این گزارش فنی به زبان **فارسی** به صورت یک فایل PDF با نام **Convolution_Report_Persian** به همراه فایل‌های پروژه موجود است.

Project Name: FPGA-Based Convolution Implementation (Project page on GitHub)

Designer Name: Milad Zadrafee

Project Creation Date and Version: November 2025 – Version 1.0

Design Software and Language: Xilinx ISE 14.7 – VHDL

Contact Details:

https://www.linkedin.com/in/zadrafee
https://www.github.com/zadrafee

This document is part of a series of FPGA projects published on the LogLab website. To view other related projects and technical articles, visit my official website:

http://www.LogLab.ir

Description of the main project files and folders

Convolution_MATLAB	Compare.m	Creating input and comparing output of ISE and MATLAB
	Convolution.slx	Convolution simulation in Simulink
Convolution_VHDL	Convolution_VHDL.xise	Convolution Project at ISE
	Convolution_Top.vhd	Top Convolution Module in ISE
	Convolution_VHDL_tb.vhd	Convolution Benchmark Test in ISE
	Multiply_Conjugate.vhd	Complex Multiplier and Conjugate Module
Convolution_Report_Persian.pdf		Convolution Technical Report in Persian
Convolution_Report_English.pdf		Convolution technical report in English
Convolution_RTL.pdf		RTL schematic of the implemented convolution scheme
xfft_ds260.pdf		IP FFT Datasheet Version 7.1 Native

Abstract

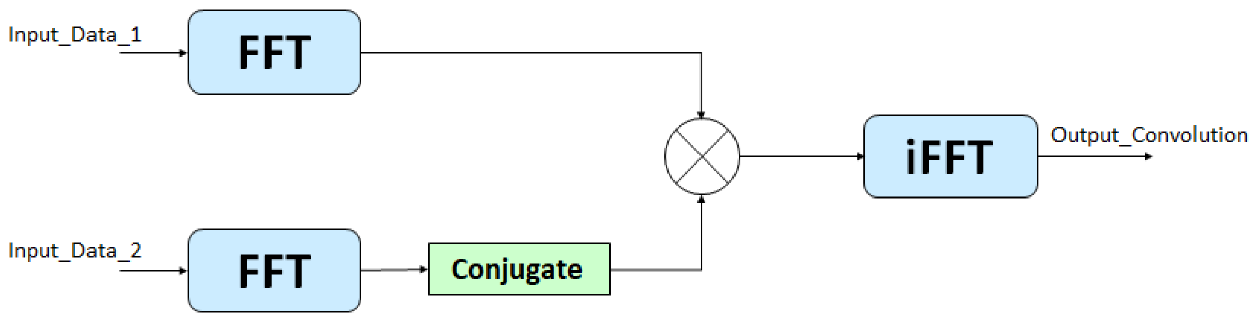
In this project, discrete convolution operation is implemented using VHDL language and in Xilinx ISE environment. This design is done with the aim of fast data processing on FPGA and can be used as part of image and signal processing systems.

Introduction

Convolution is a mathematical operator used to combine two signals or functions to show how the shape of one affects the other.

In simpler terms, convolution means moving (shifting) one signal over another and calculating the amount of overlap at each position. The result of this process is a new signal that represents the degree of similarity or interaction between the two signals over time. This is a simple definition of convolution, far from complex mathematical calculations and proofs. In the field of signal and image processing, convolution helps us perform operations such as filtering, feature extraction, noise removal, edge detection, or data smoothing.

