

Parallel Pipelined Hardware Acceleration of Fast Fourier Transforms on FPGA

Simi Zerine Sleeba¹, Karthika B R¹, Krishna Sudhan¹, Lakshmi Priya Unni¹, Lin John¹

¹Department of Electronics and Communication Engineering,
Rajagiri School of Engineering and Technology,
Kochi, India

Article Info

Article history:

Received : 22 September 2024

Revised: 09 November 2024

Accepted: 05 December 2024

Keywords:

Fast Fourier Transform
Hardware accelerator
Block RAM
Field Programmable Gate Array
Parallel Pipelining

ABSTRACT

The Fast Fourier Transform (FFT) is widely used in digital signal processing applications and particularly for implementing convolution operation for real-time object detection using CNN. This paper proposes an efficient hardware architecture for Radix-2 FFT computation, implemented on an FPGA, employing multiple parallel and pipelined stages of butterfly units. The proposed architecture utilises Block RAM to store inputs and twiddle factor values to compute the transform. The hardware for the proposed architecture is synthesised on a Zync Ultrascale FPGA and its performance is evaluated using parameters such as critical path delay, throughput, device utilisation and power consumption. The performance of the proposed parallel pipelined architecture for 8 point FFT, measured in FFTOPS, is found to be 67% higher than the non-pipelined architecture. Performance comparison with the state-of-the-art parallel pipelined methods confirm the acceleration achieved by the proposed FFT architecture. A comprehensive comparison of the proposed hardware with the synthesised version of the FFT IP core bundled with the Vivado Design suite is also presented in the paper.

Copyright © 2024 Institute of Advanced Engineering and Science.
All rights reserved.

Corresponding Author:

Simi Zerine Sleeba
Department of Electronics and Communication Engineering,
Rajagiri School of Engineering and Technology,
Kochi, India.
Email: simizs@rajagiritech.edu.in

1. INTRODUCTION

In the rapidly evolving landscape of signal processing, the demand for efficient and high-performance implementations of signal transforms is growing exponentially. Signal transforms are integral to many applications spanning telecommunications, image, video and audio processing, necessitating efficient hardware solutions for enhanced system performance and real-time processing. The Fast Fourier Transform (FFT) plays a pivotal role in signal processing tasks due to its efficiency and versatility. The rising complexity of Fast Fourier transforms requires optimized hardware solutions. The widespread use of this transform underlines the importance of its efficient hardware implementations.

Fourier Transform provides a mathematical technique for representing time-domain signals in the frequency domain. For discrete and periodic signals, the one dimensional Discrete Fourier Transform (DFT) is computed as per Equation 1.

$$X[k] = \sum_{n=0}^{N-1} x[n].e^{-i\frac{2\pi kn}{N}} \quad (1)$$

While the DFT accurately transforms signals, its computational complexity of $O(N^2)$ renders it unsuitable for large signal transformations. Fast Fourier Transform (FFT) is an algorithm which reduces the computational complexity of DFT to $O(N \log N)$, using a divide and conquer strategy. Optimizing Fast Fourier Transform (FFT) algorithms for high-performance computing presents several challenges. The basic Radix-2 FFT computation uses a 2 point butterfly structure, which is the most simple and scalable unit in terms of hardware implementation. The scope for accelerating FFT operation lies in exploiting parallelism and pipelining techniques in hardware implementation. However, achieving high throughput while minimizing resource utilization and power consumption remains a significant challenge. This paper addresses these challenges by proposing a novel parallel and pipelined architecture for FFT on FPGA hardware, aiming to significantly improve the speed and efficiency of FFT implementations.

The rest of the paper is organised as follows. Sections 2. and 3. give an account of the previous work related to algorithms and hardware implementation of FFT and the motivation behind this paper. Section 4. explains the proposed architecture in detail. Sections 5. and 6. cover the methodology used for implementation and analysis and a detailed discussion of the results. In Section 7. , we conclude the paper with a few pointers towards future work.

2. RELATED WORK

A survey of previous work covers FFT algorithms, hardware architectures, memory access techniques, Address Generation Units (AGU), and pipelining methods. A first set of papers propose hardware modelling of several FFT algorithms such as Cooley-Tukey, Good-Thomas, Radix-2 and Rader algorithms using Verilog hardware description language and implementation on Xilinx FPGA [1], [2]. The results revealed that the Radix-2 FFT method utilized the fewest number of slices, while the Good-Thomas method exhibited better performance. In terms of flip-flop utilization, both the Cooley-Tukey and Good-Thomas approaches utilized fewer numbers compared to the Radix-2 and Rader approaches. The Rader method had the lowest operating frequency among all proposed FFT approaches on FPGA. Additionally, it was observed that the FPGA area utilization increased with the number of FFT points for all methods.

In one of the hardware implementations, a memory-based recursive FFT design is proposed, where memory access is evenly distributed to enhance the utilization efficiency of SRAM ports [3]. Their design requires fewer butterfly iterations, resulting in reduced gate count and power consumption. Another implementation for FFT and IFFT using Xilinx's Virtex2P FPGAs and VHDL reports that it takes $0.6 \mu s$ to obtain the FFT result and $0.72 \mu s$ for the IFFT result [4]. The fact that both FFT and IFFT implementations take less than 1 ms is crucial for real-time applications. A similar hardware realisation introduces a parallel memory access technique which computes FFT by employing a permutation for each b-tuple's elements at the butterfly output [5]. This technique leads to improvements in the data and twiddle address generation circuit, the interconnection, and the control, compared to previously published results. The improvement is twofold: first, in terms of transistor count, and second, in the delay induced by the address generation and control circuits. Moreover, the resulting architecture can be effectively utilized in applications with mixed radix requirements, especially in cases where the radices are powers of 2, as well as in continuous flow organizations. Another work presents an FFT computing method that reduces memory usage for storing twiddle factors by approximately 50 % by storing smaller twiddle factors in SRAM while keeping signal data in external memory space [6]. This method enhances computing efficiency by minimizing cycle stalling due to cache misses in data memory.

