FFT Compiler IP Core User's Guide

August 2011
IPUG54_01.9

Table of Contents

Lattice's Fast Fourier transform (FFT) Compiler intellectual property (IP) core offers forward and inverse Fast Fourier Transforms 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. The FFT Compiler IP core offers two choices of implementation: high performance (Streaming I/O) and low resource (Burst I/O). In the 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. The low resource implementation can be used when it is required to use less slice (logic unit of Lattice FPGA devices) and Embedded Block RAM (EBR) and Digital Signal Processor (DSP) resources or if the device is 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. The FFT Compiler IP core also allows the number of FFT points to be varied dynamically through a port.

Quick Facts

Table 1-1 through Table 1-5 give quick facts about the FFT Compiler IP core for LatticeECP™, LattceECP2™, LatticeECP2M™, LatticeECP3™, and LatticeXP2™ devices, respectively.

Table 1-1. FFT Compiler IP Core for LatticeECP Devices Quick Facts

		FFT IP Configuration			
		High performance 256 points	**High performance 1024 points**	**Low performance 256 points**	**Low performance 256 points**
Core Requirements	FPGA Families Supported	LatticeECP			
	Minimal Device Needed	LFECP6E			
Resource Utilization	Targeted Device	LFECP33E-5F672C			
	Data Path Width	16	16	16	16
	LUTs	2100	2700	800	1000
	sysMEM EBRs	3	5	3	5
	Registers	1700	2100	800	900
	MULT18X18ADDSUB	6	8	2	2
Design Tool Support	Lattice Implementation	Lattice Diamond® 1.0 or ispLEVER® 8.1			
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1			
	Simulation	Aldec Active-HDL 8.2 Lattice Edition			
		Mentor Graphics ModelSim SE 6.3F			

Table 1-2. FFT Compiler IP Core for LatticeECP 2Devices Quick Facts

		FFT IP Configuration			
		High performance 256 points	High performance 1024 points	Low performance 256 points	Low performance 256 points
Core Requirements	FPGA Families Supported	Lattice ECP2			
	Minimal Device Needed	LFE2-6E	LFE2-12E	LFE2-6E	LFE2-6E
Resource Utilization	Targeted Device	LFE2-50E-7F672C			
	Data Path Width	16	16	16	16
	LUTs	2100	2700	800	900
	sysMEM EBRs	3	6	3	3
	Registers	1700	2100	800	800
	MULT18X18ADDSUB	6	8	2	2
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1			
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1			
	Simulation	Aldec Active-HDL 8.2 Lattice Edition			
		Mentor Graphics ModelSim SE 6.3F			

Table 1-3. FFT Compiler IP Core for LatticeECP2M Devices Quick Facts

		FFT IP Configuration			
		High performance 256 points	High performance 1024 points	Low performance 256 points	Low performance 256 points
Core Requirements	FPGA Families Supported	Lattice ECP2M			
	Minimal Device Needed	LFE2M20E			
Resource Utilization	Targeted Device	LFE2M35E-7F672C			
	Data Path Width	16	16	16	16
	LUTs	2100	2700	900	1000
	sysMEM EBRs	3	6	3	3
	Registers	1700	2100	800	900
	MULT18X18ADDSUB	6	8	2	2
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1			
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1			
	Simulation	Aldec Active-HDL 8.2 Lattice Edition			
		Mentor Graphics ModelSim SE 6.3F			

Table 1-4. FFT Compiler IP Core for LatticeECP3 Devices Quick Facts

		FFT IP Configuration			
		High performance 256 points	High performance 1024 points	Low performance 256 points	Low performance 256 points
Core Requirements	FPGA Families Supported	Lattice ECP3			
	Minimal Device Needed	LFE3-35EA			
Resource Utilization	Targeted Device	LFE3-95E-8FN672CES			
	Data Path Width	16	16	16	16
	LUTs	2100	2700	900	1000
	sysMEM EBRs	3	6	3	3
	Registers	1700	2100	800	900
	MULT18X18C	12	16	4	4
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1			
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1			
	Simulation	Aldec Active-HDL 8.2 Lattice Edition			
		Mentor Graphics ModelSim SE 6.3F			

Table 1-5. FFT Compiler IP Core for LatticeXP2 Devices Quick Facts

		FFT IP Configuration			
		High performance 256 points	High performance 1024 points	Low performance 256 points	Low performance 256 points
Core Requirements	FPGA Families Supported	Lattice XP2			
	Minimal Device Needed	LFXP2-5E	LFXP2-8E	LFXP2-5E	LFXP2-5E
Resource Utilization	Targeted Device	LFXP2-17E-7F484CES			
	Data Path Width	16	16	16	16
	LUTs	2100	2700	900	1000
	sysMEM EBRs	3	6	3	3
	Registers	1700	2100	800	900
	MULT18X18ADDSUB	6	8	2	2
Design Tool Support	Lattice Implementation	Diamond 1.0 or ispLEVER 8.1			
	Synthesis	Synopsys Synplify Pro for Lattice D-2009.12L-1			
	Simulation	Aldec Active-HDL 8.2 Lattice Edition			
		Mentor Graphics ModelSim SE 6.3F			

Features

- Wide range of points 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

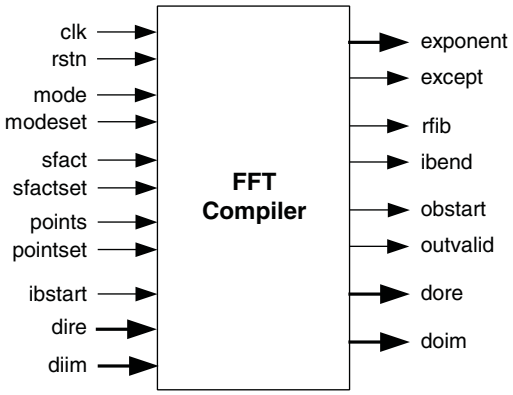
- Data precision of 8 to 24 bits
- Twiddle factor precision of 8 to 24 bits
- Natural order for input and choice of bit-reversed or natural order for output
- Support for arbitrary gaps between input data blocks in high-performance realization
- Block floating point scaling support in low resource configurations

This chapter provides a functional description of the Lattice FFT Compiler IP core.

Block Diagram

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.

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

where W_N is given by:

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

The inverse DFT is given by:

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

However the output of the FFT Compiler IP core will differ 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 will match with Equation (3) for this scaling mode. Using other scaling modes will result in outputs scaled by other appropriate scale factors.

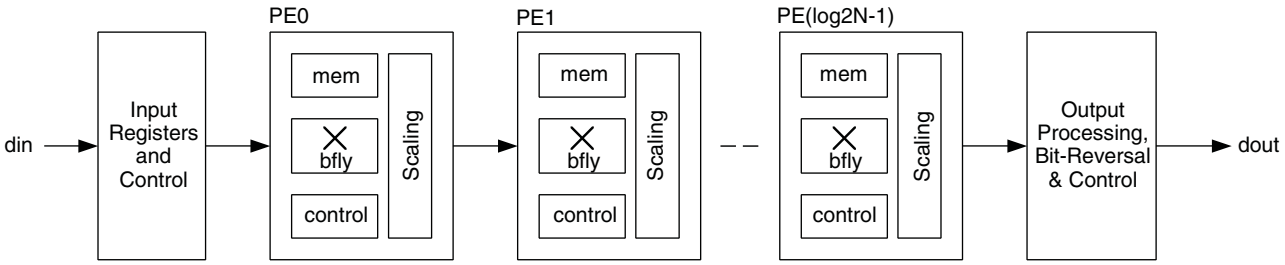
The FFT Compiler IP corer offers two different implementation modes - High Performance and Low Resource. In High Performance mode, the FFT Compiler IP core can continuously read in and give out one data sample per clock, block after block. The FFT throughput is equal to the clock rate when the data blocks 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.

