FFT Compiler IP Core - Lattice Radiant Software

User Guide

FPGA-IPUG-02153-1.3

November 2022

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
EBR	Embedded Block RAM
DIF	Decimation-in-Frequency
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
GUI	Graphical User Interface
RAM	Random Access Memory

1. Introduction

The Lattice Semiconductor Fast Fourier Transform (FFT) Compiler IP Core offers forward and inverse FFTs for point sizes from 64 to 16384. The FFT Compiler IP Core can be configured to perform forward FFT, inverse FFT (IFFT), or port selectable forward/inverse FFT. It offers two modes of implementation: High Performance (Streaming I/O) and Low Resource (Burst I/O).

In High Performance implementation, the FFT Compiler IP Core can perform real-time computations with continuous data streaming in and out at clock rate. There can also be arbitrary gaps between data blocks allowing discontinuous data blocks.

In Low Resource implementation, the requirement is to use less slice (logic unit of Lattice FPGA devices), Embedded Block RAM (EBR), and Digital Signal Processor (DSP) resources. The device could also be too small to accommodate the High-Performance version.

To account for the data growth in fine register length implementations, the FFT Compiler IP Core allows several different modes (fixed and dynamic) for scaling data after each radix-2 stage of the FFT computation. The Low Resource version also supports block floating point arithmetic that provides increased dynamic range for intermediate computations. It allows the number of FFT points to be varied dynamically through a port.

1.1. Quick Facts

Table 1.1 presents a summary of the FFT Compiler IP Core.

Table 1.1. Quick Facts

IP Requirements	Supported FPGA Families	CrossLink™-NX, Certus™-NX, CerturPro™-NX, MachXO5-NX, Lattice Avant
Resource Utilization	Targeted Devices	LIFCL-40, LIFCL-17, LFD2NX-40, LFD2NX-17, LFCPNX-100, LFMXO5-25, LAV-AT-500E
	Resources	See Table A.1 and Table A.2
Design Tool Support	Lattice Implementation	IP Core v1.x.x – Lattice Radiant™ Software 2.2 or later
	Synthesis	Lattice Synthesis Engine
		Synopsys® Synplify Pro® for Lattice
	Simulation	For a list of supported simulators, see the Lattice Radiant software user guide.

1.2. Features

The key features of FFT Compiler IP Core include:

- Wide range of point sizes: 64, 128, 256, 512, 1024, 2048, 4096, 8192, and 16384
- Choice of high-performance (streaming I/O) or Low Resource (burst I/O) versions
- Run-time variable FFT point size
- Forward, inverse, or port-configurable forward/inverse transform modes
- Choice of no scaling, fixed scaling (RS111/RS211), or dynamically variable stage-wise scaling
- Data precision of 8 to 24 bits
- Twiddle factor precision of 8 to 24 bits
- Natural order for input and choice of bit-reversed or natural order for output
- Support for arbitrary gaps between input data blocks in high-performance realization
- Block floating point scaling support in Low Resource configurations

1.3. Conventions

1.3.1. Nomenclature

The nomenclature used in this document is based on Verilog HDL.

1.3.2. Signal Names

Signal Names that end with:

- _*i* are input signals
- _*o* are output signals

1.3.3. Attribute

The names of attributes in this document are formatted in title case and italicized (*Attribute Name*).

2. Functional Description

2.1. Overview

Figure 2.1 shows the interface diagram for the FFT compiler. The diagram shows all of the available ports for the IP. It should be noted that not all the I/O ports are available for a chosen configuration.

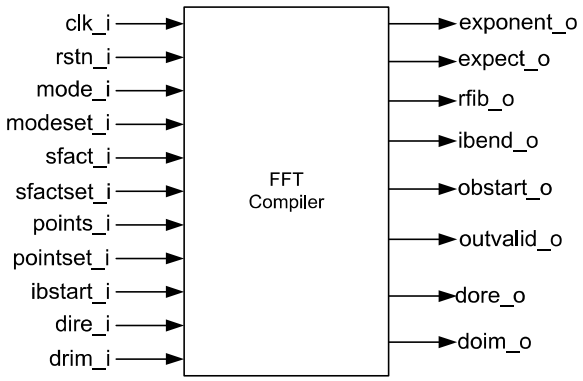

Figure 2.1. FFT Compiler Interface Diagram

FFT is a fast algorithm to implement the following N point Discrete Fourier Transform (DFT) function.

$$X(\mathrm{k}) = \sum_{n=0}^{N-1} x(\mathrm{n}) W_N^{nk} \tag{1}$$

where W_N is given by:

$$W_N = e^{\frac{2n}{N}} \tag{2}$$

The inverse DFT is given by:

$$X(n) = \frac{1}{n} \sum_{k=0}^{N-1} X\,(k) W_N^{-nk} \tag{3}$$

However, the output of the FFT Compiler IP Core differs from the true output by a scale factor determined by the scaling scheme. If *right-shift by 1 at all stages* scaling mode (RS111) is used, there is a division by 2 at every stage resulting in an output that is 1/Nth of the true output of Equation (1). The output for inverse FFT matches with Equation (3) for this scaling mode. Using other scaling modes results in outputs scaled by other appropriate scale factors.

In High Performance mode, the FFT Compiler IP Core can continuously read in and give out one data sample per clock, block after block. The FFT throughput is equal to the clock rate when the data blocks are applied continuously. Low Resource mode uses less logic and memory resources, but requires 4 to 8 block periods to compute the FFT for one block of input data. Both versions of FFT do not allow breaks in the data stream within a block but do allow arbitrary additional gaps between data blocks.

2.1.1. High Performance Architecture

The implementation diagram for the High Performance FFT is shown in Figure 2.2.

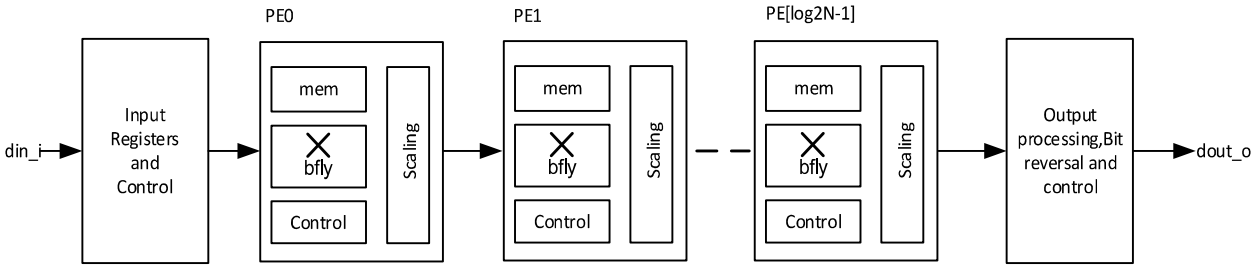

Figure 2.2. Implementation Diagram for High-Performance FFT

The High Performance FFT implementation consists of several processing elements (PEs) connected in cascade. The number of PEs is equal to log2 N, where N is the number of FFT points. Each PE has a radix-2 decimation-in-frequency (DIF) butterfly (bfly), a memory (mem), an address generation and control logic (control), and a scaling unit (scaling). Some of the butterflies include a twiddle multiplier and a twiddle factor memory. The scaling unit performs a division by 2, a division by 4, or no division, depending on the scaling mode and scale factor inputs at the port. There is an input-processing block at the beginning of the PE chain and an output-processing block at the end of the PE chain. The input processing block has registers and control logic for handshake signals and dynamic mode control. The output-processing block contains handshake, mode control and bit-reversal logic, if configured for natural ordered output.

