FPGA-Based Radix-2 DIF FFT Implementation for 8-Point Transform

Objective

To implement Radix-2 Decimation in Frequency (DIF) algorithm for an 8-point Fast Fourier Transform (FFT) on an FPGA (Field-Programmable Gate Array).

Introduction

The hardware implementation of FFT approaches is a challenging issue where the digital signal processors (DSPs) and the field programmable gate array (FPGA) chips are two considering designing environments for implementing different schemes of FFT approaches. The FPGA devices provide fully programmable system on-chip environments by incorporating the programmability of programmable logic devices and the architecture of gate arrays.

Decimation-in-time (DIT) Fast Fourier Transform (FFT) algorithms represent a class of highly regarded computational techniques employed for efficiently calculating the discrete Fourier transform (DFT) of a given data sequence. DIT FFT algorithms are particularly well-suited for instances where the transform size adheres to a power-of-two form, denoted as N = 2^k, where 'k' represents an integer.
DIT FFT algorithms are known for their efficiency when applied to transform sizes conforming to a power-of-two structure, making adept use of the inherent binary nature of the decomposition.

Butterfly diagrams

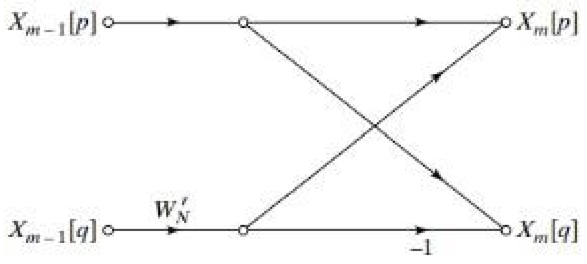
Quite similar to block diagrams, butterfly diagrams help us to understand the formula with the help of the diagrams and actions.
The needed functions in butterfly diagram:

The junction:

The point where two lines intersect is the junction where the outputs of those two lines add up and insert another output.

The coefficients:

To scale the input or multiply the input, a coefficient is written above the line, which simply indicates the multiplication of the complex number with the input present in the line.



Radix Method

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \qquad k = 0, 1, \dots, N-1,$$

The Radix-2 method for Fast Fourier Transform (FFT) calculation is a powerful technique that takes advantage of the inherent symmetry and periodicity of the patterns found within the coefficients of a discrete signal. This method offers a computationally efficient means of performing Fourier analysis, making it a cornerstone of various signal processing applications.

To comprehend the Radix-2 FFT algorithm, it is crucial to understand the underlying principles and key formulas involved:

1. Rearranging the Data:
The first step in Radix-2 FFT is to rearrange the input data in a way that reduces the complexity of the computation. This rearrangement separates the input series into two sets: one consisting of the even-indexed elements and the other of odd-indexed elements.

Formula:

$$X[k] = \sum_{n=0}^{N/2-1} x[2n] W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1] W_{N/2}^{kn} \qquad k = 0, 1, \dots, N-1,$$

The result of this rearrangement is that the original sequence, of length N, is divided into two sub-sequences, each with a length of N/2. This process continues recursively, dividing the data into smaller and smaller segments until we reach the core of the "divide and conquer" approach.

2. The "Divide and Conquer" Approach:

As we recursively divide the input sequence into smaller segments, the computation proceeds in a way that can be likened to a "divide and conquer" strategy. The FFT algorithm applies this technique to break down complex problems into simpler sub-problems.

Formula:

$$X[k] = G[((k))_{N/2}] + W_N^k H[((k))_{N/2}] \qquad k = 0, 1, \ldots, N-1.$$

$$G[k] = \sum_{r=0}^{(N/2)-1} g[r] W_{N/2}^{rk} = \sum_{\ell=0}^{(N/4)-1} g[2\ell] W_{N/2}^{2\ell k} + \sum_{\ell=0}^{(N/4)-1} g[2\ell+1] W_{N/2}^{(2\ell+1)k},$$

or

$$G[k] = \sum_{\ell=0}^{(N/4)-1} g[2\ell] W_{N/4}^{\ell k} + W_{N/2}^k \sum_{\ell=0}^{(N/4)-1} g[2\ell+1] W_{N/4}^{\ell k}.$$

Similarly, $H[k]$ can be represented as

$$H[k] = \sum_{\ell=0}^{(N/4)-1} h[2\ell] W_{N/4}^{\ell k} + W_{N/2}^k \sum_{\ell=0}^{(N/4)-1} h[2\ell+1] W_{N/4}^{\ell k}.$$

Each level of division introduces new series, each of which represents a portion of the original signal, and the process continues until we reach sequences of size 2, which are then combined to obtain the final FFT results.