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



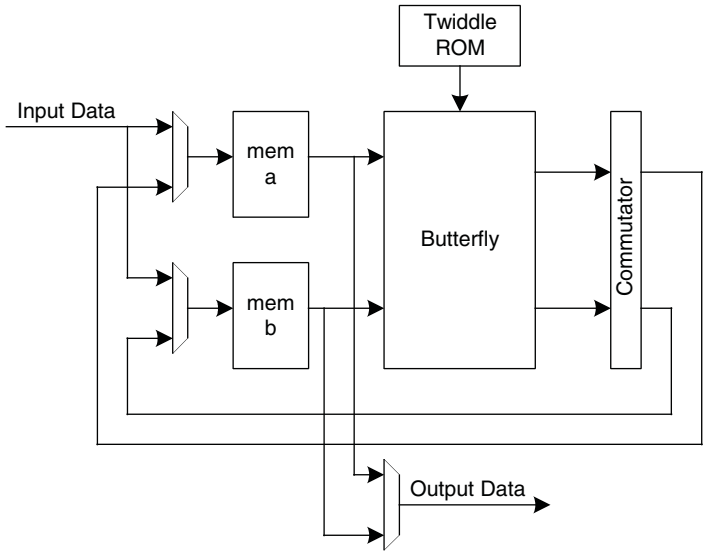
The high performance FFT implementation consists of several processing elements (PEs) connected in cascade. The number of PEs is equal to $\log_2 N$, where N is the number of FFT points. Each PE has a radix-2 decimation-in-frequency (DIF) butterfly (bfly), a memory (mem), an address generation and control 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.

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) will be unloaded first, followed by the data in memory b (N/2 to N-1), both in a bit-reversed order.

Configuring the FFT Compiler

Number of Points

The number of points can be either fixed and specified in the GUI or specified through the points input port. If the FFT core is going to be used for fixed points, it is better to configure it that way, to avoid the additional resource utilization in dynamic points configurations. In dynamic points configurations, the hardware resources used correspond to the maximum number of points. So to minimize resource utilization, the minimum and maximum points must be set to the minimum and maximum values expected in the usage.

Architecture

The operation mode must be chosen to suit the throughput requirements and resource availability. If block floating point scaling is required, then the Low Resource architecture must be selected.

Output Order

Output data can be chosen either to be in bit-reversed order or natural order. Natural order output is preferred in many applications, where the output is directly fed to the following stage. But in applications where a FIFO or a buffer is used between the output of FFT and the input to the next processing block, as in multi-clock systems, bit-reversal can be done in that glue memory. Selecting bit-reversed output results in reduced latency and/or lesser EBR usage. In low resource modes, the bit-reversed order is applicable separately to the lower and upper half of the output. For a 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 Mode

The selection of scaling mode depends on the statistical properties of the data and the trade-off between increased dynamic range and occasional overflows. If a large dynamic range is required for the internal computations and if the low-resource implementation is used, block floating point will be the best choice. If memory and logic resource utilization (and possibly throughput reduction) is not a problem, full precision arithmetic (scaling selected as “None”) can be used. The data width grows by one bit every stage requiring bigger multipliers, wider data path logic and wider memories. The output width is equal to input width + $\log_2 N$.

Among the fixed scale factors, RS111 is a more uniform scaling method than RS211. If the input data magnitude is known to be higher on the average, the RS211 method may be employed. Dynamic scale mode is for more complex scenarios where the statistical properties of the input differ between data blocks and the user wants to use more than one scaling scheme.

Precision Reduction Method

Precision reduction method selection applies to the scaling process after every stage. When scaling is employed (a divide by 2 or 4), the scaled data is obtained either by truncating the data (discarding last one or two bits) or rounding the data to the nearest number in the scaled precision (discarding one or two bits and making correction to the output data based on discarded bits). Truncation results in less logic utilization while rounding improves the accuracy of results.

Signal Descriptions

The top-level interface diagram for the FFT Compiler 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	Bits	I/O	Description
Clock and Reset			
clk	1	I	System clock.
rstn	1	I	System wide asynchronous active-low reset signal.
Data Input and Output			
dire	8 - 24	I	Data in real. Real part of the input data.
diim	8 - 24	I	Data in imaginary. Imaginary part of the input data.
dore	8 - 24	O	Data out real. Real part of the output data.
doim	8 - 24	O	Data out imaginary. Imaginary part of the output data.
exponent	$Log_2(N)+1$	O	Exponent output for block floating point. This value denotes the effective scaling that was done during block floating scaling.
Configuration Signals			
mode	1	I	FFT mode signal. Programs core to perform forward or inverse FFT. 0 - Forward FFT 1 - Inverse FFT The value at mode is loaded into the system whenever modeset input goes high. The changes are effective from the start of the next input data block, i.e., for an ibstart going high during or after modeset.
modeset	1	I	Set FFT mode signal. When this input signal goes high, the value at mode port is read and the FFT mode (forward or inverse FFT) is set.

Table 2-1. Top level I/O interface (Continued)

Port	Bits	I/O	Description
sfact	12 - 28	I	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 input goes high. The changes are effective from the start of the next input data block, i.e., for an ibstart going high during or after sfactset.
sfactset	1	I	Set scale factor signal. When this signal goes high, stage-wise scaling factors are set with the values at sfact input port.
points	3 if maximum points = 128 4, otherwise	I	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 will be read as 6 and a value greater than 14 will be read as 14.
pointset	1	I	Set number of points signal. When this input signal goes high, the value at points 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, i.e., for a valid ibstart applied after pointset going high. For low resource mode, pointset can be applied during or before ibstart. For high performance mode, pointset must be applied five cycles before ibstart.
Status and Handshake Signals			
ibstart	1	I	Input block start. 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 signal during an input read cycle is ignored.
except	1	O	Exception. 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	1	O	Ready for input block. This signal indicates that the core is ready to receive the next block of input data. The driving system can assert ibstart one cycle after rfib goes high. After ibstart goes high, the core will pull rfib low in the next clock cycle.
ibend	1	O	Input block end. 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	1	O	Output block start. This signal is asserted high by the core for one clock cycle, to signify the start of an output block of data.
outvalid	1	O	Output data valid. This signal indicates that the core is now giving out valid output data through dore and doim ports.

Interfacing with the FFT Compiler

Configuration Signals

There are three dynamic configuration signals used in the FFT compiler: mode, sfact and points. The user independently selects each of these. The configuration signals are sampled and stored when the corresponding set signals are active. Specifically mode is set when modeset is active, sfact is set when sfactset is active, and points is set when pointset 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 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 is high. Refer to

Figure 2-6 for an illustration of configuration signal timing. In high performance implementation, the pointset must be applied five cycles before ibstart for it to be effective for the next block.

Handshake Signals

The input ibstart 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 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 is asserted high by the core. The output control signal rfib indicates that the core is ready for a new input block. One cycle after ibstart, the output rfib goes low and it remains low until one cycle before the next block can be applied. The external driving system can check for rfib and start an input block in the next cycle after rfib goes high. Refer to Figure 2-6 for an illustration of configuration signal timing.

Exponent Output

The output port exponent 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:

True (i)FFT output = (dore * 2^exponent) + j (doim * 2^exponent)

Exceptions

Exceptions occur if there is an internal overflow in the computation of an output block. An exception is notified by the except signal going high during a valid output. The except 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 output may be left unconnected leading to a slightly reduced resource utilization.

Timing Specifications

The top-level timing diagrams for several cases are given in Figure 2-4 through Figure 2-7.

