



FFT Compiler IP

IP Version: v1.6.0

User Guide

FPGA-IPUG-02153-1.8

December 2025

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 all associated risk is the responsibility entirely of the Buyer. The information provided herein is for informational purposes only and may contain technical inaccuracies or omissions, and may be otherwise rendered inaccurate for many reasons, and Lattice assumes no obligation to update or otherwise correct or revise this information. 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. LATTICE PRODUCTS AND SERVICES ARE NOT DESIGNED, MANUFACTURED, OR TESTED FOR USE IN LIFE OR SAFETY CRITICAL SYSTEMS, HAZARDOUS ENVIRONMENTS, OR ANY OTHER ENVIRONMENTS REQUIRING FAIL-SAFE PERFORMANCE, INCLUDING ANY APPLICATION IN WHICH THE FAILURE OF THE PRODUCT OR SERVICE COULD LEAD TO DEATH, PERSONAL INJURY, SEVERE PROPERTY DAMAGE OR ENVIRONMENTAL HARM (COLLECTIVELY, "HIGH-RISK USES"). FURTHER, BUYER MUST TAKE PRUDENT STEPS TO PROTECT AGAINST PRODUCT AND SERVICE FAILURES, INCLUDING PROVIDING APPROPRIATE REDUNDANCIES, FAIL-SAFE FEATURES, AND/OR SHUT-DOWN MECHANISMS. LATTICE EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS OF THE PRODUCTS OR SERVICES FOR HIGH-RISK USES. 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.

Inclusive Language

This document was created consistent with Lattice Semiconductor's inclusive language policy. In some cases, the language in underlying tools and other items may not yet have been updated. Please refer to Lattice's inclusive language [FAQ 6878](#) for a cross reference of terms. Note in some cases such as register names and state names it has been necessary to continue to utilize older terminology for compatibility.

Contents

Contents	3
Abbreviations in This Document.....	6
1. Introduction	7
1.1. Overview of the IP	7
1.2. Quick Facts	7
1.3. IP Support Summary	7
1.4. Features	8
1.5. Licensing and Ordering Information	8
1.6. Hardware Support	9
1.7. Minimum Device Requirements	9
1.8. Naming Conventions	9
1.8.1. Nomenclature	9
1.8.2. Signal Names	9
1.8.3. Attribute Names	9
2. Functional Description.....	10
2.1. IP Architecture Overview	10
2.1.1. High-Performance Architecture	11
2.1.2. High-Performance Radix-4 Architecture	11
2.1.3. Low Resource Architecture.....	11
2.2. Clocking	12
2.3. Reset	12
2.4. Interfacing with the FFT Compiler	13
2.4.1. Handshake Signals	13
2.4.2. Configuration Signals.....	14
2.4.3. Exponent Output	15
2.4.4. Exceptions	15
2.5. Output Scaling.....	16
2.6. Output Latency.....	17
3. IP Parameter Description.....	19
3.1. Points/Mode Attributes	19
3.2. Scaling/Width Attributes.....	20
3.3. Implementation Attributes	21
3.4. IP Parameter Settings for Example Use Cases.....	22
4. Signal Description	23
5. Example Design.....	25
5.1. Example Design Supported Configuration	25
5.2. Overview of the Example Design and Features	25
5.3. Example Design Components.....	26
5.4. Generating the Example Design	26
5.4.1. Using the Example Design Sample Project	27
5.4.2. Changing Configuration of the Example Design	28
5.4.3. Limitations of the Example Design	28
5.5. Simulating the Example Design	28
5.6. Hardware Testing	30
6. Designing with the IP	31
6.1. Generating and Instantiating the IP	31
6.1.1. Generated Files and File Structure	33
6.2. Design Implementation.....	33
6.3. Timing Constraints	33
6.4. Running Functional Simulation	33
6.4.1. Simulation Results	35
Appendix A. Resource Utilization	37

References	42
Technical Support Assistance	43
Revision History.....	44

Figures

Figure 2.1. FFT Compiler Interface Diagram	10
Figure 2.2. Implementation Diagram for High-Performance FFT	11
Figure 2.3. Low-Resource FFT Data Flow Diagram	12
Figure 2.4. High-Performance (Streaming I/O) for 64 Points Timing Diagram	13
Figure 2.5. Low Resource (Burst I/O) for 64 Points Timing Diagram	13
Figure 2.6. Handshake Signals for High-Performance and High-Performance Radix-4 FFT Timing Diagram	14
Figure 2.7. Handshake Signals for Low Resource FFT Timing Diagram	15
Figure 2.8. Exception Timing Diagram	16
Figure 5.1. FFT Compiler Example Design Block Diagram.....	26
Figure 5.2. Example Design Project File List	27
Figure 5.3. Adding Debug Files to Example Design.....	27
Figure 5.4. Simulation Wizard for Example Design.....	28
Figure 5.5. Add and Reorder Source for Example Design	29
Figure 5.6. Example Design Simulation Waveform.....	29
Figure 5.7. 1024-Point FFT Compiler IP Sample Output	30
Figure 5.8. Input Data and Handshake Signals	30
Figure 5.9. Output Data and Handshake Signals	30
Figure 6.1. Module/IP Block Wizard	31
Figure 6.2. IP Configuration	32
Figure 6.3. Check Generated Result	32
Figure 6.4. Simulation Wizard.....	34
Figure 6.5. Add and Reorder Source	34
Figure 6.6. Simulation Waveform	35
Figure 6.7. FFT Compiler Simulation Testbench Block Diagram	35
Figure 6.8. Simulation Waveform for Input Data Block (Forward FFT).....	36
Figure 6.9. Simulation Waveform for Output Data Block (Forward FFT).....	36
Figure 6.10. Simulation Waveform for Input Data Block (Inverse FFT)	36
Figure 6.11. Simulation Waveform for Output Data Block (Inverse FFT)	36

Tables

Table 1.1. Summary of the FFT Compiler IP.....	7
Table 1.2. FFT Compiler IP Support Readiness	7
Table 2.1. Output Latency for Bit-Reversed Output Order based on FFT Point Size	17
Table 2.2. Output Latency for Natural Output Order based on FFT Point Size.....	18
Table 3.1. Points/Mode Attributes	19
Table 3.2. Scaling/Width Attributes	20
Table 3.3. Implementation Attributes	21
Table 3.4 FFT Compiler IP Example Use Cases.....	22
Table 4.1. Top level I/O interface	23
Table 5.1. FFT Compiler IP Configuration Supported by the Example Design	25
Table 5.2 Generated File List for Example Design	26
Table 6.1. Generated File List	33
Table A.1. LAV-AT-E70-1LFG1156C Device Resource Utilization	37
Table A.2. LIFCL-33-8USG84C Device Resource Utilization	38

Table A.3. LFD2NX-9-7MG121C Device Resource Utilization38

Table A.4. LFD2NX-17-7MG121C Device Resource Utilization39

Table A.5. LFD2NX-28-7MG121C Device Resource Utilization39

Table A.6. LFD2NX-40-7MG121C Device Resource Utilization40

Table A.7. LN2-CT-20-1CBG484C Device Resource Utilization40

Table A.8. LFCPNX-100-8LFG672I Device Resource Utilization41

Abbreviations in This Document

A list of abbreviations used in this document.

Abbreviations	Definition
bfly	Butterfly
BFM	Bus Functional Model
EBR	Embedded Block RAM
DIF	Decimation-in-Frequency
DFT	Discrete Fourier Transform
DSP	Digital Signal Processor
DUT	Design Under Test
FFT	Fast Fourier Transform
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
I/O	Input/Output
IFFT	Inverse Fast Fourier Transform
IP	Intellectual Property
GUI	Graphical User Interface
HDL	Hardware Description Language
LVDS	Low Voltage Differential Signaling
mem	Memory
PE	Processing Element
RAM	Random Access Memory
ROM	Read Only Memory

1. Introduction

1.1. Overview of the IP

The Lattice Semiconductor Fast Fourier Transform (FFT) Compiler IP Core provides 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. There are three modes of implementation: High-Performance (Streaming I/O), High-Performance Radix-4 (Low Latency 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.

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.

1.2. Quick Facts

Table 1.1. Summary of the FFT Compiler IP

IP Requirements	Supported Devices	CrossLink™-NX, Certus™-NX, CertusPro™-NX, Certus-NX-RT, CertusPro-NX-RT, MachXO5™-NX, Lattice Avant™, and Certus-N2.
	IP Changes ¹	For a list of changes to the IP, refer to the FFT Compiler IP Release Notes (FPGA-RN-02069) .
Resource Utilization	Resources	See Appendix A. Resource Utilization
Design Tool Support	Lattice Implementation	IP Core v1.6.0 – Lattice Radiant™ Software 2025.2 and Lattice Propel™ Builder 2025.2
	Synthesis	Lattice Synthesis Engine Synopsys Synplify Pro® for Lattice
	Simulation	For a list of supported simulators, see the Lattice Radiant Software user guide.

Note:

1. In some instances, the IP may be updated without changes to the user guide. This user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

1.3. IP Support Summary

Table 1.2. FFT Compiler IP Support Readiness

Device Family	Clock Frequency ¹	Architecture	FFT Mode	Output Order	Scaling Mode	Radiant Timing Model	Hardware Validated
Avant	80 MHz	High-Performance	Forward	Natural	None	Preliminary	Yes
			Forward	Natural	RS111	Preliminary	Yes
			Forward	Natural	Dynamic Through Port	Preliminary	No
			Forward	Bit-reversed	None	Preliminary	Yes
			Forward	Bit-reversed	RS111	Preliminary	Yes
			Forward	Bit-reversed	Dynamic Through Port	Preliminary	No
			Inverse	Natural	None	Preliminary	Yes
			Inverse	Natural	RS111	Preliminary	Yes

Device Family	Clock Frequency ¹	Architecture	FFT Mode	Output Order	Scaling Mode	Radiant Timing Model	Hardware Validated
			Inverse	Natural	Dynamic Through Port	Preliminary	No
			Inverse	Bit-reversed	None	Preliminary	Yes
			Inverse	Bit-reversed	RS111	Preliminary	Yes
			Inverse	Bit-reversed	Dynamic Through Port	Preliminary	No
Avant	80 MHz	Low Resource	Forward	Natural	None	Preliminary	Yes
			Forward	Natural	RS111	Preliminary	Yes
			Forward	Natural	Dynamic Through Port	Preliminary	No
			Forward	Natural	Block Floating Point	Preliminary	Yes
			Forward	Bit-reversed	None	Preliminary	Yes
			Forward	Bit-reversed	RS111	Preliminary	Yes
			Forward	Bit-reversed	Dynamic Through Port	Preliminary	No
			Forward	Bit-reversed	Block Floating Point	Preliminary	Yes
			Inverse	Natural	None	Preliminary	Yes
			Inverse	Natural	RS111	Preliminary	Yes
			Inverse	Natural	Dynamic Through Port	Preliminary	No
			Inverse	Natural	Block Floating Point	Preliminary	Yes
			Inverse	Bit-reversed	None	Preliminary	Yes
			Inverse	Bit-reversed	RS111	Preliminary	Yes
			Inverse	Bit-reversed	Dynamic Through Port	Preliminary	No
			Inverse	Bit-reversed	Block Floating Point	Preliminary	Yes
Avant	80 MHz	High Performance Radix-4	Forward	Natural	None	Preliminary	Yes
			Forward	Natural	Block Floating Point	Preliminary	Yes
			Inverse	Natural	None	Preliminary	Yes
			Inverse	Natural	Block Floating Point	Preliminary	Yes

Notes:

1. This is the system clock frequency used during hardware validation. For the actual frequency supported by the IP, refer to [Appendix A. Resource Utilization](#).

1.4. Features

Key features of the FFT Compiler IP include:

- Wide range of point sizes: 64, 128, 256, 512, 1024, 2048, 4096, 8192, and 16384.
- Choice of High-Performance (streaming I/O), High-Performance Radix-4 (low latency streaming I/O), or Low Resource (burst I/O) architecture.
- 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 and High-Performance Radix-4 configurations.

1.5. Licensing and Ordering Information

The FFT Compiler IP is provided at no additional cost with the Lattice Radiant software.

1.6. Hardware Support

Refer to the [Example Design](#) section for more information on the boards used.

1.7. Minimum Device Requirements

There is no limitation in device speed grade for FFT Compiler IP. See [Appendix A. Resource Utilization](#) for minimum required resources to instantiate this IP and maximum clock frequency supported.

