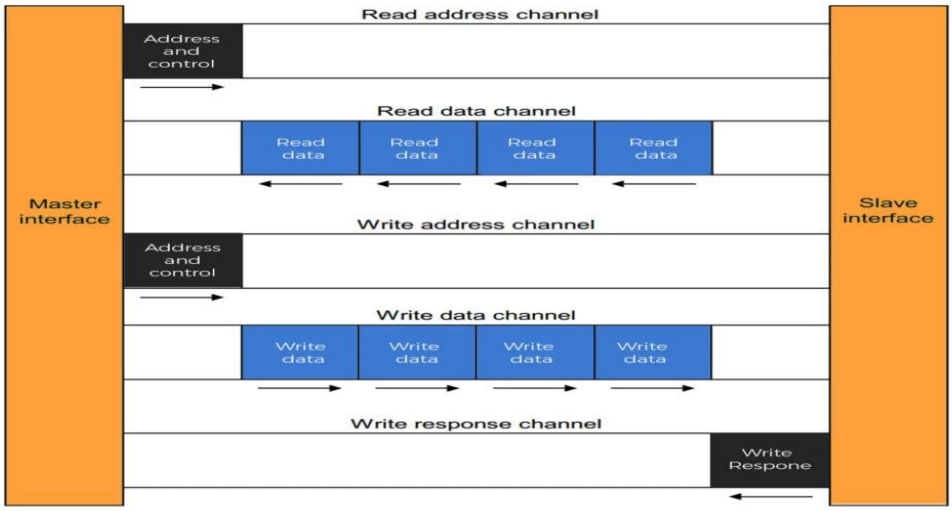
3. AXI4-Stream

Supports: High-speed unidirectional streaming data

- No address channel (data only)
- Continuous data flow with handshaking (`TVALID`, `TREADY`)
- Used for video, audio, FFT streams, etc.

Understanding AXI4-LITE Signals

AXI4-Lite is a simplified version of the AXI4 protocol, designed for low-bandwidth, memory-mapped control interfaces. It supports only single 32-bit read and write transactions without burst capabilities, making it ideal for accessing control and status registers in custom IPs. In this project, AXI4-Lite was used to wrap the FFT IP core, enabling easy integration with the Zynq processing system. Through memory-mapped registers, input data could be written and output data read using software running on the ARM processor. Its simplicity and low resource usage make AXI4-Lite highly suitable for lightweight peripheral communication within modern SoC architectures.



- Figure showing the channels involved in AXI4 LITE interface

Creating a finite state machine (FSM) for AXI4-Lite involves carefully analysing the protocol's five independent channels: write address (AW), write data (W), write response (B), read address (AR), and read data (R). Each channel uses a handshake mechanism with `VALID` and `READY` signals to manage data flow and ensure synchronized communication. The FSM must coordinate these handshakes to handle both read and write operations correctly.

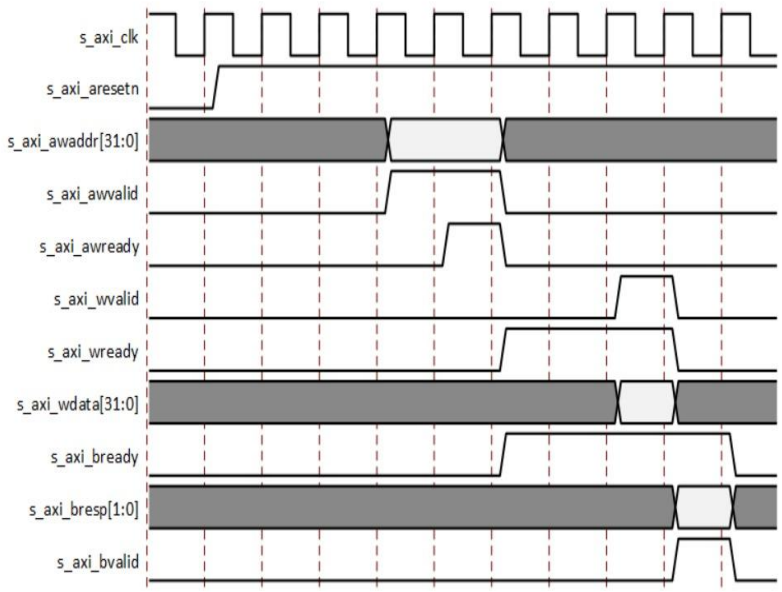
For **write transactions**, the FSM first waits for both `AWVALID` and `WVALID` to be asserted by the master, indicating valid address and data. It then asserts `AWREADY` and `WREADY` to accept them. Once both address and data are latched, the FSM transitions to a response state where it asserts `BVALID`, signalling that the write is complete. It waits for `BREADY` from the master to complete the handshake and return to idle.