The High Performance FFT implementation enables streaming I/O operation, where the data is processed at clock speed without any gaps between blocks. This implementation can also be employed for burst I/O situations by using the handshake signals.

2.1.2. Low Resource Architecture

The Low Resource implementation employs only one physical radix-2 butterfly and reuses the same butterfly over multiple time periods to perform all stages of the FFT computation. Hence the resource requirement (EBR and slices) is lower compared to the high-performance version. Depending on the number of points, an N-point FFT computation may require 4N to 8N cycles. The implementation diagram for the low-resource FFT is shown in Figure 2.3.

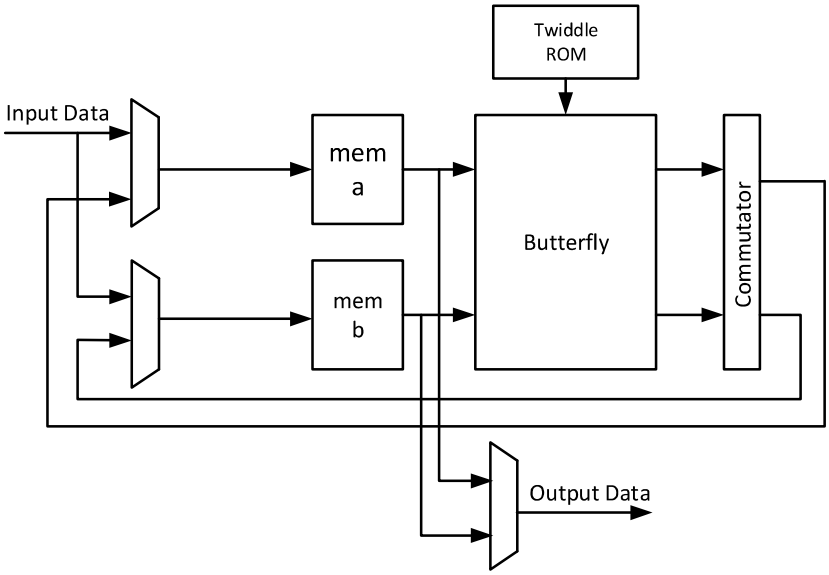

Figure 2.3. Low-Resource FFT Data Flow Diagram

As Figure 2.3 shows, the FFT module is built with a butterfly reading from and writing to two memories at the same time. There is a commutator after the butterfly to handle the writing sequence of the intermediate outputs. The twiddle memory contains the pre-computed twiddle factors for the FFT. When an input block is applied, the first half of the block is written into memory **a** and the second half into memory **b** in a bit-reversed order. The butterfly reads from the two memories, performs stage 0 computation and writes out the intermediate results to the same sites in each memory. Again, for stage 1 computation, the butterfly reads from the two memories, performs computation and writes back into the two memories through the commutator. A similar process of reading, computing and writing continues for each of the remaining stages. For every block of input data read, four to eight blocks of computation time is required for this scheme. Due to the twin memory architecture, when data is unloaded form FFT in bit-reversed mode, the data in memory a (points 0 to N/2-1) is unloaded first, followed by the data in memory b (N/2 to N-1), both in a bit-reversed order.

2.2. Signal Descriptions

The top-level interface diagram for the FFT Compiler IP Core is shown in Figure 2.1 and the details of the I/O ports are summarized in Table 2.1.

Table 2.1. Top level I/O interface

Port	Direction	Bits	Description
Clock and Reset			
clk_i	Input	1	System clock
rstn_i	Input	1	System wide asynchronous active-low reset signal
Data Input and Output			
dire_i	Input	*Input Data Width*	Real part of the input data
diim_i	Input	*Input Data Width*	Imaginary part of the input data
dore_o	Output	*Output Data Width*	Real part of the output data
doim_o	Output	*Output Data Width*	Imaginary part of the output data
exponent_o	Output	Log2(N) + 1	This value denotes the effective scaling that was done during block floating scaling. Available only when *Scaling Mode* == Block Floating Point.
Configuration Signals			
mode_i	Input	1	When asserted, core will perform inverse FFT else core will perform forward FFT. The value at mode_i is loaded into the system whenever modeset_i input goes high. The changes are effective from the start of the next input data block, i.e., for an ibstart_i going high during or after modeset_i.
modeset_i	Input	1	Sets the FFT mode signal. When this signal goes high, the value at mode_i port is read and the FFT mode (forward or inverse FFT) is set.
sfact_i	Input	A	Stage-wise scaling factors. This signal is a concatenation of individual 2-bit stage scaling factors. The most significant 2 bits correspond to stage 1 scale factor, the next significant 2 bits to stage 2 scale factor and so on. When the number of point is not a power of 4, the last stage has only 1-bit scaling factor. Each scaling factor denotes the number of right shifts performed to that stage's output data. The scale factors are loaded into the system when sfactset_i input goes high. The changes are effective from the start of the next input data block, for example, for an ibstart_i going high during or after sfactset_i.

Port	Direction	Bits	Description
sfactset_i	Input	1	Set scale factor signal. When this signal goes high, stage-wise scaling factors are set with the values at sfact_i input port.
points_i	Input	3 if *Maximum Points* == 128 ,otherwise, 4	Number of FFT points. This input is used to specify the number of points in the dynamic points mode. The value at this port must be equal to the log2 of the number of points represented in unsigned binary form. The valid range of values is from 6 to 14. A value less than 6 is read as 6 and a value greater than 14 is read as 14.
pointset_i	Input	1	Set the number of points signal. When this input signal goes high, the value at points_i port is read and the number of FFT points is set accordingly. The new number of points is effective from the next block of data, for example, for a valid ibstart_i applied after pointset_i going high. For Low Resource mode, pointset_i can be applied during or before ibstart_i. For High Performance mode, pointset_i must be applied five cycles before ibstart_i.
Status and Handshake Signals			
ibstart_i	Input	1	Input block start signal. Asserted high by the user to identify the start of an input data block. Once this signal goes high for a cycle, the core enters an input read cycle, during which input data is read in N consecutive cycles (N is the number of FFT points). Any ibstart_i signal during an input read cycle is ignored.
except_o	Output	1	Exception output signal. Denotes that an exception (overflow) has occurred in the computation. This could be due to the use of a wrong set of scaling factors. The exception always corresponds to a problem with the data that is currently output and not with a problem with the data that is being processed.
rfib_o	Output	1	Ready for input block output signal. This signal indicates that the core is ready to receive the next block of input data. The driving system can assert ibstart_i one cycle after rfib_o goes high. After ibstart_i goes high, the core pulls rfib_o low in the next clock cycle.
ibend_o	Output	1	Input block end output signal. This signal goes high for one cycle to coincide with the last sample of the current input data block that is being read through input ports.
obstart_o	Output	1	Output block start output signal. This signal is asserted high by the core for one clock cycle, to signify the start of an output block of data.
outvalid_o	Output	1	Output data valid output signal. This signal indicates that the core is now giving out valid output data through dore_o and doim_o ports.