1.8. Naming Conventions

1.8.1. Nomenclature

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

1.8.2. Signal Names

- `_n` are active low (asserted when value is logic 0)
- `_i` are input signals
- `_o` are output signals

1.8.3. Attribute Names

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

2. Functional Description

2.1. IP Architecture Overview

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

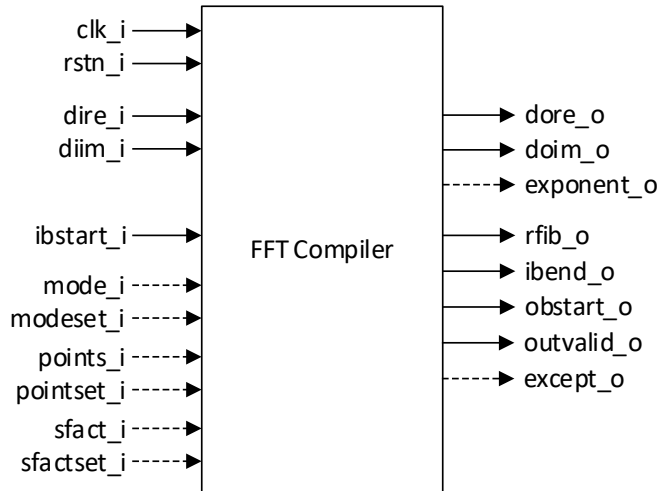


Figure 2.1. FFT Compiler Interface Diagram

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

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

where W_N is given by:

$$W_N = e^{-\frac{j2\pi}{N}} \quad (2)$$

The inverse DFT is given by:

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

The output of the FFT Compiler IP Core may differ from the true FFT results shown in Equations (1) and (3) due to a scaling factor determined by the selected scaling mode. For more information on the available scaling options, refer to the [Scaling/Width Attributes](#) section. To accurately compute the true FFT output, see the [Output Scaling](#) section.

There are three available architectures: High-Performance, High-Performance Radix-4, and Low Resource. In High-Performance and High-Performance Radix-4 modes, the FFT Compiler IP Core can continuously process one data sample per clock cycle, block after block, achieving a throughput equal to the clock rate when data blocks are applied without interruption. The FFT throughput is equal to the clock rate when data blocks are applied continuously. The Low Resource mode uses less logic and memory resources but requires 4 to 8 block periods to compute the FFT for a single block of input data. All architectures do not allow breaks in the data stream within each block, but 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](#).

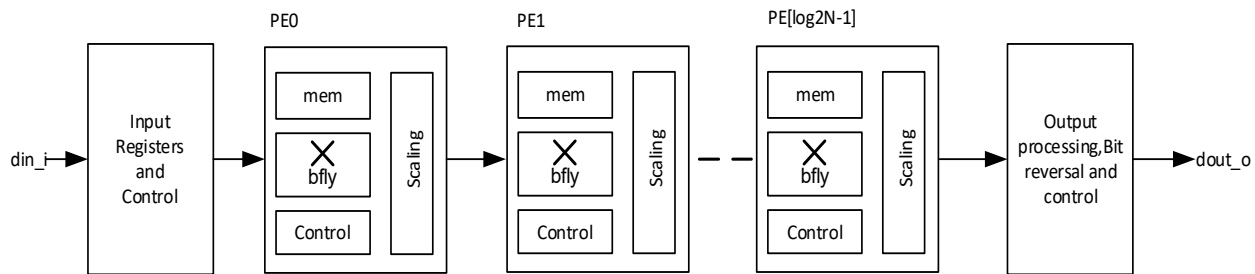


Figure 2.2. Implementation Diagram for High-Performance FFT

The High-Performance FFT implementation consists of several processing elements (PEs) connected in cascade. The number of PEs is equal to $\log_2 N$, where N is the number of FFT points. Each PE has a radix-2 decimation-in-frequency (DIF) butterfly (bfly), a memory (mem) unit, 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 signals, 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. For more information on this operation, refer to the [Handshake Signals](#) section.

2.1.2. High-Performance Radix-4 Architecture

High-Performance Radix-4 architecture extends the capabilities of the High-Performance architecture. The implementation diagram remains identical to [Figure 2.2](#) but utilizes a radix-4 butterfly instead of a radix-2 butterfly. Each PE comprises a radix-4 DIF butterfly with twiddle multiplier and twiddle factor memory, a memory unit, and an address generation and control logic. The number of PEs required is determined by the formula $\log_4 N$, where N is the number of FFT points. For FFT sizes where N is a power of 4, this ensures complete combination of all FFT points. In cases where N is not a power of 4, the architecture appends a single radix-2 PE after the radix-4 stages to handle the remaining computation. The handshake signals are consistent with those used in the High-Performance architecture.

To reduce the output latency of FFT, this architecture can be utilized, noting that computational complexity and resource utilization will increase.

2.1.3. 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](#).

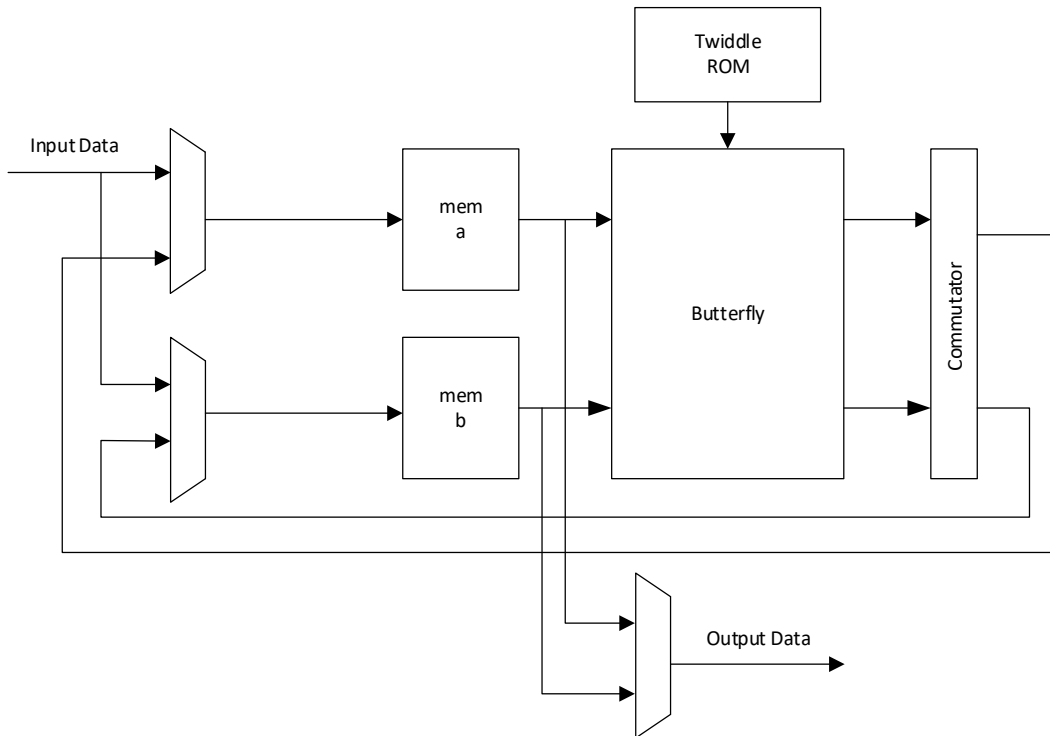


Figure 2.3. Low-Resource FFT Data Flow Diagram

As shown in [Figure 2.3](#), 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 are required for this scheme. Due to the twin memory architecture, when data is unloaded from 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. Clocking

The FFT Compiler IP requires one system clock, *clk_i*. The system clock provides a timing reference and is used to operate the IP's internal logic. Refer to [Appendix A. Resource Utilization](#) for the maximum supported frequency.

2.3. Reset

The FFT Compiler IP requires one system reset, *rst_n_i*, which is an asynchronous active low reset. The reset assertion can be asynchronous, but the reset negation must be synchronous. When *rst_n_i* is asserted, output ports are forced to their reset values and IP core is reset. To ensure proper propagation within the IP, *rst_n_i* must be asserted for at least two clock cycles.

2.4. Interfacing with the FFT Compiler

2.4.1. Handshake Signals

Figure 2.4 illustrates the handshake signals for input and output of a 64-point FFT in the High-Performance architecture. The handshake signals are the same for the High-Performance Radix-4 architecture.

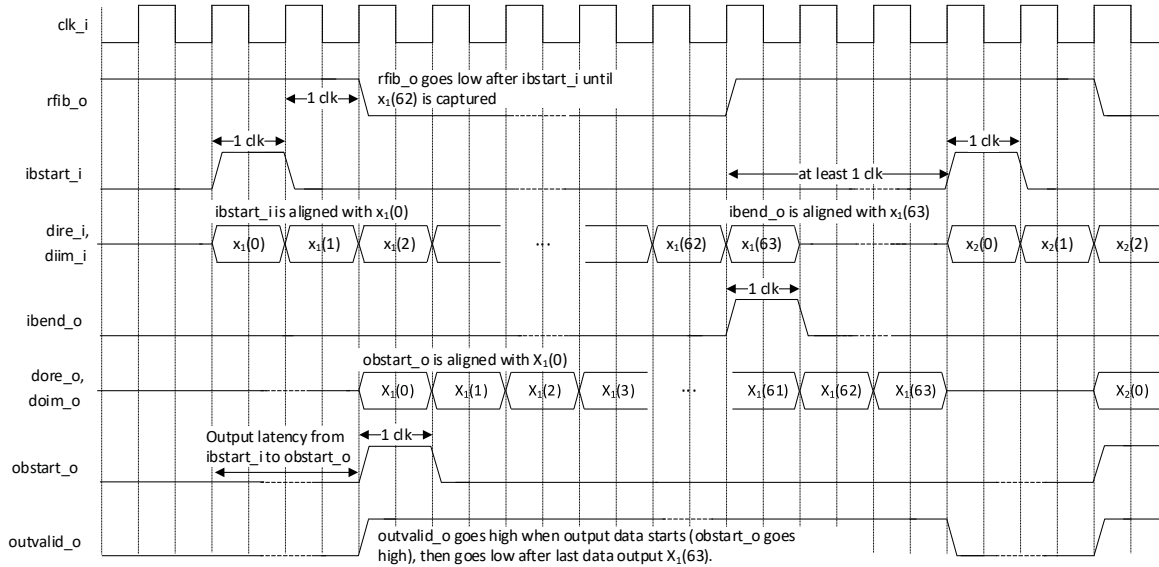


Figure 2.4. High-Performance (Streaming I/O) for 64 Points Timing Diagram

Figure 2.5 illustrates the handshake signals for input and output of a 64-point FFT in the Low Resource architecture.

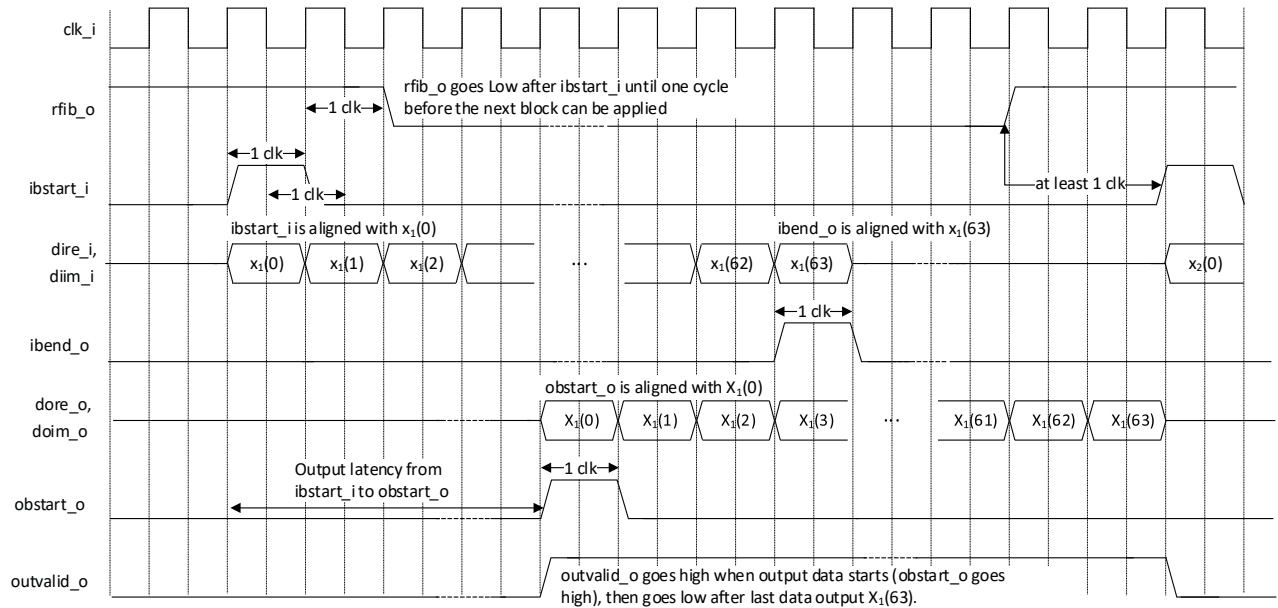


Figure 2.5. Low Resource (Burst I/O) for 64 Points Timing Diagram

The input *ibstart_i* is used to specify the start of a data block and must coincide with the 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 N input points are read. When the last input data point in a block is being read, 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 started. You can check *rfib_o* and start an input block in the next cycle after *rfib_o* goes high.

2.4.2. Configuration Signals

There are three configuration signals in the FFT compiler: *mode_i*, *sfact_i*, and *points_i*. You can set each one independently. These signals are stored when their corresponding set signals are active:

- *mode_i* is set when *modeset_i* is active.
- *sfact_i* is set when *sfactset_i* is active.
- *points_i* is set when *pointset_i* is active.

For these settings to be effective for a data block, the set signals must be applied after the previous data block has been processed, which means all input data is captured and all corresponding output data is generated. The set signals must be set at or before the start of that block (when *ibstart_i* goes active).