For **read transactions**, the FSM waits for `ARVALID`, then asserts `ARREADY` to accept the address. Once the address is latched and data is ready, the FSM asserts `RVALID` and places the data on the `RDATA` bus. The transaction completes when the master asserts `RREADY`.

This FSM ensures proper sequencing, avoids deadlocks, and guarantees AXI4-Lite protocol compliance while enabling reliable integration with system-on-chip designs.

Write Signals

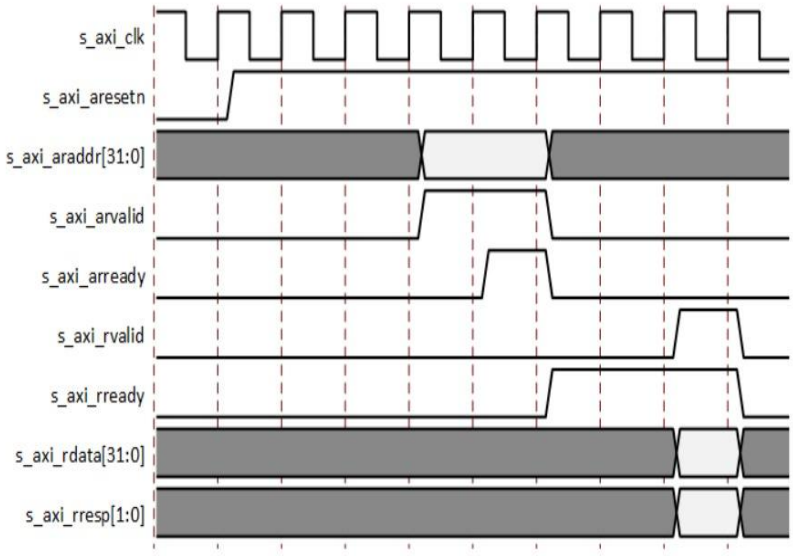