In a recent work, 3D-FFT operation is accelerated by incorporating Heterogeneous Embedded Blocks (HEBs) in FPGAs [7]. The paper provides an estimate of the number of HEBs to be embedded and also offers an approach to develop effective HEBs. In the next few papers being reviewed, FFT pipelined architectures with Address Generation Unit (AGU) are presented [8], [9], [10]. One such work presents a radix-2 multipath delay commutator FFT pipelined architecture for computing real-valued FFT and Hermitian-symmetric IFFT signals using only real datapaths. The generated outputs are in natural order. By integrating the two FFT processors and registers, the need for a bit reversal circuit, as presented in prior designs, is eliminated. Transferring the twiddle factor to subsequent stages transforms the real structure of the FFT. These features make the FFT

processor superior in terms of throughput and hardware complexity. The next paper presents the architecture of an Address Generation Unit (AGU) designed for audio DSP applications, which is capable of automatically calculating butterfly input/output data addresses for FFT operations [11]. This paper reports a 73 % reduction in size compared to the prior FGAU architecture proposed previously.

A large number of articles related to FFT hardware implementation can be broadly classified as serial and parallel pipelining models [21]. Serial pipelined FFT architectures use delay feedback loops, delay commutators or feed forward mechanism and deliver an output of one data per clock cycle [22]. On the other hand, parallel pipelined FFT architectures process N data per clock cycle, where $N > 1$ [23]. Majority of the parallel pipelined architectures are unfolded versions of serial pipelined counterparts [24]. All of these models use rotators to shuffle the inputs prior to a pipeline stage and their implementations report optimum resource utilisation and reasonable speed of operation. The radix-2 DIF algorithm is obtained by performing trivial rotation at each odd stage and then moving data to the following stage [26]. Some articles report that the hardware utilization based on the digit-serial approach is higher than that of the bit-parallel implementation, whereas the critical path of the digit-serial multiplier is smaller. As multiplication is often the critical operation in the complete FFT architecture, the digit-serial-based FFT architecture can operate at a faster speed [13]. Twiddle factor for FFT computation is stored in ROM and the digit-serial architecture not only reduces the access rate of the ROM but is also suitable for ROM reduction techniques. Parallel pipelined FFT architectures without using multiplier hardware also claim design optimisation and enhanced performance [14].

Conventionally, two dimensional FFT is computed by first computing one dimensional FFT of each row of the image and then a 1-D FFT on each column of the row transforms [27]. By integrating more advanced algorithms for row column decomposition, significant improvement in computation speed is achieved [28].

3. MOTIVATION

We apply the Convolutional Neural Networks (CNN) algorithm for an object detection application with real-time video input. Convolution is the most important mathematical operation performed in each layer of CNN. The Fast Fourier Convolution Network (FFCN) is a type of neural network that uses the Fast Fourier Transform (FFT) to speed up the computation of convolutions, making CNNs more efficient [31]. FFT based convolution provides an advantage in computational cost by reducing potential calculation complexity compared to conventional methods like general matrix multiplication and Winograd algorithms [28]. Convolution operation is performed for each layer of the CNN, by computing the two-dimensional FFT of the input image and a filter separately and multiplying the resultant matrices in the final step. The 2-D FFT of input image is computed by decomposing into two arrays of 1-D DFTs. Then, 1-D FFT of each row is computed followed by 1-D FFT on each column of the row transforms [30].

The objective of our FFT hardware implementation is to maximize throughput so as to achieve real-time object detection. Hence, we eliminate all feedback and feed-forward loops proposed in previous work reviewed in Section 2. and exploit the inherent parallelism of FPGA hardware to the maximum extent. Since FPGAs enable hardware design through software-controlled configuration, improving circuit design simply involves modifying, debugging, and downloading the new configuration code in a short time. Given that FFT and IFFT algorithms require high computational density and consume significant time in real-time applications, FPGA implementations offer a convenient solution. From the review of the current state-of-the-art, many gaps have been identified as follows:

- Lack of a parallel pipelined architecture for FFT which utilises the parallelism offered by FPGA to maximise the throughput and operation speed.
- Limited research on reducing memory usage for storing twiddle factors while considering cache operations.
- Absence of detailed studies on the effectiveness of Address Generation Units (AGUs) in optimizing FFT architectures.

The proposed work presents the implementation of an efficient FFT algorithm on FPGA which maximises throughput and operating speed while achieving optimum resource utilization and memory access by employing a parallel pipelined architecture. Further, the suitability of the proposed architecture for real-time

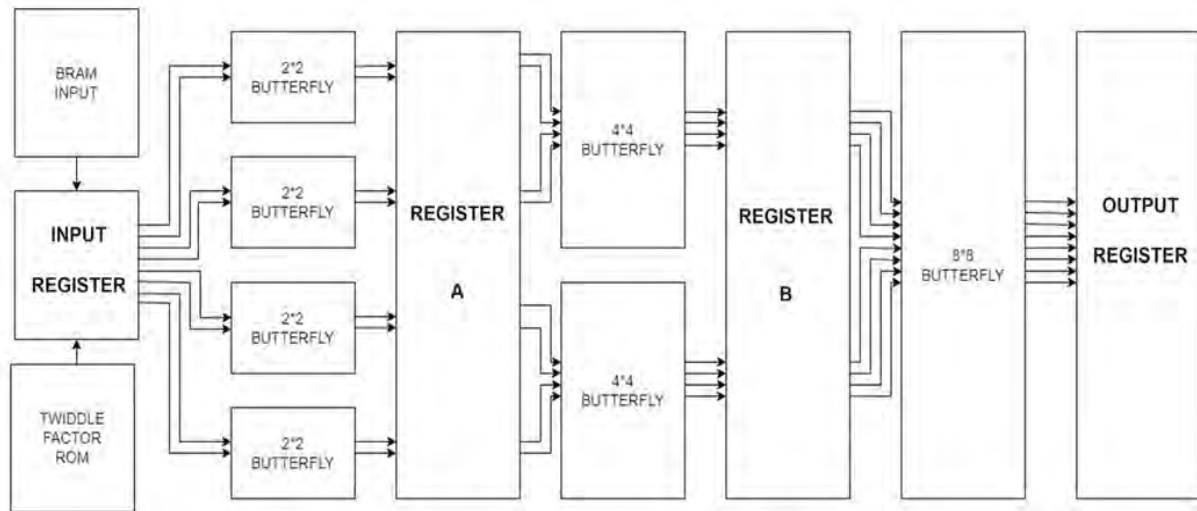


Figure 1. Block diagram of proposed parallel pipelined architecture for 8 point FFT