Exception: In the High-Performance implementation, set signals must be applied at least five cycles before *ibstart_i*.

Figure 2.6 illustrates the configuration signal timing for the High-Performance and High-Performance Radix-4 implementations, while Figure 2.7 illustrates the configuration signal timing for the Low Resource implementation.

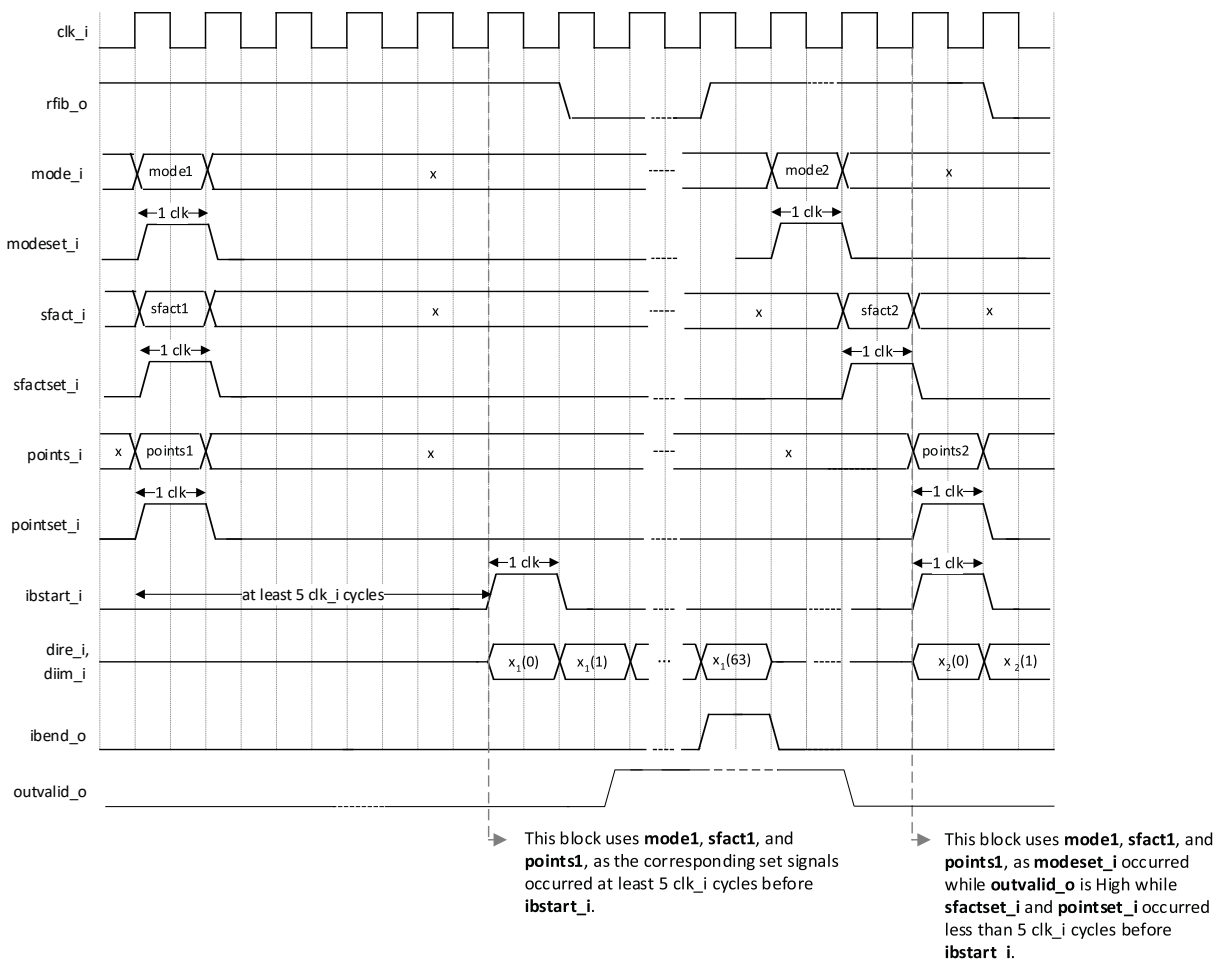


Figure 2.6. Handshake Signals for High-Performance and High-Performance Radix-4 FFT Timing Diagram



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. Refer to the [Output Scaling](#) section for more information.

Exceptions occur if there is an internal overflow during the computation of an output block. This is indicated by the *except_o* signal going high during a valid output. The *except_o* signal will go high if one or more overflows occur during the computation of a block. The severity and number of overflow exceptions in a block depend on the scaling scheme used and the property of the input data.

Figure 2.8 illustrates the timing of the *except_o* signal assertion.

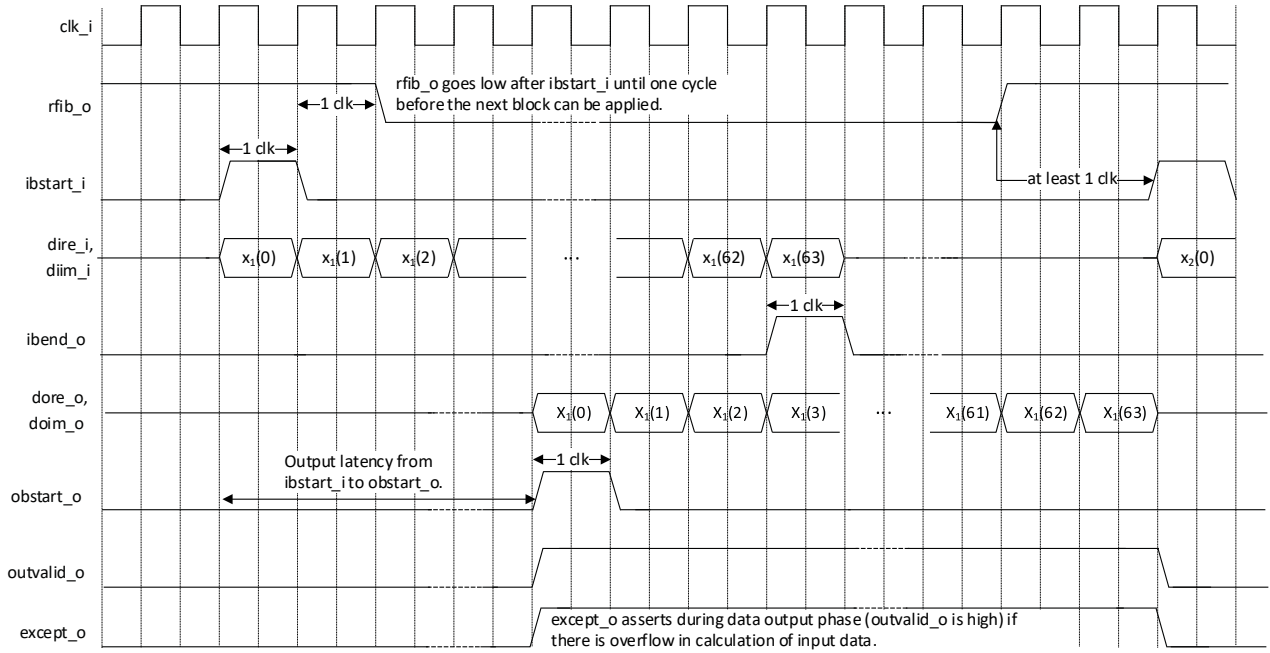


Figure 2.8. Exception Timing Diagram

2.5. Output Scaling

The equations below describe the true FFT and inverse FFT outputs, depending on the scaling mode, where N is the number of FFT points.

If the Scaling mode is set to *None*,

- the true FFT output is given by:

$$\text{true FFT} = \text{dore}_o + j(\text{doim}_o)$$

- the true inverse FFT is given by:

$$\text{true iFFT} = \frac{1}{N} \times ((\text{dore}_o) + j(\text{doim}_o))$$

If the Scaling mode is set to *RS111*,

- the true FFT output is given by:

$$\text{true FFT} = (\text{dore}_o \times 2^{\log_2 N}) + j(\text{doim}_o \times 2^{\log_2 N})$$

- the true inverse FFT is given by:

$$\text{true iFFT} = (\text{dore}_o) + j(\text{doim}_o)$$

If the Scaling mode is set to *RS211*,

- the true FFT output is given by:

$$\text{true FFT} = (\text{dore}_o \times 2^{(\log_2 N)+1}) + j(\text{doim}_o \times 2^{(\log_2 N)+1})$$

- the true inverse FFT is given by:

$$\text{true iFFT} = 2 \times ((\text{dore}_o) + j(\text{doim}_o))$$

If the Scaling mode is set to *Dynamic Through Port*,

- the scaling factor is given by:

$$dtp \text{ scaling factor} = sfact_i[0:1] + sfact_i[2:3] + \dots + sfact_i[MSB - 1:MSB]$$

Note: If *Points Variability == Fixed*, the overall scaling factor is the sum of the individual 2-bit scaling values for each stage, based on the number of FFT points.

- the true FFT output is given by:

$$true \text{ FFT} = (dore_o \times 2^{dtp \text{ scaling factor}}) + j(doim_o \times 2^{dtp \text{ scaling factor}})$$

- the true inverse FFT is given by:

$$true \text{ iFFT} = \frac{1}{N} \times ((dore_o \times 2^{dtp \text{ scaling factor}}) + j(doim_o \times 2^{dtp \text{ scaling factor}}))$$

If the Scaling mode is set to *Block Floating Point*,

- the true FFT output is given by:

$$true \text{ FFT} = (dore_o \times 2^{exponent_o}) + j(doim_o \times 2^{exponent_o})$$

- the true inverse FFT is given by:

$$true \text{ iFFT} = \frac{1}{N} \times ((dore_o \times 2^{exponent_o}) + j(doim_o \times 2^{exponent_o}))$$

2.6. Output Latency

Table 2.1 provides the latency through the IP Core as a function of FFT point size and implementation mode when bit-reversed output order is selected. The output latency is measured as the number of *clk_i* cycles from the assertion of *ibstart_i* to the start of output at *dore_o* and *doim_o*.

Table 2.1. Output Latency for Bit-Reversed Output Order based on FFT Point Size

FFT Point Size	Low Resource Mode ¹ (Number of <i>clk_i</i> cycles)	High-Performance Mode ¹ (Number of <i>clk_i</i> cycles)	High-Performance Radix-4 Mode (Number of <i>clk_i</i> cycles)
64	275	83	112
128	595	152	200
256	1299	282	360
512	2835	543	688
1024	6163	1057	1330
2048	13331	2086	2626
4096	28691	4136	5188
8192	61459	8237	10316
16384	131091	16431	20558

Note:

- Output latency is measured with *Precision Reduction Method = Truncation* and *Scaling = None*. When using *Precision Reduction Method = Rounding* or other scaling modes, the latency may vary and can increase by:
 - Up to 10 clock cycles in the Low Resource mode.
 - Up to $\log_2(\text{FFT point size})$ clock cycles in the High-Performance and High-Performance Radix-4 modes.

Table 2.2 provides the latency through the IP Core as a function of FFT point size and implementation mode when natural output order is selected. The output latency is measured as the number of *clk_i* cycles from assertion of *ibstart_i* to start of output at *dore_o* and *doim_o*.

Table 2.2. Output Latency for Natural Output Order based on FFT Point Size

FFT Point Size	Low Resource Mode ¹ (Number of <i>clk_i</i> cycles)	High-Performance Mode ¹ (Number of <i>clk_i</i> cycles)	High-Performance Radix-4 Mode (Number of <i>clk_i</i> cycles)
64	275	145	112
128	595	278	200
256	1299	536	360
512	2835	1053	688
1024	6163	2079	1330
2048	13331	4132	2626
4096	28691	8230	5188
8192	61459	16427	10316
16384	131091	32813	20558

Note:

- Output latency is measured with *Precision Reduction Method = Truncation* and *Scaling = None*. When using *Precision Reduction Method = Rounding* or other scaling modes, the latency may vary and can increase by:
 - Up to 10 clock cycles in the Low Resource mode.
 - Up to $\log_2(\text{FFT point size})$ clock cycles in the High-Performance and High-Performance Radix-4 modes.

3. IP Parameter Description

The configurable attributes of the FFT Compiler IP are shown in the following tables. You can configure the IP by setting the attributes accordingly in the IP Catalog's Module/IP wizard of the Lattice Radiant software.

Wherever applicable, default values are in **bold**.