Notes:

- If Points Variability == Variable N = Maximum Points else N = Number of Points
- If Points Variability == Fixed : A = 11 when Number of Points == 64, A = 13 when Number of Points == 128, …, A = 27 when Number of Points == 16384
- If Points Variability == Variable : A = 11 when Maximum Points == 64, A = 13 when Maximum Points == 128, …, A = 27 when Maximum Points == 16384

2.3. Attribute Summary

The configurable attributes of the FFT Compiler IP Core are shown in Table 2.2 and are described in Table 2.3.The attributes can be configured through the IP Catalog's Module/IP wizard of the Lattice Radiant software.

Table 2.2. Attributes Table

Attribute	Selectable Values	Default	Dependency on Other Attributes
Points/Mode			
Number of Points			
Points Variability	Fixed, Variable	Fixed	—
Number of Points	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	64	*Points Variability == Fixed*
Maximum Points	128, 256, 512, 1024, 2048, 4096, 8192, 16384	128	*Points Variability == Variable* *Maximum Points* available are always greater than *Minimum Points* setting
Minimum Points	64,128, 256, 512, 1024, 2048, 4096, 8192	64	*Points Variability == Variable* *Minimum Points* available is always less than *Maximum Points* setting
Architecture			
Architecture	High Performance, Low Resource	Low Resource	—
FFT Mode			
FFT Mode	Forward, Inverse, Dynamic Through Port	Forward	—
Output Order			
Output order	Bit-reversed, Natural	Bit-reversed	—
Scaling/Width			
Scaling Mode			
Scaling Mode	None, RS111, RS211, Dynamic Through Port, Block Floating Point	RS111	Block Floating Point is only available when *Architecture == Low Resource*
Fix Last Stage Scaling to RS111 Truncation	True, False	True	Editable when *Scaling Mode* == Dynamic Through Port and *Architecture* == High performance
Data Width			
Input Data Width	8 to 24	16	—
Output Data Width	8 to 32 When *Scaling Mode* == Dynamic Through Port or *Scaling Mode* == None, *Output Data Width* will be *Input Data Width* + log2 [Points] , otherwise *Output data width* will be *Input Data Width*	16	Display information only.
Twiddle Factor Width	8 to 24	16	—
Precision Reduction Method			
Precision Reduction	Truncation, Rounding	Truncation	Editable when *Scaling Mode* != None
Truncate Last Stage	0,1	1	Editable only when *Architecture* == High performance, *Precision Reduction* == Rounding, *Scaling Mode* == RS111 or *Scaling Mode* == RS211

Attribute	Selectable Values	Default	Dependency on Other Attributes
Implementation			
Multiplier Type	DSP Block Based, LUT based	DSP Block Based	—
Multiplier Pipeline	2, 3, 4	3	Editable when *Architecture* == Low Resource and *Multiplier Type* == LUT based
Adder Pipeline	0, 1	0	Editable when *Architecture* == Low resource and *Scaling Mode* != Block Floating Point
Memory Type	EBR Memory, Distributed Memory, Automatic	EBR Memory	—

Table 2.3. Attributes Description

Attribute	Description
Points/Mode	
Points Variability	Allows you to specify fixed or variable number of points
Number of Points	Specifies the number of FFT points if *Points variability* == Fixed
Minimum Points	Denotes the minimum for the points range if *Points variability* == Variable
Maximum Points	Denotes the maximum for the points range if *Points variability* == Variable
Architecture	This option selects either High-Performance (streaming I/O) or Low Resource (Burst I/O) architecture.
FFT Mode	This parameter configures operating mode of the core.
Output order	This parameter specifies whether the output data is in bit-reversed or natural order. Each is described as: Natural order output: Output is directly fed to the following stage Bit Reversal: Applicable separately to the lower and upper half of the output. For an N point FFT, the first N/2 points are available in bit-reversed order first, followed by the second N/2 points in a bit-reversed order.
Scaling/Width	
Scaling Mode	This parameter defines whether the data is scaled or not after each radix-2 butterfly and if so, what kind of scaling is used. Each is describe as: • None: There is no scaling at the output of butterflies. • RS111: Results in a fixed scaling of right shift by 1 in all FFT stages. • RS211: Results in a fixed scaling of right shift by 2 in the first stage and right shift by 1 in the subsequent stages. • Dynamic Through Port: The scale factors for the FFT stages are read dynamically from the input port sfact_i for every data block. • Block Floating Point Scaling: The dynamic range for the intermediate computation is increased by extracting a common exponent for all the data points in each stage and using the full arithmetic width for processing only the mantissa. An additional output port exponent_o is added to the FFT compiler core when this option is selected. This option is not available for the High performance.
Fix Last Stage Scaling to RS111 Truncation	Can be enabled to improve scaling performance.
Data Width	
Input Data Width	Specifies input data width of either of the components: real or imaginary
Output Data Width	Specifies output data width of either of the components: real or imaginary
Twiddle Factor Width	Specifies twiddle factor width of either of the components: real or imaginary
Precision Reduction	Selects the scaling process to be applied after every stage. • Truncation: Truncating the data (discarding last one or two bits). This will result to less logic utilization. • Rounding: Rounding the data to the nearest number in the scaled precision (discarding one or two bits and making correction to the output based on discarded bits). This will improve the accuracy of the results.
Truncate Last Stage	This can be enabled to have better throughput.

Attribute	Description
Implementation	
Multiplier Type	This option specifies whether DSP blocks or LUTs are used for implementing multipliers and multiply-add components.
Adder Pipeline	This option is used to specify an additional pipeline after the adders. This can be enabled to have better performance at the cost of slightly increased utilization and latency.
Multiplier Pipeline	This option is used to specify the pipeline of LUT based multipliers. Higher values for pipeline leads to better performance at the cost of slightly increased utilization and latency.
Memory Type	This parameter specifies the balance between using EBR and distributed memories. If EBR memory is selected, EBRs are used for memory depths 32 and higher. If the Distributed Memory option is selected, EBR memories are used only for depths 512 or more and the rest uses distributed memories. In the automatic option, the IP generator uses a pre-defined setting to select the EBR and distributed memories based on the FFT parameters.

2.4. Interfacing with the FFT Compiler

2.4.1. Configuration Signals

There are three dynamic configuration signals used in the FFT compiler: mode_i, sfact_i, and points_i. You can independently select each of these. The configuration signals are sampled and stored when the corresponding set signals are active. Specifically, mode_i is set when modeset_i is active, sfact_i is set when sfactset_i is active, and points_i is set when pointset_i becomes active. However, these values must be set during or before the start of a block for them to be effective for that block. In other words, the set signals must be active at or before ibstart_i going active, for them to be effective for that block. There are a couple of exceptions to this rule. In low-resource implementation and in bit-reversed output mode, the set signals are ignored when outvalid_o is high. Refer to Figure 2.5 for an illustration of configuration signal timing. In High Performance implementation, the pointset_i must be applied five cycles before ibstart_i for it to be effective for the next block.