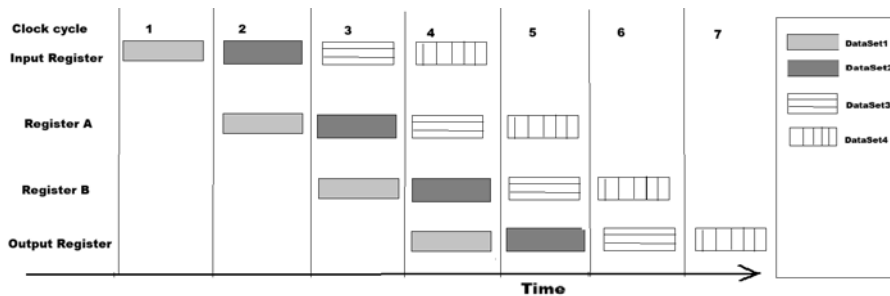


Figure 2. Timing diagram showing the progress of 4 sets of 8 point FFT computation through the pipeline registers in the proposed architecture

object detection application is analysed by comparing the performance parameters with a recently proposed multipath serial commutator technique and with the FFT IP Core provided by Vivado tool.

4. PROPOSED FFT ARCHITECTURE

We adopt the radix-2 FFT computation using Decimation in Time (DIT) method, because of the simplicity of the radix-2 butterfly unit. The DIT method requires the inputs to be fed in bit reversed order, while the outputs are obtained in natural order [15],[18]. The proposed architecture is explained using a comprehensive block diagram of 8 point FFT computation as shown in Figure 1.

The architecture is divided into three pipelined stages separated by registers. The eight input values, each with a real and imaginary component, are stored as external source in a configuration file (.coe file) and are stored into a Block RAM (BRAM). The input data is serialized and bit-reversed before being fed into the first stage as parallel inputs. The first stage, Stage 1 consists of four 2x2 butterfly modules, which processes the inputs and produce intermediate outputs which are stored in Register A for further processing. In Stage 2, the input data is read from Register A and processed by two 4x4 butterfly units, with output values stored in Register B, which serve as input for Stage 3. In the final stage ie., Stage 3, the output values from Register B are processed by a single 8x8 butterfly unit. The processed data from the 8x8 butterfly stage is collected in the output register, representing the final FFT output. The timing diagram showing the progress of four sets of input data through the pipeline registers is shown in Figure 2. This progressive implementation approach using registers as buffers between stages helps to synchronise dataflow and ensure data availability for processing. By increasing the number of stages, the proposed model can be scaled to larger FFT sizes, with increased

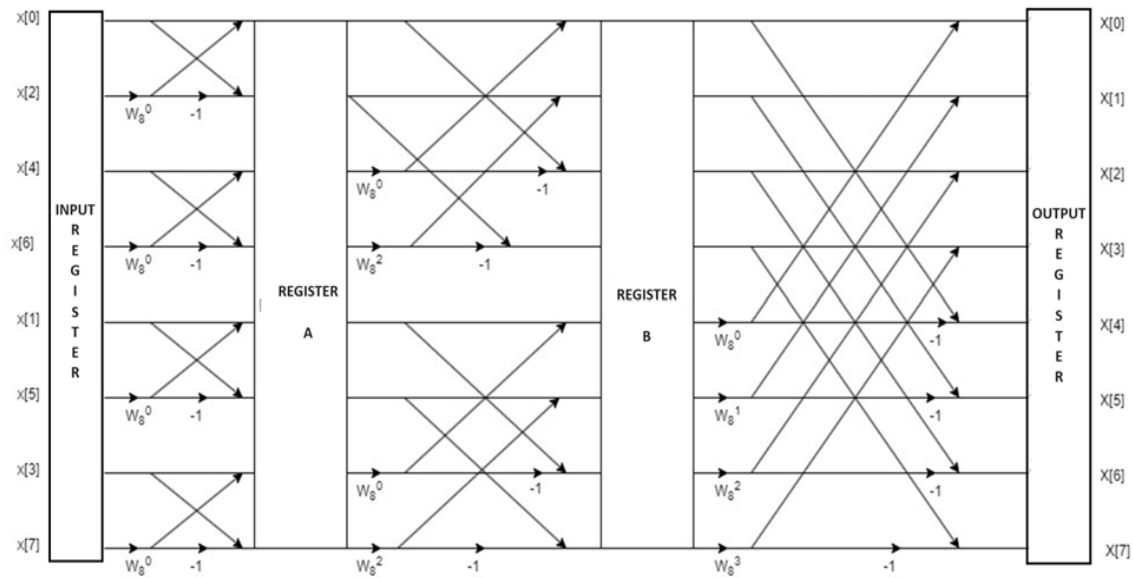


Figure 3. Parallel pipelining for 8 point FFT

resource utilisation. However, the input image in the target application is segmented into tiles such that two dimensional 8 point FFT can be performed on each tile and segment FFTs are combined together to obtain the convolution result. Since this paper focuses on the hardware acceleration of FFT on FPGA, tile segmentation and convolution of images is out of the scope of the paper and is reserved for future work.