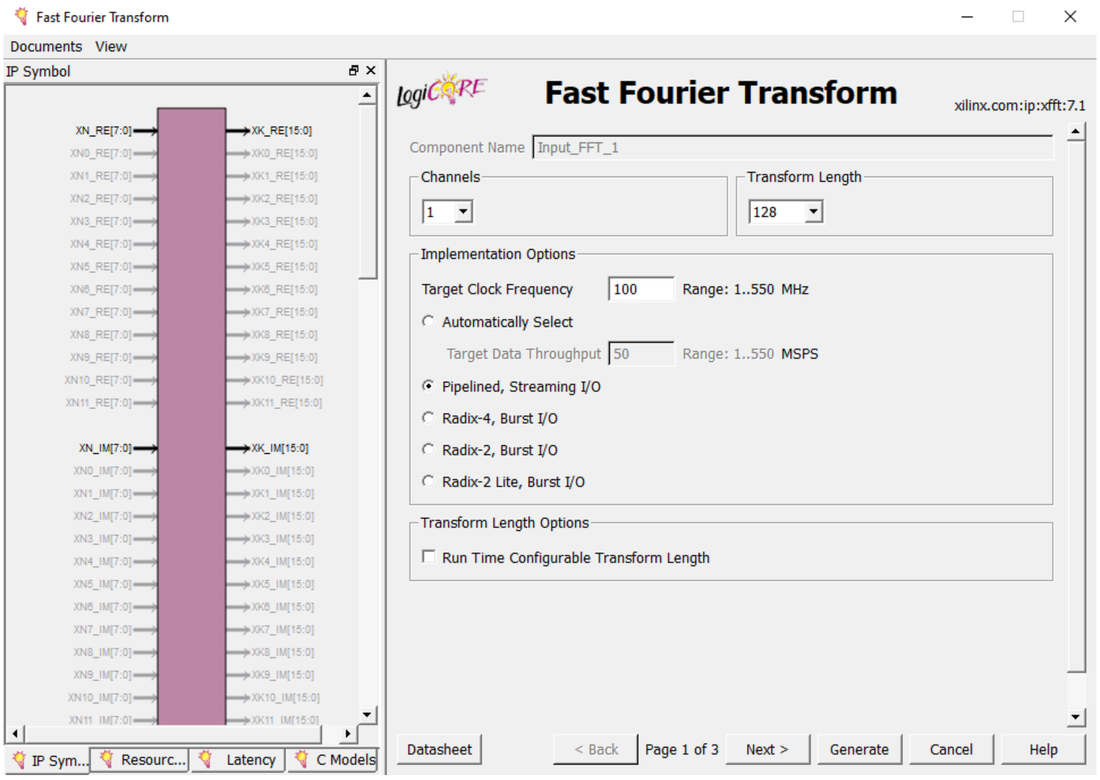
General description of the FFT system and IP



In the image above, you can see the general diagram of the convolution system implemented in this project. The sample applied to the Input_Data_1 input was generated by the MATLAB software using the rand function, which is a 128-digit Vector, and the sample applied to the Input_Data_2 input is the same 128 previous samples that have been shifted by 10 digits. The code for generating these inputs is available in the Compare file in the Convolution_MATLAB folder.

The method is exactly the same principles as given in the Signal and System books. We give two inputs, each containing 128 samples, to two FFTs with identical specifications. Of course, it is important to state here that using two FFTs is easier to understand, while a more principled method is to use a two-channel FFT, which uses fewer hardware resources, but the output generation speed is also relatively slower. Our applied inputs are three-digit integers smaller than 200, so the FFT inputs must be 8 binary bits, so in the FFT GUI settings page, we must set the Transform Length to 128. In this project, it is assumed that our circuit clock is 100Mhz. So we set the Target Clock Frequency to 100Mhz.

In this project, Native FFT version 7.1 has been used (the datasheet of this FFT is attached to the project files), so obviously this code cannot be used directly for FFT with the AXI4-Stream communication protocol and requires some modifications and changes. Of course, the principles of work are the same, you just need to be familiar with how the AXI4-Stream communication protocol works to apply the necessary changes to the code for correct operation.



LogiCORE **Fast Fourier Transform** xilinx.com:ip:xfft:7.1

Component Name: Input_FFT_1

Channels: 1

Transform Length: 128

Implementation Options

Target Clock Frequency: 100 Range: 1..550 MHz

Automatically Select

Target Data Throughput: 50 Range: 1..550 MSPS

[✓ Pipelined, Streaming I/O]