2.4.2. Handshake Signals

The input ibstart_i is used to specify the start of a data block and it is assumed to coincide with first data point in the input block. This signal also sets the configuration of the core based on the values set by the corresponding set signals. Once ibstart_i is valid, the core starts reading the input in consecutive clocks without gap until all the N- points are read. When the last data in a block is being read from input, the output ibend_o is asserted high by the core. The output control signal rfib_o indicates that the core is ready for a new input block. One cycle after ibstart_i, the output rfib_o goes low and it remains low until one cycle before the next block can be applied. The external driving system can check for rfib_o and start an input block in the next cycle after rfib_o goes high. Refer to Figure 2.5 for an illustration of configuration signal timing.

2.4.3. Exponent Output

The output port exponent_o gives the value of the exponent of the multiplicative factor for the output to get the true FFT output. The value of exponent is an unsigned number. The true (i)FFT output is given by:

$$\text{True (i)FFT output} = (\text{dore\_o} \times 2^{\text{exponent\_o}}) + j\,(\text{doim\_o} \times 2^{\text{exponent\_o}})$$

2.4.4. Exceptions

Exceptions occur if there is an internal overflow in the computation of an output block. An exception is notified by the except_o signal going high during a valid output. The except_o signal goes high if one or more overflows occur during the computation of a block. The severity and number of overflow exceptions in a block depends on the scaling scheme used and the property of the input data. If the user is using an appropriate scaling method for the expected input data and can tolerate occasional exceptions, the except_o output may be left unconnected leading to a slightly reduced resource utilization.

2.5. Timing Specifications

The top-level timing diagrams for several cases are given in Figure 2.4 and Figure 2.6.