4.1. Parallel Pipelining

To accelerate FFT computation time and use hardware resources efficiently, a parallel-pipelined architecture is used in this work as shown in Figure 3. Pipelining improves the throughput by allowing multiple sets of computations to progress simultaneously. The initial inputs are stored in the BRAM. In our proposed architecture, as soon as the output of Stage 1 becomes available in Register A, the second set of input data is fed into the Stage 1 input buffers. The computation delay in each stage is minimised using multiple butterfly units which process inputs in parallel. Parallelism in the architecture ensures that the outputs are generated at roughly the same time in each stage. Each of the three pipelined stages in the proposed architecture take exactly one clock cycle for completing its processing and storing the output in corresponding register. As a result of pipelining, in each clock cycle, three set of FFT computations will be progressing through stage 1, stage 2 and stage 3 simultaneously. Unlike the traditional FFT implementation, which delivers FFT output after every three clock cycles, the proposed architecture delivers an output set at the end of each clock cycle after an initial delay of three clock cycles.

There are several components that constitute the architecture of the FPGA-based FFT accelerator which are discussed below:

4.2. Block RAM (BRAM)

The input values and twiddle factor values are stored in Block RAM. The values are loaded into BRAM by storing them as a .coe file. A .coe-file is a text-based file containing a header and initialization data (usually in hexadecimal or binary format) for the BRAM.

4.3. Twiddle Factor ROM

Disabling the write enable in a BRAM converts the BRAM into a ROM, where we store the Twiddle factors. These values can be computed while processing using specialized algorithms like CORDIC [19],[20], or stored as a LUT. Similar to the input values the twiddle factors are also stored as an external file .coe in the BRAM in IEEE 754 32-bit fixed point format. Twiddle factor values are accessed using a 3 bit counter

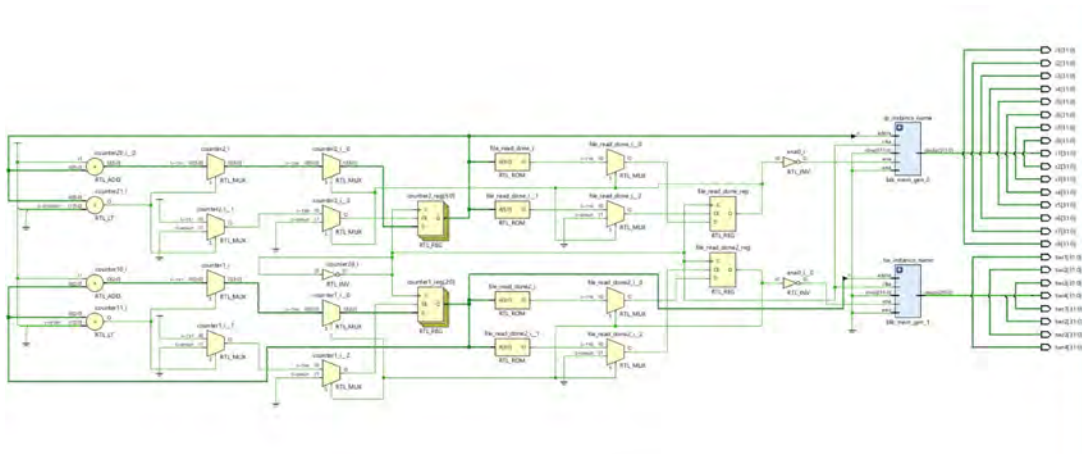


Figure 4. Schematic of address counter for reading input values and twiddle Factor from BRAM

which varies from 0 to 7. This counter controls the addressing of twiddle factor values stored in BRAM, ensuring that the appropriate twiddle factor is fetched and applied at each stage of the FFT computation. The sequential access pattern simplifies the control logic and enables efficient utilization of twiddle factors without unnecessary overhead.

4.4. Registers

After each butterfly operation, the resultant values need to be stored temporarily before they are used for further computations. The Register A and Register B act as intermediate storage for the processed data before further processing.

4.5. Address Counter

Counters are used to increment the address of input values and read them sequentially. For 8 point FFT computation, a real value and an imaginary value corresponding to 8 data points will be stored in 16 consecutive input locations in BRAM. A flag is employed until which the counter increments at each clock. When the flag is raised i.e., if counter is greater than 15, reading will stop and all inputs will be assigned to 16 individual ports. These ports are later assigned to the first set of input registers. Twiddle factors are also read in a similar manner. Schematic diagram of address counter generated from synthesis of Verilog code is shown in Figure 4.

4.6. Fixed Point Arithmetic Units

In our proposed method, arithmetic operations of FFT are done using fixed point numbers. Fixed point numbers are preferred over floating point numbers because comparatively, floating point numbers use more hardware resources, area and have a higher latency [13]. Also, floating point twiddle factor values cannot be stored as a .coe file and loaded into the BRAM. The input values and the twiddle factor values are stored in 32 bit IEEE 754 fixed point format, where the first bit represents the sign bit, the next 8 bits represent the exponent, which determines the position of the binary point, and the remaining 23 bits represent the mantissa, which holds the fractional part of the number. In applications where floating-point arithmetic may be impracticable or resource-intensive, IEEE 754 fixed-point representation offers a flexible and effective way to express numerical values in binary form.

4.7. Fixed Point Adder

It is used both as an adder and a subtractor. The decision to add/subtract is given as an external control or decided internally using an operand. There exist four cases of input signs and extra checks to determine the output sign bit. For addition, the significant bits of the result is calculated by adding the significands of both the operands and normalize if necessary and the exponent is determined by the exponent of the operand with the larger exponent value. For subtraction, the two's complement of the second operand's significand is added to the significand of the first operand and the exponent is determined using the priority encoder. The final result is obtained by combining the sign bit adjusted significand and exponent. The butterfly unit's adder stage makes