Radix-4, Burst I/O

Radix-2, Burst I/O

Radix-2 Lite, Burst I/O

Transform Length Options

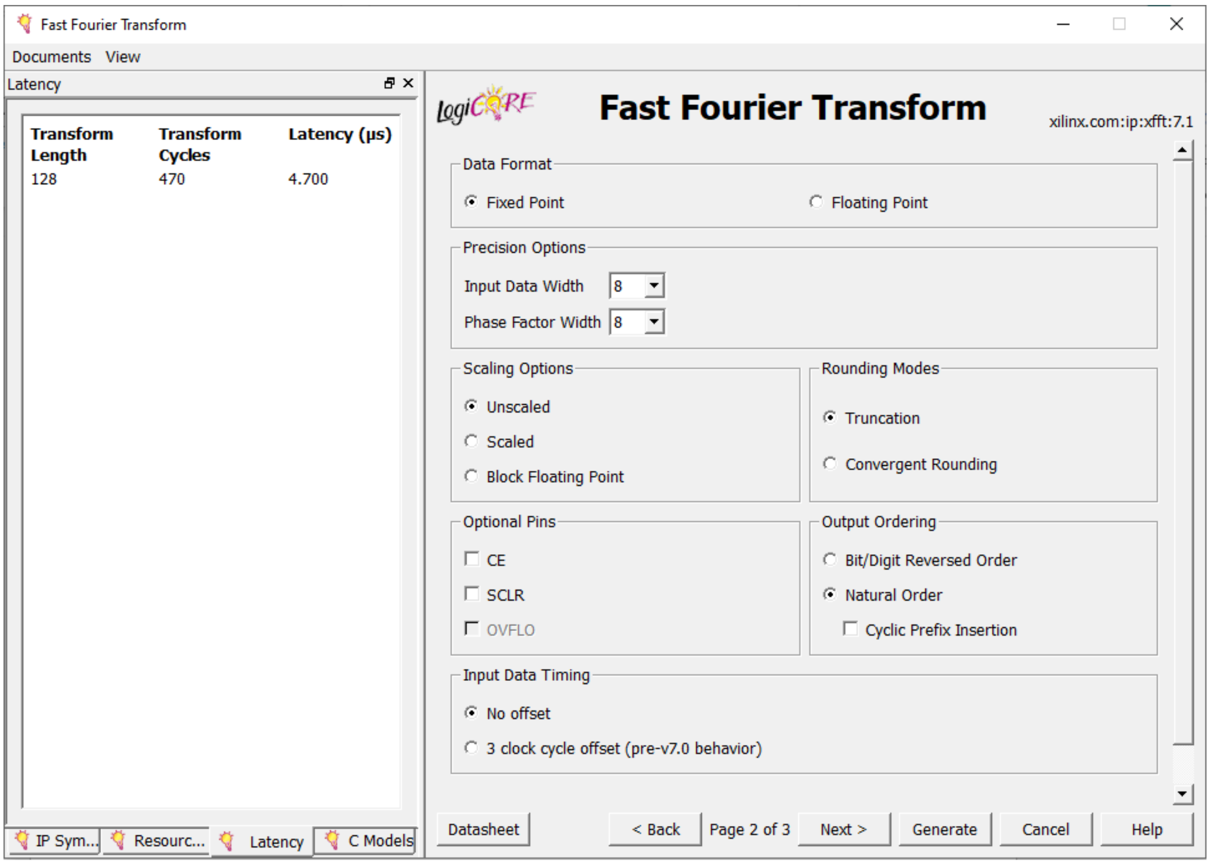
☐ Run Time Configurable Transform Length

In the Implementation Options section, we specify the type of architecture that FFT is going to implement. In the table below, you can see the differences between each implementation type.

Architecture type	Throughput	Latency	Resource consumption	Application
Pipelined, Streaming	Very high	Low	High	Real-time processing
Radix-4, Burst	High	Medium	Medium	Semi-living systems
Radix-2, Burst	Medium	High	Low	Small systems
Radix-2 Lite, Burst	Low	High	Very Low	Low-cost applications

In this project, we set the implementation type to Pipelined, Streaming to have high Throughput and low Latency. Our FFT Latency in this project is 4.7µs.

If your project requires you to change the transform length (number of FFT points) live, you should use the Transform Length Options section and check the Run Time Configurable Transform Length option.


Fast Fourier Transform

Latency

Transform Length	Transform Cycles	Latency (µs)
128	470	4.700

Fast Fourier Transform xilinx.com:ip:xfft:7.1

Data Format: [✓ Fixed Point] Floating Point

Precision Options: Input Data Width 8; Phase Factor Width 8

Scaling Options: [✓ Unscaled] Scaled Block Floating Point

Rounding Modes: [✓ Truncation] Convergent Rounding

Optional Pins: ☐ CE ☐ SCLR ☐ OVFLO

Output Ordering: Bit/Digit Reversed Order [✓ Natural Order] ☐ Cyclic Prefix Insertion

Input Data Timing: [✓ No offset] 3 clock cycle offset (pre-v7.0 behavior)

On the next page of the IP FFT settings, you can specify the type of quantization. If you select Fixed Point, the minimum number of bits with the lowest error rate is considered for all main and intermediate inputs. By selecting the Floating Point option, all calculations are performed with the highest accuracy and, of course, it uses a lot of resources, which is not desirable for us. As we said earlier, we choose 8-bit inputs. The Phase Factor coefficient is the same as the Twiddle Factor coefficient in Butterflies. The exact selection of this coefficient requires calculations, but it is usually chosen as a number equal to the number of inputs.

Changing the location of the binary point is called Scaling. We set it to Unscaled according to our requirements in this project.

The Rounding Modes section specifies the type of bit width reduction of signals. If it is set to Truncation, it cuts off the low-value bits without rounding, but the other mode, Convergent Rounding, in addition to cutting and reducing the bits to create better accuracy, it also rounds them up appropriately, which obviously uses more hardware resources. The choice of this option depends on the type of your project and the accuracy required, as well as the available hardware resources. In this project, we set it to Truncation.

