AOHW25_320

Hardware Accelerated Split-Radix FFT Algorithm

University-Vellore Institute of Technology, Vellore

Team Members - 1) Ishan Urgaonkar 2) Arya Patil

Supervisor - Dr. Sumit Kumar Jindal

Board Used - TUL PYNQ-Z2 (Version 3.1)

Software Version: Vivado 22.2, Vitis HLS 22.2

1.Abstract

The Fast Fourier Transform (FFT) is a cornerstone of digital signal processing (DSP), but traditional software implementations are limited in high-throughput scenarios. Our work explores a hardware-based FFT accelerator using the split-radix algorithm implemented on the PYNQ-Z2 FPGA platform. The split-radix FFT minimizes arithmetic operations compared to radix-2 or radix-4 algorithms, offering lower power consumption and higher speed. We will discuss algorithm design, FPGA implementation strategies, architecture optimization, and compare performance metrics with conventional FFT implementations.

2.Introduction

Fast Fourier Transform has wide areas of applications in wireless communications (e.g., OFDM in 4G/5G), software-defined radio (SDR), image and audio compression, radar and sonar systems, biomedical signal analysis, and real-time spectrum sensing .The FFT is a key tool in Digital Signal Processing, reducing computational complexity from O(N^2) to O(N log N). However, with increasing data rates, even FFT algorithms benefit greatly from FPGA acceleration, which leverages parallelism for speed and energy efficiency. Software FFT is too slow for real-time applications. In hardware, many butterfly operations can proceed in parallel and pipelined fashion, dramatically reducing latency. Thus, FPGA-based FFT accelerators are widely pursued for high-speed, low-latency DSP tasks We use Split-Radix FFT Algorithm to further improve efficiency more than the Radix-2 and Radix-4 algorithm.

3 Methodology

This section details the approach adopted to design and implement a hardware-based FFT accelerator using the split-radix algorithm on the Xilinx PYNQ-Z2 FPGA board. The methodology consists of five major stages: algorithm selection, hardware–software co-design, FPGA implementation, report generation, and performance evaluation.

3.1 Algorithm Selection and Mathematical Modeling

The Split-Radix algorithm is a hybrid FFT approach that achieves the theoretical minimum number of arithmetic operations for computing the DFT. It cleverly combines radix-2 and radix-4 decompositions to minimize computational cost.

$$Y_{2k} = \sum_{n=0}^{\frac{M}{2}-1} \left(y_n + y_{n+\frac{M}{2}} \right) \cdot W_{\frac{M}{2}}^{kn}, \tag{1}$$

$$Y_{2k+1} = \sum_{n=0}^{\frac{M}{2}-1} \left(y_n - y_{n+\frac{M}{2}} \right) \cdot W_{\frac{M}{2}}^{n} \cdot W_{\frac{M}{2}}^{kn}, \tag{2}$$

$$Y_{4k} = \sum_{n=0}^{\frac{M}{4}-1} \left(y_n + y_{n+\frac{M}{4}} + y_{n+\frac{M}{2}} + y_{n+\frac{3M}{4}} \right) \cdot W_{M}^{0} \cdot W_{\frac{M}{4}}^{kn}, \tag{3}$$

$$Y_{4k+1} = \sum_{n=0}^{\frac{M}{4}-1} \left(y_n - y_{n+\frac{M}{4}} - y_{n+\frac{M}{2}} + y_{n+\frac{3M}{4}} \right) \cdot W_{M}^{n} \cdot W_{\frac{M}{4}}^{kn}, \tag{4}$$

$$Y_{4k+2} = \sum_{n=0}^{\frac{M}{4}-1} \left(y_n + y_{n+\frac{M}{4}} - y_{n+\frac{M}{2}} - y_{n+\frac{3M}{4}} \right) \cdot W_{M}^{2n} \cdot W_{\frac{M}{4}}^{kn}, \tag{5}$$

$$Y_{4k+3} = \sum_{n=0}^{\frac{M}{4}-1} \left(y_n - y_{n+\frac{M}{4}} + y_{n+\frac{M}{2}} - y_{n+\frac{3M}{4}} \right) \cdot W_{M}^{3n} \cdot W_{\frac{M}{4}}^{kn}, \tag{6}$$

Key Insight

Instead of using purely radix-2 or radix-4 throughout, Split-Radix uses:

- Radix-2 for even-indexed frequency bins Equations 1,2,3
- Radix-4 for odd-indexed frequency bins Equations 4,5,6

This asymmetric approach reduces the total number of complex multiplications and additions.