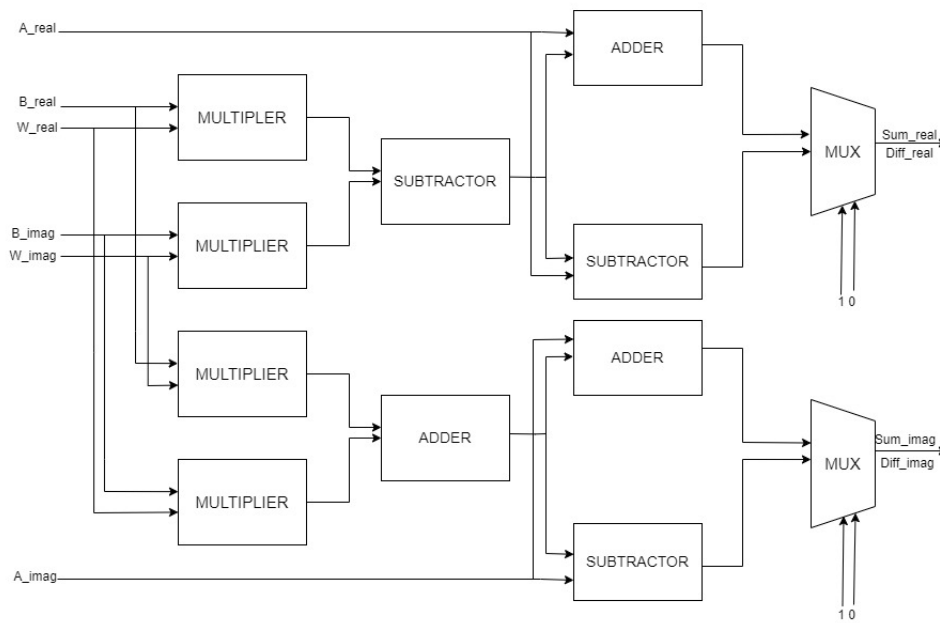


Figure 5. Schematic diagram of Butterfly Unit

it possible to compute the intricate arithmetic operations needed for FFT transformations. The adder aids in the precise and effective computation of the FFT method by carrying out addition and subtraction operations on the products acquired from the multiplication step.

4.8. Fixed Point Multiplier

The multiplier module undertakes a series of sequential steps to compute the product of two input operands. Initially, it determines the signs of the result based on the signs of the input bits and computes an exception flag based on the exponents of the operands. Following this, the module proceeds to normalize the operands by shifting their mantissas and adding the hidden bits. Subsequently, it calculates the product of the normalized operands, and the result is checked for normalization. The mantissa of the product is then computed by adding the mantissa bits and a rounding bit. Concurrently, the module computes the sum of the exponents and adjusts the final exponent according to normalization. Additionally, it sets a zero flag if any of the operands are zero, and the overflow and underflow conditions are set based on the exponents and zero flag. Finally, the final result is constructed based on the flags and the calculated values, thus completing the multiplication process. This systematic approach ensures accurate and efficient computation of the product while adhering to IEEE 754 standards.

4.9. Butterfly Unit

The Butterfly Unit is the basic processing unit of the FFT and is crucial to the overall functioning of the transform. It involves combining pairs of data points from different stages of the computation. These operations involve the application of twiddle factors (complex coefficients) to the data. It's called a "butterfly" because of the way data points move and interact during the operation, resembling the shape of a butterfly's wings. It exploits the symmetry and periodicity properties of sinusoidal functions. The butterfly unit is implemented as a modular component in the hardware description language. The following equations are used as the logic of the butterfly module:

$$\begin{aligned} \text{Sum_real} &= A_real + (B_real \times W_real - B_imag \times W_imag) \\ \text{Sum_imag} &= A_imag + (B_imag \times W_real + B_real \times W_imag) \\ \text{Diff_real} &= A_real - (B_real \times W_real - B_imag \times W_imag) \\ \text{Diff_imag} &= A_imag - (B_imag \times W_real + B_real \times W_imag) \end{aligned}$$

Name	Value	999,996 pa	999,998 pa	1,000,000 pa
r1[31:0]	0000000000	00000000000000000000000000000001		
i1[31:0]	0000000000	00000000000000000000000000000010		
r2[31:0]	0000000000	00000000000000000000000000000011		
i2[31:0]	0000000000	00000000000000000000000000000100		
r3[31:0]	0000000000	00000000000000000000000000000101		
i3[31:0]	0000000000	00000000000000000000000000000110		
r4[31:0]	0000000000	00000000000000000000000000000111		
i4[31:0]	0000000000	00000000000000000000000000001000		
r5[31:0]	0000000000	00000000000000000000000000001001		
i5[31:0]	0000000000	00000000000000000000000000001010		
r6[31:0]	0000000000	00000000000000000000000000001011		
i6[31:0]	0000000000	00000000000000000000000000001100		
r7[31:0]	0000000000	00000000000000000000000000001101		
i7[31:0]	0000000000	00000000000000000000000000001110		
r8[31:0]	0000000000	00000000000000000000000000001111		
i8[31:0]	0000000000	00000000000000000000000000010000		
twr1[31:0]	0011111110	00111111000000000000000000000000		
twi1[31:0]	0000000000	00000000000000000000000000000000		
twr2[31:0]	0011111100	001111100110100111110111101000		
twi2[31:0]	0011111100	001111100110100111110111101000		
twr3[31:0]	0000000000	00000000000000000000000000000000		
twi3[31:0]	1011111110	10111111000000000000000000000000		
twr4[31:0]	1011111100	101111100110100111110111101000		
twi4[31:0]	1011111100	101111100110100111110111101000		
counter2[5:0]	000001	000001		
counter1[2:0]	001	001		
clk_internal	1			

Figure 6. Input (Real and Imaginary) values and Twiddle factor fixed point values read from BRAM and assigned to ports of input register.

The block schematic of the butterfly unit is shown in Figure 5. The adder/subtractor and multiplier units shown in the schematic perform fixed point arithmetic operations as explained in sections 4.7. and 4.8..

5. METHODOLOGY USED

The hardware of the proposed FFT accelerator is modeled in Vivado Design Suite by developing Verilog code. Each of the functional units such as fixed point adder/subtractor, fixed point multiplier and butterfly units are designed using Verilog Hardware Description Language (HDL) and a top module is used to create multiple instances of the same unit so as to enable parallel pipelined processing of data. The Vivado Design tool runs the experimental setup for the proposed architecture. Hardware synthesis is carried out with Zynq Ultrascale+ MPSoC as the target device, which is a ready-to-use embedded software and digital circuit development board. The implementation utilizes a .coe file to load input values for FFT computation into the BRAM. The address counter which reads the input values for 8 point FFT computation from BRAM and assigns them to the 16 input registers is also modeled in Verilog and synthesised using Vivado as shown in Figure 6. The synthesised hardware for the proposed accelerator for 8 point FFT is given in Figure 7. Figure 8 shows the waveforms of the 16 (real and imaginary) input values and twiddle factors read from BRAM and assigned to the input register ports for 8 point FFT computation. We also discuss the implementation of the FFT IP core available within Vivado and compare its performance with our custom implementation. Through detailed simulations and analysis, we evaluate the throughput, resource utilization, and processing speed of our FPGA-based FFT accelerator.