Figure 2-4. Timing Diagram for Streaming I/O with Continuous Data Blocks (64 Points)

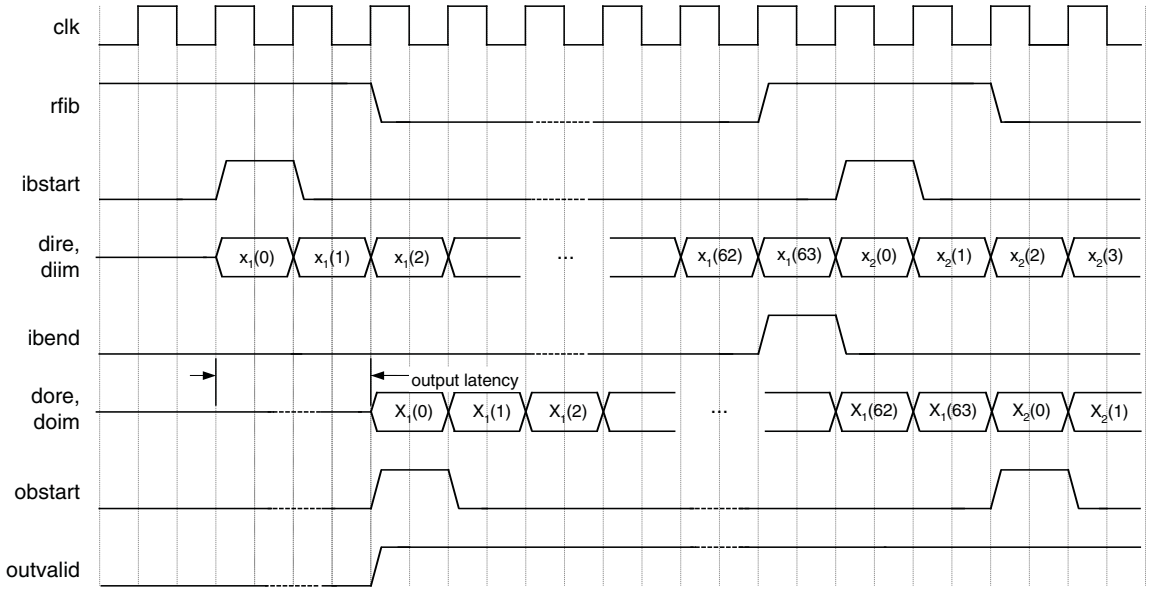


Figure 2-5. Timing Diagram for Streaming I/O with Gaps Between Data Blocks (64 Points)

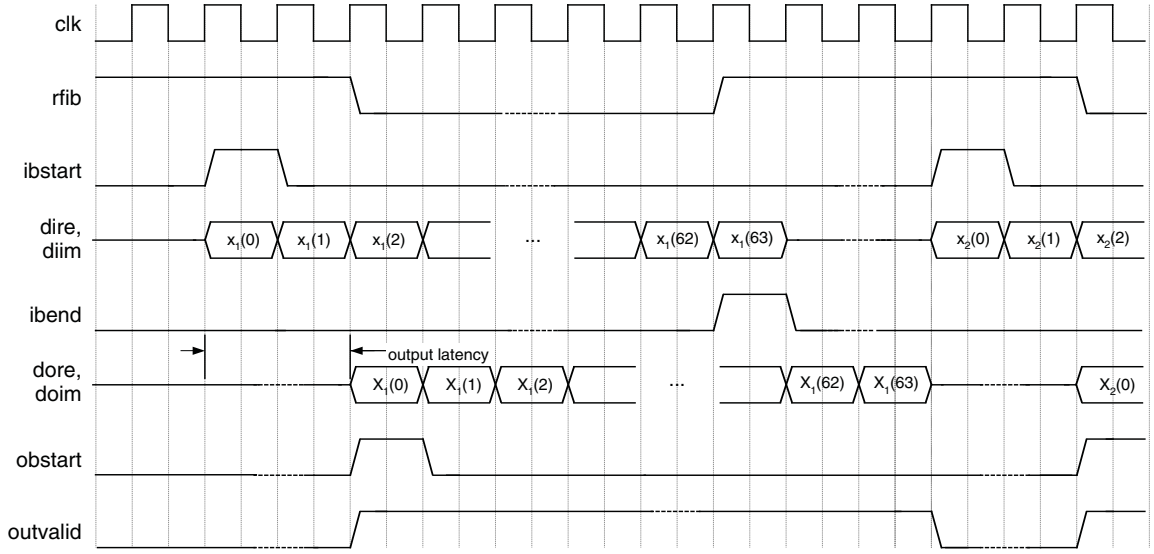


Figure 2-6. Timing Diagram Showing Handshake Signals for Low Resource FFT

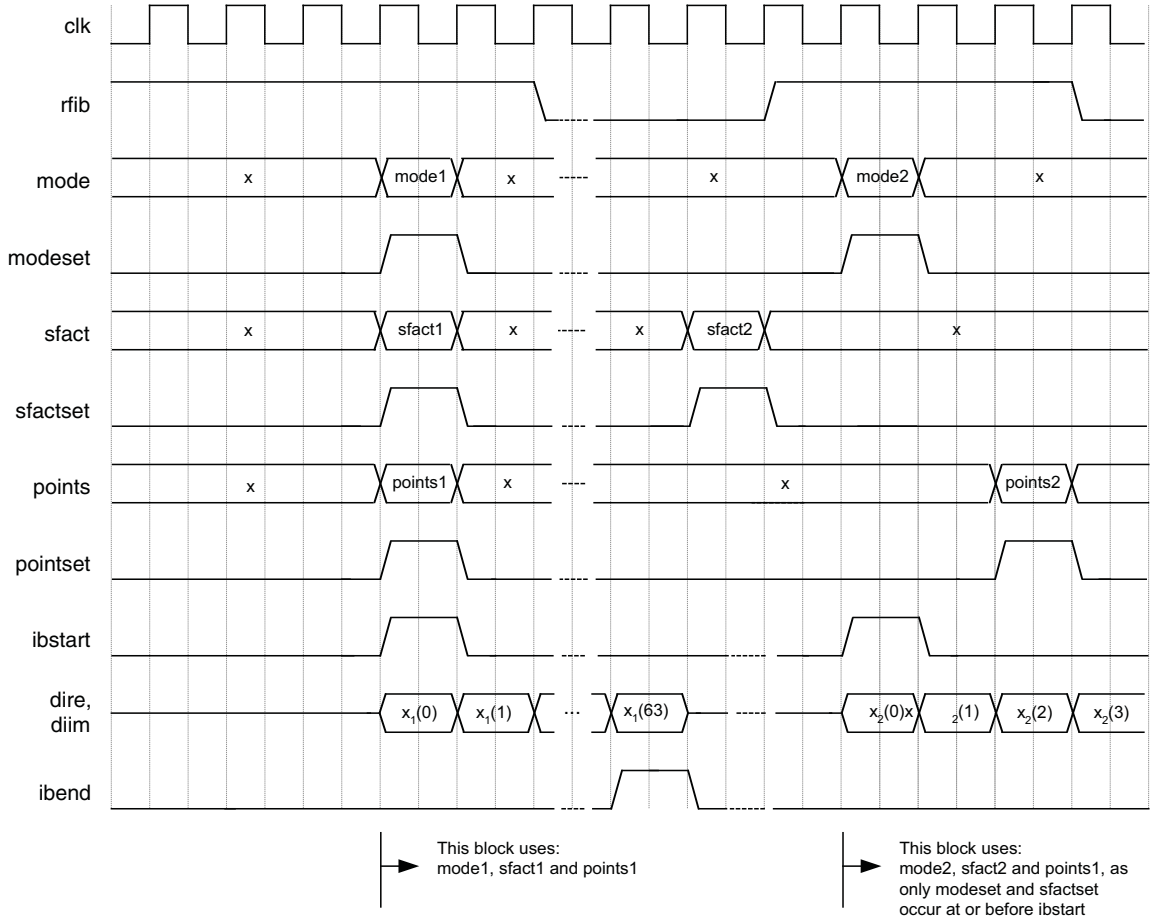
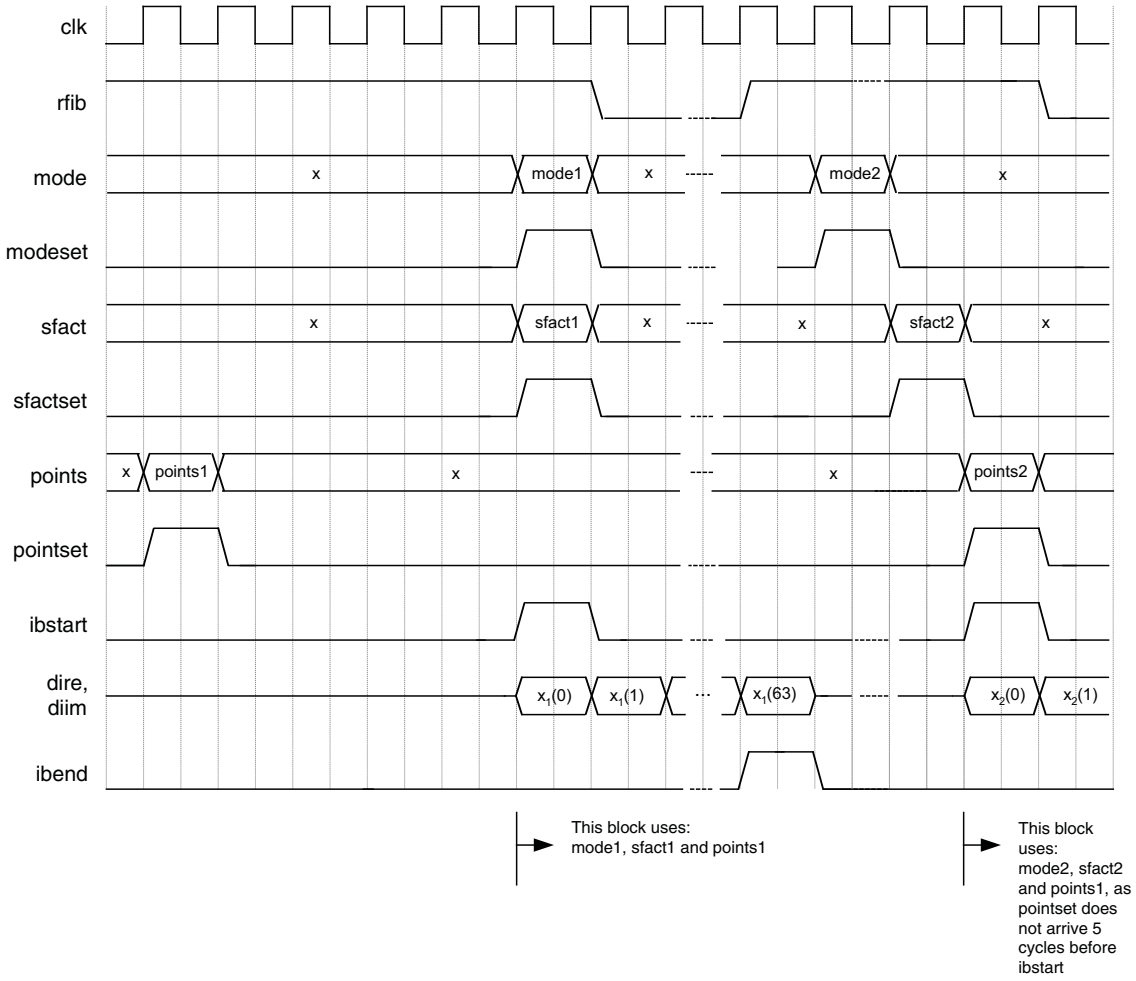