Figure: AXI4-Lite Write Timing Diagram



- Figure showing all the signals involved during write operation to slave

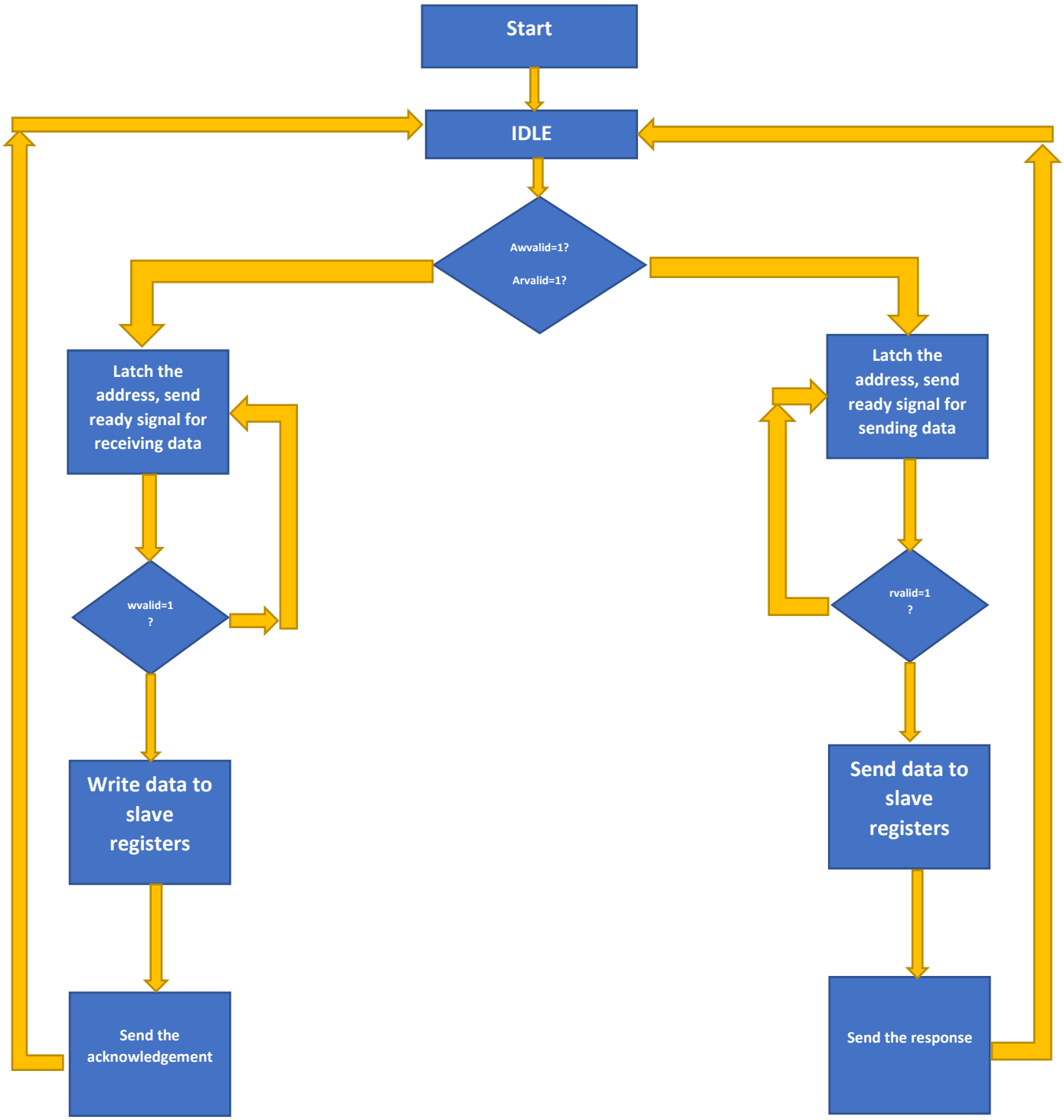
Read Signals

Figure: AXI4-Lite Read Timing Diagram

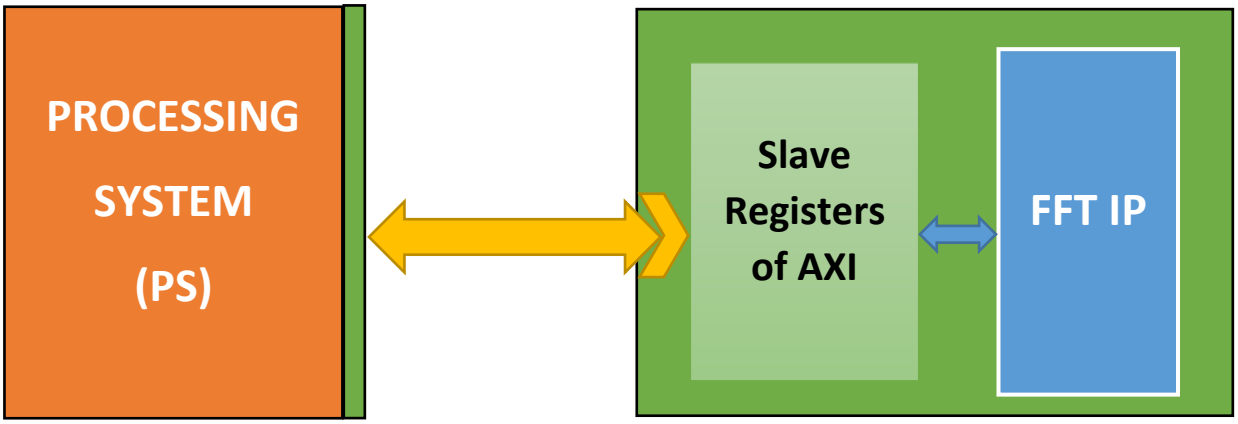


- Figure showing all signals involved during read operation

FSM for AXI4 LITE

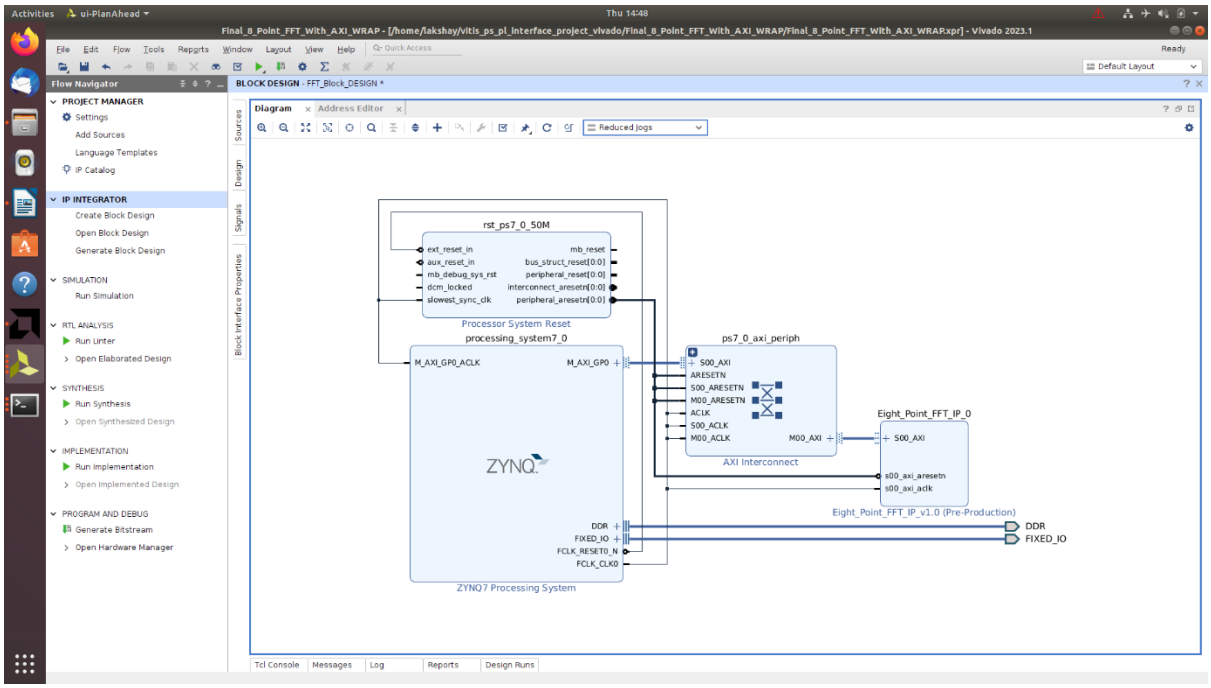


Creating AXI Wrap for FFT IP



Creating AXI wrap of the FFT IP allows us to write data directly to the 26 slave registers, which are