The CE option is short for Clock Enable and is used to control the clock signal.

SCLR is short for Synchronous Clear and is a signal to clear or reset the entire FFT core simultaneously during operation. When the SCLR pin is activated (1), all internal FFT registers are cleared. Counters, buffers, and internal status are returned to their original state. This is done on the active edge of the clock (clk), which is why it is called Synchronous. And when the signal becomes ‘0’ again, the FFT is ready to receive data from the beginning.

Difference between SCLR and ARESET or ACLK

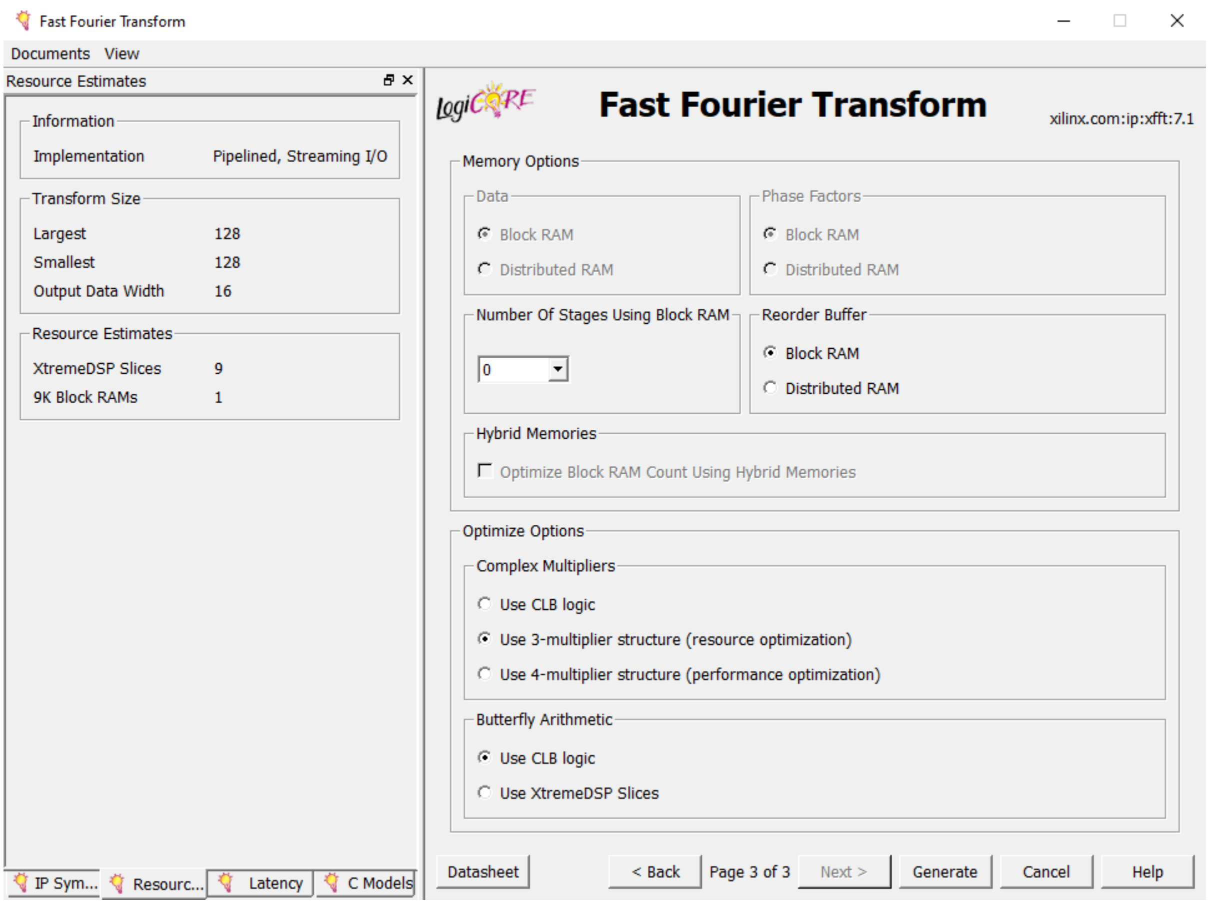
As mentioned, SCLR has a Synchronous Reset reset type and only operates on the clock edge, which usually has an active level of ‘1’. However, in newer IPs, ARESETN or ACLK, their reset type is Asynchronous Reset and operates immediately without dependence on the clock, and usually has an active level of ‘0’.