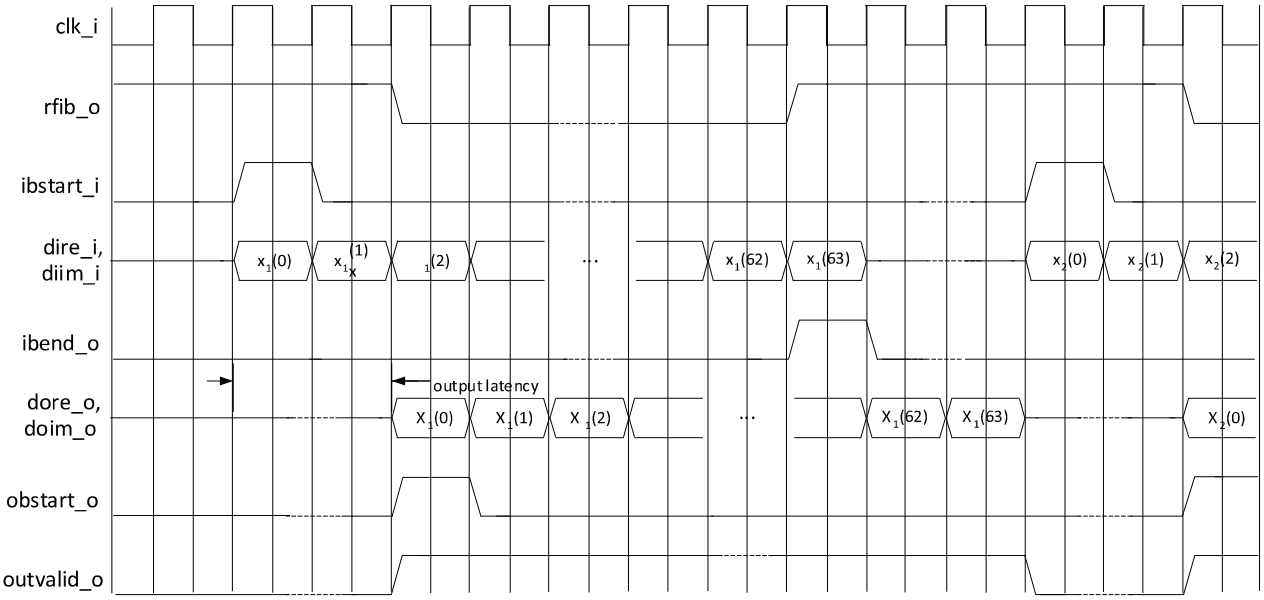

Figure 2.4. Timing Diagram for Streaming I/O for 64 Points

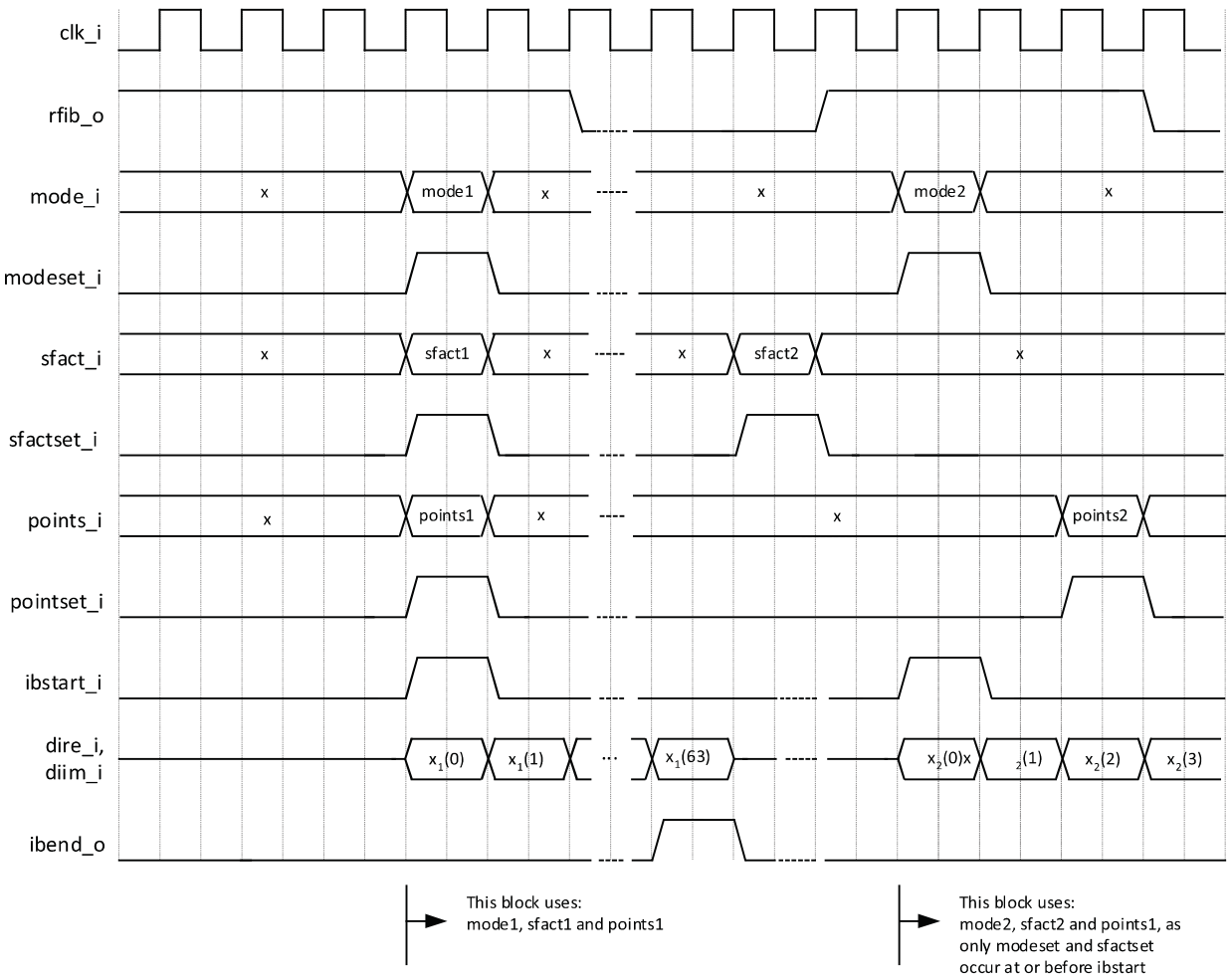

Figure 2.5. Timing Diagram Showing Handshake Signals for Low Resource FFT

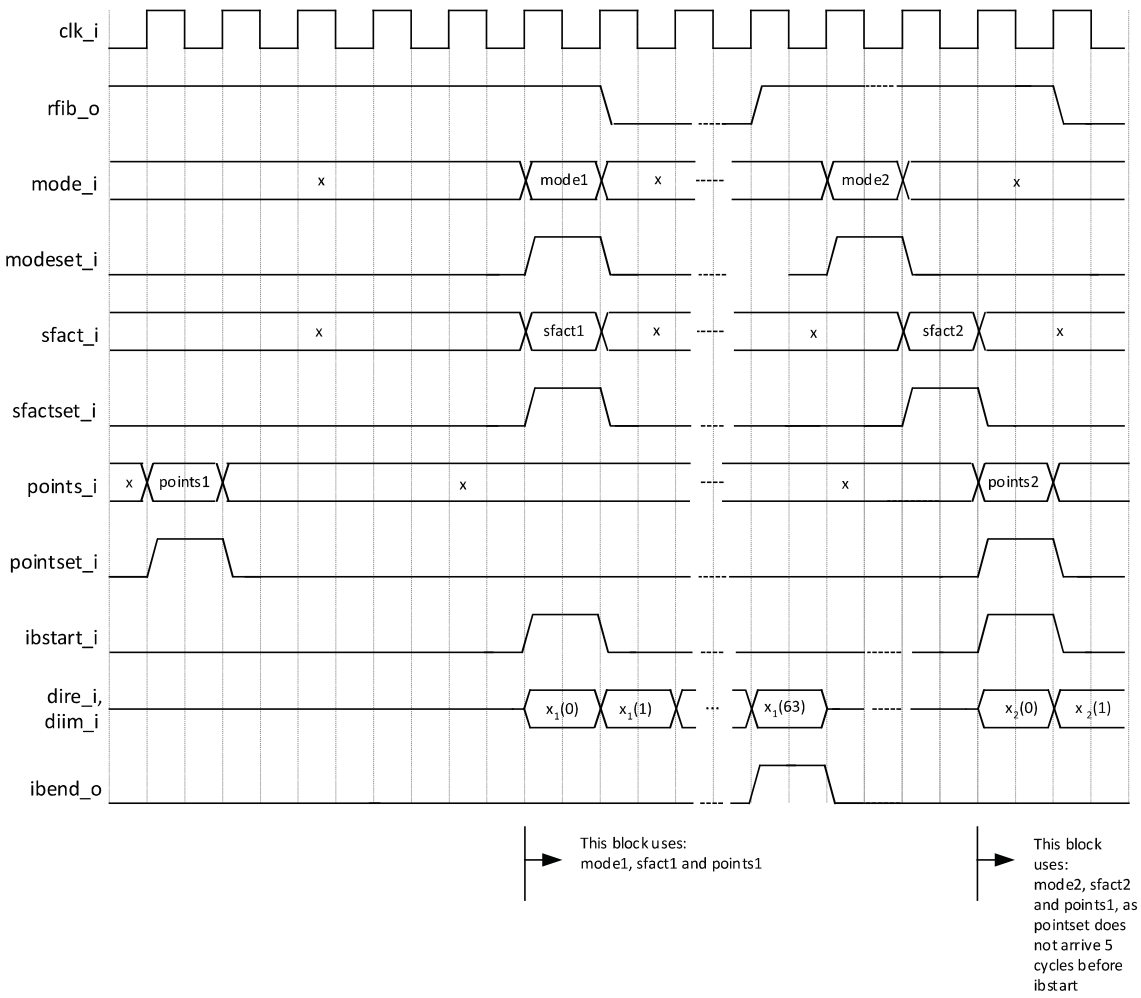

Figure 2.6. Timing Diagram Showing Handshake Signals for High Performance FFT

2.6. Output Latency

Table 2.44 provides the latency through the IP Core as a function of FFT point size and implementation mode.

Table 2.4. Local User Interface Functional Groups

FFT Point Size	Low Resource Mode	High Performance Mode
64	278	83
128	598	152
256	1302	282
512	2838	543
1024	6166	1057
2048	13334	2086
4096	28694	4136
8192	61462	8237
16384	131094	16431

3. IP Generation and Evaluation

This section provides information on how to generate the 2D Scaler IP Core using the Lattice Radiant software and how to run simulation and synthesis. For more details on the Lattice Radiant software, refer to the Lattice Radiant Software User Guide.

3.1. Licensing the IP

An IP core-specific license string is required enable full use of the FFT Compiler IP Core in a complete, top-level design. You can fully evaluate the IP Core through functional simulation and implementation (synthesis, map, place and route) without an IP license string. This IP Core supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core, which operate in hardware for a limited time (approximately four hours) without requiring an IP license string. See Hardware Evaluation section for further details. However, a license string is required to enable timing simulation and to generate bitstream file that does not include the hardware evaluation timeout limitation.

3.2. Generation and Synthesis

The Lattice Radiant software allows you to customize and generate modules and IPs and integrate them into the device's architecture. The procedure for generating the FFT Compiler IP Core in Lattice Radiant software is described below.

To generate the FFT Compiler IP Core:

1. Create a new Lattice Radiant software project or open an existing project.
2. In the **IP Catalog** tab, double-click on **FFT Compiler** under **IP**, **DSP** category. The **Module/IP Block Wizard** opens as shown in Figure 3.1. Enter values in the **Component name** and the **Create in** fields and click **Next**.