defined by us in the slave AXI interface. The data is then taken by the IP from the slave registers as input and further processing is done until the outputs are generated. After the generation of the outputs, data is put back to the slave registers which is then read by the Processing System through the read cycle.

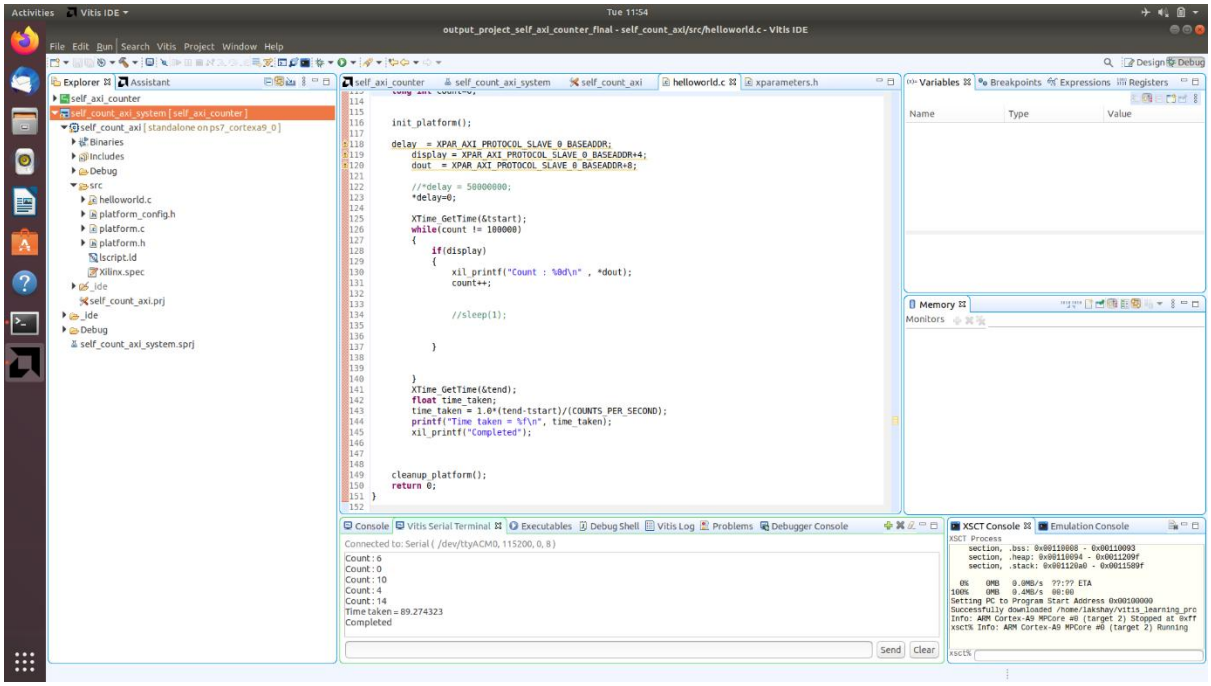
Thus, doing the above process in Vivado and then adding the ZYNQ Processing system to generate the block diagram gives the result shown below.