In the Output Ordering section, the output display type is determined, whether it is Bit/Digit Reversed Order or Natural Order. In the Natural Order method, the outputs are output from the FFT in the same natural order as their indexes. However, in the Bit/Digit Reversed Order method, the FFT core uses a special structure called Butterfly to be fast. This structure causes the data order at the input or output to no longer be in the usual order (0,1,2,3,…). Therefore, at the end of the FFT calculation, the data becomes “scrambled” meaning their indexes are shifted in a specific order called Bit-Reversed Order or Digit-Reversed Order.

New number	Bit-Reversed	Binary	Natural Index
0	000	000	0
4	100	001	1
2	010	010	2
6	110	011	3
1	001	100	4
5	101	101	5
3	011	110	6

In this design, since we need the outputs to be sorted by their Indexes, we use the Natural Order method.

At the end of this section, there is a section called Input Data Timing, which is related to old FFT IPs with versions before 7. If you are using version 7 or later, set this option to No Offset.


Fast Fourier Transform

Documents View

Resource Estimates

Information

Implementation	Pipelined, Streaming I/O

Transform Size

Largest	128
Smallest	128
Output Data Width	16

Resource Estimates

XtremeDSP Slices	9
9K Block RAMs	1

LogiCORE **Fast Fourier Transform** xilinx.com:ip:xfft:7.1

Memory Options

Data: [✓ Block RAM] Distributed RAM

Phase Factors: [✓ Block RAM] Distributed RAM

Number Of Stages Using Block RAM: 0

Reorder Buffer: [✓ Block RAM] Distributed RAM

Hybrid Memories: ☐ Optimize Block RAM Count Using Hybrid Memories

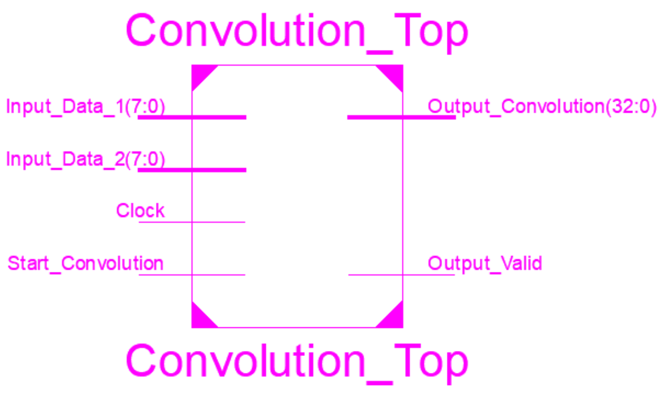
Optimize Options

Complex Multipliers: Use CLB logic [✓ Use 3-multiplier structure (resource optimization)] Use 4-multiplier structure (performance optimization)

Butterfly Arithmetic: [✓ Use CLB logic] Use XtremeDSP Slices

On the last page, there are settings related to memory, where you can specify the type of Data and Phase Factors memory, which is Block RAM or Distributed RAM. Usually, because Distributed RAM memories occupy a lot of space in our FPGA and slow down the FPGA chip, they are set to Block RAM by default. The last section of the settings, Optimize Options, is for optimizing complex multipliers, and according to the description written by Xilinx under the Complex Multiplier section, the second option, Use 3-multiplier structure, is the optimal option in terms of resources consumed, and the third option, Use 4-multiplier structure, produces better results in terms of performance (meaning Throughput and Latency), but consumes more hardware resources.