Figure 2-7. Timing Diagram Showing Handshake Signals for High Performance FFT



Output Latency

Table 2-2 gives the latency through the IP core as a function of FFT point size and implementation mode.

Table 2-2. Latency through FFT IP Core (Clock Cycles)

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

Parameter Settings

The IPexpress™ tool is used to create IP and architectural modules in the Diamond and ispLEVER software. Refer to "IP Core Generation" on page 21 for a description on how to generate the IP.

Table 3-1 provides the list of user configurable parameters for the FFT Compiler IP core. The parameter settings are specified using the FFT Compiler IP core Configuration GUI in IPexpress.

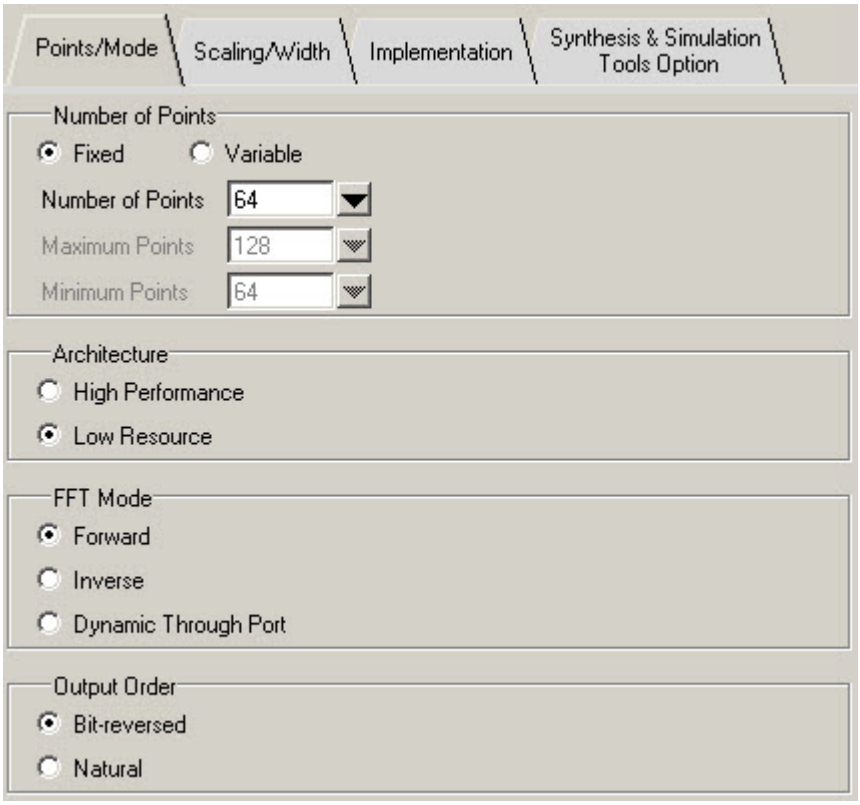
Table 3-1. FFT Compiler Parameter Description

Name	Range	Default
Number of Points		
Points variability	Fixed, Variable	Fixed
Number of Points (N)	64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384	64
Minimum points	64, 128, 256, 512, 1024, 2048, 4096, 8192	64
Maximum points	{128, 256, 512, 1024, 2048, 4096, 8192, 16384} only values > `Min points` are valid	128
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 Mode		
Scaling Mode	None, RS111, RS211, Dynamic Through Port, Block Floating Point	RS111
Data Widths		
Input Data Width	8 to 24	16
Twiddle Factor Width	8 to 24	16
Precision Reduction Method		
Precision Reduction	Truncation, Rounding	Truncation
Implementation		
Multiplier Type	DSP block based, LUT based	DSP block based
Adder Pipeline	0,1	0
Multiplier Pipeline	2,3,4	3
Memory Type	EBR Memory, Distributed Memory, Automatic	EBR Memory
Synthesis & Simulation Tools		
Synthesis Tools	Support Synplify Support Precision RTL Sythesis	Both are selected
Simulation Tools	Support ModelSim Support Active-HDL	Both are selected

Points/Mode Tab

Figure 3-1 shows the contents of the Points/Mode tab.

Figure 3-1. Points/Mode Tab



Number of Points

This parameter allows the user to specify fixed or variable number of points.

Number of Points
This parameter specifies the number of FFT points if points variability is “FIxed.”

Maximum Points
This parameter denotes the maximum for the points range if points variability is “Variable.”

Minimum Points
This parameter denotes the minimum for the points range if points variability is “Variable.”