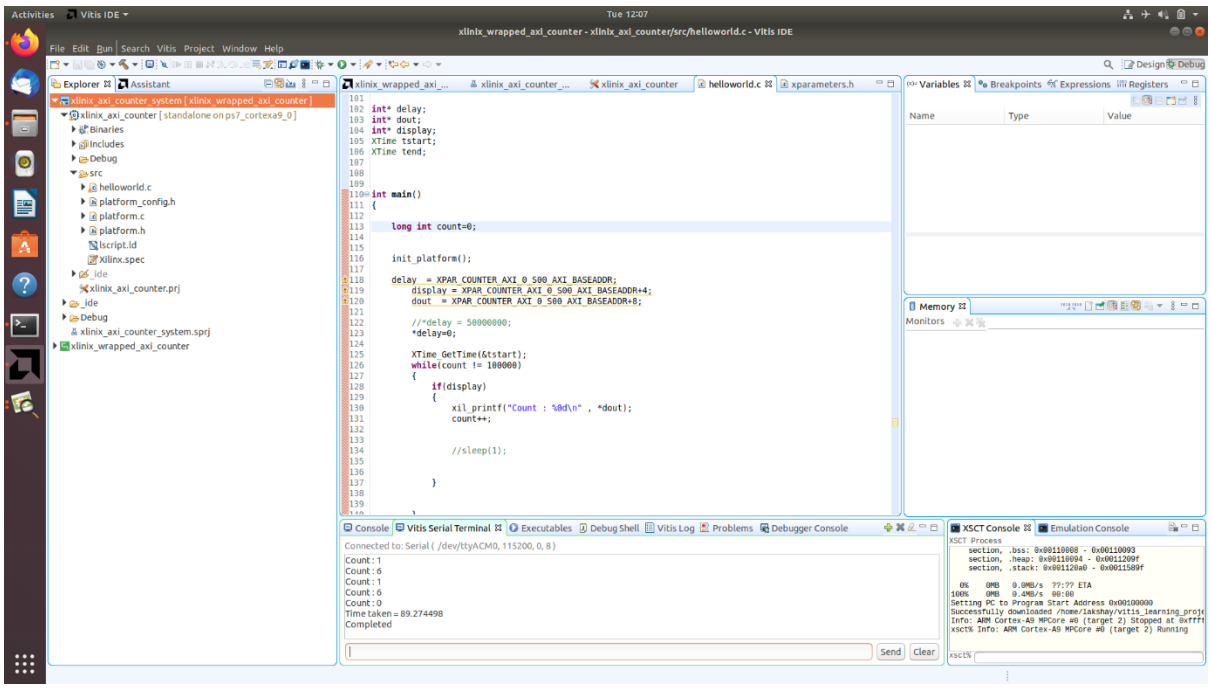
- Figure showing the block diagram of FFT IP integrated along with ZYNQ PS

The block design is then used to create an HDL wrapper. This HDL wrapper is then used as the Verilog code to generate bitstream of the entire environment. The environment is then exported as a .XSA file and imported in the Vitis software for developing testing programs on the PS side.

Also, an experiment was done to compare the results of the execution time achieved by using pre-defined Xilinx AXI wrapper with the customised made AXI wrapper. A difference was seen in the fourth decimal digit after a counter was run for 100000 counts using both the wrappers.

