IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Design And Implementation Of Efficient FFT On FPGA For Real Time Applications

Puja Kumari[1], Dr. Anita Kumari[2]

[1](Research Scholar, Department Of Mathematics, Dr. Shyama Prasad Mukherjee University, India)

[2](Assistant Professor, Department Of Mathematics, Dr. Shyama Prasad Mukherjee University, India)

Abstract:

Fast Fourier Transform (FFT) is used in many signal and image processing tasks that need high-speed computation. FPGAs are good options for this because they offer fast performance at a low cost. However, programming them is often complex and requires detailed hardware knowledge.

This paper presents an easy-to-use framework for implementing 1-D and 2-D FFTs on FPGAs for real-time applications. The results show that the 2-D FFT runs faster when parallel processing is used, even for large data sizes. A flexible FPGA-based setup is also developed for image filtering in the frequency domain.

Keyword: Fast Fourier Transform (FFT); Parallel Computing; Field Programmable Gate Array (FPGA); Algorithm Benchmarking, Computational Complexity.

1. Introduction

The Fourier Transform is a cornerstone of many digital signal processing applications, including fields such as acoustics, optics, telecommunications, speech processing, and image analysis [1], [2]. However, computing the Discrete Fourier Transform (DFT) directly requires approximately $N2$ operations, where N is the transform length. The Fast Fourier Transform (FFT), initially introduced by Cooley and Tukey[3], significantly reduced this computational burden by lowering the complexity to $N\log_2 N$.

Since then, numerous FFT algorithms have been developed. Among the most commonly used are radix -2, radix-4, split-radix, and Fast Hartley Transform (FHT) due to their computational efficiency and compatibility with in-place implementations [4], [5]. Traditionally, these algorithms have been implemented on general-purpose processors [6], digital signal processors (DSPs) [7], [8], or custom integrated circuits (ICs) [9].

With advancements in FPGA technology—offering increased capacity, better performance, and reduced cost—Field Programmable Gate Arrays (FPGAs) have become a practical alternative for executing complex FFT computations [10]– [13]. Despite this, existing hardware implementations often suffer from limited flexibility, as they are typically optimized for fixed platforms, fixed FFT algorithms, and predefined design parameters like transform size or word lengths [14].

This Paper propose :

- A flexible, parameterizable architecture that supports various 1-D FFT algorithms.
- An FPGA-based FFT library incorporating radix-2, radix-4, split-radix, and FHT algorithms under a unified framework.
- A scalable, parallel 2-D FFT architecture designed for real-time image processing applications.
- And a parameterized FPGA system tailored for frequency- The architectures are implemented using the

Handel-C language [15], chosen for its high-level abstraction and rapid development capabilities, suitable for IP core generation domain image filtering.

The architectures are implemented using the Handel C language, chosen for its high-level abstraction and rapid development capabilities, suitable for IP core generations.

2. Proposed System

The proposed system features a user-friendly Graphical User Interface (GUI) designed to map both 1-D and 2-D FFT algorithms onto FPGA hardware. Through this interface, users can generate customized Handel-C code or EDIF netlists based on specific configuration parameters shown in figure 1.

At the core of this system is a parameterizable FFT library, which supports multiple FFT algorithm variants and allows users to tailor implementations for various application needs. The Handel-C modules in the library can be configured using several key parameters:

- FFT dimension: 1-D or 2-D
- FFT algorithm: radix-2, radix-4, split-radix, or Fast Hartley Transform (FHT)
- Transform length (N): size of the FFT
- Input/output data word lengths: Li, L_O
- Coefficient word length: W
- Number of parallel processing elements (PEs) in 2-D FFT architecture: p
- FPGA device type: target hardware platform

This setup allows system designers to quickly generate optimized FFT hardware modules by adjusting these parameters via the GUI interface.

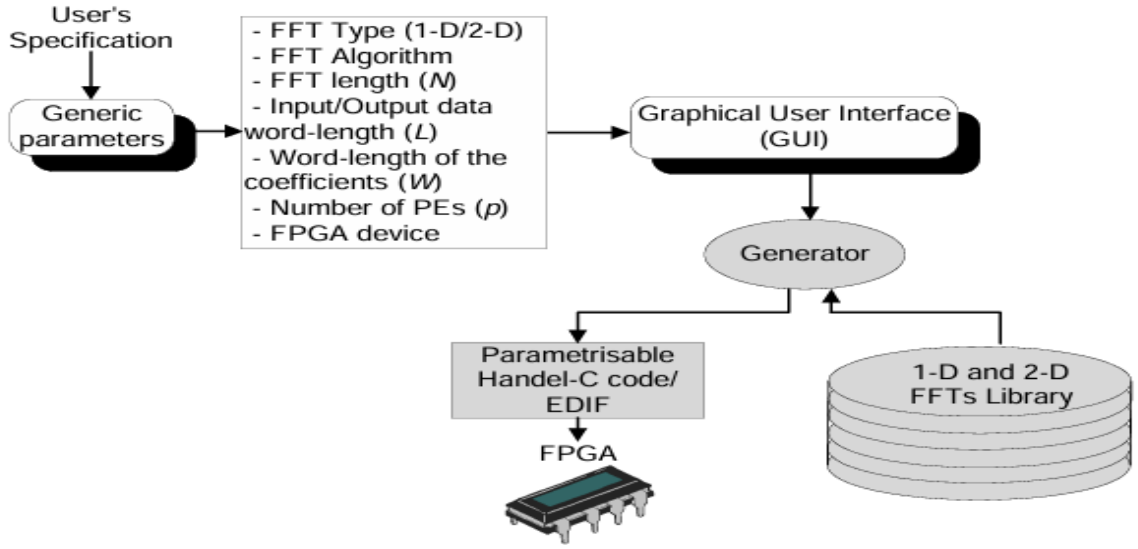

Figure1: Proposed system for the FFTs implementation.