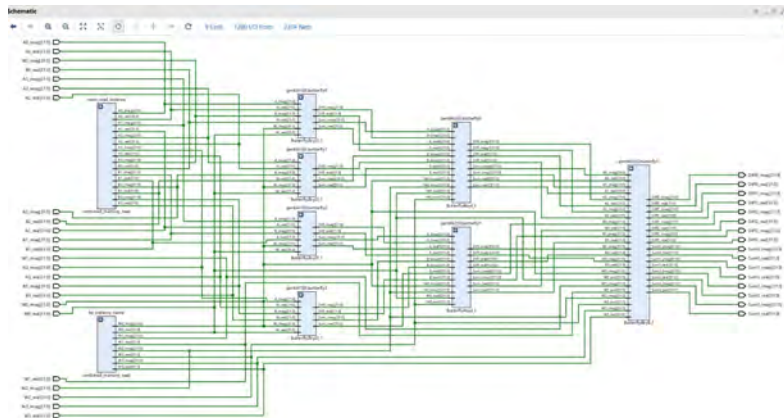


Figure 7. Schematic of the proposed hardware accelerator for 8 point FFT computation

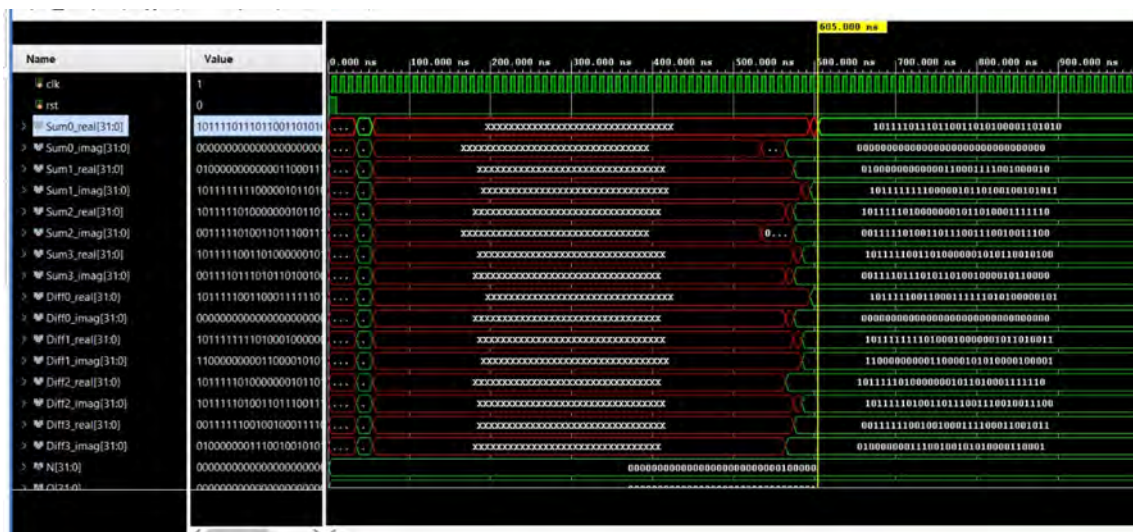


Figure 8. 8 point FFT computation result for one set of data

6. ANALYSIS OF RESULTS

Parameters	FFT IP Core	Proposed FFT accelerator
Critical Path delay	1200 ns	605 ns
Utilisation		
BRAM:	1	1
LUT:	4623	803
Flipflop:	6288	789
On chip Power	34.4 W	3.06 W
Throughput	426 Mbps	846 Mbps

Table 1. Comparison of synthesis parameters for 8 point FFT

The hardware for the proposed accelerator is synthesised and the schematic obtained for 8 point FFT is as shown in Figure 7. The synthesis results obtained for one set of data from Vivado is shown in Figure 8. Table 1 summarises the device utilisation, critical path latency and power consumption obtained from the synthesis report. The critical path delay of the proposed hardware is 605 ns, whereas synthesis of FFT IP core provided by Xilinx Vivado 2022.2 (LogiCORE) reports a delay of 1200ns. The versatile hardware architecture of IP core which accomodates fixed point and floating point interfaces accounts for increased hardware and

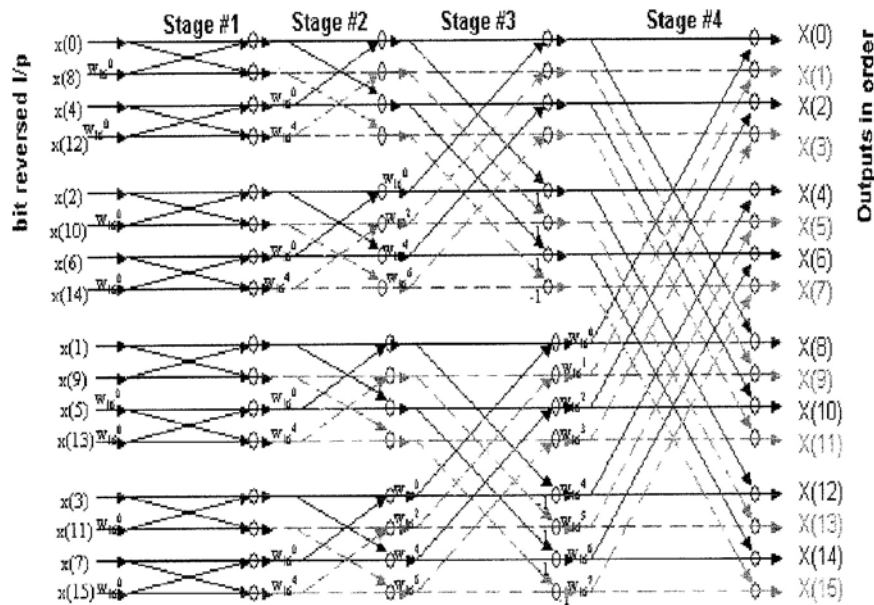


Figure 9. Butterfly diagram for 16 point FFT computation

longer delay of IP core in comparison with our hardware which is optimised for fixed point FFT computation. The proposed hardware uses 1 BRAM, 803 LUTs and 789 flipflops. The total on chip (static and dynamic) power consumption is 3.06 W. The proposed FFT uses parallelism and pipelining to its advantage for reducing the computation time and optimizing the algorithm.