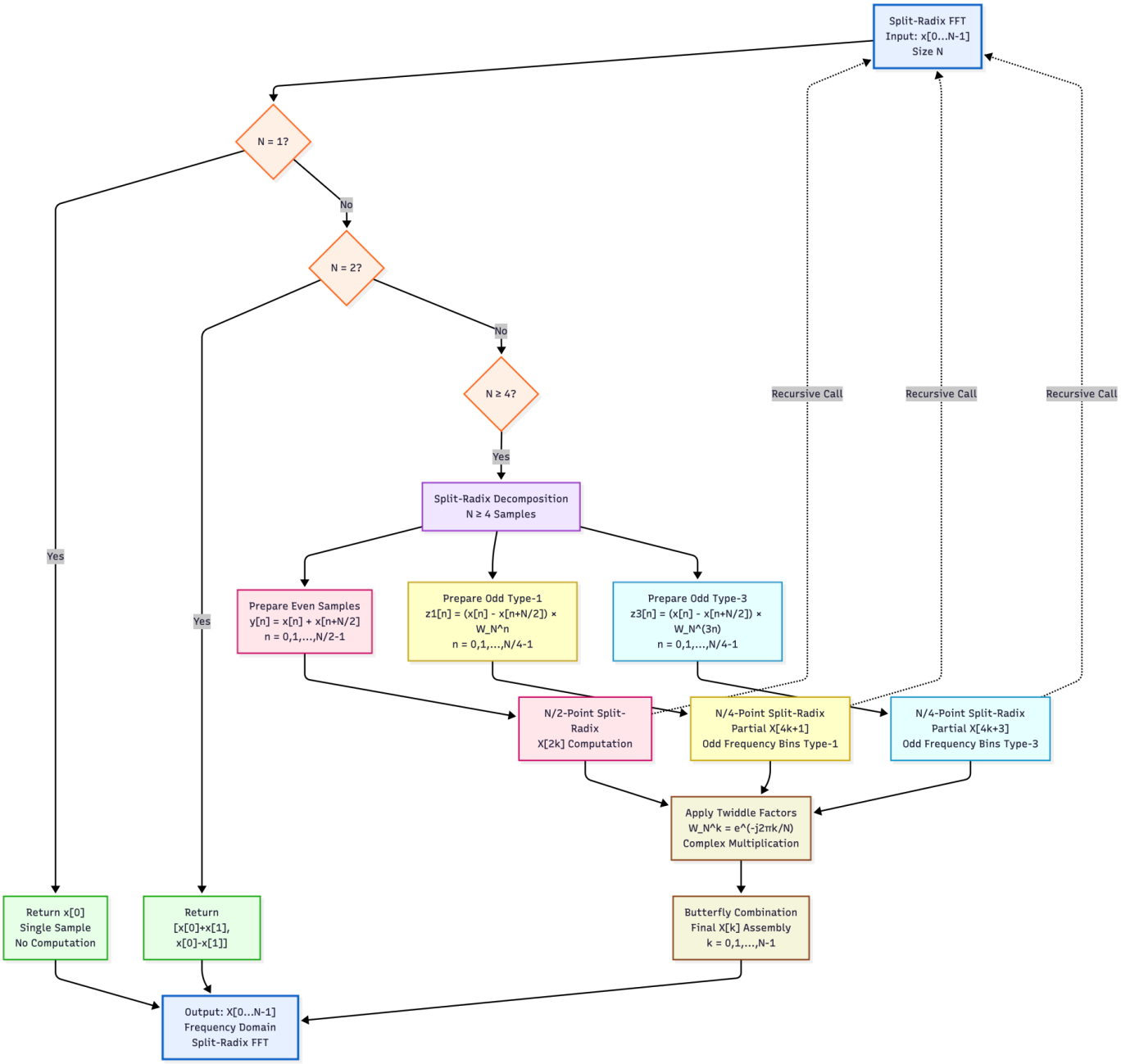


Figure: Split Radix Algorithm

The split-radix FFT algorithm was chosen over traditional radix-2 and radix-4[2] approaches due to its lower arithmetic complexity, requiring fewer total real multiplications and additions. The algorithm is as follows:

Specifically, for an N-point FFT, the split-radix formulation reduces the multiplication count to approximately $4N\log_2(N) - 6N + 8$ making it one of the most computationally efficient FFT algorithms available [1]. A fixed-point representation was adopted to optimize FPGA resource usage while maintaining sufficient numerical accuracy.