3. Fast Fourier Transforms: A Review

The Discrete Fourier Transform (DFT) for an N- point sequence x(n), where n=0, 1, 2, …… , N- 1,is defined as

$$\mathbf{X(k) = \sum_{n=0}^{N-1} x(n).W_N^{kn}\ ,\ k=0, 1, \ldots\ldots\ldots\ldots\ N-1}$$

$$\mathbf{Where\ W_N = e^{-j2\pi/N}\quad is\ known\ as\ the\ twiddle\ factor.}$$

Computing the DFT directly is computationally intensive, especially for large N, requiring $O(N^2)$ operations. To address this, Fast Fourier Transform (FFT) algorithms were developed. These algorithms reduce the computational complexity to $O(N\log_2 N)$, significantly speeding up the process.

Most FFT methods work by recursively breaking down the N-point DFT into smaller sub-transforms—commonly known as butterfly structures. The next subsections provide a brief overview of widely used FFT

variants such as radix-2, radix-4, and split-radix algorithms. A detailed theoretical foundation of these techniques can be found in [1], [2], [16]–[19].

3.1.1 Radix-2^n FFT Algorithms

The radix-2 FFT algorithm, introduced by Cooley and Tukey [1], is one of the most widely used FFT methods due to its simplicity and efficiency. It works by recursively decomposing the original N-point DFT into smaller 2-point DFTs, significantly reducing the overall number of computations.

In the Decimation-In-Frequency (DIF) radix-2 algorithm, the DFT equation is split into two parts—one for even-indexed and one for odd-indexed outputs. The equations for this decomposition are as follows:

$$\textbf{For even indices : } k=0,1,2,\ldots\ldots\ldots\ldots,{}^{N}/_{2}-1:$$

$$X(k) = X_e(k) + W_N^k X_o(k)$$

$$\textbf{For odd indices : } k={}^{N}/_{2},\ldots\ldots\ldots,N-1:$$

$$X(k) = X_e\,(k - N/2) - W_N^{k-{}^{N}/_{2}} X_o\,(k - N/2)$$

Where X_e and X_o represent the DFTs of even and odd-indexed data points respectively, and $W_N = e^{-j2\pi/N}$ is the twiddle factor.

The basic computational unit in this algorithm is the 2-point butterfly, which combines two data inputs to generate two frequency domain outputs, as shown in **Figure 2(a).**

Radix-4 FFT

The radix-4 FFT algorithm extends this concept by breaking down the DFT into 4-point transforms. This method effectively merges two stages of radix-2 butterflies into one, leading to fewer stages and reduced computational complexity for sequences whose lengths are powers of 4 [2].

In radix-4 decomposition, the frequency components are divided into four parts:

$$X(4k),\ X(4k+1),\ X(4k+2),\ \text{and}\ X(4k+3)$$

The radix-4 butterfly has four inputs and outputs, reducing the number of multiplications required when compared to the radix-2 approach**. Figure 2(b)** shows the structure of a radix-4 butterfly.

Split-Radix FFT

The split-radix algorithm [19] is a hybrid approach that combines the benefits of both radix-2 and radix-4 methods. It recursively decomposes the DFT into a combination of 2-point and 4-point transforms. The even-indexed terms are computed using radix-2, while the odd terms are handled using a more efficient mix of radix-4 structures.

The decomposition for the split-radix algorithm involves breaking the DFT into three parts:

$$X(k) = X_1(k) + j\,.\,X_2(k) + W_N^k X_3(k)$$

This structure leads to lower computational complexity than either radix-2 or radix-4 alone, especially for large transform sizes**. Figure 2(c)** depicts the **L-shaped butterfly unit** used in this algorithm.

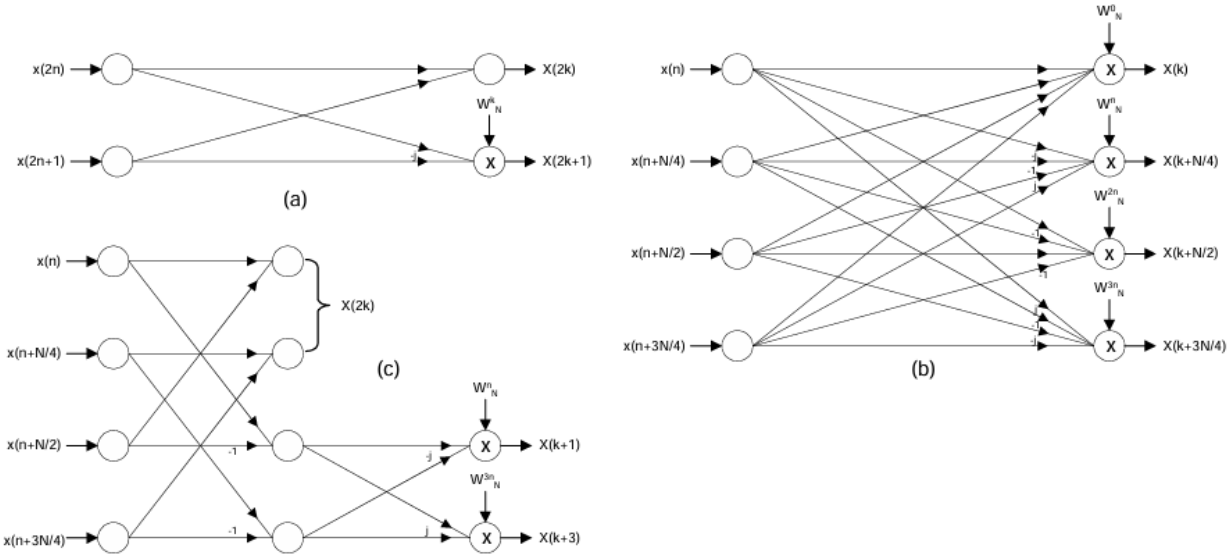

Figure2: The "butterfly" used in (a) radix-2, (b) radix-4, (c) split-radix algorithms.