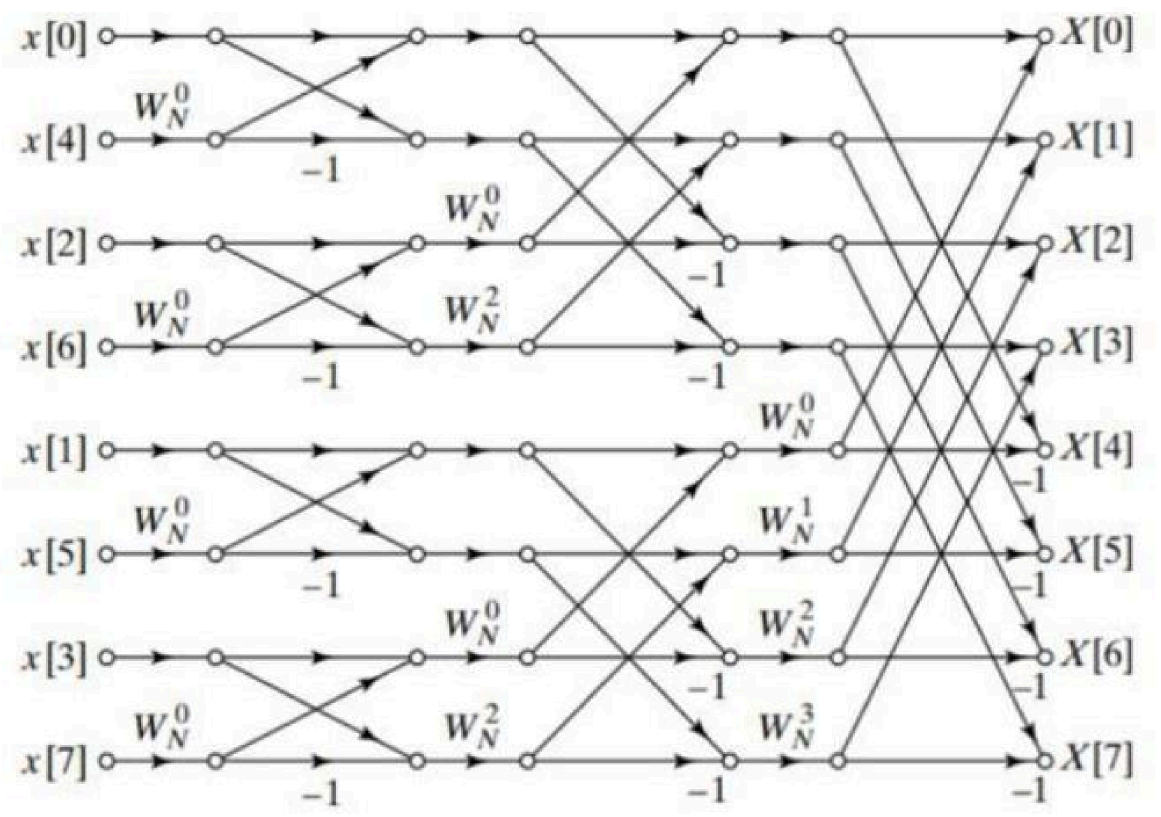
3. Twiddle Factors:

During the division of the series, a key element that emerges is the concept of "twiddle factors." These factors are coefficients that are multiplied at every stage when merging two smaller length series to create a larger one.

Formula:

$$W_N = e^{-j2\pi/N}$$

Note: The initial order of the inputs is in reverse binary digit order.



A video file is attached to demonstrate the code's operation on the computer with explanation.

Challenges

1. Low-Level Abstraction: Implementing the FFT algorithm in a programming language is straightforward, while doing so in Verilog presents a considerable challenge. The declaration of variables and functions, as well as the flow of code, is not as intuitive in Verilog compared to high-level languages like Python or Java.

2. Array Handling: Care must be taken when declaring arrays in Verilog, as FPGA architectures interpret them differently. This can lead to difficulties in accessing individual array elements independently, making proper array management essential.

3. Simultaneous Execution: Unlike high-level programming languages where functions are called sequentially, Verilog modules execute concurrently. This parallel execution can complicate the control flow and synchronisation of data.

4. Memory Allocation: In Verilog, separate memory allocation is necessary for input, output, real, and imaginary data parts. Managing memory efficiently becomes a critical aspect of the implementation.

5. Hardcoding Input/Output Ordering: Unlike in high-level languages where dynamic data structures can be used, Verilog often requires hardcoding the order of inputs and outputs. This reduces design flexibility and may introduce potential errors.

6. Recursive Loop Constraints: Verilog has limitations in implementing recursive loop patterns commonly found in FFT algorithms. Designers must resort to iterative or hard-coded solutions, adding complexity to the implementation.