6.1. Performance Evaluation

The efficiency of the proposed FFT accelerator is evaluated by calculating FFT operations completed per second (FFTOPS). At a frequency of operation of 40 MHz, the performance of the proposed hardware for 8 point FFT is obtained as 1,652,892 FFTOPS, which is 67% higher compared to FFT without pipelining. Unlike the sequential manner of conventional FFT computation, our proposed hardware uses parallelism and 3 stage pipelining, which accounts for the performance acceleration. Multipath Serial Commutator (MSC) model is the most superior parallel pipelined FFT technique found in literature, which has been designed for any FFT size and radix [26][25]. MSC architecture uses a large number of butterfly units, rotators and permutation units at the intermediate stages which increase the critical path delay and resource utilisation, and thereby lowering the throughput. Our proposed model eliminates rotators and permuter units and the same functionality is achieved by suitably assigning inputs from intermediate buffers to 2x2 butterfly units. Our implementation shows higher resource utilisation with the advantage of higher throughput required for real-time object detection.

6.2. Scalability Evaluation

The proposed architecture can be scaled to compute higher order FFTs. For example, 16 point FFT can be computed by introducing a fourth stage, Stage 4 before the output register as shown in Figure 9. Since the output computation in butterfly units of each stage takes place in parallel, the increased delay caused by the addition of a pipeline stage in 16 point FFT is only 25% higher than that of 8 point FFT.

7. CONCLUSION

The FPGA-based hardware accelerator presented in this paper demonstrates the realization of a high performance FFT computation, which can be utilised in CNN for real-time object detection. Through the arrangement of several butterfly units as various parallel pipelined stages, including BRAM, memory registers, adder, and multiplier units, we develop a robust architecture that can accelerate FFT computation. By evaluating parameters like critical path delay, throughput, power consumption, and resource utilization, we evaluate the effectiveness of our FPGA-based FFT accelerator. The proposed hardware can be easily scaled to compute

higher point FFTs like 16,32,64 and so on, with the addition of pipeline stages. Evaluation of the acceleration of CNN which implements the proposed FFT accelerator remains as a future challenge.

8. ACKNOWLEDGEMENT

This work is partially supported by the Chip to Startup project of Ministry of Electronics and IT, Government of India.