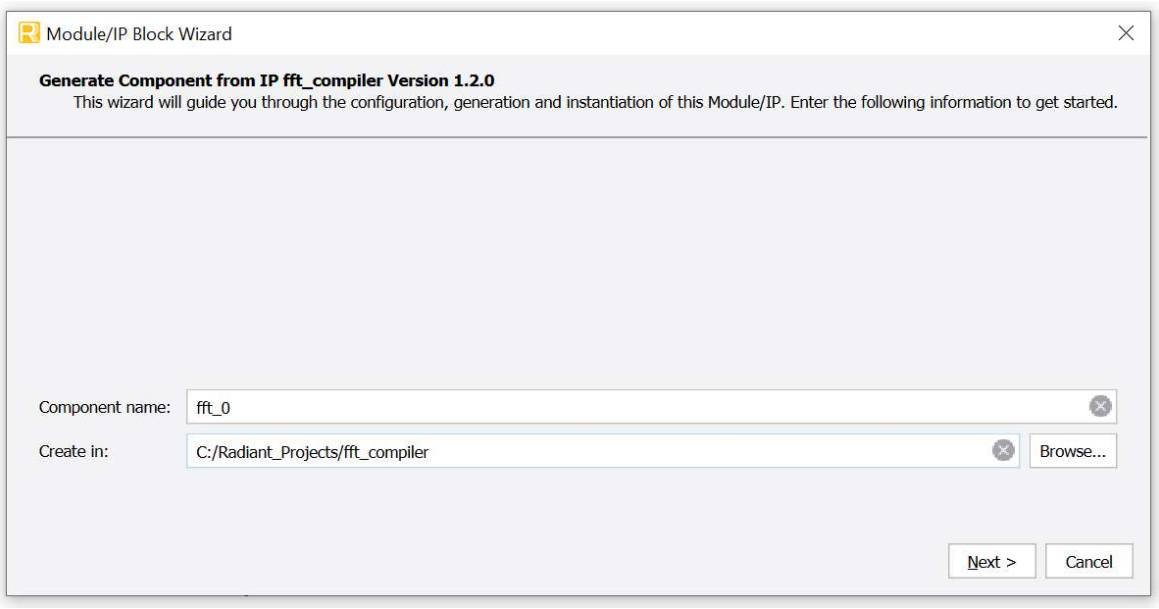

Figure 3.1. Module/IP Block Wizard

3. In the module's dialog box of the **Module/IP Block Wizard** window, customize the selected FFT Compiler IP Core using drop-down menus and check boxes. As a sample configuration, see Figure 3.2. For configuration options, see the Attribute Summary section.

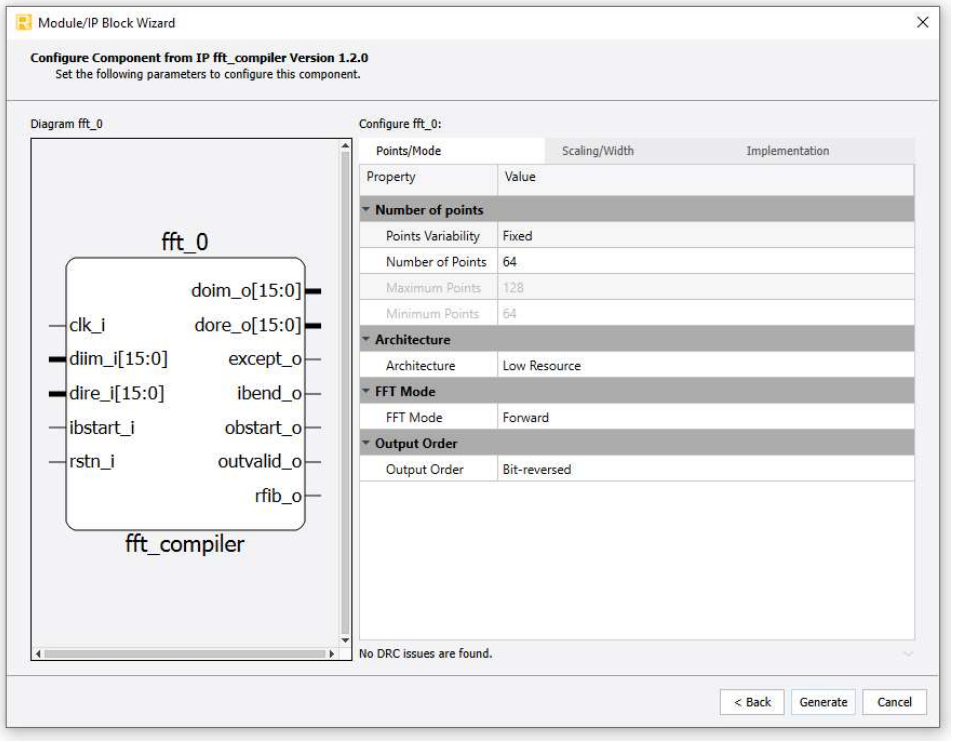

Figure 3.2. Configure User Interface of FFT Compiler IP Core

4. Click **Generate**. The **Check Generated Result** dialog box opens, showing design block messages and results as shown in Figure 3.3.

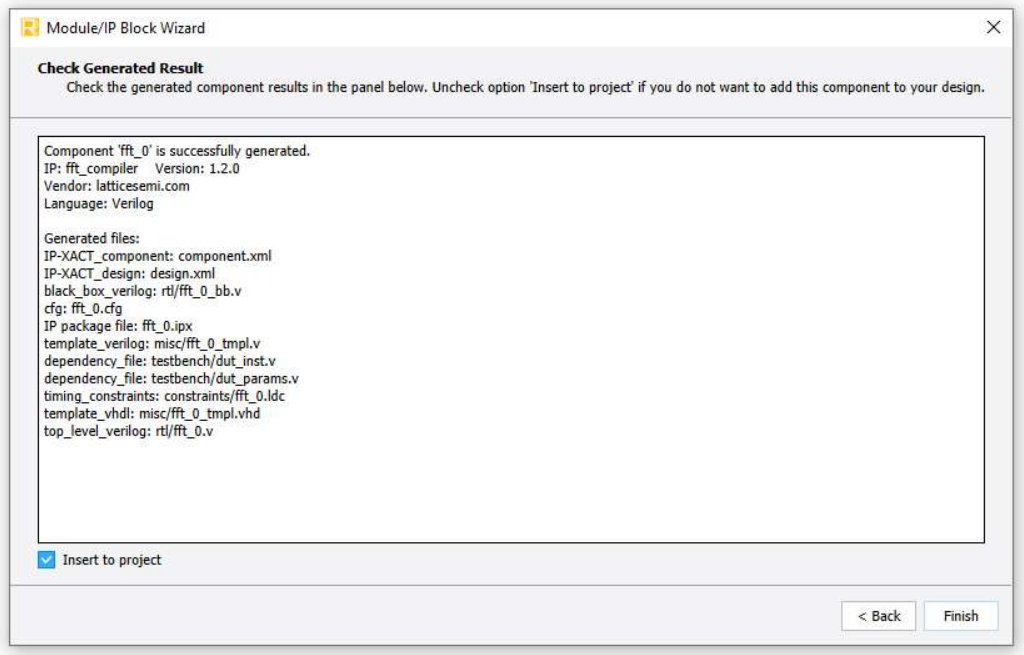

Figure 3.3. Check Generated Result

5. Click the **Finish** button. All the generated files are placed under the directory paths in the **Create in** and the **Component name** fields shown in Figure 3.1.

The generated FFT Compiler IP Core package includes the black box (<Component name>_bb.v) and instance templates (<Component name>_tmpl.v/vhd) that can be used to instantiate the core in a top-level design. An example RTL top-level reference source file (<Component name>.v) that can be used as an instantiation template for the IP Core is also provided. You may also use this top-level reference as the starting template for the top-level for their complete design. The generated files are listed in Table 3.1.

Table 3.1. Generated File List

Attribute	Description
<Component name>.ipx	This file contains the information on the files associated to the generated IP.
<Component name>.cfg	This file contains the parameter values used in IP configuration.
component.xml	Contains the ipxact:component information of the IP.
design.xml	Documents the configuration parameters of the IP in IP-XACT 2014 format.
rtl/<Component name>.v	This file provides an example RTL top file that instantiates the IP core.
rtl/<Component name>_bb.v	This file provides the synthesis black box.
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	These files provide instance templates for the IP core.