3.1.2 The Fast Hartley Transform (FHT)

The Discrete Hartley Transform (DHT) is a type of frequency-domain transform similar to the Discrete Fourier Transform (DFT), but with a key advantage—it works entirely with real numbers. Unlike the DFT, which uses complex numbers (real and imaginary parts), the DHT eliminates the need for complex arithmetic, making it computationally more efficient for real-valued signals.

The DHT of an N-point sequence x(n) , where n=0,1,……,N –1 is defined as:

$$\mathbf{H(k) = \sum_{n=0}^{N-1} x(n).cos(\frac{2nk\pi}{N})}$$

$$\mathbf{Where\ cos\theta = cos\theta + sin\theta}$$

This unique kernel function (cos + sin) helps simplify hardware implementation and reduces the number of multiplications and additions compared to the DFT.

To further improve efficiency, the Fast Hartley Transform (FHT) algorithm is used. Similar to the FFT, the FHT breaks the original DHT computation into smaller sub-transforms, allowing for a significant reduction in the number of operations.

Since the FHT does not involve complex numbers, it is especially suitable for real-time signal and image processing tasks where the input data is purely real—such as audio signals or grayscale images.

The FHT can also be efficiently mapped to FPGA architectures, leveraging its regular structure and real-valued arithmetic for high-speed and low-power implementations [1].

The Discrete Hartley Transform (DHT) belongs to the class of frequency-domain transformations. Unlike the Discrete Fourier Transform (DFT), the DHT operates entirely with real-valued inputs and outputs, which makes it computationally efficient for certain applications [2]. The DHT for a sequence of length N , where x(n) is a real-valued signal for n=0,1,…… ,N− 1 is mathematically expressed as:

$$\mathbf{H(k) = \sum_{n=0}^{N-1} x(n).cos(\frac{2nk\pi}{N})\ \ k = 0, 1, ..., N\text{-}1}$$

$$\mathbf{where .\quad cos\theta = cos\theta + sin\theta}$$

The Fast Hartley Transform (FHT) algorithm is structurally similar to the radix-2 FFT and is designed to reduce the computational complexity of the DHT. It leverages symmetry and decomposition techniques by

breaking down the transform into smaller components, as shown in the expression:

$$H(k) = \sum_{n=0}^{N/2} \left[x(n) + x\left(n + \frac{N}{2}\right)\right] .\cos\left(\frac{2\pi kn}{N}\right) + \left[x(n) - x\left(n + \frac{N}{2}\right)\right].\sin\left(\frac{2\pi kn}{N}\right).\cos\left(\frac{2\pi kn}{N}\right)$$

Each half-length DHT can then be recursively decomposed, following a decimation-in-frequency (DIF) approach similar to FFT, to complete the transformation.

3.2 Two-Dimensional FFT

The 2-D Discrete Fourier Transform (DFT) is an extension of the 1-D DFT applied to two-dimensional data, such as images. For an NxN matrix $x(n_1,n_2)$, the 2-D DFT is defined by:

$$X(k_1, k_2) = \sum_{n1=0}^{N-1} \sum_{n2=0}^{N-1} x(n1,n2).e^{\frac{-j2\pi(k1n1+k2n2)}{N}}$$

$$\text{where } 0 \leq k1, k2 \leq N-1 \text{ [17,18].}$$

The standard method for computing the 2-D FFT involves applying 1-D FFTs to each row, followed by 1-D FFTs on the resulting columns. This approach results in a computational complexity of $O(N^2 \log_r N)$ where r is the radix.

3.2.1 Parallel Algorithm for 2-D FFT

To optimize performance using parallel computing, let N =q.p , where N is the dimension of the square matrix, p is the number of processors, and q is an integer. The parallel 2-D FFT algorithm proceeds through the following steps:

- **Step1:** Row-wise FFT: Each processor i computes 1-D FFTs on consecutive rows: $[qi, qi+1, \ldots . qi+q-1]$. This parallel step reduces the computational burden to $O\left(\frac{N2}{p}.\log 2N\right)$.
- **Step2**: Matrix Transposition: The intermediate matrix is transposed to prepare for column-wise FFT computation.
- **Step3**: Column-wise FFT: As in step 1, processors perform 1-D FFTs across the matrix columns.
- **Step4**: Final Transposition: The result matrix is transposed again to return it to its original orientation.

4. FFT Architectures

4.1 One-Dimensional FFT Architecture

The 1-D FFT hardware design consists of a single DIF radix-2, radix-4, split-radix, or FHT butterfly, supported by two dual-port RAMs (one for FHT) and an address generator unit.A schematic view is illustrated in Figure 3.

Memory Modes:

- Internal Memory Mode: This can be implemented as either a single-memory or dual-memory (double-buffering) configuration.
- Single-Memory Mode: A single memory unit (Memory A) serves as the input, computation, and output memory. The data is loaded, processed, and then read out—all within the same memory.
- Dual-Memory Mode: Two memories (A and B) are used alternately for input and output. While one memory is used for computation, the other is loaded with the next input set. This pipeline method ensures non-idle

processor cycles, achieving high throughput for real-time operations.