Architecture

This option selects either High-Performance (Streaming I/O) or Low Resource (Burst I/O) architecture. The high performance implementation offers high throughput and allows streaming input and output data, with optional gaps between blocks. The low resource implementation results in low memory and logic resource utilization, but takes multiple block periods (typically between 4 to 8) of computation time to process each data block.

FFT Mode

This parameter configures operating mode of the core to forward FFT, inverse FFT or dynamically variable forward/inverse FFT. In the “Dynamic Through Port” mode, the FFT mode can be set to forward or inverse FFT using the input ports mode and modeset.

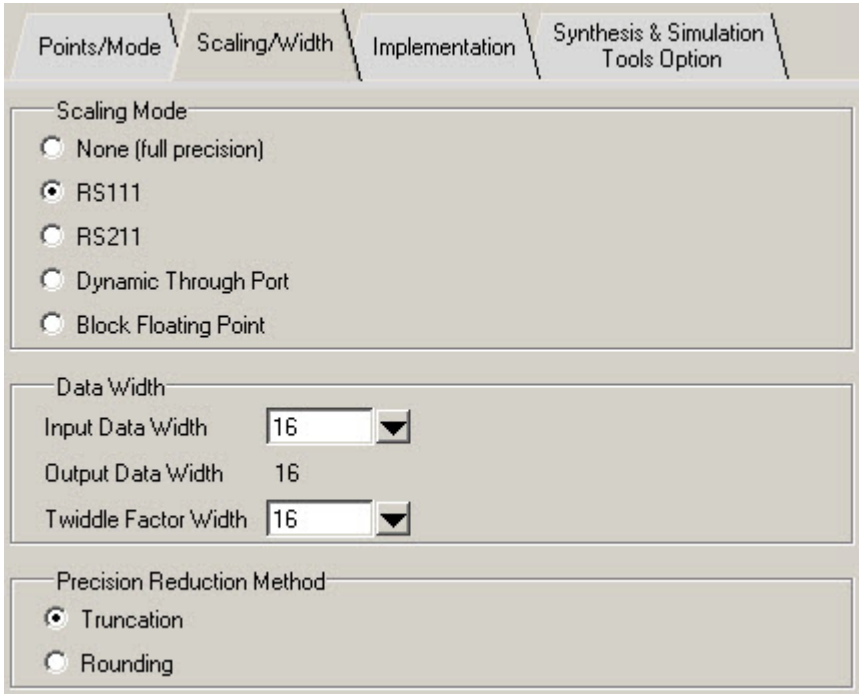
Output Order

This parameter specifies whether the output data is in bit-reversed or natural order.

Scaling Width Tab

Figure 3-1 shows the contents of the Scaling Width tab.

Figure 3-2. Scaling Width Tab



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. The value can be None, RS111, RS211, Dynamic Through Port and Block Floating Point.

- If the value is “None” there is no scaling at the output of butterflies.
- The option RS111 results in a fixed scaling of “right shift by 1” in all FFT stages.
- The option RS211 results in a fixed scaling of “right shift by 2” in the first stage and “right shift by 1” in the subsequent stages.
- If Scale Mode is set to “Dynamic Through Port”, the scale factors for the FFT stages are read dynamically from the input port sfact for every data block. In the dynamic scaling mode, for high performance architecture, there is an option to set the last stage (or the last two stages, if FFT Mode is dynamic) scaling to fixed scaling to improve the performance.
- In 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 is added to the FFT compiler core when this option is selected. This option is not available for the Streaming I/O mode.

Data Width

Input Data Width

This parameter specifies input data width (width of either of the components: real or imaginary).

Twiddle Factor Width
This parameter specifies twiddle factor width (width of either of the components: real or imaginary).

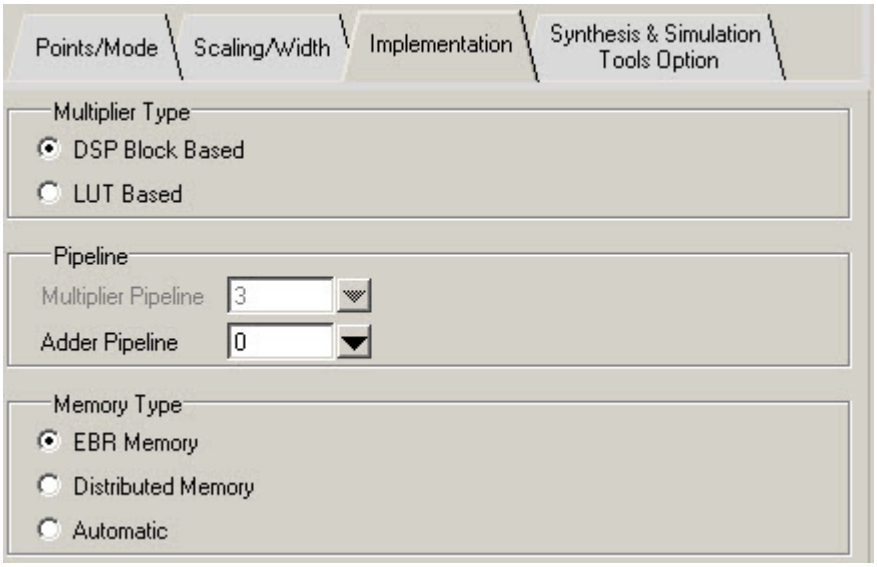
Precision Reduction Method

This parameter specifies whether the data is truncated or rounded nearest during scaling. In rounding mode, for high performance architecture, there is an additional option to set the last stage (or the last two, if FFT Mode is dynamic) to truncation. Setting the last stage to truncation results in better throughput.

Implementation Tab

Figure 3-3 shows the contents of the Implementation tab.

Figure 3-3. Implementation Tab



Multiplier Type

This option specifies whether DSP blocks or LUTs are used for implementing multipliers and multiply-add components.

Pipeline

Multiplier Pipeline
This option is used to specify the pipeline of LUT based multipliers in low resource mode. Higher values for pipeline leads to better performance at the cost of slightly increased utilization and latency.

Adder Pipeline
This option is used to specify an additional pipeline after the adders. Additional pipeline leads to better performance at the cost of slightly increased utilization and latency. This option is available only when architecture is low resource and the scale mode is not block floating point.

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.

Synthesis and Simulation Tools Tab

Figure 3-3 shows the contents of the Synthesis and Simulation Tools tab.

Figure 3-4. Synthesis and Simulation Tools Tab

Support Synplify

If selected, IPexpress generates evaluation scripts and other associated files required to synthesize the top-level design using the Synplify synthesis tool.

Support Precision

If selected, IPexpress generates evaluation script and other associated files required to synthesize the top-level design using the Precision synthesis tool.

Support ModelSim

If selected, IPexpress generates evaluation script and other associated files required to run simulation using the ModelSim simulator.

Support ALDEC

If selected, IPexpress generates evaluation script and other associated files required to run simulation using the ALDEC simulator.

IP Core Generation

This chapter provides information on licensing the FFT Compiler IP core, generating the core using the Diamond or ispLEVER software IPexpress tool, running functional simulation, and including the core in a top-level design. The Lattice FFT Compiler IP core can be used in LatticeECP3, LatticeECP2/M, LatticeECP, and LatticeXP2 device families.