REFERENCES

- [1] A. Chahardahcherik, Y. S. Kavian, O. Strobel, and R. Rejeb. (2011) 'Implementing FFT algorithms on FPGA', *International Journal of Computer Science and Network Security*, Vol. 11, No. 11, pp.148—156
- [2] W. Cooley and J. W. Tukey(1965) 'An algorithm for the machine calculation of complex Fourier series', *Mathematics of Computation*, Vol. 19, No. 90, pp.297–301
- [3] K. Harikrishna, T. R. Rao, and V. A. Labay (2011) 'FPGA implementation of FFT algorithm for IEEE 802.16 e (mobile wimax)', *International Journal of Computer Theory and Engineering*, Vol. 3, No. 2, pp.197–203.
- [4] I. Az, S. Sahin and M. A. Cavuslu (2007) 'Implementation of Fast Fourier and Inverse Fast Fourier Transforms in FPGA', *Proceedings of 3rd International Symposium on Electrical, Electronic and Computer Engineering*, pp. 7–10.
- [5] V. Kitsakis, K. Nakos, D. Reisis, and N. Vlassopoulos (2018) 'Parallel memory accessing for FFT architectures', *Journal of Signal Processing Systems*, Vol. 90, No. 11, pp.1593–1607.
- [6] T.Y. Sun and Y.H. Yu (2009) 'Memory usage reduction method for FFT implementations on DSP based Embedded System', *Proceedings of IEEE 13th International Symposium on Consumer Electronics*, pp.812—815.
- [7] B. S. C. Varma,, K. Paul, and M. Balakrishnan (2013) 'Accelerating 3D-FFT using hard embedded blocks in FPGAs', *Proceedings of the 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*, pp.92–97.
- [8] G. P. Kumar, M. S. Chandra, K. S. Prasanna, and M. Mahesh (2021) 'Parallel memory accessing for FFT architectures', *Journal of Signal Processing Systems*, Vol. 90, No. 11, pp.1593–1607.
- [9] I.S.Uzun, A.Amira and A.Bouridane (2005) 'FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing', *IEE Proceedings - Vision, Image, and Signal Processing*, Vol. 152, No. 3, pp. 283–295.
- [10] R.V.Benadikar and V. Jayasree (2018) 'Comparative Study Of FPGA Implementation Of Parallel 1-D FFT Algorithm Using Radix-2 And Radix-4 Butterfly Elements', *IOSR Journal of VLSI and Signal Processing*, Vol. 8, No. 4, pp. 23–47.
- [11] J. Y. Kim and M. H. Sunwoo (2004) 'Design of Address Generation Unit for audio DSP', *IEEE International Symposium on Intelligent Signal Processing and Communication Systems*, pp. 616–619.
- [12] Z. Kaya, M. Garrido and J. Takala (2023) 'Memory-Based FFT Architecture With Optimized Number of Multiplexers and Memory Usage', *IEEE Transactions on Circuits and Systems-II: Express Briefs*, Vol. 70, No. 8, pp.3084–3088.
- [13] Y.N. Chang and K. K. Parhi (2003) 'An efficient pipelined FFT architecture', *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 50, No. 6, pp. 322–325.
- [14] P.K.Godi, B.T.Krishna and P. Kottipalli (2020) 'Design optimisation of multiplier-free parallel pipelined FFT on field programmable gate array', *IET Circuits, Devices and Systems*, Vol.14, No. 7, pp. 995-1000.
- [15] M.D. Hameed Pasha and P. Radhika (2018) 'A Pipelined FFT Architecture to Process Two Independent Data Streams', *Journal of Emerging Technologies and Innovative Research*, Vol.5, No. 1, pp.802-807.
- [16] S. Josue Saenz, J. J. Raygoza P., E. C. Becerra A., S. O. Cisneros and J. R. Dominguez, 'FPGA design and implementation of radix-2 Fast Fourier Transform algorithm with 16 and 32 points', 2015 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), Mexico, 2015, pp. 1-6.
- [17] A. Changela, M. Zaveri and D. Verma, 'FPGA implementation of high-performance, resource-efficient Radix-16 CORDIC rotator based FFT algorithm', *Integration*, Vol. 73, pp. 89-100.
- [18] M.S.Gilan and B. Maham, 'Optimized power and speed of Split-Radix, Radix-4 and Radix-2 FFT structures'. *EURASIP Journal of Advanced Signal Processing*, 81(2024),
- [19] H.N. Nguyen, S.A. Khan, C. Kim and J.Kim (2018) 'A pipelined processor using an optimal hybrid rotation scheme for complex multiplication: Design, FPGA implementation and Analysis', *Electronics*, Vol.7, No. 8, pp. 137
- [20] A.S. Raghuvanshi, A. Kumar and S. Gavel (2021) 'Throughput Radix8 Based FFT Architecture', *Proceeding of Fifth International Conference on Microelectronics, Computing and Communication Systems*, Lecture Notes in Electrical Engineering, vol 748. Springer, Singapore.
- [21] M. Garrido(2022). 'A Survey on Pipelined FFT Hardware Architectures', *Journal of Signal Processing Systems*, 94(11), 1345-1364.
- [22] Carl Ingemarsson and Oscar Gustafsson (2018). 'SFF—The Single-Stream FPGA-Optimized Feedforward FFT Hardware Architecture'. *Journal of Signal Processing Systems*, 90(11),1583–1592.
- [23] Hsu, S. C., Shen-ju-huang, Chen., S. G., Lin, S. C. and Garrido, M. (2020). 'A 128-point multi-path SC FFT architecture'. *Proceedings of the IEEE International Symposium of Circuits and Systems* (pp. 1–5).
- [24] M. Garrido, S.J. Huang and S.G. Chen (2018). 'Feedforward FFT Hardware Architectures based on Rotator Allocation', *IEEE Transactions on Circuits and Systems Part I*, 65(2), 581–592.
- [25] Z. Kaya and M. Garrido (2024) 'Optimised 4 Parallel 1024-point MSC FFT', *IEEE Access*, Vol.12,84110-84121.
- [26] G.-T. Deng, M. Garrido, S.-G. Chen, and S.-J. Huang (2023), "Radix-2k MSC FFT architectures," *IEEE Access*, vol. 11, pp. 81497–81510, 2023
- [27] N. Shirazi, P.M. Athanas, and A.L. Abbott (1995). 'Implementation of a 2-D fast Fourier transform on an FPGA-based custom computing machine Field-Programmable Logic and Applications. Lecture Notes in Computer Science, vol 975.
- [28] J. S. Kim, C. -L. Yu, L. Deng, S. Kestur, V. Narayanan and C. Chakrabarti, 'FPGA architecture for 2D Discrete Fourier Transform based on 2D decomposition for large-sized data', *IEEE Workshop on Signal Processing Systems*, Tampere, Finland,121-126.

- [29] D. Jeon, M. Seok, C. Chakrabarti, D. Blaauw and D. Sylvester, 'A super-pipelined energy efficient subthreshold 240 MS/s FFT core in 65 nm CMOS', IEEE Journal on Solid-State Circuits, 47(1), 23– 34.
- [30] V.Nair, M. Chatterjee, N. Tavakoli, A.Siami Namin and C. Snoeyink(2020) 'Optimizing CNN using Fast Fourier Transformation for Object Recognition'.arXiv 234-239. 10.1109/ICMLA51294.2020.00046.
- [31] Y. Zhang, F. Li, H.Xu, X. Li, X.Jiang, S. Efficient Convolutional Neural Networks Utilizing Fine-Grained Fast Fourier Transforms. Electronics 2024, 13, 3765.

BIOGRAPHIES OF AUTHORS



Simi Zerine Sleeba    is presently working as Assistant Professor in the Department of Electronics and Communication Engineering at Rajagiri School of Engineering and Technology. She was awarded PhD in VLSI System Design from Cochin University of Science and Technology in 2018. She has over 20 years of teaching and research experience and has authored many research articles in international journals and conferences. Her main research interests are Multi-core computer architecture, Network-on-Chip, FPGA Based System Design, Thermal aware and Approximate arithmetic circuits. She can be contacted at simizs@rajagiritech.edu.in.



Karthika B.R. completed her B.Tech degree in Electronics and Communication Engineering from Rajagiri School of Engineering and Technology. Her areas of interest include FPGA based system design and digital signal processing. She can be contacted at email: karthikabr02@gmail.com.



Krishna Sudhan completed her B.Tech degree in Electronics and Communication Engineering from Rajagiri School of Engineering and Technology. Her areas of interest include Digital signal processing methods, VLSI system design and FPGA based design. She can be contacted at email: krishnasudhan2001@gmail.com.



Lakshmi Priya Unni completed her B.Tech degree in Electronics and Communication Engineering from Rajagiri School of Engineering and Technology. Her areas of interest include VLSI hardware accelerators and ASIC design. She can be contacted at email: lakshmi priya357@gmail.com.



Lin John completed her B.Tech degree in Electronics and Communication Engineering from Rajagiri School of Engineering and Technology. Her areas of interest include VLSI system design and digital signal processing on FPGA. She can be contacted at email: lynnjohn140@gmail.com.