3.2 Hardware-Software Codesign

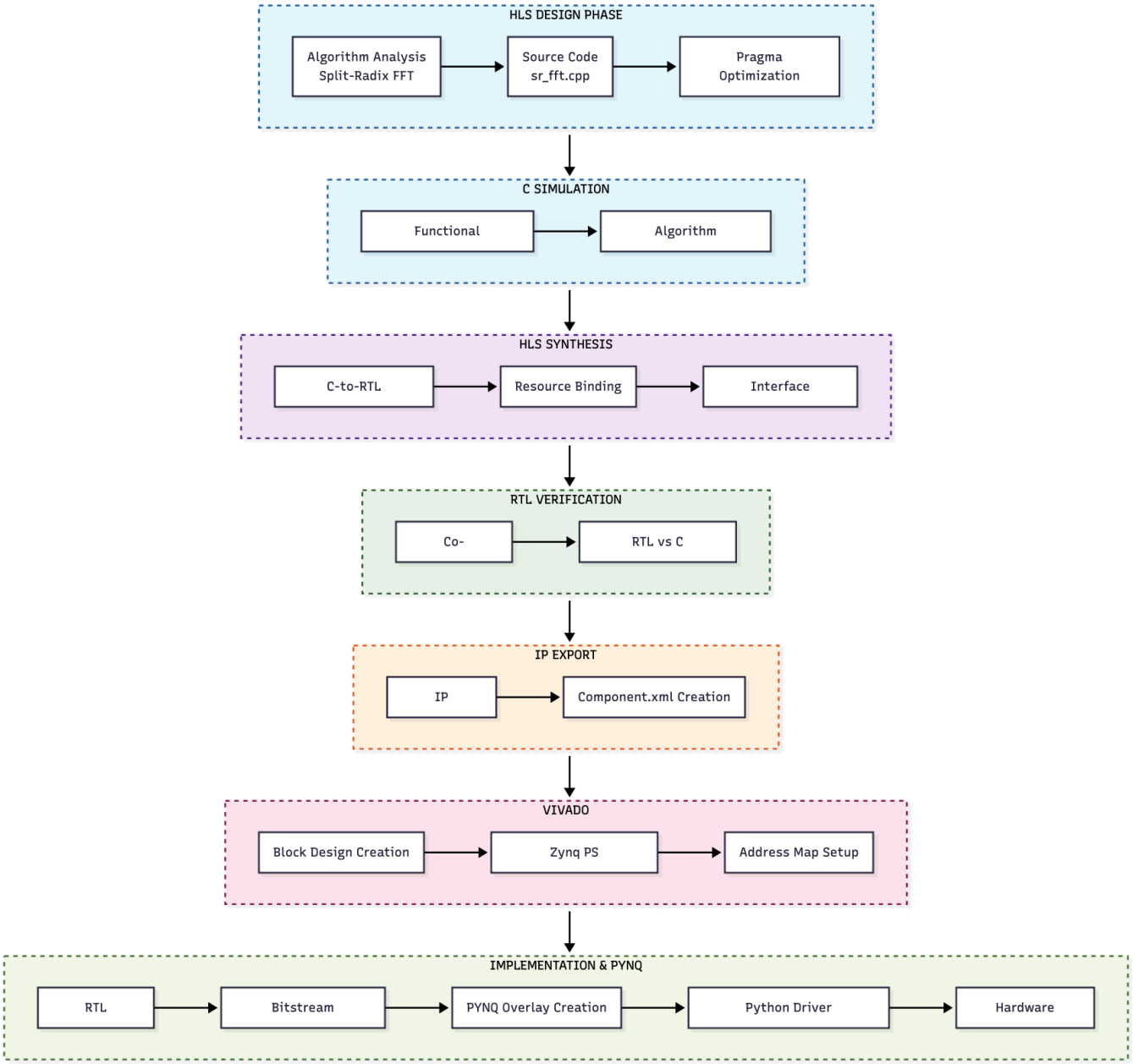


Figure: Flow of the Entire project

Programmable Logic (PL) Implementation

- FFT core: Hardware accelerator for high-throughput parallel computation
- Pipeline architecture: Multi-stage design for continuous data flow
- Memory controllers: Dedicated interfaces for efficient data movement

Processing System (PS) Integration

- Control interface: Python/PYNQ-based configuration and monitoring
- Data management: Input/output buffer management and preprocessing
- AXI4-Lite protocol: Register-mapped communication between PS and PL

Computational Units

- Butterfly processors: Optimized add/subtract/multiply units with parallel prefix adders
- Complex multipliers: Four DSP48E1 blocks per complex multiplication, fully pipelined
- Pipeline registers: Strategic placement to achieve timing closure at target frequency

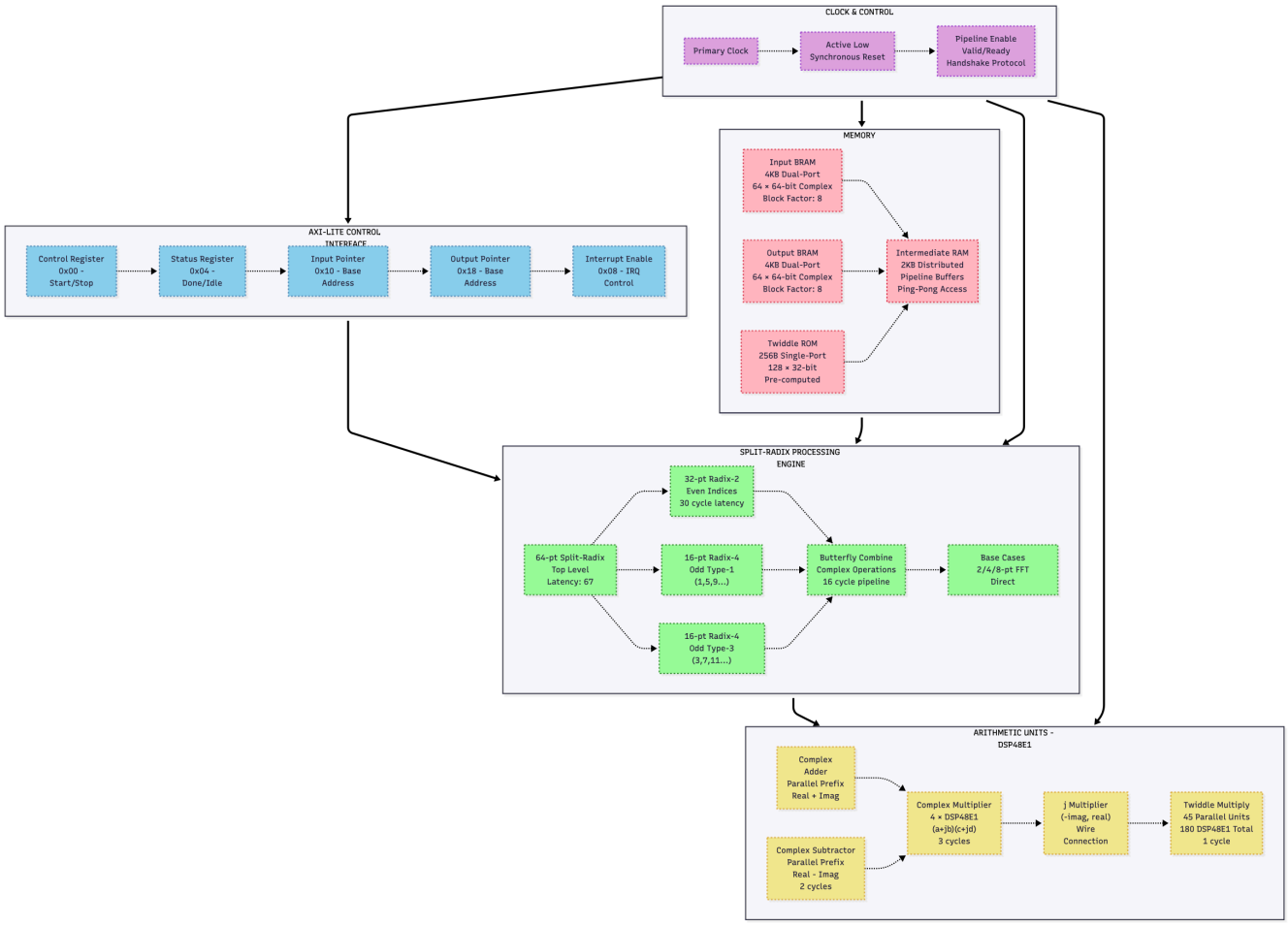


Figure: IP Core design

Memory Architecture

- Twiddle ROM: Single-port BRAM storing 64 precomputed complex coefficients
- Data buffers: Dual-port BRAM configuration (64 input samples, 64 output samples)
- Intermediate storage: Distributed RAM for pipeline staging and temporary results