Licensing the IP Core

An IP license is required to enable full, unrestricted use of the FFT Compiler IP core in a complete, top-level design. An IP license that specifies the IP core and device family is required to enable full use of the core in Lattice devices. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The FFT Compiler IP core also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (several hours) without requiring an IP license (see "Instantiating the Core" on page 25 for further details). However, a license is required to enable timing simulation, to open the design in the Diamond or ispLEVER EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

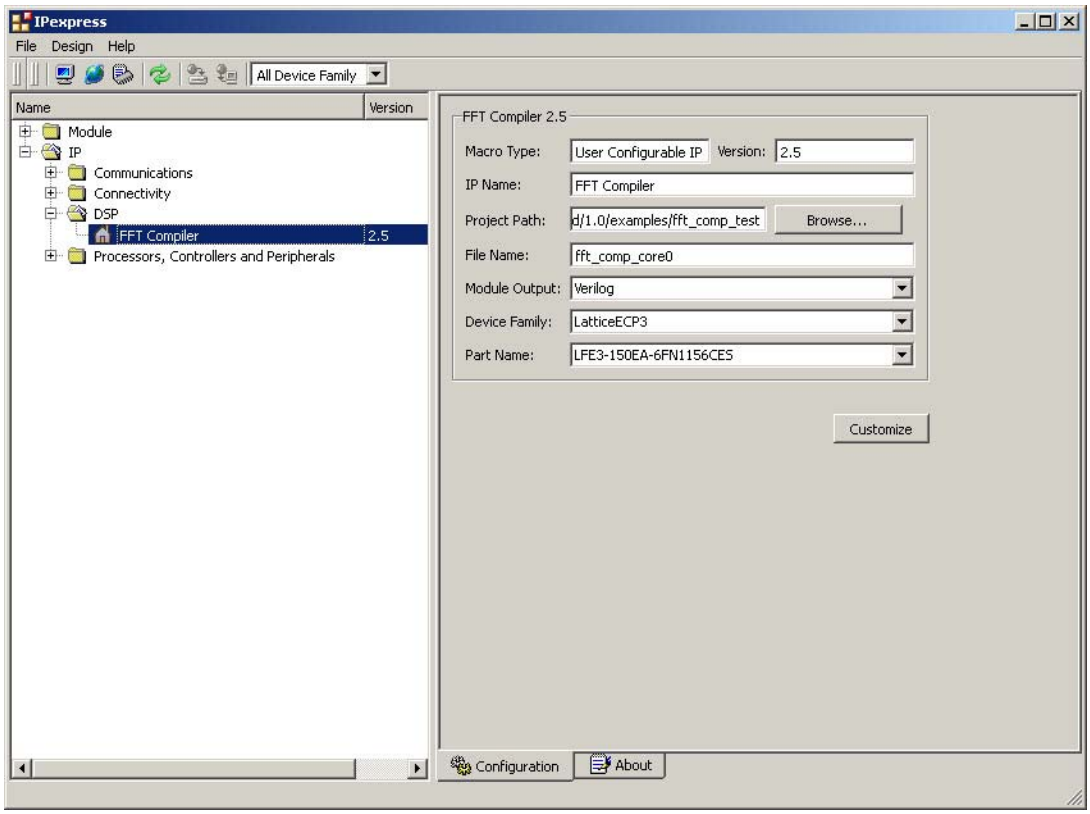
Getting Started

The FFT Compiler IP core is available for download from the Lattice IP Server using the IPexpress tool. The IP files are automatically installed using InstallShield technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the FFT Compiler IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be loaded.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **(ispLEVER) Design Entry Type** – Verilog HDL or VHDL.
- **Device Family** – Device family to which IP is to be targeted (e.g. LatticeSCM, Lattice ECP2M, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family.

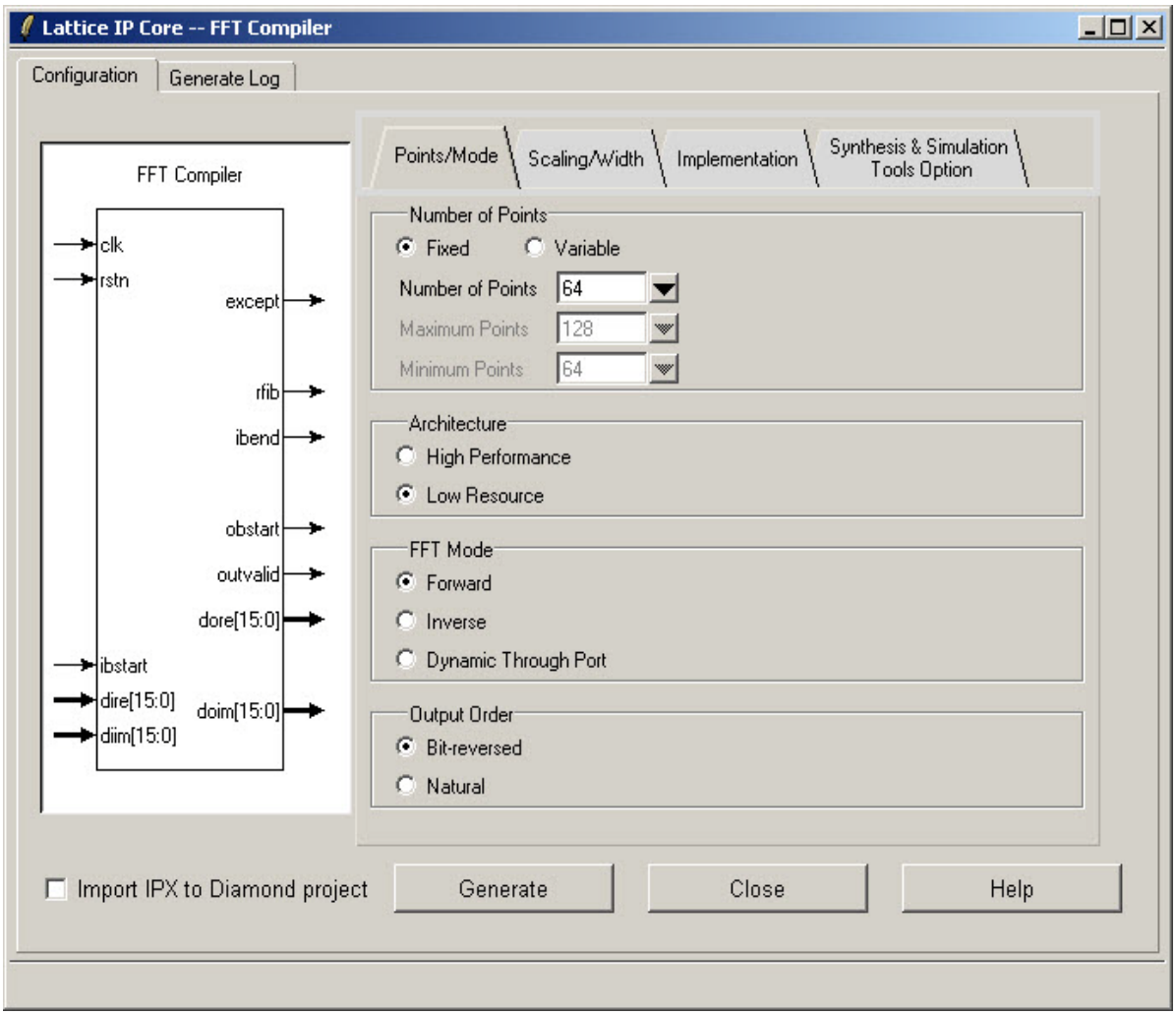
Figure 4-1. IPexpress Dialog Box (Diamond Version)



Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output (Design Entry in ispLEVER), Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration, the user clicks the **Customize** button in the IPexpress tool dialog box to display the FFT Compiler IP core Configuration GUI, as shown in Figure 4-2. From this dialog box, the user can select the IP parameter options specific to their application. Refer to “Parameter Settings” on page 16 for more information on the parameter settings.

Figure 4-2. Configuration Dialog Box (Diamond Version)



IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified "Project Path" directory. The directory structure of the generated files is shown in Figure 4-3.