3.1. Points/Mode Attributes

Table 3.1. Points/Mode Attributes

Points/Mode Attributes		
Attribute	Selectable Values	Description
Number of Points		
Points Variability	Fixed , Variable	Allows you to specify fixed or variable number of points. If Architecture == High-Performance Radix-4, Points Variability is always fixed.
Number of Points	64 , 128, 256, 512, 1024, 2048, 4096, 8192, 16384	Specifies the number of FFT points if Points variability == Fixed.
Maximum Points	128 , 256, 512, 1024, 2048, 4096, 8192, 16384	Denotes the minimum for the points range if Points variability == Variable.
Minimum Points	64 , 128, 256, 512, 1024, 2048, 4096, 8192	Denotes the maximum for the points range if Points variability == Variable.
Architecture		
Architecture	High-Performance, High-Performance Radix-4, Low Resource	This option selects the FFT architecture from the following: <ul style="list-style-type: none"> High Performance: streaming I/O High-Performance Radix-4: streaming I/O, reduced latency. Low Resource: Burst I/O
FFT Mode		
FFT Mode	Forward , Inverse, Dynamic Through Port	This parameter configures the operating mode of the core.
Output Order		
Output order	Bit-reversed , Natural	This parameter specifies whether the output data is bit-reversed or in natural order. Each is described as: <ul style="list-style-type: none"> Natural: Output is directly fed to the following stage. Bit-reversed: Output order is based on bit-reversed indices. This option results in reduced latency and/or lesser EBR usage. In applications where a FIFO or a buffer is used between the output of the FFT and the input to the next processing block, as in multi-clock systems, bit reversal can be done in that glue memory. In the Low Resource architecture, this applies separately to the upper and lower halves of the output. For an N-point FFT, the first N/2 points are output in bit-reversed order of their indices, followed by the second N/2 points in bit-reversed order of their indices.

3.2. Scaling/Width Attributes

Table 3.2. Scaling/Width Attributes

Scaling/Width Attributes		
Attribute	Selectable Values	Description
Scaling Mode		
Scaling Mode	None, RS111 , RS211, Dynamic Through Port, Block Floating Point	<p>This parameter defines whether the data is scaled after each radix-2 butterfly, and if so, specifies the type of scaling used. Each is described as follows:</p> <ul style="list-style-type: none"> None: There is no scaling at the output of butterflies. RS111: Results in a fixed scaling of <i>right shift by 1</i> in all FFT stages. This option is available only if <i>Architecture</i> == Low Resource or High-Performance. RS211: Results in a fixed scaling of <i>right shift by 2</i> in the first stage and <i>right shift by 1</i> in the subsequent stages. This option is available only if <i>Architecture</i> == Low Resource or High-Performance. Dynamic Through Port: The scale factors for the FFT stages are read dynamically from the input port <i>sfact_i</i> for every data block. This option is available only if <i>Architecture</i> == Low Resource. Block Floating Point: 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 <i>exponent_o</i> is added to the FFT compiler core when this option is selected. This option is available only if <i>Architecture</i> == Low Resource or High-Performance Radix-4.
Data Width		
Input Data Width	8 to 24, 16	Specifies input data width of either of the components: real or imaginary.
Output Data Width	8 to 38, 16	Specifies output data width of either of the components: real or imaginary. When <i>Scaling Mode</i> == Dynamic Through Port or <i>Scaling Mode</i> == None, <i>Output Data Width</i> will be <i>Input Data Width</i> + log ₂ [Points]; otherwise, <i>Output Data Width</i> will be <i>Input Data Width</i> .
Twiddle Factor Width	8 to 24, 16	Specifies twiddle factor width of either of the components: real or imaginary.
Precision Reduction Method		
Precision Reduction	Truncation , Rounding	<p>Selects the scaling process to be applied after every stage:</p> <ul style="list-style-type: none"> Truncation: Truncating the data (discarding the last one or two bits). This results in less logic utilization. Rounding: Rounding the data to the nearest number in the scaled precision (discarding one or two bits and making corrections to the output based on discarded bits). This improves the accuracy of the results.
Truncate Last Stage	0, 1	<p>This can be enabled to have better throughput. This option is available only if <i>Architecture</i> = <i>High-Performance</i>. Additionally, it can be modified only if the following are set:</p> <ul style="list-style-type: none"> <i>Scaling Mode</i> = <i>RS111</i> or <i>RS211</i> <i>Precision Reduction Method</i> = <i>Rounding</i> <p>Otherwise, it is displayed for information only.</p>

3.3. Implementation Attributes

Table 3.3. Implementation Attributes

Implementation Attributes		
Attribute	Selectable Values	Description
Multiplier Type		
Multiplier Type	DSP Block Based, LUT Based	This option specifies whether DSP blocks or LUTs are used for implementing multipliers and multiply-add components.
Pipeline		
Multiplier Pipeline	2, 3, 4	This option is used to specify an additional pipeline after the adders. This can be set to have better performance at the cost of slightly increased utilization and latency. This option is only active if <i>Architecture = Low Resource</i> and <i>Multiplier Type = LUT Based</i> . Otherwise, it is displayed for information only.
Adder Pipeline	0, 1	This option is used to specify the pipeline of LUT based multipliers. Higher values for adder pipeline lead to better performance at the cost of slightly increased utilization and latency. This option is only active if <i>Architecture = Low Resource</i> . Otherwise, it is displayed for information only.
Memory Type		
Memory Type	EBR Memory, Distributed Memory, Automatic	<p>This parameter specifies the balance between using EBR and distributed memories.</p> <ul style="list-style-type: none">EBR Memory: EBRs are used for memory depths of 32 and higher.Distributed Memory: EBR memories are used only for depths of 512 or more, while the rest uses distributed memories.Automatic: The IP generator automatically selects between EBR and distributed memories based on the IP parameters.

3.4. IP Parameter Settings for Example Use Cases

Table 3.4 FFT Compiler IP Example Use Cases

Attributes	Streaming I/O	Burst I/O	Inverse FFT	Dynamic FFT
Number of Points				
Points Variability	Fixed	Fixed	Fixed	Variable
Number of Points	1024	1024	1024	4096
Maximum Points	—	—	—	4096
Minimum Points	—	—	—	64
Architecture				
Architecture	High-Performance	Low Resource	High-Performance	High-Performance
FFT Mode				
FFT Mode	Forward	Forward	Inverse	Dynamic Through Port
Output Order				
Output order	Natural	Natural	Natural	Natural
Scaling Mode				
Scaling Mode	None	None	None	Dynamic Through Port
Data Width				
Input Data Width	16	16	16	16
Output Data Width	—	—	—	—
Twiddle Factor Width	24	16	16	16
Precision Reduction Method				
Precision Reduction	Rounding	Truncation	Truncation	Truncation
Truncate Last Stage	1	—	—	—
Multiplier Type				
Multiplier Type	DSP Block Based	LUT Based	DSP Block Based	DSP Block Based
Pipeline				
Multiplier Pipeline	—	2	—	—
Adder Pipeline	—	0	—	—
Memory Type				
Memory Type	Distributed Memory	Automatic	Distributed Memory	Distributed Memory

4. Signal Description

This section describes the FFT Compiler IP ports.

Table 4.1. Top level I/O interface

Port	Direction	Width	Output Reset Value	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	<i>Input Data Width</i>	—	Real part of the input data.
diim_i	Input	<i>Input Data Width</i>	—	Imaginary part of the input data.
dore_o	Output	<i>Output Data Width</i>	0	Real part of the output data.
doim_o	Output	<i>Output Data Width</i>	0	Imaginary part of the output data.
exponent_o	Output	See Table Note 1.	0	This value denotes the effective scaling that is done during block floating scaling. If <i>Points Variability</i> == <i>Variable</i> , then N = Maximum Points, else N = Number of Points.
Configuration Signals²				
mode_i	Input	1	—	When asserted, the core performs inverse FFT, else core performs forward FFT. Available only when <i>FFT Mode</i> == Dynamic Through Port.
modeset_i	Input	1	—	Sets the FFT mode signal. When this signal goes high, the value at <i>mode_i</i> port is read and the FFT mode (forward or inverse FFT) is set. Available only when <i>FFT Mode</i> == Dynamic Through Port.
sfact_i	Input	$2 \times \log_2 N$	—	Stage-wise scaling factors for N-point FFT with $\log_2 N$ stages. This signal is a concatenation of 2-bit scaling factors for each FFT stage. The most significant 2 bits correspond to stage 1 scale factor, the next 2 bits to stage 2 scale factor, up to the last FFT stage. If the number of points is not a power of 4, the last stage has only 1-bit scaling factor. Each 2-bit scaling factor denotes the number of right shifts performed to the output data of that stage. Available only when <i>Scaling Mode</i> == Dynamic Through Port.
sfactset_i	Input	1	—	Sets scale factor signal. When this signal goes high, stage-wise scaling factors are set with the values at the <i>sfact_i</i> input port. Available only when <i>Scaling Mode</i> == Dynamic Through Port.
points_i	Input	3 if <i>Maximum Points</i> == 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 \log_2 of the number of points represented in unsigned binary form. The valid range of values is from 6 to 14. Any value less than $\log_2(\text{Minimum Points})$ is read as $\log_2(\text{Minimum Points})$, and any value greater than $\log_2(\text{Maximum Points})$ is read as $\log_2(\text{Maximum Points})$. Available only when <i>Points Variability</i> == <i>Variable</i> .
pointset_i	Input	1	—	Sets the number of FFT points. When this input signal goes high, the value at <i>points_i</i> port is read and the number of FFT points is set accordingly. Available only when <i>Points Variability</i> == <i>Variable</i> .

Port	Direction	Width	Output Reset Value	Description
Status and Handshake Signals²				
ibstart_i	Input	1	—	Input block start signal. Asserted high 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 <i>ibstart_i</i> signal during an input read cycle is ignored.
except_o	Output	1	0	Exception output signal. Indicates that an exception (overflow) has occurred during computation. This may result from using incorrect scaling factors. The exception always pertains to the data currently being output, not the data being processed. For more details, refer to the Exceptions section. Available only when at least one of the following settings is met: <ul style="list-style-type: none"> <i>Scaling Mode != Block Floating Point</i> <i>Architecture == High-Performance</i> <i>Architecture == High-Performance Radix-4</i> <i>Architecture == Low Resource and Scaling Mode != None</i>
rfib_o	Output	1	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 <i>ibstart_i</i> one cycle after <i>rfib_o</i> goes high. After <i>ibstart_i</i> goes high, the core pulls <i>rfib_o</i> low in the next clock cycle.
ibend_o	Output	1	0	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	0	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	0	Output data valid output signal. This signal indicates that the core is now giving out valid output data through <i>dore_o</i> and <i>doim_o</i> ports.

Notes:

1. If the architecture is High-Performance Radix-4, the width of *exponent_o* is 6. If the architecture is Low Resource, the width of *exponent_o* is 5 if FFT points are greater than 256; otherwise, the width is 4.
2. Refer to the [Interfacing with the FFT Compiler](#) section for more information on the usage of the configuration, handshake, and status signals.

5. Example Design

The FFT Compiler example design allows you to compile, simulate, and test the FFT Compiler IP on the Lattice Avant-E Evaluation Board.

5.1. Example Design Supported Configuration

Table 5.1. FFT Compiler IP Configuration Supported by the Example Design

Attributes	Streaming I/O	Burst I/O	Inverse FFT
Number of Points			
Points Variability	Fixed	Fixed	Fixed
Number of Points	1024	64	128
Maximum Points	—	—	—
Minimum Points	—	—	—
Architecture			
Architecture	High-Performance	Low Resource	High-Performance
FFT Mode			
FFT Mode	Forward	Forward	Inverse
Output order	Natural	Bit-reversed	Natural
Scaling Mode			
Scaling Mode	None	RS111	None
Data Width			
Input Data Width	16	16	16
Output Data Width	—	—	—
Twiddle Factor Width	16	16	16
Precision Reduction Method			
Precision Reduction	Rounding	Truncation	Truncation
Truncate Last Stage	1	—	—
Multiplier Type			
Multiplier Type	DSP Block Based	DSP Block Based	DSP Block Based
Pipeline			
Multiplier Pipeline	—	—	—
Adder Pipeline	—	0	—
Memory Type			
Memory Type	EBR Memory	EBR Memory	EBR Memory

5.2. Overview of the Example Design and Features

Key features of the example design include:

- Pre-generated FFT input data stored in memory.
- Transmission of input data and handshake signals to the DUT according to the target configuration.
- Configurable FFT Compiler IP for static FFT settings.

5.3. Example Design Components

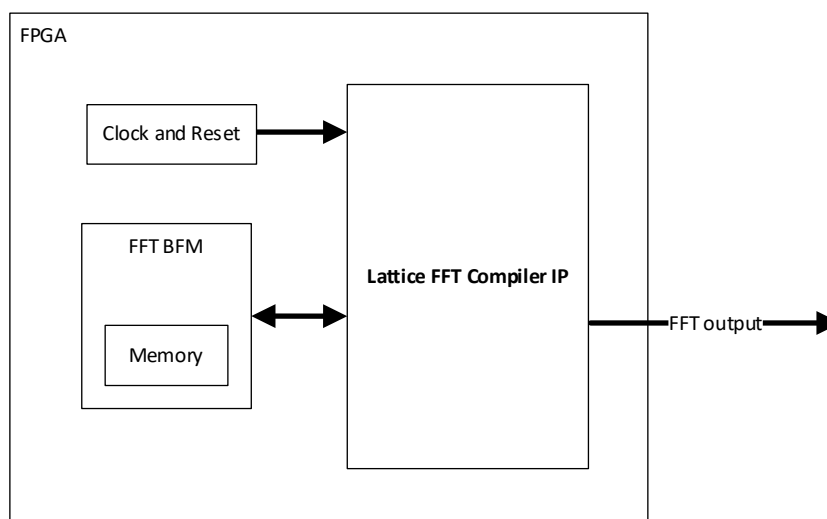


Figure 5.1. FFT Compiler Example Design Block Diagram

The FFT Compiler example design includes the following blocks:

- **FFT Compiler IP** – This is the DUT. It includes internal memory that stores pre-generated twiddle factors used during FFT computation.
- **FFT BFM** – This block sends the input data stored in memory to the DUT and handles the handshake signals for operation. For Forward FFT, the input is a time-domain single-tone sinusoidal signal with a randomly generated frequency. For Inverse FFT, the input is a frequency-domain signal with energy concentrated at a randomly selected target frequency.