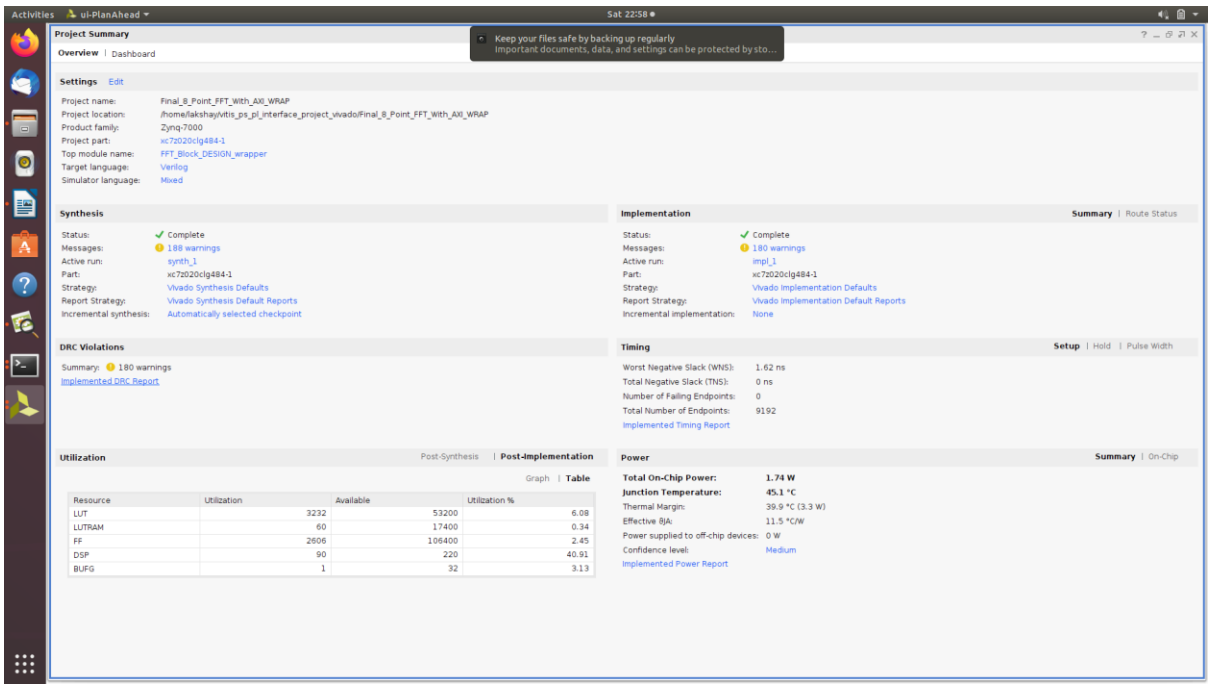


- Figure showing time taken by self AXI wrapped counter and Xilinx wrapped counter to perform 100000 counts.



Thus, it can be clearly seen that our self-made wrapper works faster than Xilinx wrapper and also reduces the number of code lines to do the same task.

On implementing the system using Vivado, and achieving successful timing closure the following summary results were attained.

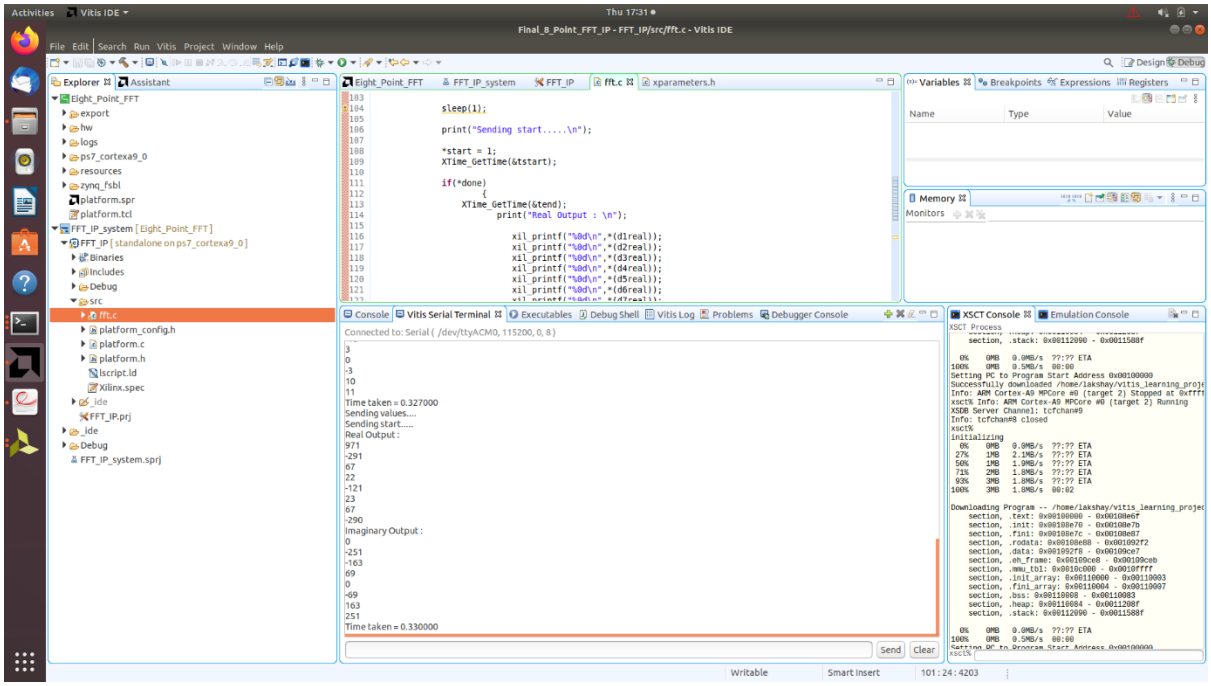


- Figure showing the summary results of the FFT IP

On final implementation of the FFT IP on ZEDBOARD and verifying its functionality by writing the required C code in Vitis, an execution time of 3.2 microseconds per FFT operation was achieved. The architecture made is scalable for 16 and 64 points FFT with slight alteration in the RTL design.

Alongside this, with the PYQN flow, it is aimed to the entire Verilog synthesisable code for the given FFT module by python scripting. This will allow to create complex modules for further extension of the module just by changing the input variables to the python script.

This hardware-software integration is crucial, as it allows real-time validation of the FFT core under embedded system constraints. It also opens up opportunities for benchmarking the performance of hardware FFT versus software FFT implementations. The final goal is to extend the solution for use with the Shakti processor by modifying the interface to comply with Shakti's peripheral protocols.



- Figure showing the output results obtained after giving inputs to the FFT IP through Vitis

In conclusion, this project demonstrates a complete design flow from algorithmic understanding to hardware synthesis and embedded integration. It highlights the benefits of hardware acceleration in signal processing tasks and lays the groundwork for more advanced digital designs in future academic or industrial applications.