3.3. Running Functional Simulation

After the IP is generated, running functional simulation can be performed using different available simulators. The default simulator already has pre-compiled libraries ready for simulation. Choosing a non-default simulator, however, may require additional steps.

To run functional simulation using default simulator:

1. Click the  button located on the **Toolbar** to initiate the **Simulation Wizard** shown in Figure 3.4.

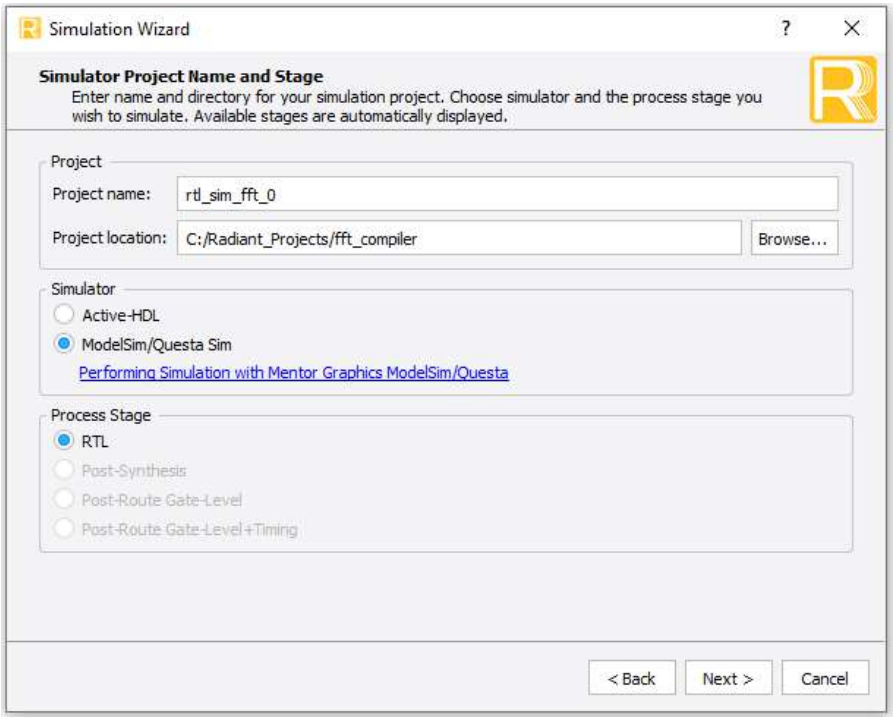

Figure 3.4. Simulation Wizard

2. Click **Next** to open the **Add and Reorder Source** window as shown in Figure 3.5.

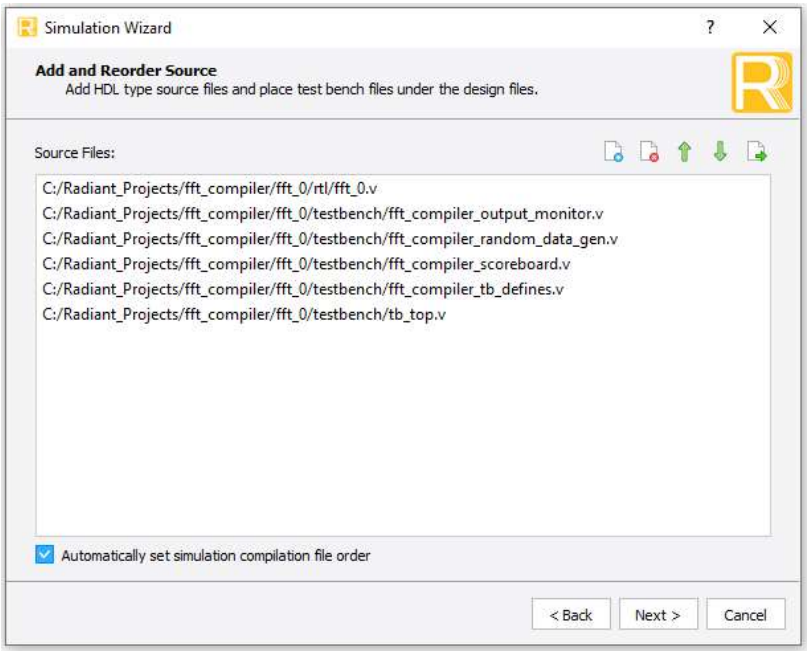

Figure 3.5. Adding and Reordering Source

3. Click **Next**. The **Summary** window is shown. Click **Finish** to run the simulation.

Note: It is necessary to follow the procedure above until it is fully automated in the Lattice Radiant software suite.

The results of the simulation in our example are provided in Figure 3.6.

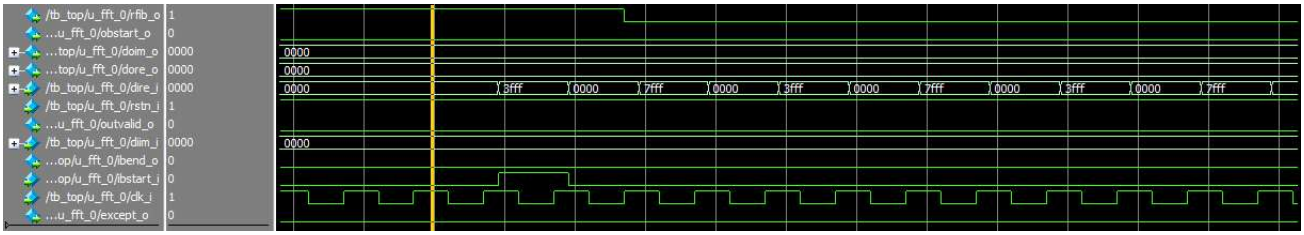

Figure 3.6. Simulation Waveform

3.4. Hardware Evaluation

The FFT Compiler IP Core supports Lattice's IP hardware evaluation capability when used with Lattice FPGA devices built on the Lattice Nexus™ platform. This makes it possible to create versions of the IP Core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default. To change this setting, go to Project > Active Strategy > LSE/Synplify Pro Settings.

4. Ordering Part Number

The Ordering Part Number (OPN) for this IP Core are the following:

- FFT-COMP-CNX-U – FFT Compiler for CrossLink-NX – Single Design License
- FFT-COMP-CNX-UT – FFT Compiler for Crosslink-NX – Site License
- FFT-COMP-CTNX-U – FFT Compiler for Certus-NX – Single Design License
- FFT-COMP-CTNX-UT – FFT Compiler for Certus-NX – Site License
- FFT-COMP-CPNX-U – FFT Compiler for CertusPro-NX – Single Design License
- FFT-COMP-CPNX-UT – FFT Compiler for CertusPro-NX – Site License
- FFT-COMP-XO5-U – FFT Compiler for MachXO5-NX - Single Design License
- FFT-COMP-XO5-UT – FFT Compiler for MachXO5-NX - Site License
- FFT-COMP-XO5-US – FFT Compiler for MachXO5-NX - 1 Year Subscription License
- FFT-COMP-AVE-U – FFT Compiler for Avant-E – Single Design License
- FFT-COMP-AVE-UT – FFT Compiler for Avant-E – Site License
- FFT-COMP-AVE-US – FFT Compiler for Avant-E – 1-Year Subscription License

Appendix A. Resource Utilization

Table A.1 shows the resource utilization of the FFT Compiler IP Core for the LFMXO5-25-9BBG400I device using Synplify Pro of Lattice Radiant Software 2022.1. Default configuration is used, and some attributes are changed from the default value to show the effect on the resource utilization.