5.4. Generating the Example Design

You can use the Lattice Radiant software to generate and use the example design. A sample Lattice Radiant software project file for the Lattice Avant devices is provided in the package. By using the sample project, you can run functional simulation, software implementation flow, and hardware testing.

Table 5.2 lists the files generated with the IP that are required for using the example design.

Table 5.2 Generated File List for Example Design

Attribute	Description
eval/source	Contains all the design modules needed for example design implementation including testbench files for functional simulation.
eval/sw/fft_ip	Pre-generated FFT Compiler Soft IP.
eval/sw/osc_ip	Pre-generated OSC Soft IP.
eval/sw/pll_ip	Pre-generated PLL Soft IP.
eval/sw/fft_compiler_ed.rdf	Sample Lattice Radiant software project in RDF format.
eval/sw/fft_compiler_ed.sty	Sample Lattice Radiant software project strategy file.
eval/fft_compiler_ed.pdc	Sample post-synthesis constraint file in PDC format for the example design. Pin location constraints are pre-generated for the Lattice Avant E Evaluation Board only.

5.4.1. Using the Example Design Sample Project

The sample project includes all the files required by the example design including the PDC file. To use the example design sample project, follow these steps:

1. Open the sample project provided: *eval/sw/fft_compiler_ed.rdf*.

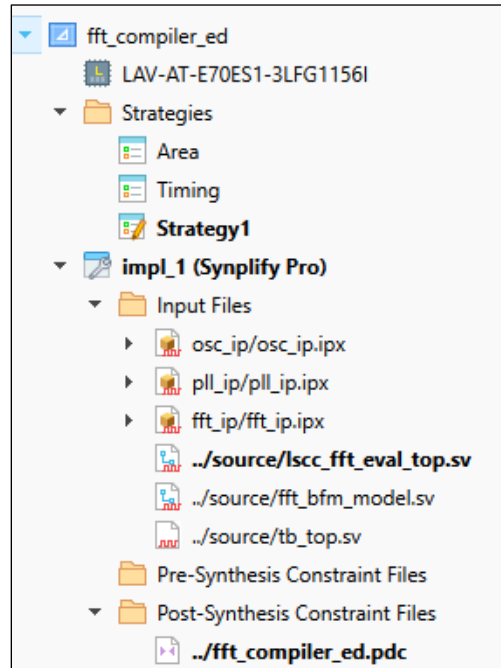


Figure 5.2. Example Design Project File List

2. If you want to enable Reveal debugging, right-click on **Debug Files** > **fft_compiler_ed.rvl** and select **Set as Active Debug File**. Refer to the [Reveal User Guide for Radiant Software](#) for details on how to use the Reveal Analyzer.

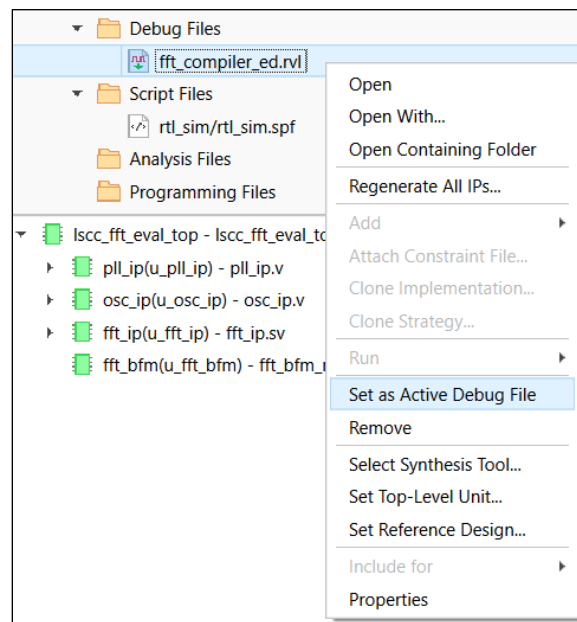



Figure 5.3. Adding Debug Files to Example Design

- Click  (Run All) located on the toolbar to perform the Lattice Radiant software full design compilation, which generates the example design bitstream file for the hardware test.

5.4.2. Changing Configuration of the Example Design

The sample project generated instantiates the Streaming I/O configuration in [Table 5.1](#). If you want to test a different FFT Compiler IP configuration, double-click on *fft_ip/fft_ip.ipx* and configure as intended.


5.4.3. Limitations of the Example Design

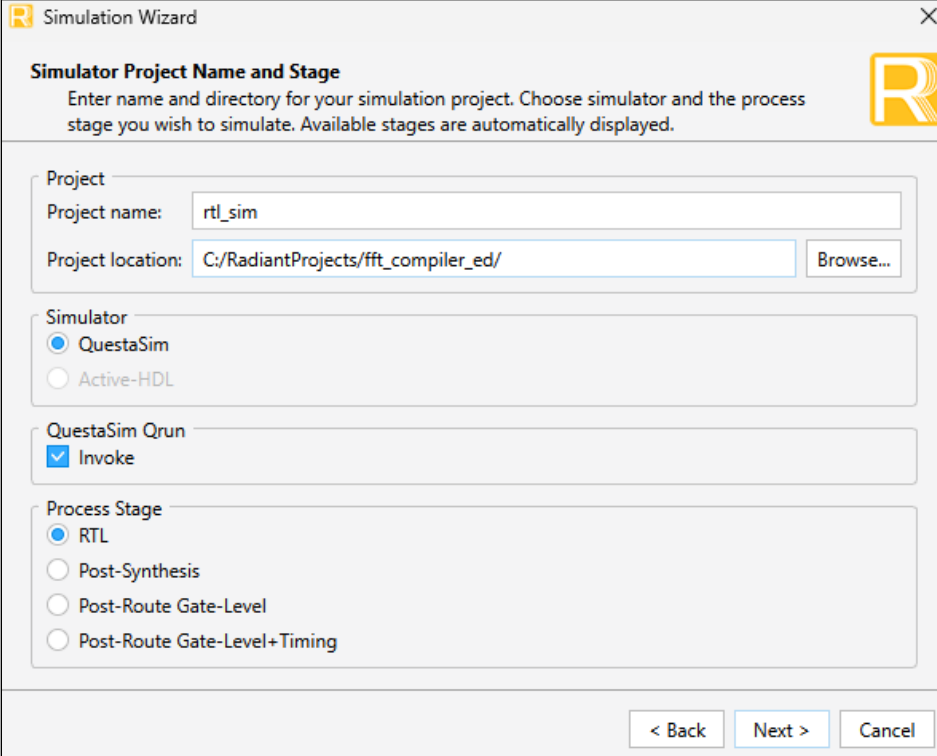
This example design is limited to testing static FFT configurations. The following features are not supported:

- Points Variability == Variable
- FFT Mode == Dynamic Through Port
- Scaling Mode == Dynamic Through Port

5.5. Simulating the Example Design

To run functional simulation:

- Click  located on the toolbar to initiate the **Simulation Wizard**, as shown in the figure below.



Simulation Wizard

Simulator Project Name and Stage
Enter name and directory for your simulation project. Choose simulator and the process stage you wish to simulate. Available stages are automatically displayed.

Project
Project name:
Project location:

Simulator
☒ QuestaSim
☐ Active-HDL

QuestaSim Qrun
☒ Invoke

Process Stage
☒ RTL
☐ Post-Synthesis
☐ Post-Route Gate-Level
☐ Post-Route Gate-Level+Timing

Figure 5.4. Simulation Wizard for Example Design

5.6. Hardware Testing

The bitstream generated in the [Generating the Example Design](#) section is programmed to the evaluation board using the Lattice Radiant Programmer software. When the design is successfully programmed, the D29 LED lights up. Then, press SW4 to start the test.

Output data and handshake signals are observed through Reveal Debugging.

Figure 5.7, Figure 5.8, and Figure 5.9 show sample output of the example design with the 1024-point FFT Compiler IP:

- Input *ibstart_i* aligns with the first input data.
- Output *rfib_o* goes Low after *ibstart_i* is asserted and returns High when *ibend_o* is asserted.
- Output *obstart_o* aligns with the first output data.
- Output *outvalid_o* remains High while output data is transmitted (for 1024 clock cycles).

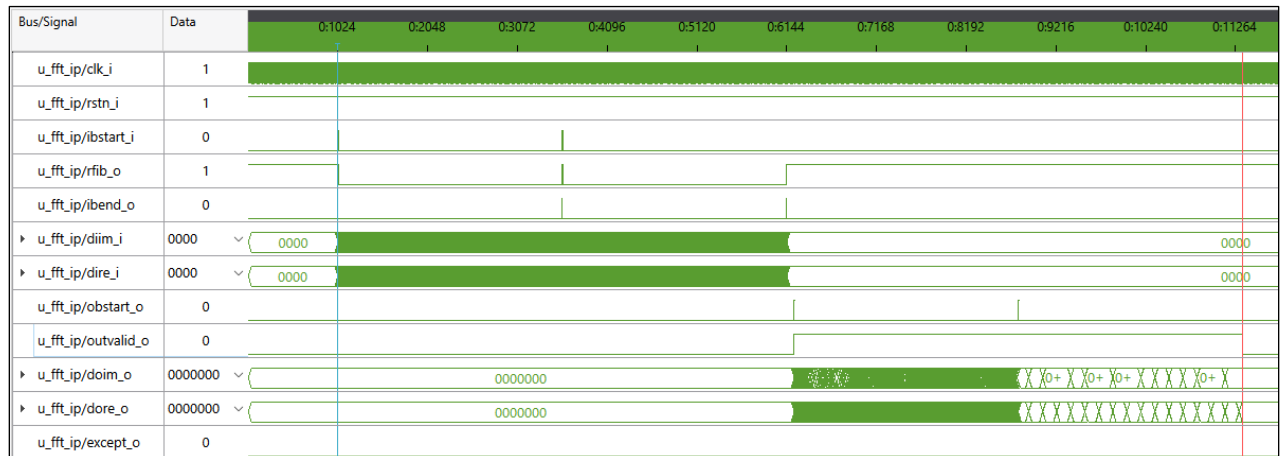


Figure 5.7. 1024-Point FFT Compiler IP Sample Output

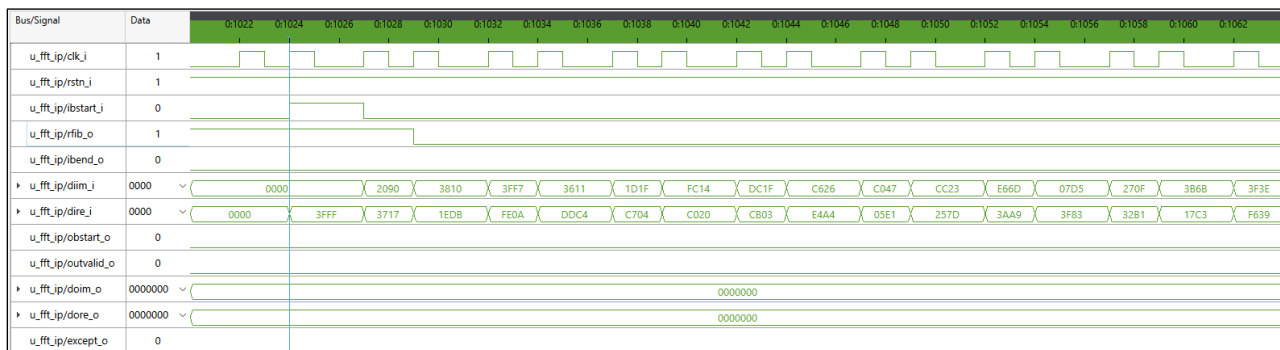


Figure 5.8. Input Data and Handshake Signals

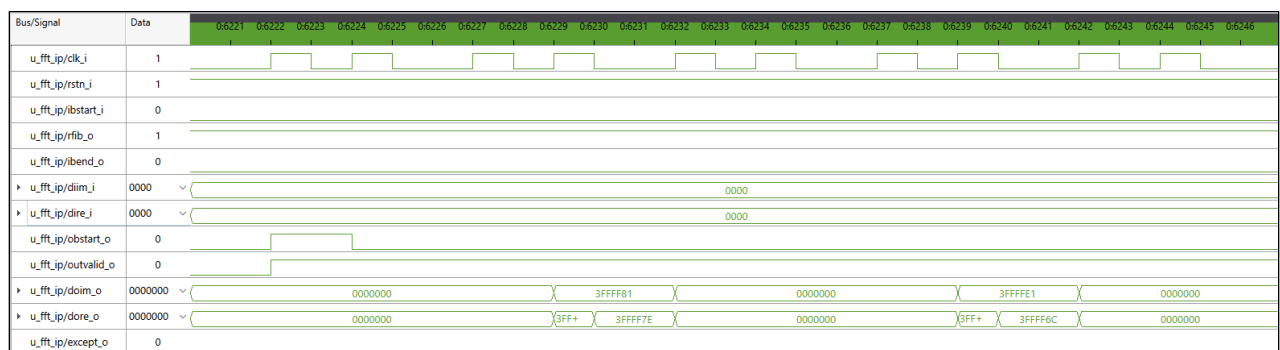


Figure 5.9. Output Data and Handshake Signals

6. Designing with the IP

This section provides information on how to generate the 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.