CONCLUSION

The goal of this project was to design, implement, and verify a scalable Fast Fourier Transform (FFT) IP core using the Cooley-Tukey algorithm in Verilog, and to integrate this core into a practical system through an AXI4-Lite interface for real-time application and testing. The entire design process, from RTL-level development to system-level verification, was carried out on the Xilinx Zynq FPGA platform, ensuring a complete hardware-software co-design approach. Through this project, a robust and reusable FFT IP was successfully created that can perform 8-point FFT and is scalable up to 64-point FFTs with minor modifications.

The Cooley-Tukey algorithm was selected for its divide-and-conquer efficiency and wide acceptance in digital signal processing applications. The implementation in Verilog focused on a radix-2 decimation-in-time (DIT) structure, balancing computational depth and logic complexity. Key architectural components such as butterfly units, twiddle factor generation, pipelining, and fixed-point arithmetic were implemented with performance and area in mind. Fixed-point arithmetic was chosen instead of floating-point due to resource constraints on the FPGA and the desire to optimize for speed and area without significant precision loss for practical signal processing applications.

The FFT module was tested initially in simulation to verify correctness of the transformation and internal states. Thorough testbench design ensured that each module, including butterfly stages, complex multipliers, and bit-reversal logic, behaved as expected. Once simulation results were validated, the design was synthesized and implemented using Xilinx Vivado. The design met all timing constraints across all clock domains and passed static timing analysis, confirming that the design is free from timing violations and suitable for deployment in real-time applications.

A significant achievement in this project was the successful development of a custom AXI4-Lite wrapper around the FFT IP. The AXI4-Lite protocol, a simplified version of the AXI4 interface, is ideal for register-level memory-mapped communication and is widely supported across embedded platforms, especially the ARM Cortex-A9 processing system in the Zynq SoC. The AXI4-Lite interface enabled control and data registers in the FFT IP to be accessed by the processing system (PS), providing a bridge between software and hardware.

The FSM (finite state machine) for AXI communication was carefully designed to follow the AXI handshaking rules for both read and write channels. Separate channels for address, data, and responses were handled in compliance with the AXI4-Lite protocol. Input data samples, control commands, and output results were mapped to appropriate AXI registers, and thorough testing ensured that data consistency and latency requirements were met. Register write and read operations were handled synchronously with the core clock, ensuring glitch-free operation during software-driven data streaming.

Once the AXI-wrapped IP was integrated into a block design in Vivado and exported to Vitis, real-time system-level testing was carried out. The PS communicated with the FFT IP via AXI4-Lite interface and input data samples were fed through a UART interface. The results computed by the FFT module were read back and printed via serial output. Vitis was used to

develop and run a C program on the PS that controlled the IP core, initiated FFT computations, and retrieved the results. The output matched the expected FFT results for various input sets, confirming the correctness and stability of the IP in real-time conditions.

This real-time verification step validated not only the functionality of the FFT logic but also the reliability of the AXI communication infrastructure and the overall system integration. The ability to feed input data via UART and retrieve processed results in real-time mimics real-world use cases in embedded DSP systems, making this implementation directly applicable to a wide range of applications such as communication systems, audio processing, biomedical signal analysis, and radar.

Another strength of the project lies in the scalability of the design. Although the IP was demonstrated for an 8-point FFT, the structure was built with parameterization in mind, allowing for easy scaling to 16, 32, or 64-point FFTs by adjusting loop controls, stage depths, and memory requirements. This makes the IP highly reusable for future projects with different throughput or resolution requirements. Future enhancements could include support for inverse FFT (IFFT), dynamic reconfiguration for FFT size, and integration with a DMA engine for higher bandwidth and lower CPU involvement.

From a learning and professional development perspective, this project offered deep insights into both algorithmic and hardware design concepts. It provided hands-on experience with Verilog coding, FPGA synthesis, timing analysis, AXI protocol handling, and embedded software development. It also simulated an industry-like development flow—starting from algorithm modelling, moving to RTL coding, interface design, hardware/software integration, and finally real-time system verification.

In conclusion, the successful implementation of a scalable FFT IP core with AXI4-Lite interface and real-time verification marks the completion of a highly practical and technically rich project. The integration of a custom signal processing IP into an SoC platform, coupled with the ability to interact with it through software, demonstrates a holistic understanding of digital design, system integration, and embedded development. The final design not only meets functional requirements but also sets a solid foundation for further extension and product-level deployment in real-world FPGA-based systems.