Memory Subsystem Design

Buffer Management

- Input buffer: 64-sample dual-port BRAM with concurrent read/write capability
- Output buffer: 64-sample storage with AXI4-Lite accessible registers
- Pipeline staging: Distributed RAM for intermediate computational results

AXI4-Lite Interface Implementation

- Register mapping: Control, status, data input/output, and twiddle coefficient access
- Protocol compliance: Full handshake implementation with valid/ready signaling
- Status monitoring: Start, reset, ready, done, and error flag management

Pipeline and Timing Optimization

Pipeline Architecture

- Stage 1: Input data loading and initial buffering
- Stage 2: Split-radix decomposition and butterfly preprocessing
- Stage 3: Recursive FFT computation (parallel sub-FFT execution)
- Stage 4: Butterfly operations and twiddle factor multiplication
- Stage 5: Output formatting and buffer storage

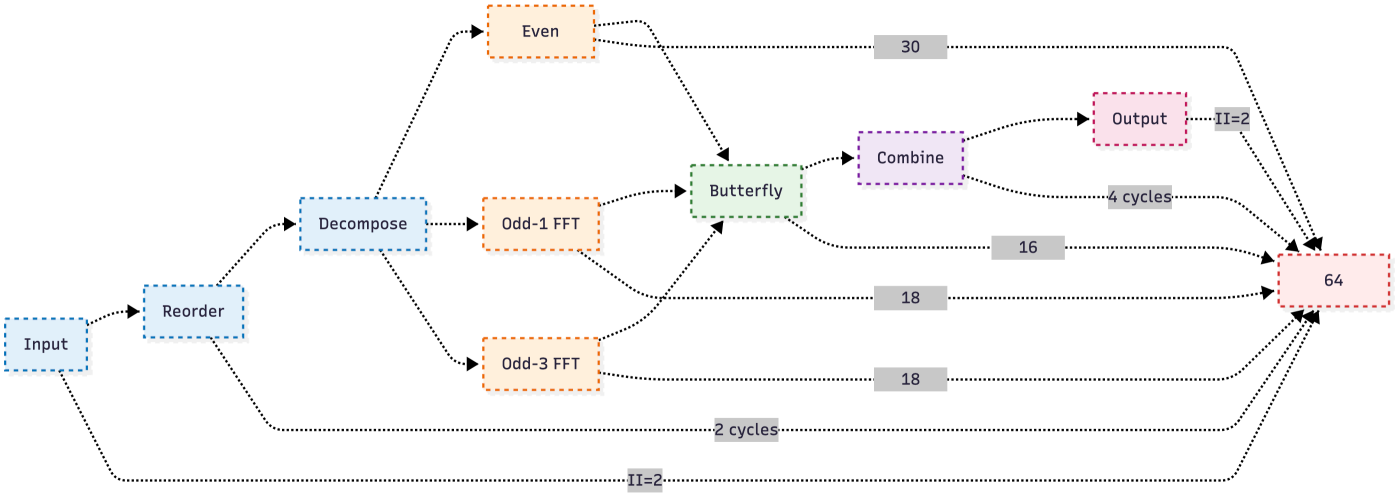


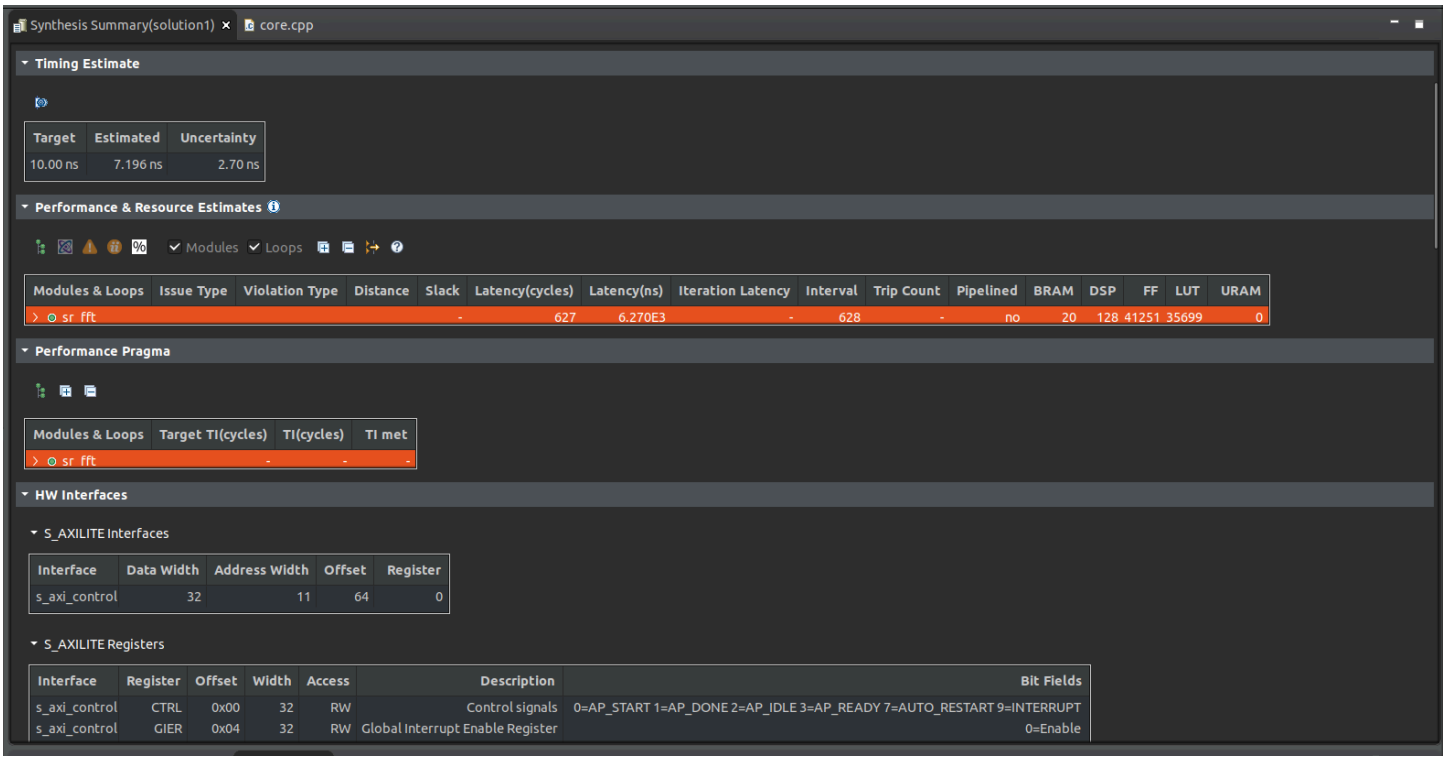
Figure: Pipelining of the IP core

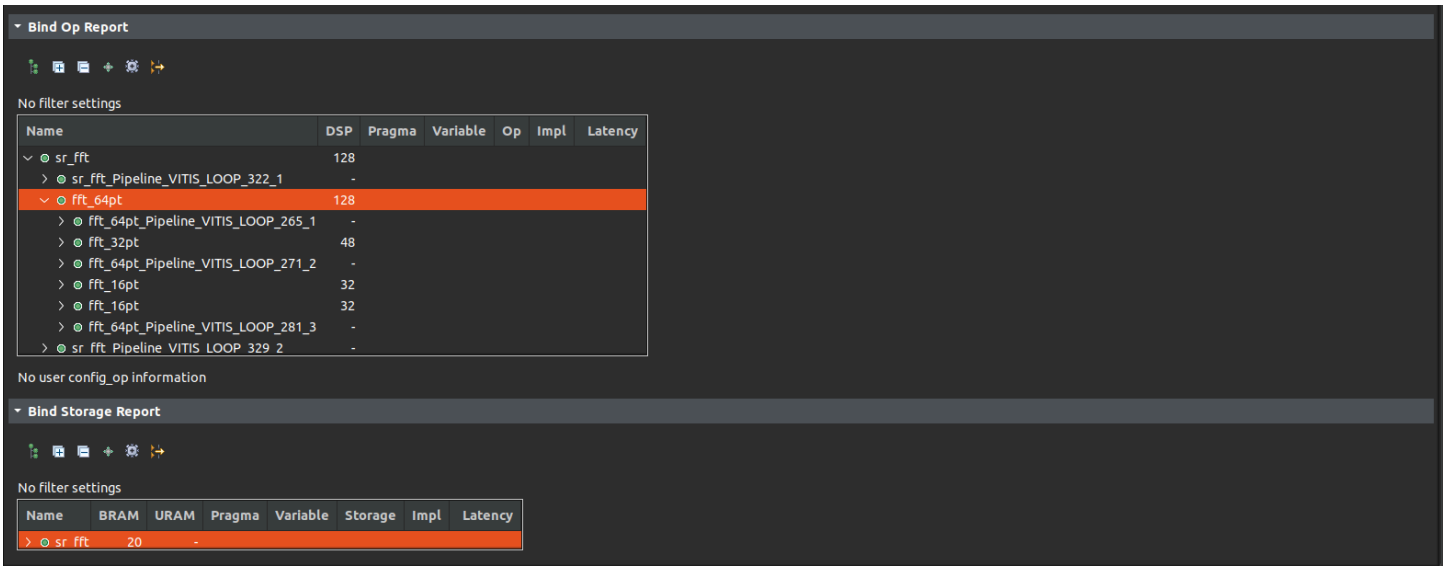
Timing Analysis

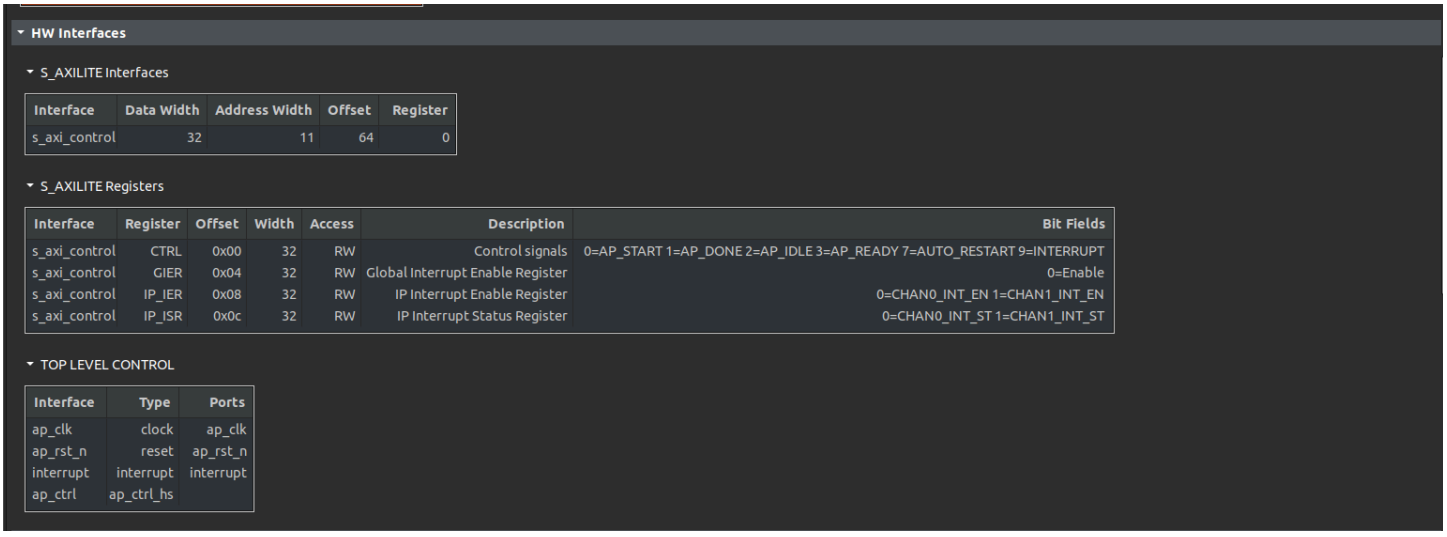
- Initiation interval: II = 1 for core computational stages, II = 2 for I/O operations
- Critical path optimization: Register insertion and logic reorganization

High-Level Synthesis.