7. Resource Constraints: FPGA devices have limited resources, necessitating careful resource management to balance performance with these constraints.

8. Clock Domain Management: Handling data transfers between different clock domains and ensuring proper synchronization is crucial but complex in Verilog.

9. Testing and Debugging: Verifying the correctness of an FFT implementation in Verilog requires specialized tools and debugging techniques due to the low-level hardware nature of Verilog.

10. Numerical Precision: Managing numerical precision and addressing rounding errors, especially when dealing with fixed-point arithmetic, is vital to ensure the accuracy of the FFT implementation.

11. Real-Time Requirements: Applications requiring real-time FFT processing demand strict adherence to timing constraints, which can be challenging to meet.

12. Optimization Trade-offs: Achieving a balance between latency and throughput while optimizing the FFT algorithm for FPGA performance is a complex task.

Improvements

1. Using VHDL for Improved Performance: VHDL can be a viable alternative to Verilog and might offer improved performance for certain applications. The choice between VHDL and Verilog often depends on the specific project requirements and designer preferences.

2. Parallelization of Modules: Instead of designing modules to run sequentially, one can explore parallelization techniques by utilising alternative logic structures. This approach can significantly enhance processing speed and efficiency.

3. Memory Allocation for Decimal Values: To avoid the need for multiplying by a power of ten to manage decimal points in data, a more efficient approach could involve separate memory

allocation for values after the decimal point. This method simplifies data handling and maintains numerical precision.

4. Reducing Hardcoding: By leveraging a comprehensive understanding of Verilog's properties and functionalities, it is possible to minimise hardcoding. Designers can apply logic patterns and parameterization to make the code more flexible and adaptable, thus enhancing the maintainability and scalability of the implementation.

These strategies can lead to more efficient and adaptable designs while also improving the overall performance of FPGA-based systems.

Applications

Certainly, here are various applications of the Fast Fourier Transform (FFT):

1. Signal Processing:
   - Audio Processing: FFT is used for audio compression, equalization, noise reduction, and spectral analysis.
   - Image Processing: In image processing, FFT helps with image filtering, enhancement, and feature extraction.

2. Communication Systems:
   - Wireless Communication: FFT is employed for modulation and demodulation in wireless communication systems, including Wi-Fi, 4G, and 5G.
   - OFDM (Orthogonal Frequency-Division Multiplexing): FFT is crucial for OFDM, used in communication standards like DVB, LTE, and WiMAX.

3. Biomedical Signal Processing:
   - Electroencephalography (EEG): FFT is applied to EEG signals for brain wave analysis and monitoring.
   - Electrocardiography (ECG): FFT is used for heart rate variability analysis and identifying specific frequency components in ECG data.

4. Seismology:
   - Earthquake Analysis: Seismologists use FFT to analyze seismic signals and identify the frequencies associated with earthquakes and other geological events.

5. Radar and Sonar:
   - Radar Imaging: FFT is used in radar signal processing for target detection, range estimation, and Doppler shift analysis.
   - Sonar Signal Processing: FFT is employed in underwater acoustics to analyze sonar signals for detecting objects in the water.

The Fast Fourier Transform (FFT) is widely used across various scientific, engineering, and industrial applications due to its efficiency in analyzing signals in the frequency domain.

References

[1] **“Discrete-Time Signal Processing” Alan V. Oppenheim Ronald W. Schafer Third Edition**
[2] K. Sobaihi, A. Hammoudeh, D. Scammell, “**FPGA Implementation of OFDM Transceiver for a 60GHz Wireless Mobile Radio System**” International Conference on Reconfigurable Computing and FPGAs (ReConFig), pp. 185-189, 2010.
[3] W.Wolf, **FPGA-Based System Design**. Englewood Cliffs, Prentice- Hall, 2004.
[4] V. Gautam, K.C. Ray, P. Haddow, “**Hardware efficient design of Variable Length FFT Processor**”, 14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp. 309 –312, 2011.
[5] Long Pang, Bocheng Zhu, He Chen, “ **Design and realization of small point FFT processor based on twiddle factor classification**”, International Conference on Electronics, Communications and Control (ICECC), pp.1396 – 1399, 2011.
[6] I. Good, “**The Relationship between Two Fast Fourier Transforms**,” IEEE Transactions on Computers, vol. 20, pp. 310–317 ,1971
[7] S. Winograd, “**On computing the discrete Fourier transform**,” Math. Comp., 32, pp. 175-199, 1978.
[8] D. Cohen, **“Simplified control of FFT hardware**,” IEEE Trans. Acoustics, Speech, Sig. Proc., pp. 577-579, Dec. 1976.
FPGAFFTExample.ps (mit.edu)
(PDF) Implementing FFT algorithms on FPGA (researchgate.net)