Explanation of modules

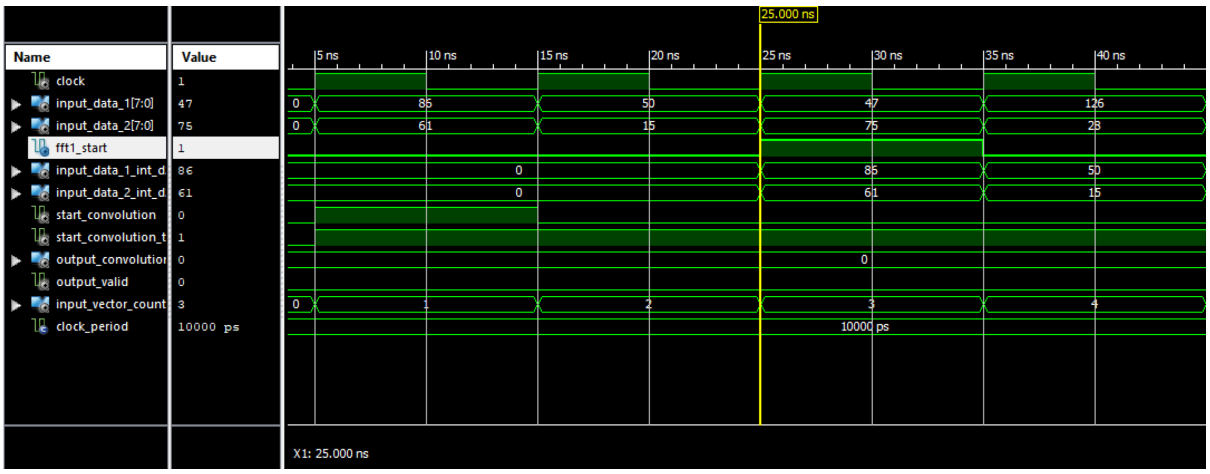


The main module of our project is called Convolution_Top, this module has inputs Clock (single bit), Input_Data_1 and Input_Data_2 (8 bits), Start_Convolution (single bit) and outputs Output_Convolution (33 bits) and Output_Valid (single bit). The beating heart of our project is the FFT IP, so we need to familiarize ourselves with its ports. The table below lists the FFT pins along with their descriptions.

Port name	Abbreviated	Description
clk	Clock	Clock signal
start	Start	Start executing the FFT operation
xn_re	$x_{(n)}$ Real	The real part of the input data
xn_im	$x_{(n)}$ Imaginary	Imaginary part of input data
fwd_inv	Forward/Inverse	Specify the type of conversion (FFT=1 and IFFT=0)
fwd_inv_we	Forward/Inverse Write Enable	Enable recording of fwd_inv value
rfd	Ready For Data	When it becomes 1, it means the module is ready to receive new data
xn_index	$x_{(n)}$ Index	The number (index) of the data to be sent
busy	Busy	Indicates that the FFT is processing
edone	Early Done	Early termination of synchronization or control
done	Done	The FFT operation is completely finished
dv	Data Valid	The output data is valid and ready to read
xk_index	$x_{(k)}$ Index	FFT output sample number (index)
xk_re	$x_{(k)}$ Real	Real part of the FFT output
xk_im	$x_{(k)}$ Imaginary	Imaginary part of the FFT output

The inputs are connected to the xn_re and xn_im ports through the defined intermediate signals. After the FFTs are finished, the FFT outputs should be complex multiplied together according to the schematic given at the beginning of this article. Before the outputs are multiplied together, the output of one of the FFTs should be conjugated, that is, its imaginary part should be multiplied by a negative number, and then the multiplication operation is performed. For this purpose, a module called Multiply_Conjugate has been designed to perform both the conjugation and complex multiplication operations at the same time. Then, after calculating the product, which is a 25-bit signal (25 bits for the real part and 25 bits for the imaginary part), we should give this value to the final FFT and set it in inverse mode. The rest of the iFFT explanation is the same as the previous steps except that the iFFT input bit width is 25 bits and we set the Phase Factor or Twiddle Factor to 16 bits and also give the fwd_inv signal a value of ‘0’ to work in reverse.

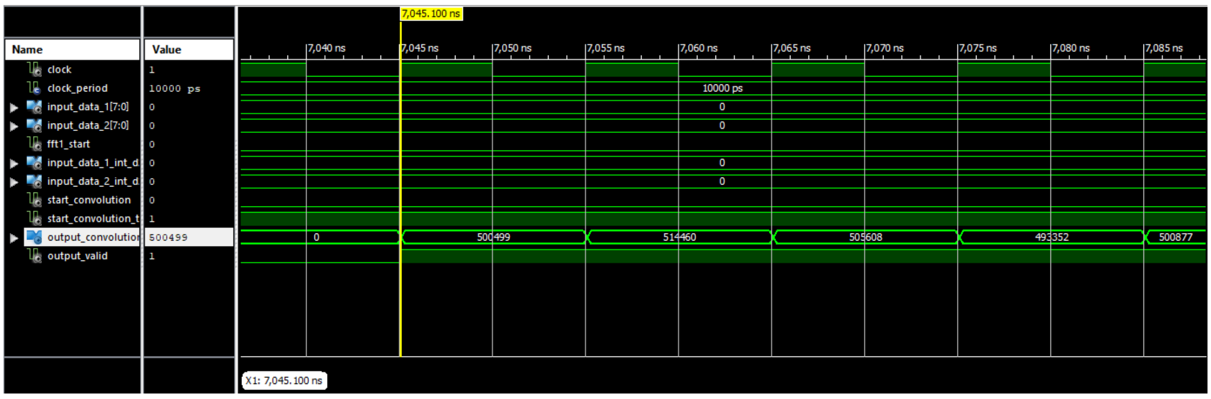
Note: In the VHDL code, we have delayed the inputs by 2 clocks or in other words, we have created a two-clock Delay Line. What is the reason for this? Look at the figure below:



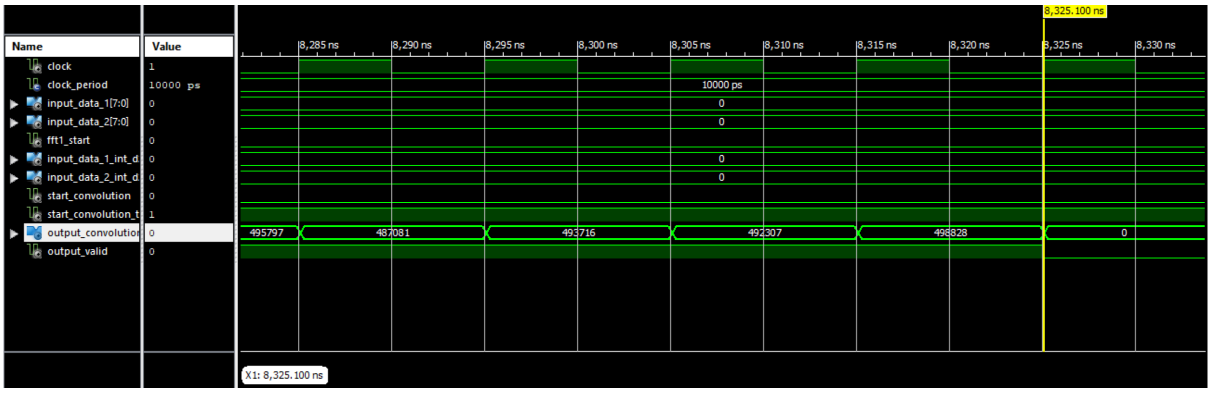
Due to the code structure and coding type of the Start signal, our FFT, which we denote by FFT1_Start, has a two-clock delay. In order to compensate for these two-clock delays, we inevitably have to apply the input signal to the FFT with two-clock delays so that all samples are completely transferred from the beginning.

Considering that the clock period of our circuit is 10ns and we must apply 128 samples to the FFT and each sample is applied in one clock, it takes 1280ns to apply all 128 samples to the FFT without considering the Propagation Delays. The iFFT output generation starts at 7045ns and at 8325ns, the conversion of our 128 samples is completed, in other words, our iFFT output is complete. Of course, it is clear that the hardware test of this project on the FPGA results in different times (longer) than the values mentioned in this article.

The waveform of the iFFT response start in the ISim software


Name	Value
clock	1
clock_period	10000 ps
input_data_1[7:0]	0
input_data_2[7:0]	0
fft1_start	0
input_data_1_int_d	0
input_data_2_int_d	0
start_convolution	0
start_convolution_t	1
output_convolutio	500499
output_valid	1

The waveform of the final part of the FFT response in ISm software


Name	Value
clock	1
clock_period	10000 ps
input_data_1[7:0]	0
input_data_2[7:0]	0
fft1_start	0
input_data_1_int_d	0
input_data_2_int_d	0
start_convolution	0
start_convolution_t	1
output_convolutio	0
output_valid	0

Additional notes

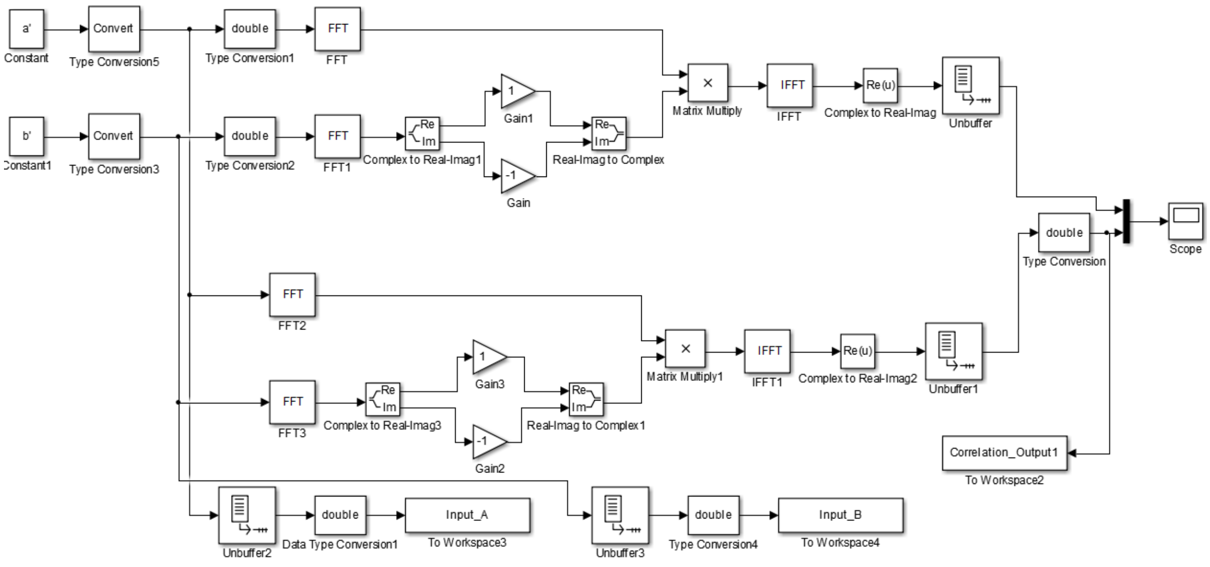
Be careful that for the Test bench to function properly, you must change the path (address) of the text files according to the path in your operating system, otherwise you will encounter an error message.