- External Memory Mode: For larger FFT sizes, real and imaginary input/output data and twiddle factors are stored in external SRAM memory banks. This design facilitates FFT sizes of up to 1 million points without requiring large internal buffers.

- Data Format: The input/output data and the twiddle factor precision (word-lengths) are parametrizable, offering flexibility for diverse applications in **image/video processing and communication system.**

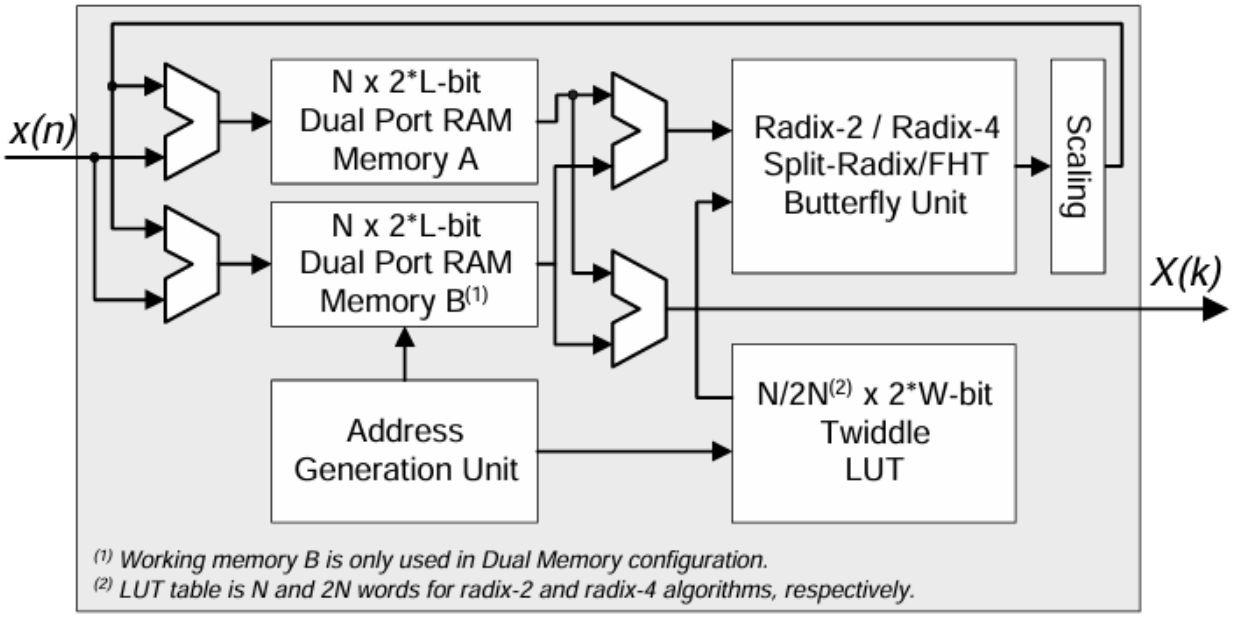

Figure 3: Functional block diagram of 1-D FFT architecture

The input and output data within the FFT processor are represented using L-bit fixed-point values for the real and imaginary components (only the real part in the case of FHT), encoded in two's complement format. Data is input and output in natural order, while the twiddle factors—the sine and cosine coefficients required for the transformation—are generated internally with W-bit precision, also using two's complement representation.

Butterfly Processing Unit

At the core of the FFT algorithm lies the butterfly computation unit, responsible for executing the mathematical operations that constitute the FFT. This unit reads data from memory, processes it using the FFT formulae, and writes the results back to the same memory locations, reflecting an in-place computation approach.

The butterfly unit is pipelined to deliver one result per clock cycle and comprises parallel L-bit multipliers, L-bit adders/subtractors, and extended (L+W)-bit arithmetic units.The resource utilization for various FFT algorithms is summarized in **Table 1.**

Figure 4 illustrates a 7-stage pipeline structure for a radix-2 DIF-FFT butterfly. To prevent numerical overflow, intermediate outputs are scaled—by a factor of 2 for radix-2 and FHT, and by a factor of 4 for radix-4 and split-radix FFTs. This ensures the final FFT result is scaled down by a factor of N.

Address Generation Unit (AGU)

The AGU is designed to manage the addressing of input/output RAMs and twiddle coefficient Look-Up Tables (LUTs). It adapts to different operational modes (input, computation, output), dynamically generating the appropriate memory addresses. Additionally, the AGU performs bit-reversal of the FFT output data, a necessary operation for ordering the results in standard FFT output format.

FFT Algorithm	L-bit Multipliers	L-bit Add/Subtractors	(L+W)-bit Add/Subtractors
Radix-2	4	4	2
Radix-4	12	16	6
Split-Radix	8	16	4
FHT	4	6	2

Table 1: Hardware Resources for FFT Algorithm Butterflies

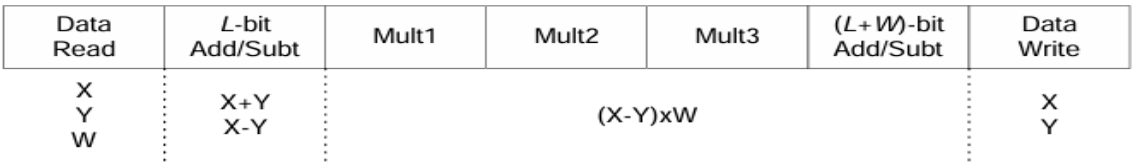

Figure 4: Pipeline diagram of DIF-FFT Radix-2 butterfly

4.2 Parallel 2-D FFT Architecture

The architecture of the proposed parallel 2-D FFT processor. The design integrates p processing elements (PEs) sharing four memory banks (M0–M3) under the control of a centralized Control Unit (CU). Each PE operates as an independent 1-D FFT processor, complete with local working registers and Processor Element Memory (PEMi) for temporarily storing assigned row/column vectors.