Table A.1. Resource Utilization (For LFMXO5-25-9BBG400I)

Configuration	Clk Fmax (MHz)[1]	Registers	LUTs	EBRs	DSPs[2]
Default	200	999	673	3	4
Architecture: High Performance, Others = Default	200	1128	1527	1	8
Architecture: High Performance, *Multiplier Type*: LUT-based	131.631	2338	4343	1	0
FFT Mode: Dynamic Through Port, Others = Default	200	1004	674	3	4
Input Data Width: 24, *Twiddle Factor Width*: 24, Others = Default	200	1483	973	6	16
Multiplier Type: LUT-based, *Memory Type*: Distributed Memory, Others = Default	140.154	1280	2364	0	0

Note:

1. Fmax is generated when the FPGA design only contains FFT Compiler IP Core, and the target frequency is 100MHz. These values may be reduced when user logic is added to the FPGA design.
2. Number of Multipliers

Table A.2 shows the resource utilization of the FFT Compiler IP Core for the LFMXO5-25-7BBG400I device using Synplify Pro of Lattice Radiant Software 2022.1. Default configuration is used, and some attributes are changed from the default value to show the effect on the resource utilization.

Table A.2. Resource Utilization (For LFMXO5-25-7BBG400I)

Configuration	Clk Fmax (MHz)[1]	Registers	LUTs	EBRs	DSPs[2]
Default	164.772	999	673	3	4
Architecture: High Performance, Others = Default	167.757	1128	1527	1	8
Architecture: High Performance, *Multiplier Type*: LUT-based	84.402	2110	4349	1	0
FFT Mode: Dynamic Through Port, Others = Default	171.527	1004	674	3	4
Input Data Width: 24, *Twiddle Factor Width*: 24, Others = Default	162.153	1483	973	6	16
Multiplier Type: LUT-based, *Memory Type*: Distributed Memory, Others = Default	85.121	1280	2364	0	0

Note:

1. Fmax is generated when the FPGA design only contains FFT Compiler IP Core, and the target frequency is 100MHz. These values may be reduced when user logic is added to the FPGA design.
2. Number of Multipliers

Table A.3 shows the resource utilization of the FFT Compiler IP Core for the LAV-AT-500E-3LFG1156I device using Synplify Pro of Lattice Radiant Software 2022.1. Default configuration is used, and some attributes are changed from the default value to show the effect on the resource utilization.

Table A.3. Resource Utilization (for LAV-AT-500E-3LFG1156I)

Configuration	Clk Fmax (MHz)[1]	Registers	LUTs	EBRs	DSPs
Default	251.762	830	638	3.0	4
Architecture: High Performance, Others = Default	-	-	-	-	-
Architecture: High Performance, *Multiplier Type*: LUT-based	156.323	1674	4021	1.0	0
FFT Mode: Dynamic Through Port, Others = Default	251.762	835	639	3.0	4
Input Data Width: 24, *Twiddle Factor Width*: 24, Others = Default	185.632	1921	1156	3.0	16
Multiplier Type: LUT-based, *Memory Type*: Distributed Memory, Others = Default	178.508	1296	2163	0	0

Note:

1. Fmax is generated when the FPGA design only contains FFT Compiler IP Core, and the target frequency is 100MHz. These values may be reduced when user logic is added to the FPGA design.

Table A.4 shows the resource utilization of the FFT Compiler IP Core for the LAV-AT-500E-1LFG1156I device using Synplify Pro of Lattice Radiant Software 2022.1. Default configuration is used, and some attributes are changed from the default value to show the effect on the resource utilization.

Table A.4. Resource Utilization (for LAV-AT-500E-1LFG1156I)

Configuration	Clk Fmax (MHz)[1]	Registers	LUTs	EBRs	DSPs
Default	225.887	830	638	3.0	4
Architecture: High Performance, Others = Default	-	-	-	-	-
Architecture: High Performance, *Multiplier Type*: LUT-based	145.815	1674	4021	1.0	0
FFT Mode: Dynamic Through Port, Others = Default	237.304	835	639	3.0	4
Input Data Width: 24, *Twiddle Factor Width*: 24, Others = Default	162.655	1921	1156	3.0	16
Multiplier Type: LUT-based, *Memory Type*: Distributed Memory, Others = Default	165.673	1296	2163	0	0

Note:

1. Fmax is generated when the FPGA design only contains FFT Compiler IP Core, and the target frequency is 100MHz. These values may be reduced when user logic is added to the FPGA design.

Appendix B. Limitations

The following configurations are not yet support for Avant devices:

- 'High Performance' Architecture with 'DSP Block Based' Multiplier Type Implementation

References

- CrossLink-NX FPGA Web Page at www.latticesemi.com
- Certus-NX FPGA-Web Page at www.latticesemi.com
- CertusPro-NX FPGA Web Page at www.latticesemi.com

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Document Revision 1.3, Lattice Radiant SW Version 2022.1, November 2022

Section	Change Summary
Introduction	Updated Table 1.1. Quick Facts: • Added Lattice Avant to Supported FPGA Families. • Added LAV-AT-500E to Targeted Devices.
Appendix A: Resource Utilization	• Updated *LFMXO5-25-9BBG400I* and *LFMXO5-25-7BBG400I* resource data using Radiant 2022.1. • Added *LAV-AT-500E-3LFG1156I* and *LAV-AT-500E-1LFG1156I.*
Ordering Part Number	Added part numbers for Avant-E.
Appendix B: Limitations	Added IP Configuration limitation for Avant devices.

Document Revision 1.2, Lattice Radiant SW Version 3.2, May 2022

Section	Change Summary
Introduction	Updated Table 1.1. Quick Facts • Added MachXO5-NX to Supported FPGA Families • Added LFMXO5-25 to Targeted Devices
IP Generation and Evaluation	Updated Figure 3.1. Module/IP Block Wizard, Figure 3.2. Configure User Interface of FFT Compiler IP Core, and Figure 3.3. Check Generated Result.
Ordering Part Number	Added the following part numbers: • FFT-COMP-XO5-U - FFT Compiler for MachXO5-NX - Single Design License • FFT-COMP-XO5-UT - FFT Compiler for MachXO5-NX - Site License • FFT-COMP-XO5-US - FFT Compiler for MachXO5-NX - 1 Year Subscription License
Appendix A: Resource Utilization	Updated resource utilization for *LFMXO5-25-9BBG400I* and *LFMXO5-25-7BBG400I*.

Document Revision 1.1, Lattice Radiant SW Version 3.0, June 2021

Section	Change Summary
All	Minor adjustments in formatting.
Introduction	Updated section content, including Table 1.1 to add CertusPro-NX support.
Ordering Part Number	Added part numbers for CertusPro-NX.
References	Added webpage for CertusPro-NX.

Document Revision 1.0, Lattice Radiant SW Version 2.1, December 2020

Section	Change Summary
All	Initial release

www.latticesemi.com