Figure 4-3. Lattice FFT Compiler IP Core Directory Structure

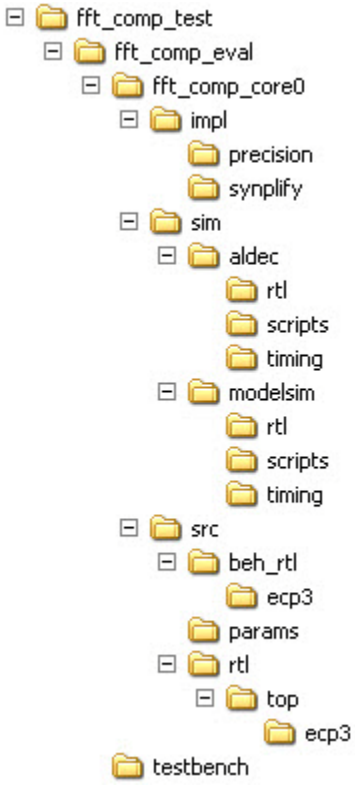


Table 4-1 provides a list of key files created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

Table 4-1. File List

File	Description
<*username*>_inst.v	This file provides an instance template for the IP.
<*username*>.v	This file provides the FFT core for simulation.
<*username*>_beh.v	This file provides a behavioral simulation model for the FFT core.
<*username*>_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>.ngo	The ngo files provide the synthesized IP core.
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*username*e>_top.[v,vhd]	This file provides a module which instantiates the FFT core. This file can be easily modified for the user's instance of the FFT core. This file is located in the `fft_comp_eval/<username>/src/rtl/top` directory.

Table 4-1. File List (Continued)

File	Description
twidx<*username*>.mem	Twiddle factor to initialize ROM. The "x" in file name can be 0,1,2, etc.
<*username*>_generate.tcl	This file is created when GUI "Generate" button is pushed and generation is invoked. This file may be run from command line.
<*username*>_generate.log	This is the IPexpress scripts log file.
<*username*>_gen.log	This is the IPexpress IP generation log file.

Instantiating the Core

The \<fft_comp_eval> and subtending directories provide files supporting FFT IP core evaluation. The \<fft_comp_eval> directory shown in Figure 4-3 contains files and folders with content that is constant for all configurations of the FFT. The \<username> subfolder (\fft_comp_core0 in this example) contains files and folders with content specific to the username configuration. The \fft_comp_eval directory is created by IPexpress the first time the core is generated and updated each time the core is regenerated. A \<*username*> directory is created by IPexpress each time the core is generated and regenerated each time the core with the same file name is regenerated. A separate \<*username*> directory is generated for cores with different names, e.g. \fft_compiler_eval0, \fft_compiler_eval1, etc.

Running Functional Simulation

Simulation support for the FFT IP core is provided for Aldec Active-HDL (Verilog and VHDL) simulator and for Mentor Graphics ModelSim simulator. The functional simulation includes a configuration-specific behavioral model of the FFT IP core. The test bench sources stimulus to the core, and monitors output from the core. The generated IP core package includes the configuration-specific behavior model (<*username*>_beh.v) for functional simulation in the "Project Path" root directory. The simulation scripts supporting ModelSim evaluation simulation is provided in \<*project_dir*>\fft_comp_eval\<*username*>\sim\modelsim\scripts. The simulation script supporting Aldec evaluation simulation is provided in \<*project_dir*>\fft_comp_eval\<*username*>\sim\aldec\scripts. Both ModelSim and Aldec simulation are supported via test bench files provided in \<*project_dir*>\fft_comp_eval\testbench. Models required for simulation are provided in the corresponding \models folder. Users may run the Aldec evaluation simulation by doing the following:

1. Open Active-HDL.
2. Under the Tools tab, select **Execute Macro**.
3. Browse to folder \<*project_dir*>\fft_comp_eval\<*username*>\sim\aldec\scripts and execute one of the "do" scripts shown.

Users may run the ModelSim evaluation simulation by doing the following:

1. Open ModelSim.
2. Under the File tab, select **Change Directory** and choose the folder <*project_dir*>\fft_comp_eval\<*username*>\sim\modelsim\scripts.
3. Under the Tools tab, select **Execute Macro** and execute the ModelSim "do" script shown.

Note: *When the simulation completes, a pop-up window will appear asking "Are you sure you want to finish?" Answer* ***No*** *to analyze the results. Answering* ***Yes*** *closes ModelSim.*

Synthesizing and Implementing the Core in a Top-Level Design

Synthesis support for the FFT IP core is provided for Mentor Graphics Precision or Synopsys Synplify. The FFT IP core itself is synthesized and is provided in NGO format when the core is generated in IPexpress. Users may syn-

thesize the core in their own top-level design by instantiating the core in their top-level as described previously and then synthesizing the entire design with either Synplify or Precision RTL Synthesis. The following text describes the evaluation implementation flow for Windows platforms. The flow for Linux and UNIX platforms is described in the Readme file included with the IP core. The top-level files <username>_top.v are provided in \<*project_dir*>\fft_comp_eval\<*username*>\src\rtl\top. Push-button implementation of the reference design is supported via Diamond or ispLEVER project files, <user-name>.syn, located in the following directory: \<*project_dir*>\fft_comp_eval\<*username*>\impl\(*synplify or precision*).

To use this project file in Diamond:

1. Choose **File > Open > Project**.
2. Browse to \<*project_dir*>\fft_comp_eval\<*username*>\impl\synplify (or precision) in the Open Project dialog box.
3. Select and open <*username*>.ldf. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.

4. Select the **Process** tab in the left-hand GUI window.

5. Implement the complete design via the standard Diamond GUI flow.

To use this project file in ispLEVER:

1. Choose **File > Open Project.**
2. Browse to \<*project_dir*>\fft_comp_eval\<*username*>\impl\synplify (or precision) in the Open Project dialog box.
3. Select and open <username>.syn. At this point, all of the files needed to support top-level synthesis and implementation will be imported to the project.
4. Select the device top-level entry in the left-hand GUI window.
5. Implement the complete design via the standard ispLEVER GUI flow.

Hardware Evaluation

The FFT Compiler IP core supports supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (several hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

Enabling Hardware Evaluation in Diamond

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

Enabling Hardware Evaluation in ispLEVER

In the Processes for Current Source pane, right-click the **Build Database** process and choose **Properties** from the dropdown menu. The hardware evaluation capability may be enabled/disabled in the Properties dialog box. It is enabled by default.

Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

Regenerating an IP Core in Diamond

To regenerate an IP core in Diamond:

1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP you wish to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the T**arget** box.
4. If you want to generate a new set of files in a new location, set the new location in the **IPX Target File** box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.
5. Click **Regenerate.** The module's dialog box opens showing the current option settings.
6. In the dialog box, choose the desired options. To get information about the options, click **Help**. Also, check the About tab in IPexpress for links to technical notes and user guides. IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.
7. To import the module into your project, if it's not already there, select **Import IPX to Diamond Project** (not available in stand-alone mode).
8. Click **Generate**.
9. Check the Generate Log tab to check for warnings and error messages.
10. Click **Close**.

The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above.

Regenerating an IP Core in ispLEVER

To regenerate an IP core in ispLEVER:

1. In the IPexpress tool, choose **Tools > Regenerate IP/Module**.
2. In the Select a Parameter File dialog box, choose the Lattice Parameter Configuration (.lpc) file of the IP core you wish to regenerate, and click **Open**.
3. The Select Target Core Version, Design Entry, and Device dialog box shows the current settings for the IP core in the Source Value box. Make your new settings in the Target Value box.
4. If you want to generate a new set of files in a new location, set the location in the LPC Target File box. The base of the .lpc file name will be the base of all the new file names. The LPC Target File must end with an .lpc extension.
5. Click **Next**. The IP core's dialog box opens showing the current option settings.

6. In the dialog box, choose desired options. To get information about the options, click **Help**. Also, check the About tab in the IPexpress tool for links to technical notes and user guides. The IP core might come with additional information. As the options change, the schematic diagram of the IP core changes to show the I/O and the device resources the IP core will need.
7. Click **Generate**.
8. Click the **Generate Log** tab to check for warnings and error messages.

Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

Lattice Technical Support

There are a number of ways to receive technical support.