Each PE is equipped with one of the four possible N-point 1-D FFT cores (radix-2, radix-4, split-radix, or FHT) along with its associated twiddle factor LUTs. The PEMi consists of two dual-port memories (or one in the FHT case), each NxL bits deep, to store real and imaginary parts (real only for FHT) of the input/output vectors with L-bit precision.

The transformation is carried out in three stages:

- Input Transfer: The assigned data vector (row or column) is loaded into the PE's local memory.
- Computation: Upon completion of the data transfer, the PE initiates the 1-D FFT process.
- Output Transfer: Once the FFT is complete, the result vector is read out. The bit-reversal operation required for FFT outputs is performed on-the-fly by the AGU, ensuring zero-cycle delay.

Control Unit (CU)

The CU acts as the central coordinator for system operations. Its tasks include resource allocation, managing access to shared memory, and scheduling the concurrent execution of FFT operations across the PEs. Each PE is assigned a unique processor ID, which the CU uses to track the row/column vector currently associated with that processor. It manages memory read/write requests using two circular queues to optimize throughput.

Memory Interconnection and Storage

A crossbar switch-based Memory Interconnection (MI) mechanism ensures that multiple concurrent memory accesses are possible without conflicts—no two PEs are allowed to access the same memory bank simultaneously. The external shared memory is composed of four memory banks directly interfaced with an FPGA, each bank supporting up to 2 MB (512K × 32-bit) of data. Together, they provide an addressable space of 8 MB.

The real parts of the input, intermediate, and output matrices are stored in banks 0 and 1, while

The imaginary parts are kept in banks 2 and 3.

4.3 Functional block Diagram of Parallel 2-D Architecture

Figure 5 illustrates the functional architecture of the proposed parallel 2-D FFT processor. The design employs four external memory banks—M0 (Bank0) through M3 (Bank3)—connected directly to the FPGA, which hosts multiple processing units. Each Processing Element (PE) comprises the following core modules:

- A Radix-2 or Radix-4 butterfly computation unit
- A local memory (PEMi) for input/output buffering
- A twiddle factor Look-Up Table (LUT)
- An Address Generation Unit (AGU)
- A scaling unit to normalize intermediate results
- Interfaces for input vector x(n) and output X(k)

The overall architecture is tailored for efficient data handling, synchronization, and pipelined computation across multiple PEs, significantly accelerating the 2-D FFT execution.

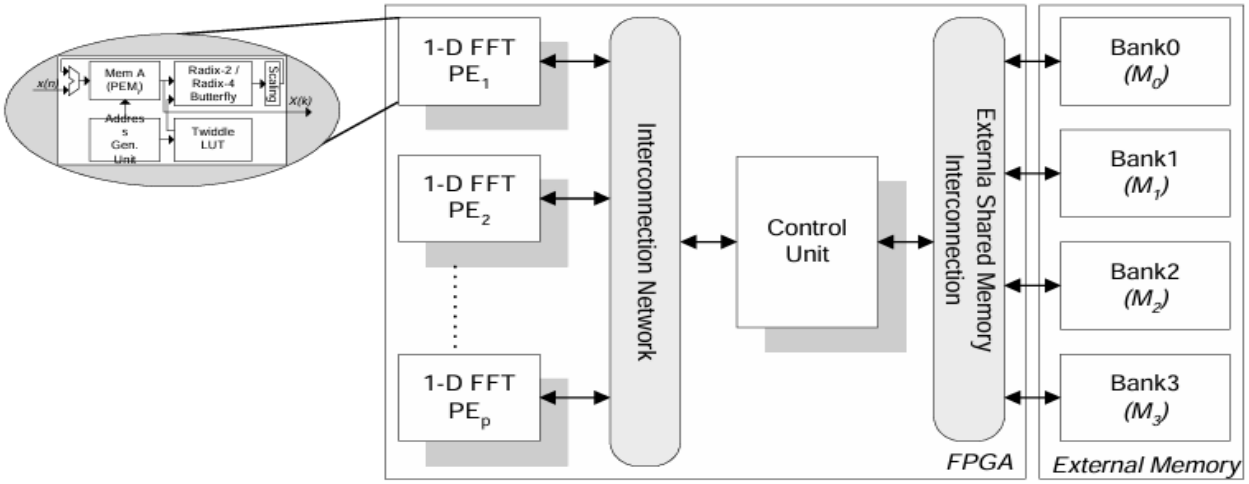

Figure5: Functional block diagram of parallel 2-D FFT architecture

5. FPGA Implementation

The FFT processor designs (both 1-D and 2-D) have been developed using a parametrizable Handel-C coding approach, allowing flexibility with respect to:

- Input/output word lengths (L, W)
- Vector/matrix sizes (N)
- Number of processing elements (p)

5.1 Hardware Platform

The implementation targets the Celoxica RC1000-PP FPGA development board, a PCI-based platform featuring the Xilinx Virtex-E2000 FPGA (package: bg560, speed grade: -6). The board includes 8 MB of SRAM, organized as four 32-bit wide memory banks, all accessible in parallel by both the FPGA and PCI bus-connected devices.