REFERENCES

- https://en.wikipedia.org/wiki/Q_(number_format) **[Number Format]**

- https://people.scs.carleton.ca/~maheshwa/courses/5703COMP/16Fall/FFT_Report.pdf **[FFT using Cooley Tukey Algorithm]**

- https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm **[CT TECHNIQUE]**

- https://www.algorithm-archive.org/contents/cooley_tukey/cooley_tukey.html

- https://www.amd.com/en/products/adaptive-socs-and-fpgas/intellectual-property/axi_interconnect.html **[AXI Interconnect]**

- https://www.amd.com/en/products/adaptive-socs-and-fpgas/intellectual-property/axi.html **[AXI4 LITE]**

- https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-ide.html **[Vitis Environment]**

GLOSSARY

AXI (Advanced eXtensible Interface):
A family of microcontroller buses defined by ARM, used for high-performance and high-frequency data transfer. AXI interfaces are widely used for on-chip communication in SoCs.

AXI4-Lite:
A simplified version of the AXI4 protocol. It supports only single data transfers (no bursts) and is commonly used for low-throughput control interfaces between processors and peripherals.

AXI Wrapper:
A hardware module that encapsulates a custom IP core with AXI-compatible interfaces. It acts as a bridge, allowing external systems to communicate with the IP using standard AXI protocols.

Bit-Reversal:
A technique used in FFT computations to rearrange input or output indices into bit-reversed order to facilitate efficient computation and mapping of butterfly operations.

Butterfly Unit:
A fundamental computation block in FFT, responsible for combining two complex numbers in a specific pattern using addition and subtraction along with twiddle factors.

Cortex-A9:
A dual-core ARM processor integrated into the Xilinx Zynq SoC, used in this project for controlling the FFT IP and handling data transmission via UART.

Cooley-Tukey Algorithm:
A divide-and-conquer algorithm used for efficient computation of the Fast Fourier Transform (FFT). It reduces the computational complexity from $O(N^2)$ to $O(N \log N)$.

DMA (Direct Memory Access):
A method for transferring data between memory and peripherals without involving the CPU. Though not implemented here, it is often used for high-throughput FFT systems.

FFT (Fast Fourier Transform):
An algorithm that efficiently computes the Discrete Fourier Transform (DFT) of a signal. It is widely used in signal processing for frequency domain analysis.

Fixed-Point Arithmetic:
A representation of real numbers using integers and a fixed number of fractional bits. It is more resource-efficient than floating-point in hardware implementations.

FSM (Finite State Machine):
A control logic design pattern used to manage sequential operations. In this project, an FSM was used to manage AXI read/write operations.

IP Core (Intellectual Property Core):
A reusable hardware module (e.g., FFT) described in HDL (such as Verilog) and intended for integration into larger hardware designs.

IFFT (Inverse Fast Fourier Transform):
The reverse operation of the FFT. It converts frequency domain data back into time domain. This feature is suggested for future work.

Latency:
The time delay between input and corresponding output in a system. Achieving low latency is important for real-time FFT applications.

Pipelining:
A technique used to increase throughput by overlapping different stages of computation. It is used in FFT to process multiple inputs concurrently.

PL (Programmable Logic):
The reconfigurable hardware portion of the FPGA (e.g., in Zynq SoC), where custom logic such as the FFT IP is implemented.

PS (Processing System):
The hard processor portion of the Zynq SoC (e.g., ARM Cortex-A9) that communicates with the PL through interfaces like AXI.

Real-Time Verification:
Testing a system with actual inputs and outputs in real time to validate its performance under practical operating conditions.

Register Map:
A memory layout that defines how software can access control and data registers in a hardware IP via a memory-mapped interface.

RTL (Register Transfer Level):
A design abstraction in HDL (like Verilog) where the system is described in terms of data flow between registers and logical operations.

Scalability:
The ability of a design to handle larger FFT sizes (e.g., from 8-point to 64-point) with minimal architectural changes.

Simulation:
The process of testing and verifying hardware logic using software models before synthesis. Used here to test the FFT IP before implementation.

Synthesis:
The process of converting RTL code into a gate-level netlist that can be implemented on an FPGA.

Timing Closure:
The process of ensuring that all timing constraints in a hardware design are met, allowing the circuit to function correctly at the desired clock frequency.

Twiddle Factors:
Precomputed complex exponential constants used in FFT to combine inputs across stages of the butterfly network.

UART (Universal Asynchronous Receiver/Transmitter):
A serial communication protocol used for transmitting and receiving data between the FPGA and external devices such as a PC or processor.

Vitis:
A development platform by Xilinx for writing and running software on the processing system of an FPGA (such as ARM cores in Zynq).

Verilog:
A hardware description language (HDL) used to describe and model electronic systems, such as the FFT IP core developed in this project.

Vivado:
A design suite from Xilinx used for synthesis, implementation, IP integration, and bitstream generation of FPGA designs.

Zynq SoC:
A Xilinx System on Chip that combines ARM-based Processing System (PS) and FPGA-based Programmable Logic (PL), used for implementing hardware-software co-design projects like this FFT system.