Online Forums

The first place to look is Lattice Forums (http://www.latticesemi.com/support/forums.cfm). Lattice Forums contain a wealth of knowledge and are actively monitored by Lattice Applications Engineers.

Telephone Support Hotline

Receive direct technical support for all Lattice products by calling Lattice Applications from 5:30 a.m. to 6 p.m. Pacific Time.

- For USA & Canada: 1-800-LATTICE (528-8423)
- For other locations: +1 503 268 8001

In Asia, call Lattice Applications from 8:30 a.m. to 5:30 p.m. Beijing Time (CST), +0800 UTC. Chinese and English language only.

- For Asia: +86 21 52989090

E-mail Support

- techsupport@latticesemi.com
- techsupport-asia@latticesemi.com

Local Support

Contact your nearest Lattice Sales Office.

Internet

www.latticesemi.com

References

[1] Lawrence R. Rabiner and Bernard Gold, “Theory and Application of Digital Signal Processing”, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

LatticeECP

- HB1000, *LatticeECP/EC Family Handbook*

LatticeECP2/M

- HB1003, *LatticeECP2/M Family Handbook*

LatticeECP3

- HB1009, *LatticeECP3 Family Handbook*

LatticeXP2

- DS1009, *Lattice XP2 Datasheet*

Revision History

Date	Document Version	IP Version	Change Summary
—	—	—	Previous Lattice releases.
September 2006	01.2	2.0	Added LatticeECP2 device support.
February 2007	01.3	2.1	Added LatticeECP2M device support.
			Updated for ispLEVER 6.1 SP1.
May 2007	01.4	2.2	Added support for LatticeXP2 FPGA family.
February 2008	01.5	2.3	Updated appendices.
May 2009	01.6	2.4	Added support for LatticeECP3, updated figures and tables.
June 2010	01.7	2.4	Divided document into chapters. Added table of contents.
			Added Quick Facts tables in Chapter 1, “Introduction.”
			Added new content in Chapter 4, “IP Core Generation.”
August 2010	01.8	2.5	Added support for Diamond software throughout.
August 2011	01.9	2.5	Added Output Latency section.

Appendix A:

Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the FFT Compiler IP core. The IP configurations shown in this chapter were generated using the IPexpress software tool.

IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the Diamond and ispLEVER design tools. Details regarding the usage of IPexpress can be found in the IPexpress and Diamond or ispLEVER help system. For more information on the Diamond or ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

LatticeECP Devices

Table A-1. Performance and Resource Utilization[1]

# Points	Operating Mode	SLICEs	LUTs	Registers	sysMEM™ EBRs	sysDSP™ Blocks	f_{MAX} (MHz)
64	Low resource	488	588	705	3	1	213
64	High performance	998	1444	1174	1	2	208
256	Low resource	575	732	772	3	1	212
256	High performance	1424	2060	1636	3	3	200
1024	Low resource	702	957	861	5	1	192
1024	High performance	1781	2589	2056	7	4	182
8192	Low resource	769	1055	913	36	1	189
8192	High performance	2579	3950	2762	38	6	179

1. Performance and utilization data are generated targeting a LFECP33E-5F672C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP family.

Ordering Part Number

The Ordering Part Number (OPN) for the FFT Compiler for LatticeECP/EC devices is FFT-COMP-E2-U2.

LatticeECP2 Devices

Table A-2. Performance and Resource Utilization[1]

# Points	Operating Mode	SLICEs	LUTs	Registers	sysMEM EBRs	sysDSP Blocks	f_{MAX} (MHz)
64	Low resource	499	608	695	3	1	283
64	High performance	723	1435	1174	1	2	283
256	Low resource	578	745	755	3	1	250
256	High performance	1049	2078	1636	3	3	250
1024	Low resource	666	894	804	3	1	258
1024	High performance	1324	2626	2056	6	4	258
8192	Low resource	787	1123	897	18	1	240
8192	High performance	1952	3880	2762	21	6	240

1. Performance and utilization data are generated targeting a LFE2-50-7F672C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP2 family.

Ordering Part Number

The Ordering Part Number (OPN) for the FFT Compiler for LatticeECP2 devices is FFT-COMP-P2-U2.

LatticeECP2M Devices

Table A-3. Performance and Resource Utilization[1]

# Points	Operating Mode	SLICEs	LUTs	Registers	sysMEM EBRs	sysDSP Blocks	f_{MAX} (MHz)
64	Low resource	534	681	730	3	1	286
64	High performance	989	1437	1174	1	2	243
256	Low resource	616	830	789	3	1	292
256	High performance	1419	2070	1636	3	3	266
1024	Low resource	718	991	839	3	1	273
1024	High performance	1782	2618	2056	6	4	261
8192	Low resource	787	1123	897	18	1	234
8192	High performance	2518	3867	2762	21	6	229

1. Performance and utilization data are generated targeting a LFE2M-35E-7F672C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP2M family.

Ordering Part Number

The Ordering Part Number (OPN) for the FFT Compiler for LatticeECP2M devices is FFT-COMP-PM-U2.

LatticeECP3 Devices

Table A-4. Performance and Resource Utilization[1]

# Points	Operating Mode	SLICEs	LUTs	Registers	sysMEM EBRs	sysDSP Blocks	f_{MAX} (MHz)
64	Low resource	493	668	727	3	2	285
64	High performance	733	1433	1174	1	2	243
256	Low resource	575	821	783	3	2	255
256	High performance	1065	2074	1636	3	3	266
1024	Low resource	653	964	820	3	2	254
1024	High performance	1329	2602	2056	6	4	261
8192	Low resource	722	1089	870	18	2	258
8192	High performance	1957	3843	2762	21	6	229

1. Performance and utilization data are generated targeting a LFE3-95E-8FN672CES device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family.

Ordering Part Number

The Ordering Part Number (OPN) for the FFT Compiler for LatticeECP3 devices is FFT-COMP-E3-U2.

LatticeXP2 Devices

Table A-5. Performance and Resource Utilization[1]

# Points	Operating Mode	SLICEs	LUTs	Registers	sysMEM EBRs	sysDSP Blocks	f_{MAX} (MHz)
64	Low resource	534	681	730	3	1	261
64	High performance	989	1437	1174	1	2	243
256	Low resource	616	830	789	3	1	226
256	High performance	1419	2070	1636	3	3	266
1024	Low resource	718	991	839	3	1	237
1024	High performance	1782	2618	2056	6	4	261

1. Performance and utilization data are generated targeting a LFXP2-17E-7F484C device using Lattice Diamond 1.0 and Synplify Pro D-2009.12L-1 software. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeXP2 family.

Ordering Part Number

The Ordering Part Number (OPN) for the FFT Compiler for LatticeXP2 devices is FFT-COMP-X2-U2.

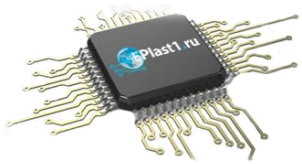
Компания «ЭлектроПласт» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Оперативные поставки широкого спектра электронных компонентов отечественного и импортного производства напрямую от производителей и с крупнейших мировых складов;
- Поставка более 17-ти миллионов наименований электронных компонентов;
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- Лицензия ФСБ на осуществление работ с использованием сведений, составляющих государственную тайну;
- Поставка специализированных компонентов (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Aeroflex, Peregrine, Syfer, Eurofarad, Texas Instrument, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits,General Dynamics и др.);

Помимо этого, одним из направлений компании «ЭлектроПласт» является направление «Источники питания». Мы предлагаем Вам помощь Конструкторского отдела:

- Подбор оптимального решения, техническое обоснование при выборе компонента;
- Подбор аналогов;
- Консультации по применению компонента;
- Поставка образцов и прототипов;
- Техническая поддержка проекта;
- Защита от снятия компонента с производства.



Как с нами связаться

Телефон: 8 (812) 309 58 32 (многоканальный)
Факс: 8 (812) 320-02-42
Электронная почта: org@eplast1.ru
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, дом 2, корпус 4, литера А.