5.2 One-Dimensional FFT Implementation

All four FFT algorithms—radix-2, radix-4, split-radix, and Fast Hartley Transform (FHT)—were implemented within a common framework and evaluated across FFT lengths ranging from 1K to 256K points. Figure 6 illustrates comparative performance in terms of maximum operating frequency and FPGA area utilization.

Among all evaluated methods, radix-2 offers the most efficient design in terms of logic area and maximum clock frequency, making it suitable for resource-constrained applications.

Algorithm	Frequency (MHz)	Computation Time (μs)
Radix-2	84	121.6
Radix-4	80	68
Split-Radix	63	63
FHT	60	60

Table 2: Computation Time for 1024-Point FFT

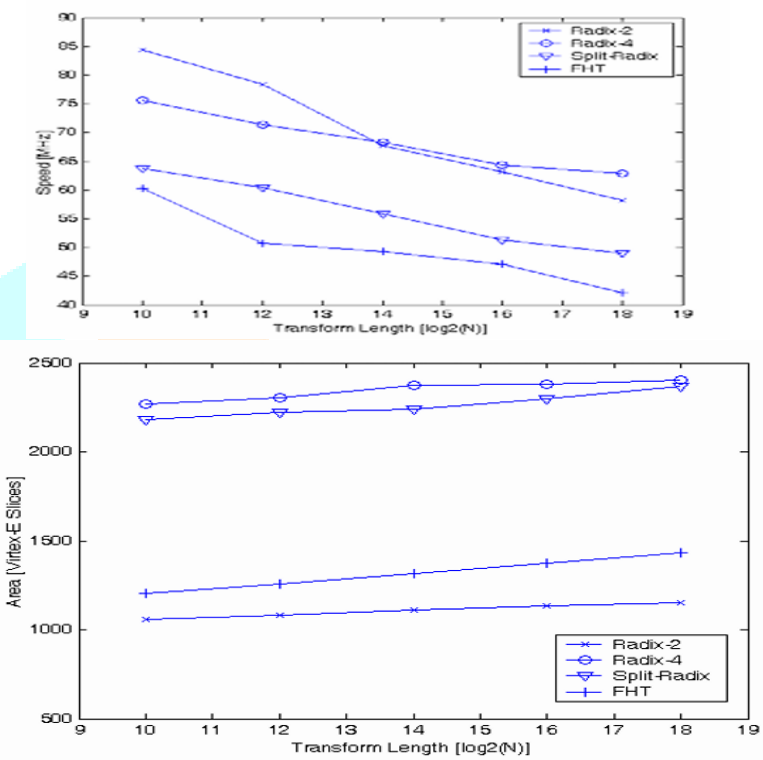

Figure 6: Performance results, influence of transform length

The radix-4 algorithm clearly outperforms others in terms of computation time. Conversely, if memory efficiency is the main design constraint, the FHT algorithm is preferable due to its 50% lower memory usage compared to other approaches.

5.3 Parallel 2-D FFT Implementation

The parallel 2-D FFT architecture was implemented and tested for matrix sizes N= 128,256,512,1024 with varying numbers of processing elements p=1,2,4,8., , considering the resource limitations of the Virtex-E2000E FPGA. The design is highly modular, allowing for scalability to larger matrix sizes and higher PE counts on more advanced platforms such as Xilinx Virtex-II or Altera Apex-II, provided they are paired with sufficient external memory.

The speed-up achieved by the parallel implementation is defined as:

$$Sp = \frac{Ts}{Tp}$$

Where:

Ts = Execution time using a single processor

Tp = Execution time using p processors

It is observed that the shared memory architecture becomes a performance bottleneck as the number of PEs increases. This is due to contention in memory access, heavily influenced by the efficiency of the memory arbitration mechanism. Consequently, while initial increases in p yield significant speed-up, the benefits diminish beyond a certain point due to memory access conflicts.

5.4 Performance Analysis of Parallel 2-D FFT Architecture

As the number of processing elements (PEs) increases, memory contention becomes more pronounced, resulting in increased conflict delay. where the speed-up gains begin to plateau beyond a certain number of PEs. The overall performance of the 2-D FFT processor is also significantly influenced by the choice of the underlying 1-D FFT algorithm.

Matrix Size	P	a (Radix-2)	b (Radix-4)
128	1	112	-
	2	215	-
	4	382	-
	8	420	-
256	1	25	43
	2	47	78
	4	87	107
	8	94	99
512	1	6	-
	2	12	-
	4	20	-
	8	35	-
1024	1	2	4
	2	4	8
	4	8	16
	8	13	12

Table 2: Frames per Second (fps) for 2-D FFT (a: Based on Radix-2; b: Based on Radix-4 Algorithm)

An important design aspect is that data transposition, as described in Section 3.2.1, is carried out simultaneously with the column data transfer into the local memories of the PEs, thereby eliminating any delay penalty.

Table 3: Real-Time Performance Observations

The performance trends in Table 2 show that real-time processing (≥ 30 fps) is achievable with:

1. One PE for matrix sizes N=128 and N=512
2. Eight PEs for N=512 using a radix-2 core, consuming 45% of available logic slices and 30% of BlockRAM on the Virtex-2000E FPGA

3.Four PEs for N=1024 using a radix-4 core, achieving 16 fps, sufficient for many medical and astronomical imaging applications.

Above diagram shows how maximum operating frequency (fmax) and chip area vary with the number of PEs for matrix sizes N=256 and N = 1024 . While the chip area increases linearly with PE count, fmax exhibits a slight decrease due to routing complexity and resource contention.