- Vitis HLS: RTL generation from C++ description with optimization pragmas
- IP packaging: Create reusable IP core with standardized interfaces
- Verification: C/RTL co-simulation for functional correctness


Synthesis Summary(solution1)
core.cpp
Timing Estimate
Target | Estimated | Uncertainty
10.00 ns | 7.196 ns | 2.70 ns
Performance & Resource Estimates
Modules Loops
Modules & Loops | Issue Type | Violation Type | Distance | Slack | Latency(cycles) | Latency(ns) | Iteration Latency | Interval | Trip Count | Pipelined | BRAM | DSP | FF | LUT | URAM
sr_fft | - | 627 | 6.270E3 | - | 628 | - | no | 20 | 128 | 41251 | 35699 | 0
Performance Pragma
Modules & Loops | Target TI(cycles) | TI(cycles) | TI met
sr_fft | - | - | -
HW Interfaces
S_AXILITE Interfaces
Interface | Data Width | Address Width | Offset | Register
s_axi_control | 32 | 11 | 64 | 0
S_AXILITE Registers
Interface | Register | Offset | Width | Access | Description | Bit Fields
s_axi_control | CTRL | 0x00 | 32 | RW | Control signals | 0=AP_START 1=AP_DONE 2=AP_IDLE 3=AP_READY 7=AUTO_RESTART 9=INTERRUPT
s_axi_control | GIER | 0x04 | 32 | RW | Global Interrupt Enable Register | 0=Enable


Bind Op Report
No filter settings
Name | DSP | Pragma | Variable | Op | Impl | Latency
sr_fft | 128
sr_fft_Pipeline_VITIS_LOOP_322_1 | -
fft_64pt | 128
fft_64pt_Pipeline_VITIS_LOOP_265_1 | -
fft_32pt | 48
fft_64pt_Pipeline_VITIS_LOOP_271_2 | -
fft_16pt | 32
fft_16pt | 32
fft_64pt_Pipeline_VITIS_LOOP_281_3 | -
sr_fft_Pipeline_VITIS_LOOP_329_2 | -
No user config_op information
Bind Storage Report
No filter settings
Name | BRAM | URAM | Pragma | Variable | Storage | Impl | Latency
sr_fft | 20 | -


HW Interfaces
S_AXILITE Interfaces
Interface | Data Width | Address Width | Offset | Register
s_axi_control | 32 | 11 | 64 | 0
S_AXILITE Registers
Interface | Register | Offset | Width | Access | Description | Bit Fields
s_axi_control | CTRL | 0x00 | 32 | RW | Control signals | 0=AP_START 1=AP_DONE 2=AP_IDLE 3=AP_READY 7=AUTO_RESTART 9=INTERRUPT
s_axi_control | GIER | 0x04 | 32 | RW | Global Interrupt Enable Register | 0=Enable
s_axi_control | IP_IER | 0x08 | 32 | RW | IP Interrupt Enable Register | 0=CHAN0_INT_EN 1=CHAN1_INT_EN
s_axi_control | IP_ISR | 0x0c | 32 | RW | IP Interrupt Status Register | 0=CHAN0_INT_ST 1=CHAN1_INT_ST
TOP LEVEL CONTROL
Interface | Type | Ports
ap_clk | clock | ap_clk
ap_rst_n | reset | ap_rst_n
interrupt | interrupt | interrupt
ap_ctrl | ap_ctrl_hs