In the files of this project, there is a folder called Convolution_MATLAB, which contains input generation files, comparison of results from ISE and MATLAB, and simulation file in MATLAB software. The simulation file (Simulink) called Convolution simulates exactly the same project in Simulink. The purpose of simulating this design in MATLAB software is to quantize inputs and operators and obtain a Fixed point model.

According to the IP FFT datasheet, this project can be implemented on Virtex-7, Kintex-7, Virtex-6, Virtex-5, Virtex-4, Spartan-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP/XA series chips. For newer series of Xilinx chips, you should use FFT with newer versions, whose communication protocol is AXI4-Stream.

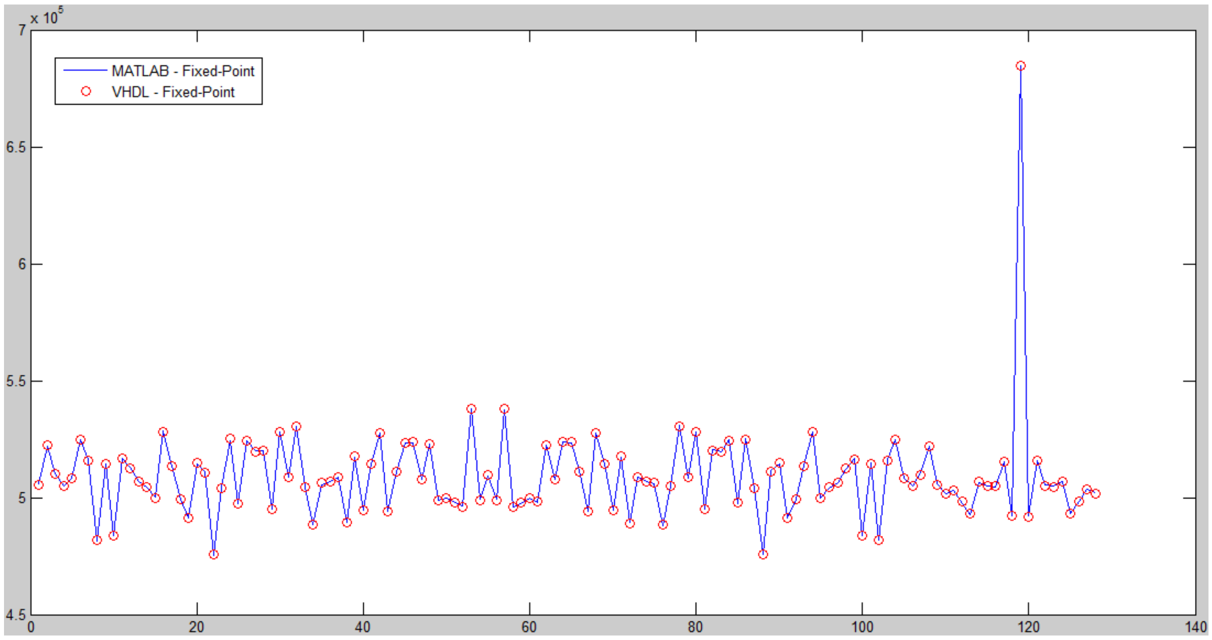
Results and Performance Analysis

The figure below shows a view of the Simulink environment in MATLAB software.



The upper part of the image above implements the floating point model of the convolution scheme and the lower part models the fixed point part of the convolution.

We have compared the output of the VHDL implementation of the convolution in ISE software with the output of the MATLAB software, the result of this comparison is shown in the following figure:



The blue lines are the simulation results in MATLAB software and the red hollow circles are the simulation results in ISE software implemented in VHDL, as you can see. The results match with high accuracy.