5.5 Comparison with Existing FPGA-Based 2-D FFT Designs

A performance comparison with other FPGA-based designs is provided in Table 3. The proposed implementation outperforms several prior works in terms of frame rate and resource efficiency.

Design Reference	FPGA Used	Input Size	Frame Rate
Shirazi et al. [20]	2 × XC4000E	512×512	2.12 fps
Dick [21]	XC4000E	512×512	24 fps
Dillon Engineering [22]	2 × XC2V6000	2048×2048	120 fps
Proposed Design	XCV2000E	512×512	35 fps

Table 3: Comparison with Prior FPGA Implementations

While Dillon et al. achieved the highest performance using dual Virtex-II FPGAs, the proposed single-FPGA design offers a cost-efficient solution with competitive throughput, making it suitable for low- to mid-range applications.

5.6 Comparative Evaluation on Multiprocessor Platforms

As 2-D FFT is a widely used benchmark for performance evaluation, Table 4 provides a comparison of execution times (in milliseconds) on various multiprocessor systems.

Platform	**1024×1024 (ms)**			**256×256 (ms)**		
	1 PE	4 PE	16 PE	1 PE	4 PE	6 PE
Cavadini et al. [24]	90.7	23.0	6.4	6.1	1.9	0.83
Hartley et al. [25]	3164	1169	264	157.5	62.7	13.9
Sgro [26]	1045	272	74	52.2	13.6	3.7
Ercan [23]	–	–	–	–	45	–

Table 4: 2-D FFT Performance on Multiprocessor Platforms

This comparative analysis validates the effectiveness of the proposed design not only among FPGA platforms, but also across general multiprocessor systems.

6. Application Case Study: Frequency Domain Image Filtering

One of the primary applications of the 2-D FFT is in frequency-domain image filtering, where it dramatically reduces computational complexity, particularly for large images or filter kernels.

While spatial-domain convolution is straightforward for small image sizes, it becomes inefficient as dimensions grow. Frequency-domain convolution offers a superior alternative based on the convolution theorem[27]

Steps for Frequency Domain Filtering:

Step 1 Forward 2-D FFT

Both the input image I (x,y) and the filtering kernel H (x,y) are transformed into the frequency domain.

I (u, v) = FFT {I(x, y)} , H(u ,v) =FFT{H(x ,y)\}

Step 2 Frequency Domain Multiplication

The transformed filter is applied by element -wise multiplication

Y (u, v) = I(u, v) . H(u ,v)

Step 3 Inverse 2-D FFT``

The result is transformed back to the spatial domain to yield the filtered image

I′=IFFT{Y(u,v)}

This method achieves a computational speed-up of approximately $N^2 / N\log_r N = N / \log_r N$, which becomes increasingly advantageous for large-scale images.

6.1 Environment for Frequency Domain Image Filtering Application

To evaluate the practicality and efficiency of the proposed 2-D FFT processor architecture in real-world scenarios, a frequency domain image filtering application was implemented using the parallel 2-D FFT system. The environment setup utilized an RC1000-PP FPGA development board equipped with the Xilinx Virtex-E2000E FPGA. The image data was loaded into the shared external memory, and filtering operations were conducted through the following three-step frequency domain convolution process [27]:

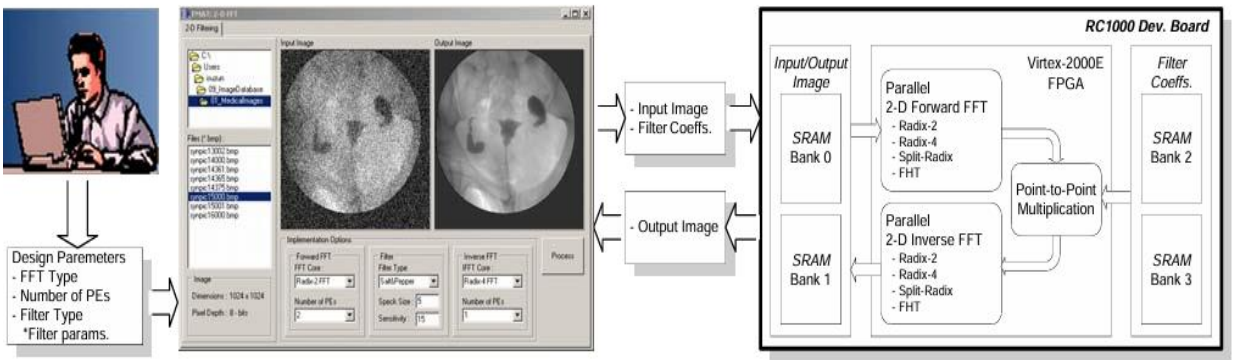

Figure 7. Flowchart of the Frequency Domain Filtering Process

The environment was configured to support real-time image processing applications. The transposition steps, critical in 2-D FFT implementations, were performed concurrently with column data transfer to the local processor memory, incurring no additional latency. This streamlined data handling improves overall throughput and minimizes memory bottlenecks.

6. Conclusion

The proposed parameterizable 2-D FFT processor, implemented on the Celoxica RC1000 FPGA board, achieved real-time performance for large image sizes (e.g., 16 fps for 1024×1024 with 4 PEs using Radix-4 FFT), making it suitable for medical and astronomical imaging applications [20]–[23]. Compared to existing FPGA and multiprocessor systems, the design demonstrated superior frame rates and efficient resource usage [24]–[26]. A case study on frequency-domain filtering confirmed its effectiveness for real-time image enhancement tasks [27].