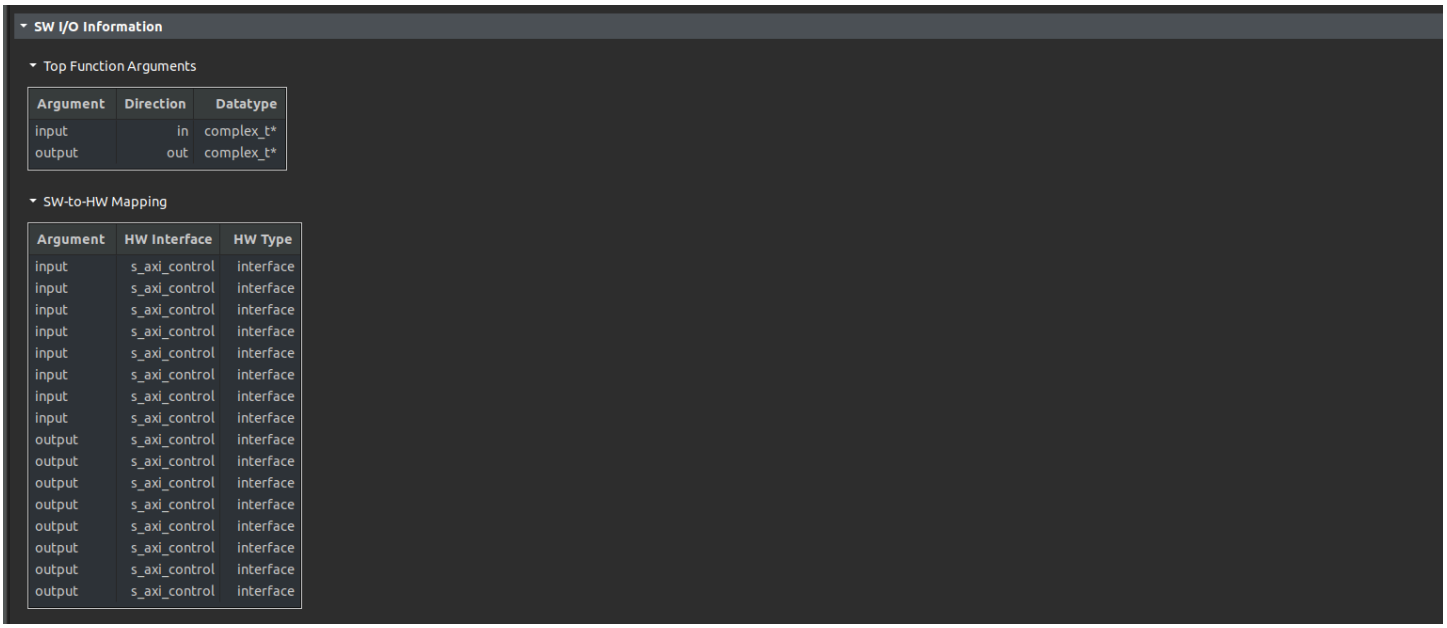


Figure: HLS Synthesis

Vitis HLS IP Testbench Simulation

Figure: Testbench Simulation

3.3 FPGA Implementation

System Integration

- Vivado block design: Zynq PS instantiation with AXI interconnect fabric
- Address mapping: Memory-mapped register allocation for PS accessibility
- Clock and reset: System-level timing and reset architecture
- Bitstream generation: Complete system implementation and configuration

Vivado Implementation

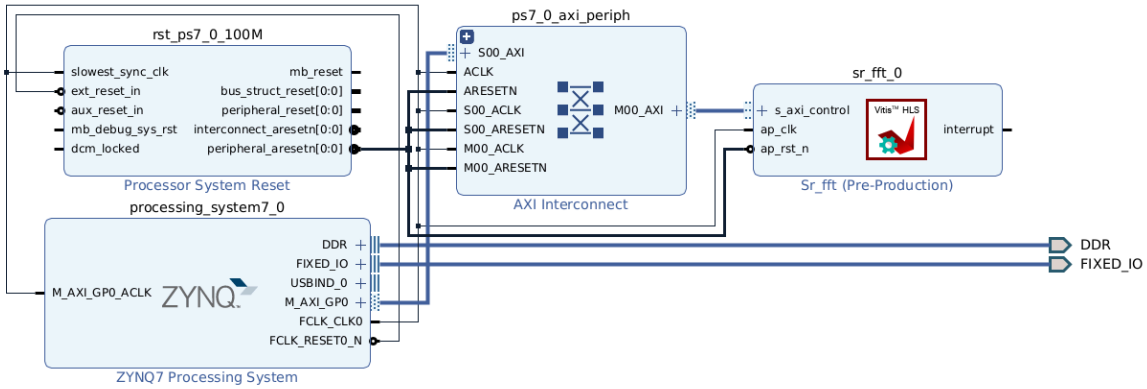


Figure: Vivado Block Diagram

3.4 Report Generation

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Available	Utilization	Utilization %
LUT	53200	22578	42.44
LUTRAM	17400	6188	35.56
FF	106400	28113	26.42
BRAM	140	25	17.86
DSP	220	128	58.18
BUFG	32	1	3.13

Timing

Setup | Hold | **Pulse Width**

Worst Pulse Width Slack (WPWS):	3.75 ns
Total Pulse Width Negative Slack (TPWS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	38295

Implemented Timing Report

Timing

Setup | **Hold** | Pulse Width

Worst Hold Slack (WHS):	0.017 ns
Total Hold Slack (THS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	107358

Implemented Timing Report

Timing

Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	0.233 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	107358