Note: The screenshots provided are for reference only. Details may vary depending on the version of the IP or software being used. If there have been no significant changes to the GUI, a screenshot may reflect an earlier version of the IP.

6.1. Generating and Instantiating the IP

You can use the Lattice Radiant software to generate IP modules and integrate them into the device architecture. The steps below describe how to generate the FFT Compiler IP in the Lattice Radiant software.

To generate the FFT Compiler IP:

1. Create a new Lattice Radiant software project or open an existing project.
2. In the **IP Catalog** tab, double-click **FFT Compiler** under **IP, DSP** category. The **Module/IP Block Wizard** opens as shown in [Figure 6.1](#). Enter values in the **Component name** and the **Create in** fields and click **Next**.

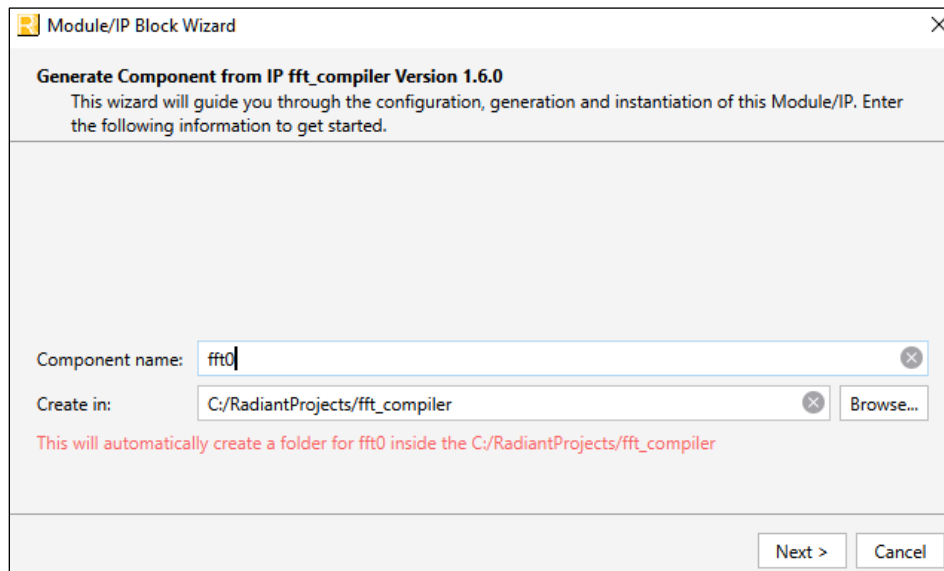


Figure 6.1. Module/IP Block Wizard

3. In the next **Module/IP Block Wizard** window, customize the selected FFT Compiler IP using drop-down lists and check boxes. [Figure 6.2](#) shows an example configuration of the FFT Compiler IP. For details on the configuration options, refer to the [IP Parameter Description](#) section.

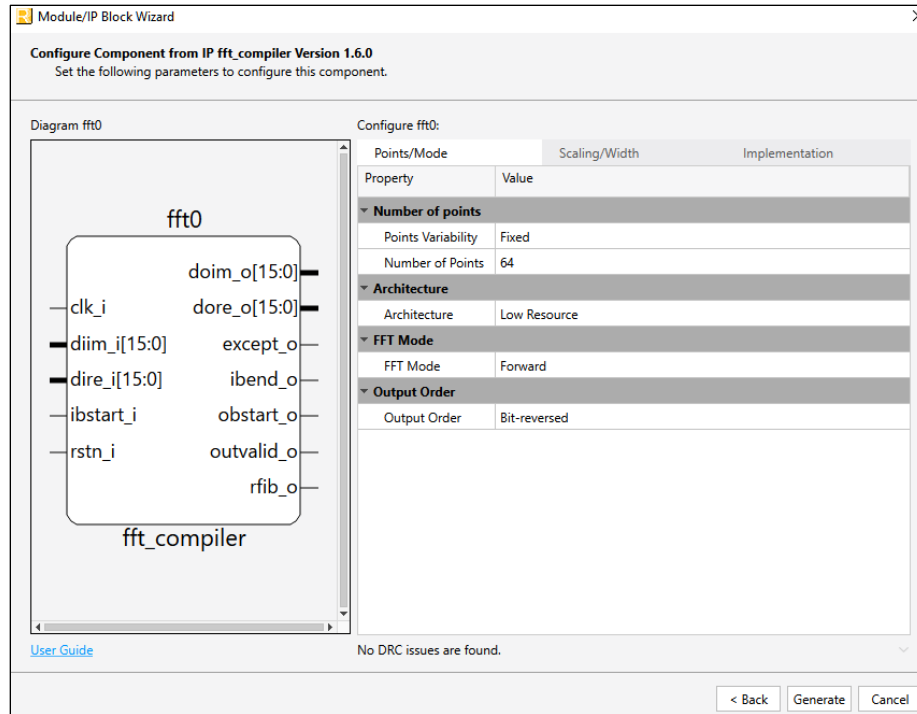


Figure 6.2. IP Configuration

- Click **Generate**. The **Check Generated Result** dialog box opens, showing design block messages and results as shown in Figure 6.3.

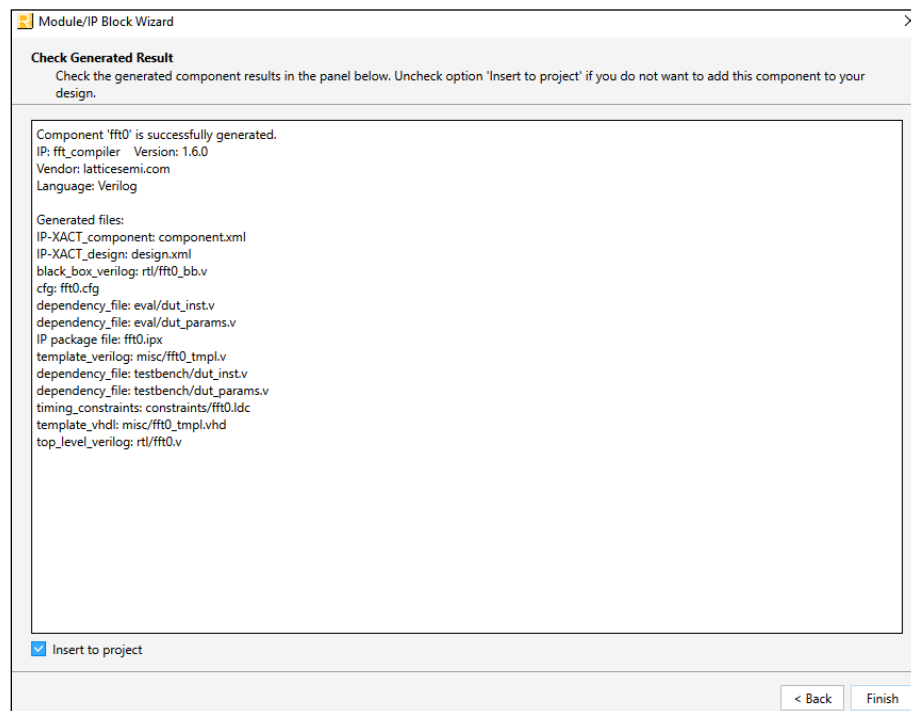


Figure 6.3. Check Generated Result

- Click **Finish**. All the generated files are placed under the directory paths in the **Create in** and the **Component name** fields shown in Figure 6.1.

6.1.1. Generated Files and File Structure

The generated FFT Compiler module package includes the closed-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 module 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 6.1](#).

Table 6.1. 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 module.
rtl/<Component name>_bb.v	This file provides the synthesis closed-box.
misc/<Component name>_tmpl.v misc /<Component name>_tmpl.vhd	These files provide instance templates for the module.
eval/constraint.pdc	This file provides information on how to constrain this IP in your design. Refer to the Timing Constraints section for guidance on how to use this file.

6.2. Design Implementation

Completing your design includes additional steps to specify analog properties, pin assignments, and timing and physical constraints. You can add and edit the constraints using the Device Constraint Editor or by manually creating a PDC File.

Post-Synthesis constraint files (.pdc) contain both timing and non-timing constraint.pdc source files for storing logical timing/physical constraints. Constraints that are added using the Device Constraint Editor are saved to the active .pdc file. The active post-synthesis design constraint file is then used as input for post-synthesis processes.

Refer to the relevant sections in the Lattice Radiant Software User Guide for more information on how to create or edit constraints and how to use the Device Constraint Editor.

6.3. Timing Constraints

You need to provide proper timing and physical design constraints to ensure the design meets the desired performance goals on the FPGA. Add the content of the following IP constraint file to your design constraints:

```
<IP_Instance_Path>/<IP_Instance_Name>/eval/constraint.pdc
```

The constraint file has been verified during IP evaluation with the IP instantiated directly in the top-level module. You can modify the constraints in this file with thorough understanding of the effect of each constraint.


To use this constraint file, copy the contents of the constraint.pdc to the top-level design constrain for post-synthesis.

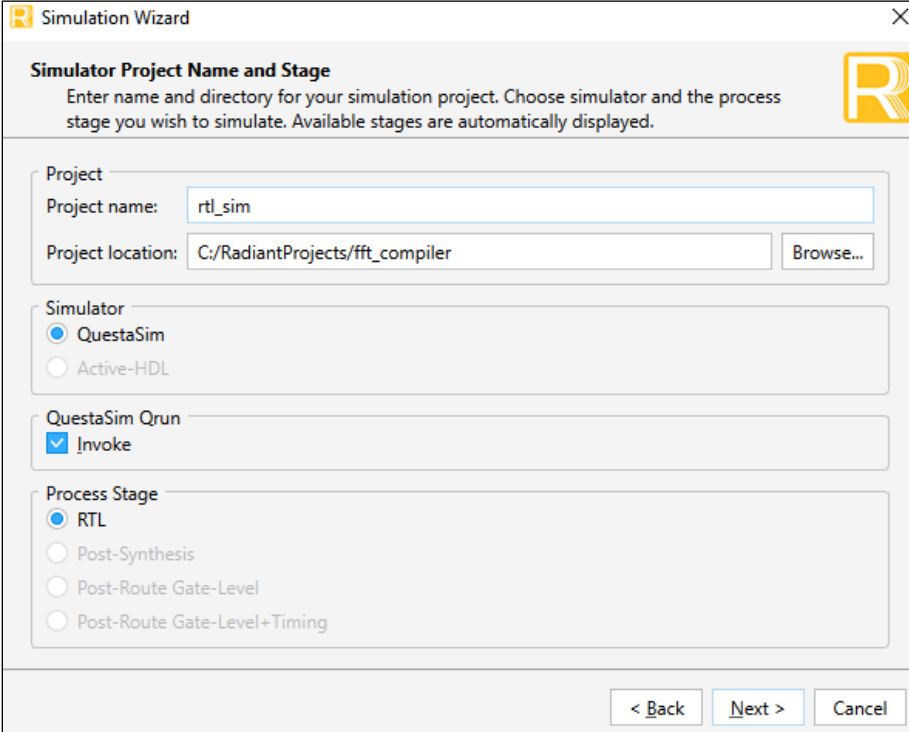
Refer to the [Lattice Radiant Timing Constraints Methodology \(FPGA-AN-02059\)](#) for details on how to constrain your design.

6.4. Running Functional Simulation

You can run functional simulation after the IP is generated.

To run functional simulation:

1. Click  located on the **Toolbar** to initiate the **Simulation Wizard** shown in [Figure 6.4](#).



Simulation Wizard

Simulator Project Name and Stage
Enter name and directory for your simulation project. Choose simulator and the process stage you wish to simulate. Available stages are automatically displayed.