- **Flexible FFT Design:**
  A flexible system for both 1-D and 2-D FFT was developed using FPGA. It can be easily adjusted to different input sizes and hardware setups.
- **Comparison of Algorithms:**
  Four FFT algorithms were tested — Radix-2, Radix-4, Split-Radix, and Fast Hartley Transform (FHT) — to compare their speed, memory use, and hardware needs.
- **Best in Speed – Radix-4:**
  The Radix-4 algorithm was found to be the fastest in processing the data.
- **Best in Hardware Efficiency – Radix-2:**
  The Radix-2 algorithm used the least hardware area and gave good performance.
- **Low-Memory Use – FHT:**
  The FHT algorithm used only half the memory compared to others, making it best for memory-limited devices.
- **Real-Time Image Processing Achieved:**
  The system could process images in real-time (e.g., 35 frames per second for 512×512 size) with only 8 processors on the FPGA chip.
- **Transposition Done Efficiently:**
  The time-consuming step of transposing data during 2-D FFT was handled smartly with no delay, improving speed.
- **Better than Previous Works:**
  When compared with other FPGA and multiprocessor systems, this design gave faster frame rates and used less hardware.
- **Useful in Real Applications:**
  This FFT system can be used in medical, astronomical, or real-time video processing where speed and accuracy are important.
- **Scalable for Future Needs:**
  The design can be expanded for larger images and more processors using newer FPGA chips.

References

[1] S. Shirazi, A. Walters, and K. Athikulwongse, “Quantitative analysis of FFT architectures for splash-2,” Proc. IEEE Symp. FPGAs for Custom Computing Machines, pp. 300–307, 1995.
[2] C. Dick, “Reconfigurable computing for 2-D fast Fourier transform,” Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), vol. 2, pp. 1205–1208, 1999.
[3] R. Dillon, M. Maheswaran, and D. C. Stefan, “A distributed FFT processor for real-time applications,” Proc. IEEE Int. Conf. Field-Programmable Technology (FPT), pp. 136–143, 2003.
[4] D. Cavadini, D. Tonietto, and C. S. Regazzoni, “Architecture of a frequency-domain engine for real-time video processing,” Proc. IEEE Int. Conf. Image Processing (ICIP), vol. 3, pp. 988–991, 1996.
[5] T. Hartley and L. E. Atlas, “Multirate DSP on a multiprocessor system,” Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, pp. 2205–2208, 1992.
[6] G. Sgro, G. Palmieri, and M. Russo, “A parallel implementation of the 2D FFT on a multiprocessor SHARC DSP system,” Proc. IEEE Int. Symp. Circuits and Systems (ISCAS), vol. 5, pp. 453–456, 1999.
[7] A. Ercan, “High-speed 2D FFT implementation using SHARC DSPs,” Proc. IEEE Int. Conf. Signal

Processing Applications and Technology (ICSPAT), pp. 422–427, 2001.
[8] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, Discrete-Time Signal Processing, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1999.
[9] A. S. Tanenbaum and T. Austin, Structured Computer Organization, 6th ed. Pearson, 2012. (Only if memory hierarchy is cited in the original)
[10] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput., vol. 19, no. 90, pp. 297–301, Apr. 1965. (Original FFT)
[11] Datasheet, "FFT Megacore Function User Guide", Altera Ltd., 2002.
[12] Datasheet, "Xilinx 1024-point FFT/IFFT Core Datasheet", Xilinx Ltd., 2002.
[13] Datasheet, "CS248 FFT/IFFT Core Datasheet", Amphion Ltd., 2002.
[14] Datasheet,"FFT/WinFFT/Convolver Transform", Mentor Graphics, 2002.
[15] URL: www.celoxica.com.
[16] Datasheet, "RC1000 Reconfigurable hardware development platform", Celocixa Ltd., 2001.
[17] E.O. Brigham, "The Fast Fourier Transform and its Application", Prentice Hall, 1988.
[18] C.S. Burrus and T.W. Parks, "DFT/FFT and Convolution Algorithms", Wiley, New York, 1985.
[19] R.C. Gonzales and R. E. Woods. "Digital Image Processing", Addison-Wesley, 2002.
[20] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," in Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, Apr. 1995, pp. 155–162.
[21] C. Dick, "Reconfigurable Architectures for 2D-DSP: The Impact of Polynomial Transforms," in Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines, Apr. 1996, pp. 202–210.
[22] T. Dillon and J. McAllister, "A High-Speed Parallel 2-D FFT Processor using Xilinx Virtex-II FPGAs," in Proceedings of the International Conference on Field Programmable Logic and Applications, 2003, pp. 443–452.
[23] A. Ercan, "Parallel Real-Time 2-D FFT Implementation on Multiprocessor SHARC DSP Platform," IEEE Transactions on Consumer Electronics, vol. 47, no. 2, pp. 232–239, May 2001.
[24] R. Cavadini, R. Scrofani, and R. Andraka, "A 1024×1024 Real-Time Frequency-Domain Engine Based on a VLSI FFT Core," in Proceedings of the 1995 IEEE Workshop on Signal Processing Systems, 1995, pp. 121–126.
[25] R. Hartley, J. Brown, and M. Meyer, "Real-Time FFT Processing on TMS320C40 DSP-Based Multiprocessor Systems," in Proceedings of the 1993 International Conference on Acoustics, Speech, and Signal Processing (ICASSP), vol. 4, Apr. 1993, pp. 488–491.
[26] A. Sgro, "High Performance DSP Systems using SHARC Multiprocessors for Real-Time 2-D FFT Computation," in Proceedings of the 2000 IEEE Signal Processing Workshop on DSP Systems Design and Implementation, Oct. 2000, pp. 81–86.
[27] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 2nd ed., Prentice Hall, 2022.