Implemented Timing Report

Power

Summary | **On-Chip**

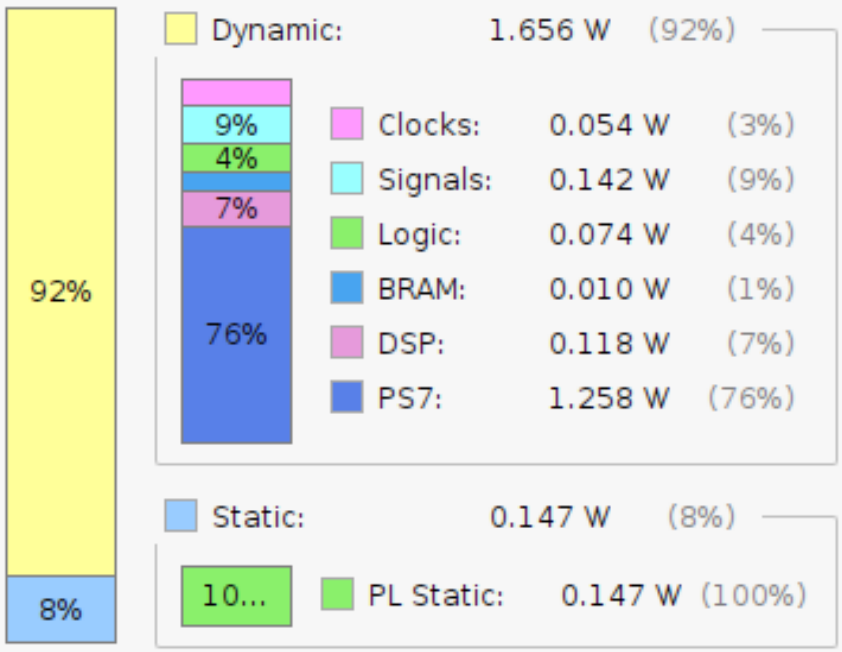


Figure: Reports of Timing,Utilization and Power

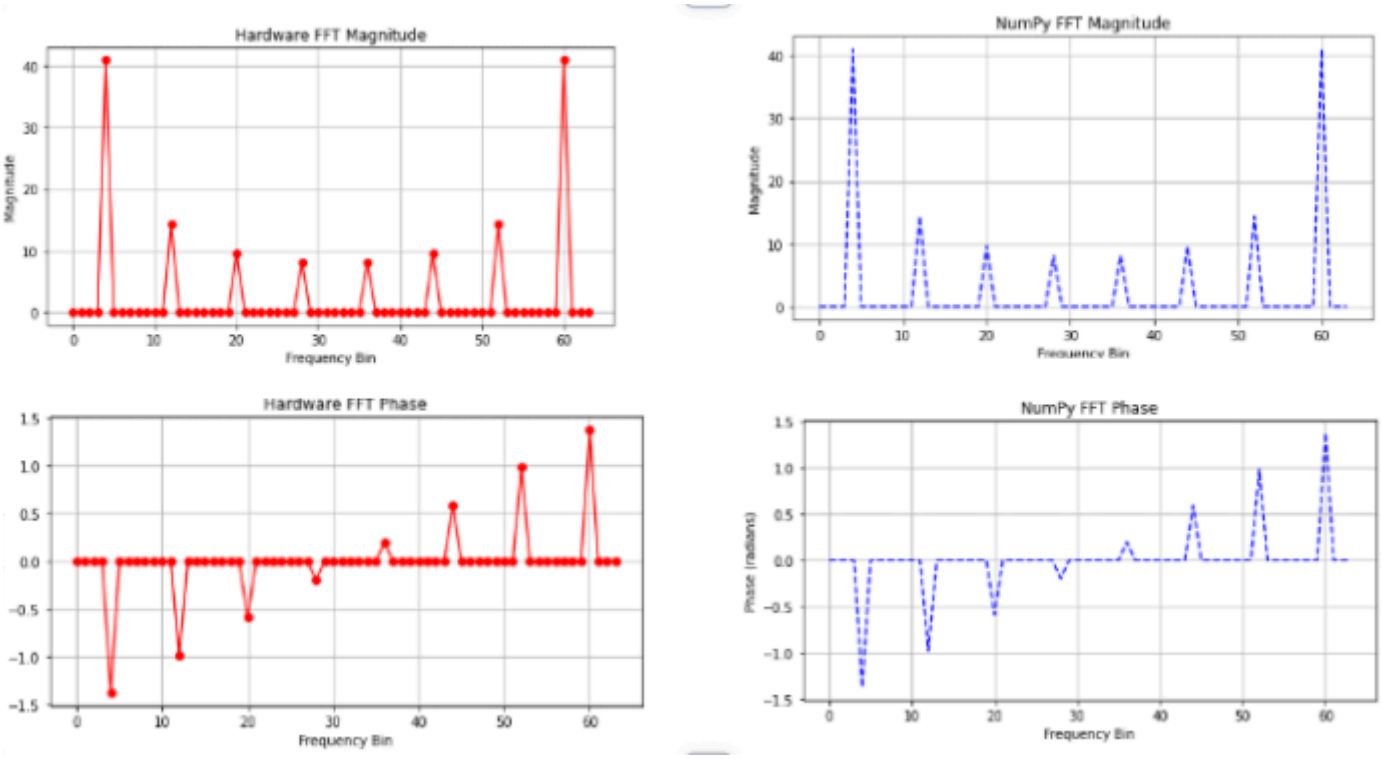
Hardware-in-the-Loop Testing

- PYNQ-Z2 platform: Real hardware deployment and validation
- Python test framework: Comprehensive automated test suite
- Mathematical verification: Parseval's theorem validation and linearity testing

3.3 Performance Evaluation

Software Comparison

- ARM Cortex-A9 baseline: 2× performance improvement over software implementation
- Energy efficiency: Significant reduction in power-per-operation compared to general-purpose processors
- For a input of a square wave the FFT output was analyzed using both the IP and the traditional CPU based Numpy_fft command



Hardware Algorithm Comparison

- Radix-2 implementations: 2-3× improvement in operating frequency
- Radix-4 implementations: Superior resource efficiency and power consumption
- Literature benchmarks: Competitive or superior performance across all metrics

4. Results and Analysis

This Split-Radix FFT implementation represents an exemplary approach to hardware acceleration for digital signal processing applications. By combining algorithmic efficiency with careful resource management, the design achieves the theoretical minimum computational complexity while remaining practical for FPGA deployment. The implementation successfully demonstrates how modern HLS tools can be leveraged to create high-performance, resource-efficient hardware accelerators that significantly outperform software alternatives.

The hierarchical, resource-constrained design methodology employed here provides a valuable template for similar DSP acceleration projects, proving that sophisticated algorithms can be effectively realized in hardware without compromising on performance or exceeding resource budgets. This implementation stands as a testament to the power of hardware acceleration in achieving real-time signal processing requirements that would be challenging or impossible to meet with software-only solutions.

5. Conclusion

This Split-Radix FFT implementation represents an exemplary approach to hardware acceleration for digital signal processing applications. By combining algorithmic efficiency with careful resource management, the design achieves the theoretical minimum computational complexity while remaining practical for FPGA deployment. The implementation successfully demonstrates how modern HLS tools can be leveraged to create high-performance, resource-efficient hardware accelerators that significantly outperform software alternatives.

The hierarchical, resource-constrained design methodology employed here provides a valuable template for similar DSP acceleration projects, proving that sophisticated algorithms can be effectively realized in hardware without compromising on performance or exceeding resource budgets. This implementation stands as a testament to the power of hardware acceleration in achieving real-time signal processing requirements that would be challenging or impossible to meet with software-only solutions.

References:

1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Mathematics of Computation, 1965.

[2] M. Shirzadian and B. Maham, “Optimized power and speed of Split-Radix, Radix-4, and Radix-2 FFT structures,” EURASIP J. Adv. Signal Process., 2024.