Project
Project name:
Project location:

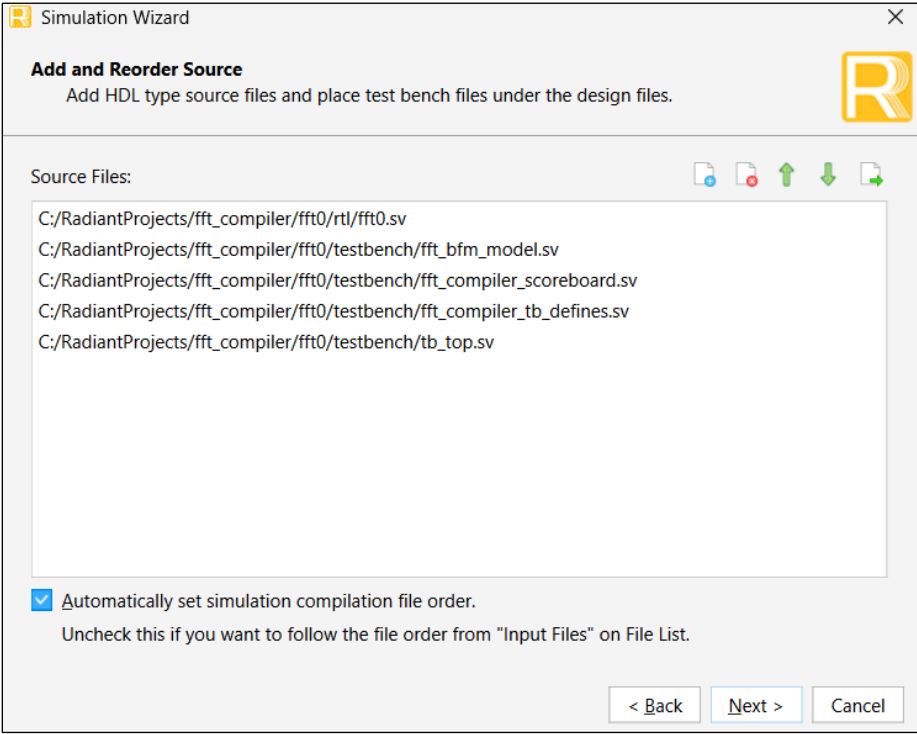
Simulator
☒ QuestaSim
☐ Active-HDL

QuestaSim Qrun
☒ Invoke

Process Stage
☒ RTL
☐ Post-Synthesis
☐ Post-Route Gate-Level
☐ Post-Route Gate-Level+Timing

Figure 6.4. Simulation Wizard

2. Click **Next** to open the **Add and Reorder Source** window as shown in [Figure 6.5](#).



Simulation Wizard

Add and Reorder Source
Add HDL type source files and place test bench files under the design files.

Source Files:

- C:/RadiantProjects/fft_compiler/fft0/rtl/fft0.sv
- C:/RadiantProjects/fft_compiler/fft0/testbench/fft_bfm_model.sv
- C:/RadiantProjects/fft_compiler/fft0/testbench/fft_compiler_scoreboard.sv
- C:/RadiantProjects/fft_compiler/fft0/testbench/fft_compiler_tb_defines.sv
- C:/RadiantProjects/fft_compiler/fft0/testbench/tb_top.sv

☒ Automatically set simulation compilation file order.
Uncheck this if you want to follow the file order from "Input Files" on File List.

Figure 6.5. Add and Reorder Source

3. Click **Next**. The **Summary** window is shown.
 4. Click **Finish** to run the simulation.
- The waveform in [Figure 6.6](#) shows an example simulation result.

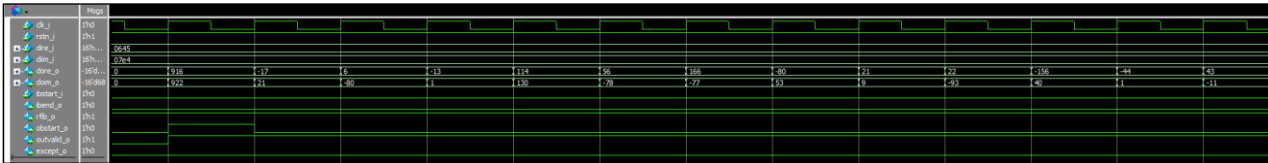


Figure 6.6. Simulation Waveform

6.4.1. Simulation Results

[Figure 6.7](#) shows the block diagram of the generated testbench for simulating the IP. The testbench has the following components:

- **FFT BFM** – This module transmits pre-generated input data along with the appropriate handshake signals. For Forward FFT, the input is a time-domain single-tone sinusoidal signal with a randomly generated frequency. For Inverse FFT, the input is a frequency-domain signal with energy concentrated at a randomly selected target frequency.
- **DUT** – This is the FFT Compiler IP core. It includes internal memory that stores pre-generated twiddle factors used during FFT computation.
- **FFT Scoreboard** – This module evaluates the DUT outputs *dore_o* and *doim_o* by comparing them against the expected results generated using the Python *numpy.fft* library. For Forward FFT, the expected output is a frequency-domain signal with a peak at the input signal frequency. For Inverse FFT, the expected output is a time-domain sinusoidal waveform whose frequency matches the input frequency component.

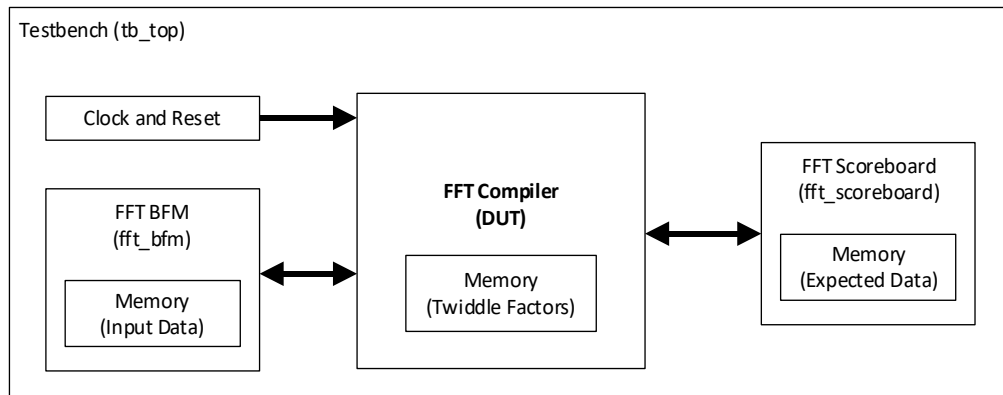


Figure 6.7. FFT Compiler Simulation Testbench Block Diagram

[Figure 6.8](#) illustrates the time-domain sinusoidal input data for a 1024-point forward FFT. The signal *ibstart_i* is asserted for one clock cycle, during which the first data points of *dire_i* and *diim_i* are transmitted simultaneously with the assertion of *ibstart_i*. The signal *ibend_o* is asserted after all 1024 data points have been input.

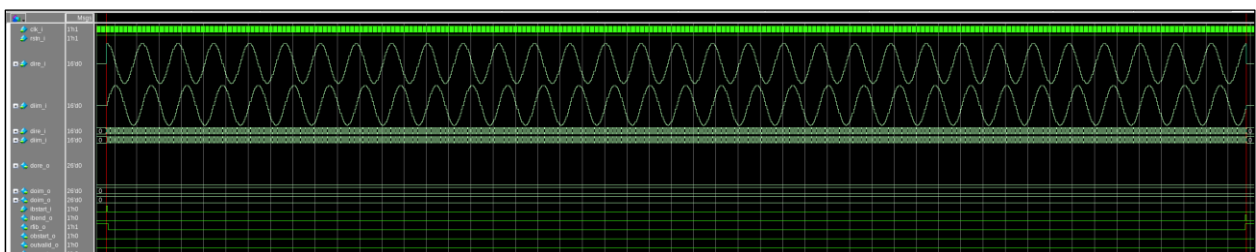


Figure 6.8. Simulation Waveform for Input Data Block (Forward FFT)

Figure 6.9 shows the frequency-domain output data of a 1024-point forward FFT, with a peak at the corresponding frequency from the input. The assertion of *obstart_o* indicates the beginning of the output data transmission. The first data points of *dore_o* and *doim_o* are output simultaneously with the assertion of *obstart_o*. The signal *outvalid_o* remains high while the 1024 output data points are being transmitted.

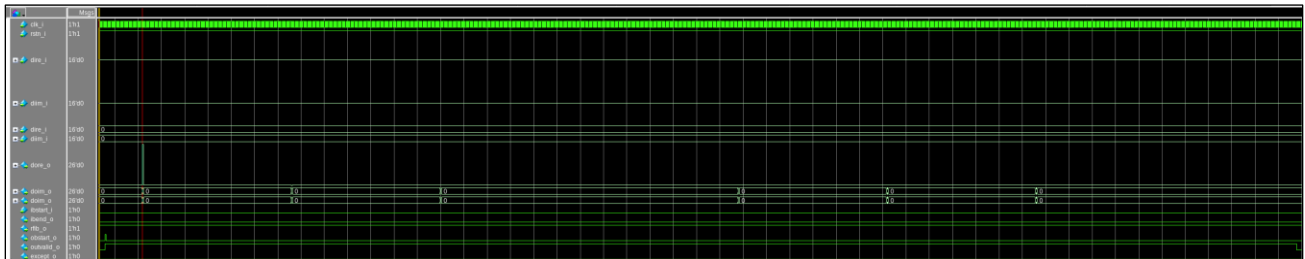


Figure 6.9. Simulation Waveform for Output Data Block (Forward FFT)

Figure 6.10 illustrates the frequency-domain input used for a 1024-point inverse FFT, with peaks at frequencies corresponding to the desired sinusoidal output.

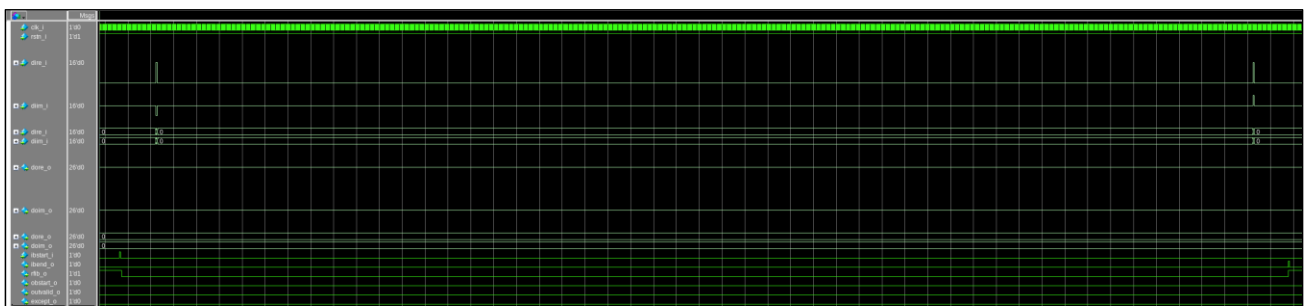


Figure 6.10. Simulation Waveform for Input Data Block (Inverse FFT)

Figure 6.11 presents the resulting time-domain sinusoidal waveform generated from the 1024-point inverse FFT.

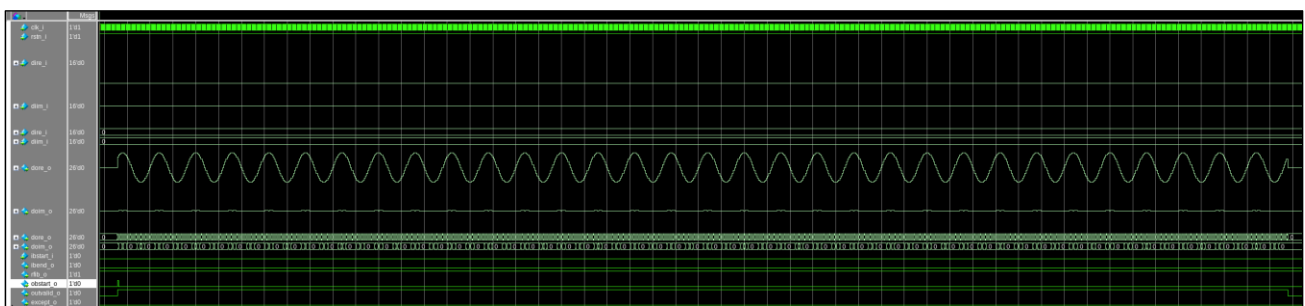


Figure 6.11. Simulation Waveform for Output Data Block (Inverse FFT)

Appendix A. Resource Utilization

This section shows the resource utilization of the FFT Compiler IP core for different devices. Default configuration is used, and some attributes are changed from the default value to show the effect on the resource utilization.

The default configurations are set as follows:

- Points Variability = Fixed
- Number of Points = 64
- Architecture = Low Resource
- FFT Mode = Forward
- Output Order = Bit-reversed
- Scaling Mode = RS111
- Input Data Width = 16
- Twiddle Factor Width = 16
- Precision Reduction Method = Truncation
- Multiplier Type = DSP Block Based
- Adder Pipeline = 0
- Memory Type = EBR Memory

Table A.1 shows the resource utilization of the FFT Compiler IP Core for the LAV-AT-E70-1LFG1156C device using Synplify Pro of the Lattice Radiant software 2025.1.

Table A.1. LAV-AT-E70-1LFG1156C Device Resource Utilization

Configuration	Clk Fmax (MHz) ^{1,2}	Registers	LUTs	EBRs	DSPs
Default	310.559	826	752	3	4
Architecture: High-Performance, Others = Default	319.693	1139	1585	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	150.444	1295	1541	1	24
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	129.870	1813	4072	1	0
FFT Mode: Dynamic Through Port, Others = Default	250.000	793	705	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	157.431	1570	1228	3	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	207.598	1522	2225	0	0

Notes:

1. Fmax is generated when the FPGA design contains only the FFT Compiler IP and the target frequency is 100 MHz. These values may be reduced when user logic is added to the FPGA design.
2. The clock port is configured with the LVDS I/O type. Performance grade: 1.

Table A.2 shows the resource utilization of the FFT Compiler IP Core for the LIFCL-33-8USG84C device using Synplify Pro of the Lattice Radiant software 2025.1.

Table A.2. LIFCL-33-8USG84C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs (Multipliers)
Default	185.632	824	752	3	4
Architecture: High-Performance, Others = Default	200.000	1122	1608	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	172.861	1354	1685	1	16
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	132.310	2770	4482	1	0
FFT Mode: Dynamic Through Port, Others = Default	194.326	829	753	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	162.232	1436	1123	6	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	199.561	1626	2442	0	0

Note:

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

Table A.3 shows the resource utilization of the FFT Compiler IP Core for the LFD2NX-9-7MG121C device using Synplify Pro of the Lattice Radiant software 2024.2.

Table A.3. LFD2NX-9-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs
Default	194.212	824	752	3	4
Architecture: High-Performance, Others = Default	200	1128	1620	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	Configuration does not fit	Configuration does not fit	Configuration does not fit	Configuration does not fit	Configuration does not fit
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	120.948	2242	4445	1	0
FFT Mode: Dynamic Through Port, Others = Default	187.161	829	753	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	Configuration does not fit	Configuration does not fit	Configuration does not fit	Configuration does not fit	Configuration does not fit
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	165.508	1616	2435	0	0

Note:

1. Fmax is generated when the FPGA design contains only the FFT Compiler IP and the target Frequency is 100 MHz. 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 LFD2NX-17-7MG121C device using Synplify Pro of the Lattice Radiant software 2024.2.

Table A.4. LFD2NX-17-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs
Default	194.212	824	752	3	4
Architecture: High-Performance, Others = Default	200	1128	1620	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	171.233	1360	1705	1	16
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	120.948	2242	4445	1	0
FFT Mode: Dynamic Through Port, Others = Default	187.161	829	753	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	160.798	1436	1123	6	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	165.508	1616	2435	0	0

Note:

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

Table A.5 shows the resource utilization of the FFT Compiler IP Core for the LFD2NX-28-7MG121C device using Synplify Pro of the Lattice Radiant software 2024.2.

Table A.5. LFD2NX-28-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs
Default	187.126	824	752	3	4
Architecture: High-Performance, Others = Default	200	1128	1620	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	171.233	1360	1705	1	16
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	120.005	2242	4445	1	0
FFT Mode: Dynamic Through Port, Others = Default	180.636	829	753	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	160.798	1436	1123	6	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	167.056	1616	2435	0	0

Note:

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

Table A.6 shows the resource utilization of the FFT Compiler IP Core for the LFD2NX-40-7MG121C device using Synplify Pro of the Lattice Radiant software 2024.2.

Table A.6. LFD2NX-40-7MG121C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs
Default	187.126	824	752	3	4
Architecture: High-Performance, Others = Default	200	1128	1620	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	171.233	1360	1705	1	16
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	120.005	2242	4445	1	0
FFT Mode: Dynamic Through Port, Others = Default	180.636	829	753	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	160.798	1436	1123	6	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	167.056	1616	2435	0	0

Note:

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

Table A.7 shows the resource utilization of the FFT Compiler IP Core for the LN2-CT-20-1CBG484C device using Synplify Pro of the Lattice Radiant software 2024.2.

Table A.7. LN2-CT-20-1CBG484C Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs
Default	156.691	788	640	3	4
Architecture: High-Performance, Others = Default	250.000	1145	1533	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	162.470	1301	1477	1	24
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	138.985	1794	4047	1	0
FFT Mode: Dynamic Through Port, Others = Default	201.857	793	641	3	4
Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	162.575	1570	1132	3	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	157.406	1522	2163	0	0

Note:

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

Table A.8 shows the resource utilization of the FFT Compiler IP Core for the LFCPNX-100-8LFG672I device using Synplify Pro of the Lattice Radiant software 2025.1.

Table A.8. LFCPNX-100-8LFG672I Device Resource Utilization

Configuration	Clk Fmax (MHz) ¹	Registers	LUTs	EBRs	DSPs
Default	218.914	826	752	3	4
Architecture: High-Performance, Others = Default	257.400	1124	1608	1	8
Architecture: High-Performance, Scaling Mode: None Others = Default	194.137	1356	1685	1	16
Architecture: High-Performance, Multiplier Type: LUT-based, Others = Default	128.700	3232	4313	1	0
FFT Mode: Dynamic Through Port, Others = Default	194.137	1211	1094	5	8
Number of Points: 1024 Input Data Width: 24, Twiddle Factor Width: 24, Others = Default	182.249	1504	1235	6	16
Multiplier Type: LUT-based, Memory Type: Distributed Memory, Others = Default	132.802	2427	1453	0	0

Note:

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

References

- [FFT Compiler IP Release Notes \(FPGA-RN-02069\)](#)
- [Lattice Radiant Timing Constraints Methodology \(FPGA-AN-02059\)](#)
- [Reveal User Guide for Radiant Software](#)
- [FFT Compiler IP Core](#) web page
- [Avant-E](#) web page
- [Avant-G](#) web page
- [Avant-X](#) web page
- [Certus-N2](#) web page
- [Certus-NX](#) web page
- [CertusPro-NX](#) web page
- [CrossLink-NX](#) web page
- [MachXO5-NX](#) web page
- [Lattice Propel Design Environment](#) web page
- [Lattice Radiant Software](#) web page
- [Lattice Solutions IP Cores](#) web page
- [Lattice Solutions Reference Designs](#) web page
- [Lattice Insights](#) web page for Lattice Semiconductor training courses and learning plans

Technical Support Assistance

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

For frequently asked questions, refer to the Lattice Answer Database at www.latticesemi.com/Support/AnswerDatabase.

Revision History

Note: In some instances, the IP may be updated without changes to the user guide. The user guide may reflect an earlier IP version but remains fully compatible with the later IP version. Refer to the IP Release Notes for the latest updates.

Revision 1.8, IP v1.6.0, December 2025

Section	Change Summary
All	<ul style="list-style-type: none"> Updated the IP version information on the cover page. Added a note on the IP version in the <i>Quick Facts</i> and <i>Revision History</i> sections. Made editorial fixes. Removed the <i>Timing Specifications</i> section.
Abbreviations in This Document	<ul style="list-style-type: none"> Replaced <i>acronyms</i> with <i>abbreviations</i>. Added <i>bfly</i>, <i>BFM</i>, <i>FIFO</i>, <i>I/O</i>, <i>HDL</i>, <i>LVDS</i>, <i>mem</i>, <i>PE</i>, and <i>ROM</i>.
Introduction	<ul style="list-style-type: none"> Updated the description in the Overview of the IP section. Updated Table 1.1. Summary of the FFT Compiler IP and Table 1.2. FFT Compiler IP Support Readiness. Added the following sections: <ul style="list-style-type: none"> IP Support Summary Hardware Support Minimum Device Requirements Moved the previous <i>3.1. Licensing the IP</i>, <i>3.4. Hardware Evaluation</i>, and <i>4. Ordering Part Number</i> sections to the Licensing and Ordering Information section and updated its content. Updated the description in the Features section. Added <i>Naming</i> to the Naming Conventions section header. Added the Attribute Names section.
Functional Description	<ul style="list-style-type: none"> Added <i>IP Architecture</i> to the IP Architecture Overview section header and updated its description. Updated the description in the High-Performance Architecture section. Added the following sections: <ul style="list-style-type: none"> High-Performance Radix-4 Architecture Clocking Reset Output Scaling Moved the Handshake Signals section to precede the Configuration Signals section. Updated the following sections: <ul style="list-style-type: none"> Handshake Signals Configuration Signals Exponent Output Exceptions Output Latency Updated Figure 2.1. FFT Compiler Interface Diagram. Added the following figures: <ul style="list-style-type: none"> Figure 2.4. High-Performance (Streaming I/O) for 64 Points Figure 2.5. Low Resource (Burst I/O) for 64 Points Figure 2.7. Handshake Signals for Low Resource FFT Figure 2.6. Handshake Signals for High-Performance and High-Performance Radix-4 FFT Figure 2.8. Exception Timing Diagram
IP Parameter Description	Moved the content from the previous <i>2.3. Attribute Summary</i> section to this section and updated its content.
Signal Description	Moved the content from the previous <i>2.2. Signal Descriptions</i> section to this section and updated its content.
Example Design	Added this section.

Section	Change Summary
Designing with the IP	<ul style="list-style-type: none"> Moved the content from the previous 3. <i>IP Generation and Evaluation</i> section to this section and updated its content, including all figures. Added the Simulation Results section.
Resource Utilization	<ul style="list-style-type: none"> Updated this section and added resource utilizations for the Lattice Radiant software version 2025.1. Added the <i>LFCPNX-100-8LFG672I</i> device resource utilization.
References	Added <i>FFT Compiler IP Release Notes (FPGA-RN-02069)</i> and <i>Reveal User Guide for Radiant Software, and FFT Compiler IP Core, Lattice Propel Design Environment, and Lattice Solutions Reference Designs</i> web pages.

Revision 1.7, IP v1.5.0, December 2024

Section	Change Summary
All	Added the IP version information on the cover page.
Introduction	<p>Updated Table 1.1. Quick Facts:</p> <ul style="list-style-type: none"> Added the <i>Certus-N2</i> device family to <i>Supported FPGA Family</i>. Added <i>LFD2NX-9, LFD2NX-17, LFD2NX-28, LFD2NX-40, and LN2-CT-20</i> devices to <i>Targeted Devices</i>. Updated the <i>Resources</i> and <i>Lattice Implementation</i> information.
Functional Description	Updated the W_N formula (equation 2) in the Overview section.
Ordering Part Number	<ul style="list-style-type: none"> Updated instances of <i>Single Machine</i> to <i>Single Seat</i>. Added the <i>Certus-N2</i> OPNs.
Resource Utilization	<ul style="list-style-type: none"> Updated caption and <i>Note</i> for Table A.1. LIFCL-33-8USG84C Device Resource Utilization. Updated caption for Table A.2. LAV-AT-E70-3LFG1156I Device Resource Utilization. Added resource utilizations for the Lattice Radiant software version 2024.2.
References	Added the <i>Certus-N2</i> and <i>Lattice Solutions IP Cores</i> web pages, and the <i>Lattice Radiant Timing Constraints Methodology (FPGA-AN-02059)</i> document.

Revision 1.6, Lattice Radiant SW Version 2024.1, August 2024

Section	Change Summary
Functional Description	Updated the selectable values of Output Data Width attribute in Table 2.2. Attributes Table.

Revision 1.5, Lattice Radiant SW Version 2023.2, December 2023

Section	Change Summary
All	<ul style="list-style-type: none"> Renamed document from FFT Compiler IP Core - Lattice Radiant Software to FFT Compiler IP. Removed Appendix B. Limitations. Performed minor formatting and typo edits.
Disclaimers	Updated disclaimers.
Inclusive Language	Added inclusive language boilerplate.
Introduction	Changed LAV-AT-500E to LAV-AT-E70 and added LIFCL-33, LAV-AT-G70, and LAV-AT-X70 in Table 1.1. Quick Facts.
IP Generation and Evaluation	Added the Constraining the IP section.
Ordering Part Number	<p>Updated part numbers as follows:</p> <ul style="list-style-type: none"> Removed Single Design License. Added Multi-site Perpetual License and Single Machine Annual License.
Resource Utilization	Updated the resource utilization for the latest software version.

Revision 1.4, Lattice Radiant SW Version 2022.1, May 2023

Section	Change Summary
Functional Description	<ul style="list-style-type: none">Updated the description of Output Order in Table 2.3. Attributes Description.Updated the description in the Output Latency section.
Technical Support Assistance	Added reference link to the Lattice Answer Database.

Revision 1.3, Lattice Radiant SW Version 2022.1, November 2022

Section	Change Summary
Introduction	Updated Table 1.1. Quick Facts: <ul style="list-style-type: none">Added Lattice Avant to Supported FPGA Families.Added LAV-AT-500E to Targeted Devices.
Appendix A: Resource Utilization	<ul style="list-style-type: none">Updated <i>LFMXO5-25-9BBG400I</i> and <i>LFMXO5-25-7BBG400I</i> resource data using Radiant 2022.1.Added <i>LAV-AT-500E-3LFG1156I</i> and <i>LAV-AT-500E-1LFG1156I</i>.
Ordering Part Number	Added part numbers for Avant-E.
Appendix B: Limitations	Added IP Configuration limitation for Avant devices.

Revision 1.2, Lattice Radiant SW Version 3.2, May 2022

Section	Change Summary
Introduction	Updated Table 1.1. Quick Facts <ul style="list-style-type: none">Added MachXO5-NX to Supported FPGA FamiliesAdded 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: <ul style="list-style-type: none">FFT-COMP-XO5-U - FFT Compiler for MachXO5-NX - Single Design LicenseFFT-COMP-XO5-UT - FFT Compiler for MachXO5-NX - Site LicenseFFT-COMP-XO5-US - FFT Compiler for MachXO5-NX - 1 Year Subscription License
Appendix A: Resource Utilization	Updated resource utilization for <i>LFMXO5-25-9BBG400I</i> and <i>LFMXO5-25-7BBG400I</i> .

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.

Revision 1.0, Lattice Radiant SW Version 2.1, December 2020

Section	Change Summary
All	Initial release.



www.latticesemi.com